专业人士笔记

HTML5 画布

专业人士笔记

100+ 页

专业提示和技巧

GoalKicker.com
免费编程书籍

HTML5 Canvas

Notes for Professionals

100+ pages

of professional hints and tricks

GoalKicker.com
Free Programming Books

目录

关于 1
第1章：HTML5画布入门 2
第1.1节：检测画布上的鼠标位置 2
第1.2节：画布大小和分辨率 2
第1.3节：旋转 3
第1.4节：将画布保存为图像文件 3
第1.5节：如何将Html5画布元素添加到网页 4
第1.6节：Html5画布功能与用途索引 5
第1.7节：O屏幕画布 6
第1.8节：你好，世界 6
第2章：文本 8
第2.1节：两端对齐文本 8
第2.2节：两端对齐段落 13
第2.3节：沿弧线渲染文本 17
第2.4节：曲线文本，三次和二次贝塞尔曲线 22
第2.5节：绘制文本 25
第2.6节：文本格式化 26
第2.7节：将文本换行成段落 27
第2.8节：将文本段落绘制成不规则形状 28
第2.9节：用图像填充文本 30
第3章：多边形 31
第3.1节：绘制圆角多边形 31
第3.2节：星形 32
第3.3节：正多边形 33
第4章：图像 35
第4.1节："context.drawImage"未在画布上显示图像？ 35
第4.2节：被污染的画布 35
第4.3节：使用画布裁剪图像 36
第4.4节：缩放图像以适应或填充 36
第5章：路径（仅语法） 39
第5.1节：createPattern（创建路径样式对象） 39
第5.2节：stroke（路径命令） 41
第5.3节：fill（路径命令） 45
第5.4节：clip（路径命令） 45
第5.5节：基本路径绘制命令概述：直线和曲线 47
第5.6节：lineTo（路径命令） 49
第5.7节：arc（路径命令） 50
第5.8节：quadraticCurveTo（路径命令） 52
第5.9节：bezierCurveTo（路径命令） 53
第5.10节：arcTo（路径命令） 54
第5.11节：rect（路径命令） 55
第5.12节：closePath（路径命令） 57
第5.13节：beginPath（路径命令） 58
第5.14节：lineCap（路径样式属性） 61
第5.15节：lineJoin（路径样式属性） 62
第5.16节：strokeStyle（路径样式属性） 63
第5.17节：fillStyle（路径样式属性） 65

Contents

About 1
Chapter 1: Getting started with HTML5 Canvas 2
Section 1.1: Detecting mouse position on the canvas 2
Section 1.2: Canvas size and resolution 2
Section 1.3: Rotate 3
Section 1.4: Save canvas to image file 3
Section 1.5: How to add the Html5 Canvas Element to a webpage 4
Section 1.6: An index to Html5 Canvas Capabilities & Uses 5
Section 1.7: Off screen canvas 6
Section 1.8: Hello World 6
Chapter 2: Text 8
Section 2.1: Justified text 8
Section 2.2: Justified paragraphs 13
Section 2.3: Rendering text along an arc 17
Section 2.4: Text on curve, cubic and quadratic beziers 22
Section 2.5: Drawing Text 25
Section 2.6: Formatting Text 26
Section 2.7: Wrapping text into paragraphs 27
Section 2.8: Draw text paragraphs into irregular shapes 28
Section 2.9: Fill text with an image 30
Chapter 3: Polygons 31
Section 3.1: Render a rounded polygon 31
Section 3.2: Stars 32
Section 3.3: Regular Polygon 33
Chapter 4: Images 35
Section 4.1: Is "context.drawImage" not displaying the image on the Canvas? 35
Section 4.2: The Tained canvas 35
Section 4.3: Image cropping using canvas 36
Section 4.4: Scaling image to fit or fill 36
Chapter 5: Path (Syntax only) 39
Section 5.1: createPattern (creates a path styling object) 39
Section 5.2: stroke (a path command) 41
Section 5.3: fill (a path command) 45
Section 5.4: clip (a path command) 45
Section 5.5: Overview of the basic path drawing commands: lines and curves 47
Section 5.6: lineTo (a path command) 49
Section 5.7: arc (a path command) 50
Section 5.8: quadraticCurveTo (a path command) 52
Section 5.9: bezierCurveTo (a path command) 53
Section 5.10: arcTo (a path command) 54
Section 5.11: rect (a path command) 55
Section 5.12: closePath (a path command) 57
Section 5.13: beginPath (a path command) 58
Section 5.14: lineCap (a path styling attribute) 61
Section 5.15: lineJoin (a path styling attribute) 62
Section 5.16: strokeStyle (a path styling attribute) 63
Section 5.17: fillStyle (a path styling attribute) 65

第5.18节：lineWidth（路径样式属性） 67
第5.19节：shadowColor，shadowBlur，shadowOffsetX，shadowOffsetY（路径样式属性） 68
第5.20节：createLinearGradient（创建路径样式对象） 70
第5.21节：createRadialGradient（创建路径样式对象） 73
第6章：路径 77
第6.1节：椭圆 77
第6.2节：无模糊的直线 78
第7章：沿路径导航 80
第7.1节：查找曲线上的点 80
第7.2节：寻找二次曲线的范围 81
第7.3节：寻找三次贝塞尔曲线上的点 82
第7.4节：寻找二次曲线上的点 83
第7.5节：寻找直线上的点 84
第7.6节：沿包含曲线和直线的整个路径查找点 84
第7.7节：在位置拆分贝塞尔曲线 91
第7.8节：修剪贝塞尔曲线 94
第7.9节：三次贝塞尔曲线的长度（近似值） 96
第7.10节：二次曲线的长度 97
第8章：在画布上拖动路径形状和图像 98
第8.1节：形状和图像在画布上真正的“移动”方式 98
第8.2节：在画布上拖动圆形和矩形 99
第8.3节：在画布上拖动不规则形状 103
第8.4节：在画布上拖动图像 106
第9章：媒体类型与画布 109
第9.1节：在画布上基本加载和播放视频 109
第9.2节：捕获画布并保存为webM视频 111
第9.3节：绘制svg图像 116
第9.4节：加载和显示图像 117
第10章：动画 119
第10.1节：使用requestAnimationFrame()而非setInterval()进行动画循环 119
第10.2节：在画布上动画显示图像 120
第10.3节：使用requestAnimationFrame设置帧率 121
第10.4节：使用Robert Penner的方程实现缓动 121
第10.5节：在指定间隔内动画（每1秒添加一个新矩形） 125
第10.6节：在指定时间内动画（一个动画时钟） 126
第10.7节：不要在事件处理程序中绘制动画（一个简单的绘图应用） 127
第10.8节：使用2D上下文和requestAnimationFrame的简单动画 129
第10.9节：从[x0,y0]动画到[x1,y1] 129
第11章：碰撞与交集 131
第11.1节：两个圆是否碰撞？ 131
第11.2节：两个矩形是否碰撞？ 131
第11.3节：圆和矩形是否碰撞？ 131
第11.4节：两条线段是否相交？ 131
第11.5节：线段和圆是否碰撞？ 133
第11.6节：线段和矩形是否碰撞？ 133
第11.7节：两个凸多边形是否碰撞？ 134
第11.8节：两个多边形是否碰撞？（允许凹多边形和凸多边形） 135
第11.9节：点X,Y是否在弧内？ 136
第11.10节：点X,Y是否在扇形内？ 137
第11.11节：点X,Y是否在圆内？ 138

Section 5.18: lineWidth (A path styling attribute) 67
Section 5.19: shadowColor, shadowBlur, shadowOffsetX, shadowOffsetY (path styling attributes) 68
Section 5.20: createLinearGradient (creates a path styling object) 70
Section 5.21: createRadialGradient (creates a path styling object) 73
Chapter 6: Paths 77
Section 6.1: Ellipse 77
Section 6.2: Line without blurryness 78
Chapter 7: Navigating along a Path 80
Section 7.1: Find point on curve 80
Section 7.2: Finding extent of Quadratic Curve 81
Section 7.3: Finding points along a cubic Bezier curve 82
Section 7.4: Finding points along a quadratic curve 83
Section 7.5: Finding points along a line 84
Section 7.6: Finding points along an entire Path containing curves and lines 84
Section 7.7: Split bezier curves at position 91
Section 7.8: Trim bezier curve 94
Section 7.9: Length of a Cubic Bezier Curve (a close approximation) 96
Section 7.10: Length of a Quadratic Curve 97
Chapter 8: Dragging Path Shapes & Images on Canvas 98
Section 8.1: How shapes & images REALLY(!) "move" on the Canvas 98
Section 8.2: Dragging circles & rectangles around the Canvas 99
Section 8.3: Dragging irregular shapes around the Canvas 103
Section 8.4: Dragging images around the Canvas 106
Chapter 9: Media types and the canvas 109
Section 9.1: Basic loading and playing a video on the canvas 109
Section 9.2: Capture canvas and Save as webM video 111
Section 9.3: Drawing an svg image 116
Section 9.4: Loading and displaying an Image 117
Chapter 10: Animation 119
Section 10.1: Use requestAnimationFrame() NOT setInterval() for animation loops 119
Section 10.2: Animate an image across the Canvas 120
Section 10.3: Set frame rate using requestAnimationFrame 121
Section 10.4: Easing using Robert Penners equations 121
Section 10.5: Animate at a specified interval (add a new rectangle every 1 second) 125
Section 10.6: Animate at a specified time (an animated clock) 126
Section 10.7: Don't draw animations in your event handlers (a simple sketch app) 127
Section 10.8: Simple animation with 2D context and requestAnimationFrame 129
Section 10.9: Animate from [x0,y0] to [x1,y1] 129
Chapter 11: Collisions and Intersections 131
Section 11.1: Are 2 circles colliding? 131
Section 11.2: Are 2 rectangles colliding? 131
Section 11.3: Are a circle and rectangle colliding? 131
Section 11.4: Are 2 line segments intercepting? 131
Section 11.5: Are a line segment and circle colliding? 133
Section 11.6: Are line segment and rectangle colliding? 133
Section 11.7: Are 2 convex polygons colliding? 134
Section 11.8: Are 2 polygons colliding? (both concave and convex polys are allowed) 135
Section 11.9: Is an X,Y point inside an arc? 136
Section 11.10: Is an X,Y point inside a wedge? 137
Section 11.11: Is an X,Y point inside a circle? 138

第11.12节：点X,Y是否在矩形内？ . 138
第12章：清屏 . 139
第12.1节：矩形 . 139
第12.2节：使用渐变清除画布 . 139
第12.3节：使用复合操作清除画布 . 139
第12.4节：原始图像数据 . 140
第12.5节：复杂形状 . 140
第13章：响应式设计 . 141
第13.1节：创建响应式全页画布 . 141
第13.2节：调整大小（或滚动）后的鼠标坐标 . 141
第13.3节：无调整大小事件的响应式画布动画 . 142
第14章：阴影 . 144
第14.1节：使用阴影的贴纸效果 . 144
第14.2节：如何停止进一步的阴影 . 145
第14.3节：阴影计算开销大——缓存阴影！ . 145
第14.4节：通过阴影增加视觉深度 . 146
第14.5节：内阴影 . 146
第15章：图表与示意图 . 151
第15.1节：带演示的饼图 . 151
第15.2节：带箭头的线条 . 152
第15.3节：带箭头的三次和二次贝塞尔曲线 . 153
第15.4节：楔形 . 154
第15.5节：同时具有填充和描边的弧形 . 155
第16章：变换 . 157
第16.1节：围绕中心点旋转图像或路径 . 157
第16.2节：快速绘制多个平移、缩放和旋转的图像 . 158
第16.3节：变换简介 . 159
第16.4节：用于跟踪平移、旋转和缩放形状的变换矩阵 . 160
第17章：合成 . 167
第17.1节：使用“destination-over”在现有形状后绘制 . 167
第17.2节：使用“destination-out”擦除现有形状 . 167
第17.3节：默认合成：新形状绘制在现有形状之上 . 168
第17.4节：使用“destination-in”裁剪形状内的图像 . 168
第17.5节：使用“source-in”裁剪形状内的图像 . 168
第17.6节：使用“source-atop”实现内阴影 . 169
第17.7节：使用“globalAlpha”改变不透明度 . 169
第17.8节：使用“difference”反转或取反图像 . 170
第17.9节：黑白与“彩色” . 170
第17.10节：使用“饱和度”增加颜色对比度 . 171
第17.11节：使用“亮度”的棕褐色效果 . 171
第18章：使用“getImageData”和“putImageData”的像素操作 . 173
第18.1节：“context.getImageData”简介 . 173
鸣谢 . 175
你可能也喜欢 . 176

Section 11.12: Is an X,Y point inside a rectangle? 138
Chapter 12: Clearing the screen 139
Section 12.1: Rectangles 139
Section 12.2: Clear canvas with gradient 139
Section 12.3: Clear canvas using composite operation 139
Section 12.4: Raw image data 140
Section 12.5: Complex shapes 140
Chapter 13: Responsive Design 141
Section 13.1: Creating a responsive full page canvas 141
Section 13.2: Mouse coordinates after resizing (or scrolling) 141
Section 13.3: Responsive canvas animations without resize events 142
Chapter 14: Shadows 144
Section 14.1: Sticker effect using shadows 144
Section 14.2: How to stop further shadowing 145
Section 14.3: Shadowing is computationally expensive -- Cache that shadow! 145
Section 14.4: Add visual depth with shadows 146
Section 14.5: Inner shadows 146
Chapter 15: Charts & Diagrams 151
Section 15.1: Pie Chart with Demo 151
Section 15.2: Line with arrowheads 152
Section 15.3: Cubic & Quadratic Bezier curve with arrowheads 153
Section 15.4: Wedge 154
Section 15.5: Arc with both fill and stroke 155
Chapter 16: Transformations 157
Section 16.1: Rotate an Image or Path around it's centerpoint 157
Section 16.2: Drawing many translated, scaled, and rotated images quickly 158
Section 16.3: Introduction to Transformations 159
Section 16.4: A Transformation Matrix to track translated, rotated & scaled shape(s) 160
Chapter 17: Compositing 167
Section 17.1: Draw behind existing shapes with "destination-over" 167
Section 17.2: Erase existing shapes with "destination-out" 167
Section 17.3: Default compositing: New shapes are drawn over Existing shapes 168
Section 17.4: Clip images inside shapes with "destination-in" 168
Section 17.5: Clip images inside shapes with "source-in" 168
Section 17.6: Inner shadows with "source-atop" 169
Section 17.7: Change opacity with "globalAlpha" 169
Section 17.8: Invert or Negate image with "difference" 170
Section 17.9: Black & White with "color" 170
Section 17.10: Increase the color contrast with "saturation" 171
Section 17.11: Sepia FX with "luminosity" 171
Chapter 18: Pixel Manipulation with "getImageData" and "putImageData" 173
Section 18.1: Introduction to "context.getImageData" 173
Credits 175
You may also like 176

关于

About

第1章：HTML5画布入门

第1.1节：检测画布上的鼠标位置

本示例将展示如何获取相对于画布的鼠标位置，使得(0,0)为HTML5画布的左上角。事件对象的e.clientX和e.clientY获取的是相对于文档顶部的鼠标位置，要将其转换为相对于画布顶部的位置，需要从clientX和clientY中减去画布的left和right位置。

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.font = "16px Arial";

canvas.addEventListener("mousemove", function(e) {
    var cRect = canvas.getBoundingClientRect();        // 获取CSS位置及宽高
    var canvasX = Math.round(e.clientX - cRect.left);  // 从鼠标X位置中减去画布的'left'
    var canvasY = Math.round(e.clientY - cRect.top);   // 从鼠标Y位置中减去画布的'top'
    ctx.clearRect(0, 0, canvas.width, canvas.height);  // (0,0)为画布左上角
    ctx.fillText("X: "+canvasX+", Y: "+canvasY, 10, 20);
});
```

可运行示例

使用Math.round是为了确保 x、 y位置为整数，因为画布的边界矩形可能不是整数位置。

第1.2节：画布大小和分辨率

画布的大小是其在页面上占据的区域，由CSS的宽度和高度属性定义。

```
canvas {
宽度 : 1000px;
   高度 : 1000px;
}
```

画布分辨率定义了其包含的像素数量。分辨率通过设置画布元素的宽度和高度属性来设定。如果未指定，画布默认为300×150像素。

以下画布将使用上述CSS大小，但由于未指定宽度和高度，分辨率将为300×150。

```
<canvas id="my-canvas"></canvas>
```

这将导致每个像素被不均匀拉伸。像素纵横比为1:2。当画布被拉伸时，浏览器将使用双线性滤波。这会导致被拉伸的像素出现模糊效果。

为了获得最佳效果，使用画布时请确保画布分辨率与显示尺寸匹配。

根据上述CSS样式，为匹配显示尺寸，添加画布，其width和height设置为与样式定义相同的像素数。

```
<canvas id = "my-canvas" width = "1000" height = "1000"></canvas>
```

Chapter 1: Getting started with HTML5 Canvas

Section 1.1: Detecting mouse position on the canvas

This example will show how to get the mouse position relative to the canvas, such that (0,0) will be the top-left hand corner of the HTML5 Canvas. The e.clientX and e.clientY will get the mouse positions relative to the top of the document, to change this to be based on the top of the canvas we subtract the left and right positions of the canvas from the client X and Y.

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.font = "16px Arial";

canvas.addEventListener("mousemove", function(e) {
    var cRect = canvas.getBoundingClientRect();        // Gets CSS pos, and width/height
    var canvasX = Math.round(e.clientX - cRect.left);  // Subtract the 'left' of the canvas
    var canvasY = Math.round(e.clientY - cRect.top);   // from the X/Y positions to make
    ctx.clearRect(0, 0, canvas.width, canvas.height);  // (0,0) the top left of the canvas
    ctx.fillText("X: "+canvasX+", Y: "+canvasY, 10, 20);
});
```

Runnable Example

The use of Math.round is due to ensure the x,y positions are integers, as the bounding rectangle of the canvas may not have integer positions.

Section 1.2: Canvas size and resolution

The size of a canvas is the area it occupies on the page and is defined by the CSS width and height properties.

```
canvas {
   width : 1000px;
   height : 1000px;
}
```

The canvas resolution defines the number of pixels it contains. The resolution is set by setting the canvas element width and height properties. If not specified the canvas defaults to 300 by 150 pixels.

The following canvas will use the above CSS size but as the width and height is not specified the resolution will be 300 by 150.

```
<canvas id="my-canvas"></canvas>
```

This will result in each pixel being stretched unevenly. The pixel aspect is 1:2. When the canvas is stretched the browser will use bilinear filtering. This has an effect of blurring out pixels that are stretched.

For the best results when using the canvas ensure that the canvas resolution matches the display size.

Following on from the CSS style above to match the display size add the canvas with the width and height set to the same pixel count as the style defines.

```
<canvas id = "my-canvas" width = "1000" height = "1000"></canvas>
```

第1.3节：旋转

2D上下文的rotate(r)方法将画布围绕原点按指定的弧度数 r 旋转。

HTML

```
<canvas id="canvas" width=240 height=240 style="background-color:#808080;">
</canvas>

<button type="button" onclick="rotate_ctx();">旋转上下文</button>
```

JavaScript

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
var ox = canvas.width / 2;
var oy = canvas.height / 2;
ctx.font = "42px serif";
ctx.textAlign = "center";
ctx.textBaseline = "middle";
ctx.fillStyle = "#FFF";
ctx.fillText("Hello World", ox, oy);

rotate_ctx = function() {
  // 将原点平移到画布中心 (ox, oy)
ctx.translate(ox, oy);
  // 将角度转换为弧度，弧度 = (Math.PI/180)*角度。
ctx.rotate((Math.PI / 180) * 15);
  ctx.fillText("Hello World", 0, 0);
  // 平移回原点
ctx.translate(-ox, -oy);
};
```

JSfiddle 在线演示

第1.4节：将画布保存为图像文件

你可以使用方法canvas.toDataURL()将画布保存为图像文件，该方法返回画布图像数据的data URI。

该方法可以接受两个可选参数canvas.toDataURL(type, encoderOptions)：type是图像格式（如果省略，默认是image/png）；encoderOptions是一个介于0到1之间的数字，表示图像质量（默认值为0.92）。

这里我们绘制一个画布，并将画布的数据 URI 附加到“下载到 myImage.jpg”链接上。

HTML

```
<canvas id="canvas" width=240 height=240 style="background-color:#808080;">
</canvas>
<p></p>
<a id="download" download="myImage.jpg" href="" onclick="download_img(this);">下载到
myImage.jpg</a>
```

JavaScript

```
var canvas = document.getElementById("canvas");
```

Section 1.3: Rotate

The rotate(r) method of the 2D context rotates the canvas by the specified amount r of *radians* around the origin.

HTML

```
<canvas id="canvas" width=240 height=240 style="background-color:#808080;">
</canvas>

<button type="button" onclick="rotate_ctx();">Rotate context</button>
```

JavaScript

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
var ox = canvas.width / 2;
var oy = canvas.height / 2;
ctx.font = "42px serif";
ctx.textAlign = "center";
ctx.textBaseline = "middle";
ctx.fillStyle = "#FFF";
ctx.fillText("Hello World", ox, oy);

rotate_ctx = function() {
  // translate so that the origin is now (ox, oy) the center of the canvas
  ctx.translate(ox, oy);
  // convert degrees to radians with radians = (Math.PI/180)*degrees.
  ctx.rotate((Math.PI / 180) * 15);
  ctx.fillText("Hello World", 0, 0);
  // translate back
  ctx.translate(-ox, -oy);
};
```

Live demo on JSfiddle

Section 1.4: Save canvas to image file

You can save a canvas to an image file by using the method canvas.toDataURL(), that returns the *data URI* for the canvas' image data.

The method can take two optional parameters canvas.toDataURL(type, encoderOptions): type is the image format (if omitted the default is image/png); encoderOptions is a number between 0 and 1 indicating image quality (default is 0.92).

Here we draw a canvas and attach the canvas' data URI to the "Download to myImage.jpg" link.

HTML

```
<canvas id="canvas" width=240 height=240 style="background-color:#808080;">
</canvas>
<p></p>
<a id="download" download="myImage.jpg" href="" onclick="download_img(this);">Download to
myImage.jpg</a>
```

JavaScript

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
var ox = canvas.width / 2;
var oy = canvas.height / 2;
ctx.font = "42px serif";
ctx.textAlign = "center";
ctx.textBaseline = "middle";
ctx.fillStyle = "#800";
ctx.fillRect(ox / 2, oy / 2, ox, oy);

download_img = function(el) {
  // 从canvas对象获取图像URI
  var imageURI = canvas.toDataURL("image/jpg");
  el.href = imageURI;
};
```

JSfiddle上的实时演示。

第1.5节：如何将Html5 Canvas元素添加到网页中

Html5-Canvas ...

- 是一个Html5元素。
- 支持大多数现代浏览器（Internet Explorer 9及以上版本）。
- 是一个可见元素，默认透明
- 默认宽度为300像素，默认高度为150像素。
- 需要JavaScript，因为所有内容必须通过编程方式添加到Canvas中。

示例：使用Html5标记和JavaScript创建一个Html5-Canvas元素：

```
&lt;!doctype html&gt;
&lt;html&gt;
&lt;head&gt;
&lt;style&gt;
body{ background-color:white; }
    #canvasHtml5{border:1px solid red; }
    #canvasJavascript{border:1px solid blue; }
&lt;/style&gt;
&lt;script&gt;
window.onload=(function(){

    // 使用JavaScript添加一个canvas元素
     var canvas=document.createElement('canvas');
     canvas.id='canvasJavascript'
document.body.appendChild(canvas);

}); // 结束 $(function(){});
</script>
</head>
<body>

     <!-- 使用html添加一个画布元素 -->
     <canvas id='canvasHtml5'></canvas>

</body>
</html>
```

```
var ctx = canvas.getContext("2d");
var ox = canvas.width / 2;
var oy = canvas.height / 2;
ctx.font = "42px serif";
ctx.textAlign = "center";
ctx.textBaseline = "middle";
ctx.fillStyle = "#800";
ctx.fillRect(ox / 2, oy / 2, ox, oy);

download_img = function(el) {
  // get image URI from canvas object
  var imageURI = canvas.toDataURL("image/jpg");
  el.href = imageURI;
};
```

Live demo on JSfiddle.

Section 1.5: How to add the Html5 Canvas Element to a webpage

Html5-Canvas ...

- Is an Html5 element.
- Is supported in most modern browsers (Internet Explorer 9+).
- Is a visible element that is transparent by default
- Has a default width of 300px and a default height of 150px.
- Requires JavaScript because all content must be programmatically added to the Canvas.

Example: Create an Html5-Canvas element using both Html5 markup and JavaScript:

```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvasHtml5{border:1px solid red; }
    #canvasJavascript{border:1px solid blue; }
</style>
<script>
window.onload=(function(){

    // add a canvas element using javascript
    var canvas=document.createElement('canvas');
    canvas.id='canvasJavascript'
    document.body.appendChild(canvas);

}); // end $(function(){});
</script>
</head>
<body>

    <!-- add a canvas element using html -->
    <canvas id='canvasHtml5'></canvas>

</body>
</html>
```

第1.6节：Html5画布功能与用途索引

画布的功能

画布允许你通过编程在网页上绘制：

- 图像，
- 文本，
- 线条和曲线。

画布绘图可以进行广泛的样式设置：

- 描边宽度，
- 描边颜色，
- 形状填充颜色，
- 不透明度，
- 阴影，
- 线性渐变和径向渐变，
- 字体样式，
- 字体大小，
- 文本对齐，
- 文本可以描边、填充或同时描边和填充，
- 图像调整大小，
- 图像裁剪，
- 合成

画布的用途

绘图可以组合并定位在画布上的任何位置，因此它可以用于创建：

- 绘画/素描应用，
- 快节奏的互动游戏，
- 数据分析如图表，
- 类似Photoshop的图像处理，
- 类似Flash的广告和炫目的网页内容。

Canvas允许你操作图像的红色、绿色、蓝色和Alpha（透明度）分量颜色。这使得canvas能够以类似Photoshop的效果处理图像。

- 在像素级别重新着色图像的任何部分（如果使用HSL，你甚至可以在保留重要的光照和饱和度的同时重新着色图像，这样结果看起来不会像有人随意涂抹的），
- “抠图”图像中人物/物体周围的背景，
- 检测并洪泛填充图像的部分区域（例如，将用户点击的花瓣颜色从绿色改为黄色——仅仅是被点击的那个花瓣！），
- 进行透视变形（例如，将图像包裹在杯子的曲面上），
- 检查图像内容（例如，面部识别），
- 回答关于图像的问题：这张我的停车位照片里有车停着吗？
- 应用标准图像滤镜（灰度、棕褐色等）
- 应用你能想象到的任何特殊图像滤镜（Sobel边缘检测），合成图像。如果
- 亲爱的苏奶奶没能来家庭聚会，只需将她“photoshop”进聚会照片中。不喜欢菲尔表哥——只需将他“photoshop”掉，
- 播放视频 / 从视频中抓取一帧，
- 将canvas内容导出为.jpg | .png图像（你甚至可以选择裁剪或注释图像，

Section 1.6: An index to Html5 Canvas Capabilities & Uses

Capabilities of the Canvas

Canvas lets you programmatically draw onto your webpage:

- Images,
- Texts,
- Lines and Curves.

Canvas drawings can be extensively styled:

- stroke width,
- stroke color,
- shape fill color,
- opacity,
- shadowing,
- linear gradients and radial gradients,
- font face,
- font size,
- text alignment,
- text may be stroked, filled or both stroked & filled,
- image resizing,
- image cropping,
- compositing

Uses of the Canvas

Drawings can be combined and positioned anywhere on the canvas so it can be used to create:

- Paint / Sketch applications,
- Fast paced interactive games,
- Data analyses like charts, graphs,
- Photoshop-like imaging,
- Flash-like advertising and Flashy web content.

Canvas allows you to manipulate the Red, Green, Blue & Alpha component colors of images. This allows canvas to manipulate images with results similar to Photoshop.

- Recolor any part of an image at the pixel level (if you use HSL you can even recolor an image while retaining the important Lighting & Saturation so the result doesn't look like someone slapped paint on the image),
- "Knockout" the background around a person/item in an image,
- Detect and Floodfill part of an image (eg, change the color of a user-clicked flower petal from green to yellow -- just that clicked petal!),
- Do Perspective warping (e.g. wrap an image around the curve of a cup),
- Examine an image for content (eg. facial recognition),
- Answer questions about an image: Is there a car parked in this image of my parking spot?,
- Apply standard image filters (grayscale, sepia, etc)
- Apply any exotic image filter you can dream up (Sobel Edge Detection),
- Combine images. If dear Grandma Sue couldn't make it to the family reunion, just "photoshop" her into the reunion image. Don't like Cousin Phil -- just "photoshop him out,
- Play a video / Grab a frame from a video,
- Export the canvas content as a .jpg | .png image (you can even optionally crop or annotate the image and

并将结果导出为新图像），

关于移动和编辑画布绘图（例如创建动作游戏）：

- 在画布上绘制完成后，现有的绘图无法被移动或编辑。这个关于画布绘图可移动的常见误解值得澄清：现有的画布绘图不能被编辑或移动！
- 画布绘制速度非常非常快。画布可以在一瞬间绘制数百个图像、文本、线条和曲线。它在可用时使用GPU来加速绘制。
- 画布通过快速且反复地绘制某物，然后在新位置重新绘制，来创造运动的错觉。就像电视一样，这种不断的重绘给眼睛带来运动的幻觉。

第1.7节：离屏画布

在使用画布时，很多时候你需要一个画布来保存一些原始像素数据。创建一个离屏画布并获取2D上下文非常简单。离屏画布同样会使用可用的图形硬件进行渲染。

下面的代码简单地创建了一个画布并用蓝色像素填充它。

```
function createCanvas(width, height){
     var canvas = document.createElement("canvas"); // 创建一个画布元素
    canvas.width = width;
canvas.height = height;
     return canvas;
}

var myCanvas = createCanvas(256,256); // 创建一个256乘256像素的小画布
var ctx = myCanvas.getContext("2d");
ctx.fillStyle = "blue";
ctx.fillRect(0,0,256,256);
```

离屏画布通常会用于许多任务，你可能会有多个画布。为了简化画布的使用，你可以将画布上下文附加到画布上。

```
function createCanvasCTX(width, height){
     var canvas = document.createElement("canvas"); // 创建一个画布元素
    canvas.width = width;
canvas.height = height;
canvas.ctx = canvas.getContext("2d");
     return canvas;
}
var myCanvas = createCanvasCTX(256,256); // 创建一个256乘256像素的小画布
myCanvas.ctx.fillStyle = "blue";
myCanvas.ctx.fillRect(0,0,256,256);
```

第1.8节：你好，世界

HTML

```
<canvas id="canvas" width=300 height=100 style="background-color:#808080;">
</canvas>
```

JavaScript

```
var canvas = document.getElementById("canvas");
```

export the result as a new image),

About moving and editing canvas drawings (for example to create an action game):

- After something has been drawn on the canvas, that existing drawing cannot be moved or edited. This common misconception that canvas drawings are movable is worth clarifying: *Existing canvas drawings cannot be edited or moved!*
- Canvas draws very, very quickly. Canvas can draw hundreds of images, texts, lines & curves in a fraction of a second. It uses the GPU when available to speed up drawing.
- Canvas creates the illusion of motion by quickly and repeatedly drawing something and then redrawing it in a new position. Like television, this constant redrawing gives the eye the illusion of motion.

Section 1.7: Off screen canvas

Many times when working with the canvas you will need to have a canvas to hold some intrum pixel data. It is easy to create an offscreen canvas, obtain a 2D context. An offscreen canvas will also use the available graphics hardware to render.

The following code simply creates a canvas and fills it with blue pixels.

```
function createCanvas(width, height){
    var canvas = document.createElement("canvas"); // create a canvas element
    canvas.width = width;
    canvas.height = height;
    return canvas;
}

var myCanvas = createCanvas(256,256); // create a small canvas 256 by 256 pixels
var ctx = myCanvas.getContext("2d");
ctx.fillStyle = "blue";
ctx.fillRect(0,0,256,256);
```

Many times the offscreen canvas will be used for many tasks, and you may have many canvases. To simplify the use of the canvas you can attach the canvas context to the canvas.

```
function createCanvasCTX(width, height){
    var canvas = document.createElement("canvas"); // create a canvas element
    canvas.width = width;
    canvas.height = height;
    canvas.ctx = canvas.getContext("2d");
    return canvas;
}
var myCanvas = createCanvasCTX(256,256); // create a small canvas 256 by 256 pixels
myCanvas.ctx.fillStyle = "blue";
myCanvas.ctx.fillRect(0,0,256,256);
```

Section 1.8: Hello World

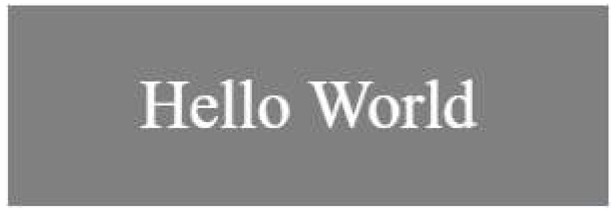
HTML

```
<canvas id="canvas" width=300 height=100 style="background-color:#808080;">
</canvas>
```

JavaScript

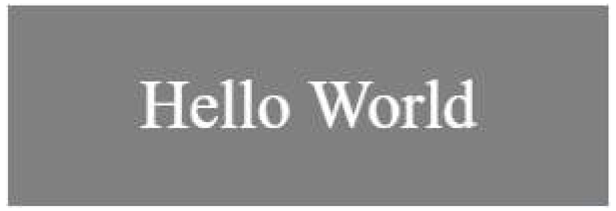
```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
ctx.font = "34px serif";
ctx.textAlign = "center";
ctx.textBaseline="middle";
ctx.fillStyle = "#FFF";
ctx.fillText("Hello World",150,50);
```

结果



```
var ctx = canvas.getContext("2d");
ctx.font = "34px serif";
ctx.textAlign = "center";
ctx.textBaseline="middle";
ctx.fillStyle = "#FFF";
ctx.fillText("Hello World",150,50);
```

Result

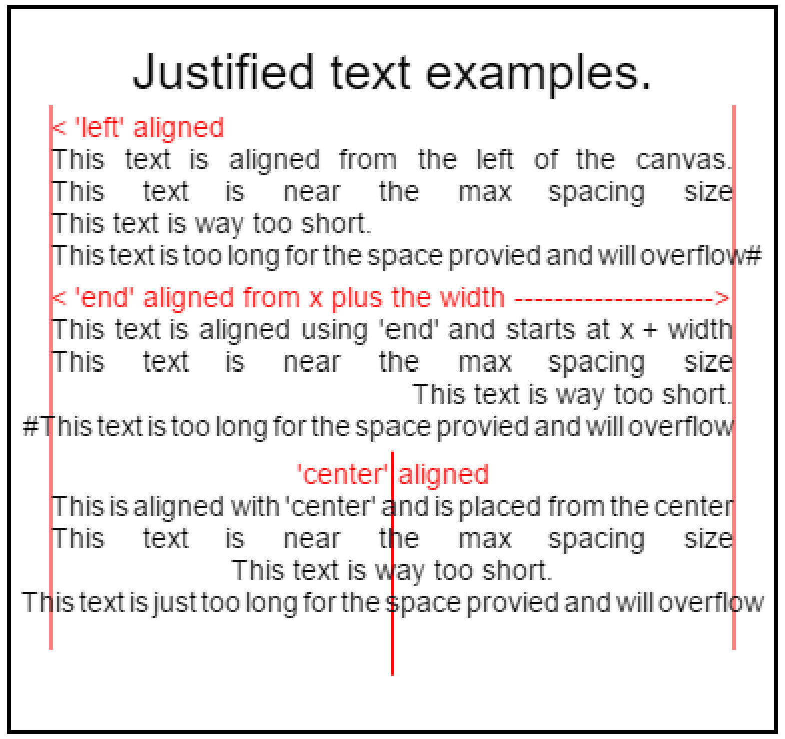


第二章：文本

2.1节：两端对齐文本

此示例渲染两端对齐文本。它通过扩展CanvasRenderingContext2D的原型或作为全局对象justifiedText（可选，见注释A）来增加额外功能。

示例渲染。



```
Justified text examples.
< 'left' aligned
This text is aligned from the left of the canvas.
This text is near the max spacing size
This text is way too short.
This text is too long for the space provied and will overflow#
< 'end' aligned from x plus the width -------------------->
This text is aligned using 'end' and starts at x + width
This text is near the max spacing size
This text is way too short.
#This text is too long for the space provied and will overflow
'center' aligned
This is aligned with 'center' and is placed from the center
This text is near the max spacing size
This text is way too short.
This text is just too long for the space provied and will overflow
```

渲染此图像的代码位于底部的使用示例中。

示例

该函数作为一个匿名立即调用函数。

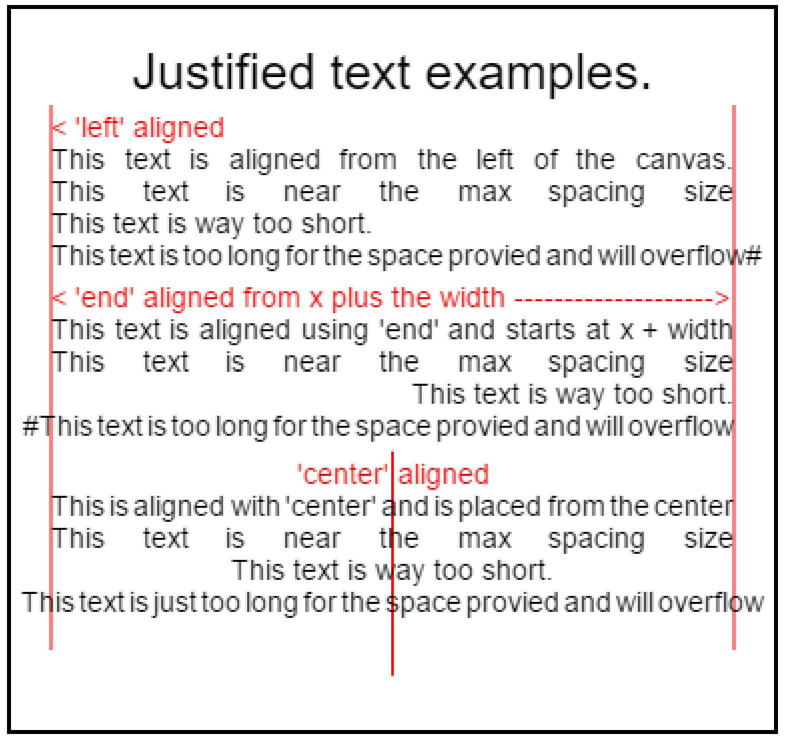
```
(function(){
    const FILL = 0;        // 用于指示填充文本渲染的常量
    const STROKE = 1;
    const MEASURE = 2;
    var renderType = FILL; // 内部使用以设置填充或描边文本

    var maxSpaceSize = 3; // 最大空格大小的乘数。如果更大则不应用对齐
    var minSpaceSize = 0.5; // 最小空格大小的乘数
    var renderTextJustified = function(ctx,text,x,y,width){
        var words, wordsWidth, count, spaces, spaceWidth, adjSpace, renderer, i, textAlign,
useSize, totalWidth;
textAlign = ctx.textAlign; // 获取当前对齐设置
        ctx.textAlign = "left";
wordsWidth = 0;
        words = text.split(" ").map(word => {
            var w = ctx.measureText(word).width;
            wordsWidth += w;
            return {
width : w,
                word : word,
```

Chapter 2: Text

Section 2.1: Justified text

This example renders justified text. It adds extra functionality to the `CanvasRenderingContext2D` by extending its prototype or as a global object `justifiedText` (optional see Note A).

Example rendering.



```
Justified text examples.
< 'left' aligned
This text is aligned from the left of the canvas.
This text is near the max spacing size
This text is way too short.
This text is too long for the space provied and will overflow#
< 'end' aligned from x plus the width -------------------->
This text is aligned using 'end' and starts at x + width
This text is near the max spacing size
This text is way too short.
#This text is too long for the space provied and will overflow
'center' aligned
This is aligned with 'center' and is placed from the center
This text is near the max spacing size
This text is way too short.
This text is just too long for the space provied and will overflow
```

Code to render this image is in the usage examples at the bottom.

The Example

The function as a anonymous immediately invoked function.

```
(function(){
    const FILL = 0;        // const to indicate filltext render
    const STROKE = 1;
    const MEASURE = 2;
    var renderType = FILL; // used internal to set fill or stroke text

    var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justificatoin applied
    var minSpaceSize = 0.5; // Multiplier for minimum space size
    var renderTextJustified = function(ctx,text,x,y,width){
        var words, wordsWidth, count, spaces, spaceWidth, adjSpace, renderer, i, textAlign,
useSize, totalWidth;
        textAlign = ctx.textAlign; // get current align settings
        ctx.textAlign = "left";
        wordsWidth = 0;
        words = text.split(" ").map(word => {
            var w = ctx.measureText(word).width;
            wordsWidth += w;
            return {
                width : w,
                word : word,
            };
        });
        // count = 单词数, spaces = 空格数, spaceWidth 正常空格大小
        // adjSpace 新空格大小 >= 最小大小。useSize 用于渲染的最终空格大小
count = words.length;
spaces = count - 1;
spaceWidth = ctx.measureText(" ").width;
        adjSpace = Math.max(spaceWidth * minSpaceSize, (width - wordsWidth) / spaces);
        useSize = adjSpace > spaceWidth * maxSpaceSize ? spaceWidth : adjSpace;
totalWidth = wordsWidth + useSize * spaces
        if(renderType === MEASURE){ // 如果是测量则返回大小
          ctx.textAlign = textAlign;
            return totalWidth;
        }
              renderer = renderType === FILL ? ctx.fillText.bind(ctx) : ctx.strokeText.bind(ctx); // 填充
或描边
        switch(textAlign){
            case "right":
x -= totalWidth;
                break;
            case "end":
x += width - totalWidth;
                break;
            case "center": // 有意落入默认情况
             x -= totalWidth / 2;
            default:
        }
        if(useSize === spaceWidth){ // 如果空格大小未改变
          renderer(text,x,y);
        } else {
            for(i = 0; i < count; i += 1){
                renderer(words[i].word,x,y);
                x += words[i].width;
x += useSize;
            }
        }
ctx.textAlign = textAlign;
    }
    // 解析 vet 并设置 settings 对象。
    var justifiedTextSettings = function(settings){
        var min,max;
        var vetNumber = (num, defaultNum) => {
num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
            if(num < 0){
num = defaultNum;
            }
            return num;
        }
        if(settings === undefined || settings === null){
            return;
        }
max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
        min = vetNumber(settings.minSpaceSize, minSpaceSize);
        if(min > max){
            return;
        }
minSpaceSize = min;
        maxSpaceSize = max;
    }
    // 定义填充文本
    var fillJustifyText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
```

```
            };
        });
        // count = num words, spaces = number spaces, spaceWidth normal space size
        // adjSpace new space size >= min size. useSize Resulting space size used to render
        count = words.length;
        spaces = count - 1;
        spaceWidth = ctx.measureText(" ").width;
        adjSpace = Math.max(spaceWidth * minSpaceSize, (width - wordsWidth) / spaces);
        useSize = adjSpace > spaceWidth * maxSpaceSize ? spaceWidth : adjSpace;
        totalWidth = wordsWidth + useSize * spaces
        if(renderType === MEASURE){ // if measuring return size
            ctx.textAlign = textAlign;
            return totalWidth;
        }
        renderer = renderType === FILL ? ctx.fillText.bind(ctx) : ctx.strokeText.bind(ctx); // fill
or stroke
        switch(textAlign){
            case "right":
                x -= totalWidth;
                break;
            case "end":
                x += width - totalWidth;
                break;
            case "center": // intentional fall through to default
                x -= totalWidth / 2;
            default:
        }
        if(useSize === spaceWidth){ // if space size unchanged
            renderer(text,x,y);
        } else {
            for(i = 0; i < count; i += 1){
                renderer(words[i].word,x,y);
                x += words[i].width;
                x += useSize;
            }
        }
        ctx.textAlign = textAlign;
    }
    // Parse vet and set settings object.
    var justifiedTextSettings = function(settings){
        var min,max;
        var vetNumber = (num, defaultNum) => {
            num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
            if(num < 0){
                num = defaultNum;
            }
            return num;
        }
        if(settings === undefined || settings === null){
            return;
        }
        max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
        min = vetNumber(settings.minSpaceSize, minSpaceSize);
        if(min > max){
            return;
        }
        minSpaceSize = min;
        maxSpaceSize = max;
    }
    // define fill text
    var fillJustifyText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
```

```
renderType = FILL;
renderTextJustified(this, text, x, y, width);
    }
    // define stroke text
    var strokeJustifyText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
        renderType = STROKE;
renderTextJustified(this, text, x, y, width);
    }
    // define measure text
    var measureJustifiedText = function(text, width, settings){
        justifiedTextSettings(settings);
        renderType = MEASURE;
        return renderTextJustified(this, text, 0, 0, width);
    }
    // code point A
    // 设置原型
CanvasRenderingContext2D.prototype.fillJustifyText = fillJustifyText;
    CanvasRenderingContext2D.prototype.strokeJustifyText = strokeJustifyText;
    CanvasRenderingContext2D.prototype.measureJustifiedText = measureJustifiedText;
    // 代码点 B

    // 可选代码，如果您不想扩展 CanvasRenderingContext2D 原型
    /* 从这里开始取消注释直到结束注释
    window.justifiedText = {
fill : function(ctx, text, x, y, width, settings){
            justifiedTextSettings(settings);
renderType = FILL;
            renderTextJustified(ctx, text, x, y, width);
        },
stroke : function(ctx, text, x, y, width, settings){
            justifiedTextSettings(settings);
renderType = STROKE;
            renderTextJustified(ctx, text, x, y, width);
        },
measure : function(ctx, text, width, settings){
            justifiedTextSettings(settings);
renderType = MEASURE;
            return renderTextJustified(ctx, text, 0, 0, width);
        }
    }
到这里*/
})();
```

> 注意 A： 如果您不想扩展 CanvasRenderingContext2D 原型，请从示例中删除所有位于 // code point A 和 // code point B 之间的代码，并取消注释标记为 /*从这里取消注释直到结束注释的代码

使用方法

向 CanvasRenderingContext2D 添加了三个函数，所有创建的 2D 上下文对象均可使用。

- ctx.fillJustifyText(text, x, y, width, [settings]);
- ctx.strokeJustifyText(text, x, y, width, [settings]);
- ctx.measureJustifiedText(text, width, [settings]);

填充和描边文本函数用于填充或描边文本，参数相同。 measureJustifiedText 将返回

```
        renderType = FILL;
        renderTextJustified(this, text, x, y, width);
    }
    // define stroke text
    var strokeJustifyText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
        renderType = STROKE;
        renderTextJustified(this, text, x, y, width);
    }
    // define measure text
    var measureJustifiedText = function(text, width, settings){
        justifiedTextSettings(settings);
        renderType = MEASURE;
        return renderTextJustified(this, text, 0, 0, width);
    }
    // code point A
    // set the prototypes
    CanvasRenderingContext2D.prototype.fillJustifyText = fillJustifyText;
    CanvasRenderingContext2D.prototype.strokeJustifyText = strokeJustifyText;
    CanvasRenderingContext2D.prototype.measureJustifiedText = measureJustifiedText;
    // code point B

    // optional code if you do not wish to extend the CanvasRenderingContext2D prototype
    /* Uncomment from here to the closing comment
    window.justifiedText = {
        fill : function(ctx, text, x, y, width, settings){
            justifiedTextSettings(settings);
            renderType = FILL;
            renderTextJustified(ctx, text, x, y, width);
        },
        stroke : function(ctx, text, x, y, width, settings){
            justifiedTextSettings(settings);
            renderType = STROKE;
            renderTextJustified(ctx, text, x, y, width);
        },
        measure : function(ctx, text, width, settings){
            justifiedTextSettings(settings);
            renderType = MEASURE;
            return renderTextJustified(ctx, text, 0, 0, width);
        }
    }
    to here*/
})();
```

> **Note A:** If you do not wish to extend the `CanvasRenderingContext2D` prototype Remove from the example all code between `// code point A` and `// code point B` and uncomment the code marked `/* Uncomment from here to the closing comment`

How to use

Three functions are added to the `CanvasRenderingContext2D` and are available to all 2D context objects created.

- ctx.fillJustifyText(text, x, y, width, [settings]);
- ctx.strokeJustifyText(text, x, y, width, [settings]);
- ctx.measureJustifiedText(text, width, [settings]);

Fill and stroke text function fill or stroke text and use the same arguments. `measureJustifiedText` will return the

文本实际渲染的宽度。根据当前设置，该宽度可能等于、小于或大于参数 width。

> 注意： 方括号 [和] 内的参数为可选项。

函数参数

- text：要渲染的文本字符串。
- x, y：渲染文本的坐标。
- **width：** 对齐文本的宽度。文本将通过增加/减少单词间的空格来适应宽度。如果单词间的空格大于 maxSpaceSize（默认值 = 6）倍，则将使用正常间距，文本将不会填满所需宽度。如果间距小于 minSpaceSize（默认值 = 0.5）倍正常间距，则使用最小间距，文本将超出请求的宽度。
- settings：可选。包含最小和最大空格大小的对象。

settings 参数是可选的，如果未包含，文本渲染将使用最后定义的设置或默认值（如下所示）。

最小值和最大值均为分隔单词的[空格]字符的最小和最大尺寸。默认的 maxSpaceSize= 6 表示当字符间空格大于 6 * ctx.measureText(" ").width 时，文本将不进行对齐。如果要对齐的文本中空格小于 minSpaceSize = 0.5（默认值 0.5）* ctx.measureText(" ").width，则间距将设置为 minSpaceSize * ctx.measureText(" ").width，结果文本将超出对齐宽度。

应用以下规则，min 和 max 必须是数字。如果不是，则相关值不会被更改。如果 minSpaceSize 大于 maxSpaceSize，则两个输入设置均无效，min 和 max 不会被更改。

带有默认值的示例设置对象

```
settings = {
maxSpaceSize : 6;   // 最大空间大小的乘数。
minSpaceSize : 0.5; // 最小间距大小的乘数
};
```

> 注意： 这些文本函数对2D上下文的textAlign属性引入了细微的行为变化。'left'、'right'、'center'和'start'的行为如预期，但'end'不会从函数参数 x 的右侧对齐，而是从 x + width 的右侧对齐

> 注意： 设置（最小和最大间距大小）对所有2D上下文对象都是全局的。

用法示例

```
var i = 0;
text[i++] = "这段文本从画布的左侧对齐。";
text[i++] = "这段文本接近最大间距大小";
text[i++] = "这段文本太短了。";
text[i++] = "这段文本对于提供的空间来说太长，将会溢出#";
```

actual width that text would be rendered at. This may be equal, less, or greater than the argument width depending on current settings.

> **Note:** Arguments inside [and] are optional.

Function arguments

- **text:** String containing the text to be rendered.
- **x, y:** Coordinates to render the text at.
- **width:** Width of the justified text. Text will increase/decrease spaces between words to fit the width. If the space between words is greater than maxSpaceSize (default = 6) times normal spacing will be used and the text will not fill the required width. If the spacing is less than minSpaceSize (default = 0.5) time normal spacing then the min space size is used and the text will overrun the width requested
- **settings:** Optional. Object containing min and max space sizes.

The settings argument is optional and if not included text rendering will use the last setting defined or the default (shown below).

Both min and max are the min and max sizes for the [space] character separating words. The default maxSpaceSize = 6 means that when the space between characters is > 63 * ctx.measureText(" ").width text will not be justified. If text to be justified has spaces less than minSpaceSize = 0.5 (default value 0.5) * ctx.measureText(" ").width the spacing will be set to minSpaceSize * ctx.measureText(" ").width and the resulting text will overrun the justifying width.

The following rules are applied, min and max must be numbers. If not then the associate values will not be changed. If minSpaceSize is larger than maxSpaceSize both input setting are invalid and min max will not be changed.

Example setting object with defaults

```
settings = {
    maxSpaceSize : 6;   // Multiplier for max space size.
    minSpaceSize : 0.5; // Multiplier for minimum space size
};
```

> **NOTE:** These text functions introduce a subtle behaviour change for the textAlign property of the 2D context. 'Left', 'right', 'center' and 'start' behave as is expected but 'end' will not align from the right of the function argument x but rather from the right of x + width

> **Note:** settings (min and max space size) are global to all 2D context objects.

USAGE Examples

```
var i = 0;
text[i++] = "This text is aligned from the left of the canvas.";
text[i++] = "This text is near the max spacing size";
text[i++] = "This text is way too short.";
text[i++] = "This text is too long for the space provied and will overflow#";
```

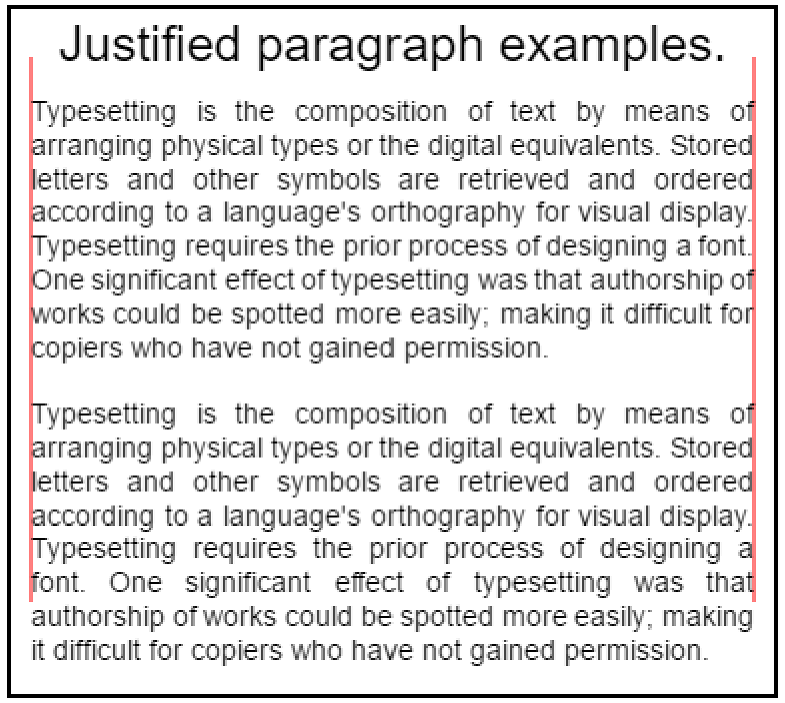
```
text[i++] = "这段文本使用'end'对齐，起始于 x + width";
text[i++] = "这段文本接近最大间距大小";
text[i++] = "这段文本太短了。";
text[i++] = "#这段文本对于提供的空间来说太长，将会溢出";
text[i++] = "这段文本使用'center'对齐，起始于中心位置";
text[i++] = "这段文本接近最大间距大小";
text[i++] = "这段文字太短了。";
text[i++] = "这段文字对于提供的空间来说太长了，会溢出";

// ctx 是二维上下文
// canvas 是画布

ctx.clearRect(0,0,w,h);
ctx.font = "25px arial";
ctx.textAlign = "center"
var left = 20;
var center = canvas.width / 2;
var width = canvas.width-left*2;
var y = 40;
var size = 16;
var i = 0;
ctx.fillText("两端对齐文本示例。",center,y);
y+= 40;
ctx.font = "14px arial";
ctx.textAlign = "left"
var ww = ctx.measureJustifiedText(text[0], width);
var setting = {
    maxSpaceSize : 6,
    minSpaceSize : 0.5
}
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(left,y - size * 2);
ctx.lineTo(left, y + size * 15);
ctx.moveTo(canvas.width - left,y - size * 2);
ctx.lineTo(canvas.width - left, y + size * 15);
ctx.stroke();
ctx.textAlign = "left";
ctx.fillStyle = "red";
ctx.fillText("< 'left' 对齐",left,y - size)
ctx.fillStyle = "black";
ctx.fillJustifyText(text[i++], left, y, width, setting);   // 设置被记住
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
y += 2.3*size;
ctx.fillStyle = "red";
ctx.fillText("< 从 x 加宽度开始的 'end' 对齐 -------------------->",left,y - size)
ctx.fillStyle = "black";
ctx.textAlign = "end";
ctx.fillJustifyText(text[i++], left, y, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);

y += 40;
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(center,y - size * 2);
ctx.lineTo(center, y + size * 5);
ctx.stroke();
ctx.textAlign = "center";
```

```
text[i++] = "This text is aligned using 'end' and starts at x + width";
text[i++] = "This text is near the max spacing size";
text[i++] = "This text is way too short.";
text[i++] = "#This text is too long for the space provied and will overflow";
text[i++] = "This is aligned with 'center' and is placed from the center";
text[i++] = "This text is near the max spacing size";
text[i++] = "This text is way too short.";
text[i++] = "This text is just too long for the space provied and will overflow";

// ctx is the 2d context
// canvas is the canvas

ctx.clearRect(0,0,w,h);
ctx.font = "25px arial";
ctx.textAlign = "center"
var left = 20;
var center = canvas.width / 2;
var width = canvas.width-left*2;
var y = 40;
var size = 16;
var i = 0;
ctx.fillText("Justified text examples.",center,y);
y+= 40;
ctx.font = "14px arial";
ctx.textAlign = "left"
var ww = ctx.measureJustifiedText(text[0], width);
var setting = {
    maxSpaceSize : 6,
    minSpaceSize : 0.5
}
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(left,y - size * 2);
ctx.lineTo(left, y + size * 15);
ctx.moveTo(canvas.width - left,y - size * 2);
ctx.lineTo(canvas.width - left, y + size * 15);
ctx.stroke();
ctx.textAlign = "left";
ctx.fillStyle = "red";
ctx.fillText("< 'left' aligned",left,y - size)
ctx.fillStyle = "black";
ctx.fillJustifyText(text[i++], left, y, width, setting);  // settings is remembered
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
y += 2.3*size;
ctx.fillStyle = "red";
ctx.fillText("< 'end' aligned from x plus the width -------------------->",left,y - size)
ctx.fillStyle = "black";
ctx.textAlign = "end";
ctx.fillJustifyText(text[i++], left, y, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);

y += 40;
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(center,y - size * 2);
ctx.lineTo(center, y + size * 5);
ctx.stroke();
ctx.textAlign = "center";
ctx.fillStyle = "red";
ctx.fillText("'center' aligned",center,y - size)
ctx.fillStyle = "black";
ctx.fillJustifyText(text[i++], center, y, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);
```

第2.2节：两端对齐段落

将文本渲染为两端对齐的段落。 需要示例两端对齐文本

示例渲染


Justified paragraph examples.

Typesetting is the composition of text by means of arranging physical types or the digital equivalents. Stored letters and other symbols are retrieved and ordered according to a language's orthography for visual display. Typesetting requires the prior process of designing a font. One significant effect of typesetting was that authorship of works could be spotted more easily; making it difficult for copiers who have not gained permission.

Typesetting is the composition of text by means of arranging physical types or the digital equivalents. Stored letters and other symbols are retrieved and ordered according to a language's orthography for visual display. Typesetting requires the prior process of designing a font. One significant effect of typesetting was that authorship of works could be spotted more easily; making it difficult for copiers who have not gained permission.

顶部段落设置了setting.compact = true，底部为false，行距为1.2而非默认的1.5。
由本示例底部的代码用法示例渲染。

示例代码

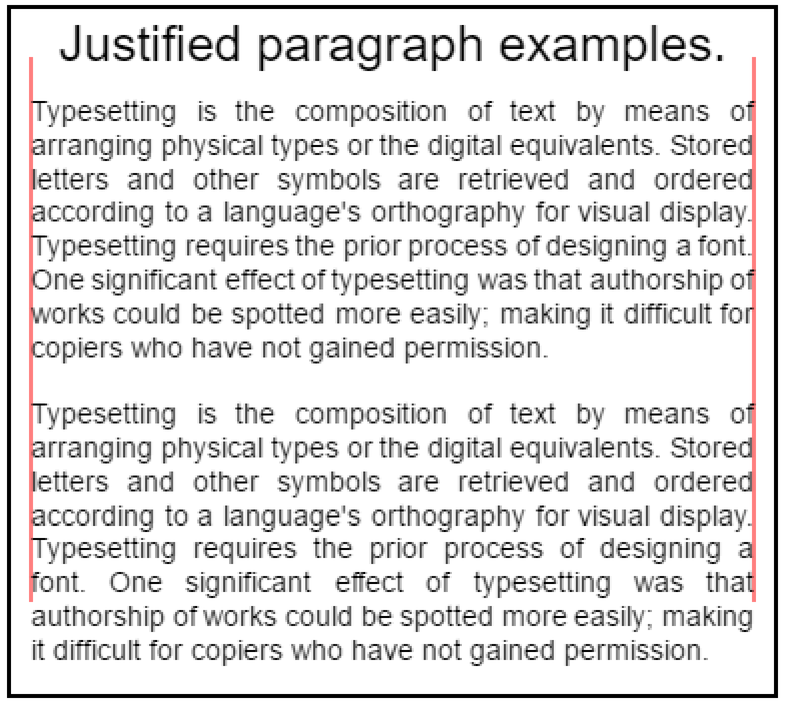
```
 // 需要两端对齐文本扩展
(function(){
    // code point A
    if(typeof CanvasRenderingContext2D.prototype.fillJustifyText !== "function"){
        throw new ReferenceError("缺少必需的
CanvasRenderingContext2D 对齐文本扩展的对齐段落扩展");
    }
    var maxSpaceSize = 3; // 最大空格大小的乘数。如果更大，则不应用对齐
    var minSpaceSize = 0.5; // 最小空格大小的乘数
    var compact = true; // 如果为 true，则尽可能多地容纳单词。如果为 false，则尽量使间距接近正常

    var lineSpacing = 1.5; // 行间距
    const noJustifySetting = {  // 此设置强制关闭两端对齐。用于渲染段落的最后一行

        minSpaceSize : 1,
        maxSpaceSize : 1,
    }
```

```
ctx.fillStyle = "red";
ctx.fillText("'center' aligned",center,y - size)
ctx.fillStyle = "black";
ctx.fillJustifyText(text[i++], center, y, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);
```

Section 2.2: Justified paragraphs

Renders text as justified paragraphs. **REQUIRES** the example **Justified text**

Example render


Justified paragraph examples.

Typesetting is the composition of text by means of arranging physical types or the digital equivalents. Stored letters and other symbols are retrieved and ordered according to a language's orthography for visual display. Typesetting requires the prior process of designing a font. One significant effect of typesetting was that authorship of works could be spotted more easily; making it difficult for copiers who have not gained permission.

Typesetting is the composition of text by means of arranging physical types or the digital equivalents. Stored letters and other symbols are retrieved and ordered according to a language's orthography for visual display. Typesetting requires the prior process of designing a font. One significant effect of typesetting was that authorship of works could be spotted more easily; making it difficult for copiers who have not gained permission.

Top paragraph has **setting.compact = true** and bottom **false** and line spacing is **1.2** rather than the default **1.5**. Rendered by code usage example bottom of this example.

Example code

```
 // Requires justified text extensions
(function(){
    // code point A
    if(typeof CanvasRenderingContext2D.prototype.fillJustifyText !== "function"){
        throw new ReferenceError("Justified Paragraph extension missing requiered
CanvasRenderingContext2D justified text extension");
    }
    var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justificatoin applied
    var minSpaceSize = 0.5; // Multiplier for minimum space size
    var compact = true; // if true then try and fit as many words as possible. If false then try to
get the spacing as close as possible to normal
    var lineSpacing = 1.5; // space between lines
    const noJustifySetting = {  // This setting forces justified text off. Used to render last line
of paragraph.
        minSpaceSize : 1,
        maxSpaceSize : 1,
    }
    // 解析 vet 并设置 settings 对象。
    var justifiedTextSettings = function(settings){
        var min, max;
        var vetNumber = (num, defaultNum) => {
num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
            return num < 0 ? defaultNum : num;
        }
        if(settings === undefined || settings === null){ return; }
        compact = settings.compact === true ? true : settings.compact === false ? false : compact;
        max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
        min = vetNumber(settings.minSpaceSize, minSpaceSize);
        lineSpacing = vetNumber(settings.lineSpacing, lineSpacing);
        if(min > max){ return; }
        minSpaceSize = min;
        maxSpaceSize = max;
    }
    var getFontSize = function(font){  // 获取字体大小。
        var numFind = /[0-9]+/;
        var number = numFind.exec(font)[0];
        if(isNaN(number)){
            throw new ReferenceError("justifiedPar 找不到字体大小");
        }
        return Number(number);
    }
    function justifiedPar(ctx, text, x, y, width, settings, stroke){
        var spaceWidth, minS, maxS, words, count, lines, lineWidth, lastLineWidth, lastSize, i,
renderer, fontSize, adjSpace, spaces, word, lineWords, lineFound;
        spaceWidth = ctx.measureText(" ").width;
minS = spaceWidth * minSpaceSize;
       maxS = spaceWidth * maxSpaceSize;
       words = text.split(" ").map(word => {  // 测量所有单词。
            var w = ctx.measureText(word).width;
            return {
width : w,
                word : word,
            };
        });
        // count = 单词数，spaces = 空格数，spaceWidth 正常空格大小
        // adjSpace 新空格大小 >= 最小大小。useSize 用于渲染的最终空格大小
count = 0;
        lines = [];
        // 通过从单词数组中移动单词来创建行，直到间距达到最佳。如果
紧凑
        // 如果为真，则为真并尽可能匹配更多单词。否则，它将尝试使
间距尽可能接近正常间距
        // 尽可能接近正常间距
while(words.length > 0){
            lastLineWidth = 0;
            lastSize = -1;
            lineFound = false;
            // 每行必须至少有一个单词。
word = words.shift();
            lineWidth = word.width;
            lineWords = [word.word];
            count = 0;
while(lineWidth < width && words.length > 0){ // 向行中添加单词
             word = words.shift();
lineWidth += word.width;
                lineWords.push(word.word);
                count += 1;
spaces = count - 1;
                adjSpace =  (width - lineWidth) / spaces;
```

```
    // Parse vet and set settings object.
    var justifiedTextSettings = function(settings){
        var min, max;
        var vetNumber = (num, defaultNum) => {
            num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
            return num < 0 ? defaultNum : num;
        }
        if(settings === undefined || settings === null){ return; }
        compact = settings.compact === true ? true : settings.compact === false ? false : compact;
        max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
        min = vetNumber(settings.minSpaceSize, minSpaceSize);
        lineSpacing = vetNumber(settings.lineSpacing, lineSpacing);
        if(min > max){ return; }
        minSpaceSize = min;
        maxSpaceSize = max;
    }
    var getFontSize = function(font){  // get the font size.
        var numFind = /[0-9]+/;
        var number = numFind.exec(font)[0];
        if(isNaN(number)){
            throw new ReferenceError("justifiedPar Cant find font size");
        }
        return Number(number);
    }
    function justifiedPar(ctx, text, x, y, width, settings, stroke){
        var spaceWidth, minS, maxS, words, count, lines, lineWidth, lastLineWidth, lastSize, i,
renderer, fontSize, adjSpace, spaces, word, lineWords, lineFound;
        spaceWidth = ctx.measureText(" ").width;
        minS = spaceWidth * minSpaceSize;
        maxS = spaceWidth * maxSpaceSize;
        words = text.split(" ").map(word => {  // measure all words.
            var w = ctx.measureText(word).width;
            return {
                width : w,
                word : word,
            };
        });
        // count = num words, spaces = number spaces, spaceWidth normal space size
        // adjSpace new space size >= min size. useSize Resulting space size used to render
        count = 0;
        lines = [];
        // create lines by shifting words from the words array until the spacing is optimal. If
compact
        // true then will true and fit as many words as possible. Else it will try and get the
spacing as
        // close as possible to the normal spacing
        while(words.length > 0){
            lastLineWidth = 0;
            lastSize = -1;
            lineFound = false;
            // each line must have at least one word.
            word = words.shift();
            lineWidth = word.width;
            lineWords = [word.word];
            count = 0;
            while(lineWidth < width && words.length > 0){ // Add words to line
                word = words.shift();
                lineWidth += word.width;
                lineWords.push(word.word);
                count += 1;
                spaces = count - 1;
                adjSpace =  (width - lineWidth) / spaces;
                if(minS > adjSpace){  // 如果间距小于最小值，则移除最后一个单词并结束该行
                 lineFound = true;
words.unshift(word);
lineWords.pop();
                } else{
                    if(!compact){ // 如果不是紧凑模式
                        if(adjSpace < spaceWidth){ // 如果小于正常空格宽度
                            if(lastSize === -1){
lastSize = adjSpace;
                            }
                            // 检查与最后一个单词的距离是否更接近空格宽度
                            if(Math.abs(spaceWidth - adjSpace) < Math.abs(spaceWidth - lastSize)){
                           lineFound = true; // 是，保留它
                            }else{
words.unshift(word);  // 否，如果移除最后一个单词会更合适
                           lineWords.pop();
lineFound = true;
                            }
                        }
                    }
                }
lastSize = adjSpace; // 记住间距
            }
lines.push(lineWords.join(" ")); // 和这一行
        }
        // 行已经处理完，获取字体大小，渲染，并渲染所有行。最后一行可能需要作为普通行渲染，因此放在循环外
        。
fontSize = getFontSize(ctx.font);
        renderer = stroke === true ? ctx.strokeJustifyText.bind(ctx) :
ctx.fillJustifyText.bind(ctx);
        for(i = 0; i < lines.length - 1; i ++){
            renderer(lines[i], x, y, width, settings);
            y += lineSpacing * fontSize;
        }
        if(lines.length > 0){ // 最后一行如果是左对齐或起始对齐且不做两端对齐
            if(ctx.textAlign === "left" || ctx.textAlign === "start"){
renderer(lines[lines.length - 1], x, y, width, noJustifySetting);
                ctx.measureJustifiedText("", width, settings);
            } else{
renderer(lines[lines.length - 1], x, y, width);
            }
        }
        // 返回关于段落的详细信息。
y += lineSpacing * fontSize;
        return {
nextLine : y,
            fontSize : fontSize,
            lineHeight : lineSpacing * fontSize,
        };
    }
    // define fill
    var fillParagraphText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
        settings = {
minSpaceSize : minSpaceSize,
            maxSpaceSize : maxSpaceSize,
        };
        return justifiedPar(this, text, x, y, width, settings);
    }
    // define stroke
    var strokeParagraphText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
```

```
                if(minS > adjSpace){  // if spacing less than min remove last word and finish line
                    lineFound = true;
                    words.unshift(word);
                    lineWords.pop();
                }else{
                    if(!compact){ // if compact mode
                        if(adjSpace < spaceWidth){ // if less than normal space width
                            if(lastSize === -1){
                                lastSize = adjSpace;
                            }
                            // check if with last word on if its closer to space width
                            if(Math.abs(spaceWidth - adjSpace) < Math.abs(spaceWidth - lastSize)){
                                lineFound = true; // yes keep it
                            }else{
                                words.unshift(word);  // no better fit if last word removes
                                lineWords.pop();
                                lineFound = true;
                            }
                        }
                    }
                }
                lastSize = adjSpace; // remember spacing
            }
            lines.push(lineWords.join(" ")); // and the line
        }
        // lines have been worked out get font size, render, and render all the lines. last
        // line may need to be rendered as normal so it is outside the loop.
        fontSize = getFontSize(ctx.font);
        renderer = stroke === true ? ctx.strokeJustifyText.bind(ctx) :
ctx.fillJustifyText.bind(ctx);
        for(i = 0; i < lines.length - 1; i ++){
            renderer(lines[i], x, y, width, settings);
            y += lineSpacing * fontSize;
        }
        if(lines.length > 0){ // last line if left or start aligned for no justify
            if(ctx.textAlign === "left" || ctx.textAlign === "start"){
                renderer(lines[lines.length - 1], x, y, width, noJustifySetting);
                ctx.measureJustifiedText("", width, settings);
            }else{
                renderer(lines[lines.length - 1], x, y, width);
            }
        }
        // return details about the paragraph.
        y += lineSpacing * fontSize;
        return {
            nextLine : y,
            fontSize : fontSize,
            lineHeight : lineSpacing * fontSize,
        };
    }
    // define fill
    var fillParagraphText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
        settings = {
            minSpaceSize : minSpaceSize,
            maxSpaceSize : maxSpaceSize,
        };
        return justifiedPar(this, text, x, y, width, settings);
    }
    // define stroke
    var strokeParagraphText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
```

```
settings = {
            minSpaceSize : minSpaceSize,
            maxSpaceSize : maxSpaceSize,
        };
        return justifiedPar(this, text, x, y, width, settings,true);
    }
CanvasRenderingContext2D.prototype.fillParaText = fillParagraphText;
    CanvasRenderingContext2D.prototype.strokeParaText = strokeParagraphText;
})();
```

> 注意 这会扩展CanvasRenderingContext2D的原型。如果您不希望这样发生，请使用示例Justified text来了解如何将此示例更改为全局命名空间的一部分。

> 注意 如果此示例找不到该函数，将抛出ReferenceError错误
> CanvasRenderingContext2D.prototype.fillJustifyText

使用方法

```
ctx.fillParaText(text, x, y, width, [settings]);
ctx.strokeParaText(text, x, y, width, [settings]);
```

有关参数详情，请参见Justified text。方括号[和]中的参数为可选项。

settings参数有两个额外属性。

- compact: 默认值为true。如果为true，尝试在每行中尽可能多地排列单词。如果为false，则尝试使单词间距尽可能接近正常间距。
- lineSpacing 默认值为1.5。每行间距默认1.5，表示从一行到下一行的距离，以字体大小为单位。

设置对象中缺失的属性将使用其默认值或最后有效值。只有当新值有效时，属性才会被更改。对于compact，有效值仅为布尔值true或false，真值（truthy）不被视为有效。

返回对象

这两个函数返回一个包含信息的对象，帮助你放置下一段落。该对象
包含以下属性。

- nextLine 段落像素后的下一行位置。
- **fontSize** 字体大小。（请注意仅使用以像素定义的字体，例如 `14px arial`）
- **lineHeight** 从一行到下一行的像素距离

此示例使用一个简单的算法，每次处理一行，以找到段落的最佳适配。这并不意味着它是最佳适配（而是算法的最佳）。你可能希望通过对生成的行创建多遍行算法来改进该算法。将单词从一行末尾移动到下一行开头，或从开头移回末尾。当整个段落的间距变化最小且最接近正常文本间距时，效果最佳。

由于此示例依赖于对齐文本示例，代码非常相似。你可能希望将两者合并为一个函数。用本示例中使用的函数替换另一个示例中的justifiedTextSettings函数。然后将本示例中的其余代码复制到对齐文本示例的匿名函数体中。你将不再需要测试// Code point A处的依赖，可以将其移除。

```
        settings = {
            minSpaceSize : minSpaceSize,
            maxSpaceSize : maxSpaceSize,
        };
        return justifiedPar(this, text, x, y, width, settings,true);
    }
    CanvasRenderingContext2D.prototype.fillParaText = fillParagraphText;
    CanvasRenderingContext2D.prototype.strokeParaText = strokeParagraphText;
})();
```

> **NOTE** this extends the `CanvasRenderingContext2D` prototype. If you do not wish this to happen use the example **Justified text** to work out how to change this example to be part of the global namespace.

> **NOTE** Will throw a ReferenceError if this example can not find the function
> `CanvasRenderingContext2D.prototype.fillJustifyText`

How to use

```
ctx.fillParaText(text, x, y, width, [settings]);
ctx.strokeParaText(text, x, y, width, [settings]);
```

See **Justified text** for details on arguments. Arguments between [and] are optional.

The `settings` argument has two additional properties.

- **compact:** Default `true`. If true tries to pack as many words as possible per line. If false the tries to get the word spacing as close as possible to normal spacing.
- **lineSpacing** Default `1.5`. Space per line default `1.5` the distance from on line to the next in terms of font size

Properties missing from the settings object will default to their default values or to the last valid values. The properties will only be changed if the new values are valid. For `compact` valid values are only booleans `true` or `false` Truthy values are not considered valid.

Return object

The two functions return an object containing information to help you place the next paragraph. The object contains the following properties.

- **nextLine** Position of the next line after the paragraph pixels.
- **fontSize** Size of the font. (please note only use fonts defined in pixels eg `14px arial`)
- **lineHeight** Distance in pixels from one line to the next

This example uses a simple algorithm that works one line at to time to find the best fit for a paragraph. This does not mean that it the best fit (rather the algorithm's best) You may wish to improve the algorithm by creating a multi pass line algorithm over the generated lines. Moving words from the end of one line to the start of the next, or from the start back to the end. The best look is achieved when the spacing over the entire paragraph has the smallest variation and is the closest to the normal text spacing.

As this example is dependent on the **Justified text** example the code is very similar. You may wish to move the two into one function. Replace the function `justifiedTextSettings` in the other example with the one used in this example. Then copy all the rest of the code from this example into the anonymous function body of the **Justified text** example. You will no longer need to test for dependencies found at `// Code point A` It can be removed.

使用示例

```
ctx.font = "25px arial";
ctx.textAlign = "center"

var left = 10;
var center = canvas.width / 2;
var width = canvas.width-left*2;
var y = 20;
var size = 16;
var i = 0;
ctx.fillText("对齐段落示例。",center,y);
y+= 30;
ctx.font = "14px arial";
ctx.textAlign = "left"
// 设置参数配置
var setting = {
    maxSpaceSize : 6,
    minSpaceSize : 0.5,
    lineSpacing : 1.2,
    compact : true,
}
// 显示左右边界。
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(left,y - size * 2);
ctx.lineTo(left, y + size * 15);
ctx.moveTo(canvas.width - left,y - size * 2);
ctx.lineTo(canvas.width - left, y + size * 15);
ctx.stroke();
ctx.textAlign = "left";
ctx.fillStyle = "black";

// 绘制段落
var line = ctx.fillParaText(para, left, y, width, setting);  // 记住设置

// 下一个段落
y = line.nextLine + line.lineHeight;
setting.compact = false;
ctx.fillParaText(para, left, y, width, setting);
```

> **注意： 对于左对齐（left）或起始对齐（start）的文本，段落的最后一行总是使用正常间距。对于所有**其他对齐方式，最后一行的处理与其他行相同。

> 注意： 你可以用空格缩进段落的开头。虽然这可能导致不同段落之间不一致。了解函数的工作原理并进行修改总是有益的。一个练习是向设置中添加一个选项，使首行缩进固定量。提示：while 循环需要临时使第一个单词看起来更宽（+ 缩进）words[0].width += ？
>
> 然后在渲染行时缩进首行。

第2.3节：沿弧线渲染文本

本示例展示了如何沿弧线渲染文本。内容包括如何通过扩展CanvasRenderingContext2D的原型来添加功能。

本示例源自stackoverflow上的回答Circular Text。

Usage example

```
ctx.font = "25px arial";
ctx.textAlign = "center"

var left = 10;
var center = canvas.width / 2;
var width = canvas.width-left*2;
var y = 20;
var size = 16;
var i = 0;
ctx.fillText("Justified paragraph examples.",center,y);
y+= 30;
ctx.font = "14px arial";
ctx.textAlign = "left"
// set para settings
var setting = {
    maxSpaceSize : 6,
    minSpaceSize : 0.5,
    lineSpacing : 1.2,
    compact : true,
}
// Show the left and right bounds.
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(left,y - size * 2);
ctx.lineTo(left, y + size * 15);
ctx.moveTo(canvas.width - left,y - size * 2);
ctx.lineTo(canvas.width - left, y + size * 15);
ctx.stroke();
ctx.textAlign = "left";
ctx.fillStyle = "black";

// Draw paragraph
var line = ctx.fillParaText(para, left, y, width, setting);  // settings is remembered

// Next paragraph
y = line.nextLine + line.lineHeight;
setting.compact = false;
ctx.fillParaText(para, left, y, width, setting);
```

> **Note:** For text aligned `left` or `start` the last line of tha paragraph will always have normal spacing. For all other alignments the last line is treated like all others.

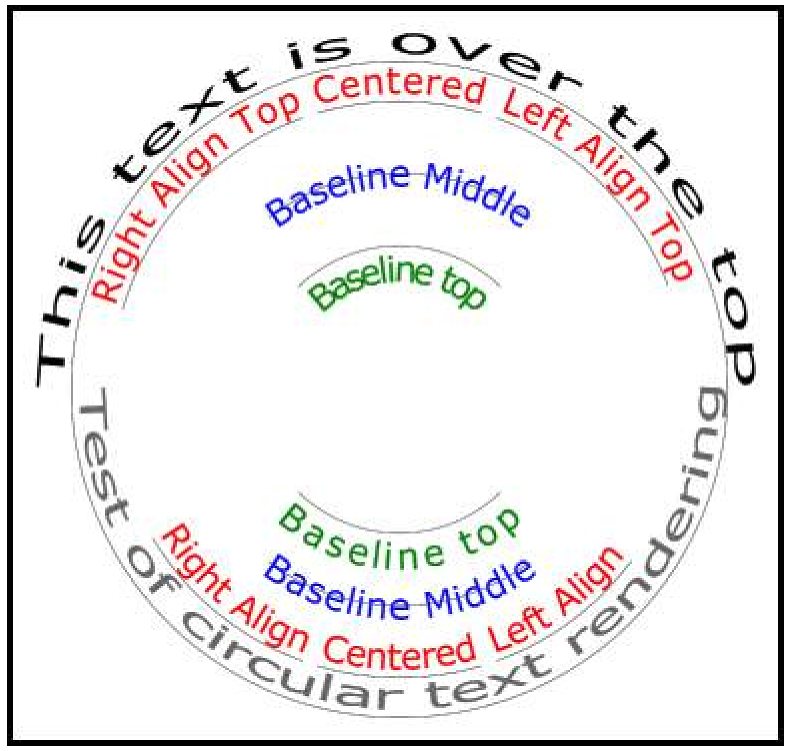
> **Note:** You can inset the start of the paragraph with spaces. Though this may not be consistent from paragraph to paragraph. It is always a good thing to learn what a function is doing and modifying it. An exercise would be to add a setting to the settings that indents the first line by a fixed amount. Hint the while loop will need to temporarily make the first word appear larger (+ indent) `words[0].width += ?` and then when rendering lines indent the first line.

Section 2.3: Rendering text along an arc

This example shows how to render text along an arc. It includes how you can add functionality to the `CanvasRenderingContext2D` by extending its prototype.

This examples is derived from the stackoverflow answer Circular Text.

示例渲染


This text is over the top
Right Align Top Centered Left Align Top
Baseline Middle
Baseline top
Test of circular text rendering
Right Align Centered Left Align
Baseline Middle
Baseline top

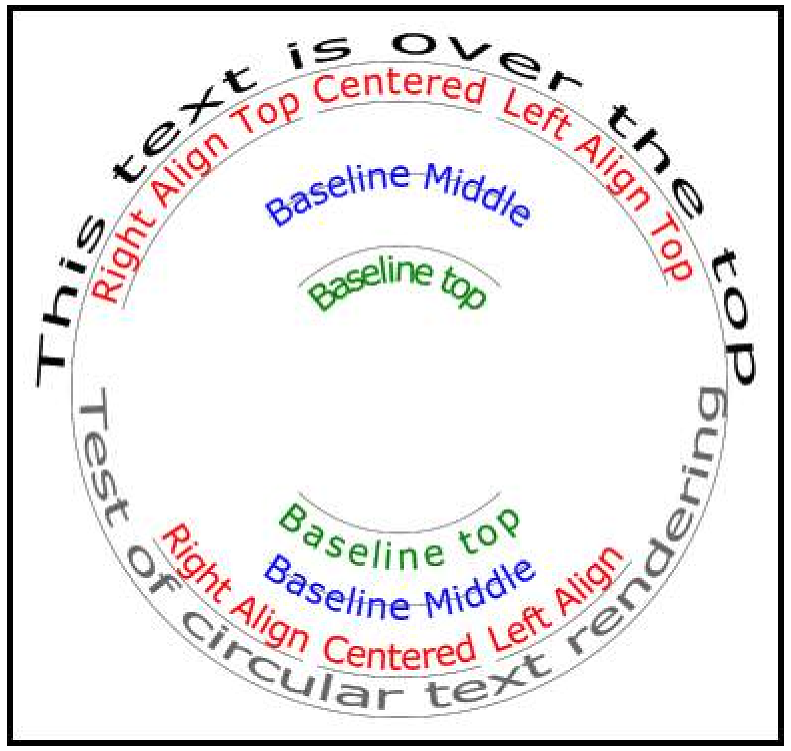
示例代码

该示例向2D上下文原型添加了3个新的文本渲染函数。

- **ctx.fillCircleText(text, x, y, radius, start, end, forward);**
- **ctx.strokeCircleText(text, x, y, radius, start, end, forward);**
- **ctx.measureCircleText(text, radius);**

```
(function(){
    const FILL = 0;        // 用于指示填充文本渲染的常量
    const STROKE = 1;
    var renderType = FILL; // 内部用于设置填充或描边文本const multiplyCurrentTran
    sform = true; // 如果为true，渲染时使用当前变换// 如果为false，使用绝对坐标，这样会稍快一些

                                                    // 渲染后，currentTransform会恢复为默认变换




    // 测量圆形文本
    // ctx：画布上下文
    // text：要测量的文本字符串
    // r：半径（像素）
    //
    // 返回文本的尺寸指标
    //
    // 宽度：文本的像素宽度
    // angularWidth：文本的弧度角宽度
    // pixelAngularSize：像素的弧度角大小
    var measure = function(ctx, text, radius){
        var textWidth = ctx.measureText(text).width; // 获取文本的总宽度
        return {
width                   : textWidth,
```

Example rendering


This text is over the top
Right Align Top Centered Left Align Top
Baseline Middle
Baseline top
Test of circular text rendering
Right Align Centered Left Align
Baseline Middle
Baseline top

Example code

The example adds 3 new text rendering functions to the 2D context prototype.

- **ctx.fillCircleText(text, x, y, radius, start, end, forward);**
- **ctx.strokeCircleText(text, x, y, radius, start, end, forward);**
- **ctx.measureCircleText(text, radius);**

```
(function(){
    const FILL = 0;        // const to indicate filltext render
    const STROKE = 1;
    var renderType = FILL; // used internal to set fill or stroke text
    const multiplyCurrentTransform = true; // if true Use current transform when rendering
                                           // if false use absolute coordinates which is a little
quicker
                                           // after render the currentTransform is restored to
default transform



    // measure circle text
    // ctx: canvas context
    // text: string of text to measure
    // r: radius in pixels
    //
    // returns the size metrics of the text
    //
    // width: Pixel width of text
    // angularWidth : angular width of text in radians
    // pixelAngularSize : angular width of a pixel in radians
    var measure = function(ctx, text, radius){
        var textWidth = ctx.measureText(text).width; // get the width of all the text
        return {
            width                  : textWidth,
angularWidth        : (1 / radius) * textWidth,
            pixelAngularSize     : 1 / radius
        };
    }

    // 沿圆弧显示文本
    // ctx：画布上下文
    // text：要测量的文本字符串
    // x,y：圆心位置
    // r：圆的像素半径
    // start：起始角度（弧度）
    // [end]：可选。如果包含，文本对齐将被忽略，文本将被缩放以适应 start 和 end 之间
    ；
    // [forward]：可选，默认值为 true。如果为 true，文本方向为正向；如果为 false，方向为反向

    var circleText = function (ctx, text, x, y, radius, start, end, forward) {
        var i, textWidth, pA, pAS, a, aw, wScale, aligned, dir, fontSize;
        if(text.trim() === "" || ctx.globalAlpha === 0){ // 不渲染空字符串或
透明
            return;
        }
        if(isNaN(x) || isNaN(y) || isNaN(radius) || isNaN(start) || (end !== undefined && end !==
null && isNaN(end))){ //
            throw TypeError("circle text arguments requires a number for x,y, radius, start, and
end.")
        }
aligned = ctx.textAlign;        // 保存当前的 textAlign，以便在结束时恢复

dir = forward ? 1 : forward === false ? -1 : 1;  // 设置方向，如果不是 true 或 false，则将 forward 视为 true

pAS = 1 / radius;               // 获取像素的角度大小（弧度）        textWidth = ctx.measureText(tex
t).width; // 获取文本的宽度if (end !== undefined && end !== null) { // 如果提供了 end，则将文本适
        配在 start 和 end 之间

pA = ((end - start) / textWidth) * dir;
            wScale = (pA / pAS) * dir;
        } 否则 {                    // 如果未提供结束位置，则校正对齐的起始和结束位置
            // 如果未给出 forward，则交换圆形文本的顶部以读取正确方向
            if(forward === null || forward === undefined){
                if(((start % (Math.PI * 2)) + Math.PI * 2) % (Math.PI * 2) > Math.PI){
                    dir = -1;
                }
            }
pA = -pAS * dir ;
            wScale = -1 * dir;
            switch (aligned) {
            case "center":       // 如果居中，则围绕宽度的一半移动
              start -= (pA * textWidth )/2;
              end = start + pA * textWidth;
                 break;
            case "right":// 有意落入 case "end"
            case "end":
end = start;
                start -= pA * textWidth;
                break;
            case "left":  // 有意落入 case "start"
            case "start":
end = start + pA * textWidth;
            }
        }

ctx.textAlign = "center";                    // 渲染时的对齐方式
```

```
            angularWidth        : (1 / radius) * textWidth,
            pixelAngularSize    : 1 / radius
        };
    }

    // displays text along a circle
    // ctx: canvas context
    // text: string of text to measure
    // x,y: position of circle center
    // r: radius of circle in pixels
    // start: angle in radians to start.
    // [end]: optional. If included text align is ignored and the text is
    //        scaled to fit between start and end;
    // [forward]: optional default true. if true text direction is forwards, if false  direction is
backward
    var circleText = function (ctx, text, x, y, radius, start, end, forward) {
        var i, textWidth, pA, pAS, a, aw, wScale, aligned, dir, fontSize;
        if(text.trim() === "" || ctx.globalAlpha === 0){ // don't render empty string or
transparent
            return;
        }
        if(isNaN(x) || isNaN(y) || isNaN(radius) || isNaN(start) || (end !== undefined && end !==
null && isNaN(end))){ //
            throw TypeError("circle text arguments requires a number for x,y, radius, start, and
end.")
        }
        aligned = ctx.textAlign;        // save the current textAlign so that it can be restored at
end
        dir = forward ? 1 : forward === false ? -1 : 1;  // set dir if not true or false set
forward as true
        pAS = 1 / radius;               // get the angular size of a pixel in radians
        textWidth = ctx.measureText(text).width; // get the width of all the text
        if (end !== undefined && end !== null) { // if end is supplied then fit text between start
and end
            pA = ((end - start) / textWidth) * dir;
            wScale = (pA / pAS) * dir;
        } else {                 // if no end is supplied correct start and end for alignment
            // if forward is not given then swap top of circle text to read the correct direction
            if(forward === null || forward === undefined){
                if(((start % (Math.PI * 2)) + Math.PI * 2) % (Math.PI * 2) > Math.PI){
                    dir = -1;
                }
            }
            pA = -pAS * dir ;
            wScale = -1 * dir;
            switch (aligned) {
            case "center":       // if centered move around half width
                start -= (pA * textWidth )/2;
                end = start + pA * textWidth;
                break;
            case "right":// intentionally falls through to case "end"
            case "end":
                end = start;
                start -= pA * textWidth;
                break;
            case "left":  // intentionally falls through to case "start"
            case "start":
                end = start + pA * textWidth;
            }
        }

        ctx.textAlign = "center";                     // align for rendering
        a = start;                                    // 设置起始角度
        for (var i = 0; i < text.length; i += 1) {    // 遍历每个字符
            aw = ctx.measureText(text[i]).width * pA; // 获取文本的角宽度
            var xDx = Math.cos(a + aw / 2);           // 从中心点x,y获取y轴向量
out
            var xDy = Math.sin(a + aw / 2);
            if(multiplyCurrentTransform){ // 变换乘以当前变换
              ctx.save();
                if (xDy < 0) { // 文本是否颠倒。如果是，则翻转它
                 ctx.transform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius + x, xDy *
radius + y);
                } 否则 {
                    ctx.transform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x, xDy *
radius + y);
                }
            } else{
                if (xDy < 0) { // 文本是否颠倒。如果是，则翻转它
                 ctx.setTransform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius + x, xDy
* radius + y);
                } 否则 {
                    ctx.setTransform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x, xDy *
radius + y);
                }
            }
            if(renderType === FILL){
ctx.fillText(text[i], 0, 0);    // 渲染字符
            }else{
ctx.strokeText(text[i], 0, 0);  // 渲染字符
            }
            if(multiplyCurrentTransform){   // 恢复当前变换
              ctx.restore();
            }
a += aw;                      // 进入下一个角度
        }
        // 全部完成，清理。
        if(!multiplyCurrentTransform){
ctx.setTransform(1, 0, 0, 1, 0, 0); // 恢复变换
        }
ctx.textAlign = aligned;            // 恢复文本对齐方式
    }
    // 定义填充文本
    var fillCircleText = function(text, x, y, radius, start, end, forward){
        renderType = FILL;
circleText(this, text, x, y, radius, start, end, forward);
    }
    // define stroke text
    var strokeCircleText = function(text, x, y, radius, start, end, forward){
        renderType = STROKE;
circleText(this, text, x, y, radius, start, end, forward);
    }
    // define measure text
    var measureCircleTextExt = function(text,radius){
        return measure(this, text, radius);
    }
    // 设置原型
CanvasRenderingContext2D.prototype.fillCircleText = fillCircleText;
    CanvasRenderingContext2D.prototype.strokeCircleText = strokeCircleText;
    CanvasRenderingContext2D.prototype.measureCircleText = measureCircleTextExt;
})();
```

函数说明

```
        a = start;                                    // set the start angle
        for (var i = 0; i < text.length; i += 1) {    // for each character
            aw = ctx.measureText(text[i]).width * pA; // get the angular width of the text
            var xDx = Math.cos(a + aw / 2);           // get the yAxies vector from the center x,y
out
            var xDy = Math.sin(a + aw / 2);
            if(multiplyCurrentTransform){ // transform multiplying current transform
                ctx.save();
                if (xDy < 0) { // is the text upside down. If it is flip it
                    ctx.transform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius + x, xDy *
radius + y);
                } else {
                    ctx.transform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x, xDy *
radius + y);
                }
            }else{
                if (xDy < 0) { // is the text upside down. If it is flip it
                    ctx.setTransform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius + x, xDy
* radius + y);
                } else {
                    ctx.setTransform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x, xDy *
radius + y);
                }
            }
            if(renderType === FILL){
                ctx.fillText(text[i], 0, 0);    // render the character
            }else{
                ctx.strokeText(text[i], 0, 0);  // render the character
            }
            if(multiplyCurrentTransform){  // restore current transform
                ctx.restore();
            }
            a += aw;                     // step to the next angle
        }
        // all done clean up.
        if(!multiplyCurrentTransform){
            ctx.setTransform(1, 0, 0, 1, 0, 0); // restore the transform
        }
        ctx.textAlign = aligned;            // restore the text alignment
    }
    // define fill text
    var fillCircleText = function(text, x, y, radius, start, end, forward){
        renderType = FILL;
        circleText(this, text, x, y, radius, start, end, forward);
    }
    // define stroke text
    var strokeCircleText = function(text, x, y, radius, start, end, forward){
        renderType = STROKE;
        circleText(this, text, x, y, radius, start, end, forward);
    }
    // define measure text
    var measureCircleTextExt = function(text,radius){
        return measure(this, text, radius);
    }
    // set the prototypes
    CanvasRenderingContext2D.prototype.fillCircleText = fillCircleText;
    CanvasRenderingContext2D.prototype.strokeCircleText = strokeCircleText;
    CanvasRenderingContext2D.prototype.measureCircleText = measureCircleTextExt;
})();
```

Function descriptions

此示例向CanvasRenderingContext2D的原型添加了3个函数：fillCircleText、strokeCircleText和measureCircleText

CanvasRenderingContext2D.fillCircleText(text, x, y, radius, start, [end, [forward]]);

CanvasRenderingContext2D.strokeCircleText(text, x, y, radius, start, [end, [forward]]);

- ***text: 要渲染的文本，字符串类型。***
- x,y: 圆心位置，数字类型。
- **radius:** 圆的半径，单位为像素
- **start:** 起始角度，弧度制。
- [end]: 可选。如果包含，ctx.textAlign将被忽略，文本会被缩放以适应起始角度和结束角度之间的区域。
- [forward]: 可选，默认值为 'true'。如果为 true，文本方向为正向；如果为 'false'，方向为反向。

这两个函数使用 textBaseline 来垂直定位文本，使其围绕半径排列。为了获得最佳效果，请使用 ctx.TextBaseline。

如果任何数值参数为 NaN，函数将抛出TypeError。

如果text参数修剪后为空字符串，或ctx.globalAlpha = 0，函数将直接跳过，不执行任何操作。

CanvasRenderingContext2D.measureCircleText(text, radius);

```
 - **text:** 要测量的文本字符串。
 - **radius:** 圆的半径，单位为像素。
```

返回一个包含用于渲染圆形文本的各种尺寸指标的对象

```
   - **width:** 文本正常渲染时的像素宽度
   - **angularWidth:** 文本的角宽度，单位为弧度。
   - **pixelAngularSize:** 像素的角宽度，单位为弧度。
```

使用示例

```
const rad = canvas.height * 0.4;
const text = "Hello circle TEXT!";
const fontSize = 40;
const centX = canvas.width / 2;
const centY = canvas.height / 2;
ctx.clearRect(0,0,canvas.width,canvas.height)

ctx.font = fontSize + "px verdana";
ctx.textAlign = "center";
ctx.textBaseline = "bottom";
ctx.fillStyle = "#000";
ctx.strokeStyle = "#666";

// 文本从 Math.PI 到 0 (180 - 0 度) 下方拉伸
ctx.fillCircleText(text, centX, centY, rad, Math.PI, 0);

// 文本在 Math.PI * 1.5 (270 度) 顶部居中显示
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);

// 文本在 Math.PI * 1.5 (270 度) 顶部下方显示
ctx.textBaseline = "top";
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);
```

This example adds 3 functions to the CanvasRenderingContext2D **prototype**. fillCircleText, strokeCircleText, and measureCircleText

CanvasRenderingContext2D.fillCircleText(text, x, y, radius, start, [end, [forward]]);

CanvasRenderingContext2D.strokeCircleText(text, x, y, radius, start, [end, [forward]]);

- **text:** Text to render as String.
- **x,y**: Position of circle center as Numbers.
- **radius:** radius of circle in pixels
- **start:** angle in radians to start.
- **[end]:** optional. If included ctx.textAlign is ignored and the text is scaled to fit between start and end.
- **[forward]:** optional default 'true'. if true text direction is forwards, if 'false' direction is backward.

Both functions use the textBaseline to position the text vertically around the radius. For the best results use ctx.TextBaseline.

Functions will throw a TypeError is any of the numerical arguments as NaN.

If the text argument trims to an empty string or ctx.globalAlpha = 0 the function just drops through and does nothing.

CanvasRenderingContext2D.measureCircleText(text, radius);

```
 - **text:** String of text to measure.
 - **radius:** radius of circle in pixels.
```

Returns a Object containing various size metrics for rendering circular text

```
   - **width:** Pixel width of text as it would normaly be rendered
   - **angularWidth:** angular width of text in radians.
   - **pixelAngularSize:** angular width of a pixel in radians.
```

Usage examples

```
const rad = canvas.height * 0.4;
const text = "Hello circle TEXT!";
const fontSize = 40;
const centX = canvas.width / 2;
const centY = canvas.height / 2;
ctx.clearRect(0,0,canvas.width,canvas.height)

ctx.font = fontSize + "px verdana";
ctx.textAlign = "center";
ctx.textBaseline = "bottom";
ctx.fillStyle = "#000";
ctx.strokeStyle = "#666";

// Text under stretched from Math.PI to 0 (180 - 0 deg)
ctx.fillCircleText(text, centX, centY, rad, Math.PI, 0);

// text over top centered at Math.PI * 1.5 ( 270 deg)
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);

// text under top centered at Math.PI * 1.5 ( 270 deg)
ctx.textBaseline = "top";
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);
```

```
// 文本在 Math.PI * 1.5（270 度）顶部居中显示
ctx.textBaseline = "middle";
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);


// 使用 measureCircleText 获取角度大小
var circleTextMetric = ctx.measureCircleText("Text to measure", rad);
console.log(circleTextMetric.width);            // 如果正常渲染文本的宽度
console.log(circleTextMetric.angularWidth);     // 文本的角度宽度
console.log(circleTextMetric.pixelAngularSize); // 像素的角度大小



// 使用 measureText 绘制文本周围的弧线
ctx.textBaseline = "middle";
var width = ctx.measureCircleText(text, rad).angularWidth;
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);

// 绘制文本周围的弧线
ctx.strokeStyle= "red";
ctx.lineWidth = 3;
ctx.beginPath();
ctx.arc(centX, centY, rad + fontSize / 2,Math.PI * 1.5 - width/2,Math.PI*1.5 + width/2);
ctx.arc(centX, centY, rad - fontSize / 2,Math.PI * 1.5 + width/2,Math.PI*1.5 - width/2,true);
ctx.closePath();
ctx.stroke();
```

注意：渲染的文本只是圆形文本的近似。例如，如果渲染两个字母"l"，这两条线将不会平行，但如果渲染一个"H"，两边缘将是平行的。这是因为每个字符尽可能接近所需方向渲染，而不是每个像素都被正确变换以创建圆形文本。

注意：本示例中定义的 const multiplyCurrentTransform = true; 用于设置所使用的变换方法。如果为 false，圆形文本渲染的变换是绝对的，不依赖于当前的变换状态。文本不会受到之前的缩放、旋转或平移变换的影响。这将提高渲染函数的性能，函数调用后变换将被设置为默认的 setTransform(1,0,0,1,0,0)

如果 multiplyCurrentTransform = true（本示例中默认设置），文本将使用当前变换，这样文本可以被缩放、平移、倾斜、旋转等，但需要在调用 fillCircleText 和 strokeCircleText 函数之前修改当前变换。根据2D上下文的当前状态，这可能比 multiplyCurrentTransform = false 稍慢一些。

第2.4节：曲线上的文本，三次和二次贝塞尔曲线

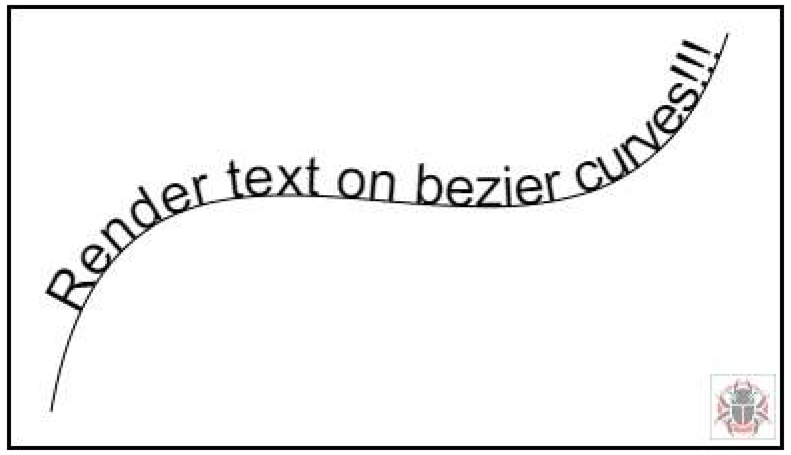
```
// text over top centered at Math.PI * 1.5 ( 270 deg)
ctx.textBaseline = "middle";
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);


// Use measureCircleText to get angular size
var circleTextMetric = ctx.measureCircleText("Text to measure", rad);
console.log(circleTextMetric.width);            // width of text if rendered normally
console.log(circleTextMetric.angularWidth);     // angular width of text
console.log(circleTextMetric.pixelAngularSize); // angular size of a pixel



// Use measure text to draw a arc around the text
ctx.textBaseline = "middle";
var width = ctx.measureCircleText(text, rad).angularWidth;
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);

// render the arc around the text
ctx.strokeStyle= "red";
ctx.lineWidth = 3;
ctx.beginPath();
ctx.arc(centX, centY, rad + fontSize / 2,Math.PI * 1.5 - width/2,Math.PI*1.5 + width/2);
ctx.arc(centX, centY, rad - fontSize / 2,Math.PI * 1.5 + width/2,Math.PI*1.5 - width/2,true);
ctx.closePath();
ctx.stroke();
```

NOTE: The text rendered is only an approximation of circular text. For example if two l's are rendered the two lines will not be parallel, but if you render a "H" the two edges will be parallel. This is because each character is rendered as close as possible to the required direction, rather than each pixel being correctly transformed to create circular text.

NOTE: `const multiplyCurrentTransform = true;` defined in this example is used to set the transformation method used. If `false` the transformation for circular text rendering is absolute and does not depend on the current transformation state. The text will not be effected by any previous scale, rotate, or translate transforms. This will increase the performance of the render function, after the function is called the transform will be set to the default `setTransform(1,0,0,1,0,0)`

If `multiplyCurrentTransform = true` (set as default in this example) the text will use the current transform so that the text can be scaled translated, skewed, rotated, etc but modifying the current transform befor calling the `fillCircleText` and `strokeCircleText` functions. Depending on the current state of the 2D context this may be somewhat slower then `multiplyCurrentTransform = false`

Section 2.4: Text on curve, cubic and quadratic beziers



textOnCurve(text,offset,x1,y1,x2,y2,x3,y3,x4,y4)

在二次和三次曲线上渲染文本。

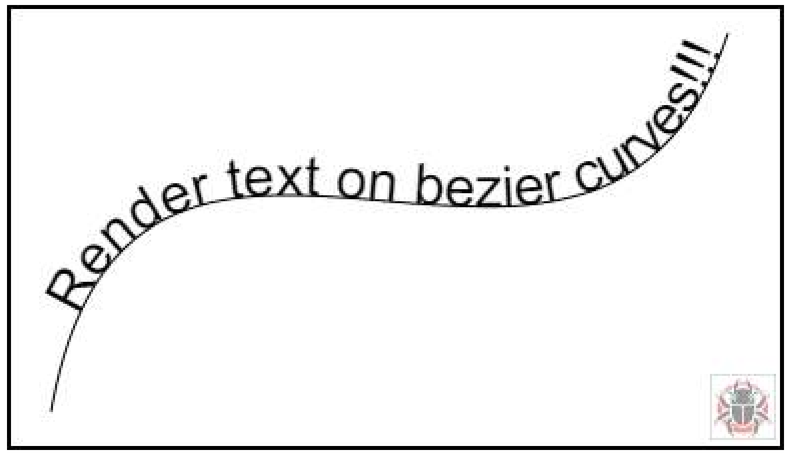
- `text` 是要渲染的文本
- `offset` 从曲线起点到文本的距离 >= 0
- `x1,y1` - `x3,y3` 二次曲线的点，或
- `x1,y1` - `x4,y4` 三次曲线的点，或

示例用法：

```
textOnCurve("Hello world!",50,100,100,200,200,300,100); // 在二次曲线上绘制文本
                                                         // 从曲线起点偏移50像素


textOnCurve("Hello world!",50,100,100,200,200,300,100,400,200);
                                                         // 在三次曲线上绘制文本
                                                         // 从曲线起点偏移50像素
```

函数及曲线辅助函数

```
// 传入8个值用于三次贝塞尔曲线
// 传入6个值用于二次贝塞尔曲线
// 从曲线起点渲染文本
var textOnCurve = function(text,offset,x1,y1,x2,y2,x3,y3,x4,y4){
    ctx.save();
ctx.textAlign = "center";
    var widths = [];
    for(var i = 0; i < text.length; i ++){
        widths[widths.length] = ctx.measureText(text[i]).width;
    }
    var ch = curveHelper(x1,y1,x2,y2,x3,y3,x4,y4);
    var pos = offset;
    var cpos = 0;

    for(var i = 0; i < text.length; i ++){
        pos += widths[i] / 2;
        cpos = ch.forward(pos);
        ch.tangent(cpos);
ctx.setTransform(ch.vect.x, ch.vect.y, -ch.vect.y, ch.vect.x, ch.vec.x, ch.vec.y);
        ctx.fillText(text[i],0,0);

pos += widths[i] / 2;
    }
ctx.restore();
}
```



textOnCurve(text,offset,x1,y1,x2,y2,x3,y3,x4,y4)

Renders text on quadratic and cubic curves.

- `text` is the text to render
- `offset` distance from start of curve to text >= 0
- `x1,y1` - `x3,y3` points of quadratic curve or
- `x1,y1` - `x4,y4` points of cubic curve or

Example usage:

```
textOnCurve("Hello world!",50,100,100,200,200,300,100); // draws text on quadratic curve
                                                         // 50 pixels from start of curve


textOnCurve("Hello world!",50,100,100,200,200,300,100,400,200);
                                                         // draws text on cubic curve
                                                         // 50 pixels from start of curve
```

The Function and curver helper function

```
// pass 8 values for cubic bezier
// pass 6 values for quadratic
// Renders text from start of curve
var textOnCurve = function(text,offset,x1,y1,x2,y2,x3,y3,x4,y4){
    ctx.save();
    ctx.textAlign = "center";
    var widths = [];
    for(var i = 0; i < text.length; i ++){
        widths[widths.length] = ctx.measureText(text[i]).width;
    }
    var ch = curveHelper(x1,y1,x2,y2,x3,y3,x4,y4);
    var pos = offset;
    var cpos = 0;

    for(var i = 0; i < text.length; i ++){
        pos += widths[i] / 2;
        cpos = ch.forward(pos);
        ch.tangent(cpos);
        ctx.setTransform(ch.vect.x, ch.vect.y, -ch.vect.y, ch.vect.x, ch.vec.x, ch.vec.y);
        ctx.fillText(text[i],0,0);

        pos += widths[i] / 2;
    }
    ctx.restore();
}
```

曲线辅助函数旨在提高在贝塞尔曲线上查找点的性能。

```
// 辅助函数用于定位贝塞尔曲线上的点。
function curveHelper(x1, y1, x2, y2, x3, y3, x4, y4){
    var tx1, ty1, tx2, ty2, tx3, ty3, tx4, ty4;
    var a,b,c,u;
    var vec,currentPos,vec1,vect;
    vec = {x:0,y:0};
    vec1 = {x:0,y:0};
    vect = {x:0,y:0};
    quad = false;
    currentPos = 0;
    currentDist = 0;
    if(x4 === undefined || x4 === null){
        quad = true;
        x4 = x3;
y4 = y3;
    }
    var estLen = Math.sqrt((x4 - x1) * (x4 - x1) + (y4 - y1) * (y4 - y1));
    var onePix = 1 / estLen;
    function posAtC(c){
tx1 = x1; ty1 = y1;
        tx2 = x2; ty2 = y2;
        tx3 = x3; ty3 = y3;
        tx1 += (tx2 - tx1) * c;
        ty1 += (ty2 - ty1) * c;
        tx2 += (tx3 - tx2) * c;
        ty2 += (ty3 - ty2) * c;
        tx3 += (x4 - tx3) * c;
        ty3 += (y4 - ty3) * c;
        tx1 += (tx2 - tx1) * c;
        ty1 += (ty2 - ty1) * c;
        tx2 += (tx3 - tx2) * c;
        ty2 += (ty3 - ty2) * c;
        vec.x = tx1 + (tx2 - tx1) * c;
        vec.y = ty1 + (ty2 - ty1) * c;
        return vec;
    }
    function posAtQ(c){
tx1 = x1; ty1 = y1;
        tx2 = x2; ty2 = y2;
        tx1 += (tx2 - tx1) * c;
        ty1 += (ty2 - ty1) * c;
        tx2 += (x3 - tx2) * c;
        ty2 += (y3 - ty2) * c;
        vec.x = tx1 + (tx2 - tx1) * c;
        vec.y = ty1 + (ty2 - ty1) * c;
        return vec;
    }
    function forward(dist){
        var step;
helper.posAt(currentPos);

        while(currentDist < dist){
            vec1.x = vec.x;
vec1.y = vec.y;
            currentPos += onePix;
helper.posAt(currentPos);
currentDist += step = Math.sqrt((vec.x - vec1.x) * (vec.x - vec1.x) + (vec.y - vec1.y)
* (vec.y - vec1.y));

        }
```

The curve helper function is designed to increase the performance of finding points on the bezier.

```
// helper function locates points on bezier curves.
function curveHelper(x1, y1, x2, y2, x3, y3, x4, y4){
    var tx1, ty1, tx2, ty2, tx3, ty3, tx4, ty4;
    var a,b,c,u;
    var vec,currentPos,vec1,vect;
    vec = {x:0,y:0};
    vec1 = {x:0,y:0};
    vect = {x:0,y:0};
    quad = false;
    currentPos = 0;
    currentDist = 0;
    if(x4 === undefined || x4 === null){
        quad = true;
        x4 = x3;
        y4 = y3;
    }
    var estLen = Math.sqrt((x4 - x1) * (x4 - x1) + (y4 - y1) * (y4 - y1));
    var onePix = 1 / estLen;
    function posAtC(c){
        tx1 = x1; ty1 = y1;
        tx2 = x2; ty2 = y2;
        tx3 = x3; ty3 = y3;
        tx1 += (tx2 - tx1) * c;
        ty1 += (ty2 - ty1) * c;
        tx2 += (tx3 - tx2) * c;
        ty2 += (ty3 - ty2) * c;
        tx3 += (x4 - tx3) * c;
        ty3 += (y4 - ty3) * c;
        tx1 += (tx2 - tx1) * c;
        ty1 += (ty2 - ty1) * c;
        tx2 += (tx3 - tx2) * c;
        ty2 += (ty3 - ty2) * c;
        vec.x = tx1 + (tx2 - tx1) * c;
        vec.y = ty1 + (ty2 - ty1) * c;
        return vec;
    }
    function posAtQ(c){
        tx1 = x1; ty1 = y1;
        tx2 = x2; ty2 = y2;
        tx1 += (tx2 - tx1) * c;
        ty1 += (ty2 - ty1) * c;
        tx2 += (x3 - tx2) * c;
        ty2 += (y3 - ty2) * c;
        vec.x = tx1 + (tx2 - tx1) * c;
        vec.y = ty1 + (ty2 - ty1) * c;
        return vec;
    }
    function forward(dist){
        var step;
        helper.posAt(currentPos);

        while(currentDist < dist){
            vec1.x = vec.x;
            vec1.y = vec.y;
            currentPos += onePix;
            helper.posAt(currentPos);
            currentDist += step = Math.sqrt((vec.x - vec1.x) * (vec.x - vec1.x) + (vec.y - vec1.y)
* (vec.y - vec1.y));

        }
currentPos -= ((currentDist - dist) / step) * onePix
        currentDist -= step;
helper.posAt(currentPos);
currentDist += Math.sqrt((vec.x - vec1.x) * (vec.x - vec1.x) + (vec.y - vec1.y) * (vec.y -
vec1.y));
        return currentPos;
    }

    function tangentQ(pos){
a = (1-pos) * 2;
b = pos * 2;
vect.x = a * (x2 - x1) + b * (x3 - x2);
        vect.y = a * (y2 - y1) + b * (y3 - y2);
        u = Math.sqrt(vect.x * vect.x + vect.y * vect.y);
        vect.x /= u;
vect.y /= u;
    }
    function tangentC(pos){
        a  = (1-pos)
        b  = 6 * a * pos;
        a *= 3 * a;
        c  = 3 * pos * pos;
vect.x  = -x1 * a + x2 * (a - b) + x3 * (b - c) + x4 * c;
        vect.y  = -y1 * a + y2 * (a - b) + y3 * (b - c) + y4 * c;
        u = Math.sqrt(vect.x * vect.x + vect.y * vect.y);
vect.x /= u;
        vect.y /= u;
    }
    var helper = {
        vec : vec,
        vect : vect,
        forward : forward,
    }
    if(quad){
helper.posAt = posAtQ;
        helper.tangent = tangentQ;
    }else{
helper.posAt = posAtC;
        helper.tangent = tangentC;
    }
    return helper
}
```

第2.5节：绘制文本

绘制到画布不仅限于形状和图像。你也可以在画布上绘制文本。

要在画布上绘制文本，先获取画布的引用，然后在上下文上调用fillText方法。

```
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');
ctx.fillText("My text", 0, 0);
```

传入fillText的三个必需参数是：

1.你想显示的文本
2.水平（x轴）位置
3.垂直（y轴）位置

此外，还有第四个可选参数，可以用来指定文本的最大宽度

```
        currentPos -= ((currentDist - dist) / step) * onePix
        currentDist -= step;
        helper.posAt(currentPos);
        currentDist += Math.sqrt((vec.x - vec1.x) * (vec.x - vec1.x) + (vec.y - vec1.y) * (vec.y -
vec1.y));
        return currentPos;
    }

    function tangentQ(pos){
        a = (1-pos) * 2;
        b = pos * 2;
        vect.x = a * (x2 - x1) + b * (x3 - x2);
        vect.y = a * (y2 - y1) + b * (y3 - y2);
        u = Math.sqrt(vect.x * vect.x + vect.y * vect.y);
        vect.x /= u;
        vect.y /= u;
    }
    function tangentC(pos){
        a  = (1-pos)
        b  = 6 * a * pos;
        a *= 3 * a;
        c  = 3 * pos * pos;
        vect.x  = -x1 * a + x2 * (a - b) + x3 * (b - c) + x4 * c;
        vect.y  = -y1 * a + y2 * (a - b) + y3 * (b - c) + y4 * c;
        u = Math.sqrt(vect.x * vect.x + vect.y * vect.y);
        vect.x /= u;
        vect.y /= u;
    }
    var helper = {
        vec : vec,
        vect : vect,
        forward : forward,
    }
    if(quad){
        helper.posAt = posAtQ;
        helper.tangent = tangentQ;
    }else{
        helper.posAt = posAtC;
        helper.tangent = tangentC;
    }
    return helper
}
```

Section 2.5: Drawing Text

Drawing to canvas isn't just limited to shapes and images. You can also draw text to the canvas.

To draw text on the canvas, get a reference to the canvas and then call the `fillText` method on the context.

```
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');
ctx.fillText("My text", 0, 0);
```

The three **required** arguments that are passed into `fillText` are:

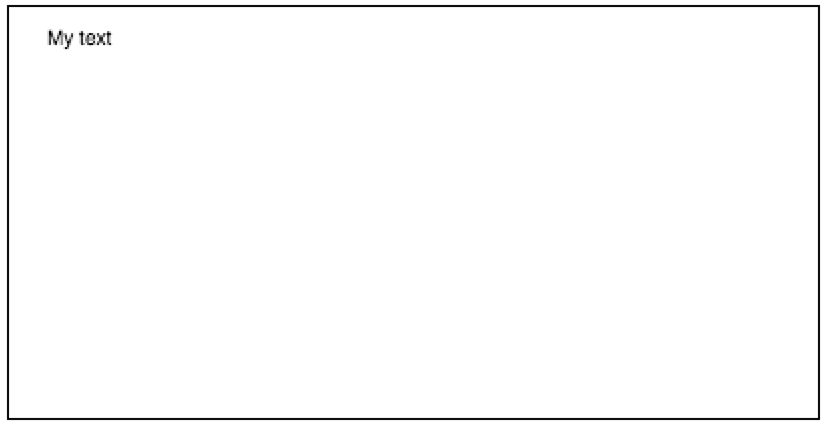
1. The text that you would like to display
2. The horizontal (x-axis) position
3. The vertical (y-axis) position

Additionally, there is a fourth **optional** argument, which you can use to specify the maximum width of your text in

像素。在下面的示例中，值200限制文本的最大宽度为200像素：

```
ctx.fillText("My text", 0, 0, 200);
```

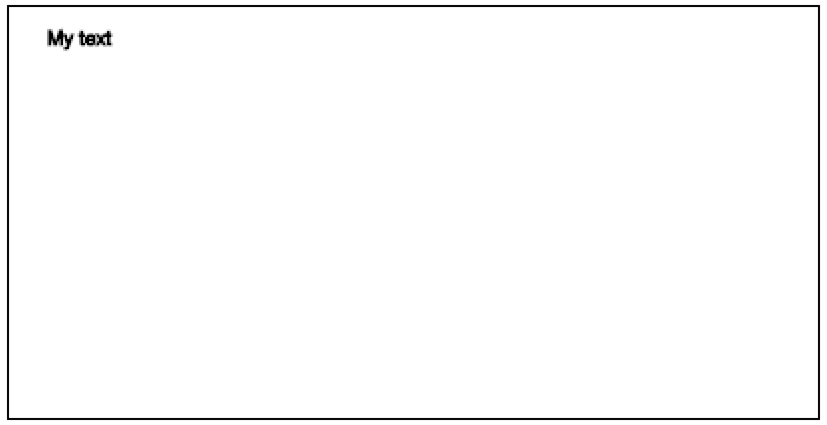
结果：



你也可以不填充文字，只绘制轮廓，使用strokeText方法：

```
ctx.strokeText("My text", 0, 0);
```

结果：



如果没有应用任何字体格式属性，画布默认以10像素无衬线字体渲染文本，这使得fillText和strokeText方法的结果难以区分。详情请参见"格式化文本"示例，了解如何增大字体大小及对文本应用其他美化效果。

第2.6节：格式化文本

fillText和strokeText方法提供的默认字体格式并不十分美观。
幸运的是，画布API提供了用于格式化文本的属性。

使用font属性你可以指定：

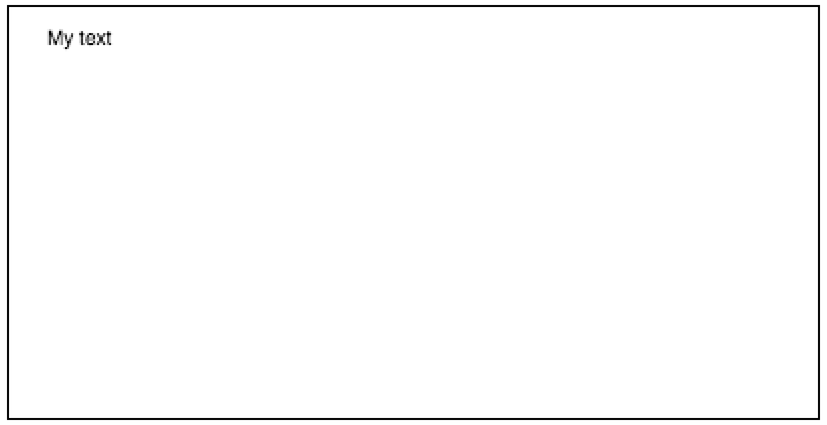
- 字体样式
- 字体变体
- 字体粗细
- 字体大小 / 行高
- 字体系列

例如：

pixels. In the example below the value of 200 restricts the maximum width of the text to 200px:

```
ctx.fillText("My text", 0, 0, 200);
```

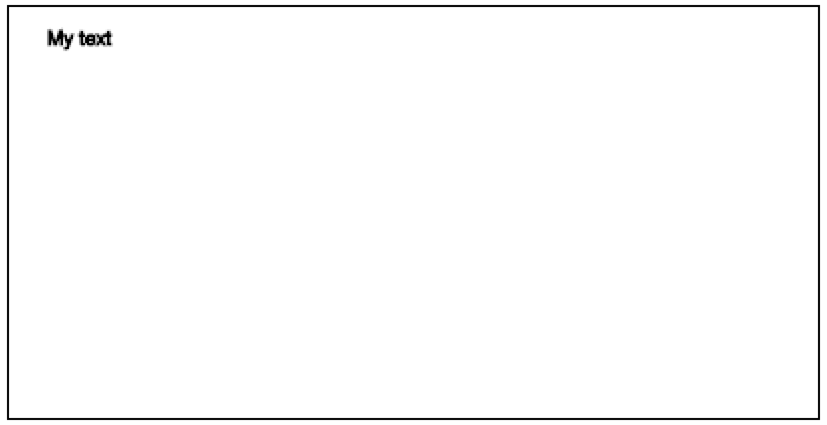
Result:



You can also draw text without a fill, and just an outline instead, using the strokeText method:

```
ctx.strokeText("My text", 0, 0);
```

Result:



Without any font formatting properties applied, the canvas renders text at 10px in sans-serif by default, making it hard to see the difference between the result of the fillText and strokeText methods. See the Formatting Text example for details on how to increase text size and apply other aesthetic changes to text.

Section 2.6: Formatting Text

The default font formatting provided by the fillText and strokeText methods isn't very aesthetically appealing. Fortunately the canvas API provides properties for formatting text.

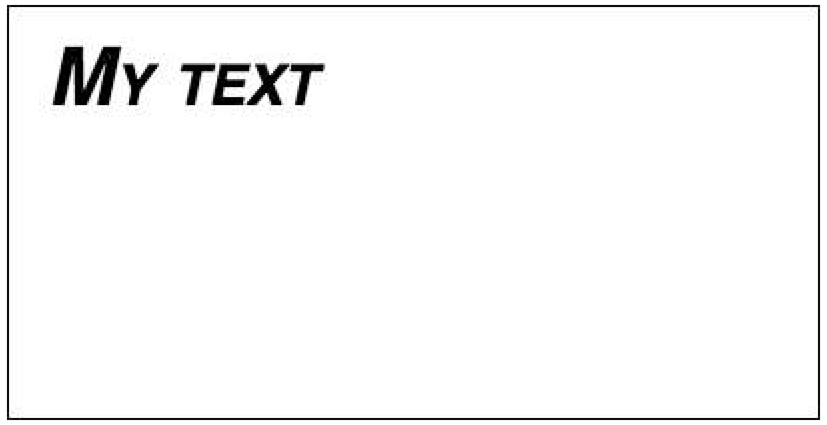
Using the font property you can specify:

- font-style
- font-variant
- font-weight
- font-size / line-height
- font-family

For example:

```
ctx.font = "italic small-caps bold 40px Helvetica, Arial, sans-serif";
ctx.fillText("My text", 20, 50);
```

结果：



使用textAlign属性，你也可以将文本对齐方式更改为：

- 左对齐
- 居中
- 右对齐
- 结束（与右对齐相同）
- 开始（与左对齐相同）

例如：

```
ctx.textAlign = "center";
```

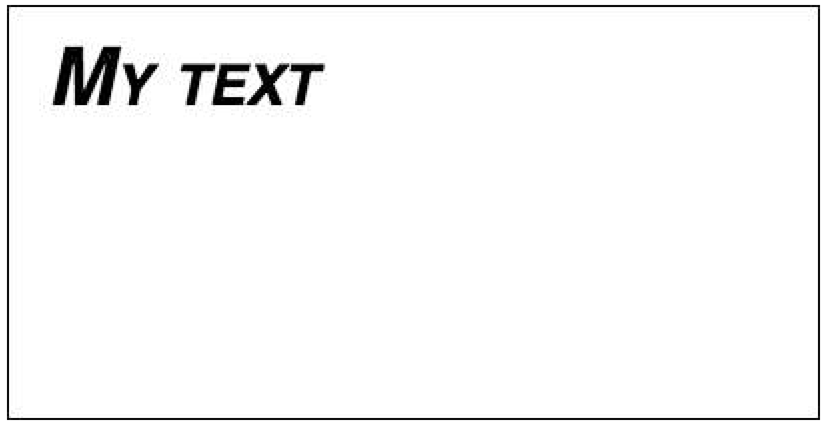
第2.7节：将文本换行成段落

原生Canvas API没有在达到指定最大宽度时自动换行的方法。此示例将文本换行成段落。

```
function wrapText(text, x, y, maxWidth, fontSize, fontFace){
  var firstY=y;
  var words = text.split(' ');
  var line = '';
  var lineHeight=fontSize*1.286; // 10-18像素大小的良好近似值

ctx.font=fontSize+" "+fontFace;
  ctx.textBaseline='top';

  for(var n = 0; n < words.length; n++) {
    var testLine = line + words[n] + ' ';
    var metrics = ctx.measureText(testLine);
    var testWidth = metrics.width;
    if(testWidth > maxWidth) {
      ctx.fillText(line, x, y);
      if(n<words.length-1){
          line = words[n] + ' ';
          y += lineHeight;
      }
    }
    else {
line = testLine;
    }
```

```
ctx.font = "italic small-caps bold 40px Helvetica, Arial, sans-serif";
ctx.fillText("My text", 20, 50);
```

Result:



Using the `textAlign` property you can also change text alignment to either:

- left
- center
- right
- end (same as right)
- start (same as left)

For example:

```
ctx.textAlign = "center";
```

Section 2.7: Wrapping text into paragraphs

Native Canvas API does not have a method to wrap text onto the next line when a desired maximum width is reached. This example wraps text into paragraphs.

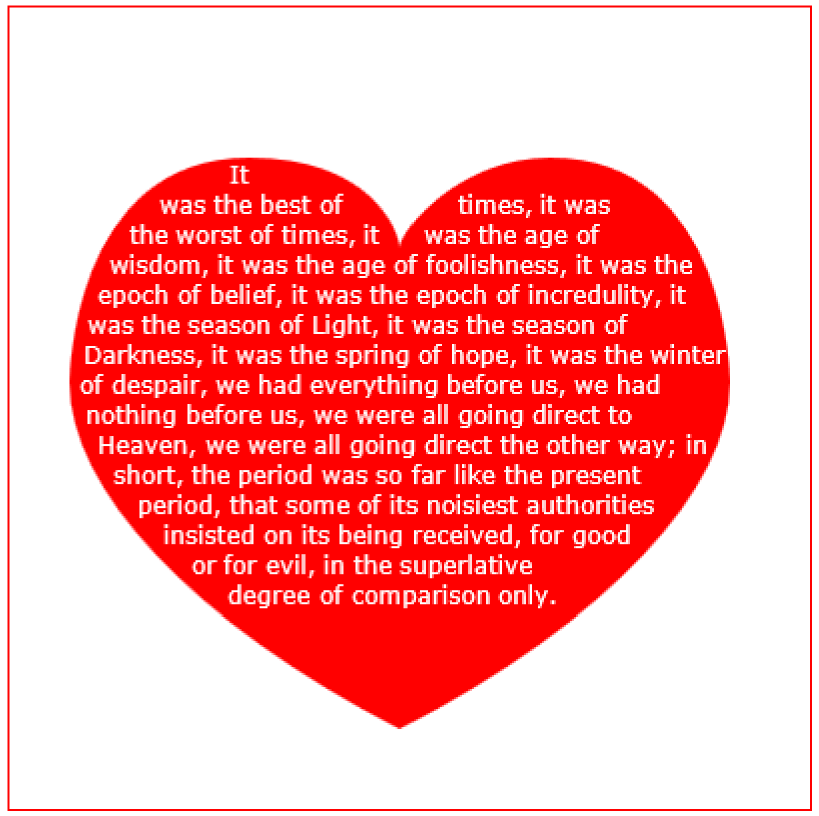
```
function wrapText(text, x, y, maxWidth, fontSize, fontFace){
  var firstY=y;
  var words = text.split(' ');
  var line = '';
  var lineHeight=fontSize*1.286; // a good approx for 10-18px sizes

  ctx.font=fontSize+" "+fontFace;
  ctx.textBaseline='top';

  for(var n = 0; n < words.length; n++) {
    var testLine = line + words[n] + ' ';
    var metrics = ctx.measureText(testLine);
    var testWidth = metrics.width;
    if(testWidth > maxWidth) {
      ctx.fillText(line, x, y);
      if(n<words.length-1){
          line = words[n] + ' ';
          y += lineHeight;
      }
    }
    else {
      line = testLine;
    }
  }
ctx.fillText(line, x, y);
}
```

第2.8节：将文本段落绘制到不规则形状中

此示例将文本段落绘制到画布中所有具有不透明像素的部分。

其工作原理是找到下一个足够大以容纳指定单词的不透明像素块，并用该指定单词填充该像素块。

不透明像素可以来自任何来源：路径绘制命令和/或图像。



It
was the best of times, it was
the worst of times, it was the age of
wisdom, it was the age of foolishness, it was the
epoch of belief, it was the epoch of incredulity, it
was the season of Light, it was the season of
Darkness, it was the spring of hope, it was the winter
of despair, we had everything before us, we had
nothing before us, we were all going direct to
Heaven, we were all going direct the other way; in
short, the period was so far like the present
period, that some of its noisiest authorities
insisted on its being received, for good
or for evil, in the superlative
degree of comparison only.

```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; padding:10px; }
    #canvas{border:1px solid red;}
</style>
<script>
window.onload=(function(){

    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;

    var fontsize=12;
    var fontface='verdana';
    var lineHeight=parseInt(fontsize*1.286);

    var text='那是最好的时代，那是最坏的时代，那是智慧的年代，那是愚昧的年代，那是信仰的纪元，那是不信的纪元，那是
```

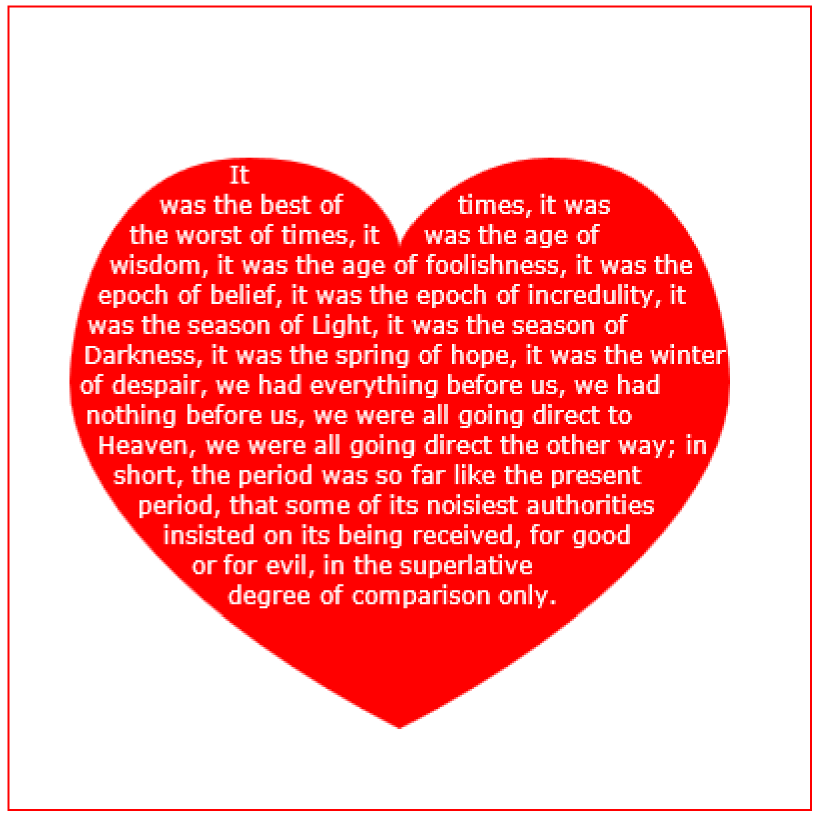
```
  }
  ctx.fillText(line, x, y);
}
```

Section 2.8: Draw text paragraphs into irregular shapes

This example draws text paragraphs into any portions of the canvas that have opaque pixels.

It works by finding the next block of opaque pixels that is large enough to contain the next specified word and filling that block with the specified word.

The opaque pixels can come from any source: Path drawing commands and /or images.



It
was the best of times, it was
the worst of times, it was the age of
wisdom, it was the age of foolishness, it was the
epoch of belief, it was the epoch of incredulity, it
was the season of Light, it was the season of
Darkness, it was the spring of hope, it was the winter
of despair, we had everything before us, we had
nothing before us, we were all going direct to
Heaven, we were all going direct the other way; in
short, the period was so far like the present
period, that some of its noisiest authorities
insisted on its being received, for good
or for evil, in the superlative
degree of comparison only.

```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; padding:10px; }
    #canvas{border:1px solid red;}
</style>
<script>
window.onload=(function(){

    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;

    var fontsize=12;
    var fontface='verdana';
    var lineHeight=parseInt(fontsize*1.286);

    var text='It was the best of times, it was the worst of times, it was the age of wisdom, it was
the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the
```

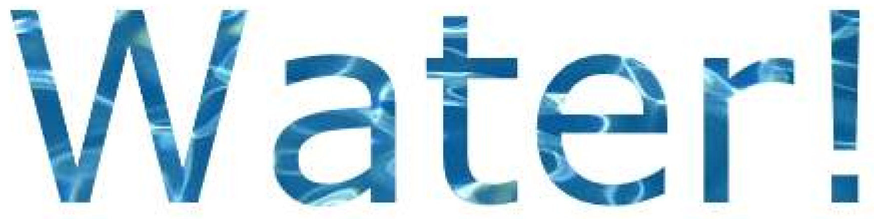
```
光明的季节，那是黑暗的季节，那是希望的春天，那是绝望的冬天，我们面前拥有一切，我们面前一无所有，我们都直奔天堂，我
们都直奔另一个方向；简言之，那一时期与现在的时期极为相似，以至于一些最喧嚣的权威坚持认为，无论好坏，都只能以最高级的比
较形式来接受它。';


    var words=text.split(' ');
    var wordWidths=[];
ctx.font=fontsize+'px '+fontface;
    for(var i=0;i<words.length;i++){ wordWidths.push(ctx.measureText(words[i]).width); }
    var spaceWidth=ctx.measureText(' ').width;
    var wordIndex=0
    var data=[];

    // 演示：绘制心形
    // 注意：形状可以是任何不透明的绘图——甚至是一张图片
    ctx.scale(3,3);
ctx.beginPath();
ctx.moveTo(75,40);
ctx.bezierCurveTo(75,37,70,25,50,25);
    ctx.bezierCurveTo(20,25,20,62.5,20,62.5);
    ctx.bezierCurveTo(20,80,40,102,75,120);
    ctx.bezierCurveTo(110,102,130,80,130,62.5);
    ctx.bezierCurveTo(130,62.5,130,25,100,25);
    ctx.bezierCurveTo(85,25,75,37,75,40);
    ctx.fillStyle='red';
ctx.fill();
ctx.setTransform(1,0,0,1,0,0);

    // 用文本填充心形
ctx.fillStyle='white';
    var imgDataData=ctx.getImageData(0,0,cw,ch).data;
    for(var i=0;i<imgDataData.length;i+=4){
        data.push(imgDataData[i+3]);
    }
placeWords();

    // 依次将单词绘制到下一个可用的不透明像素块中

    function placeWords(){
        var sx=0;
        var sy=0;
        var y=0;
        var wordIndex=0;
ctx.textBaseline='top';
        while(y<ch && wordIndex<words.length){
            sx=0;
sy=y;
            var startingIndex=wordIndex;
            while(sx<cw && wordIndex<words.length){
                var x=getRect(sx,sy,lineHeight);
                var available=x-sx;
                var spacer=spaceWidth;  // spacer=0 to have no left margin
                var w=spacer+wordWidths[wordIndex];
while(available>=w){
                    ctx.fillText(words[wordIndex],spacer+sx,sy);
                    sx+=w;
available-=w;
                    spacer=spaceWidth;
                    wordIndex++;
w=spacer+wordWidths[wordIndex];
                }
sx=x+1;
```

```
season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of
despair, we had everything before us, we had nothing before us, we were all going direct to Heaven,
we were all going direct the other way; in short, the period was so far like the present period,
that some of its noisiest authorities insisted on its being received, for good or for evil, in the
superlative degree of comparison only.';
    var words=text.split(' ');
    var wordWidths=[];
    ctx.font=fontsize+'px '+fontface;
    for(var i=0;i<words.length;i++){ wordWidths.push(ctx.measureText(words[i]).width); }
    var spaceWidth=ctx.measureText(' ').width;
    var wordIndex=0
    var data=[];

    // Demo: draw Heart
    // Note: the shape can be ANY opaque drawing -- even an image
    ctx.scale(3,3);
    ctx.beginPath();
    ctx.moveTo(75,40);
    ctx.bezierCurveTo(75,37,70,25,50,25);
    ctx.bezierCurveTo(20,25,20,62.5,20,62.5);
    ctx.bezierCurveTo(20,80,40,102,75,120);
    ctx.bezierCurveTo(110,102,130,80,130,62.5);
    ctx.bezierCurveTo(130,62.5,130,25,100,25);
    ctx.bezierCurveTo(85,25,75,37,75,40);
    ctx.fillStyle='red';
    ctx.fill();
    ctx.setTransform(1,0,0,1,0,0);

    // fill heart with text
    ctx.fillStyle='white';
    var imgDataData=ctx.getImageData(0,0,cw,ch).data;
    for(var i=0;i<imgDataData.length;i+=4){
        data.push(imgDataData[i+3]);
    }
    placeWords();

    // draw words sequentially into next available block of
    //    available opaque pixels
    function placeWords(){
        var sx=0;
        var sy=0;
        var y=0;
        var wordIndex=0;
        ctx.textBaseline='top';
        while(y<ch && wordIndex<words.length){
            sx=0;
            sy=y;
            var startingIndex=wordIndex;
            while(sx<cw && wordIndex<words.length){
                var x=getRect(sx,sy,lineHeight);
                var available=x-sx;
                var spacer=spaceWidth;  // spacer=0 to have no left margin
                var w=spacer+wordWidths[wordIndex];
                while(available>=w){
                    ctx.fillText(words[wordIndex],spacer+sx,sy);
                    sx+=w;
                    available-=w;
                    spacer=spaceWidth;
                    wordIndex++;
                    w=spacer+wordWidths[wordIndex];
                }
                sx=x+1;
            }
y=(wordIndex>startingIndex)?y+lineHeight:y+1;
        }
    }

    // 查找一块不透明像素的矩形区域
    function getRect(sx,sy,height){
        var x=sx;
        var y=sy;
        var ok=true;
while(ok){
            if(data[y*cw+x]<250){ok=false;}
            y++;
            if(y>=sy+height){
y=sy;
x++;
                if(x>=cw){ok=false;}
            }
        }
        return(x);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <h4>注意：形状必须闭合且内部 alpha>=250 </h4>
    <canvas id="canvas" width=400 height=400></canvas>
</body>
</html>
```

第2.9节：用图像填充文本

此示例使用指定的图像填充文本。

重要！指定的图像必须在调用此函数之前完全加载，否则绘制将失败。使用 image.onload 确保图像完全加载。



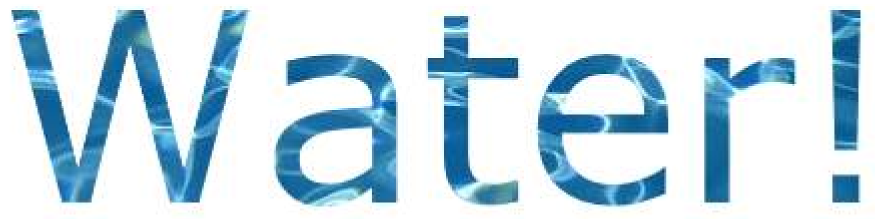
```
function drawImageInsideText(canvas,x,y,img,text,font){
    var c=canvas.cloneNode();
    var ctx=c.getContext('2d');
    ctx.font=font;
ctx.fillText(text,x,y);
    ctx.globalCompositeOperation='source-atop';
    ctx.drawImage(img,0,0);
canvas.getContext('2d').drawImage(c,0,0);
}
```

```
            }
            y=(wordIndex>startingIndex)?y+lineHeight:y+1;
        }
    }

    // find a rectangular block of opaque pixels
    function getRect(sx,sy,height){
        var x=sx;
        var y=sy;
        var ok=true;
        while(ok){
            if(data[y*cw+x]<250){ok=false;}
            y++;
            if(y>=sy+height){
                y=sy;
                x++;
                if(x>=cw){ok=false;}
            }
        }
        return(x);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <h4>Note: the shape must be closed and alpha>=250 inside</h4>
    <canvas id="canvas" width=400 height=400></canvas>
</body>
</html>
```

Section 2.9: Fill text with an image

This example fills text with a specified image.

Important! The specified image must be fully loaded before calling this function or the drawing will fail. Use `image.onload` to be sure the image is fully loaded.

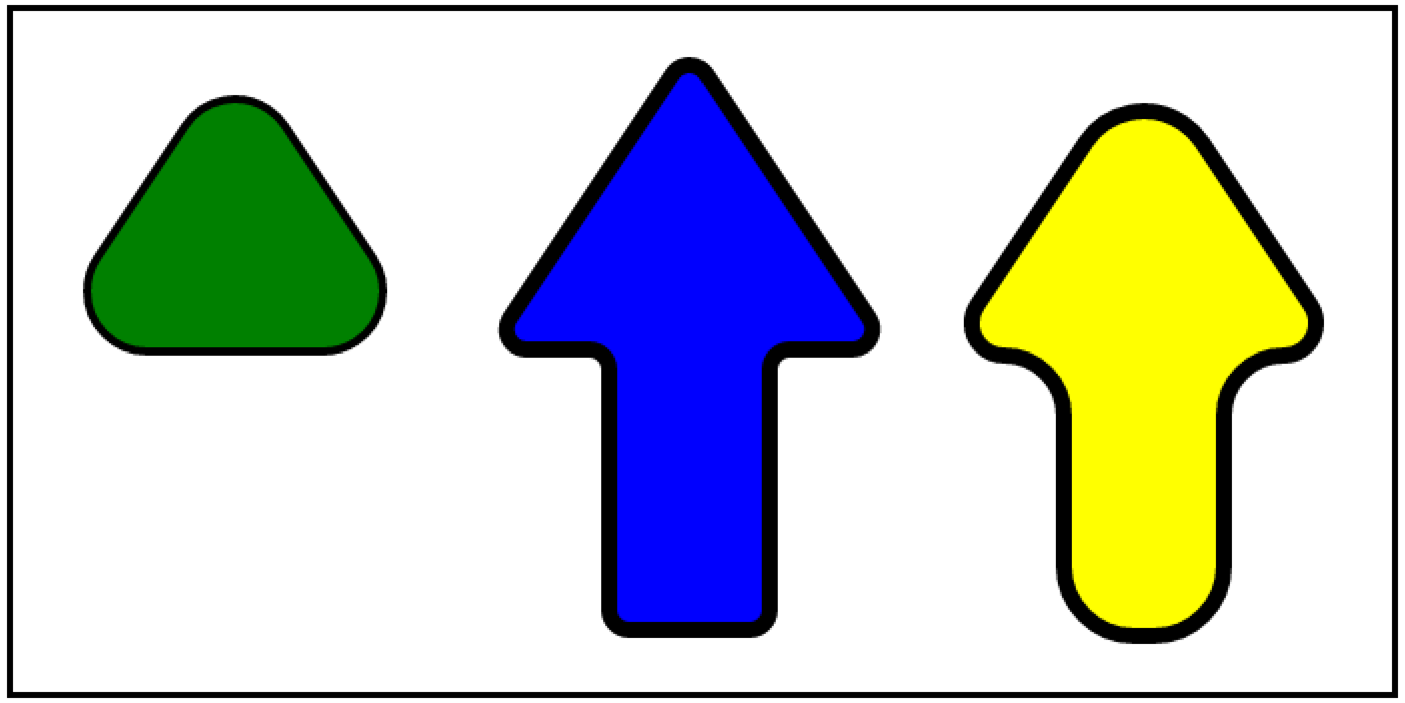


```
function drawImageInsideText(canvas,x,y,img,text,font){
    var c=canvas.cloneNode();
    var ctx=c.getContext('2d');
    ctx.font=font;
    ctx.fillText(text,x,y);
    ctx.globalCompositeOperation='source-atop';
    ctx.drawImage(img,0,0);
    canvas.getContext('2d').drawImage(c,0,0);
}
```

第3章：多边形

第3.1节：绘制圆角多边形

从一组点[{x:?,y:?},{x:?,y:?},...,{x:?,y:?}]创建带有半径为radius的圆角路径。如果角度太小以至于无法容纳该半径，或者角之间的距离不足以留出空间，则圆角半径将被缩小以达到最佳适配。



使用示例

```
var triangle = [
    { x: 200, y : 50 },
    { x: 300, y : 200 },
    { x: 100, y : 200 }
];
var cornerRadius = 30;
ctx.lineWidth = 4;
ctx.fillStyle = "Green";
ctx.strokeStyle = "black";
ctx.beginPath(); // 开始一条新路径
roundedPoly(triangle, cornerRadius);
ctx.fill();
ctx.stroke();
```

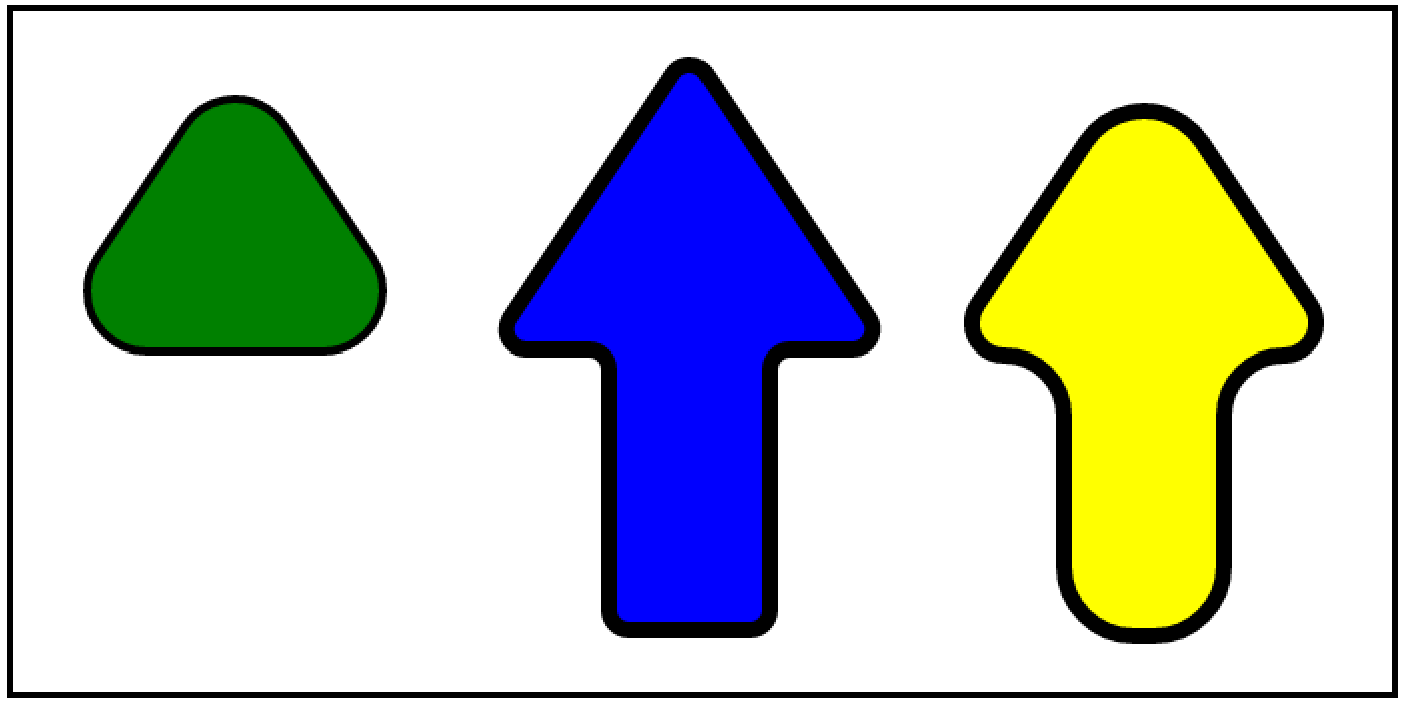
渲染函数

```
var roundedPoly = function(points,radius){
    var i, x, y, len, p1, p2, p3, v1, v2, sinA, sinA90, radDirection, drawDirection, angle,
halfAngle, cRadius, lenOut;
    var asVec = function (p, pp, v) { // 将点转换为具有长度和归一化的向量
v.x = pp.x - p.x; // x,y 作为向量
v.y = pp.y - p.y;
v.len = Math.sqrt(v.x * v.x + v.y * v.y); // 向量长度
v.nx = v.x / v.len; // 归一化
v.ny = v.y / v.len;
v.ang = Math.atan2(v.ny, v.nx); // 向量的方向
    }
v1 = {};
```

Chapter 3: Polygons

Section 3.1: Render a rounded polygon

Creates a path from a set of points [{x:?,y:?},{x:?,y:?},...,{x:?,y:?}] with rounded corners of radius. If the corner angle is too small to fit the radius or the distance between corners does not allow room the corners radius is reduced to a best fit.



Usage Example

```
var triangle = [
    { x: 200, y : 50 },
    { x: 300, y : 200 },
    { x: 100, y : 200 }
];
var cornerRadius = 30;
ctx.lineWidth = 4;
ctx.fillStyle = "Green";
ctx.strokeStyle = "black";
ctx.beginPath(); // start a new path
roundedPoly(triangle, cornerRadius);
ctx.fill();
ctx.stroke();
```

Render function

```
var roundedPoly = function(points,radius){
    var i, x, y, len, p1, p2, p3, v1, v2, sinA, sinA90, radDirection, drawDirection, angle,
halfAngle, cRadius, lenOut;
    var asVec = function (p, pp, v) { // convert points to a line with len and normalised
        v.x = pp.x - p.x; // x,y as vec
        v.y = pp.y - p.y;
        v.len = Math.sqrt(v.x * v.x + v.y * v.y); // length of vec
        v.nx = v.x / v.len; // normalised
        v.ny = v.y / v.len;
        v.ang = Math.atan2(v.ny, v.nx); // direction of vec
    }
    v1 = {};
v2 = {};
len = points.length;                          // 点的数量
    p1 = points[len - 1];                         // 从路径末尾开始
    for (i = 0; i < len; i++) {                   // 处理每个拐角
p2 = points[(i) % len];               // 当前被圆角处理的拐角点
      p3 = points[(i + 1) % len];
        // 获取作为向外的向量的拐角点
asVec(p2, p1, v1);                        // 从拐角点向后的向量
        asVec(p2, p3, v2);                        // 从拐角点向前的向量
        // 计算拐角的叉积（角度的反正弦）
sinA = v1.nx * v2.ny - v1.ny * v2.nx;     // 叉积
        // 计算第一条线与第二条线垂线的叉积
        sinA90 = v1.nx * v2.nx - v1.ny * -v2.ny; // 第二条线法线的叉积
        angle = Math.asin(sinA);                  // 计算角度
radDirection = 1;                        // 可能需要反转半径方向
        drawDirection = false;                    // 可能需要逆时针绘制弧线
        // 找到圆心所在的正确象限
        if (sinA90 < 0) {
            if (angle < 0) {
angle = Math.PI + angle; // 加180度以移动到第三象限
            } else {
angle = Math.PI - angle; // 移回第二象限
             radDirection = -1;
drawDirection = true;
            }
        } 否则 {
            if (angle > 0) {
radDirection = -1;
                drawDirection = true;
            }
        }
halfAngle = angle / 2;
        // 获取从角点到圆角接触线的点的距离
lenOut = Math.abs(Math.cos(halfAngle) * radius / Math.sin(halfAngle));
        if (lenOut > Math.min(v1.len / 2, v2.len / 2)) { // 如果超过半条线长度则修正
          lenOut = Math.min(v1.len / 2, v2.len / 2);
            // 调整圆角半径以适应
cRadius = Math.abs(lenOut * Math.sin(halfAngle) / Math.cos(halfAngle));
        } else {
cRadius = 半径;
        }
x = p2.x + v2.nx * lenOut; // 沿第二条线从角点移动到圆角圆触点处

y = p2.y + v2.ny * lenOut;
x += -v2.ny * cRadius * radDirection; // 从线段移开到圆心
      y += v2.nx * cRadius * radDirection;
        // x,y 是圆角圆的圆心
ctx.arc(x, y, cRadius, v1.ang + Math.PI / 2 * radDirection, v2.ang - Math.PI / 2 *
radDirection, drawDirection); // 顺时针绘制弧线
p1 = p2;
        p2 = p3;
    }
ctx.closePath();
}
```

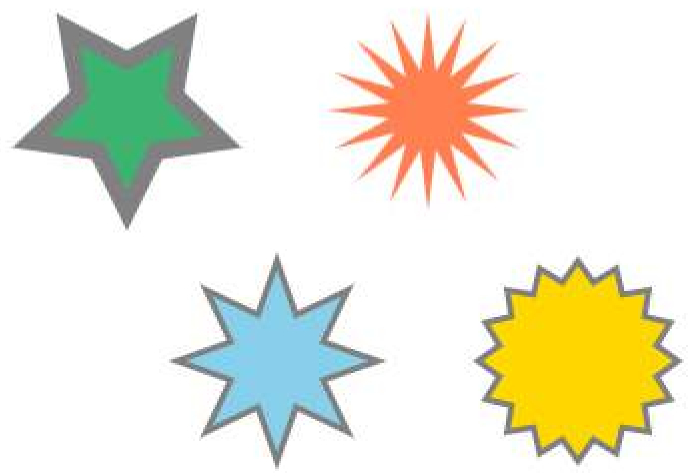
第3.2节：星形

绘制具有灵活样式（大小、颜色、点数）的星形。

```
    v2 = {};
    len = points.length;                         // number points
    p1 = points[len - 1];                        // start at end of path
    for (i = 0; i < len; i++) {                  // do each corner
        p2 = points[(i) % len];                  // the corner point that is being rounded
        p3 = points[(i + 1) % len];
        // get the corner as vectors out away from corner
        asVec(p2, p1, v1);                       // vec back from corner point
        asVec(p2, p3, v2);                       // vec forward from corner point
        // get corners cross product (asin of angle)
        sinA = v1.nx * v2.ny - v1.ny * v2.nx;    // cross product
        // get cross product of first line and perpendicular second line
        sinA90 = v1.nx * v2.nx - v1.ny * -v2.ny; // cross product to normal of line 2
        angle = Math.asin(sinA);                 // get the angle
        radDirection = 1;                        // may need to reverse the radius
        drawDirection = false;                   // may need to draw the arc anticlockwise
        // find the correct quadrant for circle center
        if (sinA90 < 0) {
            if (angle < 0) {
                angle = Math.PI + angle; // add 180 to move us to the 3 quadrant
            } else {
                angle = Math.PI - angle; // move back into the 2nd quadrant
                radDirection = -1;
                drawDirection = true;
            }
        } else {
            if (angle > 0) {
                radDirection = -1;
                drawDirection = true;
            }
        }
        halfAngle = angle / 2;
        // get distance from corner to point where round corner touches line
        lenOut = Math.abs(Math.cos(halfAngle) * radius / Math.sin(halfAngle));
        if (lenOut > Math.min(v1.len / 2, v2.len / 2)) { // fix if longer than half line length
            lenOut = Math.min(v1.len / 2, v2.len / 2);
            // ajust the radius of corner rounding to fit
            cRadius = Math.abs(lenOut * Math.sin(halfAngle) / Math.cos(halfAngle));
        } else {
            cRadius = radius;
        }
        x = p2.x + v2.nx * lenOut; // move out from corner along second line to point where rounded
circle touches
        y = p2.y + v2.ny * lenOut;
        x += -v2.ny * cRadius * radDirection; // move away from line to circle center
        y += v2.nx * cRadius * radDirection;
        // x,y is the rounded corner circle center
        ctx.arc(x, y, cRadius, v1.ang + Math.PI / 2 * radDirection, v2.ang - Math.PI / 2 *
radDirection, drawDirection); // draw the arc clockwise
        p1 = p2;
        p2 = p3;
    }
    ctx.closePath();
}
```

Section 3.2: Stars

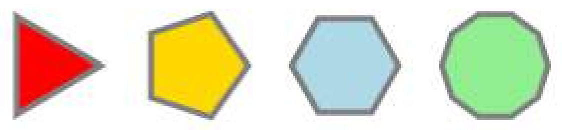
Draw stars with flexible styling (size, colors, number-of-points).



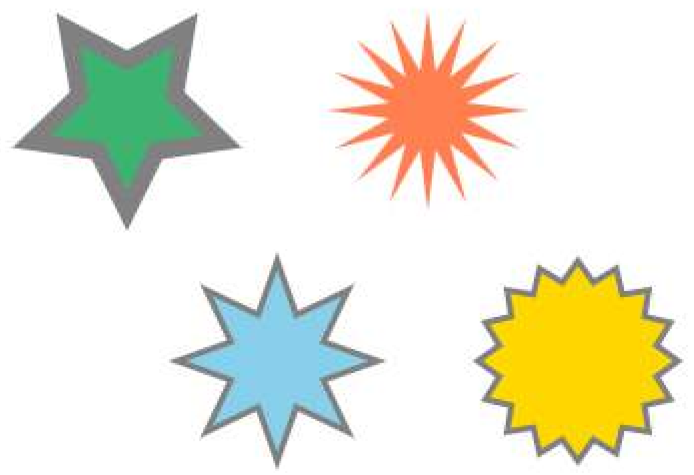
```
// 用法：
drawStar(75,75,5,50,25,'mediumseagreen','gray',9);
drawStar(150,200,8,50,25,'skyblue','gray',3);
drawStar(225,75,16,50,20,'coral','transparent',0);
drawStar(300,200,16,50,40,'gold','gray',3);

// centerX, centerY：星星的中心点
// points：星星外部的顶点数
// inner：星星内部顶点的半径
// outer：星星外部顶点的半径
// fill, stroke：应用的填充色和描边色
// line：描边的线宽

function drawStar(centerX, centerY, points, outer, inner, fill, stroke, line) {
    // 定义星星
ctx.beginPath();
    ctx.moveTo(centerX, centerY+outer);
    for (var i=0; i < 2*points+1; i++) {
        var r = (i%2 == 0)? outer : inner;
        var a = Math.PI * i/points;
ctx.lineTo(centerX + r*Math.sin(a), centerY + r*Math.cos(a));
    };
ctx.closePath();
    // 绘制
ctx.fillStyle=fill;
    ctx.fill();
ctx.strokeStyle=stroke;
    ctx.lineWidth=line;
    ctx.stroke()
}
```

第3.3节：正多边形

正多边形的所有边长都相等。



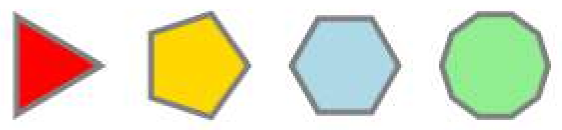
```
// 用法：
drawRegularPolygon(3,25,75,50,6,'gray','red',0);
drawRegularPolygon(5,25,150,50,6,'gray','gold',0);
```



```
// Usage:
drawStar(75,75,5,50,25,'mediumseagreen','gray',9);
drawStar(150,200,8,50,25,'skyblue','gray',3);
drawStar(225,75,16,50,20,'coral','transparent',0);
drawStar(300,200,16,50,40,'gold','gray',3);

// centerX, centerY: the center point of the star
// points: the number of points on the exterior of the star
// inner: the radius of the inner points of the star
// outer: the radius of the outer points of the star
// fill, stroke: the fill and stroke colors to apply
// line: the linewidth of the stroke

function drawStar(centerX, centerY, points, outer, inner, fill, stroke, line) {
    // define the star
    ctx.beginPath();
    ctx.moveTo(centerX, centerY+outer);
    for (var i=0; i < 2*points+1; i++) {
        var r = (i%2 == 0)? outer : inner;
        var a = Math.PI * i/points;
        ctx.lineTo(centerX + r*Math.sin(a), centerY + r*Math.cos(a));
    };
    ctx.closePath();
    // draw
    ctx.fillStyle=fill;
    ctx.fill();
    ctx.strokeStyle=stroke;
    ctx.lineWidth=line;
    ctx.stroke()
}
```

Section 3.3: Regular Polygon

A regular polygon has all sides equal length.



```
// Usage:
drawRegularPolygon(3,25,75,50,6,'gray','red',0);
drawRegularPolygon(5,25,150,50,6,'gray','gold',0);
drawRegularPolygon(6,25,225,50,6,'gray','lightblue',0);
drawRegularPolygon(10,25,300,50,6,'gray','lightgreen',0);

函数
drawRegularPolygon(边数,半径,中心X,中心Y,线宽,线色,填充色,旋转弧度){

    var 角度=Math.PI*2/边数;
    ctx.translate(中心X,中心Y);
    ctx.rotate(旋转弧度);
   ctx.beginPath();
ctx.moveTo(半径,0);
    for(var i=1;i<边数;i++){
        ctx.rotate(角度);
        ctx.lineTo(半径,0);
    }
ctx.closePath();
    ctx.fillStyle=fillColor;
    ctx.strokeStyle = strokeColor;
    ctx.lineWidth = strokeWidth;
    ctx.stroke();
ctx.fill();
ctx.rotate(angles*-(sideCount-1));
    ctx.rotate(-rotationRadians);
ctx.translate(-centerX,-centerY);
}
```

```
drawRegularPolygon(6,25,225,50,6,'gray','lightblue',0);
drawRegularPolygon(10,25,300,50,6,'gray','lightgreen',0);

function
drawRegularPolygon(sideCount,radius,centerX,centerY,strokeWidth,strokeColor,fillColor,rotationRadia
ns){
    var angles=Math.PI*2/sideCount;
    ctx.translate(centerX,centerY);
    ctx.rotate(rotationRadians);
    ctx.beginPath();
    ctx.moveTo(radius,0);
    for(var i=1;i<sideCount;i++){
        ctx.rotate(angles);
        ctx.lineTo(radius,0);
    }
    ctx.closePath();
    ctx.fillStyle=fillColor;
    ctx.strokeStyle = strokeColor;
    ctx.lineWidth = strokeWidth;
    ctx.stroke();
    ctx.fill();
    ctx.rotate(angles*-(sideCount-1));
    ctx.rotate(-rotationRadians);
    ctx.translate(-centerX,-centerY);
}
```

第4章：图像

第4.1节："context.drawImage"没有在画布上显示图像？

确保你的图像对象在尝试用context.drawImage绘制到画布上之前已经完全加载。
否则图像将默默地无法显示。

在JavaScript中，图像不会立即加载。相反，图像是异步加载的，在加载期间，JavaScript会继续执行image.src之后的任何代码。这意味着context.drawImage可能会在图像为空时执行，因此不会显示任何内容。

示例：确保图像完全加载后再尝试用.drawImage绘制

```
var img=new Image();
img.onload=start;
img.onerror=function(){alert(img.src+' 失败');}
img.src="someImage.png";
function start(){
    // start() 会在图像完全加载后调用，无论 start 在代码中的位置如何

}
```

示例：在尝试使用任何图像绘制之前加载多个图像

还有更多功能完善的图像加载器，但此示例说明了如何实现

```
// 第一张图像
var img1=new Image();
img1.onload=start;
img1.onerror=function(){alert(img1.src+' 加载失败。');};
img1.src="imageOne.png";
// 第二张图像
var img2=new Image();
img2.onload=start;
img1.onerror=function(){alert(img2.src+' failed to load.');};
img2.src="imageTwo.png";
//
var imgCount=2;
// start is called every time an image loads
function start(){
    // countdown until all images are loaded
    if(--imgCount>0){return;}
    // All the images are now successfully loaded
    // context.drawImage will successfully draw each one
    context.drawImage(img1,0,0);
    context.drawImage(img2,50,0);
}
```

第4.2节：被污染的画布

当从域外来源或本地文件系统添加内容时，画布会被标记为被污染。尝试访问像素数据或转换为dataURL将抛出安全错误。

```
vr image = new Image();
image.src = "file://myLocalImage.png";
```

Chapter 4: Images

Section 4.1: Is "context.drawImage" not displaying the image on the Canvas?

Make sure your image object is fully loaded before you try to draw it on the canvas with `context.drawImage`. Otherwise the image will silently fail to display.

In JavaScript, images are not loaded immediately. Instead, images are loaded asynchronously and during the time they take to load JavaScript continues executing any code that follows `image.src`. This means `context.drawImage` may be executed with an empty image and therefore will display nothing.

Example making sure the image is fully loaded before trying to draw it with .drawImage

```
var img=new Image();
img.onload=start;
img.onerror=function(){alert(img.src+' failed');}
img.src="someImage.png";
function start(){
    // start() is called AFTER the image is fully loaded regardless
    //     of start's position in the code
}
```

Example loading multiple images before trying to draw with any of them

There are more full-functioned image loaders, but this example illustrates how to do it

```
// first image
var img1=new Image();
img1.onload=start;
img1.onerror=function(){alert(img1.src+' failed to load.');};
img1.src="imageOne.png";
// second image
var img2=new Image();
img2.onload=start;
img1.onerror=function(){alert(img2.src+' failed to load.');};
img2.src="imageTwo.png";
//
var imgCount=2;
// start is called every time an image loads
function start(){
    // countdown until all images are loaded
    if(--imgCount>0){return;}
    // All the images are now successfully loaded
    // context.drawImage will successfully draw each one
    context.drawImage(img1,0,0);
    context.drawImage(img2,50,0);
}
```

Section 4.2: The Tained canvas

When adding content from sources outside your domain, or from the local file system the canvas is marked as tainted. Attempt to access the pixel data, or convert to a dataURL will throw a security error.

```
vr image = new Image();
image.src = "file://myLocalImage.png";
image.onload = function(){
ctx.drawImage(this,0,0);
ctx.getImageData(0,0,canvas.width,canvas.height);   // 抛出安全错误
}
```

此示例只是一个示范，旨在吸引对详细理解有深入了解的人进行阐述。

第4.3节：使用canvas进行图像裁剪

此示例展示了一个简单的图像裁剪函数，该函数接受一张图像和裁剪坐标，并返回裁剪后的图像。

```
function cropImage(image, croppingCoords) {
    var cc = croppingCoords;
    var workCan = document.createElement("canvas"); // 创建一个canvas
    workCan.width = Math.floor(cc.width);   // 将canvas分辨率设置为裁剪图像的大小
   workCan.height = Math.floor(cc.height);
    var ctx = workCan.getContext("2d");     // 获取2D渲染接口
    ctx.drawImage(image, -Math.floor(cc.x), -Math.floor(cc.y)); // 绘制图像，偏移以正确放置
它在裁剪区域内
image.src = workCan.toDataURL();       // 将图像源设置为canvas的Data URL
    return image;
}
```

使用方法

```
var image = new Image();
image.src = "image URL"; // 加载图片
image.onload = function () {  // 加载完成时
   cropImage(
        this, {
x : this.width / 4,      // 裁剪时保持中心
       y : this.height / 4,
width : this.width / 2,
        height : this.height / 2,
    });
document.body.appendChild(this);  // 将图片添加到DOM中
};
```

第4.4节：缩放图片以适应或填充

缩放以适应

意味着整个图片都将可见，但如果图片的宽高比与画布不同，左右或上下可能会有一些空白。示例显示了缩放以适应的图片。两侧的蓝色是因为图片的宽高比与画布不同。

```
image.onload = function(){
    ctx.drawImage(this,0,0);
    ctx.getImageData(0,0,canvas.width,canvas.height);  // throws a security error
}
```

This example is just a stub to entice someone with a detailed understanding elaborate.

Section 4.3: Image cropping using canvas

This example shows a simple image cropping function that takes an image and cropping coordinates and returns the cropped image.

```
function cropImage(image, croppingCoords) {
    var cc = croppingCoords;
    var workCan = document.createElement("canvas"); // create a canvas
    workCan.width = Math.floor(cc.width);  // set the canvas resolution to the cropped image size
    workCan.height = Math.floor(cc.height);
    var ctx = workCan.getContext("2d");    // get a 2D rendering interface
    ctx.drawImage(image, -Math.floor(cc.x), -Math.floor(cc.y)); // draw the image offset to place
it correctly on the cropped region
    image.src = workCan.toDataURL();       // set the image source to the canvas as a data URL
    return image;
}
```

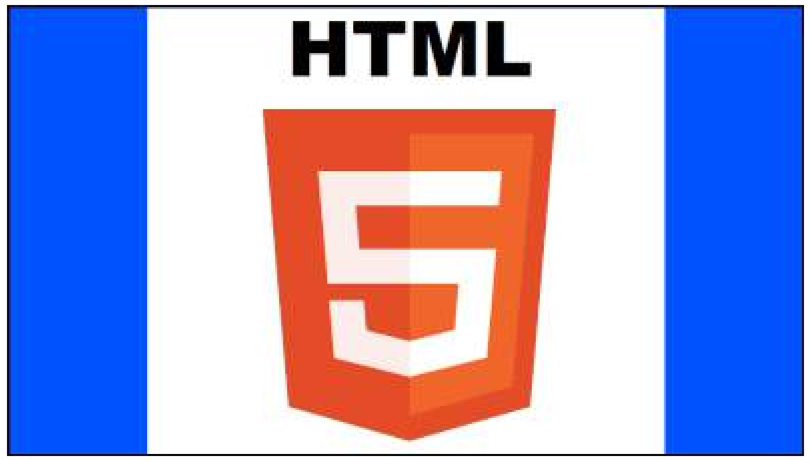
To use

```
var image = new Image();
image.src = "image URL"; // load the image
image.onload = function () {  // when loaded
    cropImage(
        this, {
        x : this.width / 4,     // crop keeping the center
        y : this.height / 4,
        width : this.width / 2,
        height : this.height / 2,
    });
    document.body.appendChild(this);  // Add the image to the DOM
};
```

Section 4.4: Scaling image to fit or fill

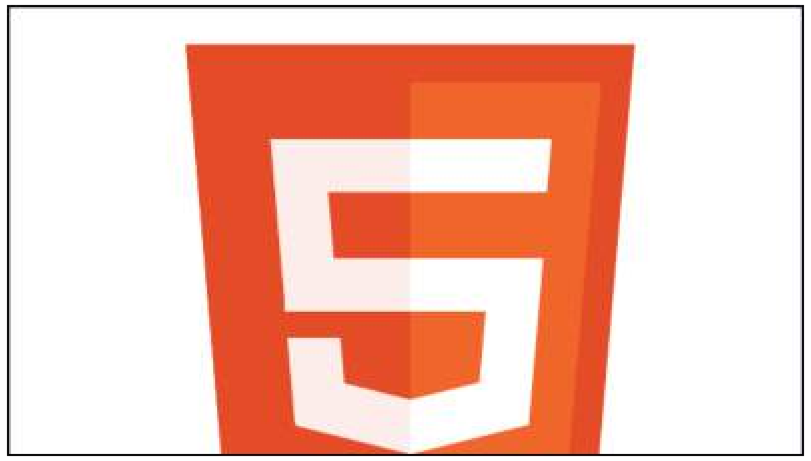
Scaling to fit

Means that the whole image will be visible but there may be some empty space on the sides or top and bottom if the image is not the same aspect as the canvas. The example shows the image scaled to fit. The blue on the sides is due to the fact that the image is not the same aspect as the canvas.



缩放以填充

意味着图像被缩放，以便图像覆盖画布上的所有像素。如果图像的宽高比与画布不同，则图像的某些部分将被裁剪。示例显示图像被缩放以填充画布。
注意图像的顶部和底部不再可见。

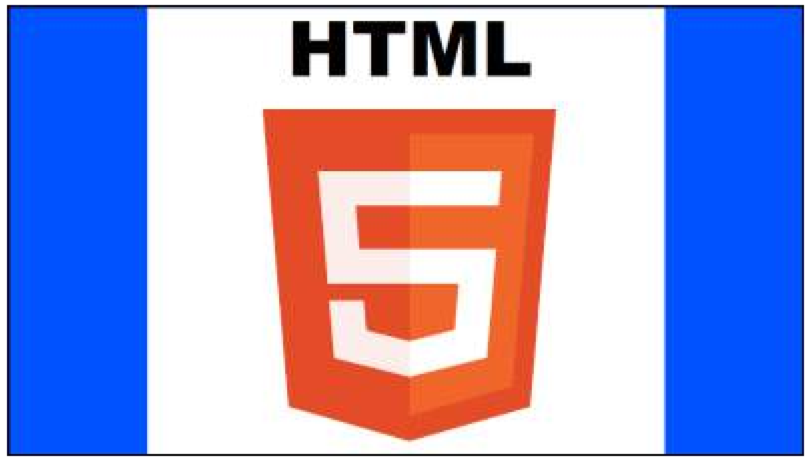


示例 缩放以适应

```
var image = new Image();
image.src = "imgURL";
image.onload = function(){
    scaleToFit(this);
}

function scaleToFit(img){
    // 获取缩放比例
    var scale = Math.min(canvas.width / img.width, canvas.height / img.height);
    // 获取图像左上角位置
    var x = (canvas.width / 2) - (img.width / 2) * scale;
    var y = (canvas.height / 2) - (img.height / 2) * scale;
    ctx.drawImage(img, x, y, img.width * scale, img.height * scale);
}
```

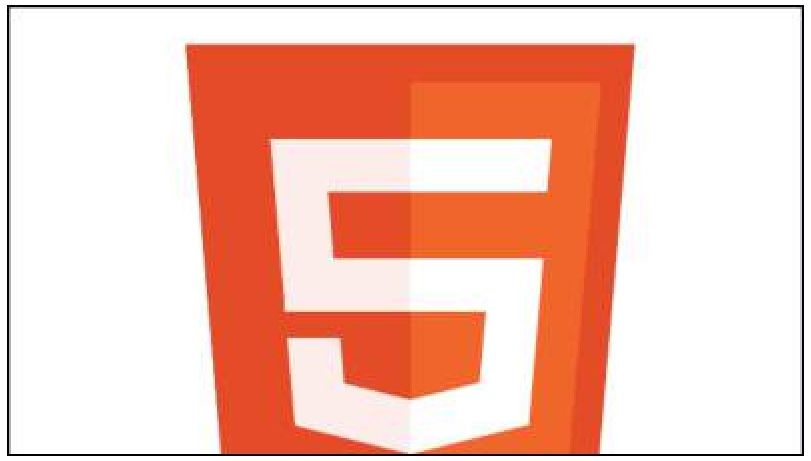
示例 缩放以填充

```
var image = new Image();
image.src = "imgURL";
image.onload = function(){
    scaleToFill(this);
}

function scaleToFill(img){
```



Scaling to fill

Means that the image is scaled so that all the canvas pixels will be covered by the image. If the image aspect is not the same as the canvas then some parts of the image will be clipped. The example shows the image scaled to fill. Note how the top and bottom of the image are no longer visible.



Example Scale to fit

```
var image = new Image();
image.src = "imgURL";
image.onload = function(){
    scaleToFit(this);
}

function scaleToFit(img){
    // get the scale
    var scale = Math.min(canvas.width / img.width, canvas.height / img.height);
    // get the top left position of the image
    var x = (canvas.width / 2) - (img.width / 2) * scale;
    var y = (canvas.height / 2) - (img.height / 2) * scale;
    ctx.drawImage(img, x, y, img.width * scale, img.height * scale);
}
```

Example Scale to fill

```
var image = new Image();
image.src = "imgURL";
image.onload = function(){
    scaleToFill(this);
}

function scaleToFill(img){
    // 获取缩放比例
    var scale = Math.max(canvas.width / img.width, canvas.height / img.height);
    // 获取图像左上角位置
    var x = (canvas.width / 2) - (img.width / 2) * scale;
    var y = (canvas.height / 2) - (img.height / 2) * scale;
    ctx.drawImage(img, x, y, img.width * scale, img.height * scale);
}
```

这两个函数唯一的区别是获取缩放比例。fit 使用最小适应缩放比例，而 fill 使用最大适应缩放比例。

```
    // get the scale
    var scale = Math.max(canvas.width / img.width, canvas.height / img.height);
    // get the top left position of the image
    var x = (canvas.width / 2) - (img.width / 2) * scale;
    var y = (canvas.height / 2) - (img.height / 2) * scale;
    ctx.drawImage(img, x, y, img.width * scale, img.height * scale);
}
```

The only differance between the two functions is getting the scale. The fit uses the min fitting scale will the fill uses the max fitting scale.

第5章：路径（仅语法）

第5.1节：createPattern（创建路径样式对象）

```
var pattern = createPattern(imageObject,repeat)
```

创建一个可重用的图案（对象）。

该对象可以被分配给任何strokeStyle和/或fillStyle。

然后 stroke() 或 fill() 将使用该对象的图案绘制路径。

参数：

- imageObject 是将用作图案的图像。图像来源可以是：
  - HTMLImageElement --- 一个 img 元素或一个新的 Image()，
  - HTMLCanvasElement --- 一个 canvas 元素，
  - HTMLVideoElement --- 一个视频元素（将抓取当前视频帧）
  - ImageBitmap，
  - Blob。
- repeat 决定 imageObject 在画布上的重复方式（类似于 CSS 背景）。
  该参数必须用引号括起，有效值为：
  - "repeat" --- 图案将在水平方向和垂直方向填满画布
  - "repeat-x" --- 图案仅在水平方向重复（1 行水平）
  - "repeat-y" --- 图案仅在垂直方向重复（1 列垂直）
  - "repeat none" --- 图案仅出现一次（在左上角）

pattern 对象 是一个对象，你可以使用（并重复使用！）它使路径的描边和填充变成图案化。

旁注：图案对象不是Canvas元素或其上下文的内部对象。它是一个独立且可重用的JavaScript对象，你可以将其分配给任何你想要的路径。你甚至可以使用该对象将图案应用于不同Canvas元素上的路径(!)

关于Canvas图案的重要提示！

当你创建一个图案对象时，整个画布会被该图案“隐形”填充（受`repeat`
参数影响）。

当你对路径执行stroke()或fill()操作时，隐形的图案才会显现，但仅在被描边或填充的路径上显现。

1. 从你想用作图案的图像开始。重要(!)：确保你的图像已完全加载
   （使用`patternimage.onload`）后，才能尝试用它来创建图案。

Chapter 5: Path (Syntax only)

Section 5.1: createPattern (creates a path styling object)

```
var pattern = createPattern(imageObject,repeat)
```

Creates a reusable pattern (object).

The object can be assigned to any `strokeStyle` and/or `fillStyle`.

Then stroke() or fill() will paint the Path with the pattern of the object.

Arguments:

- **imageObject** is an image that will be used as a pattern. The source of the image can be:
  - HTMLImageElement --- a img element or a new Image(),
  - HTMLCanvasElement --- a canvas element,
  - HTMLVideoElement --- a video element (will grab the current video frame)
  - ImageBitmap,
  - Blob.
- **repeat** determines how the imageObject will be repeated across the canvas (much like a CSS background). This argument must be quote delimited and valid values are:
  - "repeat" --- the pattern will horizontally & vertically fill the canvas
  - "repeat-x" --- the pattern will only repeat horizontally (1 horizontal row)
  - "repeat-y" --- the pattern will only repeat vertically (1 vertical row)
  - "repeat none" --- the pattern appears only once (on the top left)

The pattern object is an object that you can use (and reuse!) to make your path strokes and fills become patterned.

Side Note: The pattern object is not internal to the Canvas element nor it's Context. It is a separate and reusable JavaScript object that you can assign to any Path you desire. You can even use this object to apply pattern to a Path on a different Canvas element(!)

Important hint about Canvas patterns!

When you create a pattern object, the entire canvas is "invisibly" filled with that pattern (subject to the `repeat` argument).

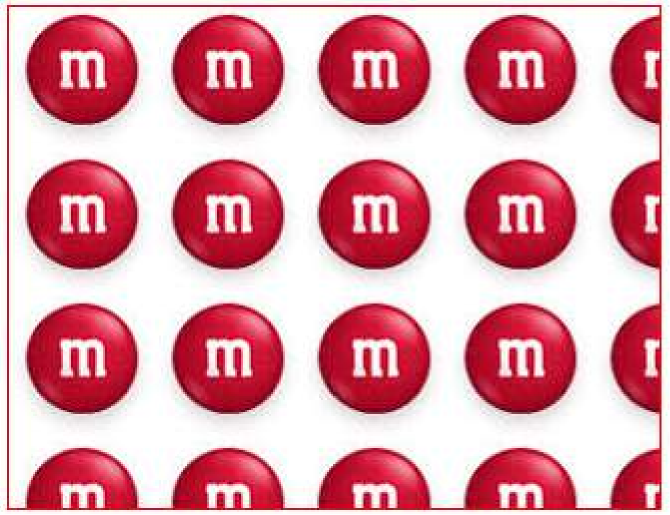
When you `stroke()` or `fill()` a path, the invisible pattern is revealed, but only revealed over that path being stroked or filled.

1. Start with an image that you want to use as a pattern. Important(!): Be sure your image has fully loaded (using `patternimage.onload`) before you attempt to use it to create your pattern.

2. 你可以这样创建图案：

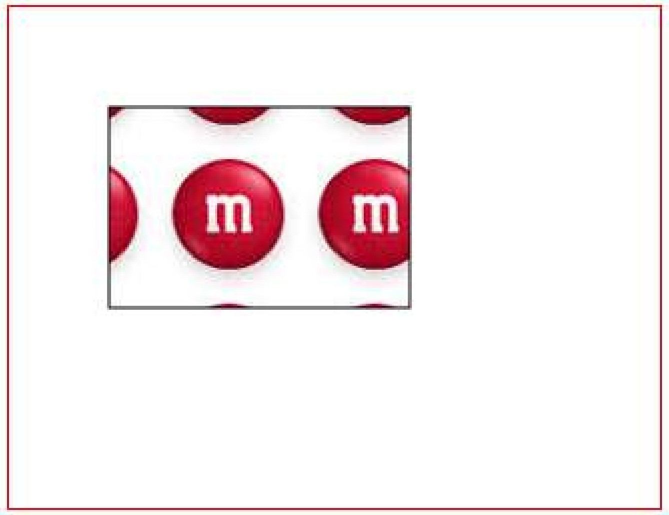
```
// 创建一个图案
var pattern = ctx.createPattern(patternImage,'repeat');
ctx.fillStyle=pattern;
```

3. 然后Canvas会“隐形”地看到你的图案创建，如下所示：



4. 但直到你使用stroke()或fill()填充图案时，你才会在画布上看到图案。

5.最后，如果你使用图案描边或填充路径，那个“不可见”的图案将在画布上变得可见……但仅在路径被绘制的地方。

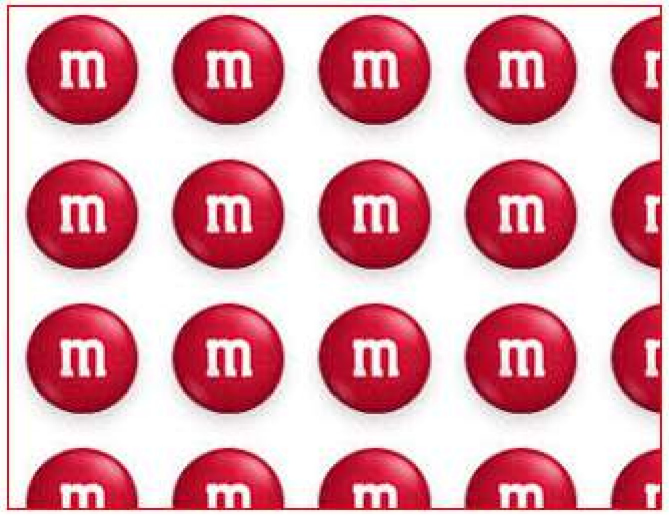


```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
```

2. You create a pattern like this:

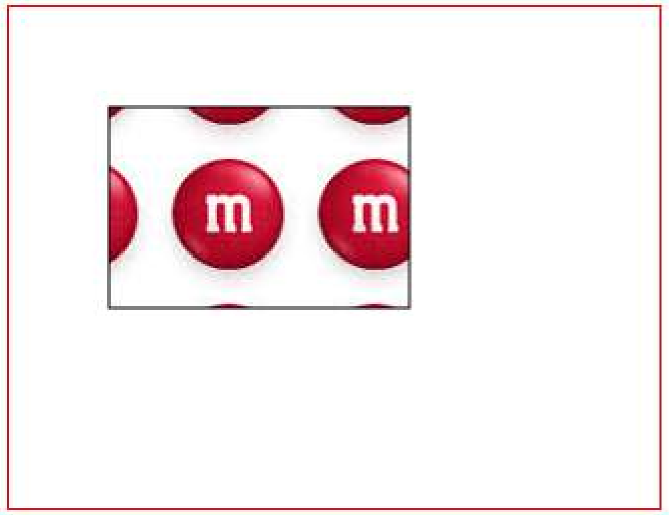
```
// create a pattern
var pattern = ctx.createPattern(patternImage,'repeat');
ctx.fillStyle=pattern;
```

3. Then Canvas will "invisibly" see your pattern creation like this:



4. But until you `stroke()` or `fill()` with the pattern, you will see none of the pattern on the Canvas.

5. Finally, if you stroke or fill a path using the pattern, the "invisible" pattern becomes visible on the Canvas ... but only where the path is drawn.



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    // 使用图案填充
    var patternImage=new Image();
    // 重要！
    // 始终使用 .onload 确保图像
    // 在使用 .createPattern 之前已完全加载
patternImage.onload=function(){
        // 创建一个图案对象
        var pattern = ctx.createPattern(patternImage,'repeat');
        // 将填充样式设置为该图案
ctx.fillStyle=pattern;
        // 用图案填充矩形
ctx.fillRect(50,50,150,100);
        // 仅作演示，描边矩形以便清晰显示
ctx.strokeRect(50,50,150,100);
    }
patternImage.src='http://i.stack.imgur.com/K9EZl.png';

}); // 窗口加载结束
</script>
</head>
<body>
    <canvas id="canvas" width=325 height=250></canvas>
</body>
</html>
```

第5.2节：stroke（路径命令）

```
context.stroke()
```

导致路径的周长根据当前context.strokeStyle进行描边，描边后的路径在画布上被可视化绘制。

在执行context.stroke（或context.fill）之前，路径存在于内存中，尚未在画布上可视化绘制。

描边绘制的异常方式

考虑这个示例路径，它绘制了一条从[0,5]到[5,5]的1像素黑线：

```
// 从[0,5]到[5,5]绘制一条1像素的黑线
context.strokeStyle='black';
context.lineWidth=1;
context.beginPath();
context.moveTo(0,5);
context.lineTo(5,5);
context.stroke();
```

问题：浏览器实际上在画布上绘制了什么？

你可能期望在y=5上得到6个黑色像素

```
    // fill using a pattern
    var patternImage=new Image();
    // IMPORTANT!
    // Always use .onload to be sure the image has
    //     fully loaded before using it in .createPattern
    patternImage.onload=function(){
        // create a pattern object
        var pattern = ctx.createPattern(patternImage,'repeat');
        // set the fillstyle to that pattern
        ctx.fillStyle=pattern;
        // fill a rectangle with the pattern
        ctx.fillRect(50,50,150,100);
        // demo only, stroke the rect for clarity
        ctx.strokeRect(50,50,150,100);
    }
    patternImage.src='http://i.stack.imgur.com/K9EZl.png';

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=325 height=250></canvas>
</body>
</html>
```

Section 5.2: stroke (a path command)

```
context.stroke()
```

Causes the perimeter of the Path to be stroked according to the current `context.strokeStyle` and the stroked Path is visually drawn onto the canvas.

Prior to executing `context.stroke` (or `context.fill`) the Path exists in memory and is not yet visually drawn on the canvas.

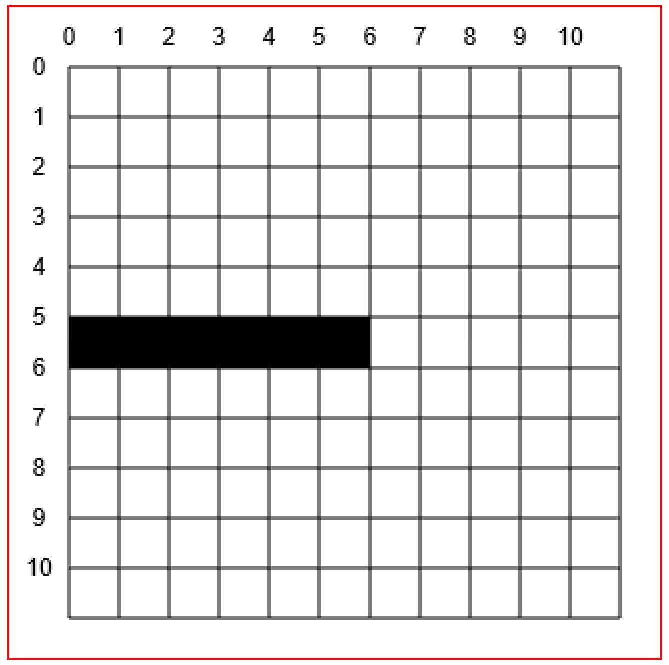
The unusual way strokes are drawn

Consider this example Path that draws a 1 pixel black line from `[0,5]` to `[5,5]`:

```
// draw a 1 pixel black line from [0,5] to [5,5]
context.strokeStyle='black';
context.lineWidth=1;
context.beginPath();
context.moveTo(0,5);
context.lineTo(5,5);
context.stroke();
```

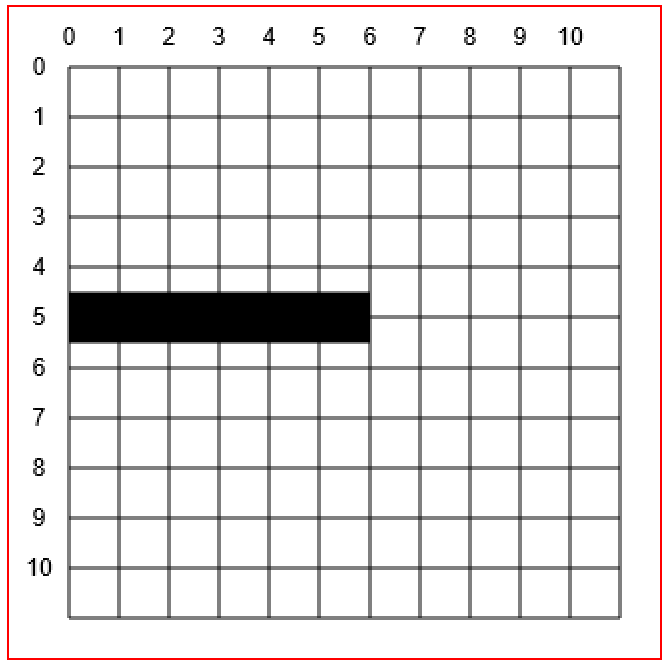
Question: What does the browser actually draw on the canvas?

You probably expect to get 6 black pixels on y=5



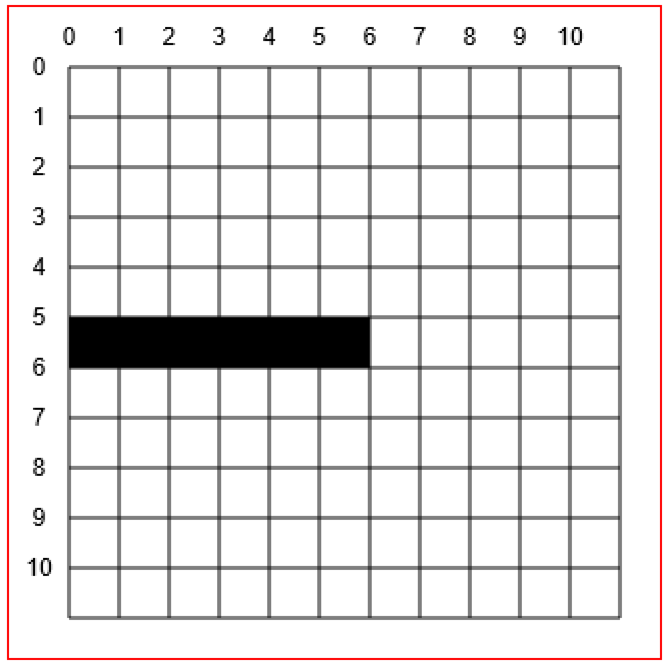
但是(!) ... 画布总是在线条定义路径的两侧各绘制一半的笔触！

所以由于线条定义在 y==5.0，画布想要在线条的 y==4.5和 y==5.5之间绘制线条



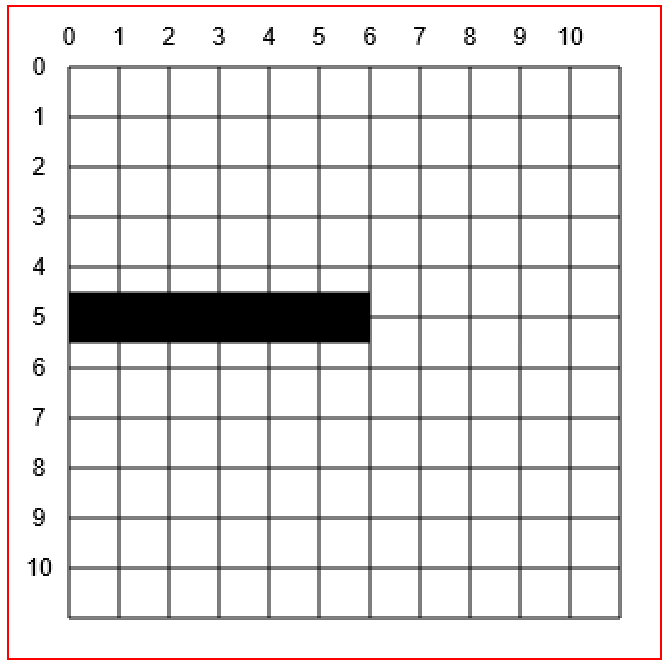
但是，再次强调(!) ... 计算机显示器无法绘制半个像素！

那么，如何处理那些不需要的半个像素（下图蓝色部分）呢？



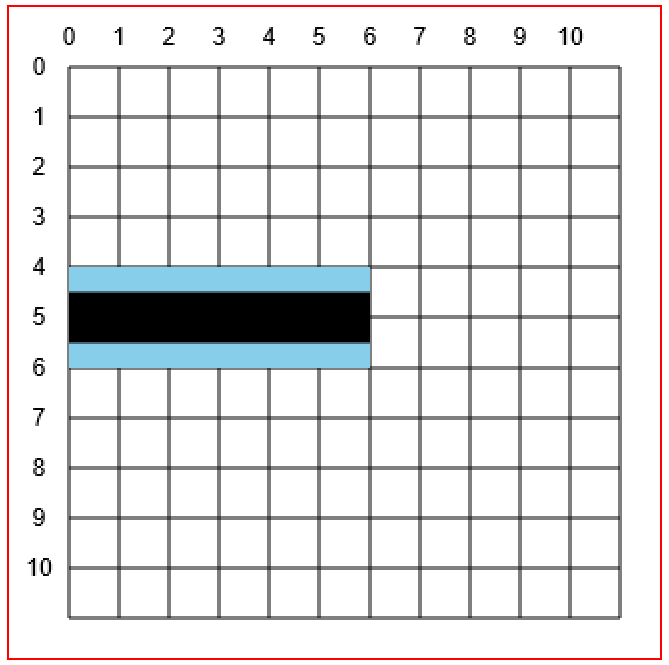
But(!) ... Canvas always draws strokes half-way to either side of the it's defined path!

So since the line is defined at `y==5.0` Canvas wants to draw the line between `y==4.5` and `y==5.5`

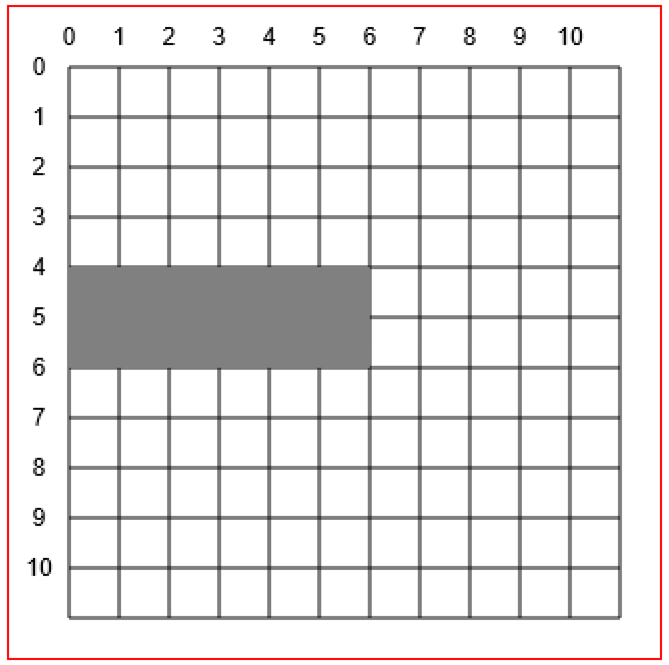


But, again(!) ... The computer display cannot draw half-pixels!

So what is to be done with the undesired half-pixels (shown in blue below)?



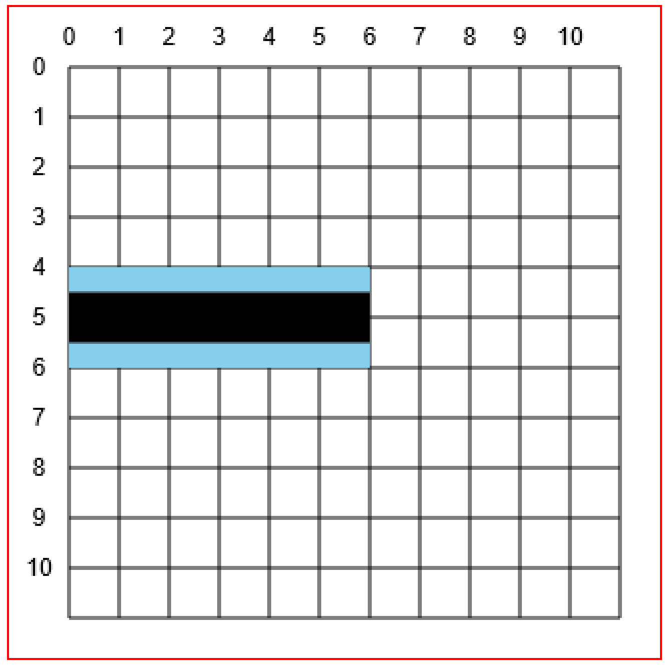
答案是画布实际上命令显示器绘制一条从4.0到6.0宽度为2像素的线。它还会将线条颜色调得比定义的黑色更浅。这种奇怪的绘制行为称为“抗锯齿”，它帮助画布避免绘制看起来锯齿状的笔触。



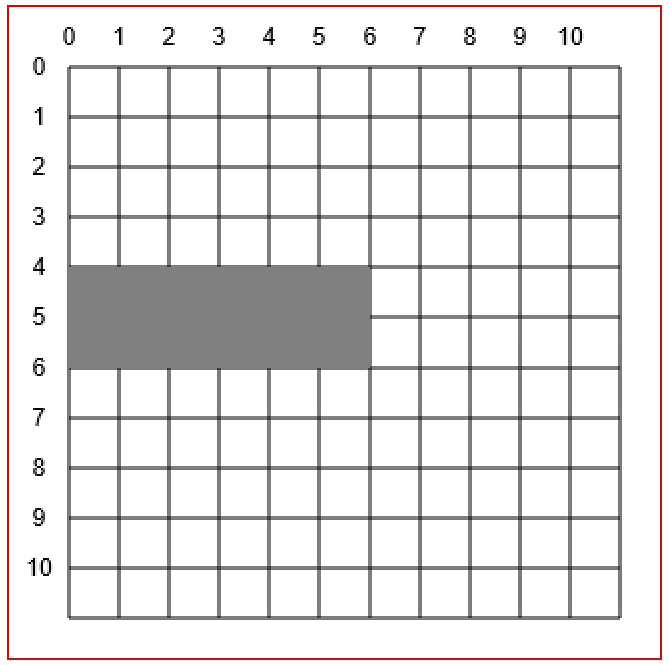
一个仅对完全水平和垂直笔触有效的调整技巧

你可以通过指定在线条的半像素位置绘制线条，获得1像素宽的纯黑线：

```
context.moveTo(0,5.5);
context.lineto(5,5.5);
```



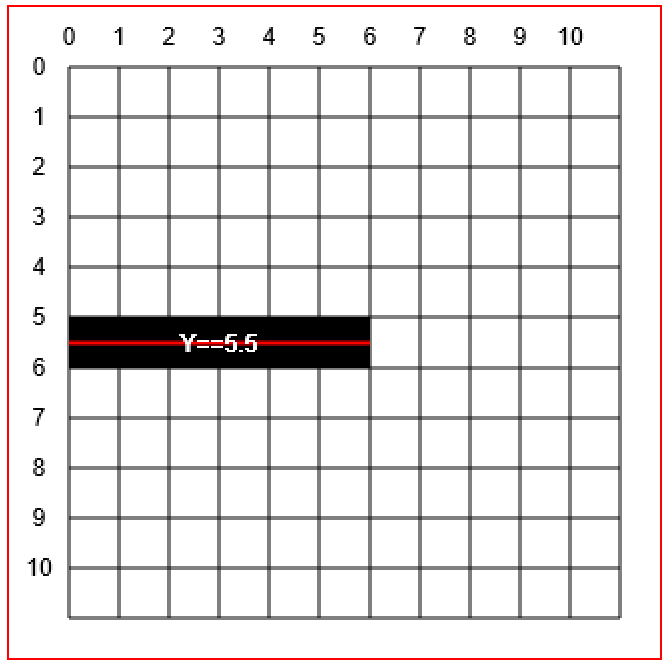
The answer is that Canvas actually orders the display to draw a 2 pixel wide line from `4.0` to `6.0`. It also colors the line lighter than the defined `black`. This strange drawing behavior is "anti-aliasing" and it helps Canvas avoid drawing strokes that look jagged.



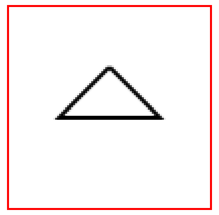
An adjusting trick that ONLY works for exactly horizontal and vertical strokes

You can get a 1 pixel solid black line by specifying the line be drawn on the half-pixel:

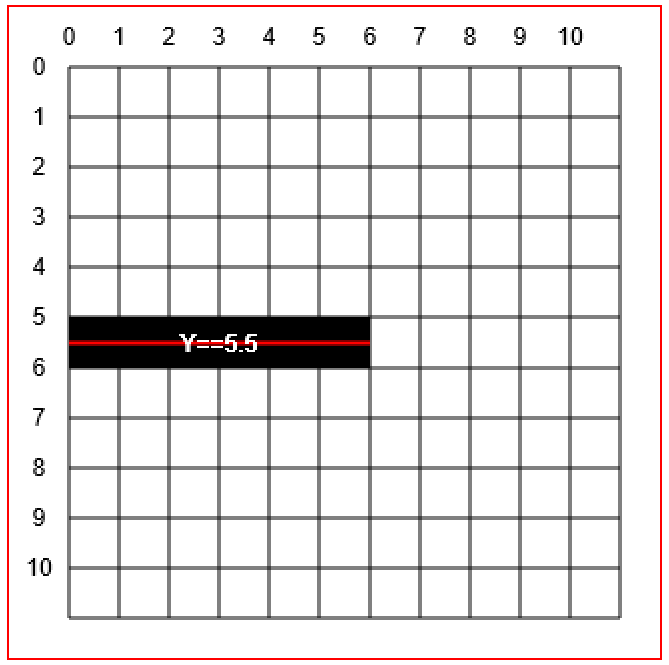
```
context.moveTo(0,5.5);
context.lineto(5,5.5);
```



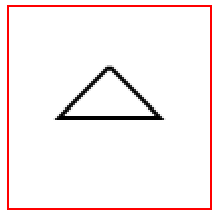
使用context.stroke()在画布上绘制描边路径的示例代码：



```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

ctx.beginPath();
    ctx.moveTo(50,30);
    ctx.lineTo(75,55);
    ctx.lineTo(25,55);
    ctx.lineTo(50,30);
    ctx.lineWidth=2;
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=100 height=100></canvas>
</body>
</html>
```



Example code using `context.stroke()` to draw a stroked Path on the canvas:



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    ctx.beginPath();
    ctx.moveTo(50,30);
    ctx.lineTo(75,55);
    ctx.lineTo(25,55);
    ctx.lineTo(50,30);
    ctx.lineWidth=2;
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=100 height=100></canvas>
</body>
</html>
```

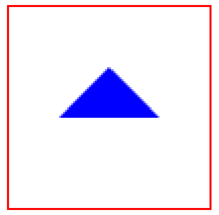
第5.3节：fill（路径命令）

```
context.fill()
```

根据当前的context.fillStyle，填充路径内部，并将填充后的路径视觉上绘制到画布上。

在执行context.fill（或context.stroke）之前，路径存在于内存中，尚未在画布上视觉绘制。

使用context.fill()在画布上绘制填充路径的示例代码：



```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

ctx.beginPath();
    ctx.moveTo(50,30);
    ctx.lineTo(75,55);
    ctx.lineTo(25,55);
    ctx.lineTo(50,30);
    ctx.fillStyle='blue';
    ctx.fill();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=100 height=100></canvas>
</body>
</html>
```

第5.4节：clip（路径命令）

```
context.clip
```

限制任何未来的绘图仅显示在当前路径内。

示例：将此图像裁剪到三角形路径内

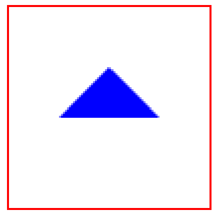
Section 5.3: fill (a path command)

```
context.fill()
```

Causes the inside of the Path to be filled according to the current `context.fillStyle` and the filled Path is visually drawn onto the canvas.

Prior to executing `context.fill` (or `context.stroke`) the Path exists in memory and is not yet visually drawn on the canvas.

Example code using `context.fill()` to draw a filled Path on the canvas:



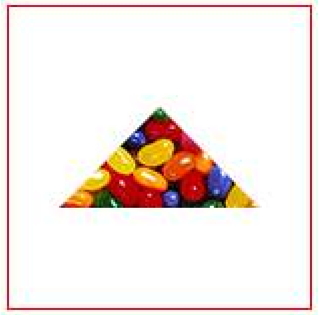
```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    ctx.beginPath();
    ctx.moveTo(50,30);
    ctx.lineTo(75,55);
    ctx.lineTo(25,55);
    ctx.lineTo(50,30);
    ctx.fillStyle='blue';
    ctx.fill();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=100 height=100></canvas>
</body>
</html>
```

Section 5.4: clip (a path command)

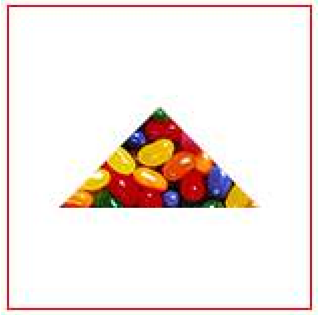
```
context.clip
```

Limits any future drawings to display only inside the current Path.

Example: Clip this image into a triangular Path



```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    var img=new Image();
img.onload=start;
img.src='http://i.stack.imgur.com/1CqWf.jpg'

    function start(){
        // 绘制一个三角形路径
        ctx.beginPath();
        ctx.moveTo(75,50);
        ctx.lineTo(125,100);
        ctx.lineTo(25,100);
        ctx.lineTo(75,50);

        // 将未来的绘图裁剪，仅显示在三角形内
ctx.clip();

        // 绘制图像
ctx.drawImage(img,0,0);
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=150 height=150></canvas>
</body>
</html>
```



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    var img=new Image();
    img.onload=start;
    img.src='http://i.stack.imgur.com/1CqWf.jpg'

    function start(){
        // draw a triangle path
        ctx.beginPath();
        ctx.moveTo(75,50);
        ctx.lineTo(125,100);
        ctx.lineTo(25,100);
        ctx.lineTo(75,50);

        // clip future drawings to appear only in the triangle
        ctx.clip();

        // draw an image
        ctx.drawImage(img,0,0);
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=150 height=150></canvas>
</body>
</html>
```

第5.5节：基本路径绘制命令概述：直线和曲线

路径

路径定义了一组可以在画布上可视绘制的直线和曲线。

路径不会自动绘制到画布上。但路径的直线和曲线可以使用可样式化的描边绘制到画布上。由直线和曲线创建的形状也可以用可样式化的填充进行填充。

路径的用途不仅限于在画布上绘制：

- 检测某个x,y坐标是否在路径形状内部。
- 定义一个裁剪区域，只有裁剪区域内的绘制内容才可见。裁剪区域外的任何绘制内容都不会被绘制（即透明）——类似于CSS的overflow属性。

基本路径绘制命令包括：

- beginPath
- moveTo
- lineTo
- arc
- quadraticCurveTo
- bezierCurveTo
- arcTo
- rect
- closePath

基本绘图命令的描述：

beginPath

```
context.beginPath()
```

开始组装一组新的路径命令，同时丢弃之前组装的任何路径。

丢弃是一个重要且常被忽视的点。如果不开始一条新路径，任何之前发出的路径命令将会自动被重绘。

它还将绘图“笔”移动到画布的左上角原点（即坐标[0,0]）。

moveTo

```
context.moveTo(startX, startY)
```

将当前绘图笔位置移动到坐标[startX, startY]。

默认情况下，所有路径绘制都是连接在一起的。因此，一条线或曲线的终点就是下一条线或曲线的起点。这可能导致绘制出连接两个相邻图形的意外线条。
context.moveTo命令基本上是“提起绘图笔”，并将其放置在新的坐标位置，从而避免自动绘制连接线。

lineTo

Section 5.5: Overview of the basic path drawing commands: lines and curves

Path

A path defines a set of lines and curves which can be visibly drawn on the Canvas.

A path is not automatically drawn on the Canvas. But the path's lines & curves can be drawn onto the Canvas using a styleable stroke. And the shape created by the lines and curves can also be filled with a styleable fill.

Paths have uses beyond drawing on the Canvas:

- Hit testing if an x,y coordinate is inside the path shape.
- Defining a clipping region where only drawings inside the clipping region will be visible. Any drawings outside the clipping region will not be drawn (==transparent) -- similar to CSS overflow.

The basic path drawing commands are:

- beginPath
- moveTo
- lineTo
- arc
- quadraticCurveTo
- bezierCurveTo
- arcTo
- rect
- closePath

Description of the basic drawing commands:

beginPath

```
context.beginPath()
```

Begins assembling a new set of path commands and also discards any previously assembled path.

The discarding is an important and often overlooked point. If you don't begin a new path, any previously issued path commands will automatically be redrawn.

It also moves the drawing "pen" to the top-left origin of the canvas (==coordinate[0,0]).

moveTo

```
context.moveTo(startX, startY)
```

Moves the current pen location to the coordinate [startX,startY].

By default all path drawings are connected together. So the ending point of one line or curve is the starting point of the next line or curve. This can cause an unexpected line to be drawn connecting two adjacent drawings. The `context.moveTo` command basically "picks up the drawing pen" and places it at a new coordinate so the automatic connecting line is not drawn.

lineTo

```
context.lineTo(endX, endY)
```

从当前绘图笔位置绘制一条线段到坐标[endX, endY]你可以组合多个.lineTo命令来

绘制折线。例如，你可以组合3条线段形成一个三角形。

arc

```
context.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle)
```

给定中心点、半径以及起始和结束角度，绘制一个圆弧。角度以弧度表示。
要将角度转换为弧度，可以使用以下公式：弧度 = 角度 * Math.PI / 180;。

角度0从圆弧中心直接向右。要绘制完整的圆，可以使 endingAngle =
startingAngle + 360度（360度 == Math.PI2*）*：`*context.arc(10,10,20,0,Math.PI*2);

默认情况下，圆弧顺时针绘制。一个可选的 [true|false] 参数指示圆弧逆时针绘制：context.arc(10,10,20,0,Math.PI*2,true)

quadraticCurveTo

```
context.quadraticCurveTo(controlX, controlY, endingX, endingY)
```

从当前画笔位置开始绘制一条二次曲线到指定的结束坐标。另一个给定的控制坐标决定曲线的形状（弯曲度）。

bezierCurveTo

```
context.bezierCurveTo(control1X, control1Y, control2X, control2Y, endingX, endingY)
```

从当前画笔位置开始绘制一条三次贝塞尔曲线到指定的结束坐标。另两个给定的控制坐标决定曲线的形状（弯曲度
）。

arcTo

```
context.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
```

绘制一个给定半径的圆弧。该圆弧顺时针绘制，位于当前画笔位置与给定的两个点：点1和点2所形成的扇形内。

自动绘制一条连接当前画笔位置和圆弧起点的直线，位于圆弧之前。

rect

```
context.rect(leftX, topY, width, height)
```

根据左上角坐标及宽度和高度绘制一个矩形。

context.rect 是一个独特的绘图命令，因为它添加的是不相连的矩形。这些不相连的矩形不会自动用线连接。

closePath

```
context.lineTo(endX, endY)
```

Draws a line segment from the current pen location to coordinate [endX,endY]

You can assemble multiple `.lineTo` commands to draw a polyline. For example, you could assemble 3 line segments to form a triangle.

arc

```
context.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle)
```

Draws a circular arc given a centerpoint, radius and starting & ending angles. The angles are expressed as radians. To convert degrees to radians you can use this formula: `radians = degrees * Math.PI / 180;`.

Angle 0 faces directly rightward from the center of the arc. To draw a complete circle you can make endingAngle = startingAngle + 360 degrees (360 degrees == Math.PI2*):* `*context.arc(10,10,20,0,Math.PI*2);

By default, the arc is drawn clockwise, An optional [true|false] parameter instructs the arc to be drawn counter-clockwise: `context.arc(10,10,20,0,Math.PI*2,true)`

quadraticCurveTo

```
context.quadraticCurveTo(controlX, controlY, endingX, endingY)
```

Draws a quadratic curve starting at the current pen location to a given ending coordinate. Another given control coordinate determines the shape (curviness) of the curve.

bezierCurveTo

```
context.bezierCurveTo(control1X, control1Y, control2X, control2Y, endingX, endingY)
```

Draws a cubic Bezier curve starting at the current pen location to a given ending coordinate. Another 2 given control coordinates determine the shape (curviness) of the curve.

arcTo

```
context.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
```

Draws a circular arc with a given radius. The arc is drawn clockwise inside the wedge formed by the current pen location and given two points: Point1 & Point2.

A line connecting the current pen location and the start of the arc is automatically drawn preceding the arc.

rect

```
context.rect(leftX, topY, width, height)
```

Draws a rectangle given a top-left corner and a width & height.

The `context.rect` is a unique drawing command because it adds disconnected rectangles. These disconnected rectangles are not automatically connected by lines.

closePath

```
context.closePath()
```

绘制一条从当前画笔位置回到路径起点的线。

例如，如果你绘制了两条线形成三角形的两条边，closePath 会通过绘制第三条边，将三角形"闭合"，即从第二条边的终点回到第一条边的起点。

该命令的名称常常导致误解。context.closePath 并不是 context.beginPath 的结束符。再次强调，closePath 命令是绘制一条线——它并不"关闭"一个 beginPath。

第5.6节：lineTo（路径命令）

```
context.lineTo(endX, endY)
```

从当前画笔位置绘制一条线段到坐标[endX,endY]

```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 参数
    var startX=25;
    var startY=20;
    var endX=125;
    var endY=20;

    // 使用"moveTo"和"lineTo"命令绘制一条线段
   ctx.beginPath();
ctx.moveTo(startX,startY);
ctx.lineTo(endX,endY);
ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
```

```
context.closePath()
```

Draws a line from the current pen location back to the beginning path coordinate.

For example, if you draw 2 lines forming 2 legs of a triangle, closePath will "close" the triangle by drawing the third leg of the triangle from the 2nd leg's endpoint back to the first leg's starting point.

This command's name often causes it to be misunderstood. context.closePath is NOT an ending delimiter to context.beginPath. Again, the closePath command draws a line -- it does not "close" a beginPath.

Section 5.6: lineTo (a path command)

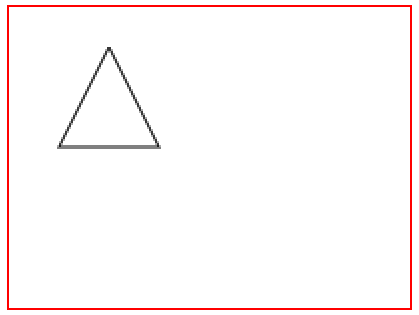
```
context.lineTo(endX, endY)
```

Draws a line segment from the current pen location to coordinate [endX,endY]

```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and it's context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // arguments
    var startX=25;
    var startY=20;
    var endX=125;
    var endY=20;

    // Draw a single line segment drawn using "moveTo" and "lineTo" commands
    ctx.beginPath();
    ctx.moveTo(startX,startY);
    ctx.lineTo(endX,endY);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
```

```
</html>
```

你可以组合多个 .lineTo 命令来绘制折线。例如，你可以组合3条线段形成一个三角形。



```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 参数
    var topVertexX=50;
    var topVertexY=20;
    var rightVertexX=75;
    var rightVertexY=70;
    var leftVertexX=25;
    var leftVertexY=70;

    // 使用
    //     "moveTo" 和多个 "lineTo" 命令绘制的一组三角形线段
ctx.beginPath();
ctx.moveTo(topVertexX,topVertexY);
    ctx.lineTo(rightVertexX,rightVertexY);
    ctx.lineTo(leftVertexX,leftVertexY);
    ctx.lineTo(topVertexX,topVertexY);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

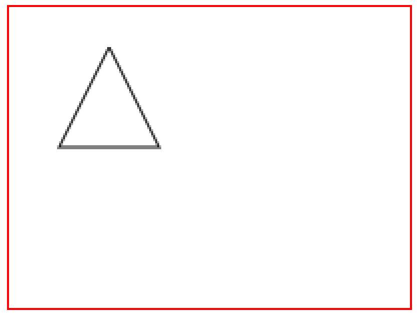
第5.7节：arc（路径命令）

```
context.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle)
```

给定中心点、半径以及起始和结束角度，绘制一个圆弧。角度以弧度表示。

```
</html>
```

You can assemble multiple .lineTo commands to draw a polyline. For example, you could assemble 3 line segments to form a triangle.



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and it's context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // arguments
    var topVertexX=50;
    var topVertexY=20;
    var rightVertexX=75;
    var rightVertexY=70;
    var leftVertexX=25;
    var leftVertexY=70;

    // A set of line segments drawn to form a triangle using
    //     "moveTo" and multiple "lineTo" commands
    ctx.beginPath();
    ctx.moveTo(topVertexX,topVertexY);
    ctx.lineTo(rightVertexX,rightVertexY);
    ctx.lineTo(leftVertexX,leftVertexY);
    ctx.lineTo(topVertexX,topVertexY);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

Section 5.7: arc (a path command)

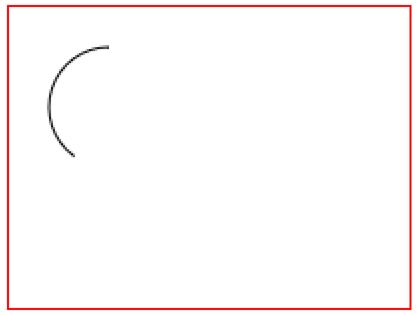
```
context.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle)
```

Draws a circular arc given a centerpoint, radius and starting & ending angles. The angles are expressed as radians.

要将角度转换为弧度，可以使用以下公式：弧度 = 角度 * Math.PI / 180;。

角度0从弧的中心直接指向正右方。

默认情况下，圆弧顺时针绘制。一个可选的 [true|false] 参数指示圆弧逆时针绘制：context.arc(10,10,20,0,Math.PI*2,true)



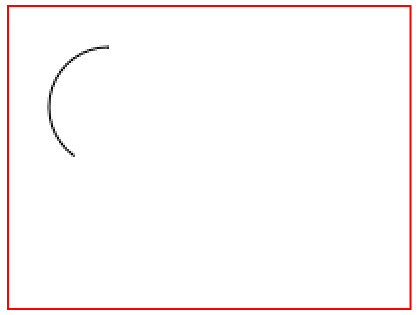
```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 参数
    var centerX=50;
    var centerY=50;
    var radius=30;
    var startingRadianAngle=Math.PI*2*;  // 起始于90度 == centerY+radius
    var endingRadianAngle=Math.PI*2*.75;  // 结束于270度（即弧度的PI*2*.75）

    // 使用"arc"命令绘制的部分圆（即弧）
ctx.beginPath();
ctx.arc(centerX, centerY, radius,  startingRadianAngle, endingRadianAngle);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

要绘制一个完整的圆，可以将 endingAngle 设为 startingAngle + 360 度（360 度 == Math.PI2）。

To convert degrees to radians you can use this formula: `radians = degrees * Math.PI / 180;`.

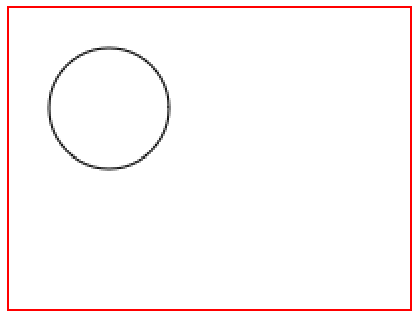
Angle 0 faces directly rightward from the center of the arc.

By default, the arc is drawn clockwise, An optional [true|false] parameter instructs the arc to be drawn counter-clockwise: `context.arc(10,10,20,0,Math.PI*2,true)`



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and its context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // arguments
    var centerX=50;
    var centerY=50;
    var radius=30;
    var startingRadianAngle=Math.PI*2*;  // start at 90 degrees == centerY+radius
    var endingRadianAngle=Math.PI*2*.75;  // end at 270 degrees (==PI*2*.75 in radians)

    // A partial circle (i.e. arc) drawn using the "arc" command
    ctx.beginPath();
    ctx.arc(centerX, centerY, radius,  startingRadianAngle, endingRadianAngle);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

To draw a complete circle you can make endingAngle = startingAngle + 360 degrees (360 degrees == Math.PI2).

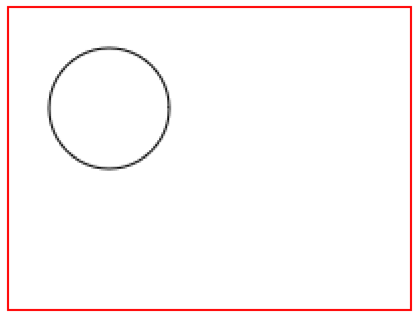


```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 参数
    var centerX=50;
    var centerY=50;
    var radius=30;
    var startingRadianAngle=0;       // 从0度开始
    var endingRadianAngle=Math.PI*2; // 结束于360度（即弧度的PI*2）

    // 使用"arc"命令绘制的完整圆
ctx.beginPath();
ctx.arc(centerX, centerY, radius,  startingRadianAngle, endingRadianAngle);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

第5.8节：quadraticCurveTo（路径命令）

```
context.quadraticCurveTo(controlX, controlY, endingX, endingY)
```

从当前画笔位置开始绘制一条二次曲线到指定的结束坐标。另一个给定的控制坐标决定曲线的形状（弯曲度）。

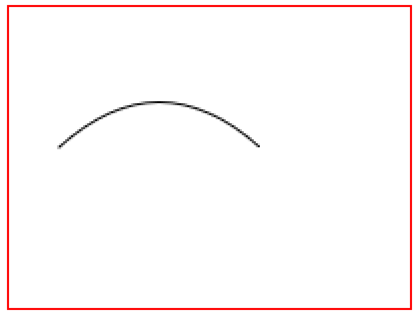


```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and its context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // arguments
    var centerX=50;
    var centerY=50;
    var radius=30;
    var startingRadianAngle=0;       // start at 0 degrees
    var endingRadianAngle=Math.PI*2; // end at 360 degrees (==PI*2 in radians)

    // A complete circle drawn using the "arc" command
    ctx.beginPath();
    ctx.arc(centerX, centerY, radius,  startingRadianAngle, endingRadianAngle);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

Section 5.8: quadraticCurveTo (a path command)

```
context.quadraticCurveTo(controlX, controlY, endingX, endingY)
```

Draws a quadratic curve starting at the current pen location to a given ending coordinate. Another given control coordinate determines the shape (curviness) of the curve.

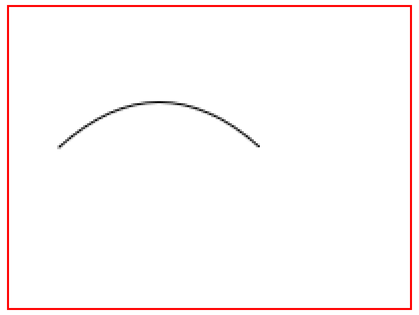


```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 参数
    var startX=25;
    var startY=70;
    var controlX=75;
    var controlY=25;
    var endX=125;
    var endY=70;

    // 使用"moveTo"和"quadraticCurveTo"命令绘制的二次曲线
   ctx.beginPath();
ctx.moveTo(startX,startY);
ctx.quadraticCurveTo(controlX,controlY,endX,endY);
    ctx.stroke();


}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

第5.9节：bezierCurveTo（路径命令）

```
context.bezierCurveTo(control1X, control1Y, control2X, control2Y, endingX, endingY)
```

从当前画笔位置开始绘制一条三次贝塞尔曲线到指定的结束坐标。另两个给定的控制坐标决定曲线的形状（弯曲度）。

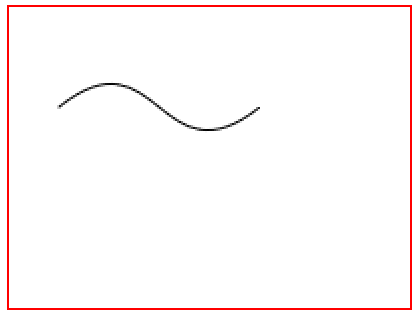


```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and it's context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // arguments
    var startX=25;
    var startY=70;
    var controlX=75;
    var controlY=25;
    var endX=125;
    var endY=70;

    // A quadratic curve drawn using "moveTo" and "quadraticCurveTo" commands
    ctx.beginPath();
    ctx.moveTo(startX,startY);
    ctx.quadraticCurveTo(controlX,controlY,endX,endY);
    ctx.stroke();


}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

Section 5.9: bezierCurveTo (a path command)

```
context.bezierCurveTo(control1X, control1Y, control2X, control2Y, endingX, endingY)
```

Draws a cubic Bezier curve starting at the current pen location to a given ending coordinate. Another 2 given control coordinates determine the shape (curviness) of the curve.



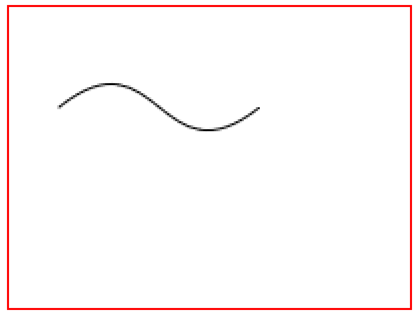
```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 参数
    var startX=25;
    var startY=50;
    var controlX1=75;
    var controlY1=10;
    var controlX2=75;
    var controlY2=90;
    var endX=125;
    var endY=50;

    // 使用"moveTo"和"bezierCurveTo"命令绘制三次贝塞尔曲线
    ctx.beginPath();
ctx.moveTo(startX,startY);
ctx.bezierCurveTo(controlX1,controlY1,controlX2,controlY2,endX,endY);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

第5.10节：arcTo（路径命令）

```
context.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
```

绘制一个给定半径的圆弧。该圆弧顺时针绘制，位于当前画笔位置与给定的两个点：点1和点2所形成的扇形内。

自动绘制一条连接当前画笔位置和圆弧起点的直线，位于圆弧之前。



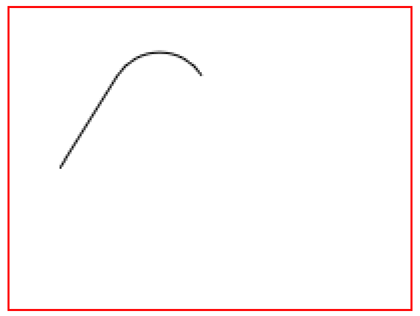
```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and it's context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // arguments
    var startX=25;
    var startY=50;
    var controlX1=75;
    var controlY1=10;
    var controlX2=75;
    var controlY2=90;
    var endX=125;
    var endY=50;

    // A cubic bezier curve drawn using "moveTo" and "bezierCurveTo" commands
    ctx.beginPath();
    ctx.moveTo(startX,startY);
    ctx.bezierCurveTo(controlX1,controlY1,controlX2,controlY2,endX,endY);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

Section 5.10: arcTo (a path command)

```
context.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
```

Draws a circular arc with a given radius. The arc is drawn clockwise inside the wedge formed by the current pen location and given two points: Point1 & Point2.

A line connecting the current pen location and the start of the arc is automatically drawn preceding the arc.

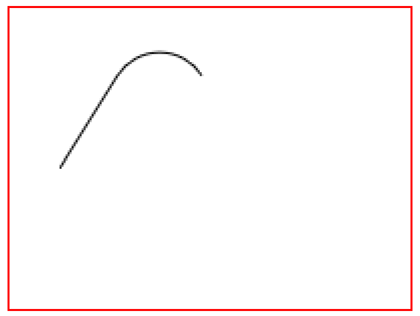


```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 参数
    var pointX0=25;
    var pointY0=80;
    var pointX1=75;
    var pointY1=0;
    var pointX2=125;
    var pointY2=80;
    var radius=25;

    // 使用“arcTo”命令绘制的圆弧。线条会自动绘制。
ctx.beginPath();
ctx.moveTo(pointX0,pointY0);
    ctx.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

第5.11节：rect（路径命令）

```
context.rect(leftX, topY, width, height)
```

根据左上角坐标及宽度和高度绘制一个矩形。

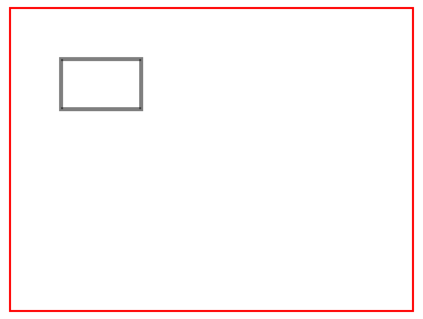


```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and it's context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // arguments
    var pointX0=25;
    var pointY0=80;
    var pointX1=75;
    var pointY1=0;
    var pointX2=125;
    var pointY2=80;
    var radius=25;

    // A circular arc drawn using the "arcTo" command. The line is automatically drawn.
    ctx.beginPath();
    ctx.moveTo(pointX0,pointY0);
    ctx.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

Section 5.11: rect (a path command)

```
context.rect(leftX, topY, width, height)
```

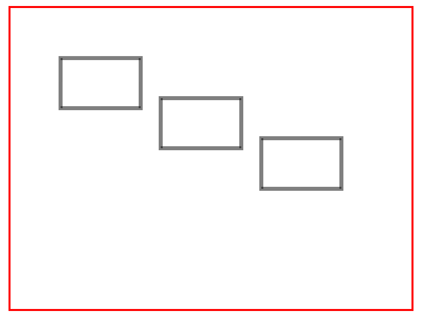
Draws a rectangle given a top-left corner and a width & height.



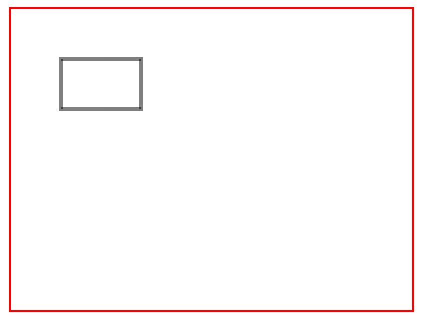
```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 参数
    var leftX=25;
    var topY=25;
    var width=40;
    var height=25;

    // 使用 "rect" 命令绘制的矩形。
ctx.beginPath();
ctx.rect(leftX, topY, width, height);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

context.rect 是一个独特的绘图命令，因为它添加的是不相连的矩形。

这些不相连的矩形不会自动用线连接起来。



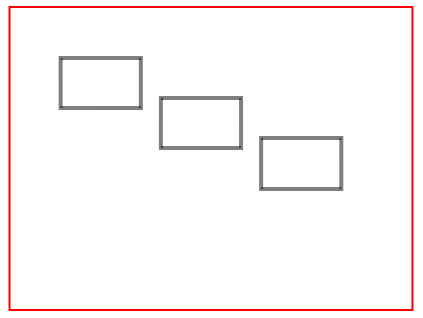
```
<!doctype html>
<html>
<head>
```



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and it's context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // arguments
    var leftX=25;
    var topY=25;
    var width=40;
    var height=25;

    // A rectangle drawn using the "rect" command.
    ctx.beginPath();
    ctx.rect(leftX, topY, width, height);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

The `context.rect` is a unique drawing command because it adds disconnected rectangles.

These disconnected rectangles are not automatically connected by lines.



```
<!doctype html>
<html>
<head>
```

```
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 参数
    var leftX=25;
    var topY=25;
    var width=40;
    var height=25;

    // 使用 "rect" 命令绘制的多个矩形。
ctx.beginPath();
ctx.rect(leftX, topY, width, height);
    ctx.rect(leftX+50, topY+20, width, height);
    ctx.rect(leftX+100, topY+40, width, height);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

第5.12节：closePath（路径命令）

```
context.closePath()
```

绘制一条从当前画笔位置回到路径起点的线。

例如，如果你绘制了两条线形成三角形的两条边，closePath 会通过从第二条边的终点回到第一条边的起点，绘制第三条边，从而“闭合”三角形。

误解解析！

该命令的名称常常导致误解。

context.closePath 不是 context.beginPath 的结束分隔符。

再次说明，closePath 命令绘制一条线——它并不“关闭”一个 beginPath。

此示例绘制了三角形的两条边，并使用 closePath 来完成（三角形闭合？）通过绘制第三条边。实际上，closePath 所做的是从第二条边的终点绘制一条线回到第一条边的起点。

```
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and it's context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // arguments
    var leftX=25;
    var topY=25;
    var width=40;
    var height=25;

    // Multiple rectangles drawn using the "rect" command.
    ctx.beginPath();
    ctx.rect(leftX, topY, width, height);
    ctx.rect(leftX+50, topY+20, width, height);
    ctx.rect(leftX+100, topY+40, width, height);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

Section 5.12: closePath (a path command)

```
context.closePath()
```

Draws a line from the current pen location back to the beginning path coordinate.

For example, if you draw 2 lines forming 2 legs of a triangle, closePath will "close" the triangle by drawing the third leg of the triangle from the 2nd leg's endpoint back to the first leg's starting point.

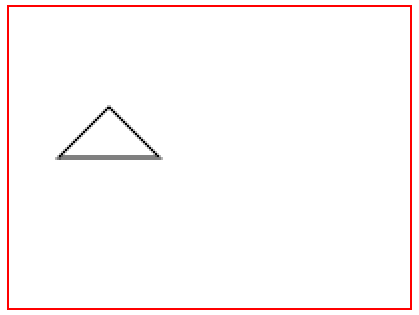
A Misconception explained!

This command's name often causes it to be misunderstood.

`context.closePath` is NOT an ending delimiter to `context.beginPath`.

Again, the closePath command draws a line -- it does not "close" a beginPath.

This example draws 2 legs of a triangle and uses `closePath` to complete (close?!) the triangle by drawing the third leg. What `closePath` is actually doing is drawing a line from the second leg's endpoint back to the first leg's starting point.



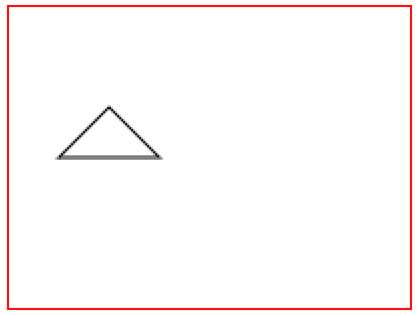
```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 参数
    var topVertexX=50;
    var topVertexY=50;
    var rightVertexX=75;
    var rightVertexY=75;
    var leftVertexX=25;
    var leftVertexY=75;

    // 使用
    //     "moveTo" 和多个 "lineTo" 命令绘制的一组三角形线段
ctx.beginPath();
ctx.moveTo(topVertexX,topVertexY);
    ctx.lineTo(rightVertexX,rightVertexY);
    ctx.lineTo(leftVertexX,leftVertexY);

    // closePath 绘制三角形的第三条边
ctx.closePath()

    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

第5.13节：beginPath（路径命令）

```
context.beginPath()
```

开始组装一组新的路径命令，同时丢弃之前组装的任何路径。

它还将绘图"笔"移动到画布的左上角原点（即坐标[0,0]）。



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and it's context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // arguments
    var topVertexX=50;
    var topVertexY=50;
    var rightVertexX=75;
    var rightVertexY=75;
    var leftVertexX=25;
    var leftVertexY=75;

    // A set of line segments drawn to form a triangle using
    //     "moveTo" and multiple "lineTo" commands
    ctx.beginPath();
    ctx.moveTo(topVertexX,topVertexY);
    ctx.lineTo(rightVertexX,rightVertexY);
    ctx.lineTo(leftVertexX,leftVertexY);

    // closePath draws the 3rd leg of the triangle
    ctx.closePath()

    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

Section 5.13: beginPath (a path command)

```
context.beginPath()
```

Begins assembling a new set of path commands and also discards any previously assembled path.

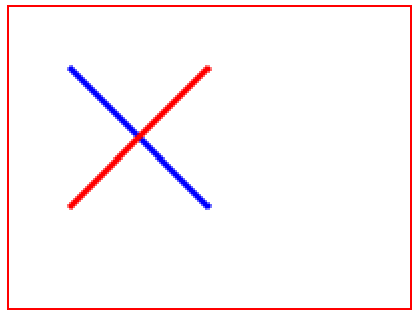
It also moves the drawing "pen" to the top-left origin of the canvas (==coordinate[0,0]).

虽然是可选的，但你应该始终以beginPath开始路径。丢弃之前路径是一个

重要且常被忽视的点。如果你不使用beginPath开始新路径，任何之前发出的路径命令将会自动被重新绘制。

这两个演示都试图用一条红色线和一条蓝色线绘制一个“X”。

第一个演示正确地使用了beginPath来开始第二条红色线。结果是“X”正确地同时有红色和蓝色的线条。



```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 绘制一条蓝线
ctx.beginPath();
    ctx.moveTo(30,30);
    ctx.lineTo(100,100);
    ctx.strokeStyle='blue';
    ctx.lineWidth=3;
ctx.stroke();

    // 绘制一条红线
ctx.beginPath();       // 重要：开始一条新路径！
ctx.moveTo(100,30);
    ctx.lineTo(30,100);
    ctx.strokeStyle='red';
    ctx.lineWidth=3;
ctx.stroke();

}); // 结束 window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

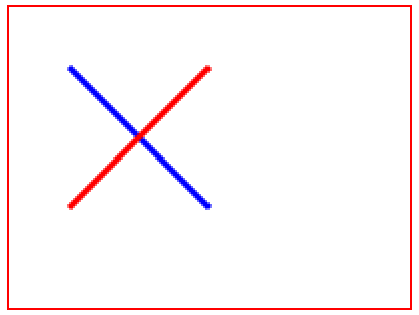
这个第二个演示错误地省略了第二次描边时的beginPath。结果是“X”错误地有了

Although optional, you should ALWAYS start a path with `beginPath`

The discarding is an important and often overlooked point. If you don't begin a new path with `beginPath`, any previously issued path commands will automatically be redrawn.

These 2 demos both attempt to draw an "X" with one red stroke and one blue stroke.

This first demo correctly uses `beginPath` to start it's second red stroke. The result is that the "X" correctly has both a red and a blue stroke.



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and it's context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // draw a blue line
    ctx.beginPath();
    ctx.moveTo(30,30);
    ctx.lineTo(100,100);
    ctx.strokeStyle='blue';
    ctx.lineWidth=3;
    ctx.stroke();

    // draw a red line
    ctx.beginPath();        // Important to begin a new path!
    ctx.moveTo(100,30);
    ctx.lineTo(30,100);
    ctx.strokeStyle='red';
    ctx.lineWidth=3;
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

This second demo incorrectly leaves out `beginPath` on the second stroke. The result is that the "X" incorrectly has

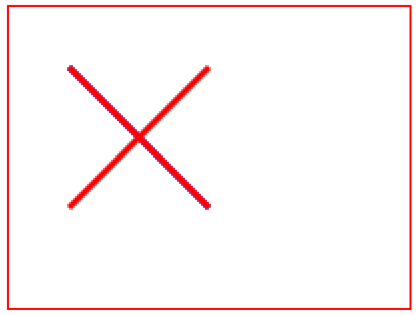
两个红色描边。

第二个stroke()绘制了第二个红色描边。

但是没有第二个beginPath，同一个第二个stroke()也错误地重绘了第一个描边。

由于第二个stroke()现在被设置为红色，第一个蓝色描边被错误着色的红色

描边覆盖了。



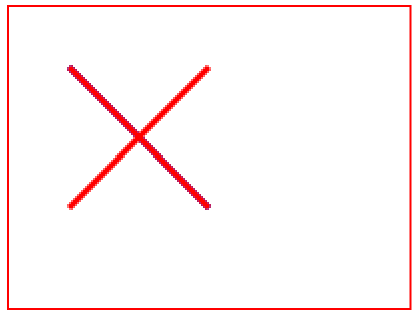
```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 获取画布元素及其上下文的引用
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 绘制一条蓝线
ctx.beginPath();
    ctx.moveTo(30,30);
    ctx.lineTo(100,100);
    ctx.strokeStyle='blue';
    ctx.lineWidth=3;
ctx.stroke();

    // 绘制一条红线
    // 注意：缺少必要的 'beginPath'！
ctx.moveTo(100,30);
    ctx.lineTo(30,100);
    ctx.strokeStyle='red';
    ctx.lineWidth=3;
ctx.stroke();

}); // 结束 window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

both red strokes.

The second stroke() is draws the second red stroke.

But without a second beginPath, that same second stroke() also incorrectly **redraws** the first stroke.

Since the second stroke() is now styled as red, the first blue stroke is **overwritten** by an incorrectly colored red stroke.



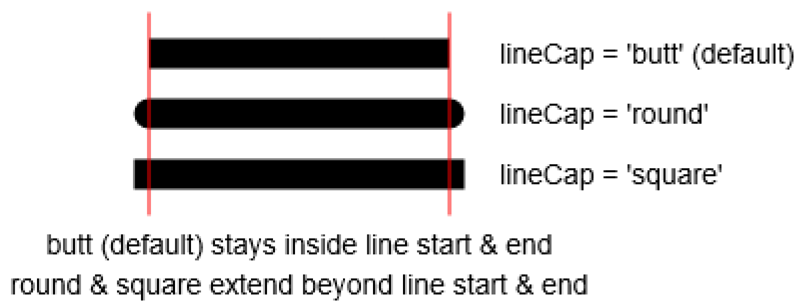
```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and it's context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // draw a blue line
    ctx.beginPath();
    ctx.moveTo(30,30);
    ctx.lineTo(100,100);
    ctx.strokeStyle='blue';
    ctx.lineWidth=3;
    ctx.stroke();

    // draw a red line
    // Note: The necessary 'beginPath' is missing!
    ctx.moveTo(100,30);
    ctx.lineTo(30,100);
    ctx.strokeStyle='red';
    ctx.lineWidth=3;
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

第5.14节：lineCap（路径样式属性）

```
context.lineCap=capStyle  // butt（默认），round, square
```

设置线条起点和终点的端点样式。

- **butt**，默认的 lineCap 样式，显示方形端点，不会超出线条的起始和结束点。
- round，显示圆形端点，会超出线条的起始和结束点。
- square，显示方形端点，会超出线条的起始和结束点。


lineCap = 'butt' (default)
lineCap = 'round'
lineCap = 'square'
butt (default) stays inside line start & end
round & square extend beyond line start & end

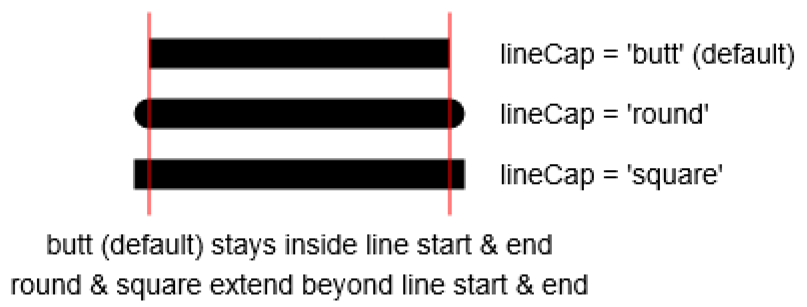
```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    ctx.lineWidth=15;

    // lineCap 默认值：butt
ctx.lineCap='butt';
    drawLine(50,40,200,40);

    // lineCap：round
   ctx.lineCap='round';
    drawLine(50,70,200,70);

     // lineCap：square
ctx.lineCap='square';
    drawLine(50,100,200,100);

    // 绘制线条的实用函数
    function drawLine(startX,startY,endX,endY){
        ctx.beginPath();
        ctx.moveTo(startX,startY);
        ctx.lineTo(endX,endY);
        ctx.stroke();
    }

    // 仅用于演示，
```

Section 5.14: lineCap (a path styling attribute)

```
context.lineCap=capStyle  // butt (default), round, square
```

Sets the cap style of line starting points and ending points.

- **butt**, the default lineCap style, shows squared caps that do not extend beyond the line's starting and ending points.
- **round**, shows rounded caps that extend beyond the line's starting and ending points.
- **square**, shows squared caps that extend beyond the line's starting and ending points.


lineCap = 'butt' (default)
lineCap = 'round'
lineCap = 'square'
butt (default) stays inside line start & end
round & square extend beyond line start & end

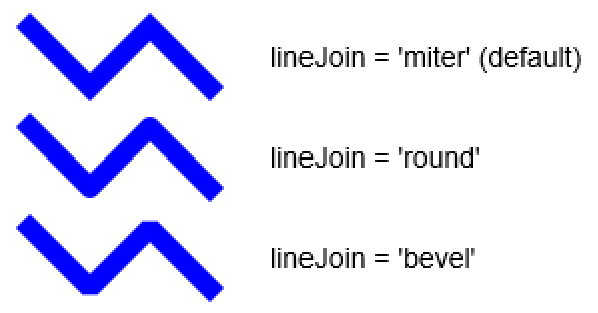
```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    ctx.lineWidth=15;

    // lineCap default: butt
    ctx.lineCap='butt';
    drawLine(50,40,200,40);

    // lineCap: round
    ctx.lineCap='round';
    drawLine(50,70,200,70);

    // lineCap: square
    ctx.lineCap='square';
    drawLine(50,100,200,100);

    // utility function to draw a line
    function drawLine(startX,startY,endX,endY){
        ctx.beginPath();
        ctx.moveTo(startX,startY);
        ctx.lineTo(endX,endY);
        ctx.stroke();
    }

    // For demo only,
    // 标尺显示哪些 lineCaps 超出端点
ctx.lineWidth=1;
   ctx.strokeStyle='red';
   drawLine(50,20,50,120);
   drawLine(200,20,200,120);

}); // window.onload 结束
</script>
</head>
<body>
    <canvas id="canvas" width=300 height=200></canvas>
</body>
</html>
```

第5.15节：lineJoin（路径样式属性）

```
context.lineJoin=joinStyle  // miter（默认），round，bevel
```

设置用于连接相邻线段的样式。

- **miter，默认值，使用尖角连接线段。**
- **round，使用圆角连接线段。**
- **斜角连接，使用钝角连接线段。**



```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    ctx.lineWidth=15;

    // lineJoin：斜接（默认）
  ctx.lineJoin='miter';
    drawPolyline(50,30);

      // lineJoin：圆角
  ctx.lineJoin='round';
    drawPolyline(50,80);

      // lineJoin：斜角
```

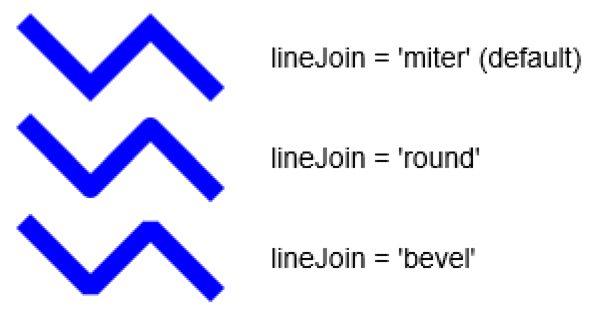
```
    // Rulers to show which lineCaps extend beyond endpoints
    ctx.lineWidth=1;
    ctx.strokeStyle='red';
    drawLine(50,20,50,120);
    drawLine(200,20,200,120);

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=300 height=200></canvas>
</body>
</html>
```

Section 5.15: lineJoin (a path styling attribute)

```
context.lineJoin=joinStyle  // miter (default), round, bevel
```

Sets the style used to connect adjoining line segments.

- **miter**, the default, joins line segments with a sharp joint.
- **round**, joins line segments with a rounded joint.
- **bevel**, joins line segments with a blunted joint.



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    ctx.lineWidth=15;

    // lineJoin: miter (default)
    ctx.lineJoin='miter';
    drawPolyline(50,30);

    // lineJoin: round
    ctx.lineJoin='round';
    drawPolyline(50,80);

    // lineJoin: bevel
ctx.lineJoin='bevel';
    drawPolyline(50,130);

    // 绘制折线的工具函数
    function drawPolyline(x,y){
        ctx.beginPath();
        ctx.moveTo(x,y);
        ctx.lineTo(x+30,y+30);
        ctx.lineTo(x+60,y);
        ctx.lineTo(x+90,y+30);
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=300 height=200></canvas>
</body>
</html>
```

第5.16节：strokeStyle（路径样式属性）

```
context.strokeStyle=color
```

设置用于描边当前路径轮廓的颜色。

这些是颜色选项（必须加引号）：

- **一个CSS命名颜色**，例如`context.strokeStyle='red'`
- **一个十六进制颜色**，例如`context.strokeStyle='#FF0000'`
- **一个RGB颜色**，例如`context.strokeStyle='rgb(red,green,blue)'` 其中red、green和blue是0-255的整数，表示每个颜色分量的强度。
- 一个HSL颜色，例如context.strokeStyle='hsl(hue,saturation,lightness)' 其中hue是颜色轮上的0-360整数，saturation和lightness是百分比（0-100%），表示每个分量的强度。
- 一个HSLA颜色，例如context.strokeStyle='hsl(hue,saturation,lightness,alpha)' 其中hue是颜色轮上的0-360整数，saturation和lightness是百分比（0-100%），表示每个分量的强度，alpha是0.00-1.00的十进制数，表示不透明度。

你也可以指定这些颜色选项（这些选项是由context创建的对象）：

- **一个线性渐变 是用context.createLinearGradient创建的线性渐变对象**
- **一个径向渐变 是用context.createRadialGradient创建的径向渐变对象**
- **一个图案 是用context.createPattern创建的图案对象**

```
    ctx.lineJoin='bevel';
    drawPolyline(50,130);

    // utility to draw polyline
    function drawPolyline(x,y){
        ctx.beginPath();
        ctx.moveTo(x,y);
        ctx.lineTo(x+30,y+30);
        ctx.lineTo(x+60,y);
        ctx.lineTo(x+90,y+30);
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=300 height=200></canvas>
</body>
</html>
```

Section 5.16: strokeStyle (a path styling attribute)

```
context.strokeStyle=color
```

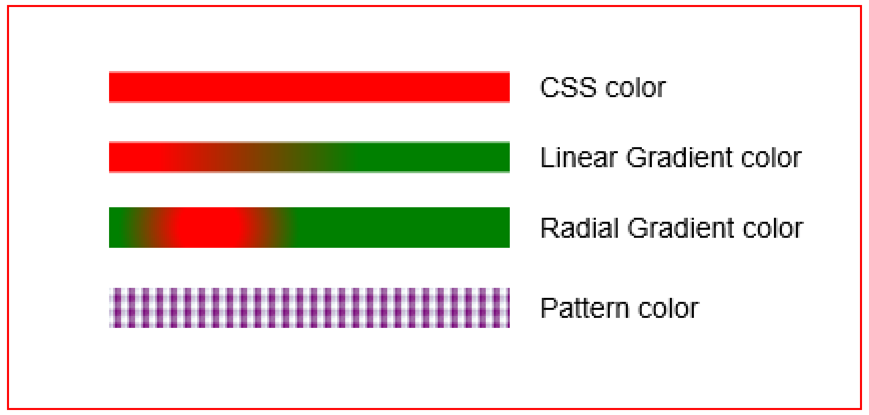
Sets the color that will be used to stroke the outline of the current path.

These are `color` options (these must be quoted):

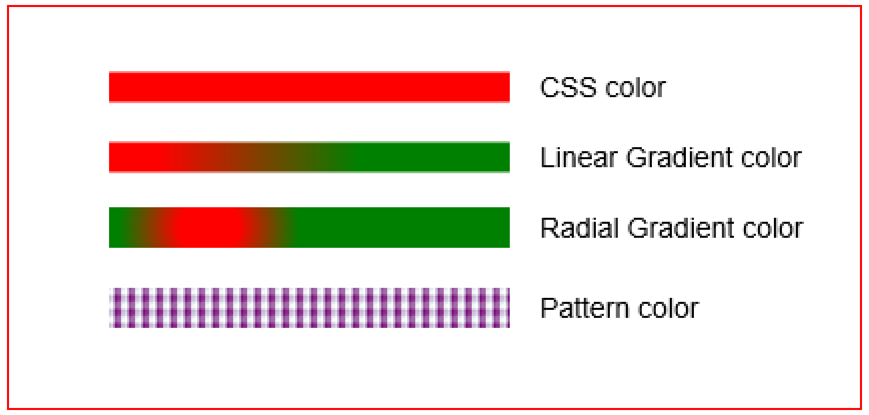
- **A CSS named color**, for example `context.strokeStyle='red'`
- **A hex color**, for example `context.strokeStyle='#FF0000'`
- **An RGB color**, for example `context.strokeStyle='rgb(red,green,blue)'` where red, green & blue are integers 0-255 indicating the strength of each component color.
- **An HSL color**, for example `context.strokeStyle='hsl(hue,saturation,lightness)'` where hue is an integer 0-360 on the color wheel and saturation & lightness are percentages (0-100%) indicating the strength of each component.
- **An HSLA color**, for example `context.strokeStyle='hsl(hue,saturation,lightness,alpha)'` where hue is an integer 0-360 on the color wheel and saturation & lightness are percentages (0-100%) indicating the strength of each component and alpha is a decimal value 0.00-1.00 indicating the opacity.

You can also specify these color options (these options are objects created by the context):

- **A linear gradient** which is a linear gradient object created with `context.createLinearGradient`
- **A radial gradient** which is a radial gradient object created with `context.createRadialGradient`
- **A pattern** which is a pattern object created with `context.createPattern`



```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    ctx.lineWidth=15;

    // 使用CSS颜色进行描边：命名颜色、RGB、HSL等
ctx.strokeStyle='red';
    drawLine(50,40,250,40);

    // 使用线性渐变描边
    var gradient = ctx.createLinearGradient(75,75,175,75);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.strokeStyle=gradient;
drawLine(50,75,250,75);

    // 使用径向渐变描边
    var gradient = ctx.createRadialGradient(100,110,15,100,110,45);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.strokeStyle=gradient;
ctx.lineWidth=20;
    drawLine(50,110,250,110);

    // 使用图案描边
    var patternImage=new Image();
patternImage.onload=function(){
        var pattern = ctx.createPattern(patternImage,'repeat');
        ctx.strokeStyle=pattern;
        drawLine(50,150,250,150);
    }
patternImage.src='https://dl.dropboxusercontent.com/u/139992952/stackoverflow/BooMu1.png';

    // 仅用于演示，按每笔画绘制标签
ctx.textBaseline='居中';
    ctx.font='14px Arial';
    ctx.fillText('CSS 颜色',265,40);
    ctx.fillText('线性渐变颜色',265,75);
```



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    ctx.lineWidth=15;

    // stroke using a CSS color: named, RGB, HSL, etc
    ctx.strokeStyle='red';
    drawLine(50,40,250,40);

    // stroke using a linear gradient
    var gradient = ctx.createLinearGradient(75,75,175,75);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.strokeStyle=gradient;
    drawLine(50,75,250,75);

    // stroke using a radial gradient
    var gradient = ctx.createRadialGradient(100,110,15,100,110,45);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.strokeStyle=gradient;
    ctx.lineWidth=20;
    drawLine(50,110,250,110);

    // stroke using a pattern
    var patternImage=new Image();
    patternImage.onload=function(){
        var pattern = ctx.createPattern(patternImage,'repeat');
        ctx.strokeStyle=pattern;
        drawLine(50,150,250,150);
    }
    patternImage.src='https://dl.dropboxusercontent.com/u/139992952/stackoverflow/BooMu1.png';

    // for demo only, draw labels by each stroke
    ctx.textBaseline='middle';
    ctx.font='14px arial';
    ctx.fillText('CSS color',265,40);
    ctx.fillText('Linear Gradient color',265,75);
ctx.fillText('径向渐变颜色',265,110);
    ctx.fillText('图案颜色',265,150);

    // 绘制线条的工具函数
    function drawLine(startX,startY,endX,endY){
        ctx.beginPath();
        ctx.moveTo(startX,startY);
        ctx.lineTo(endX,endY);
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=425 height=200></canvas>
</body>
</html>
```

第5.17节：fillStyle（路径样式属性）

```
context.fillStyle=color
```

设置用于填充当前路径内部的颜色。

这些是颜色选项（必须加引号）：

- **一个CSS命名颜色，例如context.fillStyle='红色'**
- **一个十六进制颜色，例如context.fillStyle='#FF0000'**
- **一个RGB颜色，例如context.fillStyle='rgb(红,绿,蓝)'其中红、绿和蓝是**
  0-255的整数，表示每个颜色分量的强度。
- 一个HSL颜色，例如context.fillStyle='hsl(色相,饱和度,亮度)'其中色相是0-360的整数表示色轮上的位置，饱和度和亮度是百分比（0-100%），表示每个分量的强度。

- HSLA颜色，例如context.fillStyle='hsl(色相,饱和度,亮度,透明度)'其中色相是颜色轮上的整数，范围为0-360，饱和度和亮度是百分比（0-100%），表示各成分的强度，透明度是0.00-1.00之间的小数，表示不透明度。

你也可以指定这些颜色选项（这些选项是由context创建的对象）：

- **一个线性渐变 是用context.createLinearGradient创建的线性渐变对象**
- **一个径向渐变 是用context.createRadialGradient创建的径向渐变对象**
- **一个图案 是用context.createPattern创建的图案对象**

```
    ctx.fillText('Radial Gradient color',265,110);
    ctx.fillText('Pattern color',265,150);

    // utility to draw a line
    function drawLine(startX,startY,endX,endY){
        ctx.beginPath();
        ctx.moveTo(startX,startY);
        ctx.lineTo(endX,endY);
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=425 height=200></canvas>
</body>
</html>
```

Section 5.17: fillStyle (a path styling attribute)

```
context.fillStyle=color
```

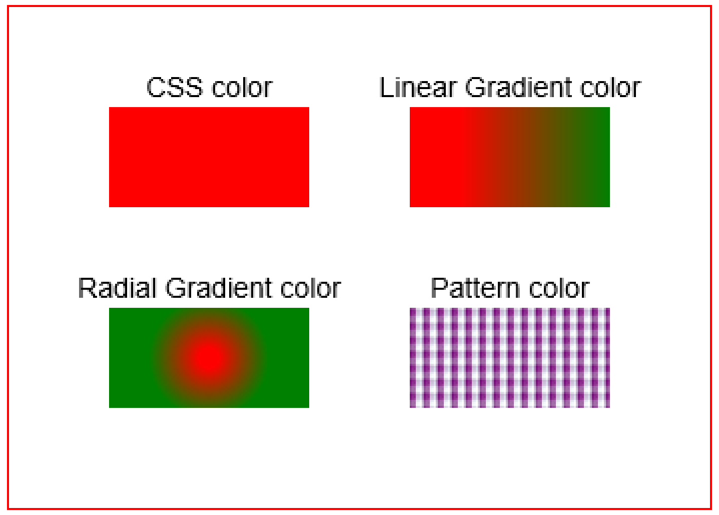
Sets the color that will be used to fill the interior of the current path.

These are color options (these must be quoted):

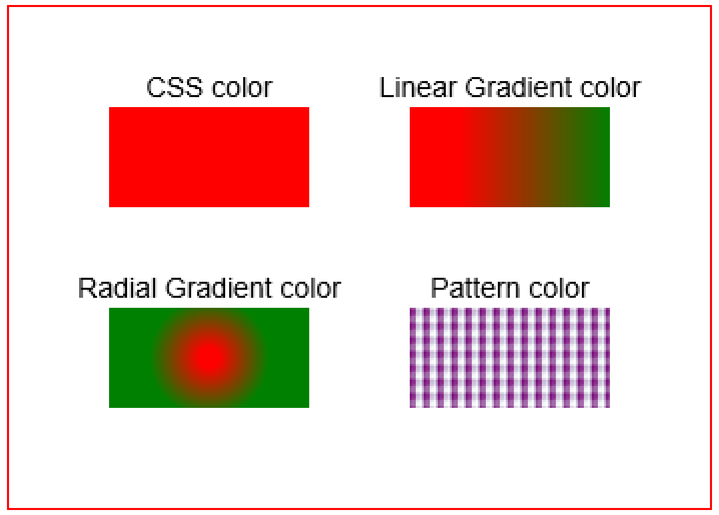
- **A CSS named color**, for example `context.fillStyle='red'`
- **A hex color**, for example `context.fillStyle='#FF0000'`
- **An RGB color**, for example `context.fillStyle='rgb(red,green,blue)'` where red, green & blue are integers 0-255 indicating the strength of each component color.
- **An HSL color**, for example `context.fillStyle='hsl(hue,saturation,lightness)'` where hue is an integer 0-360 on the color wheel and saturation & lightness are percentages (0-100%) indicating the strength of each component.
- **An HSLA color**, for example `context.fillStyle='hsl(hue,saturation,lightness,alpha)'` where hue is an integer 0-360 on the color wheel and saturation & lightness are percentages (0-100%) indicating the strength of each component and alpha is a decimal value 0.00-1.00 indicating the opacity.

You can also specify these color options (these options are objects created by the context):

- **A linear gradient** which is a linear gradient object created with `context.createLinearGradient`
- **A radial gradient** which is a radial gradient object created with `context.createRadialGradient`
- **A pattern** which is a pattern object created with `context.createPattern`



```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 使用CSS颜色进行描边：命名颜色、RGB、HSL等
ctx.fillStyle='red';
    ctx.fillRect(50,50,100,50);

    // 使用线性渐变描边
    var gradient = ctx.createLinearGradient(225,50,300,50);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.fillStyle=gradient;
ctx.fillRect(200,50,100,50);

    // 使用径向渐变描边
    var gradient = ctx.createRadialGradient(100,175,5,100,175,30);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.fillStyle=gradient;
ctx.fillRect(50,150,100,50);

    // 使用图案描边
    var patternImage=new Image();
patternImage.onload=function(){
        var pattern = ctx.createPattern(patternImage,'repeat');
        ctx.fillStyle=pattern;
ctx.fillRect(200,150,100,50);
    }
patternImage.src='http://i.stack.imgur.com/ixrWe.png';

    // 仅用于演示，按每笔画绘制标签
ctx.fillStyle='black';
    ctx.textAlign='center';
    ctx.textBaseline='middle';
```

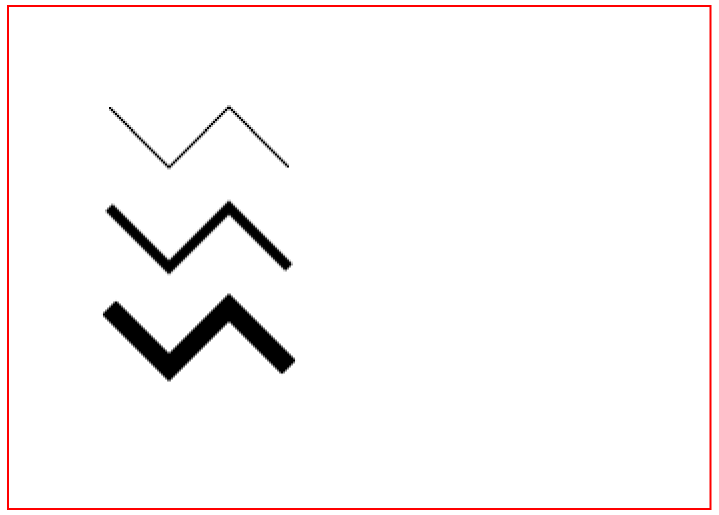


```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // stroke using a CSS color: named, RGB, HSL, etc
    ctx.fillStyle='red';
    ctx.fillRect(50,50,100,50);

    // stroke using a linear gradient
    var gradient = ctx.createLinearGradient(225,50,300,50);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.fillStyle=gradient;
    ctx.fillRect(200,50,100,50);

    // stroke using a radial gradient
    var gradient = ctx.createRadialGradient(100,175,5,100,175,30);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.fillStyle=gradient;
    ctx.fillRect(50,150,100,50);

    // stroke using a pattern
    var patternImage=new Image();
    patternImage.onload=function(){
        var pattern = ctx.createPattern(patternImage,'repeat');
        ctx.fillStyle=pattern;
        ctx.fillRect(200,150,100,50);
    }
    patternImage.src='http://i.stack.imgur.com/ixrWe.png';

    // for demo only, draw labels by each stroke
    ctx.fillStyle='black';
    ctx.textAlign='center';
    ctx.textBaseline='middle';
ctx.font='14px arial';
    ctx.fillText('CSS 颜色',100,40);
    ctx.fillText('线性渐变颜色',250,40);
    ctx.fillText('径向渐变颜色',100,140);
    ctx.fillText('图案颜色',250,140);

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=350 height=250></canvas>
</body>
</html>
```

第5.18节：lineWidth（路径样式属性）

```
context.lineWidth=lineWidth
```

设置用于描边路径轮廓的线宽



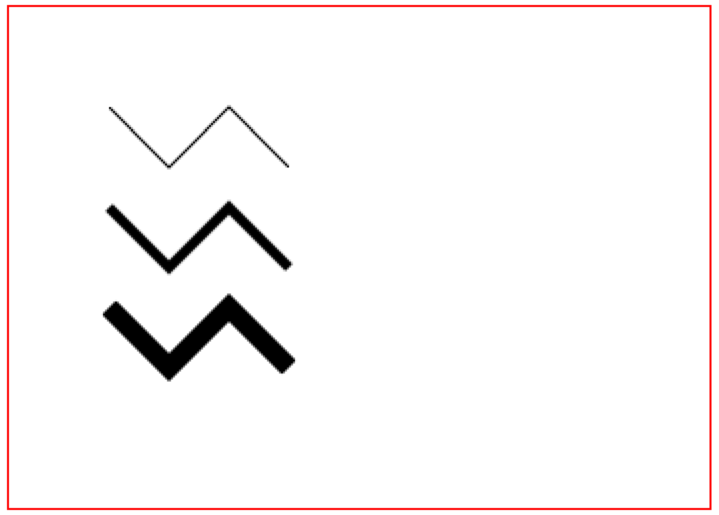
```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

ctx.lineWidth=1;
    drawPolyline(50,50);

    ctx.lineWidth=5;
    drawPolyline(50,100);

    ctx.lineWidth=10;
    drawPolyline(50,150);
```

```
    ctx.font='14px arial';
    ctx.fillText('CSS color',100,40);
    ctx.fillText('Linear Gradient color',250,40);
    ctx.fillText('Radial Gradient color',100,140);
    ctx.fillText('Pattern color',250,140);

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=350 height=250></canvas>
</body>
</html>
```

Section 5.18: lineWidth (A path styling attribute)

```
context.lineWidth=lineWidth
```

Sets the width of the line that will stroke the outline of the path



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    ctx.lineWidth=1;
    drawPolyline(50,50);

    ctx.lineWidth=5;
    drawPolyline(50,100);

    ctx.lineWidth=10;
    drawPolyline(50,150);
    // 绘制折线的工具函数
    function drawPolyline(x,y){
        ctx.beginPath();
        ctx.moveTo(x,y);
        ctx.lineTo(x+30,y+30);
        ctx.lineTo(x+60,y);
        ctx.lineTo(x+90,y+30);
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=350 height=250></canvas>
</body>
</html>
```

第5.19节：shadowColor、shadowBlur、shadowOffsetX、shadowOffsetY（路径样式属性）

```
shadowColor = 颜色        // CSS 颜色
shadowBlur =  模糊宽度        // 整数模糊宽度
shadowOffsetX = 水平偏移距离   // 阴影在水平方向上的偏移量
shadowOffsetY = 垂直偏移距离   // 阴影在垂直方向上的偏移量
```

这组属性会在路径周围添加阴影。

填充路径和描边路径都可以有阴影。

阴影在路径边缘处最暗（不透明），并随着距离路径边缘的延伸逐渐变淡。

- shadowColor 指定用于创建阴影的 CSS 颜色。
- shadowBlur 是阴影从路径向外扩展的距离。
- shadowOffsetX 是阴影在水平方向上相对于路径的偏移距离。正值使阴影向右移动，负值使阴影向左移动。
- shadowOffsetY 是阴影在垂直方向上相对于路径的偏移距离。正值使阴影向下移动，负值使阴影向上移动
  。

关于 shadowOffsetX 和 shadowOffsetY

需要注意的是，整个阴影会整体移动。这会导致阴影的一部分移到填充路径下方，因此阴影的这部分将不可见。

关于带阴影的描边

当描边带有阴影时，描边的内侧和外侧都会有阴影。阴影在描边处最暗，随着阴影向描边两侧延伸逐渐变淡。

完成后关闭阴影

在绘制阴影后，你可能想关闭阴影以绘制更多路径。要关闭阴影，
你需要将shadowColor设置为透明。

```
context.shadowColor = 'rgba(0,0,0,0)';
```

```
    // utility to draw a polyline
    function drawPolyline(x,y){
        ctx.beginPath();
        ctx.moveTo(x,y);
        ctx.lineTo(x+30,y+30);
        ctx.lineTo(x+60,y);
        ctx.lineTo(x+90,y+30);
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=350 height=250></canvas>
</body>
</html>
```

Section 5.19: shadowColor, shadowBlur, shadowOffsetX, shadowOffsetY (path styling attributes)

```
shadowColor = color         // CSS color
shadowBlur =  width         // integer blur width
shadowOffsetX = distance    // shadow is moved horizontally by this offset
shadowOffsetY = distance    // shadow is moved vertically by this offset
```

This set of attributes will add a shadow around a path.

Both filled paths and stroked paths may have a shadow.

The shadow is darkest (opaque) at the path perimeter and becomes gradiently lighter as it extends away from the path perimeter.

- **shadowColor** indicates which CSS color will be used to create the shadow.
- **shadowBlur** is the distance over which the shadow extends outward from the path.
- **shadowOffsetX** is a distance by which the shadow is shifted horizontally away from the path. A positive distance moves the shadow rightward, a negative distance moves the shadow leftward.
- **shadowOffsetY** is a distance by which the shadow is shifted vertically away from the path. A positive distance moves the shadow downward, a negative distance moves the shadow upward.

About shadowOffsetX & shadowOffsetY

It's important to note that *the whole shadow is shifted in its entirety*. This will cause part of the shadow to shift underneath filled paths and therefore part of the shadow will not be visible.

About shadowed strokes

When shadowing a stroke, both the inside and the outside of the stroke are shadowed. The shadow is darkest at the stroke and lightens as the shadow extends outward in both directions from the stroke.

Turning off shadowing when done

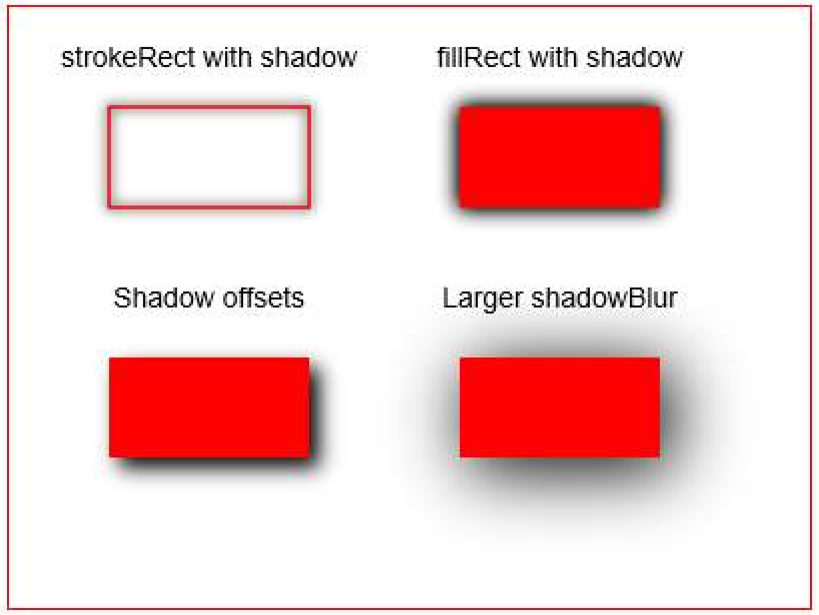
After you have drawn your shadows, you might want to turn shadowing off to draw more paths. To turn shadowing off you set the `shadowColor` to transparent.

```
context.shadowColor = 'rgba(0,0,0,0)';
```

性能考虑

阴影（如渐变）需要大量计算，因此应尽量少用阴影。

在动画时尤其要小心，因为每秒多次绘制阴影会大大影响
性能。如果需要对带阴影的路径进行动画处理，一种解决方法是预先在第二个“阴影画布”上创建带阴影的路径。
阴影画布是使用
`document.createElement` 创建的普通画布——它不会被添加到DOM中（只是一个临时画布）。然后将阴影画布
绘制到主画布上。这种方法更快，因为阴影计算不需要每秒多次进行。
你所做的只是将一个预先构建的画布复制到可见画布上。

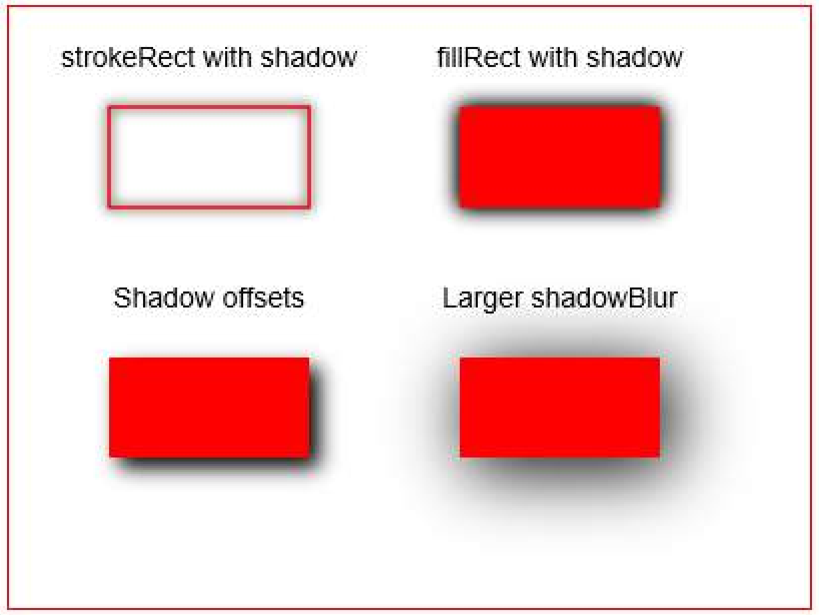


```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 带阴影的描边
ctx.shadowColor='black';
    ctx.shadowBlur=6;
ctx.strokeStyle='red';
    ctx.strokeRect(50,50,100,50);
    // 通过第二次描边使阴影加深
ctx.strokeRect(50,50,100,50);

    // 带阴影的填充
ctx.shadowColor='black';
    ctx.shadowBlur=10;
ctx.fillStyle='red';
    ctx.fillRect(225,50,100,50);
    // 通过第二次描边使阴影加深
ctx.fillRect(225,50,100,50);
```

Performance considerations

Shadows (like gradients) requires extensive computations and therefore you should use shadows sparingly.

Be especially cautious when animating because drawing shadows many times per second will greatly impact performance. A workaround if you need to animate shadowed paths is to pre-create the shadowed path on a second "shadow-canvas". The shadow-canvas is a normal canvas that is created in memory with `document.createElement` -- it is not added to the DOM (it's just a staging canvas). Then draw the shadow-canvas onto the main canvas. This is much faster because the shadow computations needn't be made many times per second. All you're doing is copying one prebuilt canvas onto your visible canvas.



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // shadowed stroke
    ctx.shadowColor='black';
    ctx.shadowBlur=6;
    ctx.strokeStyle='red';
    ctx.strokeRect(50,50,100,50);
    // darken the shadow by stroking a second time
    ctx.strokeRect(50,50,100,50);

    // shadowed fill
    ctx.shadowColor='black';
    ctx.shadowBlur=10;
    ctx.fillStyle='red';
    ctx.fillRect(225,50,100,50);
    // darken the shadow by stroking a second time
    ctx.fillRect(225,50,100,50);
    // 阴影向右下方偏移
ctx.shadowColor='black';
    ctx.shadowBlur=10;
ctx.shadowOffsetX=5;
    ctx.shadowOffsetY=5;
    ctx.fillStyle='red';
    ctx.fillRect(50,175,100,50);

    // 更宽的模糊（即从路径延伸得更远）
ctx.shadowColor='black';
    ctx.shadowBlur=35;
ctx.fillStyle='红色';
    ctx.fillRect(225,175,100,50);

    // 始终清理！关闭阴影
ctx.shadowColor='rgba(0,0,0,0)';

}); // window.onload 结束
</script>
</head>
<body>
    <canvas id="canvas" width=400 height=300></canvas>
</body>
</html>
```

第5.20节：createLinearGradient（创建路径样式对象）

```
var gradient = createLinearGradient( startX, startY, endX, endY )
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[可选地添加更多颜色停点以丰富渐变效果]
```

创建一个可重用的线性渐变（对象）。

该对象可以被分配给任何strokeStyle和/或fillStyle。

然后调用 stroke() 或 fill() 会用该对象的渐变颜色为路径着色。

创建渐变对象是一个两步过程：

1. 创建渐变对象本身。创建时你需要定义画布上渐变的起点和终点。渐变对象通过 var gradient = context.createLinearGradient 创建。
2. 然后添加构成渐变的2种（或更多）颜色。这是通过使用gradient.addColorStop向渐变对象添加多个颜色停点来完成的。

参数：

- startX,startY是渐变开始的画布坐标。在起始点（及之前），画布的颜色完全是最低的gradientPercentPosition对应的颜色。
- endX,endY是渐变结束的画布坐标。在结束点（及之后），画布的颜色完全是最高的gradientPercentPosition对应的颜色。
- gradientPercentPosition是一个介于0.00到1.00之间的浮点数，分配给某个颜色停点。它基本上是沿线段的一个百分比位置，表示该颜色停点的适用位置。
    - 渐变从百分比0.00开始，即画布上的[startX,startY]位置。

```
    // the shadow offset rightward and downward
    ctx.shadowColor='black';
    ctx.shadowBlur=10;
    ctx.shadowOffsetX=5;
    ctx.shadowOffsetY=5;
    ctx.fillStyle='red';
    ctx.fillRect(50,175,100,50);

    // a wider blur (==extends further from the path)
    ctx.shadowColor='black';
    ctx.shadowBlur=35;
    ctx.fillStyle='red';
    ctx.fillRect(225,175,100,50);

    // always clean up! Turn off shadowing
    ctx.shadowColor='rgba(0,0,0,0)';

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=400 height=300></canvas>
</body>
</html>
```

Section 5.20: createLinearGradient (creates a path styling object)

```
var gradient = createLinearGradient( startX, startY, endX, endY )
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]
```

Creates a reusable linear gradient (object).

The object can be assigned to any `strokeStyle` and/or `fillStyle`.

Then stroke() or fill() will color the Path with the gradient colors of the object.

Creating a gradient object is a 2-step process:

1. Create the gradient object itself. During creation you define a line on the canvas where the gradient will start and end. The gradient object is created with `var gradient = context.createLinearGradient`.
2. Then add 2 (or more) colors that make up the gradient. This is done by adding multiple color stops to the gradient object with `gradient.addColorStop`.

Arguments:

- **startX,startY** is the canvas coordinate where the gradient starts. At the starting point (and before) the canvas is solidly the color of the lowest `gradientPercentPosition`.
- **endX,endY** is the canvas coordinate where the gradient ends. At the ending point (and after) the canvas is solidly the color of the highest `gradientPercentPosition`.
- **gradientPercentPosition** is a float number between 0.00 and 1.00 assigned to a color stop. It is basically a percentage waypoint along the line where this particular color stop applies.
    - The gradient begins at percentage 0.00 which is [startX,startY] on the canvas.

- 渐变在百分比1.00结束，即画布上的[endX,endY]位置。
- 技术说明："百分比"一词在技术上并不准确，因为数值是从0.00到1.00，而不是0%到100%。

- CssColor是分配给该颜色停点的CSS颜色。

渐变对象是一个对象，你可以使用（并重复使用！）它来使路径描边和填充呈现渐变色。

附注：渐变对象不是 Canvas 元素或其上下文的内部对象。它是一个独立且可重用的JavaScript 对象，你可以将其分配给任何你想要的路径。你甚至可以使用该对象为不同 Canvas 元素上的路径着色(!)

颜色停靠点是沿渐变线的（百分比）路径点。在每个颜色停靠点路径点处，渐变被完全（即不透明地）填充为其指定的颜色。颜色停靠点之间的渐变线上的中间点则被填充为该颜色与前一个颜色的渐变色。

关于Canvas渐变的重要提示！

当你创建一个渐变对象时，整个画布都会被该渐变"隐形"填充。

当你stroke()或fill()一个路径时，隐形的渐变才会被显示，但仅在被描边或填充的路径上显示。

1.如果你像这样创建一个从红色到品红的线性渐变：

```
 // 创建一个线性渐变
 var gradient=ctx.createLinearGradient(100,0,canvas.width-100,0);
 gradient.addColorStop(0,'red');
gradient.addColorStop(1,'magenta');
 ctx.fillStyle=gradient;
```

2.那么Canvas将"隐形"地看到你创建的渐变如下：

3. 但在你使用stroke()或fill()应用该渐变之前，你在画布上看不到任何渐变效果。
4. 最后，如果你使用渐变来描边或填充路径，那个"不可见"的渐变将在画布上变得可见……
   但仅在路径被绘制的地方。

- The gradient ends at percentage 1.00 which is [endX,endY] on the canvas.
- *Technical note:* The term "percentage" is not technically correct since the values go from 0.00 to 1.00 rather than 0% to 100%.

- **CssColor** is a CSS color assigned to this particular color stop.

The gradient object is an object that you can use (and reuse!) to make your path strokes and fills become gradient colored.

Side Note: The gradient object is not internal to the Canvas element nor it's Context. It is a separate and reusable JavaScript object that you can assign to any Path you desire. You can even use this object to color a Path on a different Canvas element(!)

Color stops are (percentage) waypoints along the gradient line. At each color stop waypoint, the gradient is fully (==opaquely) colored with it's assigned color. Interim points along the gradient line between color stops are colored as gradients of the this and the previous color.

Important hint about Canvas gradients!

When you create a gradient object, the entire canvas is "invisibly" filled with that gradient.

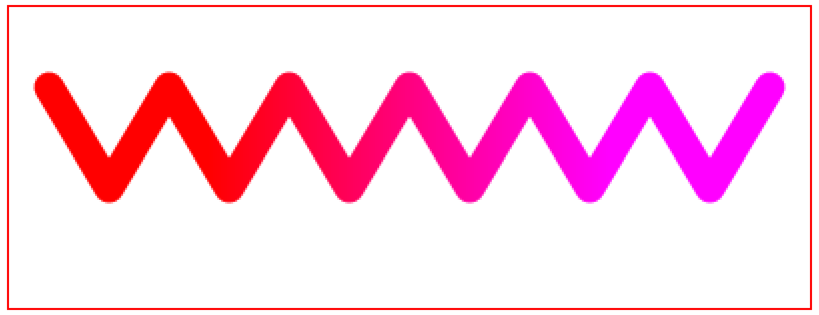
When you `stroke()` or `fill()` a path, the invisible gradient is revealed, but only revealed over that path being stroked or filled.

1. If you create a red-to-magenta linear gradient like this:

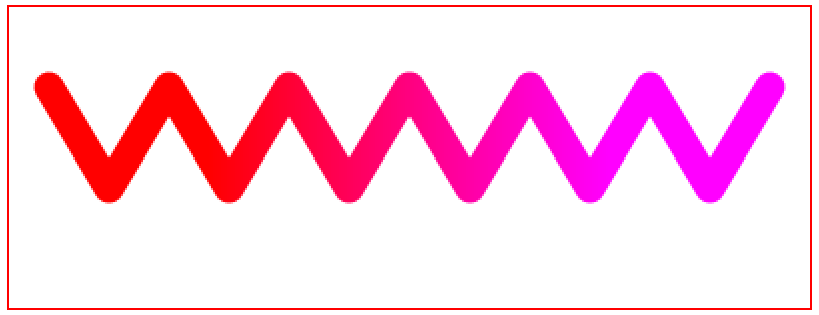
```
 // create a linearGradient
 var gradient=ctx.createLinearGradient(100,0,canvas.width-100,0);
 gradient.addColorStop(0,'red');
 gradient.addColorStop(1,'magenta');
 ctx.fillStyle=gradient;
```

2. Then Canvas will "invisibly" see your gradient creation like this:

3. But until you `stroke()` or `fill()` with the gradient, you will see none of the gradient on the Canvas.
4. Finally, if you stroke or fill a path using the gradient, the "invisible" gradient becomes visible on the Canvas ... but only where the path is drawn.



```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 创建一个线性渐变
    // 注意：在此过程中不会有任何视觉效果出现
    var gradient=ctx.createLinearGradient(100,0,canvas.width-100,0);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'magenta');

    // 创建一条折线路径
    // 注意：在此过程中不会有任何视觉效果出现
    var x=20;
    var y=40;
ctx.lineCap='round';
    ctx.lineJoin='round';
    ctx.lineWidth=15;
    ctx.beginPath();
    ctx.moveTo(x,y);
    ctx.lineTo(x+30,y+50);
    ctx.lineTo(x+60,y);
    ctx.lineTo(x+90,y+50);
    ctx.lineTo(x+120,y);
    ctx.lineTo(x+150,y+50);
    ctx.lineTo(x+180,y);
    ctx.lineTo(x+210,y+50);
    ctx.lineTo(x+240,y);
    ctx.lineTo(x+270,y+50);
    ctx.lineTo(x+300,y);
    ctx.lineTo(x+330,y+50);
    ctx.lineTo(x+360,y);

    // 设置描边样式为渐变
    // 注意：在此过程中不会有任何视觉效果出现
ctx.strokeStyle=gradient;

    // 描边路径
    // 终于！带渐变描边的路径在画布上可见了
ctx.stroke();

}); // window.onload 结束
```



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // Create a linearGradient
    // Note: Nothing visually appears during this process
    var gradient=ctx.createLinearGradient(100,0,canvas.width-100,0);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'magenta');

    // Create a polyline path
    // Note: Nothing visually appears during this process
    var x=20;
    var y=40;
    ctx.lineCap='round';
    ctx.lineJoin='round';
    ctx.lineWidth=15;
    ctx.beginPath();
    ctx.moveTo(x,y);
    ctx.lineTo(x+30,y+50);
    ctx.lineTo(x+60,y);
    ctx.lineTo(x+90,y+50);
    ctx.lineTo(x+120,y);
    ctx.lineTo(x+150,y+50);
    ctx.lineTo(x+180,y);
    ctx.lineTo(x+210,y+50);
    ctx.lineTo(x+240,y);
    ctx.lineTo(x+270,y+50);
    ctx.lineTo(x+300,y);
    ctx.lineTo(x+330,y+50);
    ctx.lineTo(x+360,y);

    // Set the stroke style to be the gradient
    // Note: Nothing visually appears during this process
    ctx.strokeStyle=gradient;

    // stroke the path
    // FINALLY! The gradient-stroked path is visible on the canvas
    ctx.stroke();

}); // end window.onload
```

```
</script>
</head>
<body>
    <canvas id="canvas" width=400 height=150></canvas>
</body>
</html>
```

第5.21节：createRadialGradient（创建路径样式对象）

```
var gradient = createRadialGradient(
centerX1, centerY1, radius1,      // 这是"显示"圆
      centerX2, centerY2, radius2       // 这是"投射光"圆
)

gradient.addColorStop(gradientPercentPosition, CssColor)
[可选地添加更多颜色停点以丰富渐变效果]
```

创建一个可重用的径向渐变（对象）。渐变对象是一个可以用来（并重复使用！）使路径描边和填充呈现渐变色的对象。

关于...

Canvas的径向渐变与传统的径向渐变截然不同。

Canvas径向渐变的"官方"（几乎难以理解！）定义在本文底部。如果你心理承受能力较弱，请不要查看！！

用（几乎能理解的）术语来说：

- 径向渐变有两个圆："投射"圆和"显示"圆。
- 投射圆向显示圆投射光线。
- 这道光线就是渐变。
- 这道渐变光的形状由两个圆的相对大小和位置决定。

创建渐变对象是一个两步过程：

1. 创建渐变对象本身。创建时你需要定义画布上渐变开始和结束的线。渐变对象通过var gradient = context.radialLinearGradient 创建。
2. 然后添加构成渐变的2种（或更多）颜色。这是通过使用gradient.addColorStop向渐变对象添加多个颜色停点来完成的。

参数：

- centerX1,centerY1,radius1 定义了第一个圆，渐变将在该圆内显示。
- centerX2,centerY2,radius2 定义了第二个圆，该圆向第一个圆投射渐变光。
- gradientPercentPosition 是一个介于0.00和1.00之间的浮点数，分配给颜色停点。它基本上是一个百分比位置，定义了该颜色停点在渐变中的应用位置。
  - 渐变从百分比0.00开始。
  - 渐变在百分比1.00结束。
  - 技术说明："百分比"一词在技术上并不准确，因为数值是从0.00到1.00，而不是0%到100%。

```
</script>
</head>
<body>
    <canvas id="canvas" width=400 height=150></canvas>
</body>
</html>
```

Section 5.21: createRadialGradient (creates a path styling object)

```
var gradient = createRadialGradient(
      centerX1, centerY1, radius1,      // this is the "display' circle
      centerX2, centerY2, radius2       // this is the "light casting" circle
)
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]
```

Creates a reusable radial gradient (object). The gradient object is an object that you can use (and reuse!) to make your path strokes and fills become gradient colored.

About...

The Canvas radial gradient is *extremely different* from traditional radial gradients.

The "official" (almost undecipherable!) definition of Canvas's radial gradient is at the bottom of this posting. Don't look at it if you have a weak disposition!!

In (almost understandable) terms:

- The radial gradient has 2 circles: a "casting" circle and a "display" circle.
- The casting circle casts light into the display circle.
- That light is the gradient.
- The shape of that gradient light is determined by the relative size and position of both circles.

Creating a gradient object is a 2-step process:

1. Create the gradient object itself. During creation you define a line on the canvas where the gradient will start and end. The gradient object is created with `var gradient = context.radialLinearGradient`.
2. Then add 2 (or more) colors that make up the gradient. This is done by adding multiple color stops to the gradient object with `gradient.addColorStop`.

Arguments:

- **centerX1,centerY1,radius1** defines a first circle where the gradient will be displayed.
- **centerX2,centerY2,radius2** defines a second circle which is casting gradient light into the first circle.
- **gradientPercentPosition** is a float number between 0.00 and 1.00 assigned to a color stop. It is basically a percentage waypoint defining where this particular color stop applies along the gradient.
  - The gradient begins at percentage 0.00.
  - The gradient ends at percentage 1.00.
  - *Technical note:* The term "percentage" is not technically correct since the values go from 0.00 to 1.00 rather than 0% to 100%.

- CssColor是分配给该颜色停点的CSS颜色。

附注：渐变对象不是 Canvas 元素或其上下文的内部对象。它是一个独立且可重用的JavaScript 对象，你可以将其分配给任何你想要的路径。你甚至可以使用该对象为不同 Canvas 元素上的路径着色(!)

颜色停靠点是沿渐变线的（百分比）路径点。在每个颜色停靠点路径点处，渐变被完全（即不透明地）填充为其指定的颜色。颜色停靠点之间的渐变线上的中间点则被填充为该颜色与前一个颜色的渐变色。

关于Canvas渐变的重要提示！

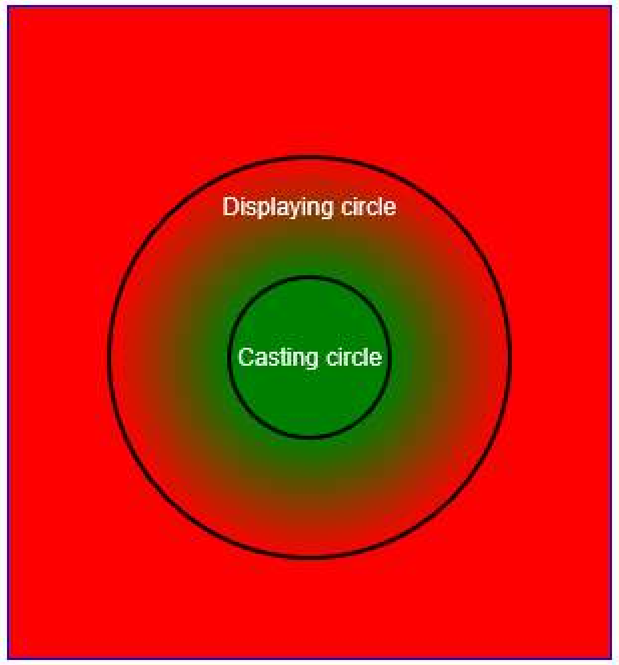
当你创建一个渐变对象时，整个径向渐变会“隐形”地投射到画布上。

当你stroke()或fill()一个路径时，隐形的渐变才会被显示，但仅在被描边或填充的路径上显示。

1.如果你创建一个绿色到红色的径向渐变，如下所示：

```
// 创建一个径向渐变
var x1=150;
var y1=150;
var x2=280;
var y2=150;
var r1=100;
var r2=120;
var gradient=ctx.createRadialGradient(x1,y1,r1,x2,y2,r2);
gradient.addColorStop(0,'red');
gradient.addColorStop(1,'green');
ctx.fillStyle=gradient;
```

2.那么Canvas将“隐形”地看到你创建的渐变如下：



3. 但在你使用stroke()或fill()应用该渐变之前，你在画布上看不到任何渐变效果。

4. 最后，如果你使用渐变来描边或填充路径，那个“不可见”的渐变将在画布上变得可见......

- **CssColor** is a CSS color assigned to this particular color stop.

Side Note: The gradient object is not internal to the Canvas element nor it's Context. It is a separate and reusable JavaScript object that you can assign to any Path you desire. You can even use this object to color a Path on a different Canvas element(!)

Color stops are (percentage) waypoints along the gradient line. At each color stop waypoint, the gradient is fully (==opaquely) colored with it's assigned color. Interim points along the gradient line between color stops are colored as gradients of the this and the previous color.

Important hint about Canvas gradients!

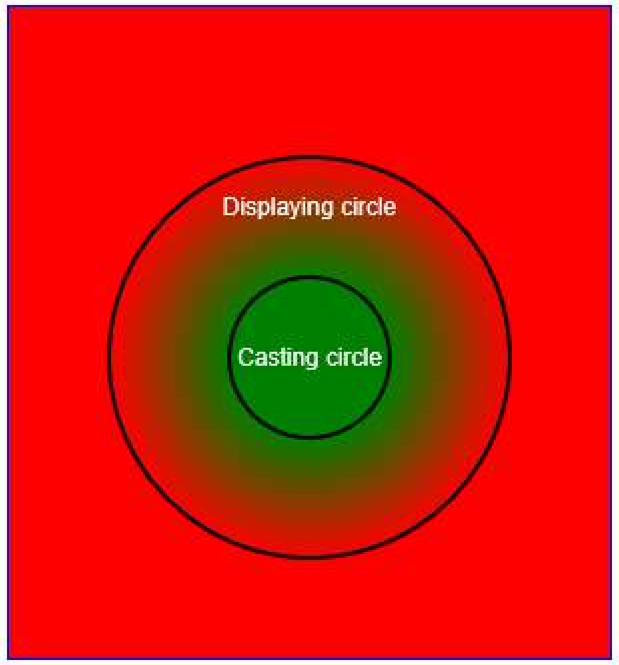
When you create a gradient object, the entire radial gradient is "invisibly" cast upon the canvas.

When you `stroke()` or `fill()` a path, the invisible gradient is revealed, but only revealed over that path being stroked or filled.

1. If you create a green-to-red radial gradient like this:

```
// create a radialGradient
var x1=150;
var y1=150;
var x2=280;
var y2=150;
var r1=100;
var r2=120;
var gradient=ctx.createRadialGradient(x1,y1,r1,x2,y2,r2);
gradient.addColorStop(0,'red');
gradient.addColorStop(1,'green');
ctx.fillStyle=gradient;
```

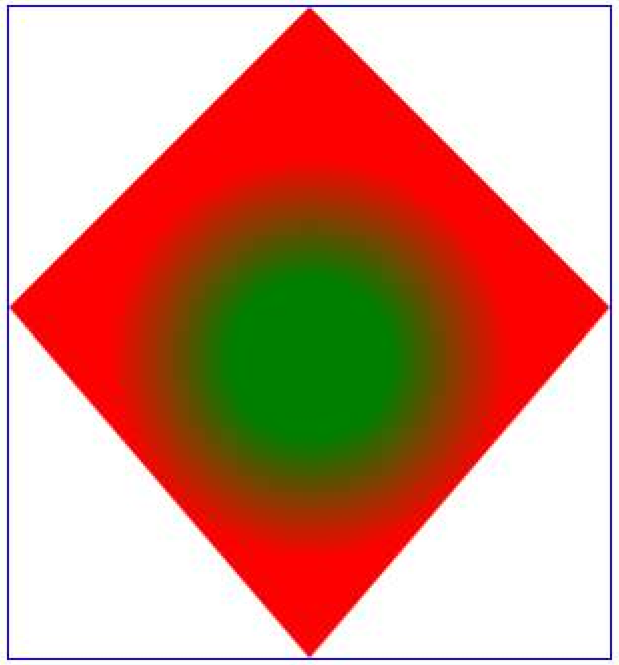
2. Then Canvas will "invisibly" see your gradient creation like this:



3. But until you `stroke()` or `fill()` with the gradient, you will see none of the gradient on the Canvas.

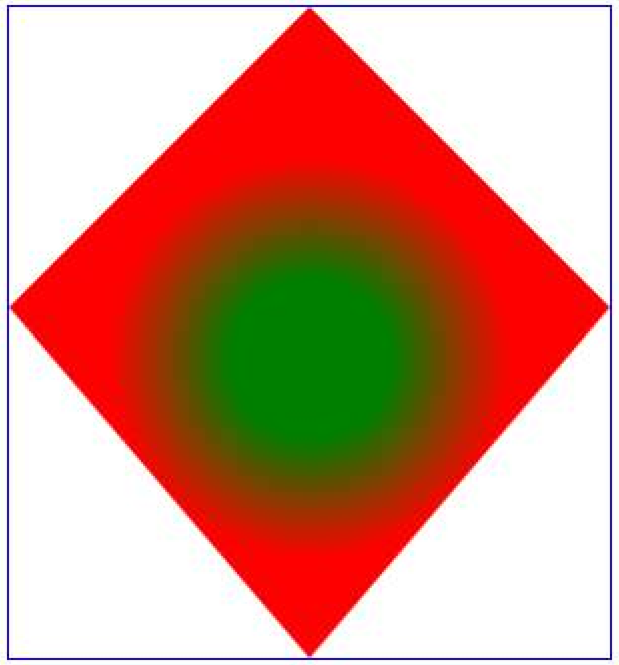
4. Finally, if you stroke or fill a path using the gradient, the "invisible" gradient becomes visible on the Canvas ...

但仅在路径被绘制的地方。



```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; padding:10px; }
    #canvas{border:1px solid blue; }
</style>
<script>
window.onload=(function(){

    // canvas 相关变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // 创建一个径向渐变
    var x1=150;
    var y1=175;
    var x2=350;
    var y2=175;
    var r1=100;
    var r2=40;
x2=x1;
    var gradient=ctx.createRadialGradient(x1,y1,r1,x2,y2,r2);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.fillStyle=gradient;

    // 用渐变填充路径
ctx.beginPath();
    ctx.moveTo(150,0);
    ctx.lineTo(300,150);
    ctx.lineTo(150,325);
    ctx.lineTo(0,150);
    ctx.lineTo(150,0);
    ctx.fill();

}); // end window.onload
</script>
```

but only where the path is drawn.



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; padding:10px; }
    #canvas{border:1px solid blue; }
</style>
<script>
window.onload=(function(){

    // canvas related vars
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // create a radial gradient
    var x1=150;
    var y1=175;
    var x2=350;
    var y2=175;
    var r1=100;
    var r2=40;
    x2=x1;
    var gradient=ctx.createRadialGradient(x1,y1,r1,x2,y2,r2);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.fillStyle=gradient;

    // fill a path with the gradient
    ctx.beginPath();
    ctx.moveTo(150,0);
    ctx.lineTo(300,150);
    ctx.lineTo(150,325);
    ctx.lineTo(0,150);
    ctx.lineTo(150,0);
    ctx.fill();

}); // end window.onload
</script>
```

```
</head>
<body>
    <canvas id="canvas" width=300 height=325></canvas>
</body>
</html>
```

官方令人害怕的细节

谁决定了`createRadialGradient`的功能？

W3C发布官方推荐规范，浏览器据此构建Html5 Canvas元素。

W3C 关于 createRadialGradient 的规范晦涩地写道：

createRadialGradient 创建什么

> createRadialGradient ... 实际上创建了一个圆锥体，该圆锥体被渐变创建时定义的两个圆所接触，圆锥体在起始圆（0.0）之前的部分使用第一个偏移量的颜色，圆锥体在结束圆（1.0）之后的部分使用最后一个偏移量的颜色，圆锥体外部未被渐变触及的区域为透明黑色。

它内部是如何工作的

> createRadialGradient(x0, y0, r0, x1, y1, r1) 方法接受六个参数，前三个表示起始圆，圆心为 (x0, y0)，半径为 r0，后三个表示结束圆，圆心为 (x1, y1)，半径为 r1。数值以坐标空间单位表示。如果 r0 或 r1 中任一为负数，则必须抛出 IndexSizeError 异常。否则，该方法必须返回一个以这两个指定圆初始化的径向 CanvasGradient。
>
> 径向渐变必须按照以下步骤渲染：
>
> 1. 如果 x0 = x1 且 y0 = y1 且 r0 = r1，则径向渐变不应绘制任何内容。中止这些步骤。
> 2. 令 x(ω) = (x1 - x0) ω + x0；令 y(ω) = (y1 - y0) ω + y0；令 r(ω) = (r1 - r0) ω + r0。令 ω 处的颜色为该位置上的渐变颜色（颜色来自上述插值和外推）。
> 3. 对于所有满足 r(ω) > 0 的 ω 值，从最接近正无穷的 ω 值开始，到以值ω最接近负无穷的值，在位置(x(ω), y(ω))绘制半径为r(ω)的圆周，颜色为ω，但仅在本次渐变渲染步骤中尚未被之前圆形涂绘的画布部分进行绘制。

```
</head>
<body>
    <canvas id="canvas" width=300 height=325></canvas>
</body>
</html>
```

The scary official details

Who decides what `createRadialGradient does?

The W3C issues the official recommended specifications that browsers use to build the Html5 Canvas element.

The W3C specification for `createRadialGradient` cryptically reads like this:

What does `createRadialGradient` create

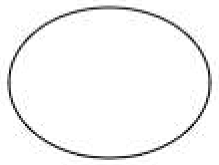
> `createRadialGradient` … effectively creates a cone, touched by the two circles defined in the creation of the gradient, with the part of the cone before the start circle (0.0) using the color of the first offset, the part of the cone after the end circle (1.0) using the color of the last offset, and areas outside the cone untouched by the gradient (transparent black).

How does it work internally

> The `createRadialGradient(x0, y0, r0, x1, y1, r1)` method takes six arguments, the first three representing the start circle with origin (x0, y0) and radius r0, and the last three representing the end circle with origin (x1, y1) and radius r1. The values are in coordinate space units. If either of r0 or r1 are negative, an IndexSizeError exception must be thrown. Otherwise, the method must return a radial CanvasGradient initialized with the two specified circles.
>
> Radial gradients must be rendered by following these steps:
>
> 1. If x0 = x1 and y0 = y1 and r0 = r1, then the radial gradient must paint nothing. Abort these steps.
> 2. Let x(ω) = (x1-x0)ω + x0; Let y(ω) = (y1-y0)ω + y0; Let r(ω) = (r1-r0)ω + r0 Let the color at ω be the color at that position on the gradient (with the colors coming from the interpolation and extrapolation described above).
> 3. For all values of ω where r(ω) > 0, starting with the value of ω nearest to positive infinity and ending with the value of ω nearest to negative infinity, draw the circumference of the circle with radius r(ω) at position (x(ω), y(ω)), with the color at ω, but only painting on the parts of the canvas that have not yet been painted on by earlier circles in this step for this rendering of the gradient.

第6章：路径

第6.1节：椭圆



注意：浏览器正在逐步添加内置的context.ellipse绘图命令，但该命令尚未被所有浏览器普遍支持（尤其是IE不支持）。以下方法在所有浏览器中均可使用。

根据所需的左上角坐标绘制椭圆：

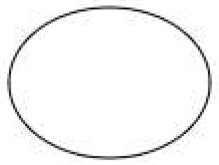
```
// 根据x,y作为左上角坐标绘制椭圆
function drawEllipse(x,y,width,height){
    var PI2=Math.PI*2;
    var ratio=height/width;
    var radius=Math.max(width,height)/2;
    var increment = 1 / radius;
    var cx=x+width/2;
    var cy=y+height/2;

ctx.beginPath();
    var x = cx + radius * Math.cos(0);
    var y = cy - ratio * radius * Math.sin(0);
    ctx.lineTo(x,y);

    for(var radians=increment; radians<PI2; radians+=increment){
        var x = cx + radius * Math.cos(radians);
        var y = cy - ratio * radius * Math.sin(radians);
        ctx.lineTo(x,y);
     }

ctx.closePath();
    ctx.stroke();
}
```

绘制一个椭圆，给定其期望的中心点坐标：

```
// 根据 cx, cy 作为椭圆的中心点坐标绘制椭圆
function drawEllipse2(cx,cy,width,height){
    var PI2=Math.PI*2;
    var ratio=height/width;
    var radius=Math.max(width,height)/2;
    var increment = 1 / radius;

ctx.beginPath();
    var x = cx + radius * Math.cos(0);
    var y = cy - ratio * radius * Math.sin(0);
    ctx.lineTo(x,y);

    for(var radians=increment; radians<PI2; radians+=increment){
        var x = cx + radius * Math.cos(radians);
        var y = cy - ratio * radius * Math.sin(radians);
        ctx.lineTo(x,y);
     }
```

Chapter 6: Paths

Section 6.1: Ellipse



Note: Browsers are in the process of adding a built-in `context.ellipse` drawing command, but this command is not universally adopted (notably not in IE). The methods below work in all browsers.

Draw an ellipse given it's desired top-left coordinate:

```
// draws an ellipse based on x,y being top-left coordinate
function drawEllipse(x,y,width,height){
    var PI2=Math.PI*2;
    var ratio=height/width;
    var radius=Math.max(width,height)/2;
    var increment = 1 / radius;
    var cx=x+width/2;
    var cy=y+height/2;

    ctx.beginPath();
    var x = cx + radius * Math.cos(0);
    var y = cy - ratio * radius * Math.sin(0);
    ctx.lineTo(x,y);

    for(var radians=increment; radians<PI2; radians+=increment){
        var x = cx + radius * Math.cos(radians);
        var y = cy - ratio * radius * Math.sin(radians);
        ctx.lineTo(x,y);
     }

    ctx.closePath();
    ctx.stroke();
}
```

Draw an ellipse given it's desired center point coordinate:

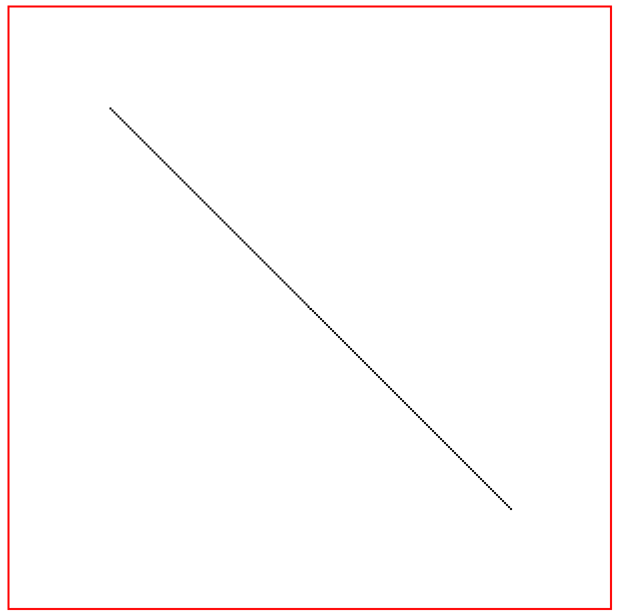
```
// draws an ellipse based on cx,cy being ellipse's centerpoint coordinate
function drawEllipse2(cx,cy,width,height){
    var PI2=Math.PI*2;
    var ratio=height/width;
    var radius=Math.max(width,height)/2;
    var increment = 1 / radius;

    ctx.beginPath();
    var x = cx + radius * Math.cos(0);
    var y = cy - ratio * radius * Math.sin(0);
    ctx.lineTo(x,y);

    for(var radians=increment; radians<PI2; radians+=increment){
        var x = cx + radius * Math.cos(radians);
        var y = cy - ratio * radius * Math.sin(radians);
        ctx.lineTo(x,y);
     }
ctx.closePath();
    ctx.stroke();
}
```

第6.2节：无模糊的线条

当 Canvas 绘制线条时，它会自动添加抗锯齿以视觉上修复“锯齿状”。结果是线条锯齿减少但更模糊。

此函数使用Bresenham的线算法绘制两点之间的线条，无抗锯齿。结果是清晰且无锯齿的线条。

重要提示： 这种逐像素方法比context.lineTo绘图方法要慢得多。



```
// 用法：
bresenhamLine(50,50,250,250);

// x,y 线起点
// xx,yy 线终点
// 绘制线起点和终点的像素
function bresenhamLine(x, y, xx, yy){
    var oldFill = ctx.fillStyle;  // 保存旧的填充样式
    ctx.fillStyle = ctx.strokeStyle; // 将描边样式赋给填充样式
    xx = Math.floor(xx);
yy = Math.floor(yy);
    x = Math.floor(x);
    y = Math.floor(y);
    // 布雷森汉姆算法
    var dx =  Math.abs(xx-x);
    var sx = x < xx ? 1 : -1;
    var dy = -Math.abs(yy-y);
    var sy = y<yy ? 1 : -1;
    var err = dx+dy;
    var errC; // 误差值
    var end = false;
    var x1 = x;
    var y1 = y;

while(!end){
ctx.fillRect(x1, y1, 1, 1); // 将每个像素绘制为矩形
```

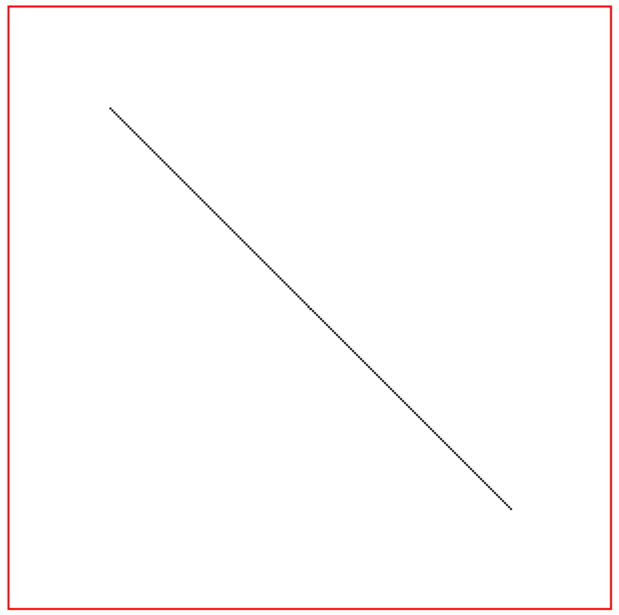
```
    ctx.closePath();
    ctx.stroke();
}
```

Section 6.2: Line without blurryness

When Canvas draws a line it automatically adds anti-aliasing to visually heal "jaggedness". The result is a line that is less jagged but more blurry.

This function draws a line between 2 points without anti-aliasing using Bresenham's_line algorithm. The result is a crisp line without the jaggedness.

Important Note: *This pixel-by-pixel method is a much slower drawing method than* `context.lineTo`.



```
// Usage:
bresenhamLine(50,50,250,250);

// x,y line start
// xx,yy line end
// the pixel at line start and line end are drawn
function bresenhamLine(x, y, xx, yy){
    var oldFill = ctx.fillStyle;  // save old fill style
    ctx.fillStyle = ctx.strokeStyle; // move stroke style to fill
    xx = Math.floor(xx);
    yy = Math.floor(yy);
    x = Math.floor(x);
    y = Math.floor(y);
    // BRENSENHAM
    var dx =  Math.abs(xx-x);
    var sx = x < xx ? 1 : -1;
    var dy = -Math.abs(yy-y);
    var sy = y<yy ? 1 : -1;
    var err = dx+dy;
    var errC; // error value
    var end = false;
    var x1 = x;
    var y1 = y;

    while(!end){
       ctx.fillRect(x1, y1, 1, 1); // draw each pixel as a rect
```

```
        if (x1 === xx && y1 === yy) {
            end = true;
        }else{
errC = 2*err;
            if (errC >= dy) {
                err += dy;
                x1 += sx;
            }
            if (errC <= dx) {
                err += dx;
                y1 += sy;
            }
        }
    }
ctx.fillStyle = oldFill; // 恢复原有填充样式
}
```

```
        if (x1 === xx && y1 === yy) {
            end = true;
        }else{
            errC = 2*err;
            if (errC >= dy) {
                err += dy;
                x1 += sx;
            }
            if (errC <= dx) {
                err += dx;
                y1 += sy;
            }
        }
    }
    ctx.fillStyle = oldFill; // restore old fill style
}
```

第7章：沿路径导航

第7.1节：查找曲线上的点

此示例在贝塞尔曲线或三次曲线上找到一个点，位置为position，其中position是曲线上单位距离，范围为0<= position <= 1。位置会被限制在该范围内，因此如果传入的值小于0或大于1，将分别被设置为0和1。

传入6个坐标用于二次贝塞尔，或8个坐标用于三次贝塞尔。

最后一个可选参数是返回的向量（点）。如果未提供，将会被创建。

示例用法

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var p4 = {x : 300, y : 100};
var point = {x : null, y : null};

// 用于三次贝塞尔
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// 或者不需要设置point，因为它是一个引用，会被设置
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// 或者创建一个新的点
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y);

// 用于二次贝塞尔
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// 或者不需要设置point，因为它是一个引用，会被设置
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// 或者创建一个新点
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y);
```

该函数

getPointOnCurve = function(position, x1, y1, x2, y2, x3, y3, [x4, y4], [vec])

> 注意： 参数[x4, y4]是可选的。

> 注意：x4,y4 如果为 null 或 undefined 表示曲线是二次贝塞尔曲线。 vec 是可选的，如果提供，将保存返回的点。如果未提供，将创建一个新的点。

```
var getPointOnCurve = function(position, x1, y1, x2, y2, x3, y3, x4, y4, vec){
    var vec, quad;
quad = false;
    if(vec === undefined){
        vec = {};
    }

    if(x4 === undefined || x4 === null){
        quad = true;
        x4 = x3;
```

Chapter 7: Navigating along a Path

Section 7.1: Find point on curve

This example finds a point on a bezier or cubic curve at `position` where `position` is he unit distance on the curve 0 <= `position` <= 1. The position is clamped to the range thus if values < 0 or > 1 are passed they will be set 0,1 respectively.

Pass the function 6 coordinates for quadratic bezier or 8 for cubic.

The last optional argument is the returned vector (point). If not given it will be created.

Example usage

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var p4 = {x : 300, y : 100};
var point = {x : null, y : null};

// for cubic beziers
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// or No need to set point as it is a referance and will be set
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// or to create a new point
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y);

// for quadratic beziers
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// or No need to set point as it is a referance and will be set
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// or to create a new point
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y);
```

The function

getPointOnCurve = function(position, x1, y1, x2, y2, x3, y3, [x4, y4], [vec])

> **Note:** Arguments inside [x4, y4] are optional.

> **Note:** x4,y4 if `null`, or `undefined` means that the curve is a quadratic bezier. `vec` is optional and will hold the returned point if supplied. If not it will be created.

```
var getPointOnCurve = function(position, x1, y1, x2, y2, x3, y3, x4, y4, vec){
    var vec, quad;
    quad = false;
    if(vec === undefined){
        vec = {};
    }

    if(x4 === undefined || x4 === null){
        quad = true;
        x4 = x3;
y4 = y3;
    }

    if(position <= 0){
        vec.x = x1;
        vec.y = y1;
        return vec;
    }
    if(position >= 1){
        vec.x = x4;
        vec.y = y4;
        return vec;
    }
c = position;
    if(quad){
x1 += (x2 - x1) * c;
        y1 += (y2 - y1) * c;
        x2 += (x3 - x2) * c;
        y2 += (y3 - y2) * c;
        vec.x = x1 + (x2 - x1) * c;
        vec.y = y1 + (y2 - y1) * c;
        return vec;
    }
x1 += (x2 - x1) * c;
    y1 += (y2 - y1) * c;
    x2 += (x3 - x2) * c;
    y2 += (y3 - y2) * c;
    x3 += (x4 - x3) * c;
    y3 += (y4 - y3) * c;
    x1 += (x2 - x1) * c;
    y1 += (y2 - y1) * c;
    x2 += (x3 - x2) * c;
    y2 += (y3 - y2) * c;
    vec.x = x1 + (x2 - x1) * c;
    vec.y = y1 + (y2 - y1) * c;
    return vec;
}
```

第7.2节：寻找二次曲线的范围

当你需要找到二次贝塞尔曲线的边界矩形时，可以使用以下高效的方法。

```
// 该方法由Blindman67发现，首先通过归一化控制点来降低算法复杂度

// x1,y1, x2,y2, x3,y3 分别是贝塞尔曲线的起点、控制点和终点坐标// [extent]
是可选参数，如果提供，范围将被添加到其中，允许你使用该函数

//        来获取多个贝塞尔曲线的范围。
// 返回范围对象（如果未提供则创建一个新的范围）
// 范围对象属性
// 上、左、右、下、宽度、高度
function getQuadraticCurevExtent(x1, y1, x2, y2, x3, y3, extent) {
    var brx, bx, x, bry, by, y, px, py;

    // 通过BM67归一化方程求解二次方程的边界
brx = x3 - x1; // 获取x范围
bx = x2 - x1; // 获取x控制点偏移
    x = bx / brx; // 归一化控制点，用于检查极值是否在范围内

    // 对y点做同样的处理
```

```
        y4 = y3;
    }

    if(position <= 0){
        vec.x = x1;
        vec.y = y1;
        return vec;
    }
    if(position >= 1){
        vec.x = x4;
        vec.y = y4;
        return vec;
    }
    c = position;
    if(quad){
        x1 += (x2 - x1) * c;
        y1 += (y2 - y1) * c;
        x2 += (x3 - x2) * c;
        y2 += (y3 - y2) * c;
        vec.x = x1 + (x2 - x1) * c;
        vec.y = y1 + (y2 - y1) * c;
        return vec;
    }
    x1 += (x2 - x1) * c;
    y1 += (y2 - y1) * c;
    x2 += (x3 - x2) * c;
    y2 += (y3 - y2) * c;
    x3 += (x4 - x3) * c;
    y3 += (y4 - y3) * c;
    x1 += (x2 - x1) * c;
    y1 += (y2 - y1) * c;
    x2 += (x3 - x2) * c;
    y2 += (y3 - y2) * c;
    vec.x = x1 + (x2 - x1) * c;
    vec.y = y1 + (y2 - y1) * c;
    return vec;
}
```

Section 7.2: Finding extent of Quadratic Curve

When you need to find the bounding rectangle of a quadratic bezier curve you can use the following performant method.

```
// This method was discovered by Blindman67 and solves by first normalising the control point thereby
reducing the algorithm complexity
// x1,y1, x2,y2, x3,y3 Start, Control, and End coords of bezier
// [extent] is optional and if provided the extent will be added to it allowing you to use the
function
//        to get the extent of many beziers.
// returns extent object (if not supplied a new extent is created)
// Extent object properties
// top, left,right,bottom,width,height
function getQuadraticCurevExtent(x1, y1, x2, y2, x3, y3, extent) {
    var brx, bx, x, bry, by, y, px, py;

    // solve quadratic for bounds by BM67 normalizing equation
    brx = x3 - x1; // get x range
    bx = x2 - x1; // get x control point offset
    x = bx / brx; // normalise control point which is used to check if maxima is in range

    // do the same for the y points
bry = y3 - y1;
by = y2 - y1;
y = by / bry;

px = x1; // 如果极大值超出范围，设置默认值
    py = y1;

    // 查找顶部/左侧，顶部/右侧，底部/左侧，或底部/右侧
    if (x < 0 || x > 1) { // 检查 x 的极值是否在曲线上
        px = bx * bx / (2 * bx - brx) + x1; // 获取 x 的极值
    }
    if (y < 0 || y > 1) { // 与 x 相同
        py = by * by / (2 * by - bry) + y1;
    }

    // 创建范围对象并添加范围
    if (extent === undefined) {
extent = {};
extent.left = Math.min(x1, x3, px);
        extent.top = Math.min(y1, y3, py);
        extent.right = Math.max(x1, x3, px);
        extent.bottom = Math.max(y1, y3, py);
    } else { // 使用传入的范围并扩展以适应此曲线
        extent.left = Math.min(x1, x3, px, extent.left);
        extent.top = Math.min(y1, y3, py, extent.top);
        extent.right = Math.max(x1, x3, px, extent.right);
        extent.bottom = Math.max(y1, y3, py, extent.bottom);
    }

extent.width = extent.right - extent.left;
    extent.height = extent.bottom - extent.top;
    return extent;
}
```

有关求解范围的更详细说明，请参见回答 To get extent of a quadratic bezier，其中包含可运行的演示。

第7.3节：沿三次贝塞尔曲线查找点

此示例查找沿三次贝塞尔曲线大致均匀分布的点数组。

它将使用context.bezierCurveTo创建的路径段分解为沿该曲线的点。

```
// 返回：沿三次贝塞尔曲线大致均匀分布的点数组
//
// 归属：Stackoverflow的@Blindman67
// 引用：
http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-moving-circle-
along-it/36827074#36827074
// 根据上述引用修改
//
// ptCount：沿曲线采样的点数
// pxTolerance：点之间允许的近似间距
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy：定义曲线的控制点
//
function plotCBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy,Dx,Dy){
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaDCx=Dx-Cx;
    var deltaBAy=By-Ay;
```

```
    bry = y3 - y1;
    by = y2 - y1;
    y = by / bry;

    px = x1; // set defaults in case maximas outside range
    py = y1;

    // find top/left, top/right, bottom/left, or bottom/right
    if (x < 0 || x > 1) { // check if x maxima is on the curve
        px = bx * bx / (2 * bx - brx) + x1; // get the x maxima
    }
    if (y < 0 || y > 1) { // same as x
        py = by * by / (2 * by - bry) + y1;
    }

    // create extent object and add extent
    if (extent === undefined) {
        extent = {};
        extent.left = Math.min(x1, x3, px);
        extent.top = Math.min(y1, y3, py);
        extent.right = Math.max(x1, x3, px);
        extent.bottom = Math.max(y1, y3, py);
    } else { // use splied extent and extend it to fit this curve
        extent.left = Math.min(x1, x3, px, extent.left);
        extent.top = Math.min(y1, y3, py, extent.top);
        extent.right = Math.max(x1, x3, px, extent.right);
        extent.bottom = Math.max(y1, y3, py, extent.bottom);
    }

    extent.width = extent.right - extent.left;
    extent.height = extent.bottom - extent.top;
    return extent;
}
```

For a more detailed look at solving for extent see answer To get extent of a quadratic bezier which includes runnable demos.

Section 7.3: Finding points along a cubic Bezier curve

This example finds an array of approximately evenly spaced points along a cubic Bezier curve.

It decomposes Path segments created with `context.bezierCurveTo` into points along that curve.

```
// Return: an array of approximately evenly spaced points along a cubic Bezier curve
//
// Attribution: Stackoverflow's @Blindman67
// Cite:
http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-moving-circle-
along-it/36827074#36827074
// As modified from the above citation
//
// ptCount: sample this many points at interval along the curve
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy: control points defining the curve
//
function plotCBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy,Dx,Dy){
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaDCx=Dx-Cx;
    var deltaBAy=By-Ay;
    var deltaCBy=Cy-By;
    var deltaDCy=Dy-Cy;
    var ax,ay,bx,by;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
        var t=i/ptCount;
ax=Ax+deltaBAx*t;
        bx=Bx+deltaCBx*t;
        cx=Cx+deltaDCx*t;
        ax+=(bx-ax)*t;
        bx+=(cx-bx)*t;
        //
ay=Ay+deltaBAy*t;
        by=By+deltaCBy*t;
        cy=Cy+deltaDCy*t;
        ay+=(by-ay)*t;
        by+=(cy-by)*t;
        var x=ax+(bx-ax)*t;
        var y=ay+(by-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
lastY=y;
        }
    }
pts.push({x:Dx,y:Dy});
    return(pts);
}
```

第7.4节：沿二次曲线寻找点

本例查找沿二次曲线大致均匀分布的一组点。

它将使用context.quadraticCurveTo创建的路径段分解为沿该曲线的点。

```
// 返回：沿二次曲线大致均匀分布的点数组
//
// 归属：Stackoverflow的@Blindman67
// 引用：
http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-moving-circle-
along-it/36827074#36827074
// 根据上述引用修改
//
// ptCount：沿曲线以间隔采样的点数
// pxTolerance：点之间允许的近似间距
// Ax,Ay,Bx,By,Cx,Cy：定义曲线的控制点
//
function plotQBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy){
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaBAy=By-Ay;
    var deltaCBy=Cy-By;
    var ax,ay;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
```

```
    var deltaCBy=Cy-By;
    var deltaDCy=Dy-Cy;
    var ax,ay,bx,by;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
        var t=i/ptCount;
        ax=Ax+deltaBAx*t;
        bx=Bx+deltaCBx*t;
        cx=Cx+deltaDCx*t;
        ax+=(bx-ax)*t;
        bx+=(cx-bx)*t;
        //
        ay=Ay+deltaBAy*t;
        by=By+deltaCBy*t;
        cy=Cy+deltaDCy*t;
        ay+=(by-ay)*t;
        by+=(cy-by)*t;
        var x=ax+(bx-ax)*t;
        var y=ay+(by-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Dx,y:Dy});
    return(pts);
}
```

Section 7.4: Finding points along a quadratic curve

This example finds an array of approximately evenly spaced points along a quadratic curve.

It decomposes Path segments created with `context.quadraticCurveTo` into points along that curve.

```
// Return: an array of approximately evenly spaced points along a Quadratic curve
//
// Attribution: Stackoverflow's @Blindman67
// Cite:
http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-moving-circle-
along-it/36827074#36827074
// As modified from the above citation
//
// ptCount: sample this many points at interval along the curve
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By,Cx,Cy: control points defining the curve
//
function plotQBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy){
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaBAy=By-Ay;
    var deltaCBy=Cy-By;
    var ax,ay;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
        var t=i/ptCount;
        ax=Ax+deltaBAx*t;
        ay=Ay+deltaBAy*t;
        var x=ax+((Bx+deltaCBx*t)-ax)*t;
        var y=ay+((By+deltaCBy*t)-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
lastY=y;
        }
    }
pts.push({x:Cx,y:Cy});
    return(pts);
}
```

第7.5节：沿直线寻找点

此示例查找沿直线大致均匀分布的点数组。

它将使用context.lineTo创建的路径段分解为沿该直线的点。

```
// 返回：沿直线大致均匀分布的点数组
//
// pxTolerance：点之间允许的大致间距
// Ax,Ay,Bx,By：定义直线的端点
//
function plotLine(pxTolerance,Ax,Ay,Bx,By){
    var dx=Bx-Ax;
    var dy=By-Ay;
    var ptCount=parseInt(Math.sqrt(dx*dx+dy*dy))*3;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<=ptCount;i++){
        var t=i/ptCount;
        var x=Ax+dx*t;
        var y=Ay+dy*t;
        var dx1=x-lastX;
        var dy1=y-lastY;
        if(dx1*dx1+dy1*dy1>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
lastY=y;
        }
    }
pts.push({x:Bx,y:By});
    return(pts);
}
```

第7.6节：沿包含曲线和直线的整个路径查找点

此示例查找沿整个路径大致均匀分布的点数组。

它将所有由`context.lineTo`、`context.quadraticCurveTo`和/或
`context.bezierCurveTo`创建的路径段分解为沿该路径的点。

```
        var t=i/ptCount;
        ax=Ax+deltaBAx*t;
        ay=Ay+deltaBAy*t;
        var x=ax+((Bx+deltaCBx*t)-ax)*t;
        var y=ay+((By+deltaCBy*t)-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Cx,y:Cy});
    return(pts);
}
```

Section 7.5: Finding points along a line

This example finds an array of approximately evenly spaced points along a line.

It decomposes Path segments created with `context.lineTo` into points along that line.

```
// Return: an array of approximately evenly spaced points along a line
//
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By: end points defining the line
//
function plotLine(pxTolerance,Ax,Ay,Bx,By){
    var dx=Bx-Ax;
    var dy=By-Ay;
    var ptCount=parseInt(Math.sqrt(dx*dx+dy*dy))*3;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<=ptCount;i++){
        var t=i/ptCount;
        var x=Ax+dx*t;
        var y=Ay+dy*t;
        var dx1=x-lastX;
        var dy1=y-lastY;
        if(dx1*dx1+dy1*dy1>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Bx,y:By});
    return(pts);
}
```

Section 7.6: Finding points along an entire Path containing curves and lines

This example finds an array of approximately evenly spaced points along an entire Path.

It decomposes all Path segments created with `context.lineTo`, `context.quadraticCurveTo` and/or
`context.bezierCurveTo` into points along that Path.

用法

```
// 路径相关变量
var A={x:50,y:100};
var B={x:125,y:25};
var BB={x:150,y:15};
var BB2={x:150,y:185};
var C={x:175,y:200};
var D={x:300,y:150};
var n=1000;
var tolerance=1.5;
var pts;

// 与画布相关的变量
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;

// 告诉上下文除了绘制路径外还要绘制航点

plotPathCommands(ctx,n,tolerance);

// 路径绘制命令
ctx.beginPath();
ctx.moveTo(A.x,A.y);
ctx.bezierCurveTo(B.x,B.y,C.x,C.y,D.x,D.y);
ctx.quadraticCurveTo(BB.x,BB.y,A.x,A.y);
ctx.lineTo(D.x,D.y);
ctx.strokeStyle='gray';
ctx.stroke();

// 告诉上下文停止绘制路径点
ctx.stopPlottingPathCommands();

// 演示：使用绘制的点逐步绘制路径
ptsToRects(ctx.getPathPoints());
function ptsToRects(pts){
     ctx.fillStyle='红色';
      var i=0;
requestAnimationFrame(animate);
    function animate(){
ctx.fillRect(pts[i].x-0.50,pts[i].y-0.50,tolerance,tolerance);
        i++;
        if(i<pts.length){ requestAnimationFrame(animate); }
    }
}
```

一个自动计算路径上点的插件

这段代码修改了这些 Canvas 上下文的绘图命令，使得命令不仅绘制线条或曲线，还会沿整个路径创建一个点的数组：

- beginPath,
- moveTo,
- lineTo,
- quadraticCurveTo,
- bezierCurveTo.

Usage

```
// Path related variables
var A={x:50,y:100};
var B={x:125,y:25};
var BB={x:150,y:15};
var BB2={x:150,y:185};
var C={x:175,y:200};
var D={x:300,y:150};
var n=1000;
var tolerance=1.5;
var pts;

// canvas related variables
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;

// Tell the Context to plot waypoint in addition to
// drawing the path
plotPathCommands(ctx,n,tolerance);

// Path drawing commands
ctx.beginPath();
ctx.moveTo(A.x,A.y);
ctx.bezierCurveTo(B.x,B.y,C.x,C.y,D.x,D.y);
ctx.quadraticCurveTo(BB.x,BB.y,A.x,A.y);
ctx.lineTo(D.x,D.y);
ctx.strokeStyle='gray';
ctx.stroke();

// Tell the Context to stop plotting waypoints
ctx.stopPlottingPathCommands();

// Demo: Incrementally draw the path using the plotted points
ptsToRects(ctx.getPathPoints());
function ptsToRects(pts){
    ctx.fillStyle='red';
    var i=0;
    requestAnimationFrame(animate);
    function animate(){
        ctx.fillRect(pts[i].x-0.50,pts[i].y-0.50,tolerance,tolerance);
        i++;
        if(i<pts.length){ requestAnimationFrame(animate); }
    }
}
```

An plug-in that automatically calculates points along the path

This code modifies these Canvas Context's drawing commands so the commands not only draw the line or curve, but also create an array of points along the entire path:

- beginPath,
- moveTo,
- lineTo,
- quadraticCurveTo,
- bezierCurveTo.

重要提示！

此代码修改了 Context 的实际绘图函数，因此当你完成沿路径绘制点后，应调用提供的stopPlottingPathCommands以将 Context 绘图函数恢复到未修改状态。

修改后的 Context 目的是允许你将点数组计算"插入"到现有代码中，而无需修改现有的路径绘制命令。但你不必使用此修改后的 Context——你也可以单独调用分解直线、二次曲线和三次贝塞尔曲线的各个函数，然后手动将这些单独的点数组合并成整个路径的单一点数组。

你可以使用提供的getPathPoints函数获取结果点数组的副本。

如果你使用修改后的 Context 绘制多个路径，点数组将包含所有多个路径绘制的点的单一合并集合。

如果你想获取单独的点数组，可以先用getPathPoints获取当前数组，然后用提供的clearPathPoints函数清除数组中的点。

```
// 修改 Canvas 的上下文以计算一组大致均匀分布的路径点，同时绘制路径。

function plotPathCommands(ctx,sampleCount,pointSpacing){
    ctx.mySampleCount=sampleCount;
ctx.myPointSpacing=pointSpacing;
    ctx.myTolerance=pointSpacing*pointSpacing;
    ctx.myBeginPath=ctx.beginPath;
ctx.myMoveTo=ctx.moveTo;
    ctx.myLineTo=ctx.lineTo;
    ctx.myQuadraticCurveTo=ctx.quadraticCurveTo;
    ctx.myBezierCurveTo=ctx.bezierCurveTo;
    // 不要直接使用 myPathPoints[] -- 使用 "ctx.getPathPoints"
ctx.myPathPoints=[];
    ctx.beginPath=function(){
        this.myLastX=0;
        this.myLastY=0;
        this.myBeginPath();
    }
ctx.moveTo=function(x,y){
        this.myLastX=x;
        this.myLastY=y;
        this.myMoveTo(x,y);
    }
ctx.lineTo=function(x,y){
        var pts=plotLine(this.myTolerance,this.myLastX,this.myLastY,x,y);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x;
        this.myLastY=y;
        this.myLineTo(x,y);
    }
ctx.quadraticCurveTo=function(x0,y0,x1,y1){
        var
pts=plotQBez(this.mySampleCount,this.myTolerance,this.myLastX,this.myLastY,x0,y0,x1,y1);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x1;
        this.myLastY=y1;
        this.myQuadraticCurveTo(x0,y0,x1,y1);
    }
ctx.bezierCurveTo=function(x0,y0,x1,y1,x2,y2){
        var
```

Important Note!

This code modifies the actual drawing functions of the Context so when you are done plotting points along the path, you should call the supplied `stopPlottingPathCommands` to return the Context drawing functions to their unmodified state.

The purpose of this modified Context is to allow you to "plug-in" the points-array calculation into your existing code without having to modify your existing Path drawing commands. But, you don't need to use this modified Context -- you can separately call the individual functions that decompose a line, a quadratic curve and a cubic Bezier curve and then manually concatenate those individual point-arrays into a single point-array for the entire path.

You fetch a copy of the resulting points-array using the supplied `getPathPoints` function.

If you draw multiple Paths with the modified Context, the points-array will contain a single concatenated set of points for all the multiple Paths drawn.

If, instead, you want to get separate points-arrays, you can fetch the current array with `getPathPoints` and then clear those points from the array with the supplied `clearPathPoints` function.

```
// Modify the Canvas' Context to calculate a set of approximately
//     evenly spaced waypoints as it draws path(s).
function plotPathCommands(ctx,sampleCount,pointSpacing){
    ctx.mySampleCount=sampleCount;
    ctx.myPointSpacing=pointSpacing;
    ctx.myTolerance=pointSpacing*pointSpacing;
    ctx.myBeginPath=ctx.beginPath;
    ctx.myMoveTo=ctx.moveTo;
    ctx.myLineTo=ctx.lineTo;
    ctx.myQuadraticCurveTo=ctx.quadraticCurveTo;
    ctx.myBezierCurveTo=ctx.bezierCurveTo;
    // don't use myPathPoints[] directly -- use "ctx.getPathPoints"
    ctx.myPathPoints=[];
    ctx.beginPath=function(){
        this.myLastX=0;
        this.myLastY=0;
        this.myBeginPath();
    }
    ctx.moveTo=function(x,y){
        this.myLastX=x;
        this.myLastY=y;
        this.myMoveTo(x,y);
    }
    ctx.lineTo=function(x,y){
        var pts=plotLine(this.myTolerance,this.myLastX,this.myLastY,x,y);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x;
        this.myLastY=y;
        this.myLineTo(x,y);
    }
    ctx.quadraticCurveTo=function(x0,y0,x1,y1){
        var
pts=plotQBez(this.mySampleCount,this.myTolerance,this.myLastX,this.myLastY,x0,y0,x1,y1);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x1;
        this.myLastY=y1;
        this.myQuadraticCurveTo(x0,y0,x1,y1);
    }
    ctx.bezierCurveTo=function(x0,y0,x1,y1,x2,y2){
        var
pts=plotCBez(this.mySampleCount,this.myTolerance,this.myLastX,this.myLastY,x0,y0,x1,y1,x2,y2);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x2;
        this.myLastY=y2;
        this.myBezierCurveTo(x0,y0,x1,y1,x2,y2);
    }
ctx.getPathPoints=function(){
        return(this.myPathPoints.slice());
    }
ctx.clearPathPoints=function(){
        this.myPathPoints.length=0;
    }
ctx.stopPlottingPathCommands=function(){
        if(!this.myBeginPath){return;}
        this.beginPath=this.myBeginPath;
        this.moveTo=this.myMoveTo;
        this.lineTo=this.myLineTo;
        this.quadraticCurveto=this.myQuadraticCurveTo;
        this.bezierCurveTo=this.myBezierCurveTo;
        this.myBeginPath=undefined;
    }
}
```

一个完整的演示：

```
// 路径相关变量
var A={x:50,y:100};
var B={x:125,y:25};
var BB={x:150,y:15};
var BB2={x:150,y:185};
var C={x:175,y:200};
var D={x:300,y:150};
var n=1000;
var tolerance=1.5;
var pts;

// 与画布相关的变量
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;

// 告诉上下文除了绘制路径外还要绘制航点

plotPathCommands(ctx,n,tolerance);

// 路径绘制命令
ctx.beginPath();
ctx.moveTo(A.x,A.y);
ctx.bezierCurveTo(B.x,B.y,C.x,C.y,D.x,D.y);
ctx.quadraticCurveTo(BB.x,BB.y,A.x,A.y);
ctx.lineTo(D.x,D.y);
ctx.strokeStyle='gray';
ctx.stroke();

// 告诉上下文停止绘制路径点
ctx.stopPlottingPathCommands();

// 使用绘制的点逐步绘制路径
ptsToRects(ctx.getPathPoints());
```

```
pts=plotCBez(this.mySampleCount,this.myTolerance,this.myLastX,this.myLastY,x0,y0,x1,y1,x2,y2);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x2;
        this.myLastY=y2;
        this.myBezierCurveTo(x0,y0,x1,y1,x2,y2);
    }
    ctx.getPathPoints=function(){
        return(this.myPathPoints.slice());
    }
    ctx.clearPathPoints=function(){
        this.myPathPoints.length=0;
    }
    ctx.stopPlottingPathCommands=function(){
        if(!this.myBeginPath){return;}
        this.beginPath=this.myBeginPath;
        this.moveTo=this.myMoveTo;
        this.lineTo=this.myLineTo;
        this.quadraticCurveto=this.myQuadraticCurveTo;
        this.bezierCurveTo=this.myBezierCurveTo;
        this.myBeginPath=undefined;
    }
}
```

A complete Demo:

```
// Path related variables
var A={x:50,y:100};
var B={x:125,y:25};
var BB={x:150,y:15};
var BB2={x:150,y:185};
var C={x:175,y:200};
var D={x:300,y:150};
var n=1000;
var tolerance=1.5;
var pts;

// canvas related variables
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;

// Tell the Context to plot waypoint in addition to
// drawing the path
plotPathCommands(ctx,n,tolerance);

// Path drawing commands
ctx.beginPath();
ctx.moveTo(A.x,A.y);
ctx.bezierCurveTo(B.x,B.y,C.x,C.y,D.x,D.y);
ctx.quadraticCurveTo(BB.x,BB.y,A.x,A.y);
ctx.lineTo(D.x,D.y);
ctx.strokeStyle='gray';
ctx.stroke();

// Tell the Context to stop plotting waypoints
ctx.stopPlottingPathCommands();

// Incrementally draw the path using the plotted points
ptsToRects(ctx.getPathPoints());
```

```
function ptsToRects(pts){
     ctx.fillStyle='红色';
     var i=0;
requestAnimationFrame(animate);
     function animate(){
ctx.fillRect(pts[i].x-0.50,pts[i].y-0.50,tolerance,tolerance);
        i++;
        if(i<pts.length){ requestAnimationFrame(animate); }
    }
}


///////////////////////////////////////
// 一个插件
///////////////////////////////////////

// 修改画布的上下文以计算一组大致均匀分布的路径点
//     在绘制路径时使用。
function plotPathCommands(ctx,sampleCount,pointSpacing){
    ctx.mySampleCount=sampleCount;
ctx.myPointSpacing=pointSpacing;
    ctx.myTolerance=pointSpacing*pointSpacing;
    ctx.myBeginPath=ctx.beginPath;
ctx.myMoveTo=ctx.moveTo;
    ctx.myLineTo=ctx.lineTo;
    ctx.myQuadraticCurveTo=ctx.quadraticCurveTo;
    ctx.myBezierCurveTo=ctx.bezierCurveTo;
    // 不要直接使用 myPathPoints[] -- 使用 "ctx.getPathPoints"
ctx.myPathPoints=[];
    ctx.beginPath=function(){
        this.myLastX=0;
        this.myLastY=0;
        this.myBeginPath();
    }
ctx.moveTo=function(x,y){
        this.myLastX=x;
        this.myLastY=y;
        this.myMoveTo(x,y);
    }
ctx.lineTo=function(x,y){
        var pts=plotLine(this.myTolerance,this.myLastX,this.myLastY,x,y);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x;
        this.myLastY=y;
        this.myLineTo(x,y);
    }
ctx.quadraticCurveTo=function(x0,y0,x1,y1){
        var
pts=plotQBez(this.mySampleCount,this.myTolerance,this.myLastX,this.myLastY,x0,y0,x1,y1);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x1;
        this.myLastY=y1;
        this.myQuadraticCurveTo(x0,y0,x1,y1);
    }
ctx.bezierCurveTo=function(x0,y0,x1,y1,x2,y2){
        var
pts=plotCBez(this.mySampleCount,this.myTolerance,this.myLastX,this.myLastY,x0,y0,x1,y1,x2,y2);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x2;
        this.myLastY=y2;
        this.myBezierCurveTo(x0,y0,x1,y1,x2,y2);
    }
```

```
function ptsToRects(pts){
    ctx.fillStyle='red';
    var i=0;
    requestAnimationFrame(animate);
    function animate(){
        ctx.fillRect(pts[i].x-0.50,pts[i].y-0.50,tolerance,tolerance);
        i++;
        if(i<pts.length){ requestAnimationFrame(animate); }
    }
}


///////////////////////////////////////
// A Plug-in
///////////////////////////////////////

// Modify the Canvas' Context to calculate a set of approximately
//     evenly spaced waypoints as it draws path(s).
function plotPathCommands(ctx,sampleCount,pointSpacing){
    ctx.mySampleCount=sampleCount;
    ctx.myPointSpacing=pointSpacing;
    ctx.myTolerance=pointSpacing*pointSpacing;
    ctx.myBeginPath=ctx.beginPath;
    ctx.myMoveTo=ctx.moveTo;
    ctx.myLineTo=ctx.lineTo;
    ctx.myQuadraticCurveTo=ctx.quadraticCurveTo;
    ctx.myBezierCurveTo=ctx.bezierCurveTo;
    // don't use myPathPoints[] directly -- use "ctx.getPathPoints"
    ctx.myPathPoints=[];
    ctx.beginPath=function(){
        this.myLastX=0;
        this.myLastY=0;
        this.myBeginPath();
    }
    ctx.moveTo=function(x,y){
        this.myLastX=x;
        this.myLastY=y;
        this.myMoveTo(x,y);
    }
    ctx.lineTo=function(x,y){
        var pts=plotLine(this.myTolerance,this.myLastX,this.myLastY,x,y);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x;
        this.myLastY=y;
        this.myLineTo(x,y);
    }
    ctx.quadraticCurveTo=function(x0,y0,x1,y1){
        var
pts=plotQBez(this.mySampleCount,this.myTolerance,this.myLastX,this.myLastY,x0,y0,x1,y1);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x1;
        this.myLastY=y1;
        this.myQuadraticCurveTo(x0,y0,x1,y1);
    }
    ctx.bezierCurveTo=function(x0,y0,x1,y1,x2,y2){
        var
pts=plotCBez(this.mySampleCount,this.myTolerance,this.myLastX,this.myLastY,x0,y0,x1,y1,x2,y2);
        Array.prototype.push.apply(this.myPathPoints,pts);
        this.myLastX=x2;
        this.myLastY=y2;
        this.myBezierCurveTo(x0,y0,x1,y1,x2,y2);
    }
```

```
ctx.getPathPoints=function(){
        return(this.myPathPoints.slice());
    }
ctx.clearPathPoints=function(){
        this.myPathPoints.length=0;
    }
ctx.stopPlottingPathCommands=function(){
        if(!this.myBeginPath){return;}
        this.beginPath=this.myBeginPath;
        this.moveTo=this.myMoveTo;
        this.lineTo=this.myLineTo;
        this.quadraticCurveto=this.myQuadraticCurveTo;
        this.bezierCurveTo=this.myBezierCurveTo;
        this.myBeginPath=undefined;
    }
}


////////////////////////////////
// 辅助函数
////////////////////////////////

// 返回：沿三次贝塞尔曲线大致均匀分布的一组点
//
// 归属：Stackoverflow的@Blindman67
// 引用：
http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-moving-circle-
along-it/36827074#36827074
// 根据上述引用修改
//
// ptCount：沿曲线采样的点数
// pxTolerance：点之间允许的近似间距
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy：定义曲线的控制点
//
function plotCBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy,Dx,Dy){
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaDCx=Dx-Cx;
    var deltaBAy=By-Ay;
    var deltaCBy=Cy-By;
    var deltaDCy=Dy-Cy;
    var ax,ay,bx,by;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
        var t=i/ptCount;
ax=Ax+deltaBAx*t;
        bx=Bx+deltaCBx*t;
        cx=Cx+deltaDCx*t;
        ax+=(bx-ax)*t;
        bx+=(cx-bx)*t;
        //
ay=Ay+deltaBAy*t;
        by=By+deltaCBy*t;
        cy=Cy+deltaDCy*t;
        ay+=(by-ay)*t;
        by+=(cy-by)*t;
        var x=ax+(bx-ax)*t;
        var y=ay+(by-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
```

```
    ctx.getPathPoints=function(){
        return(this.myPathPoints.slice());
    }
    ctx.clearPathPoints=function(){
        this.myPathPoints.length=0;
    }
    ctx.stopPlottingPathCommands=function(){
        if(!this.myBeginPath){return;}
        this.beginPath=this.myBeginPath;
        this.moveTo=this.myMoveTo;
        this.lineTo=this.myLineTo;
        this.quadraticCurveto=this.myQuadraticCurveTo;
        this.bezierCurveTo=this.myBezierCurveTo;
        this.myBeginPath=undefined;
    }
}


////////////////////////////////
// Helper functions
////////////////////////////////

// Return: a set of approximately evenly spaced points along a cubic Bezier curve
//
// Attribution: Stackoverflow's @Blindman67
// Cite:
http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-moving-circle-
along-it/36827074#36827074
// As modified from the above citation
//
// ptCount: sample this many points at interval along the curve
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy: control points defining the curve
//
function plotCBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy,Dx,Dy){
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaDCx=Dx-Cx;
    var deltaBAy=By-Ay;
    var deltaCBy=Cy-By;
    var deltaDCy=Dy-Cy;
    var ax,ay,bx,by;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
        var t=i/ptCount;
        ax=Ax+deltaBAx*t;
        bx=Bx+deltaCBx*t;
        cx=Cx+deltaDCx*t;
        ax+=(bx-ax)*t;
        bx+=(cx-bx)*t;
        //
        ay=Ay+deltaBAy*t;
        by=By+deltaCBy*t;
        cy=Cy+deltaDCy*t;
        ay+=(by-ay)*t;
        by+=(cy-by)*t;
        var x=ax+(bx-ax)*t;
        var y=ay+(by-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
lastY=y;
        }
    }
pts.push({x:Dx,y:Dy});
    return(pts);
}

// 返回：沿二次曲线大致均匀分布的点数组
//
// 归属：Stackoverflow的@Blindman67
// 引用：
http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-moving-circle-
along-it/36827074#36827074
// 根据上述引用修改
//
// ptCount：沿曲线以间隔采样的点数
// pxTolerance：点之间允许的近似间距
// Ax,Ay,Bx,By,Cx,Cy：定义曲线的控制点
//
function plotQBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy){
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaBAy=By-Ay;
    var deltaCBy=Cy-By;
    var ax,ay;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
        var t=i/ptCount;
ax=Ax+deltaBAx*t;
        ay=Ay+deltaBAy*t;
        var x=ax+((Bx+deltaCBx*t)-ax)*t;
        var y=ay+((By+deltaCBy*t)-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
lastY=y;
        }
    }
pts.push({x:Cx,y:Cy});
    return(pts);
}

// 返回：沿直线大致均匀分布的点数组
//
// pxTolerance：点之间允许的大致间距
// Ax,Ay,Bx,By：定义直线的端点
//
function plotLine(pxTolerance,Ax,Ay,Bx,By){
    var dx=Bx-Ax;
    var dy=By-Ay;
    var ptCount=parseInt(Math.sqrt(dx*dx+dy*dy))*3;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<=ptCount;i++){
```

```
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Dx,y:Dy});
    return(pts);
}

// Return: an array of approximately evenly spaced points along a Quadratic curve
//
// Attribution: Stackoverflow's @Blindman67
// Cite:
http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-moving-circle-
along-it/36827074#36827074
// As modified from the above citation
//
// ptCount: sample this many points at interval along the curve
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By,Cx,Cy: control points defining the curve
//
function plotQBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy){
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaBAy=By-Ay;
    var deltaCBy=Cy-By;
    var ax,ay;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
        var t=i/ptCount;
        ax=Ax+deltaBAx*t;
        ay=Ay+deltaBAy*t;
        var x=ax+((Bx+deltaCBx*t)-ax)*t;
        var y=ay+((By+deltaCBy*t)-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Cx,y:Cy});
    return(pts);
}

// Return: an array of approximately evenly spaced points along a line
//
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By: end points defining the line
//
function plotLine(pxTolerance,Ax,Ay,Bx,By){
    var dx=Bx-Ax;
    var dy=By-Ay;
    var ptCount=parseInt(Math.sqrt(dx*dx+dy*dy))*3;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<=ptCount;i++){
        var t=i/ptCount;
        var x=Ax+dx*t;
        var y=Ay+dy*t;
        var dx1=x-lastX;
        var dy1=y-lastY;
        if(dx1*dx1+dy1*dy1>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
lastY=y;
        }
    }
pts.push({x:Bx,y:By});
    return(pts);
}
```

第7.7节：在位置拆分贝塞尔曲线

此示例将三次和贝塞尔曲线拆分为两部分。

函数 splitCurveAt 在 position 处拆分曲线，其中 0.0 = 起点， 0.5 = 中点， 1 = 终点。它可以拆分二次和三次曲线。曲线类型由最后一个 x 参数 x4 决定。如果不是 **undefined** 或 **null**，则假定曲线为三次曲线，否则为二次曲线。

示例用法

将二次贝塞尔曲线拆分为两部分

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurves = splitCurveAt(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;
var p = newCurves
// 绘制两个新曲线
// 假设 ctx 是 canvas 2d 上下文
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

将三次贝塞尔曲线分割成两段

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var p4 = {x : 300, y : 100};
var newCurves = splitCurveAt(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y)

var i = 0;
var p = newCurves
// 绘制两个新曲线
// 假设 ctx 是 canvas 2d 上下文
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
```

```
        var t=i/ptCount;
        var x=Ax+dx*t;
        var y=Ay+dy*t;
        var dx1=x-lastX;
        var dy1=y-lastY;
        if(dx1*dx1+dy1*dy1>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Bx,y:By});
    return(pts);
}
```

Section 7.7: Split bezier curves at position

This example splits cubic and bezier curves in two.

The function `splitCurveAt` splits the curve at `position` where `0.0` = start, `0.5` = middle, and 1 = end. It can split quadratic and cubic curves. The curve type is determined by the last x argument x4. If not **undefined** or **null** then it assumes the curve is cubic else the curve is a quadratic

Example usage

Splitting quadratic bezier curve in two

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurves = splitCurveAt(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;
var p = newCurves
// Draw the 2 new curves
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

Splitting cubic bezier curve in two

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var p4 = {x : 300, y : 100};
var newCurves = splitCurveAt(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y)

var i = 0;
var p = newCurves
// Draw the 2 new curves
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.bezierCurveTo(p[i++], p[i++], p[i++], p[i++], p[i++], p[i++]);
ctx.bezierCurveTo(p[i++], p[i++], p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

分割函数

splitCurveAt = function(position, x1, y1, x2, y2, x3, y3, [x4, y4])

> 注意： 参数[x4, y4]是可选的。

> 注意：该函数包含一些可选的注释代码/* */，用于处理结果曲线可能为零长度，或落在原始曲线起点或终点之外的边界情况。尝试在position < 0或position > 1的有效范围之外分割曲线将抛出范围错误。
> 该代码可以移除，函数仍能正常工作，但可能会得到零长度的曲线。

```
// 如果 position 不在 0 < position < 1 范围内则抛出 RangeError
// 二次曲线的参数为 x1, y1, x2, y2, x3, y3
// 三次曲线的参数为 x1, y1, x2, y2, x3, y3, x4, y4
                              // 返回一个点数组，表示两条曲线。曲线类型与被分割的曲线相同
var splitCurveAt = function(position, x1, y1, x2, y2, x3, y3, x4, y4){
    var v1, v2, v3, v4, quad, retPoints, i, c;

    // =============================================================================================
    // 你可以删除这部分，因为函数仍然可以正常工作，生成的曲线也会正常渲染
    // 但其他曲线函数可能不喜欢长度为0的曲线
    // =============================================================================================
    if(position <= 0 || position >= 1){
        throw RangeError("spliteCurveAt 需要 position > 0 且 position < 1");
    }

    // =============================================================================================
    // 如果移除上述范围错误，您可以使用以下注释的一个或两个部分
sections
    // 分割曲线时 position < 0 或 position > 1 仍会创建有效曲线，但它们会
    // 超出端点范围

    // =============================================================================================
    // 锁定曲线上分割的位置。
    /* optional A
position = position < 0 ? 0 : position > 1 ? 1 : position;
    optional A end */

    // =============================================================================================
    // 接下来的注释部分将在分割结果为零长度曲线时返回原始曲线

    // 如果您需要此功能，可以取消注释
    /*  可选 B
if(position <= 0 || position >= 1){
        if(x4 === undefined || x4 === null){
            return [x1, y1, x2, y2, x3, y3];
        }else{
return [x1, y1, x2, y2, x3, y3, x4, y4];
        }
    }
```

```
ctx.moveTo(p[i++],p[i++]);
ctx.bezierCurveTo(p[i++], p[i++], p[i++], p[i++], p[i++], p[i++]);
ctx.bezierCurveTo(p[i++], p[i++], p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

The split function

splitCurveAt = function(position, x1, y1, x2, y2, x3, y3, [x4, y4])

> **Note:** Arguments inside [x4, y4] are optional.

> **Note:** The function has some optional commented `/* */` code that deals with edge cases where the resulting curves may have zero length, or fall outside the start or ends of the original curve. As is attempting to split a curve outside the valid range for `position >= 0` or `position >= 1` will throw a range error. This can be removed and will work just fine, though you may have resulting curves that have zero length.

```
// With throw RangeError if not 0 < position < 1
// x1, y1, x2, y2, x3, y3 for quadratic curves
// x1, y1, x2, y2, x3, y3, x4, y4 for cubic curves
// Returns an array of points representing 2 curves. The curves are the same type as the split curve
var splitCurveAt = function(position, x1, y1, x2, y2, x3, y3, x4, y4){
    var v1, v2, v3, v4, quad, retPoints, i, c;

    // =============================================================================================
    // you may remove this as the function will still work and resulting curves will still render
    // but other curve functions may not like curves with 0 length
    // =============================================================================================
    if(position <= 0 || position >= 1){
        throw RangeError("spliteCurveAt requires position > 0 && position < 1");
    }

    // =============================================================================================
    // If you remove the above range error you may use one or both of the following commented
sections
    // Splitting curves position < 0 or position > 1 will still create valid curves but they will
    // extend past the end points

    // =============================================================================================
    // Lock the position to split on the curve.
    /* optional A
    position = position < 0 ? 0 : position > 1 ? 1 : position;
    optional A end */

    // =============================================================================================
    // the next commented section will return the original curve if the split results in 0 length
curve
    // You may wish to uncomment this If you desire such functionality
    /*  optional B
    if(position <= 0 || position >= 1){
        if(x4 === undefined || x4 === null){
            return [x1, y1, x2, y2, x3, y3];
        }else{
            return [x1, y1, x2, y2, x3, y3, x4, y4];
        }
    }
```

```
可选 B 结束 */


retPoints = []; // 坐标数组
    i = 0;
quad = false;  // 假设为三次贝塞尔曲线
    v1 = {};
v2 = {};
     v4 = {};
     v1.x = x1;
     v1.y = y1;
     v2.x = x2;
     v2.y = y2;
     if(x4 === undefined || x4 === null){
         quad = true;  // 这是一个二次贝塞尔曲线
         v4.x = x3;
v4.y = y3;
     }else{
v3 = {};
          v3.x = x3;
          v3.y = y3;
          v4.x = x4;
          v4.y = y4;
     }
c = position;
retPoints[i++] = v1.x;  // 起点
    retPoints[i++] = v1.y;

     if(quad){ // 分割二次贝塞尔曲线
retPoints[i++] = (v1.x += (v2.x - v1.x) * c);  // 第一条曲线的新控制点
         retPoints[i++] = (v1.y += (v2.y - v1.y) * c);
v2.x += (v4.x - v2.x) * c;
v2.y += (v4.y - v2.y) * c;
retPoints[i++] = v1.x + (v2.x - v1.x) * c;  // 第一条和第二条曲线的新终点和起点
         retPoints[i++] = v1.y + (v2.y - v1.y) * c;
retPoints[i++] = v2.x;  // 第二条曲线的新控制点
         retPoints[i++] = v2.y;
retPoints[i++] = v4.x;  // 第二条曲线的新终点
         retPoints[i++] = v4.y;
          //=======================================================
          // 返回包含两条曲线的数组
          return retPoints;
     }
retPoints[i++] = (v1.x += (v2.x - v1.x) * c); // 第一条曲线第一个控制点
    retPoints[i++] = (v1.y += (v2.y - v1.y) * c);
v2.x += (v3.x - v2.x) * c;
v2.y += (v3.y - v2.y) * c;
v3.x += (v4.x - v3.x) * c;
v3.y += (v4.y - v3.y) * c;
retPoints[i++] = (v1.x += (v2.x - v1.x) * c); // 第一条曲线第二个控制点
    retPoints[i++] = (v1.y += (v2.y - v1.y) * c);
v2.x += (v3.x - v2.x) * c;
v2.y += (v3.y - v2.y) * c;
retPoints[i++] = v1.x + (v2.x - v1.x) * c; // 第一条曲线第二段的结束点和起点
    retPoints[i++] = v1.y + (v2.y - v1.y) * c;
retPoints[i++] = v2.x;  // 第二条曲线第一个控制点
    retPoints[i++] = v2.y;
retPoints[i++] = v3.x;  // 第二条曲线第二个控制点
    retPoints[i++] = v3.y;
retPoints[i++] = v4.x;  // 第二条曲线的终点
    retPoints[i++] = v4.y;
     //=======================================================
```

```
    optional B end */


    retPoints = []; // array of coordinates
    i = 0;
    quad = false;  // presume cubic bezier
    v1 = {};
    v2 = {};
    v4 = {};
    v1.x = x1;
    v1.y = y1;
    v2.x = x2;
    v2.y = y2;
    if(x4 === undefined || x4 === null){
        quad = true;  // this is a quadratic bezier
        v4.x = x3;
        v4.y = y3;
    }else{
        v3 = {};
        v3.x = x3;
        v3.y = y3;
        v4.x = x4;
        v4.y = y4;
    }
    c = position;
    retPoints[i++] = v1.x;  // start point
    retPoints[i++] = v1.y;

    if(quad){ // split quadratic bezier
        retPoints[i++] = (v1.x += (v2.x - v1.x) * c);  // new control point for first curve
        retPoints[i++] = (v1.y += (v2.y - v1.y) * c);
        v2.x += (v4.x - v2.x) * c;
        v2.y += (v4.y - v2.y) * c;
        retPoints[i++] = v1.x + (v2.x - v1.x) * c;  // new end and start of first and second curves
        retPoints[i++] = v1.y + (v2.y - v1.y) * c;
        retPoints[i++] = v2.x;  // new control point for second curve
        retPoints[i++] = v2.y;
        retPoints[i++] = v4.x;  // new endpoint of second curve
        retPoints[i++] = v4.y;
        //=======================================================
        // return array with 2 curves
        return retPoints;
    }
    retPoints[i++] = (v1.x += (v2.x - v1.x) * c); // first curve first control point
    retPoints[i++] = (v1.y += (v2.y - v1.y) * c);
    v2.x += (v3.x - v2.x) * c;
    v2.y += (v3.y - v2.y) * c;
    v3.x += (v4.x - v3.x) * c;
    v3.y += (v4.y - v3.y) * c;
    retPoints[i++] = (v1.x += (v2.x - v1.x) * c); // first curve second control point
    retPoints[i++] = (v1.y += (v2.y - v1.y) * c);
    v2.x += (v3.x - v2.x) * c;
    v2.y += (v3.y - v2.y) * c;
    retPoints[i++] = v1.x + (v2.x - v1.x) * c; // end and start point of first second curves
    retPoints[i++] = v1.y + (v2.y - v1.y) * c;
    retPoints[i++] = v2.x;  // second curve first control point
    retPoints[i++] = v2.y;
    retPoints[i++] = v3.x;  // second curve second control point
    retPoints[i++] = v3.y;
    retPoints[i++] = v4.x;  // endpoint of second curve
    retPoints[i++] = v4.y;
    //=======================================================
    // 返回包含2条曲线的数组
    return retPoints;
}
```

第7.8节：修剪贝塞尔曲线

本示例向您展示如何修剪贝塞尔曲线。

函数trimBezier修剪曲线的两端，返回从fromPos到toPos的曲线。fromPos和toPos的取值范围是0到1（含）。它可以修剪二次和三次曲线。曲线类型由最后一个x参数 x4决定。如果不是undefined或null，则假定曲线为三次曲线，否则为二次曲线。

修剪后的曲线以点数组形式返回。二次曲线返回6个点，三次曲线返回8个点。

示例用法

修剪二次曲线。

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurve = splitCurveAt(0.25, 0.75, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;
var p = newCurve
// 绘制修剪后的曲线
// 假设 ctx 是 canvas 2d 上下文
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

修剪三次曲线。

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var p4 = {x : 300, y : 100};
var newCurve = splitCurveAt(0.25, 0.75, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y)

var i = 0;
var p = newCurve
// 绘制修剪后的曲线
// 假设 ctx 是 canvas 2d 上下文
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.bezierCurveTo(p[i++], p[i++], p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

示例函数

trimBezier = function(fromPos, toPos, x1, y1, x2, y2, x3, y3, [x4, y4])

```
    // return array with 2 curves
    return retPoints;
}
```

Section 7.8: Trim bezier curve

This example show you how to trim a bezier.

The function trimBezier trims the ends off of the curve returning the curve `fromPos` to `toPos`. `fromPos` and `toPos` are in the range 0 to 1 inclusive, It can trim quadratic and cubic curves. The curve type is determined by the last x argument x4. If not `undefined` or `null` then it assumes the curve is cubic else the curve is a quadratic

The trimmed curve is returned as an array of points. 6 points for quadratic curves and 8 for cubic curves.

Example Usage

Trimming a quadratic curve.

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurve = splitCurveAt(0.25, 0.75, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;
var p = newCurve
// Draw the trimmed curve
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

Trimming a cubic curve.

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var p4 = {x : 300, y : 100};
var newCurve = splitCurveAt(0.25, 0.75, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y)

var i = 0;
var p = newCurve
// Draw the trimmed curve
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.bezierCurveTo(p[i++], p[i++], p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

Example Function

trimBezier = function(fromPos, toPos, x1, y1, x2, y2, x3, y3, [x4, y4])

注意： 参数[x4, y4]是可选的。

注意： 此函数依赖本节中示例“Split Bezier Curves At”中的函数

```
var trimBezier = function(fromPos, toPos, x1, y1, x2, y2, x3, y3, x4, y4){
    var quad, i, s, retBez;
quad = false;
    if(x4 === undefined || x4 === null){
       quad = true;  // 这是一个二次贝塞尔曲线
    }
    if(fromPos > toPos){ // 如果起点在终点之后，则交换
       i = fromPos;
fromPos = toPos
toPos = i;
    }
    // 限制在曲线上
toPos = toPos <= 0 ? 0 : toPos >= 1 ? 1 : toPos;
    fromPos = fromPos <= 0 ? 0 : fromPos >= 1 ? 1 : fromPos;
    if(toPos === fromPos){
s = splitBezierAt(toPos, x1, y1, x2, y2, x3, y3, x4, y4);
        i = quad ? 4 : 6;
retBez = [s[i], s[i+1], s[i], s[i+1], s[i], s[i+1]];
        if(!quad){
retBez.push(s[i], s[i+1]);
        }
        return retBez;
    }
    if(toPos === 1 && fromPos === 0){       // 无需修剪
        retBez = [x1, y1, x2, y2, x3, y3];  // 返回原始贝塞尔曲线
        if(!quad){
retBez.push(x4, y4);
        }
        return retBez;
    }
    if(fromPos === 0){
        if(toPos < 1){
s = splitBezierAt(toPos, x1, y1, x2, y2, x3, y3, x4, y4);
            i = 0;
retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
            if(!quad){
retBez.push(s[i++], s[i++]);
            }
        }
        return retBez;
    }
    if(toPos === 1){
        if(fromPos < 1){
s = splitBezierAt(toPos, x1, y1, x2, y2, x3, y3, x4, y4);
            i = quad ? 4 : 6;
retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
            if(!quad){
retBez.push(s[i++], s[i++]);
            }
        }
        return retBez;
    }
s = splitBezierAt(fromPos, x1, y1, x2, y2, x3, y3, x4, y4);
    if(quad){
        i = 4;
```

Note: Arguments inside [x4, y4] are optional.

Note: This function requires the function in the example Split Bezier Curves At in this section

```
var trimBezier = function(fromPos, toPos, x1, y1, x2, y2, x3, y3, x4, y4){
    var quad, i, s, retBez;
    quad = false;
    if(x4 === undefined || x4 === null){
        quad = true;  // this is a quadratic bezier
    }
    if(fromPos > toPos){ // swap is from is after to
        i = fromPos;
        fromPos = toPos
        toPos = i;
    }
    // clamp to on the curve
    toPos = toPos <= 0 ? 0 : toPos >= 1 ? 1 : toPos;
    fromPos = fromPos <= 0 ? 0 : fromPos >= 1 ? 1 : fromPos;
    if(toPos === fromPos){
        s = splitBezierAt(toPos, x1, y1, x2, y2, x3, y3, x4, y4);
        i = quad ? 4 : 6;
        retBez = [s[i], s[i+1], s[i], s[i+1], s[i], s[i+1]];
        if(!quad){
            retBez.push(s[i], s[i+1]);
        }
        return retBez;
    }
    if(toPos === 1 && fromPos === 0){       // no trimming required
        retBez = [x1, y1, x2, y2, x3, y3];  // return original bezier
        if(!quad){
            retBez.push(x4, y4);
        }
        return retBez;
    }
    if(fromPos === 0){
        if(toPos < 1){
            s = splitBezierAt(toPos, x1, y1, x2, y2, x3, y3, x4, y4);
            i = 0;
            retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
            if(!quad){
                retBez.push(s[i++], s[i++]);
            }
        }
        return retBez;
    }
    if(toPos === 1){
        if(fromPos < 1){
            s = splitBezierAt(toPos, x1, y1, x2, y2, x3, y3, x4, y4);
            i = quad ? 4 : 6;
            retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
            if(!quad){
                retBez.push(s[i++], s[i++]);
            }
        }
        return retBez;
    }
    s = splitBezierAt(fromPos, x1, y1, x2, y2, x3, y3, x4, y4);
    if(quad){
        i = 4;
toPos = (toPos - fromPos) / (1 - fromPos);
        s = splitBezierAt(toPos, s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]);
        i = 0;
retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
        return retBez;

    }
i = 6;
toPos = (toPos - fromPos) / (1 - fromPos);
s = splitBezierAt(toPos, s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]);
    i = 0;
retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
    return retBez;
}
```

第7.9节：三次贝塞尔曲线的长度（近似计算）

给定三次贝塞尔曲线的4个点，以下函数返回其长度。

方法： 三次贝塞尔曲线的长度没有直接的数学计算方法。此“暴力”方法通过对曲线上的点进行采样，计算这些点之间的总距离。

精度： 使用默认采样大小40时，近似长度的准确度超过99%。

```
// 返回：三次贝塞尔曲线长度的近似值
//
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy：曲线的4个控制点
// sampleCount [可选，默认=40]：计算的区间数
// 依赖：cubicQxy（见下文）
//
function cubicBezierLength(Ax,Ay,Bx,By,Cx,Cy,Dx,Dy,sampleCount){
    var ptCount=sampleCount||40;
    var totDist=0;
    var lastX=Ax;
    var lastY=Ay;
    var dx,dy;
    for(var i=1;i<ptCount;i++){
        var pt=cubicQxy(i/ptCount,Ax,Ay,Bx,By,Cx,Cy,Dx,Dy);
        dx=pt.x-lastX;
dy=pt.y-lastY;
totDist+=Math.sqrt(dx*dx+dy*dy);
        lastX=pt.x;
lastY=pt.y;
    }
dx=Dx-lastX;
dy=Dy-lastY;
totDist+=Math.sqrt(dx*dx+dy*dy);
    return(parseInt(totDist));
}


// 返回：沿三次贝塞尔曲线在间隔T处的[x,y]点
//
// 归属：Stackoverflow的@Blindman67
// 引用：
http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-moving-circle-
along-it/36827074#36827074
```

```
        toPos = (toPos - fromPos) / (1 - fromPos);
        s = splitBezierAt(toPos, s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]);
        i = 0;
        retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
        return retBez;

    }
    i = 6;
    toPos = (toPos - fromPos) / (1 - fromPos);
    s = splitBezierAt(toPos, s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]);
    i = 0;
    retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
    return retBez;
}
```

Section 7.9: Length of a Cubic Bezier Curve (a close approximation)

Given the 4 points of a cubic Bezier curve the following function returns its length.

Method: The length of a cubic Bezier curve does not have a direct mathematical calculation. This "brute force" method finds a sampling of points along the curve and calculates the total distance spanned by those points.

Accuracy: The approximate length is 99+% accurate using the default sampling size of 40.

```
// Return: Close approximation of the length of a Cubic Bezier curve
//
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy: the 4 control points of the curve
// sampleCount [optional, default=40]: how many intervals to calculate
// Requires: cubicQxy (included below)
//
function cubicBezierLength(Ax,Ay,Bx,By,Cx,Cy,Dx,Dy,sampleCount){
    var ptCount=sampleCount||40;
    var totDist=0;
    var lastX=Ax;
    var lastY=Ay;
    var dx,dy;
    for(var i=1;i<ptCount;i++){
        var pt=cubicQxy(i/ptCount,Ax,Ay,Bx,By,Cx,Cy,Dx,Dy);
        dx=pt.x-lastX;
        dy=pt.y-lastY;
        totDist+=Math.sqrt(dx*dx+dy*dy);
        lastX=pt.x;
        lastY=pt.y;
    }
    dx=Dx-lastX;
    dy=Dy-lastY;
    totDist+=Math.sqrt(dx*dx+dy*dy);
    return(parseInt(totDist));
}


// Return: an [x,y] point along a cubic Bezier curve at interval T
//
// Attribution: Stackoverflow's @Blindman67
// Cite:
http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-moving-circle-
along-it/36827074#36827074
```

```
// 根据上述引用修改
//
// t：曲线上的一个区间 (0<=t<=1)
// ax,ay,bx,by,cx,cy,dx,dy：定义曲线的控制点
//
function cubicQxy(t,ax,ay,bx,by,cx,cy,dx,dy) {
    ax += (bx - ax) * t;
bx += (cx - bx) * t;
    cx += (dx - cx) * t;
    ax += (bx - ax) * t;
    bx += (cx - bx) * t;
    ay += (by - ay) * t;
    by += (cy - by) * t;
    cy += (dy - cy) * t;
    ay += (by - ay) * t;
    by += (cy - by) * t;
    return({
x:ax +(bx - ax) * t,
        y:ay +(by - ay) * t
    });
}
```

第7.10节：二次曲线的长度

给定二次曲线的3个点，以下函数返回曲线的长度。

```
function quadraticBezierLength(x1,y1,x2,y2,x3,y3)
    var a, e, c, d, u, a1, e1, c1, d1, u1, v1x, v1y;

    v1x = x2 * 2;
v1y = y2 * 2;
d = x1 - v1x + x3;
d1 = y1 - v1y + y3;
e = v1x - 2 * x1;
e1 = v1y - 2 * y1;
c1 = (a = 4 * (d * d + d1 * d1));
    c1 += (b = 4 * (d * e + d1 * e1));
    c1 += (c = e * e + e1 * e1);
c1 = 2 * Math.sqrt(c1);
    a1 = 2 * a * (u = Math.sqrt(a));
    u1 = b / u;
a = 4 * c * a - b * b;
    c = 2 * Math.sqrt(c);
    return (a1 * c1 + u * b * (c1 - c) + a * Math.log((2 * u + u1 + c1) / (u1 + c))) / (4 * a1);
}
```

源自二次贝塞尔函数 F(t) = a * (1 - t)^2 + 2 * b * (1 - t) * t + c * t^2

```
// As modified from the above citation
//
// t: an interval along the curve (0<=t<=1)
// ax,ay,bx,by,cx,cy,dx,dy: control points defining the curve
//
function cubicQxy(t,ax,ay,bx,by,cx,cy,dx,dy) {
    ax += (bx - ax) * t;
    bx += (cx - bx) * t;
    cx += (dx - cx) * t;
    ax += (bx - ax) * t;
    bx += (cx - bx) * t;
    ay += (by - ay) * t;
    by += (cy - by) * t;
    cy += (dy - cy) * t;
    ay += (by - ay) * t;
    by += (cy - by) * t;
    return({
        x:ax +(bx - ax) * t,
        y:ay +(by - ay) * t
    });
}
```

Section 7.10: Length of a Quadratic Curve

Given the 3 points of a quadratic curve the following function returns the length.

```
function quadraticBezierLength(x1,y1,x2,y2,x3,y3)
    var a, e, c, d, u, a1, e1, c1, d1, u1, v1x, v1y;

    v1x = x2 * 2;
    v1y = y2 * 2;
    d = x1 - v1x + x3;
    d1 = y1 - v1y + y3;
    e = v1x - 2 * x1;
    e1 = v1y - 2 * y1;
    c1 = (a = 4 * (d * d + d1 * d1));
    c1 += (b = 4 * (d * e + d1 * e1));
    c1 += (c = e * e + e1 * e1);
    c1 = 2 * Math.sqrt(c1);
    a1 = 2 * a * (u = Math.sqrt(a));
    u1 = b / u;
    a = 4 * c * a - b * b;
    c = 2 * Math.sqrt(c);
    return (a1 * c1 + u * b * (c1 - c) + a * Math.log((2 * u + u1 + c1) / (u1 + c))) / (4 * a1);
}
```

Derived from the quadratic bezier function F(t) = a * (1 - t)2 + 2 * b * (1 - t) * t + c * t2

第8章：在画布上拖动路径形状和图像

第8.1节：形状和图像在画布上真正的“移动”方式

一个问题：画布只记住像素，而不是形状或图像

这是一个圆形沙滩球的图片，当然，你不能在图片中拖动这个球。

你可能会感到惊讶，就像图片一样，如果你在画布上画一个圆，你也不能拖动画布上的那个圆。
这是因为画布不会记住它画圆的位置。

```
// 这个弧（==圆）是不可拖动的！！
context.beginPath();
context.arc(20, 30, 15, 0, Math.PI*2);
context.fillStyle='blue';
context.fill();
```

画布不知道的是……

- ……你画圆的位置（它不知道 x,y =[20,30]）。
- ……圆的大小（它不知道半径=15）。
- ……圆的颜色。（它不知道圆是蓝色的）。

画布知道的是……

Canvas 知道其绘图表面上每个像素的颜色。

画布可以告诉你在坐标 x,y==[20,30] 处有一个蓝色像素，但它不知道这个蓝色像素是否属于一个圆形。

这意味着……

这意味着画布上绘制的所有内容都是永久的：不可移动且不可更改。

- 画布不能移动圆形或调整圆形大小。
- 画布不能重新着色圆形或擦除圆形。
- 画布无法判断鼠标是否悬停在圆形上。
- 画布无法判断圆形是否与另一个圆形发生碰撞。
- 画布不能让用户拖动圆形在画布上移动。

Chapter 8: Dragging Path Shapes & Images on Canvas

Section 8.1: How shapes & images REALLY(!) "move" on the Canvas

A problem: Canvas only remembers pixels, not shapes or images

This is an image of a circular beach ball, and of course, you can't drag the ball around the image.

It may surprise you that just like an image, if you draw a circle on a Canvas you cannot drag that circle around the canvas. That's because the canvas won't remember where it drew the circle.

```
// this arc (==circle) is not draggable!!
context.beginPath();
context.arc(20, 30, 15, 0, Math.PI*2);
context.fillStyle='blue';
context.fill();
```

What the Canvas DOESN'T know...

- ...where you drew the circle (it does not know x,y =[20,30]).
- ...the size of the circle (it does not know radius=15).
- ...the color of the circle. (it does not know the circle is blue).

What the Canvas DOES know...

Canvas knows the color of every pixel on it's drawing surface.

The canvas can tell you that at x,y==[20,30] there is a blue pixel, but it does not know if this blue pixel is part of a circle.

What this means...

This means everything drawn on the Canvas is permanent: immovable and unchangeable.

- Canvas can't move the circle or resize the circle.
- Canvas can't recolor the circle or erase the circle.
- Canvas can't say if the mouse is hovering over the circle.
- Canvas can't say if the circle is colliding with another circle.
- Canvas can't let a user drag the circle around the Canvas.

但画布可以制造出移动的幻觉

画布可以通过不断擦除圆形并在不同位置重新绘制圆形，来制造移动的幻觉。通过每秒多次重绘画布，眼睛会被欺骗，看到圆形在画布上移动。

- 清除画布
- 更新圆的位置
- 在新位置重新绘制圆
- 重复，重复，再重复……

这段代码通过不断在新位置重绘圆，制造出运动的错觉。

```
// 创建一个画布
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
ctx.fillStyle='红色';
document.body.appendChild(canvas);

// 一个表示圆的X位置的变量
var circleX=20;

// 开始在画布上动画绘制圆
// 通过不断清除并重绘圆
// 在新位置
requestAnimationFrame(animate);

function animate(){
     // 更新圆圈的X位置
     circleX++;
     // 在新位置重绘圆形
    ctx.clearRect(0,0,canvas.width,canvas.height);
    ctx.beginPath();
ctx.arc( circleX, 30,15,0,Math.PI*2 );
     ctx.fill();
     // 请求另一个 animate() 循环
requestAnimationFrame(animate);
}
```

第8.2节：在画布上拖动圆形和矩形

什么是“形状”？

通常你通过创建一个JavaScript“形状”对象来保存你的形状，每个对象代表一个形状。

```
var myCircle = { x:30, y:20, radius:15 };
```

当然，你并不是真的保存形状，而是保存如何绘制这些形状的定义。

然后将每个形状对象放入数组中，便于引用。

```
// 保存画布上绘制的形状的相关信息
var shapes=[];

// 定义一个圆并保存到 shapes[] 数组中
```

But Canvas can give the I-L-L-U-S-I-O-N of movement

Canvas can give the **illusion of movement** by continuously erasing the circle and redrawing it in a different position. By redrawing the Canvas many times per second, the eye is fooled into seeing the circle move across the Canvas.

- Erase the canvas
- Update the circle's position
- Redraw the circle in it's new position
- Repeat, repeat, repeat ...

This code gives the **illusion of movement** by continuously redrawing a circle in new positions.

```
// create a canvas
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
ctx.fillStyle='red';
document.body.appendChild(canvas);

// a variable indicating a circle's X position
var circleX=20;

// start animating the circle across the canvas
// by continuously erasing & redrawing the circle
// in new positions
requestAnimationFrame(animate);

function animate(){
    // update the X position of the circle
    circleX++;
    // redraw the circle in it's new position
    ctx.clearRect(0,0,canvas.width,canvas.height);
    ctx.beginPath();
    ctx.arc( circleX, 30,15,0,Math.PI*2 );
    ctx.fill();
    // request another animate() loop
    requestAnimationFrame(animate);
}
```

Section 8.2: Dragging circles & rectangles around the Canvas

What is a "Shape"?

You typically save your shapes by creating a JavaScript "shape" object representing each shape.

```
var myCircle = { x:30, y:20, radius:15 };
```

Of course, you're not really saving shapes. Instead, you're saving the definition of how to draw the shapes.

Then put every shape-object into an array for easy reference.

```
// save relevant information about shapes drawn on the canvas
var shapes=[];

// define one circle and save it in the shapes[] array
```

```
shapes.push( {x:10, y:20, radius:15, fillcolor:'blue'} );

// 定义一个矩形并保存到 shapes[] 数组中
shapes.push( {x:10, y:100, width:50, height:35, fillcolor:'red'} );
```

使用鼠标事件进行拖拽

拖拽形状或图像需要响应以下鼠标事件：

鼠标按下时：

测试鼠标下是否有任何形状。如果鼠标下有形状，用户意图拖动该形状。因此，保持对该形状的引用，并设置一个表示拖动正在进行的真/假isDragging标志。

鼠标移动时：

计算自上次mousemove事件以来鼠标拖动的距离，并按该距离更改被拖动
形状的位置。要更改形状的位置，需要更改该

形状对象中的 x、 y 位置属性。

鼠标松开或移出时：

用户意图停止拖动操作，因此清除“isDragging”标志。拖动完成。

演示：在画布上拖动圆形和矩形

此演示通过响应鼠标事件拖动画布上的圆形和矩形，并通过清除和重绘来产生移动的错觉。

```
// canvas 相关变量
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
document.body.appendChild(canvas);
canvas.style.border='1px solid red';

// 用于计算画布相对于窗口的位置
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
offsetY=BB.top;
}
var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
canvas.onresize=function(e){ reOffset(); }

// 保存关于画布上绘制形状的相关信息
var shapes=[];
// 定义一个圆并保存到 shapes[] 数组中
shapes.push( {x:30, y:30, radius:15, color:'blue'} );
// 定义一个矩形并保存到 shapes[] 数组中
shapes.push( {x:100, y:-1, width:75, height:35, color:'red'} );

// 拖拽相关变量
var isDragging=false;
var startX,startY;
```

```
shapes.push( {x:10, y:20, radius:15, fillcolor:'blue'} );

// define one rectangle and save it in the shapes[] array
shapes.push( {x:10, y:100, width:50, height:35, fillcolor:'red'} );
```

Using mouse events to do Dragging

Dragging a shape or image requires responding to these mouse events:

On mousedown:

Test if any shape is under the mouse. If a shape is under the mouse, the user is intending to drag that shape. So keep a reference to that shape and set a true/false `isDragging` flag indicating that a drag is in process.

On mousemove:

Calculate the distance that the mouse has been dragged since the last `mousemove` event and change the dragged shape's position by that distance. To change the shape's position, you change the `x,y` position properties in that shape's object.

On mouseup or mouseout:

The user is intending to stop the drag operation, so clear the "isDragging" flag. Dragging is completed.

Demo: Dragging circles & rectangles on the canvas

This demo drags circles & rectangles on the canvas by responding to mouse events and giving the illusion of movement by clearing and redrawing.

```
// canvas related vars
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
document.body.appendChild(canvas);
canvas.style.border='1px solid red';

// used to calc canvas position relative to window
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
}
var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
canvas.onresize=function(e){ reOffset(); }

// save relevant information about shapes drawn on the canvas
var shapes=[];
// define one circle and save it in the shapes[] array
shapes.push( {x:30, y:30, radius:15, color:'blue'} );
// define one rectangle and save it in the shapes[] array
shapes.push( {x:100, y:-1, width:75, height:35, color:'red'} );

// drag related vars
var isDragging=false;
var startX,startY;
```

```
// 保存被拖拽形状的索引（如果有）
var selectedShapeIndex;

// 在画布上绘制所有形状
drawAll();

// 监听鼠标事件
canvas.onmousedown=handleMouseDown;
canvas.onmousemove=handleMouseMove;
canvas.onmouseup=handleMouseUp;
canvas.onmouseout=handleMouseOut;

// 给定鼠标X和Y坐标（mx和my）以及形状对象
// 返回鼠标是否在形状内部的布尔值
function isMouseInShape(mx,my,shape){
    if(shape.radius){
        // 这是一个圆形
        var dx=mx-shape.x;
        var dy=my-shape.y;
        // 数学测试判断鼠标是否在圆内
        if(dx*dx+dy*dy<shape.radius*shape.radius){
            // 是的，鼠标在此圆内
            return(true);
        }
    }else if(shape.width){
        // 这是一个矩形
        var rLeft=shape.x;
        var rRight=shape.x+shape.width;
        var rTop=shape.y;
        var rBott=shape.y+shape.height;
        // 数学测试判断鼠标是否在矩形内
        if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
            return(true);
        }
    }
    // 鼠标不在任何形状内
    return(false);
}

function handleMouseDown(e){
    // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
    // 计算当前鼠标位置
startX=parseInt(e.clientX-offsetX);
    startY=parseInt(e.clientY-offsetY);
    // 测试鼠标位置是否在所有形状内
    // 如果鼠标在某个形状内则发布结果
    for(var i=0;i<shapes.length;i++){
        if(isMouseInShape(startX,startY,shapes[i])){
            // 鼠标在此形状内
            // 选择此形状
selectedShapeIndex=i;
            // 设置 isDragging 标志
isDragging=true;
            // 并返回（==停止查找
            // 鼠标下的其他形状）
            return;
        }
    }
}
```

```
// hold the index of the shape being dragged (if any)
var selectedShapeIndex;

// draw the shapes on the canvas
drawAll();

// listen for mouse events
canvas.onmousedown=handleMouseDown;
canvas.onmousemove=handleMouseMove;
canvas.onmouseup=handleMouseUp;
canvas.onmouseout=handleMouseOut;

// given mouse X & Y (mx & my) and shape object
// return true/false whether mouse is inside the shape
function isMouseInShape(mx,my,shape){
    if(shape.radius){
        // this is a circle
        var dx=mx-shape.x;
        var dy=my-shape.y;
        // math test to see if mouse is inside circle
        if(dx*dx+dy*dy<shape.radius*shape.radius){
            // yes, mouse is inside this circle
            return(true);
        }
    }else if(shape.width){
        // this is a rectangle
        var rLeft=shape.x;
        var rRight=shape.x+shape.width;
        var rTop=shape.y;
        var rBott=shape.y+shape.height;
        // math test to see if mouse is inside rectangle
        if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
            return(true);
        }
    }
    // the mouse isn't in any of the shapes
    return(false);
}

function handleMouseDown(e){
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    startX=parseInt(e.clientX-offsetX);
    startY=parseInt(e.clientY-offsetY);
    // test mouse position against all shapes
    // post result if mouse is in a shape
    for(var i=0;i<shapes.length;i++){
        if(isMouseInShape(startX,startY,shapes[i])){
            // the mouse is inside this shape
            // select this shape
            selectedShapeIndex=i;
            // set the isDragging flag
            isDragging=true;
            // and return (==stop looking for
            //     further shapes under the mouse)
            return;
        }
    }
}
```

```
function  handleMouseUp(e){
      // 如果没有拖动则返回
      if(!isDragging){return;}
      // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
      // 拖动结束 -- 清除 isDragging 标志
isDragging=false;
}

function  handleMouseOut(e){
      // 如果没有拖动则返回
      if(!isDragging){return;}
      // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
      // 拖动结束 -- 清除 isDragging 标志
isDragging=false;
}

function  handleMouseMove(e){
      // 如果没有拖动则返回
      if(!isDragging){return;}
      // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
      // 计算当前鼠标位置
mouseX=parseInt(e.clientX-offsetX);
      mouseY=parseInt(e.clientY-offsetY);
      // 鼠标从上一次 mousemove 位置拖动了多远？
      var  dx=mouseX-startX;
      var  dy=mouseY-startY;
      // 按拖动距离移动选中的形状
      var  selectedShape=shapes[selectedShapeIndex];
     selectedShape.x+=dx;
selectedShape.y+=dy;
      // 清空画布并重绘所有形状
     drawAll();
      // 更新拖动起始位置（即当前鼠标位置）
     startX=mouseX;
startY=mouseY;
}

// 清除画布并
// 重绘所有形状在它们当前的位置
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.radius){
            // 它是一个圆形
ctx.beginPath();
ctx.arc(shape.x,shape.y,shape.radius,0,Math.PI*2);
            ctx.fillStyle=shape.color;
ctx.fill();
        } else  if(shape.width){
            // 它是一个矩形
ctx.fillStyle=shape.color;
            ctx.fillRect(shape.x,shape.y,shape.width,shape.height);
        }
    }
}
```

```
function handleMouseUp(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseOut(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseMove(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // how far has the mouse dragged from its previous mousemove position?
    var dx=mouseX-startX;
    var dy=mouseY-startY;
    // move the selected shape by the drag distance
    var selectedShape=shapes[selectedShapeIndex];
    selectedShape.x+=dx;
    selectedShape.y+=dy;
    // clear the canvas and redraw all shapes
    drawAll();
    // update the starting drag position (== the current mouse position)
    startX=mouseX;
    startY=mouseY;
}

// clear the canvas and
// redraw all shapes in their current positions
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.radius){
            // it's a circle
            ctx.beginPath();
            ctx.arc(shape.x,shape.y,shape.radius,0,Math.PI*2);
            ctx.fillStyle=shape.color;
            ctx.fill();
        }else if(shape.width){
            // it's a rectangle
            ctx.fillStyle=shape.color;
            ctx.fillRect(shape.x,shape.y,shape.width,shape.height);
        }
    }
}
```

第8.3节：在画布上拖动不规则形状

大多数画布绘图要么是矩形（矩形、图像、文本块），要么是圆形（圆）。

圆形和矩形有数学测试来检查鼠标是否在其内部。这使得测试圆形和
矩形变得简单、快速且高效。你可以在几分之一秒内"命中测试"数百个圆形或矩形。

你也可以拖动不规则形状。但不规则形状没有快速的数学命中测试。幸运的是，不规则形状确实有内置的命中测试来确定一个点（鼠标）是否在形状内部：context.isPointInPath。虽然isPointInPath工作良好，但它远不如纯数学命中测试高效——通常比纯数学命中测试慢10倍。

使用isPointInPath时的一个要求是，你必须在调用isPointInPath之前立即"重新定义"被测试的路径。"重新定义"意味着你必须发出路径绘制命令（如上所述），但在用isPointInPath测试之前，你不需要对路径进行stroke()或fill()。这样你可以测试之前绘制的路径，而不必覆盖（描边/填充）画布上的那些路径。

不规则形状不必像日常的三角形那样常见。你也可以对任何极其不规则的路径进行命中测试。

这个带注释的示例展示了如何拖动不规则路径形状以及圆形和矩形：

```
// canvas 相关变量
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
document.body.appendChild(canvas);
canvas.style.border='1px solid red';

// 用于计算画布相对于窗口的位置
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
offsetY=BB.top;
}
var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
canvas.onresize=function(e){ reOffset(); }

// 保存关于画布上绘制形状的相关信息
var shapes=[];
// 定义一个圆并保存到 shapes[] 数组中
shapes.push( {x:20, y:20, radius:15, color:'blue'} );
// 定义一个矩形并保存到 shapes[] 数组中
shapes.push( {x:100, y:-1, width:75, height:35, color:'red'} );
// 定义一个三角形路径并将其保存到shapes[]数组中
shapes.push( {x:0, y:0, points:[{x:50,y:30},{x:75,y:60},{x:25,y:60}],color:'green'} );

// 拖拽相关变量
var isDragging=false;
var startX,startY;

// 保存被拖动形状的索引（如果有）
var selectedShapeIndex;

// 在画布上绘制形状
```

Section 8.3: Dragging irregular shapes around the Canvas

Most Canvas drawings are either rectangular (rectangles, images, text-blocks) or circular (circles).

Circles & rectangles have mathematical tests to check if the mouse is inside them. This makes testing circles and rectangles easy, quick and efficient. You can "hit-test" hundreds of circles or rectangles in a fraction of a second.

You can also drag irregular shapes. But irregular shapes have no quick mathematical hit-test. Fortunately, irregular shapes do have a built-in hit-test to determine if a point (mouse) is inside the shape: `context.isPointInPath`. While `isPointInPath` works well, it is not nearly as efficient as purely mathematical hit-tests -- it is often up to 10X slower than pure mathematical hit-tests.

One requirement when using `isPointInPath` is that you must "redefine" the Path being tested immediately before calling `isPointInPath`. "Redefine" means you must issue the path drawing commands (as above), but you don't need to stroke() or fill() the Path before testing it with `isPointInPath`. This way you can test previously drawn Paths without having to overwrite (stroke/fill) those previous Paths on the Canvas itself.

The irregular shape doesn't need to be as common as the everyday triangle. You can also hit-test any wildly irregular Paths.

This annotated example shows how to drag irregular Path shapes as well as circles and rectangles:

```
// canvas related vars
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
document.body.appendChild(canvas);
canvas.style.border='1px solid red';

// used to calc canvas position relative to window
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
}
var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
canvas.onresize=function(e){ reOffset(); }

// save relevant information about shapes drawn on the canvas
var shapes=[];
// define one circle and save it in the shapes[] array
shapes.push( {x:20, y:20, radius:15, color:'blue'} );
// define one rectangle and save it in the shapes[] array
shapes.push( {x:100, y:-1, width:75, height:35, color:'red'} );
// define one triangle path and save it in the shapes[] array
shapes.push( {x:0, y:0, points:[{x:50,y:30},{x:75,y:60},{x:25,y:60}],color:'green'} );

// drag related vars
var isDragging=false;
var startX,startY;

// hold the index of the shape being dragged (if any)
var selectedShapeIndex;

// draw the shapes on the canvas
```

```
drawAll();

// 监听鼠标事件
canvas.onmousedown=handleMouseDown;
canvas.onmousemove=handleMouseMove;
canvas.onmouseup=handleMouseUp;
canvas.onmouseout=handleMouseOut;

// 给定鼠标X和Y坐标（mx和my）以及形状对象
// 返回鼠标是否在形状内部的布尔值
function isMouseInShape(mx,my,shape){
    if(shape.radius){
        // 这是一个圆形
        var dx=mx-shape.x;
        var dy=my-shape.y;
        // 数学测试判断鼠标是否在圆内
        if(dx*dx+dy*dy<shape.radius*shape.radius){
            // 是的，鼠标在此圆内
            return(true);
        }
    }else if(shape.width){
        // 这是一个矩形
        var rLeft=shape.x;
        var rRight=shape.x+shape.width;
        var rTop=shape.y;
        var rBott=shape.y+shape.height;
        // 数学测试判断鼠标是否在矩形内
        if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
            return(true);
        }
    } else if(shape.points){
        // 这是一个折线路径
        // 首先重新定义路径（无需描边/填充！）
       defineIrregularPath(shape);
        // 然后使用 isPointInPath 进行命中测试
        if(ctx.isPointInPath(mx,my)){
            return(true);
        }
    }
    // 鼠标不在任何形状内
    return(false);
}

function handleMouseDown(e){
    // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
    // 计算当前鼠标位置
startX=parseInt(e.clientX-offsetX);
    startY=parseInt(e.clientY-offsetY);
    // 测试鼠标位置是否在所有形状内
    // 如果鼠标在某个形状内则发布结果
    for(var i=0;i<shapes.length;i++){
        if(isMouseInShape(startX,startY,shapes[i])){
            // 鼠标在此形状内
            // 选择此形状
selectedShapeIndex=i;
            // 设置 isDragging 标志
isDragging=true;
            // 并返回（==停止查找
            // 鼠标下的其他形状）
            return;
```

```
drawAll();

// listen for mouse events
canvas.onmousedown=handleMouseDown;
canvas.onmousemove=handleMouseMove;
canvas.onmouseup=handleMouseUp;
canvas.onmouseout=handleMouseOut;

// given mouse X & Y (mx & my) and shape object
// return true/false whether mouse is inside the shape
function isMouseInShape(mx,my,shape){
    if(shape.radius){
        // this is a circle
        var dx=mx-shape.x;
        var dy=my-shape.y;
        // math test to see if mouse is inside circle
        if(dx*dx+dy*dy<shape.radius*shape.radius){
            // yes, mouse is inside this circle
            return(true);
        }
    }else if(shape.width){
        // this is a rectangle
        var rLeft=shape.x;
        var rRight=shape.x+shape.width;
        var rTop=shape.y;
        var rBott=shape.y+shape.height;
        // math test to see if mouse is inside rectangle
        if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
            return(true);
        }
    }else if(shape.points){
        // this is a polyline path
        // First redefine the path again (no need to stroke/fill!)
        defineIrregularPath(shape);
        // Then hit-test with isPointInPath
        if(ctx.isPointInPath(mx,my)){
            return(true);
        }
    }
    // the mouse isn't in any of the shapes
    return(false);
}

function handleMouseDown(e){
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    startX=parseInt(e.clientX-offsetX);
    startY=parseInt(e.clientY-offsetY);
    // test mouse position against all shapes
    // post result if mouse is in a shape
    for(var i=0;i<shapes.length;i++){
        if(isMouseInShape(startX,startY,shapes[i])){
            // the mouse is inside this shape
            // select this shape
            selectedShapeIndex=i;
            // set the isDragging flag
            isDragging=true;
            // and return (==stop looking for
            //     further shapes under the mouse)
            return;
        }
    }
}

function handleMouseUp(e){
    // 如果没有拖动则返回
    if(!isDragging){return;}
    // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
    // 拖动结束 -- 清除 isDragging 标志
isDragging=false;
}

function handleMouseOut(e){
    // 如果没有拖动则返回
    if(!isDragging){return;}
    // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
    // 拖动结束 -- 清除 isDragging 标志
isDragging=false;
}

function handleMouseMove(e){
    // 如果没有拖动则返回
    if(!isDragging){return;}
    // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
    // 计算当前鼠标位置
mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // 鼠标从上一次 mousemove 位置拖动了多远？
    var dx=mouseX-startX;
    var dy=mouseY-startY;
    // 按拖动距离移动选中的形状
    var selectedShape=shapes[selectedShapeIndex];
   selectedShape.x+=dx;
selectedShape.y+=dy;
    // 清空画布并重绘所有形状
   drawAll();
    // 更新拖动起始位置（即当前鼠标位置）
   startX=mouseX;
startY=mouseY;
}

// 清除画布并
// 重绘所有形状在它们当前的位置
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.radius){
            // 它是一个圆形
ctx.beginPath();
ctx.arc(shape.x,shape.y,shape.radius,0,Math.PI*2);
            ctx.fillStyle=shape.color;
ctx.fill();
        }else if(shape.width){
            // 它是一个矩形
ctx.fillStyle=shape.color;
```

```
        }
    }
}

function handleMouseUp(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseOut(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseMove(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // how far has the mouse dragged from its previous mousemove position?
    var dx=mouseX-startX;
    var dy=mouseY-startY;
    // move the selected shape by the drag distance
    var selectedShape=shapes[selectedShapeIndex];
    selectedShape.x+=dx;
    selectedShape.y+=dy;
    // clear the canvas and redraw all shapes
    drawAll();
    // update the starting drag position (== the current mouse position)
    startX=mouseX;
    startY=mouseY;
}

// clear the canvas and
// redraw all shapes in their current positions
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.radius){
            // it's a circle
            ctx.beginPath();
            ctx.arc(shape.x,shape.y,shape.radius,0,Math.PI*2);
            ctx.fillStyle=shape.color;
            ctx.fill();
        }else if(shape.width){
            // it's a rectangle
            ctx.fillStyle=shape.color;
ctx.fillRect(shape.x,shape.y,shape.width,shape.height);
        }else if(shape.points){
            // 它是一个折线路径
defineIrregularPath(shape);
            ctx.fillStyle=shape.color;
            ctx.fill();
        }
    }
}

function defineIrregularPath(shape){
    var points=shape.points;
ctx.beginPath();
ctx.moveTo(shape.x+points[0].x,shape.y+points[0].y);
    for(var i=1;i<points.length;i++){
        ctx.lineTo(shape.x+points[i].x,shape.y+points[i].y);
    }
ctx.closePath();
}
```

第8.4节：在画布上拖动图像

参见此示例，了解在画布上拖动形状的通用说明。

此带注释的示例展示了如何在画布上拖动图像

```
// canvas 相关变量
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
canvas.width=378;
canvas.height=378;
var cw=canvas.width;
var ch=canvas.height;
document.body.appendChild(canvas);
canvas.style.border='1px solid red';

// 用于计算画布相对于窗口的位置
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
offsetY=BB.top;
}
var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
canvas.onresize=function(e){ reOffset(); }

// 保存关于画布上绘制形状的相关信息
var shapes=[];

// 拖拽相关变量
var isDragging=false;
var startX,startY;

// 保存被拖动形状的索引（如果有）
var selectedShapeIndex;

// 加载图片
var card=new Image();
card.onload=function(){
```

```
            ctx.fillRect(shape.x,shape.y,shape.width,shape.height);
        }else if(shape.points){
            // its a polyline path
            defineIrregularPath(shape);
            ctx.fillStyle=shape.color;
            ctx.fill();
        }
    }
}

function defineIrregularPath(shape){
    var points=shape.points;
    ctx.beginPath();
    ctx.moveTo(shape.x+points[0].x,shape.y+points[0].y);
    for(var i=1;i<points.length;i++){
        ctx.lineTo(shape.x+points[i].x,shape.y+points[i].y);
    }
    ctx.closePath();
}
```

Section 8.4: Dragging images around the Canvas

See this Example for a general explanation of dragging Shapes around the Canvas.

This annotated example shows how to drag images around the Canvas

```
// canvas related vars
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
canvas.width=378;
canvas.height=378;
var cw=canvas.width;
var ch=canvas.height;
document.body.appendChild(canvas);
canvas.style.border='1px solid red';

// used to calc canvas position relative to window
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
}
var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
canvas.onresize=function(e){ reOffset(); }

// save relevant information about shapes drawn on the canvas
var shapes=[];

// drag related vars
var isDragging=false;
var startX,startY;

// hold the index of the shape being dragged (if any)
var selectedShapeIndex;

// load the image
var card=new Image();
card.onload=function(){
      // 定义一张图片并保存到 shapes[] 数组中
shapes.push( {x:30, y:10, width:127, height:150, image:card} );
      // 在画布上绘制形状
drawAll();
      // 监听鼠标事件
canvas.onmousedown=handleMouseDown;
      canvas.onmousemove=handleMouseMove;
      canvas.onmouseup=handleMouseUp;
      canvas.onmouseout=handleMouseOut;
};
// 在这里放置你的图片源！
card.src='https://dl.dropboxusercontent.com/u/139992952/stackoverflow/card.png';


// 给定鼠标X和Y坐标（mx和my）以及形状对象
// 返回鼠标是否在形状内部的布尔值
function isMouseInShape(mx,my,shape){
      // 这个形状是图片吗？
      if(shape.image){
            // 这是一个矩形
            var rLeft=shape.x;
            var rRight=shape.x+shape.width;
            var rTop=shape.y;
            var rBott=shape.y+shape.height;
            // 数学测试以判断鼠标是否在图片内
            if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
                  return(true);
            }
      }
      // 鼠标不在任何这些形状内
      return(false);
}

function handleMouseDown(e){
      // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
      // 计算当前鼠标位置
startX=parseInt(e.clientX-offsetX);
      startY=parseInt(e.clientY-offsetY);
      // 测试鼠标位置是否在所有形状内
      // 如果鼠标在某个形状内则发布结果
      for(var i=0;i<shapes.length;i++){
            if(isMouseInShape(startX,startY,shapes[i])){
                  // 鼠标在此形状内
                  // 选择此形状
selectedShapeIndex=i;
                  // 设置 isDragging 标志
isDragging=true;
                  // 并返回（==停止查找
                  // 鼠标下的其他形状）
                  return;
            }
      }
}

function handleMouseUp(e){
      // 如果没有拖动则返回
      if(!isDragging){return;}
      // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
```

```
    // define one image and save it in the shapes[] array
    shapes.push( {x:30, y:10, width:127, height:150, image:card} );
    // draw the shapes on the canvas
    drawAll();
    // listen for mouse events
    canvas.onmousedown=handleMouseDown;
    canvas.onmousemove=handleMouseMove;
    canvas.onmouseup=handleMouseUp;
    canvas.onmouseout=handleMouseOut;
};
// put your image src here!
card.src='https://dl.dropboxusercontent.com/u/139992952/stackoverflow/card.png';


// given mouse X & Y (mx & my) and shape object
// return true/false whether mouse is inside the shape
function isMouseInShape(mx,my,shape){
    // is this shape an image?
    if(shape.image){
        // this is a rectangle
        var rLeft=shape.x;
        var rRight=shape.x+shape.width;
        var rTop=shape.y;
        var rBott=shape.y+shape.height;
        // math test to see if mouse is inside image
        if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
            return(true);
        }
    }
    // the mouse isn't in any of this shapes
    return(false);
}

function handleMouseDown(e){
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    startX=parseInt(e.clientX-offsetX);
    startY=parseInt(e.clientY-offsetY);
    // test mouse position against all shapes
    // post result if mouse is in a shape
    for(var i=0;i<shapes.length;i++){
        if(isMouseInShape(startX,startY,shapes[i])){
            // the mouse is inside this shape
            // select this shape
            selectedShapeIndex=i;
            // set the isDragging flag
            isDragging=true;
            // and return (==stop looking for
            //     further shapes under the mouse)
            return;
        }
    }
}

function handleMouseUp(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // 拖动结束 -- 清除 isDragging 标志
isDragging=false;
}

function handleMouseOut(e){
    // 如果没有拖动则返回
    if(!isDragging){return;}
    // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
    // 拖动结束 -- 清除 isDragging 标志
isDragging=false;
}

function handleMouseMove(e){
    // 如果没有拖动则返回
    if(!isDragging){return;}
    // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
    // 计算当前鼠标位置
mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // 鼠标从上一次 mousemove 位置拖动了多远？
    var dx=mouseX-startX;
    var dy=mouseY-startY;
    // 按拖动距离移动选中的形状
    var selectedShape=shapes[selectedShapeIndex];
   selectedShape.x+=dx;
selectedShape.y+=dy;
    // 清空画布并重绘所有形状
   drawAll();
    // 更新拖动起始位置（即当前鼠标位置）
   startX=mouseX;
startY=mouseY;
}

// 清除画布并
// 重绘所有形状在它们当前的位置
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.image){
            // 它是一个图片
ctx.drawImage(shape.image,shape.x,shape.y);
        }
    }
}
```

```
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseOut(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseMove(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // how far has the mouse dragged from its previous mousemove position?
    var dx=mouseX-startX;
    var dy=mouseY-startY;
    // move the selected shape by the drag distance
    var selectedShape=shapes[selectedShapeIndex];
    selectedShape.x+=dx;
    selectedShape.y+=dy;
    // clear the canvas and redraw all shapes
    drawAll();
    // update the starting drag position (== the current mouse position)
    startX=mouseX;
    startY=mouseY;
}

// clear the canvas and
// redraw all shapes in their current positions
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.image){
            // it's an image
            ctx.drawImage(shape.image,shape.x,shape.y);
        }
    }
}
```

第9章：媒体类型和画布

第9.1节：在画布上基本加载和播放视频

画布可以用来显示来自各种来源的视频。这个示例展示了如何加载视频文件资源，显示它，并添加一个简单的点击屏幕播放/暂停切换功能。

这个Stackoverflow自答问题“如何使用HTML5 canvas标签显示视频”展示了以下示例代码的实际效果。

仅仅是一张图片

对于画布来说，视频只是一个图片。你可以像绘制任何图片一样绘制它。不同之处在于视频可以播放并且有声音。

获取画布和基本设置

```
// 假设你知道如何添加画布并正确设置其大小。
var canvas = document.getElementById("myCanvas"); // 从页面获取画布
var ctx = canvas.getContext("2d");
var videoContainer; // 用于保存视频及相关信息的对象
```

创建并加载视频

```
var video = document.createElement("video"); // 创建一个视频元素
video.src = "urlOffVideo.webm";
// 视频现在将开始加载。
// 由于需要一些额外的信息，我们将把视频放在一个
// 容器对象中以方便操作
video.autoPlay = false; // 确保视频不会自动播放
video.loop = true; // 设置视频循环播放。
videoContainer = {  // 我们将根据需要添加属性
     video : video,
ready : false,
};
```

与图像元素不同，视频不必完全加载即可在画布上显示。视频还提供了许多额外的事件，可用于监控视频的状态。

在这种情况下，我们希望知道视频何时准备好播放。oncanplay 表示视频已加载足够部分可以播放，但可能还不足以播放到结尾。

```
video.oncanplay = readyToPlayVideo; // 将事件设置为下面
                                     // 可以找到的播放函数
```

或者你可以使用 oncanplaythrough，当视频加载足够可以播放到结尾时触发该事件。

```
video.oncanplaythrough = readyToPlayVideo; // 将事件设置为下面
                                            // 可以找到的播放函数
```

只能使用其中一个 canPlay 事件，不能同时使用两个。

可以播放事件（相当于图像加载完成事件）

```
function readyToPlayVideo(event){ // 这是对视频的引用
     // 视频可能与画布大小不匹配，因此找到一个缩放比例以适应
    videoContainer.scale = Math.min(
                         canvas.width / this.videoWidth,
```

Chapter 9: Media types and the canvas

Section 9.1: Basic loading and playing a video on the canvas

The canvas can be used to display video from a variety of sources. This example shows how to load a video as a file resource, display it and add a simple click on screen play/pause toggle.

This stackoverflow self answered question How do I display a video using HTML5 canvas tag shows the following example code in action.

Just an image

A video is just an image as far as the canvas is concerned. You can draw it like any image. The difference being the video can play and has sound.

Get canvas and basic setup

```
// It is assumed you know how to add a canvas and correctly size it.
var canvas = document.getElementById("myCanvas"); // get the canvas from the page
var ctx = canvas.getContext("2d");
var videoContainer; // object to hold video and associated info
```

Creating and loading the video

```
var video = document.createElement("video"); // create a video element
video.src = "urlOffVideo.webm";
// the video will now begin to load.
// As some additional info is needed we will place the video in a
// containing object for convenience
video.autoPlay = false; // ensure that the video does not auto play
video.loop = true; // set the video to loop.
videoContainer = {  // we will add properties as needed
     video : video,
     ready : false,
};
```

Unlike images elements videos don't have to be fully loaded to be displayed on the canvas. Videos also provide a host of extra events that can be used to monitor status of the video.

In this case we wish to know when the video is ready to play. oncanplay means that enough of the video has loaded to play some of it, but there may not be enough to play to the end.

```
video.oncanplay = readyToPlayVideo; // set the event to the play function that
                                     // can be found below
```

Alternatively you can use oncanplaythrough which will fire when enough of the video has loaded so that it can be played to the end.

```
video.oncanplaythrough = readyToPlayVideo; // set the event to the play function that
                                            // can be found below
```

Only use one of the canPlay events not both.

The can play event (equivalent to image onload)

```
function readyToPlayVideo(event){ // this is a reference to the video
    // the video may not match the canvas size so find a scale to fit
    videoContainer.scale = Math.min(
                         canvas.width / this.videoWidth,
canvas.height / this.videoHeight);
    videoContainer.ready = true;
    // 视频可以播放，因此将其交给显示函数处理
requestAnimationFrame(undateCanvas);
}
```

显示中

视频不会自动在画布上播放。你需要为每一帧重新绘制它。由于很难准确知道帧率及其发生时间，最好的方法是将视频当作以60fps运行来显示。如果帧率较低，则只是渲染同一帧两次。如果帧率较高，则无法看到额外的帧，因此我们忽略它们。

视频元素本质上是一个图像元素，可以像绘制任何图像一样绘制它，你可以缩放、旋转、平移视频，镜像它，淡入淡出，裁剪并只显示部分内容，绘制两次，第二次使用全局合成模式添加特效，如变亮、滤色等。

```
function updateCanvas(){
ctx.clearRect(0,0,canvas.width,canvas.height); // 虽然不总是需要
                                                    // 但你可能会从
                                                    // 之前的视频中得到坏像素，所以清除以
                                                    // 保证安全
    // 仅在加载并准备好时绘制
    if(videoContainer !== undefined && videoContainer.ready){
        // 找到视频在画布上的左上角位置
        var scale = videoContainer.scale;
        var vidH = videoContainer.video.videoHeight;
        var vidW = videoContainer.video.videoWidth;
        var top = canvas.height / 2 - (vidH /2 ) * scale;
        var left = canvas.width / 2 - (vidW /2 ) * scale;
        // 现在只需以正确的大小绘制视频
ctx.drawImage(videoContainer.video, left, top, vidW * scale, vidH * scale);
        if(videoContainer.video.paused){ // 如果未播放则显示暂停画面
          drawPayIcon();
        }
    }
    // 全部完成显示
    // 请求下一帧，时间间隔为1/60秒
requestAnimationFrame(updateCanvas);
}
```

基本播放暂停控制

现在视频已经加载并显示，我们需要的就是播放控制。我们将其设置为点击屏幕切换播放。当视频正在播放时，用户点击视频则暂停。暂停时点击则恢复播放。
我们将添加一个函数，用于使视频变暗并绘制播放图标（三角形）

```
function drawPayIcon(){
ctx.fillStyle = "black";  // 使显示变暗
    ctx.globalAlpha = 0.5;
ctx.fillRect(0,0,canvas.width,canvas.height);
     ctx.fillStyle = "#DDD"; // 播放图标颜色
     ctx.globalAlpha = 0.75; // 半透明
     ctx.beginPath(); // 创建图标路径
     var size = (canvas.height / 2) * 0.5;  // 图标大小
     ctx.moveTo(canvas.width/2 + size/2, canvas.height / 2); // 从尖端开始
    ctx.lineTo(canvas.width/2 - size/2, canvas.height / 2 + size);
ctx.lineTo(canvas.width/2 - size/2, canvas.height / 2 - size);
```

```
                    canvas.height / this.videoHeight);
    videoContainer.ready = true;
    // the video can be played so hand it off to the display function
    requestAnimationFrame(undateCanvas);
}
```

Displaying

The video will not play itself on the canvas. You need to draw it for every new frame. As it is difficult to know the exact frame rate and when they occur the best approch is to display the video as if running at 60fps. If the frame rate is lower then w just render the same frame twice. If the frame rate is higher then there is nothing that can be don to see the extra frames so we just ignore them.

The video element is just a image element and can be draw like any image, you can scale, rotate, pan the video, mirror it, fade it, clip it and display only parts, draw it twice the second time with a global composite mode to add FX like lighten, screen, etc..

```
function updateCanvas(){
    ctx.clearRect(0,0,canvas.width,canvas.height); // Though not always needed
                                                     // you may get bad pixels from
                                                     // previous videos so clear to be
                                                     // safe
    // only draw if loaded and ready
    if(videoContainer !== undefined && videoContainer.ready){
        // find the top left of the video on the canvas
        var scale = videoContainer.scale;
        var vidH = videoContainer.video.videoHeight;
        var vidW = videoContainer.video.videoWidth;
        var top = canvas.height / 2 - (vidH /2 ) * scale;
        var left = canvas.width / 2 - (vidW /2 ) * scale;
        // now just draw the video the correct size
        ctx.drawImage(videoContainer.video, left, top, vidW * scale, vidH * scale);
        if(videoContainer.video.paused){ // if not playing show the paused screen
            drawPayIcon();
        }
    }
    // all done for display
    // request the next frame in 1/60th of a second
    requestAnimationFrame(updateCanvas);
}
```

Basic play pause control

Now we have the video loaded and displayed all we need is the play control. We will make it as a click toggle play on the screen. When the video is playing and the user clicks the video is paused. When paused the click resumes play. We will add a function to darken the video and draw an play icon (triangle)

```
function drawPayIcon(){
     ctx.fillStyle = "black";  // darken display
     ctx.globalAlpha = 0.5;
     ctx.fillRect(0,0,canvas.width,canvas.height);
     ctx.fillStyle = "#DDD"; // colour of play icon
     ctx.globalAlpha = 0.75; // partly transparent
     ctx.beginPath(); // create the path for the icon
     var size = (canvas.height / 2) * 0.5;  // the size of the icon
     ctx.moveTo(canvas.width/2 + size/2, canvas.height / 2); // start at the pointy end
     ctx.lineTo(canvas.width/2 - size/2, canvas.height / 2 + size);
     ctx.lineTo(canvas.width/2 - size/2, canvas.height / 2 - size);
ctx.closePath();
     ctx.fill();
ctx.globalAlpha = 1; // restore alpha
}
```

现在播放暂停事件

```
function playPauseClick(){
     if(videoContainer !== undefined && videoContainer.ready){
          if(videoContainer.video.paused){
                videoContainer.video.play();
          }else{
videoContainer.video.pause();
          }
     }
}
// 注册事件
canvas.addEventListener("click",playPauseClick);
```

总结

使用画布播放视频非常简单，实时添加效果也很容易。然而，在格式、播放和定位方面存在一些限制。MDN 的 HTMLMediaElement 是获取视频对象完整参考的地方。

一旦图像绘制在画布上，你可以使用ctx.getImageData来访问其中的像素。或者你可以使用canvas.toDataURL来截取静止图像并下载。（仅当视频来自可信来源且不会污染画布时）

注意，如果视频有声音，播放时也会播放声音。

祝你视频制作愉快。

第9.2节：捕获画布并保存为webM视频

从画布帧创建WebM视频并在画布中播放，或上传，或下载。

示例：捕获并播放画布

```
name = "CanvasCapture"; // 放入WebM头部的Mux和Write应用名称字段
quality = 0.7; // 良好质量 1为最佳 < 0.7为一般到较差
fps = 30; // 我尝试了各种帧率，似乎都能工作
          // 进行一些测试以确定你的机器能处理的帧率，因为
          // 不同机器之间差异很大。
var video = new Groover.Video(fps,quality,name)
function capture(){
    if(video.timecode < 5000){ // 5秒
        setTimeout(capture,video.frameDelay);
    }else{
         var videoElement = document.createElement("video");
         videoElement.src = URL.createObjectURL(video.toBlob());
         document.body.appendChild(videoElement);
video = undefined; // 取消引用，因为它占用大量内存。
         return;
    }
    // 第一帧设置视频大小
    video.addFrame(canvas); // 添加当前画布帧
}
capture(); // 开始捕获
```

```
     ctx.closePath();
     ctx.fill();
     ctx.globalAlpha = 1; // restore alpha
}
```

Now the play pause event

```
function playPauseClick(){
     if(videoContainer !== undefined && videoContainer.ready){
          if(videoContainer.video.paused){
                videoContainer.video.play();
          }else{
                videoContainer.video.pause();
          }
     }
}
// register the event
canvas.addEventListener("click",playPauseClick);
```

Summary

Playing a video is very easy using the canvas, adding effect in real time is also easy. There are however some limitations on formats, how you can play and seek. MDN HTMLMediaElement is the place to get the full reference to the video object.

Once the image has been drawn on the canvas you can use `ctx.getImageData` to access the pixels it contains. Or you can use `canvas.toDataURL` to snap a still and download it. (Only if the video is from a trusted source and does not taint the canvas).

Note if the video has sound then playing it will also play the sound.

Happy videoing.

Section 9.2: Capture canvas and Save as webM video

Creating a WebM video from canvas frames and playing in canvas, or upload, or downloading.

Example capture and play canvas

```
name = "CanvasCapture"; // Placed into the Mux and Write Application Name fields of the WebM header
quality = 0.7; // good quality 1 Best < 0.7 ok to poor
fps = 30; // I have tried all sorts of frame rates and all seem to work
          // Do some test to workout what your machine can handle as there
          // is a lot of variation between machines.
var video = new Groover.Video(fps,quality,name)
function capture(){
    if(video.timecode < 5000){ // 5 seconds
         setTimeout(capture,video.frameDelay);
    }else{
         var videoElement = document.createElement("video");
         videoElement.src = URL.createObjectURL(video.toBlob());
         document.body.appendChild(videoElement);
         video = undefined; // DeReference as it is memory hungry.
         return;
    }
    // first frame sets the video size
    video.addFrame(canvas); // Add current canvas frame
}
capture(); // start capture
```

与其花费大量精力却被拒绝，不如先快速插入看看是否可接受。如果被接受，将提供完整细节。还包括额外的捕获选项以获得更好的高清捕获率（已从此版本移除，能在性能良好的机器上以50fps捕获1080p高清）。

这受到了Wammy的启发，但进行了完全重写，采用边编码边捕获的方法，大大减少了捕获期间所需的内存。可以捕获超过30秒的更好数据，处理算法更优。

> 注意帧被编码为webP图像。只有Chrome支持webP画布编码。对于其他浏览器（Firefox和Edge），您需要使用第三方webP编码器，如Libwebp Javascript通过Javascript编码WebP图像速度较慢。（如果被接受，将增加对原始webp图像的支持）。

webM编码器灵感来自Whammy: A Real Time Javascript WebM

```
var Groover = (function(){
    // 确保支持webp
    function canEncode(){
        var canvas = document.createElement("canvas");
        canvas.width = 8;
canvas.height = 8;
        return canvas.toDataURL("image/webp",0.1).indexOf("image/webp") > -1;
    }
    if(!canEncode()){
        return undefined;
    }
    var webmData = null;
    var clusterTimecode = 0;
    var clusterCounter = 0;
    var CLUSTER_MAX_DURATION = 30000;
    var frameNumber = 0;
    var width;
    var height;
    var frameDelay;
    var quality;
    var name;
    const videoMimeType = "video/webm"; // the only one.
    const frameMimeType = 'image/webp'; // 不能是其他类型
    const S = String.fromCharCode;
    const dataTypes = {
object : function(data){ return toBlob(data);},
        number : function(data){ return stream.num(data);},
        string : function(data){ return stream.str(data);},
        array  : function(data){ return data;},
double2Str : function(num){
            var c = new Uint8Array((new Float64Array([num])).buffer);
            return S(c[7]) + S(c[6]) + S(c[5]) + S(c[4]) + S(c[3]) + S(c[2]) + S(c[1]) + S(c[0]);
        }
    };

    const stream = {
num : function(num){ // 写入整数
            var parts = [];
while(num > 0){ parts.push(num & 0xff); num = num >> 8; }
            return new Uint8Array(parts.reverse());
        },
str : function(str){ // 写入字符串
            var i, len, arr;
len = str.length;
```

Rather than put in a huge effort only to be rejected, this is a quick insert to see if acceptable. Will Give full details if accepted. Also include additional capture options for better HD capture rates (removed from this version, Can capture HD 1080 at 50fps on good machines.)

This was inspired by Wammy but is a complete rewrite with encode as you go methodology, greatly reducing the memory required during capture. Can capture more than 30 seconds better data, handling algorithms.

> **Note** frames are encoded into webP images. Only Chrome supports webP canvas encoding. For other browsers (Firefox and Edge) you will need to use a 3rd party webP encoder such as Libwebp Javascript Encoding WebP images via Javascript is slow. (will include addition of raw webp images support if accepted).

The webM encoder inspired by Whammy: A Real Time Javascript WebM

```
var Groover = (function(){
    // ensure webp is supported
    function canEncode(){
        var canvas = document.createElement("canvas");
        canvas.width = 8;
        canvas.height = 8;
        return canvas.toDataURL("image/webp",0.1).indexOf("image/webp") > -1;
    }
    if(!canEncode()){
        return undefined;
    }
    var webmData = null;
    var clusterTimecode = 0;
    var clusterCounter = 0;
    var CLUSTER_MAX_DURATION = 30000;
    var frameNumber = 0;
    var width;
    var height;
    var frameDelay;
    var quality;
    var name;
    const videoMimeType = "video/webm"; // the only one.
    const frameMimeType = 'image/webp'; // can be no other
    const S = String.fromCharCode;
    const dataTypes = {
        object : function(data){ return toBlob(data);},
        number : function(data){ return stream.num(data);},
        string : function(data){ return stream.str(data);},
        array  : function(data){ return data;},
        double2Str : function(num){
            var c = new Uint8Array((new Float64Array([num])).buffer);
            return S(c[7]) + S(c[6]) + S(c[5]) + S(c[4]) + S(c[3]) + S(c[2]) + S(c[1]) + S(c[0]);
        }
    };

    const stream = {
        num : function(num){ // writes int
            var parts = [];
            while(num > 0){ parts.push(num & 0xff); num = num >> 8; }
            return new Uint8Array(parts.reverse());
        },
        str : function(str){ // writes string
            var i, len, arr;
            len = str.length;
```

```
arr = new Uint8Array(len);
            for(i = 0; i < len; i++){arr[i] = str.charCodeAt(i);}
            return arr;
        },
compInt : function(num){ // 细节不全，稍作猜测
            if(num < 128){       // 数字前缀带有一个位（1000表示字节0100，两个，0010
三个，依此类推）
num += 0x80;
                return new Uint8Array([num]);
            }else
            if(num < 0x4000){
num += 0x4000;
                return new Uint8Array([num>>8, num])
            }else
            if(num < 0x200000){
num += 0x200000;
                return new Uint8Array([num>>16, num>>8, num])
            }else
            if(num < 0x10000000){
num += 0x10000000;
                return new Uint8Array([num>>24, num>>16, num>>8, num])
            }
        }
    }
    const ids = { // header names and values
videoData           : 0x1a45dfa3,
版本              : 0x4286,
读取版本          : 0x42f7,
最大ID长度          : 0x42f2,
最大大小长度        : 0x42f3,
文档类型              : 0x4282,
文档类型版本      : 0x4287,
文档类型读取版本 : 0x4285,
段              : 0x18538067,
信息                : 0x1549a966,
时间码比例        : 0x2ad7b1,
MuxingApp           : 0x4d80,
WritingApp          : 0x5741,
Duration            : 0x4489,
Tracks              : 0x1654ae6b,
TrackEntry          : 0xae,
TrackNumber         : 0xd7,
        TrackUID            : 0x63c5,
        FlagLacing          : 0x9c,
        Language            : 0x22b59c,
        CodecID             : 0x86,
        CodecName           : 0x258688,
        TrackType           : 0x83,
        Video               : 0xe0,
        PixelWidth          : 0xb0,
        PixelHeight         : 0xba,
        Cluster             : 0x1f43b675,
        Timecode            : 0xe7,
Frame               : 0xa3,
        Keyframe            : 0x9d012a,
        FrameBlock          : 0x81,
    };
    const keyframeD64Header = '\x9d\x01\x2a'; //VP8 关键帧头 0x9d012a
    const videoDataPos = 1; // 帧数据头的数据位置
    const defaultDelay = dataTypes.double2Str(1000/25);
    const header = [  // webM 头部/块的结构，无论它们叫什么。
ids.videoData,[
```

```
            arr = new Uint8Array(len);
            for(i = 0; i < len; i++){arr[i] = str.charCodeAt(i);}
            return arr;
        },
        compInt : function(num){ // could not find full details so bit of a guess
            if(num < 128){       // number is prefixed with a bit (1000 is on byte 0100 two, 0010
three and so on)
                num += 0x80;
                return new Uint8Array([num]);
            }else
            if(num < 0x4000){
                num += 0x4000;
                return new Uint8Array([num>>8, num])
            }else
            if(num < 0x200000){
                num += 0x200000;
                return new Uint8Array([num>>16, num>>8, num])
            }else
            if(num < 0x10000000){
                num += 0x10000000;
                return new Uint8Array([num>>24, num>>16, num>>8, num])
            }
        }
    }
    const ids = { // header names and values
        videoData          : 0x1a45dfa3,
        Version            : 0x4286,
        ReadVersion        : 0x42f7,
        MaxIDLength        : 0x42f2,
        MaxSizeLength      : 0x42f3,
        DocType            : 0x4282,
        DocTypeVersion     : 0x4287,
        DocTypeReadVersion : 0x4285,
        Segment            : 0x18538067,
        Info               : 0x1549a966,
        TimecodeScale      : 0x2ad7b1,
        MuxingApp          : 0x4d80,
        WritingApp         : 0x5741,
        Duration           : 0x4489,
        Tracks             : 0x1654ae6b,
        TrackEntry         : 0xae,
        TrackNumber        : 0xd7,
        TrackUID           : 0x63c5,
        FlagLacing         : 0x9c,
        Language           : 0x22b59c,
        CodecID            : 0x86,
        CodecName          : 0x258688,
        TrackType          : 0x83,
        Video              : 0xe0,
        PixelWidth         : 0xb0,
        PixelHeight        : 0xba,
        Cluster            : 0x1f43b675,
        Timecode           : 0xe7,
        Frame              : 0xa3,
        Keyframe           : 0x9d012a,
        FrameBlock         : 0x81,
    };
    const keyframeD64Header = '\x9d\x01\x2a'; //VP8 keyframe header 0x9d012a
    const videoDataPos = 1; // data pos of frame data header
    const defaultDelay = dataTypes.double2Str(1000/25);
    const header = [  // structure of webM header/chunks what ever they are called.
        ids.videoData,[
ids.版本, 1,
            ids.读取版本, 1,
            ids.最大ID长度, 4,
            ids.最大尺寸长度, 8,
            ids.文档类型, 'webm',
            ids.文档类型版本, 2,
            ids.文档类型读取版本, 2
        ],
ids.段, [
            ids.信息, [
                ids.时间码比例, 1000000,
                ids.混流应用, 'Groover',
                ids.写入应用, 'Groover',
                ids.时长, 0
            ],
ids.轨道,[
ids.轨道条目,[
ids.轨道编号, 1,
                    ids.轨道UID, 1,
ids.拼接标志, 0,      // 总是0
                    ids.语言, 'und',    // 未定义，我认为这是这个意思
                    ids.编码ID, 'V_VP8',  // 我认为这些不能更改
                    ids.编码名称, 'VP8',  // 我认为这些不能更改
                    ids.轨道类型, 1,
ids.视频, [
                        ids.像素宽度, 0,
                        ids.像素高度, 0
                    ]
                ]
            ]
        ]
    ];
    function getHeader(){
header[3][2][3] = name;
        header[3][2][5] = name;
        header[3][2][7] =  dataTypes.double2Str(frameDelay);
        header[3][3][1][15][1] =  width;
header[3][3][1][15][3] =  height;
        function create(dat){
            var i,kv,data;
data = [];
            for(i = 0; i < dat.length; i += 2){
                kv = {i : dat[i]};
                if(Array.isArray(dat[i + 1])){
                    kv.d = create(dat[i + 1]);
                }else{
kv.d = dat[i + 1];
                }
data.push(kv);
            }
            return data;
        }
        return create(header);
    }
    function addCluster(){
webmData[videoDataPos].d.push({ i: ids.Cluster,d: [{ i: ids.Timecode, d:
Math.round(clusterTimecode)}]}); // 修复了 Round 的错误
       clusterCounter = 0;
    }
    function addFrame(frame){
        var VP8, kfS,riff;
riff = getWebPChunks(atob(frame.toDataURL(frameMimeType, quality).slice(23)));
```

```
            ids.Version, 1,
            ids.ReadVersion, 1,
            ids.MaxIDLength, 4,
            ids.MaxSizeLength, 8,
            ids.DocType, 'webm',
            ids.DocTypeVersion, 2,
            ids.DocTypeReadVersion, 2
        ],
        ids.Segment, [
            ids.Info, [
                ids.TimecodeScale, 1000000,
                ids.MuxingApp, 'Groover',
                ids.WritingApp, 'Groover',
                ids.Duration, 0
            ],
            ids.Tracks,[
                ids.TrackEntry,[
                    ids.TrackNumber, 1,
                    ids.TrackUID, 1,
                    ids.FlagLacing, 0,     // always o
                    ids.Language, 'und',   // undefined I think this means
                    ids.CodecID, 'V_VP8',  // These I think must not change
                    ids.CodecName, 'VP8',  // These I think must not change
                    ids.TrackType, 1,
                    ids.Video, [
                        ids.PixelWidth, 0,
                        ids.PixelHeight, 0
                    ]
                ]
            ]
        ]
    ];
    function getHeader(){
        header[3][2][3] = name;
        header[3][2][5] = name;
        header[3][2][7] =  dataTypes.double2Str(frameDelay);
        header[3][3][1][15][1] =  width;
        header[3][3][1][15][3] =  height;
        function create(dat){
            var i,kv,data;
            data = [];
            for(i = 0; i < dat.length; i += 2){
                kv = {i : dat[i]};
                if(Array.isArray(dat[i + 1])){
                    kv.d = create(dat[i + 1]);
                }else{
                    kv.d = dat[i + 1];
                }
                data.push(kv);
            }
            return data;
        }
        return create(header);
    }
    function addCluster(){
        webmData[videoDataPos].d.push({ i: ids.Cluster,d: [{ i: ids.Timecode, d:
Math.round(clusterTimecode)}]}); // Fixed bug with Round
        clusterCounter = 0;
    }
    function addFrame(frame){
        var VP8, kfS,riff;
        riff = getWebPChunks(atob(frame.toDataURL(frameMimeType, quality).slice(23)));
```

```
VP8 = riff.RIFF[0].WEBP[0];
        kfS = VP8.indexOf(keyframeD64Header) + 3;
        frame = {
width: ((VP8.charCodeAt(kfS + 1) << 8) | VP8.charCodeAt(kfS)) & 0x3FFF,
            height: ((VP8.charCodeAt(kfS + 3) << 8) | VP8.charCodeAt(kfS + 2)) & 0x3FFF,
            data: VP8,
riff: riff
        };
        if(clusterCounter > CLUSTER_MAX_DURATION){
            addCluster();
        }
webmData[videoDataPos].d[webmData[videoDataPos].d.length-1].d.push({
            i: ids.Frame,
d: S(ids.FrameBlock) + S( Math.round(clusterCounter) >> 8) +  S(
Math.round(clusterCounter) & 0xff) + S(128) + frame.data.slice(4),
        });
clusterCounter += frameDelay;
        clusterTimecode += frameDelay;
webmData[videoDataPos].d[0].d[3].d = dataTypes.double2Str(clusterTimecode);
    }
    function startEncoding(){
frameNumber = clusterCounter = clusterTimecode = 0;
        webmData  = getHeader();
addCluster();
    }
    function toBlob(vidData){
        var data,i,vData, len;
vData = [];
        for(i = 0; i < vidData.length; i++){
            data = dataTypes[typeof vidData[i].d](vidData[i].d);
            len  = data.size || data.byteLength || data.length;
            vData.push(stream.num(vidData[i].i));
vData.push(stream.compInt(len));
            vData.push(data)
        }
        return new Blob(vData, {type: videoMimeType});
    }
    function getWebPChunks(str){
        var offset, chunks, id, len, data;
        offset = 0;
chunks = {};
while (offset < str.length) {
            id = str.substr(offset, 4);
            // value will have top bit on (bit 32) so not simply a bitwise operation
            // Warning little endian (Will not work on big endian systems)
len = new Uint32Array(
                new Uint8Array([
           str.charCodeAt(offset + 7),
                    str.charCodeAt(offset + 6),
                    str.charCodeAt(offset + 5),
                    str.charCodeAt(offset + 4)
                ]).buffer)[0];
id = str.substr(offset, 4);
            chunks[id] = chunks[id] === undefined ? [] : chunks[id];
            if (id === 'RIFF' || id === 'LIST') {
chunks[id].push(getWebPChunks(str.substr(offset + 8, len)));
                offset += 8 + len;
            } else if (id === 'WEBP') {
chunks[id].push(str.substr(offset + 8));
                break;
            } 否则 {
chunks[id].push(str.substr(offset + 4));
```

```
        VP8 = riff.RIFF[0].WEBP[0];
        kfS = VP8.indexOf(keyframeD64Header) + 3;
        frame = {
            width: ((VP8.charCodeAt(kfS + 1) << 8) | VP8.charCodeAt(kfS)) & 0x3FFF,
            height: ((VP8.charCodeAt(kfS + 3) << 8) | VP8.charCodeAt(kfS + 2)) & 0x3FFF,
            data: VP8,
            riff: riff
        };
        if(clusterCounter > CLUSTER_MAX_DURATION){
            addCluster();
        }
        webmData[videoDataPos].d[webmData[videoDataPos].d.length-1].d.push({
            i: ids.Frame,
            d: S(ids.FrameBlock) + S( Math.round(clusterCounter) >> 8) +  S(
Math.round(clusterCounter) & 0xff) + S(128) + frame.data.slice(4),
        });
        clusterCounter += frameDelay;
        clusterTimecode += frameDelay;
        webmData[videoDataPos].d[0].d[3].d = dataTypes.double2Str(clusterTimecode);
    }
    function startEncoding(){
        frameNumber = clusterCounter = clusterTimecode = 0;
        webmData  = getHeader();
        addCluster();
    }
    function toBlob(vidData){
        var data,i,vData, len;
        vData = [];
        for(i = 0; i < vidData.length; i++){
            data = dataTypes[typeof vidData[i].d](vidData[i].d);
            len  = data.size || data.byteLength || data.length;
            vData.push(stream.num(vidData[i].i));
            vData.push(stream.compInt(len));
            vData.push(data)
        }
        return new Blob(vData, {type: videoMimeType});
    }
    function getWebPChunks(str){
        var offset, chunks, id, len, data;
        offset = 0;
        chunks = {};
        while (offset < str.length) {
            id = str.substr(offset, 4);
            // value will have top bit on (bit 32) so not simply a bitwise operation
            // Warning little endian (Will not work on big endian systems)
            len = new Uint32Array(
                new Uint8Array([
                    str.charCodeAt(offset + 7),
                    str.charCodeAt(offset + 6),
                    str.charCodeAt(offset + 5),
                    str.charCodeAt(offset + 4)
                ]).buffer)[0];
            id = str.substr(offset, 4);
            chunks[id] = chunks[id] === undefined ? [] : chunks[id];
            if (id === 'RIFF' || id === 'LIST') {
                chunks[id].push(getWebPChunks(str.substr(offset + 8, len)));
                offset += 8 + len;
            } else if (id === 'WEBP') {
                chunks[id].push(str.substr(offset + 8));
                break;
            } else {
                chunks[id].push(str.substr(offset + 4));
                break;
            }
        }
        return chunks;
    }
    function 编码器(fps, _quality = 0.8, _name = "Groover"){
        this.fps = fps;
        this.quality = quality = _quality;
        this.frameDelay = frameDelay = 1000 / fps;
        this.frame = 0;
        this.width = width = null;
        this.timecode = 0;
        this.name = name = _name;
    }
Encoder.prototype = {
        addFrame : function(frame){
            if('canvas' in frame){
                frame = frame.canvas;
            }
            if(width === null){
                this.width = width = frame.width,
                this.height = height = frame.height
                startEncoding();
            }else
            if(width !== frame.width || height !== frame.height){
                throw RangeError("帧大小错误。帧必须具有相同的大小。");
            }
addFrame(frame);
            this.frame += 1;
            this.时间码 = clusterTimecode;
        },
toBlob : 函数(){
            返回 toBlob(webmData);
        }
    }
    return {
视频: 编码器,
    }
})()
```

第9.3节：绘制SVG图像

绘制矢量SVG图像的操作与绘制光栅图像没有区别：
你首先需要将SVG图像加载到HTMLImage元素中，然后使用 drawImage()方法。

```
var image = new Image();
image.onload = 函数(){
    ctx.drawImage(this, 0,0);
}
image.src = "someFile.SVG";
```

SVG图像相比光栅图像有一些优势，因为无论你在画布上绘制多大比例，都不会失去质量
。但请注意，绘制SVG图像可能比绘制光栅图像稍微慢一些。

然而，SVG 图像比光栅图像有更多的限制。

- **出于安全考虑，不能从HTMLImageElement（<img>）中引用的SVG图像加载任何外部内容。**

  SVGImage（<image/>）元素中引用的外部样式表、外部图像、外部滤镜或

```
                break;
            }
        }
        return chunks;
    }
    function Encoder(fps, _quality = 0.8, _name = "Groover"){
        this.fps = fps;
        this.quality = quality = _quality;
        this.frameDelay = frameDelay = 1000 / fps;
        this.frame = 0;
        this.width = width = null;
        this.timecode = 0;
        this.name = name = _name;
    }
    Encoder.prototype = {
        addFrame : function(frame){
            if('canvas' in frame){
                frame = frame.canvas;
            }
            if(width === null){
                this.width = width = frame.width,
                this.height = height = frame.height
                startEncoding();
            }else
            if(width !== frame.width || height !== frame.height){
                throw RangeError("Frame size error. Frames must be the same size.");
            }
            addFrame(frame);
            this.frame += 1;
            this.timecode = clusterTimecode;
        },
        toBlob : function(){
            return toBlob(webmData);
        }
    }
    return {
        Video: Encoder,
    }
})()
```

Section 9.3: Drawing an svg image

To draw a vector SVG image, the operation is not different from a raster image :
You first need to load your SVG image into an HTMLImage element, then use the `drawImage()` method.

```
var image = new Image();
image.onload = function(){
    ctx.drawImage(this, 0,0);
}
image.src = "someFile.SVG";
```

SVG images have some advantages over raster ones, since you won't loose quality, whatever the scale you'll draw it on your canvas. But beware, it may also be a bit slower than drawing a raster image.

However, SVG images come with more restrictions than raster images.

- **For security purpose, no external content can be loaded from an SVG image referenced in an HTMLImageElement(`<img>`)**
  No external stylesheet, no external image referenced in SVGImage (**`<image/>`**) elements, no external filter or

通过 xlink:href属性（`<use xlink:href="anImage.SVG#anElement"/>`）或funcIRI
（`url( )` 属性方法等链接的元素也不允许加载外部内容。
此外，一旦在HTMLImage元素中引用，主文档中附加的样式表将不会对SVG文档产生任何影响。

最后，SVG图像内部不会执行任何脚本。
解决方法： 你需要在引用HTMLImage元素之前，将所有外部元素附加到SVG本身内部。（对于图像或字体，需要附加外部资源的dataURI版本）。

- **根元素（<svg>）必须设置宽度和高度属性为绝对值。**
  如果使用相对长度（例如%），浏览器将无法确定其相对对象。一些浏览器（Blink）会尝试猜测，但大多数浏览器会直接忽略你的图像且不绘制任何内容，且不会给出警告。

- **某些浏览器在将SVG图像绘制到画布时会污染画布。**
  具体来说，Internet Explorer < Edge版本，以及Safari 9在SVG图像中存在<foreignObject>时会出现此情况。

第9.4节：加载和显示图像

加载图像并将其放置在画布上

```
var image = new Image();  // 关于创建图像的说明
image.src = "imageURL";
image.onload = function(){
    ctx.drawImage(this,0,0);
}
```

创建图像

创建图像有多种方法

- `new Image()`
- `document.createElement("img")`
- `<img src = 'imageUrl' id='myImage'>` 作为HTML主体的一部分，并通过 `document.getElementById('myImage')`获取

图像是一个`HTMLImageElement`

Image.src属性

图像 src可以是任何有效的图像URL或编码的dataURL。有关图像格式和支持的更多信息，请参阅本主题的备注。

- `image.src = "http://my.domain.com/images/myImage.jpg"`
- `image.src = "data:image/gif;base64,R0lGODlhAQABAIAAAAUEBAAAACwAAAAAAQABAAACAkQBADs="` *

*数据URL是一个包含黑色的1×1像素gif图像

加载和错误的说明

当设置其src属性时，图像将开始加载。加载是同步的，但 onload事件直到函数或代码退出/返回后才会被调用。

如果你从页面获取一个图像（例如 document.getElementById("myImage")）且其 src已设置，它可能已经加载，也可能未加载。你可以通过 HTMLImageElement.complete检查图像的状态，该属性将

element linked by the `xlink:href` attribute (`<use xlink:href="anImage.SVG#anElement"/>`) or the funcIRI (`url()`) attribute method etc.
Also, stylesheets appended in the main document won't have any effect on the SVG document once referenced in an HTMLImage element.
Finally, no script will be executed inside the SVG Image.
Workaround : You'll need to append all external elements inside the SVG itself before referrencing to the HTMLImage element. (for images or fonts, you need to append a dataURI version of your external resources).

- **The root element (`<svg>`) must have its width and height attributes set to an absolute value.**
  If you were to use relative length (e.g `%`), then the browser won't be able to know to what it is relative. Some browsers (Blink) will try to make a guess, but most will simply ignore your image and won't draw anything, without a warning.

- **Some browsers will taint the canvas when an SVG image has been drawn to it.**
  Specifically, Internet-Explorer < Edge in any case, and Safari 9 when a `<foreignObject>` is present in the SVG image.

Section 9.4: Loading and displaying an Image

To load an image and place it on the canvas

```
var image = new Image();  // see note on creating an image
image.src = "imageURL";
image.onload = function(){
    ctx.drawImage(this,0,0);
}
```

Creating an image

There are several ways to create an image

- `new Image()`
- `document.createElement("img")`
- `<img src = 'imageUrl' id='myImage'>` As part of the HTML body and retrieved with `document.getElementById('myImage')`

The image is a `HTMLImageElement`

Image.src property

The image `src`can be any valid image URL or encoded dataURL. See this topic's Remarks for more information on image formats and support.

- `image.src = "http://my.domain.com/images/myImage.jpg"`
- `image.src = "data:image/gif;base64,R0lGODlhAQABAIAAAAUEBAAAACwAAAAAAQABAAACAkQBADs="` *

*The dataURL is a 1 by 1 pixel gif image containing black

Remarks on loading and errors

The image will begin loading when its src property is set. The loading is syncriouse but the `onload` event will not be called until the function or code has exited/returned.

If you get an image from the page (for example `document.getElementById("myImage")`) and its `src` is set it may or may not have loaded. You can check on the status of the image with `HTMLImageElement.complete` which will be

为true如果完成。这并不意味着图像已加载，它表示图像要么

- 已加载
- 发生了错误
- src属性未设置且等于空字符串""

如果图像来自不可靠的来源，并且由于各种原因可能无法访问，则会生成一个错误事件。当发生这种情况时，图像将处于损坏状态。如果您随后尝试将其绘制到画布上，将抛出以下错误

```
未捕获的 DOMException: 无法在 'CanvasRenderingContext2D' 上执行 'drawImage' ：提供的 HTMLImageEle
ment 处于 'broken' 状态。
```

通过提供 image.onerror = myImgErrorHandler 事件，您可以采取适当的措施以防止错误。

`true` if complete. This does not mean the image has loaded, it means that it has either

- loaded
- there was an error
- src property has not been set and is equal to the empty String `""`

If the image is from an unreliable source and may not be accessible for a variety of reasons it will generate an error event. When this happens the image will be in a broken state. If you then attempt to draw it onto the canvas it will throw the following error

```
Uncaught DOMException: Failed to execute 'drawImage' on 'CanvasRenderingContext2D': The
HTMLImageElement provided is in the 'broken' state.
```

By supplying the `image.onerror = myImgErrorHandler` event you can take appropriate action to prevent errors.

第10章：动画

第10.1节：动画循环中使用 requestAnimationFrame()，而非 setInterval()

requestAnimationFrame 类似于 setInterval，但具有以下重要改进：

- 动画代码与显示刷新同步以提高效率。清除+重绘代码被调度，但不会立即执行。浏览器仅在显示准备刷新时执行清除+重绘代码。这种与刷新周期的同步通过为您的代码提供最多可用时间来完成任务，从而提升动画性能。

- 每个循环总是在允许另一个循环开始之前完成。这防止了"撕裂"现象，即用户看到绘制的不完整版本。眼睛特别容易注意到撕裂现象，撕裂发生时会分散注意力。因此，防止撕裂使动画看起来更平滑、更一致。

- 当用户切换到不同的浏览器标签页时，动画会自动停止。这节省了移动设备的电量，因为设备不会浪费电力去计算用户当前看不到的动画。

设备显示屏每秒刷新约60次，因此 requestAnimationFrame 可以持续以约60"帧"每秒重绘。人眼能感知的运动帧率为20-30帧每秒，因此 requestAnimationFrame 可以轻松制造运动的错觉。

请注意，requestAnimationFrame 会在每次 animateCircle 结束时被重新调用。这是因为每个 requestAnimationFrame 只请求动画函数执行一次。

示例：简单的 `requestAnimationFrame

```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;

    // 开始动画
   requestAnimationFrame(animate);

    function animate(currentTime){

        // 绘制一个完整的随机圆
        var x=Math.random()*canvas.width;
        var y=Math.random()*canvas.height;
        var radius=10+Math.random()*15;
ctx.beginPath();
ctx.arc(x,y,radius,0,Math.PI*2);
        ctx.fillStyle='#'+Math.floor(Math.random()*16777215).toString(16);
```

Chapter 10: Animation

Section 10.1: Use requestAnimationFrame() NOT setInterval() for animation loops

`requestAnimationFrame` is similar to setInterval, it but has these important improvements:

- The animation code is synchronized with display refreshes for efficiency The clear + redraw code is scheduled, but not immediately executed. The browser will execute the clear + redraw code only when the display is ready to refresh. This synchronization with the refresh cycle increases your animation performance by giving your code the most available time in which to complete.

- Every loop is always completed before another loop is allowed to start. This prevents "tearing", where the user sees an incomplete version of the drawing. The eye particularly notices tearing and is distracted when tearing occurs. So preventing tearing makes your animation appear smoother and more consistent.

- Animation automatically stops when the user switches to a different browser tab. This saves power on mobile devices because the device is not wasting power computing an animation that the user can't currently see.

Device displays will refresh about 60 times per second so requestAnimationFrame can continuously redraw at about 60 "frames" per second. The eye sees motion at 20-30 frames per second so requestAnimationFrame can easily create the illusion of motion.

Notice that requestAnimationFrame is recalled at the end of each animateCircle. This is because each 'requestAnimatonFrameonly requests a single execution of the animation function.

Example: simple `requestAnimationFrame

```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;

    // start the animation
    requestAnimationFrame(animate);

    function animate(currentTime){

        // draw a full randomly circle
        var x=Math.random()*canvas.width;
        var y=Math.random()*canvas.height;
        var radius=10+Math.random()*15;
        ctx.beginPath();
        ctx.arc(x,y,radius,0,Math.PI*2);
        ctx.fillStyle='#'+Math.floor(Math.random()*16777215).toString(16);
ctx.fill();

        // 请求另一个动画循环
requestAnimationFrame(animate);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

为了说明 requestAnimationFrame 的优势，这个stackoverflow 问题有一个实时演示

第10.2节：在画布上动画显示图像

此示例加载并在画布上动画显示图像

重要提示！ 确保通过使用image.onload给图像足够的加载时间。

带注释的代码

```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;

    // 与动画相关的变量
    var minX=20;        // 让图像在动画中移动
    var maxX=250;       // 在 minX 和 maxX 之间
    var x=minX;         // 当前的 X 坐标
    var speedX=1;       // 图像每次循环移动1像素
    var direction=1;    // 图像方向：1==向右，-1==向左
    var y=20;           // Y 坐标

    // 加载一张新图片
    // 重要！！！必须使用 img.onload 给图片加载时间！
    var img=new Image();
img.onload=start;
img.src="https://dl.dropboxusercontent.com/u/139992952/stackoverflow/sun.png";
    function start(){
        // 图片完全加载后开始动画
       requestAnimationFrame(animate);
    }

    function animate(time){
```

```
        ctx.fill();

        // request another loop of animation
        requestAnimationFrame(animate);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

To illustrate the advantages of requestAnimationFrame this stackoverflow question has a live demo

Section 10.2: Animate an image across the Canvas

This example loads and animates and image across the Canvas

Important Hint! Make sure you give your image time to fully load by using `image.onload`.

Annotated Code

```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;

    // animation related variables
    var minX=20;        // Keep the image animating
    var maxX=250;       // between minX & maxX
    var x=minX;         // The current X-coordinate
    var speedX=1;       // The image will move at 1px per loop
    var direction=1;    // The image direction: 1==righward, -1==leftward
    var y=20;           // The Y-coordinate

    // Load a new image
    // IMPORTANT!!! You must give the image time to load by using img.onload!
    var img=new Image();
    img.onload=start;
    img.src="https://dl.dropboxusercontent.com/u/139992952/stackoverflow/sun.png";
    function start(){
        // the image is fully loaded sostart animating
        requestAnimationFrame(animate);
    }

    function animate(time){
            // 清除画布
ctx.clearRect(0,0,cw,ch);

            // 绘制
ctx.drawImage(img,x,y);

            // 更新
x += speedX * direction;
            // 保持"x"在最小值和最大值之间
            if(x<minX){ x=minX; direction*=-1; }
            if(x>maxX){ x=maxX; direction*=-1; }

            // 请求另一个动画循环
requestAnimationFrame(animate);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

第10.3节：使用requestAnimationFrame设置帧率

使用requestAnimationFrame在某些系统上可能会以超过60fps的帧率更新。60fps是默认帧率，前提是渲染能够跟上。一些系统可能以120fps甚至更高的帧率运行。

如果你使用以下方法，应该只使用60的整数除数作为帧率，以保证(60 / FRAMES_PER_SECOND) % 1 === 0 为true，否则你会得到不一致的帧率。

```
const FRAMES_PER_SECOND = 30;  // 有效值为60、30、20、15、10...
// 设置渲染下一帧的最小时间
const FRAME_MIN_TIME = (1000/60) * (60 / FRAMES_PER_SECOND) - (1000/60) * 0.5;
var lastFrameTime = 0;  // 上一帧时间
function update(time){
    if(time-lastFrameTime < FRAME_MIN_TIME){ //如果调用过早则跳过该帧
        requestAnimationFrame(update);
        return; // 返回，因为没有要做的事情
    }
lastFrameTime = time; // 记录渲染帧的时间
    // 渲染帧
requestAnimationFrame(update); // 获取下一帧
}
requestAnimationFrame(update); // 开始动画
```

第10.4节：使用Robert Penner方程的缓动

缓动使某个变量在持续时间内不均匀地变化。

"变量"必须能够表示为数字，并且可以代表各种各样的事物：

- 一个X坐标，
- 一个矩形的宽度，
- 一个旋转角度，
- 一个R,G,B颜色的红色分量。
- 任何可以表示为数字的东西。

```
        // clear the canvas
        ctx.clearRect(0,0,cw,ch);

        // draw
        ctx.drawImage(img,x,y);

        // update
        x += speedX * direction;
        // keep "x" inside min & max
        if(x<minX){ x=minX; direction*=-1; }
        if(x>maxX){ x=maxX; direction*=-1; }

        // request another loop of animation
        requestAnimationFrame(animate);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

Section 10.3: Set frame rate using requestAnimationFrame

Using requestAnimationFrame may on some systems update at more frames per second than the 60fps. 60fps is the default rate if the rendering can keep up. Some systems will run at 120fps maybe more.

If you use the following method you should only use frame rates that are integer divisions of 60 so that `(60 / FRAMES_PER_SECOND) % 1 === 0` is `true` or you will get inconsistent frame rates.

```
const FRAMES_PER_SECOND = 30;  // Valid values are 60,30,20,15,10...
// set the mim time to render the next frame
const FRAME_MIN_TIME = (1000/60) * (60 / FRAMES_PER_SECOND) - (1000/60) * 0.5;
var lastFrameTime = 0;  // the last frame time
function update(time){
    if(time-lastFrameTime < FRAME_MIN_TIME){ //skip the frame if the call is too early
        requestAnimationFrame(update);
        return; // return as there is nothing to do
    }
    lastFrameTime = time; // remember the time of the rendered frame
    // render the frame
    requestAnimationFrame(update); // get next farme
}
requestAnimationFrame(update); // start animation
```

Section 10.4: Easing using Robert Penners equations

An easing causes some **variable** to change **unevenly** over a **duration**.

"variable" must be able to be expressed as a number, and can represent a remarkable variety of things:

- an X-coordinate,
- a rectangle's width,
- an angle of rotation,
- the red component of an R,G,B color.
- anything that can be expressed as a number.

"duration" 必须能够表示为一个数字，也可以表示多种事物：

- 一段时间，
- 一段要行进的距离，
- 要执行的动画循环次数，
- 任何可以表示为

"unevenly" 意味着变量从起始值到结束值的变化是不均匀的：

- 开始时较快，结束时较慢——或反之，
- 超过结束值后又回退到结束值，随着持续时间结束，
- 在持续时间内反复弹性地前进/后退，
- 在持续时间结束时“弹跳”到结束值并停下。

归属：罗伯特·彭纳（Robert Penner）创造了缓动函数的“黄金标准”。

引用：https://github.com/danro/jquery-easing/blob/master/jquery.easing.js

```
// t: 持续时间内经过的时间（currentTime-startTime），
// b: 起始值，
// c：从初始值到结束值的总变化（endingValue - startingValue），
// d：总持续时间
var Easings={
easeInQuad: function (t, b, c, d) {
      return c*(t/=d)*t + b;
    },
easeOutQuad: function (t, b, c, d) {
      return -c *(t/=d)*(t-2) + b;
    },
easeInOutQuad: function (t, b, c, d) {
      if ((t/=d/2) < 1) return c/2*t*t + b;
      return -c/2 * ((--t)*(t-2) - 1) + b;
    },
easeInCubic: function (t, b, c, d) {
      return c*(t/=d)*t*t + b;
    },
easeOutCubic: function (t, b, c, d) {
      return c*((t=t/d-1)*t*t + 1) + b;
    },
easeInOutCubic: function (t, b, c, d) {
      if ((t/=d/2) < 1) return c/2*t*t*t + b;
      return c/2*((t-=2)*t*t + 2) + b;
    },
easeInQuart: function (t, b, c, d) {
      return c*(t/=d)*t*t*t + b;
    },
easeOutQuart: function (t, b, c, d) {
      return -c * ((t=t/d-1)*t*t*t - 1) + b;
    },
easeInOutQuart: function (t, b, c, d) {
      if ((t/=d/2) < 1) return c/2*t*t*t*t + b;
      return -c/2 * ((t-=2)*t*t*t - 2) + b;
    },
easeInQuint: function (t, b, c, d) {
      return c*(t/=d)*t*t*t*t + b;
    },
easeOutQuint: function (t, b, c, d) {
      return c*((t=t/d-1)*t*t*t*t + 1) + b;
    },
easeInOutQuint: function (t, b, c, d) {
```

"duration" must be able to be expressed as a number and can also be a variety of things:

- a period of time,
- a distance to be travelled,
- a quantity of animation loops to be executed,
- anything that can be expressed as

"unevenly" means that the variable progresses from beginning to ending values unevenly:

- faster at the beginning & slower at the ending -- or visa-versa,
- overshoots the ending but backs up to the ending as the duration finishes,
- repeatedly advances/retreats elastically during the duration,
- "bounces" off the ending while coming to rest as the duration finishes.

Attribution: Robert Penner has created the "gold standard" of easing functions.

Cite: https://github.com/danro/jquery-easing/blob/master/jquery.easing.js

```
// t: elapsed time inside duration (currentTime-startTime),
// b: beginning value,
// c: total change from beginning value (endingValue-startingValue),
// d: total duration
var Easings={
    easeInQuad: function (t, b, c, d) {
      return c*(t/=d)*t + b;
    },
    easeOutQuad: function (t, b, c, d) {
      return -c *(t/=d)*(t-2) + b;
    },
    easeInOutQuad: function (t, b, c, d) {
      if ((t/=d/2) < 1) return c/2*t*t + b;
      return -c/2 * ((--t)*(t-2) - 1) + b;
    },
    easeInCubic: function (t, b, c, d) {
      return c*(t/=d)*t*t + b;
    },
    easeOutCubic: function (t, b, c, d) {
      return c*((t=t/d-1)*t*t + 1) + b;
    },
    easeInOutCubic: function (t, b, c, d) {
      if ((t/=d/2) < 1) return c/2*t*t*t + b;
      return c/2*((t-=2)*t*t + 2) + b;
    },
    easeInQuart: function (t, b, c, d) {
      return c*(t/=d)*t*t*t + b;
    },
    easeOutQuart: function (t, b, c, d) {
      return -c * ((t=t/d-1)*t*t*t - 1) + b;
    },
    easeInOutQuart: function (t, b, c, d) {
      if ((t/=d/2) < 1) return c/2*t*t*t*t + b;
      return -c/2 * ((t-=2)*t*t*t - 2) + b;
    },
    easeInQuint: function (t, b, c, d) {
      return c*(t/=d)*t*t*t*t + b;
    },
    easeOutQuint: function (t, b, c, d) {
      return c*((t=t/d-1)*t*t*t*t + 1) + b;
    },
    easeInOutQuint: function (t, b, c, d) {
      if ((t/=d/2) < 1) return c/2*t*t*t*t*t + b;
      return c/2*((t-=2)*t*t*t*t + 2) + b;
    },
easeInSine: function (t, b, c, d) {
      return -c * Math.cos(t/d * (Math.PI/2)) + c + b;
    },
easeOutSine: function (t, b, c, d) {
      return c * Math.sin(t/d * (Math.PI/2)) + b;
    },
easeInOutSine: function (t, b, c, d) {
      return -c/2 * (Math.cos(Math.PI*t/d) - 1) + b;
    },
easeInExpo: function (t, b, c, d) {
      return (t==0) ? b : c * Math.pow(2, 10 * (t/d - 1)) + b;
    },
easeOutExpo: function (t, b, c, d) {
      return (t==d) ? b+c : c * (-Math.pow(2, -10 * t/d) + 1) + b;
    },
easeInOutExpo: function (t, b, c, d) {
      if (t==0) return b;
      if (t==d) return b+c;
      if ((t/=d/2) < 1) return c/2 * Math.pow(2, 10 * (t - 1)) + b;
      return c/2 * (-Math.pow(2, -10 * --t) + 2) + b;
    },
easeInCirc: function (t, b, c, d) {
      return -c * (Math.sqrt(1 - (t/=d)*t) - 1) + b;
    },
easeOutCirc: function (t, b, c, d) {
      return c * Math.sqrt(1 - (t=t/d-1)*t) + b;
    },
easeInOutCirc: function (t, b, c, d) {
      if ((t/=d/2) < 1) return -c/2 * (Math.sqrt(1 - t*t) - 1) + b;
      return c/2 * (Math.sqrt(1 - (t-=2)*t) + 1) + b;
    },
easeInElastic: function (t, b, c, d) {
      var s=1.70158;var p=0;var a=c;
      if (t==0) return b;  if ((t/=d)==1) return b+c;  if (!p) p=d*.3;
      if (a < Math.abs(c)) { a=c; var s=p/4; }
      else var s = p/(2*Math.PI) * Math.asin (c/a);
      return -(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )) + b;
    },
easeOutElastic: function (t, b, c, d) {
      var s=1.70158;var p=0;var a=c;
      if (t==0) return b;  if ((t/=d)==1) return b+c;  if (!p) p=d*.3;
      if (a < Math.abs(c)) { a=c; var s=p/4; }
      else var s = p/(2*Math.PI) * Math.asin (c/a);
      return a*Math.pow(2,-10*t) * Math.sin( (t*d-s)*(2*Math.PI)/p ) + c + b;
    },
easeInOutElastic: function (t, b, c, d) {
      var s=1.70158;var p=0;var a=c;
      if (t==0) return b;  if ((t/=d/2)==2) return b+c;  if (!p) p=d*(.3*1.5);
      if (a < Math.abs(c)) { a=c; var s=p/4; }
      else var s = p/(2*Math.PI) * Math.asin (c/a);
      if (t < 1) return -.5*(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )) + b;
      return a*Math.pow(2,-10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )*.5 + c + b;
    },
easeInBack: function (t, b, c, d, s) {
      if (s == undefined) s = 1.70158;
      return c*(t/=d)*t*((s+1)*t - s) + b;
    },
easeOutBack: function (t, b, c, d, s) {
      if (s == undefined) s = 1.70158;
```

```
  if ((t/=d/2) < 1) return c/2*t*t*t*t*t + b;
  return c/2*((t-=2)*t*t*t*t + 2) + b;
},
easeInSine: function (t, b, c, d) {
  return -c * Math.cos(t/d * (Math.PI/2)) + c + b;
},
easeOutSine: function (t, b, c, d) {
  return c * Math.sin(t/d * (Math.PI/2)) + b;
},
easeInOutSine: function (t, b, c, d) {
  return -c/2 * (Math.cos(Math.PI*t/d) - 1) + b;
},
easeInExpo: function (t, b, c, d) {
  return (t==0) ? b : c * Math.pow(2, 10 * (t/d - 1)) + b;
},
easeOutExpo: function (t, b, c, d) {
  return (t==d) ? b+c : c * (-Math.pow(2, -10 * t/d) + 1) + b;
},
easeInOutExpo: function (t, b, c, d) {
  if (t==0) return b;
  if (t==d) return b+c;
  if ((t/=d/2) < 1) return c/2 * Math.pow(2, 10 * (t - 1)) + b;
  return c/2 * (-Math.pow(2, -10 * --t) + 2) + b;
},
easeInCirc: function (t, b, c, d) {
  return -c * (Math.sqrt(1 - (t/=d)*t) - 1) + b;
},
easeOutCirc: function (t, b, c, d) {
  return c * Math.sqrt(1 - (t=t/d-1)*t) + b;
},
easeInOutCirc: function (t, b, c, d) {
  if ((t/=d/2) < 1) return -c/2 * (Math.sqrt(1 - t*t) - 1) + b;
  return c/2 * (Math.sqrt(1 - (t-=2)*t) + 1) + b;
},
easeInElastic: function (t, b, c, d) {
  var s=1.70158;var p=0;var a=c;
  if (t==0) return b;  if ((t/=d)==1) return b+c;  if (!p) p=d*.3;
  if (a < Math.abs(c)) { a=c; var s=p/4; }
  else var s = p/(2*Math.PI) * Math.asin (c/a);
  return -(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )) + b;
},
easeOutElastic: function (t, b, c, d) {
  var s=1.70158;var p=0;var a=c;
  if (t==0) return b;  if ((t/=d)==1) return b+c;  if (!p) p=d*.3;
  if (a < Math.abs(c)) { a=c; var s=p/4; }
  else var s = p/(2*Math.PI) * Math.asin (c/a);
  return a*Math.pow(2,-10*t) * Math.sin( (t*d-s)*(2*Math.PI)/p ) + c + b;
},
easeInOutElastic: function (t, b, c, d) {
  var s=1.70158;var p=0;var a=c;
  if (t==0) return b;  if ((t/=d/2)==2) return b+c;  if (!p) p=d*(.3*1.5);
  if (a < Math.abs(c)) { a=c; var s=p/4; }
  else var s = p/(2*Math.PI) * Math.asin (c/a);
  if (t < 1) return -.5*(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )) + b;
  return a*Math.pow(2,-10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )*.5 + c + b;
},
easeInBack: function (t, b, c, d, s) {
  if (s == undefined) s = 1.70158;
  return c*(t/=d)*t*((s+1)*t - s) + b;
},
easeOutBack: function (t, b, c, d, s) {
  if (s == undefined) s = 1.70158;
```

```
      return c*((t=t/d-1)*t*((s+1)*t + s) + 1) + b;
    },
easeInOutBack: function (t, b, c, d, s) {
      if (s == undefined) s = 1.70158;
      if ((t/=d/2) < 1) return c/2*(t*t*(((s*=(1.525))+1)*t - s)) + b;
      return c/2*((t-=2)*t*(((s*=(1.525))+1)*t + s) + 2) + b;
    },
easeInBounce: function (t, b, c, d) {
      return c - Easings.easeOutBounce (d-t, 0, c, d) + b;
    },
easeOutBounce: function (t, b, c, d) {
      if ((t/=d) < (1/2.75)) {
        return c*(7.5625*t*t) + b;
      } else if (t < (2/2.75)) {
        return c*(7.5625*(t-=(1.5/2.75))*t + .75) + b;
      } else if (t < (2.5/2.75)) {
        return c*(7.5625*(t-=(2.25/2.75))*t + .9375) + b;
      } else {
        return c*(7.5625*(t-=(2.625/2.75))*t + .984375) + b;
      }
    },
easeInOutBounce: function (t, b, c, d) {
      if (t < d/2) return Easings.easeInBounce (t*2, 0, c, d) * .5 + b;
      return Easings.easeOutBounce (t*2-d, 0, c, d) * .5 + c*.5 + b;
    },
};
```

示例用法：

```
// 包含上面的 Easings 对象
var Easings = ...

// 演示
var startTime;
var beginningValue=50;  // 起始 x 坐标
var endingValue=450;    // 结束 x 坐标
var totalChange=endingValue-beginningValue;
var totalDuration=3000; // 毫秒

var keys=Object.keys(Easings);
ctx.textBaseline='居中';
requestAnimationFrame(animate);

function animate(time){
    var PI2=Math.PI*2;
    if(!startTime){startTime=time;}
    var elapsedTime=Math.min(time-startTime,totalDuration);
    ctx.clearRect(0,0,cw,ch);
ctx.beginPath();
    for(var y=0;y<keys.length;y++){
        var key=keys[y];
        var easing=Easings[key];
        var easedX=easing(
elapsedTime,beginningValue,totalChange,totalDuration);
        if(easedX>endingValue){easedX=endingValue;}
        ctx.moveTo(easedX,y*15);
        ctx.arc(easedX,y*15+10,5,0,PI2);
        ctx.fillText(key,460,y*15+10-1);
    }
ctx.fill();
    if(time<startTime+totalDuration){
```

```
      return c*((t=t/d-1)*t*((s+1)*t + s) + 1) + b;
    },
    easeInOutBack: function (t, b, c, d, s) {
      if (s == undefined) s = 1.70158;
      if ((t/=d/2) < 1) return c/2*(t*t*(((s*=(1.525))+1)*t - s)) + b;
      return c/2*((t-=2)*t*(((s*=(1.525))+1)*t + s) + 2) + b;
    },
    easeInBounce: function (t, b, c, d) {
      return c - Easings.easeOutBounce (d-t, 0, c, d) + b;
    },
    easeOutBounce: function (t, b, c, d) {
      if ((t/=d) < (1/2.75)) {
        return c*(7.5625*t*t) + b;
      } else if (t < (2/2.75)) {
        return c*(7.5625*(t-=(1.5/2.75))*t + .75) + b;
      } else if (t < (2.5/2.75)) {
        return c*(7.5625*(t-=(2.25/2.75))*t + .9375) + b;
      } else {
        return c*(7.5625*(t-=(2.625/2.75))*t + .984375) + b;
      }
    },
    easeInOutBounce: function (t, b, c, d) {
      if (t < d/2) return Easings.easeInBounce (t*2, 0, c, d) * .5 + b;
      return Easings.easeOutBounce (t*2-d, 0, c, d) * .5 + c*.5 + b;
    },
};
```

Example Usage:

```
// include the Easings object from above
var Easings = ...

// Demo
var startTime;
var beginningValue=50;  // beginning x-coordinate
var endingValue=450;    // ending x-coordinate
var totalChange=endingValue-beginningValue;
var totalDuration=3000; // ms

var keys=Object.keys(Easings);
ctx.textBaseline='middle';
requestAnimationFrame(animate);

function animate(time){
    var PI2=Math.PI*2;
    if(!startTime){startTime=time;}
    var elapsedTime=Math.min(time-startTime,totalDuration);
    ctx.clearRect(0,0,cw,ch);
    ctx.beginPath();
    for(var y=0;y<keys.length;y++){
        var key=keys[y];
        var easing=Easings[key];
        var easedX=easing(
            elapsedTime,beginningValue,totalChange,totalDuration);
        if(easedX>endingValue){easedX=endingValue;}
        ctx.moveTo(easedX,y*15);
        ctx.arc(easedX,y*15+10,5,0,PI2);
        ctx.fillText(key,460,y*15+10-1);
    }
    ctx.fill();
    if(time<startTime+totalDuration){
```

```
requestAnimationFrame(animate);
    }
}
```

第10.5节：以指定间隔动画（每1秒添加一个新矩形）

此示例每1秒（即1秒间隔）向画布添加一个新矩形

带注释的代码：

```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;

    // 动画间隔变量
    var nextTime=0;       // 下一次动画开始的时间为 "nextTime"
    var duration=1000;    // 每1000毫秒运行一次动画

    var x=20;             // 下一个矩形绘制的X坐标

    // 开始动画
requestAnimationFrame(animate);

    function animate(currentTime){

        // 等待 nextTime 到来
        if(currentTime<nextTime){
            // 请求另一个动画循环
requestAnimationFrame(animate);
            // 时间未到，直接返回
            return;
        }
        // 设置 nextTime
nextTime=currentTime+duration;

        // 每1000毫秒添加一个矩形
ctx.fillStyle='#'+Math.floor(Math.random()*16777215).toString(16);
        ctx.fillRect(x,30,30,30);

        // 更新下一个矩形的X位置
x+=30;

        // 请求另一个动画循环
requestAnimationFrame(animate);
    }

}); // 结束 $(function(){});
```

```
        requestAnimationFrame(animate);
    }
}
```

Section 10.5: Animate at a specified interval (add a new rectangle every 1 second)

This example adds a new rectangle to the canvas every 1 second (== a 1 second interval)

Annotated Code:

```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;

    // animation interval variables
    var nextTime=0;       // the next animation begins at "nextTime"
    var duration=1000;    // run animation every 1000ms

    var x=20;             // the X where the next rect is drawn

    // start the animation
    requestAnimationFrame(animate);

    function animate(currentTime){

        // wait for nextTime to occur
        if(currentTime<nextTime){
            // request another loop of animation
            requestAnimationFrame(animate);
            // time hasn't elapsed so just return
            return;
        }
        // set nextTime
        nextTime=currentTime+duration;

        // add another rectangle every 1000ms
        ctx.fillStyle='#'+Math.floor(Math.random()*16777215).toString(16);
        ctx.fillRect(x,30,30,30);

        // update X position for next rectangle
        x+=30;

        // request another loop of animation
        requestAnimationFrame(animate);
    }

}); // end $(function(){});
```

```
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

第10.6节：在指定时间动画（一个动画时钟）

此示例动画显示一个以填充扇形表示秒数的时钟

带注释的代码：

```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 与画布相关的变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;
    // 时钟的画布样式
    ctx.strokeStyle='lightgray';
    ctx.fillStyle='skyblue';
    ctx.lineWidth=5;

    // 缓存常用值
    var PI=Math.PI;
    var fullCircle=PI*2;
    var sa=-PI/2;   // == 在 context.arc 中表示12点钟方向的角度

    // 开始动画
requestAnimationFrame(animate);

    function animate(currentTime){

        // 从系统时钟获取当前秒数值
        var date=new Date();
        var seconds=date.getSeconds();

        // 清除画布
ctx.clearRect(0,0,cw,ch);

        // 绘制一个完整的圆（即时钟表盘）；
ctx.beginPath();
        ctx.moveTo(100,100);
        ctx.arc(100,100,75,0,fullCircle);
        ctx.stroke();
        // 绘制一个表示当前秒数的扇形
ctx.beginPath();
ctx.moveTo(100,100);
ctx.arc(100,100,75,sa,sa+fullCircle*seconds/60);
        ctx.fill();
```

```
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

Section 10.6: Animate at a specified time (an animated clock)

This example animates a clock showing the seconds as a filled wedge

Annotated Code:

```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;
    // canvas styling for the clock
    ctx.strokeStyle='lightgray';
    ctx.fillStyle='skyblue';
    ctx.lineWidth=5;

    // cache often used values
    var PI=Math.PI;
    var fullCircle=PI*2;
    var sa=-PI/2;   // == the 12 o'clock angle in context.arc

    // start the animation
    requestAnimationFrame(animate);

    function animate(currentTime){

        // get the current seconds value from the system clock
        var date=new Date();
        var seconds=date.getSeconds();

        // clear the canvas
        ctx.clearRect(0,0,cw,ch);

        // draw a full circle (== the clock face);
        ctx.beginPath();
        ctx.moveTo(100,100);
        ctx.arc(100,100,75,0,fullCircle);
        ctx.stroke();
        // draw a wedge representing the current seconds value
        ctx.beginPath();
        ctx.moveTo(100,100);
        ctx.arc(100,100,75,sa,sa+fullCircle*seconds/60);
        ctx.fill();
        // 请求另一个动画循环
requestAnimationFrame(animate);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

第10.7节：不要在事件处理程序中绘制动画（一个简单的草图应用）

在mousemove事件期间，你会每秒收到30个鼠标事件。你可能无法每秒重绘30次你的绘图。即使可以，你也可能在浏览器未准备好绘制时浪费计算资源（浪费 == 跨显示刷新周期）。

因此，将用户输入事件（如mousemove）与动画绘制分开处理是有意义的。

- 在事件处理程序中，保存所有控制绘图在画布上位置的事件变量。但不要实际绘制任何内容。
- 在requestAnimationFrame循环中，使用保存的信息将所有绘图渲染到画布上。

通过不在事件处理程序中绘制，避免了强制画布以鼠标事件速度刷新复杂绘图。

通过在requestAnimationFrame中完成所有绘制，你可以获得此处描述的所有好处——使用'requestAnimationFrame'而非'setInterval'进行动画循环。

带注释的代码：

```
<!doctype html>
<html>
<head>
<style>
body{ background-color: ivory; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    function log(){console.log.apply(console,arguments);}

    // canvas变量
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;
    // 设置画布样式
   ctx.strokeStyle='skyblue';
   ctx.lineJoint='round';
   ctx.lineCap='round';
   ctx.lineWidth=6;
```

```
        // request another loop of animation
        requestAnimationFrame(animate);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

Section 10.7: Don't draw animations in your event handlers (a simple sketch app)

During `mousemove` you get flooded with 30 mouse events per second. You might not be able to redraw your drawings at 30 times per second. Even if you can, you're probably wasting computing power by drawing when the browser is not ready to draw (wasted == across display refresh cycles).

Therefore it makes sense to separate your users input events (like mousemove) from the drawing of your animations.

- In event handlers, save all the event variables that control where drawings are positioned on the Canvas. But don't actually draw anything.
- In a `requestAnimationFrame` loop, render all the drawings to the Canvas using the saved information.

By not drawing in the event handlers, you are not forcing Canvas to try to refresh complex drawings at mouse event speeds.

By doing all drawing in `requestAnimationFrame` you gain all the benefits described in here Use 'requestanimationFrame' not 'setInterval' for animation loops.

Annotated Code:

```
<!doctype html>
<html>
<head>
<style>
    body{ background-color: ivory; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    function log(){console.log.apply(console,arguments);}

    // canvas variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;
    // set canvas styling
    ctx.strokeStyle='skyblue';
    ctx.lineJoint='round';
    ctx.lineCap='round';
    ctx.lineWidth=6;
    // 处理窗口滚动和调整大小
    function reOffset(){
        var BB=canvas.getBoundingClientRect();
        offsetX=BB.left;
        offsetY=BB.top;
    }
    var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
    window.onresize=function(e){ reOffset(); }

    // 用于保存鼠标移动处理期间创建的点的变量
    var points=[];
    var lastLength=0;

    // 启动动画循环
requestAnimationFrame(draw);

    canvas.onmousemove=function(e){handleMouseMove(e);}


    function handleMouseMove(e){
        // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();

        // 获取鼠标位置
mouseX=parseInt(e.clientX-offsetX);
        mouseY=parseInt(e.clientY-offsetY);

        // 将鼠标位置保存到 points[] 数组中
        // 但不绘制任何内容
points.push({x:mouseX,y:mouseY});
    }

    function draw(){
        // 没有新增点？请求下一帧并返回
        var length=points.length;
        if(length==lastLength){requestAnimationFrame(draw);return;}

        // 绘制新增点
        var point=points[lastLength];
        ctx.beginPath();
        ctx.moveTo(point.x,point.y)
        for(var i=lastLength;i<length;i++){
            point=points[i];
            ctx.lineTo(point.x,point.y);
        }
ctx.stroke();

        // 请求另一个动画循环
requestAnimationFrame(draw);
    }

}); // end window.onload
</script>
</head>
<body>
    <h4>将鼠标移到画布上进行绘图</h4>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

```
    // handle windows scrolling & resizing
    function reOffset(){
        var BB=canvas.getBoundingClientRect();
        offsetX=BB.left;
        offsetY=BB.top;
    }
    var offsetX,offsetY;
    reOffset();
    window.onscroll=function(e){ reOffset(); }
    window.onresize=function(e){ reOffset(); }

    // vars to save points created during mousemove handling
    var points=[];
    var lastLength=0;

    // start the  animation loop
    requestAnimationFrame(draw);

    canvas.onmousemove=function(e){handleMouseMove(e);}


    function handleMouseMove(e){
        // tell the browser we're handling this event
        e.preventDefault();
        e.stopPropagation();

        // get the mouse position
        mouseX=parseInt(e.clientX-offsetX);
        mouseY=parseInt(e.clientY-offsetY);

        // save the mouse position in the points[] array
        // but don't draw anything
        points.push({x:mouseX,y:mouseY});
    }

    function draw(){
        // No additional points? Request another frame an return
        var length=points.length;
        if(length==lastLength){requestAnimationFrame(draw);return;}

        // draw the additional points
        var point=points[lastLength];
        ctx.beginPath();
        ctx.moveTo(point.x,point.y)
        for(var i=lastLength;i<length;i++){
            point=points[i];
            ctx.lineTo(point.x,point.y);
        }
        ctx.stroke();

        // request another animation loop
        requestAnimationFrame(draw);
    }

}); // end window.onload
</script>
</head>
<body>
    <h4>Move mouse over Canvas to sketch</h4>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

第10.8节：使用2D上下文和requestAnimationFrame的简单动画

本示例将向您展示如何使用画布和2D上下文创建简单动画。假设您已经知道如何创建并添加画布到DOM中并获取上下文

```
// 本示例假设已创建ctx和canvas
const textToDisplay = "这是一个使用画布来动画显示文本的示例。";
const textStyle      = "white";
const BGStyle        = "black";   // 背景样式
const textSpeed      = 0.2;       // 每毫秒像素数
const textHorMargin = 8;         // 让文本稍微超出画布边缘


ctx.font = Math.floor(canvas.height * 0.8) + "px arial"; // 字体大小为画布高度的80%
var textWidth     = ctx.measureText(textToDisplay).width; // 获取文本宽度
var totalTextSize = (canvas.width + textHorMargin * 2 + textWidth);
ctx.textBaseline  = "middle";            // 使文本垂直居中
ctx.textAlign     = "left";              // 左对齐
var textX         = canvas.width + 8;    // 从画布右侧屏幕外开始显示文本
var textOffset    = 0;                   // 文本已移动的距离

var startTime;
// 该函数每帧调用一次，约为16.66毫秒（60fps）
function update(time){              // time由requestAnimationFrame传入
    if(startTime === undefined){    // 如果是第一帧，获取起始时间的引用
       startTime = time;
    }
ctx.fillStyle = BGStyle;
ctx.fillRect(0, 0, canvas.width, canvas.height);                   // 通过绘制覆盖清空画布

textOffset    = ((time - startTime) * textSpeed) % (totalTextSize); // 向左移动文本
    ctx.fillStyle = textStyle;                                       // 设置文本样式
    ctx.fillText(textToDisplay, textX - textOffset, canvas.height / 2); // 渲染文本

    requestAnimationFrame(update);// 完成后请求下一帧
}
requestAnimationFrame(update);// 启动时请求第一帧
```

此示例的演示 在 jsfiddle

第10.9节：从 [x0,y0] 动画到 [x1,y1]

使用向量计算从 [startX,startY] 到 [endX,endY] 的增量 [x,y]

```
// dx 是X方向上需要移动的总距离
var dx = endX - startX;

// dy 是在 Y 方向上移动的总距离
var dy = endY - startY;

// 使用百分比（pct）来移动总距离
// 从 0% 开始，即起点
// 以 100% 结束，即终点
var pct=0;

// 使用 dx 和 dy 计算当前 [x,y] 在给定百分比处的位置
var x = startX + dx * pct/100;
var y = startY + dx * pct/100;
```

Section 10.8: Simple animation with 2D context and requestAnimationFrame

This example will show you how to create a simple animation using the canvas and the 2D context. It is assumed you know how to create and add a canvas to the DOM and obtain the context

```
// this example assumes ctx and canvas have been created
const textToDisplay = "This is an example that uses the canvas to animate some text.";
const textStyle     = "white";
const BGStyle       = "black";  // background style
const textSpeed     = 0.2;      // in pixels per millisecond
const textHorMargin = 8;        // have the text a little outside the canvas


ctx.font = Math.floor(canvas.height * 0.8) + "px arial"; // size the font to 80% of canvas height
var textWidth     = ctx.measureText(textToDisplay).width; // get the text width
var totalTextSize = (canvas.width + textHorMargin * 2 + textWidth);
ctx.textBaseline  = "middle";           // not put the text in the vertical center
ctx.textAlign     = "left";             // align to the left
var textX         = canvas.width + 8;   // start with the text off screen to the right
var textOffset    = 0;                  // how far the text has moved

var startTime;
// this function is call once a frame which is approx 16.66 ms (60fps)
function update(time){              // time is passed by requestAnimationFrame
    if(startTime === undefined){    // get a reference for the start time if this is the first frame
        startTime = time;
    }
    ctx.fillStyle = BGStyle;
    ctx.fillRect(0, 0, canvas.width, canvas.height);                    // clear the canvas by
drawing over it
    textOffset    = ((time - startTime) * textSpeed) % (totalTextSize); // move the text left
    ctx.fillStyle = textStyle;                                          // set the text style
    ctx.fillText(textToDisplay, textX - textOffset, canvas.height / 2); // render the text

    requestAnimationFrame(update);// all done request the next frame
}
requestAnimationFrame(update);// to start request the first frame
```

A demo of this example at jsfiddle

Section 10.9: Animate from [x0,y0] to [x1,y1]

Use vectors to calculate incremental [x,y] from [startX,startY] to [endX,endY]

```
// dx is the total distance to move in the X direction
var dx = endX - startX;

// dy is the total distance to move in the Y direction
var dy = endY - startY;

// use a pct (percentage) to travel the total distances
// start at 0% which == the starting point
// end at 100% which == then ending point
var pct=0;

// use dx & dy to calculate where the current [x,y] is at a given pct
var x = startX + dx * pct/100;
var y = startY + dx * pct/100;
```

示例代码：

```
// 画布变量
var canvas=document.createElement("canvas");
document.body.appendChild(canvas);
canvas.style.border='1px solid red';
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
// 画布样式
ctx.strokeStyle='skyblue';
ctx.fillStyle='blue';

// 动画变量
var pct=101;
var startX=20;
var startY=50;
var endX=225;
var endY=100;
var dx=endX-startX;
var dy=endY-startY;

// 开始动画循环
requestAnimationFrame(animate);

// 监听鼠标事件
window.onmousedown=(function(e){handleMouseDown(e);});
window.onmouseup=(function(e){handleMouseUp(e);});

// 持续运行的循环
// 将点从 startX,startY 动画到 endX,endY
function animate(time){
    // 演示：重新运行动画
    if(++pct>100){pct=0;}
    // 更新
x=startX+dx*pct/100;
    y=startY+dy*pct/100;
    // 绘制
ctx.clearRect(0,0,cw,ch);
    ctx.beginPath();
ctx.moveTo(startX,startY);
    ctx.lineTo(endX,endY);
    ctx.stroke();
ctx.beginPath();
    ctx.arc(x,y,5,0,Math.PI*2);
    ctx.fill()
    // 请求另一个动画循环
requestAnimationFrame(animate);
}
```

Example Code:

```
// canvas vars
var canvas=document.createElement("canvas");
document.body.appendChild(canvas);
canvas.style.border='1px solid red';
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
// canvas styles
ctx.strokeStyle='skyblue';
ctx.fillStyle='blue';

// animating vars
var pct=101;
var startX=20;
var startY=50;
var endX=225;
var endY=100;
var dx=endX-startX;
var dy=endY-startY;

// start animation loop running
requestAnimationFrame(animate);

// listen for mouse events
window.onmousedown=(function(e){handleMouseDown(e);});
window.onmouseup=(function(e){handleMouseUp(e);});

// constantly running loop
// will animate dot from startX,startY to endX,endY
function animate(time){
    // demo: rerun animation
    if(++pct>100){pct=0;}
    // update
    x=startX+dx*pct/100;
    y=startY+dy*pct/100;
    // draw
    ctx.clearRect(0,0,cw,ch);
    ctx.beginPath();
    ctx.moveTo(startX,startY);
    ctx.lineTo(endX,endY);
    ctx.stroke();
    ctx.beginPath();
    ctx.arc(x,y,5,0,Math.PI*2);
    ctx.fill()
    // request another animation loop
    requestAnimationFrame(animate);
}
```

第11章：碰撞与交集

第11.1节：两个圆是否碰撞？

```
// 圆对象：{ x:, y:, radius: }
// 如果两个圆碰撞则返回true
// c1和c2是上述定义的圆

function CirclesColliding(c1,c2){
    var dx=c2.x-c1.x;
    var dy=c2.y-c1.y;
    var rSum=c1.radius+c2.radius;
    return(dx*dx+dy*dy<=rSum*rSum);
}
```

第11.2节：两个矩形是否碰撞？

```
// 矩形对象 { x:, y:, 宽度:, 高度: }
// 如果两个矩形碰撞则返回true
// r1和r2是上述定义的矩形

function 矩形碰撞(r1,r2){
    return !(
r1.x>r2.x+r2.宽度 ||
        r1.x+r1.宽度<r2.x ||
        r1.y>r2.y+r2.高度 ||
        r1.y+r1.高度<r2.y
    );
}
```

第11.3节：圆形和矩形是否碰撞？

```
// 矩形对象: { x:, y:, 宽度:, 高度: }
// 圆形对象: { x:, y:, 半径: }
// 如果矩形和圆形碰撞则返回true

function 矩形圆形碰撞(矩形,圆形){
    var dx=Math.abs(圆形.x-(矩形.x+矩形.宽度/2));
    var dy=Math.abs(圆形.y-(矩形.y+矩形.高度/2));

    if( dx > 圆形.半径+矩形.宽度/2 ){ return(false); }
    if( dy > 圆形.半径+矩形.高度/2 ){ return(false); }

    if( dx <= 矩形.宽度 ){ return(true); }
    if( dy <= 矩形.高度 ){ return(true); }

    var dx=dx-矩形.宽度;
    var dy=dy-rect.height
    return(dx*dx+dy*dy<=circle.radius*circle.radius);
}
```

第11.4节：两条线段是否相交？

此示例中的函数在两条线段相交时返回true，否则返回false。

该示例针对性能进行了设计，并使用闭包来保存工作变量

Chapter 11: Collisions and Intersections

Section 11.1: Are 2 circles colliding?

```
// circle objects: { x:, y:, radius: }
// return true if the 2 circles are colliding
// c1 and c2 are circles as defined above

function CirclesColliding(c1,c2){
    var dx=c2.x-c1.x;
    var dy=c2.y-c1.y;
    var rSum=c1.radius+c2.radius;
    return(dx*dx+dy*dy<=rSum*rSum);
}
```

Section 11.2: Are 2 rectangles colliding?

```
// rectangle objects { x:, y:, width:, height: }
// return true if the 2 rectangles are colliding
// r1 and r2 are rectangles as defined above

function RectsColliding(r1,r2){
    return !(
        r1.x>r2.x+r2.width ||
        r1.x+r1.width<r2.x ||
        r1.y>r2.y+r2.height ||
        r1.y+r1.height<r2.y
    );
}
```

Section 11.3: Are a circle and rectangle colliding?

```
// rectangle object: { x:, y:, width:, height: }
// circle object: { x:, y:, radius: }
// return true if the rectangle and circle are colliding

function RectCircleColliding(rect,circle){
    var dx=Math.abs(circle.x-(rect.x+rect.width/2));
    var dy=Math.abs(circle.y-(rect.y+rect.height/2));

    if( dx > circle.radius+rect.width/2 ){ return(false); }
    if( dy > circle.radius+rect.height/2 ){ return(false); }

    if( dx <= rect.width ){ return(true); }
    if( dy <= rect.height ){ return(true); }

    var dx=dx-rect.width;
    var dy=dy-rect.height
    return(dx*dx+dy*dy<=circle.radius*circle.radius);
}
```

Section 11.4: Are 2 line segments intercepting?

The function in this example returns **true** if two line segments are intersecting and **false** if not.

The example is designed for performance and uses closure to hold working variables

```
    // 点对象：{x:, y:}
    // p0 和 p1 组成一条线段，p2 和 p3 组成第二条线段
    // 如果线段相交则返回 true
                    var lineSegmentsIntercept = (function(){ // 作为单例的函数，以便使用闭包

        var v1, v2, v3, cross, u1, u2;  // 工作变量被闭包保存，因此不需要
每次调用函数时创建
                                 // 这显著提升了性能。

v1 = {x : null, y : null}; // 作为向量的线段 p0, p1
        v2 = {x : null, y : null}; // 作为向量的线段 p2, p3
        v3 = {x : null, y : null}; // 从 p0 到 p2 的线段作为向量

        function lineSegmentsIntercept (p0, p1, p2, p3) {
          v1.x = p1.x - p0.x; // 作为向量的线段 p0, p1
          v1.y = p1.y - p0.y;
v2.x = p3.x - p2.x; // 作为向量的线段 p2, p3
          v2.y = p3.y - p2.y;
            if((cross = v1.x * v2.y - v1.y * v2.x) === 0){  // 叉积为0表示线段平行
                return false; // 无交点
            }
v3 = {x : p0.x - p2.x, y : p0.y - p2.y};  // 从 p0 到 p2 的线段作为向量
            u2 = (v1.x * v3.y - v1.y * v3.x) / cross; // 获取沿线段 p2 p3 的单位距离
            // 代码点 B
            if (u2 >= 0 && u2 <= 1){                     // 交点在线段 p2, p3 上
                u1 = (v2.x * v3.y - v2.y * v3.x) / cross; // 获取沿线段 p0, p1 的单位距离;
                // 代码点 A
                return (u1 >= 0 && u1 <= 1);           // 如果在线段上返回 true，否则返回 false。
                // 代码点 A 结束
            }
            return false; // 无交点;
            // 代码点 B 结束
        }
        return lineSegmentsIntercept;  // 返回带闭包的函数以优化性能。
    })();
```

使用示例

```
var p1 = {x: 100, y: 0};   // 第1行
var p2 = {x: 120, y: 200};
var p3 = {x: 0,   y: 100}; // 第2行
var p4 = {x: 100, y: 120};
var areIntersepting = lineSegmentsIntercept (p1, p2, p3, p4); // true
```

该示例可以轻松修改以返回交点。将 `code point A` 和 `A end`

之间的代码替换为

```
if(u1 >= 0 && u1 <= 1){
    return {
x : p0.x + v1.x * u1,
        y : p0.y + v1.y * u1,
    };
}
```

或者如果你想获得直线上的交点，忽略线段的起点和终点，则将 `code point B` 和 `B end` 之间的代码替换为

```
return {
x : p2.x + v2.x * u2,
```

```
    // point object: {x:, y:}
    // p0 & p1 form one segment, p2 & p3 form the second segment
    // Returns true if lines segments are intercepting
    var lineSegmentsIntercept = (function(){ // function as singleton so that closure can be used

        var v1, v2, v3, cross, u1, u2;  // working variable are closed over so they do not need
creation
                                   // each time the function is called. This gives a significant
performance boost.
        v1 = {x : null, y : null}; // line p0, p1 as vector
        v2 = {x : null, y : null}; // line p2, p3 as vector
        v3 = {x : null, y : null}; // the line from p0 to p2 as vector

        function lineSegmentsIntercept (p0, p1, p2, p3) {
            v1.x = p1.x - p0.x; // line p0, p1 as vector
            v1.y = p1.y - p0.y;
            v2.x = p3.x - p2.x; // line p2, p3 as vector
            v2.y = p3.y - p2.y;
            if((cross = v1.x * v2.y - v1.y * v2.x) === 0){  // cross prod 0 if lines parallel
                return false; // no intercept
            }
            v3 = {x : p0.x - p2.x, y : p0.y - p2.y};  // the line from p0 to p2 as vector
            u2 = (v1.x * v3.y - v1.y * v3.x) / cross; // get unit distance along line p2 p3
            // code point B
            if (u2 >= 0 && u2 <= 1){                   // is intercept on line p2, p3
                u1 = (v2.x * v3.y - v2.y * v3.x) / cross; // get unit distance on line p0, p1;
                // code point A
                return (u1 >= 0 && u1 <= 1);           // return true if on line else false.
                // code point A end
            }
            return false; // no intercept;
            // code point B end
        }
        return lineSegmentsIntercept;  // return function with closure for optimisation.
    })();
```

Usage example

```
var p1 = {x: 100, y: 0};   // line 1
var p2 = {x: 120, y: 200};
var p3 = {x: 0,   y: 100}; // line 2
var p4 = {x: 100, y: 120};
var areIntersepting = lineSegmentsIntercept (p1, p2, p3, p4); // true
```

The example is easily modified to return the point of intercept. Replace the code between `code point A` and `A end` with

```
if(u1 >= 0 && u1 <= 1){
    return {
        x : p0.x + v1.x * u1,
        y : p0.y + v1.y * u1,
    };
}
```

Or if you want to get the intercept point on the lines, ignoring the line segments start and ends replace the code between `code point B` and `B end` with

```
return {
    x : p2.x + v2.x * u2,
y : p2.y + v2.y * u2,
};
```

如果没有交点，这两种修改都会返回 false，或者返回交点，格式为 {x : xCoord, y : yCoord}

第11.5节：线段和圆是否碰撞？

```
// [x0,y0] 到 [x1,y1] 定义一条线段
// [cx,cy] 是圆心，cr 是圆半径
function isCircleSegmentColliding(x0,y0,x1,y1,cx,cy,cr){

    // 计算增量距离：源点到线段起点
    var dx=cx-x0;
    var dy=cy-y0;

    // 计算增量距离：线段起点到终点
    var dxx=x1-x0;
    var dyy=y1-y0;

    // 计算线段上的位置，归一化到0.00到1.00之间
    // == 点积除以线段增量距离的平方
    var t=(dx*dxx+dy*dyy)/(dxx*dxx+dyy*dyy);

    // 计算线段上最近点
    var x=x0+dxx*t;
    var y=y0+dyy*t;

    // 将结果限制在线段上
    if(t<0){x=x0;y=y0;}
    if(t>1){x=x1;y=y1;}

    return( (cx-x)*(cx-x)+(cy-y)*(cy-y) < cr*cr );
}
```

第11.6节：线段和矩形是否碰撞？

```
// var rect={x:,y:,width:,height:};
// var line={x1:,y1:,x2:,y2:};
// 获取线段与矩形的交点（如果有）
function lineRectCollide(line,rect){

    // p=线段起点，p2=线段终点
    var p={x:line.x1,y:line.y1};
    var p2={x:line.x2,y:line.y2};

    // 矩形上边线
    var q={x:rect.x,y:rect.y};
    var q2={x:rect.x+rect.width,y:rect.y};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
    // 右边矩形边
    var q=q2;
    var q2={x:rect.x+rect.width,y:rect.y+rect.height};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
    // 底部矩形边
    var q=q2;
    var q2={x:rect.x,y:rect.y+rect.height};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
    // 左边矩形边
    var q=q2;
```

```
    y : p2.y + v2.y * u2,
};
```

Both modifications will return false if there is no intercept or return the point of intercept as {x : xCoord, y : yCoord}

Section 11.5: Are a line segment and circle colliding?

```
// [x0,y0] to [x1,y1] define a line segment
// [cx,cy] is circle centerpoint, cr is circle radius
function isCircleSegmentColliding(x0,y0,x1,y1,cx,cy,cr){

    // calc delta distance: source point to line start
    var dx=cx-x0;
    var dy=cy-y0;

    // calc delta distance: line start to end
    var dxx=x1-x0;
    var dyy=y1-y0;

    // Calc position on line normalized between 0.00 & 1.00
    // == dot product divided by delta line distances squared
    var t=(dx*dxx+dy*dyy)/(dxx*dxx+dyy*dyy);

    // calc nearest pt on line
    var x=x0+dxx*t;
    var y=y0+dyy*t;

    // clamp results to being on the segment
    if(t<0){x=x0;y=y0;}
    if(t>1){x=x1;y=y1;}

    return( (cx-x)*(cx-x)+(cy-y)*(cy-y) < cr*cr );
}
```

Section 11.6: Are line segment and rectangle colliding?

```
// var rect={x:,y:,width:,height:};
// var line={x1:,y1:,x2:,y2:};
// Get interseting point of line segment & rectangle (if any)
function lineRectCollide(line,rect){

    // p=line startpoint, p2=line endpoint
    var p={x:line.x1,y:line.y1};
    var p2={x:line.x2,y:line.y2};

    // top rect line
    var q={x:rect.x,y:rect.y};
    var q2={x:rect.x+rect.width,y:rect.y};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
    // right rect line
    var q=q2;
    var q2={x:rect.x+rect.width,y:rect.y+rect.height};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
    // bottom rect line
    var q=q2;
    var q2={x:rect.x,y:rect.y+rect.height};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
    // left rect line
    var q=q2;
    var q2={x:rect.x,y:rect.y};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }

    // 不与矩形的任何一条边相交
    return(false);
}

// 点对象：{x:, y:}
// p0 和 p1 组成一条线段，p2 和 p3 组成第二条线段
// 获取两条线段的交点（如果有）
// 归属：http://paulbourke.net/geometry/pointlineplane/
function lineSegmentsCollide(p0,p1,p2,p3) {

    var unknownA = (p3.x-p2.x) * (p0.y-p2.y) - (p3.y-p2.y) * (p0.x-p2.x);
    var unknownB = (p1.x-p0.x) * (p0.y-p2.y) - (p1.y-p0.y) * (p0.x-p2.x);
    var denominator  = (p3.y-p2.y) * (p1.x-p0.x) - (p3.x-p2.x) * (p1.y-p0.y);

    // 测试是否重合
    // 如果 ua 和 ub 的分母和分子均为 0
    //    则两条线重合。
    if(unknownA==0 && unknownB==0 && denominator==0){return(null);}

    // 测试是否平行
    // 如果 ua 和 ub 方程的分母为 0
    //     则两条线平行。
    if (denominator == 0) return null;

    // 测试线段是否相交
unknownA /= denominator;
unknownB /= denominator;
    var isIntersecting=(unknownA>=0 && unknownA<=1 && unknownB>=0 && unknownB<=1)

    return(isIntersecting);
}
```

第11.7节：两个凸多边形是否碰撞？

使用分离轴定理判断两个凸多边形是否相交

多边形必须是凸的

归属：Markus Jarderot @ 如何检查两个旋转矩形之间的相交？

```
// 多边形对象是由顶点数组组成的多边形
//     var polygon1=[{x:100,y:100},{x:150,y:150},{x:50,y:150},...];
// 多边形必须是凸的
// 如果两个多边形碰撞则返回true

function convexPolygonsCollide(a, b){
    var polygons = [a, b];
    var minA, maxA, projected, i, i1, j, minB, maxB;

    for (i = 0; i < polygons.length; i++) {

        // 对于每个多边形，查看多边形的每条边，判断它是否将两个形状分开

        var 多边形 = 多边形集合[i];
        for (i1 = 0; i1 < 多边形.长度; i1++) {

            // 获取两个顶点以创建一条边
            var i2 = (i1 + 1) % 多边形.长度;
```

```
    var q2={x:rect.x,y:rect.y};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }

    // not intersecting with any of the 4 rect sides
    return(false);
}

// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Get interseting point of 2 line segments (if any)
// Attribution: http://paulbourke.net/geometry/pointlineplane/
function lineSegmentsCollide(p0,p1,p2,p3) {

    var unknownA = (p3.x-p2.x) * (p0.y-p2.y) - (p3.y-p2.y) * (p0.x-p2.x);
    var unknownB = (p1.x-p0.x) * (p0.y-p2.y) - (p1.y-p0.y) * (p0.x-p2.x);
    var denominator  = (p3.y-p2.y) * (p1.x-p0.x) - (p3.x-p2.x) * (p1.y-p0.y);

    // Test if Coincident
    // If the denominator and numerator for the ua and ub are 0
    //    then the two lines are coincident.
    if(unknownA==0 && unknownB==0 && denominator==0){return(null);}

    // Test if Parallel
    // If the denominator for the equations for ua and ub is 0
    //     then the two lines are parallel.
    if (denominator == 0) return null;

    // test if line segments are colliding
    unknownA /= denominator;
    unknownB /= denominator;
    var isIntersecting=(unknownA>=0 && unknownA<=1 && unknownB>=0 && unknownB<=1)

    return(isIntersecting);
}
```

Section 11.7: Are 2 convex polygons colliding?

Use the Separating Axis Theorem to determine if 2 convex polygons are intersecting

THE POLYGONS MUST BE CONVEX

Attribution: Markus Jarderot @ How to check intersection between 2 rotated rectangles?

```
// polygon objects are an array of vertices forming the polygon
//     var polygon1=[{x:100,y:100},{x:150,y:150},{x:50,y:150},...];
// THE POLYGONS MUST BE CONVEX
// return true if the 2 polygons are colliding

function convexPolygonsCollide(a, b){
    var polygons = [a, b];
    var minA, maxA, projected, i, i1, j, minB, maxB;

    for (i = 0; i < polygons.length; i++) {

        // for each polygon, look at each edge of the polygon, and determine if it separates
        // the two shapes
        var polygon = polygons[i];
        for (i1 = 0; i1 < polygon.length; i1++) {

            // grab 2 vertices to create an edge
            var i2 = (i1 + 1) % polygon.length;
            var p1 = 多边形[i1];
            var p2 = 多边形[i2];

            // 找到这条边的垂线
            var 法线 = { x: p2.y - p1.y, y: p1.x - p2.x };

minA = maxA = 未定义;
            // 对第一个形状中的每个顶点，将其投影到垂直于
边的直线上
            // 并记录这些值的最小值和最大值
            for (j = 0; j < a.长度; j++) {
投影值 = 法线.x * a[j].x + 法线.y * a[j].y;
                if (minA==undefined || projected < minA) {
                    minA = projected;
                }
                if (maxA==undefined || projected > maxA) {
                    maxA = projected;
                }
            }

            // 对第二个形状中的每个顶点，将其投影到垂直于
边的直线上
            // 并跟踪这些投影值的最小值和最大值
minB = maxB = undefined;
            for (j = 0; j < b.length; j++) {
projected = normal.x * b[j].\     tx + normal.y * b[j].y;
                if (minB==undefined || projected < minB) {
                    minB = projected;
                }
                if (maxB==undefined || projected > maxB) {
                    maxB = projected;
                }
            }

            // 如果投影之间没有重叠，说明我们正在查看的边将
[illegible]
            // 多边形，且我们知道它们没有重叠
            if (maxA < minB || maxB < minA) {
                return false;
            }
        }
    }
    return true;
};
```

第11.8节：两个多边形是否碰撞？（允许凹多边形和凸多边形）

测试所有多边形边是否相交，以确定两个多边形是否碰撞。

```
// 多边形对象是由顶点组成的多边形数组
//     var polygon1=[{x:100,y:100},{x:150,y:150},{x:50,y:150},...];
// 多边形可以是凹的也可以是凸的
// 如果两个多边形碰撞则返回true

function polygonsCollide(p1,p2){
    // 将顶点转换为线段点
    var lines1=verticesToLinePoints(p1);
    var lines2=verticesToLinePoints(p2);
    // 测试每个poly1的边与每个poly2的边是否相交
    for(i=0; i<lines1.length; i++){
```

```
            var p1 = polygon[i1];
            var p2 = polygon[i2];

            // find the line perpendicular to this edge
            var normal = { x: p2.y - p1.y, y: p1.x - p2.x };

            minA = maxA = undefined;
            // for each vertex in the first shape, project it onto the line perpendicular to the
edge
            // and keep track of the min and max of these values
            for (j = 0; j < a.length; j++) {
                projected = normal.x * a[j].x + normal.y * a[j].y;
                if (minA==undefined || projected < minA) {
                    minA = projected;
                }
                if (maxA==undefined || projected > maxA) {
                    maxA = projected;
                }
            }

            // for each vertex in the second shape, project it onto the line perpendicular to the
edge
            // and keep track of the min and max of these values
            minB = maxB = undefined;
            for (j = 0; j < b.length; j++) {
                projected = normal.x * b[j].x + normal.y * b[j].y;
                if (minB==undefined || projected < minB) {
                    minB = projected;
                }
                if (maxB==undefined || projected > maxB) {
                    maxB = projected;
                }
            }

            // if there is no overlap between the projects, the edge we are looking at separates the
two
            // polygons, and we know there is no overlap
            if (maxA < minB || maxB < minA) {
                return false;
            }
        }
    }
    return true;
};
```

Section 11.8: Are 2 polygons colliding? (both concave and convex polys are allowed)

Tests all polygon sides for intersections to determine if 2 polygons are colliding.

```
// polygon objects are an array of vertices forming the polygon
//     var polygon1=[{x:100,y:100},{x:150,y:150},{x:50,y:150},...];
// The polygons can be both concave and convex
// return true if the 2 polygons are colliding

function polygonsCollide(p1,p2){
    // turn vertices into line points
    var lines1=verticesToLinePoints(p1);
    var lines2=verticesToLinePoints(p2);
    // test each poly1 side vs each poly2 side for intersections
    for(i=0; i<lines1.length; i++){
```

```
    for(j=0; j<lines2.length; j++){
        // 测试边是否相交
        var p0=lines1[i][0];
        var p1=lines1[i][1];
        var p2=lines2[j][0];
        var p3=lines2[j][1];
        // 找到交点——多边形相交
        if(lineSegmentsCollide(p0,p1,p2,p3)){return(true);}
    }}
    // 没有边相交
    return(false);
}
// 辅助函数：将顶点转换为线段点
function verticesToLinePoints(p){
    // 确保多边形是自闭合的
    if(!(p[0].x==p[p.length-1].x && p[0].y==p[p.length-1].y)){
p.push({x:p[0].x,y:p[0].y});
    }
    var lines=[];
    for(var i=1;i<p.length;i++){
        var p1=p[i-1];
        var p2=p[i];
lines.push([
            {x:p1.x, y:p1.y},
            {x:p2.x, y:p2.y}
        ]);
    }
    return(lines);
}
// 辅助函数：测试线段相交
// 点对象：{x:, y:}
// p0 和 p1 组成一条线段，p2 和 p3 组成第二条线段
// 获取两条线段的交点（如果有）
// 归属：http://paulbourke.net/geometry/pointlineplane/
function lineSegmentsCollide(p0,p1,p2,p3) {
    var unknownA = (p3.x-p2.x) * (p0.y-p2.y) - (p3.y-p2.y) * (p0.x-p2.x);
    var unknownB = (p1.x-p0.x) * (p0.y-p2.y) - (p1.y-p0.y) * (p0.x-p2.x);
    var denominator  = (p3.y-p2.y) * (p1.x-p0.x) - (p3.x-p2.x) * (p1.y-p0.y);

    // 测试是否重合
    // 如果 ua 和 ub 的分母和分子均为 0
    //    则两条线重合。
    if(unknownA==0 && unknownB==0 && denominator==0){return(null);}

    // 测试是否平行
    // 如果 ua 和 ub 方程的分母为 0
    //     则两条线平行。
    if (denominator == 0) return null;

    // 测试线段是否相交
unknownA /= denominator;
unknownB /= denominator;
    var isIntersecting=(unknownA>=0 && unknownA<=1 && unknownB>=0 && unknownB<=1)

    return(isIntersecting);
}
```

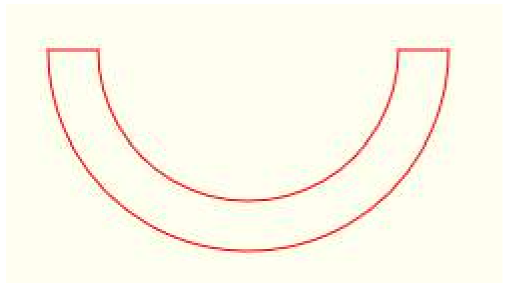
第11.9节：X,Y点是否在弧内？

测试[x,y]点是否在封闭弧内。

```
    for(j=0; j<lines2.length; j++){
        // test if sides intersect
        var p0=lines1[i][0];
        var p1=lines1[i][1];
        var p2=lines2[j][0];
        var p3=lines2[j][1];
        // found an intersection -- polys do collide
        if(lineSegmentsCollide(p0,p1,p2,p3)){return(true);}
    }}
    // none of the sides intersect
    return(false);
}
// helper: turn vertices into line points
function verticesToLinePoints(p){
    // make sure polys are self-closing
    if(!(p[0].x==p[p.length-1].x && p[0].y==p[p.length-1].y)){
        p.push({x:p[0].x,y:p[0].y});
    }
    var lines=[];
    for(var i=1;i<p.length;i++){
        var p1=p[i-1];
        var p2=p[i];
        lines.push([
            {x:p1.x, y:p1.y},
            {x:p2.x, y:p2.y}
        ]);
    }
    return(lines);
}
// helper: test line intersections
// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Get interseting point of 2 line segments (if any)
// Attribution: http://paulbourke.net/geometry/pointlineplane/
function lineSegmentsCollide(p0,p1,p2,p3) {
    var unknownA = (p3.x-p2.x) * (p0.y-p2.y) - (p3.y-p2.y) * (p0.x-p2.x);
    var unknownB = (p1.x-p0.x) * (p0.y-p2.y) - (p1.y-p0.y) * (p0.x-p2.x);
    var denominator  = (p3.y-p2.y) * (p1.x-p0.x) - (p3.x-p2.x) * (p1.y-p0.y);

    // Test if Coincident
    // If the denominator and numerator for the ua and ub are 0
    //    then the two lines are coincident.
    if(unknownA==0 && unknownB==0 && denominator==0){return(null);}

    // Test if Parallel
    // If the denominator for the equations for ua and ub is 0
    //     then the two lines are parallel.
    if (denominator == 0) return null;

    // test if line segments are colliding
    unknownA /= denominator;
    unknownB /= denominator;
    var isIntersecting=(unknownA>=0 && unknownA<=1 && unknownB>=0 && unknownB<=1)

    return(isIntersecting);
}
```

Section 11.9: Is an X,Y point inside an arc?

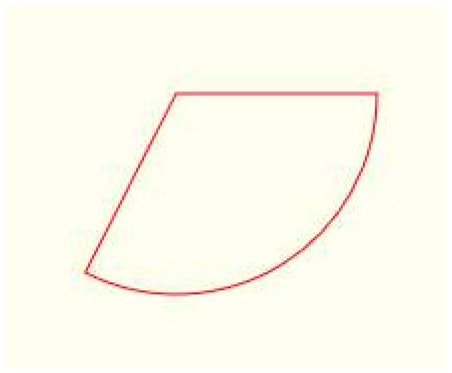
Tests if the [x,y] point is inside a closed arc.



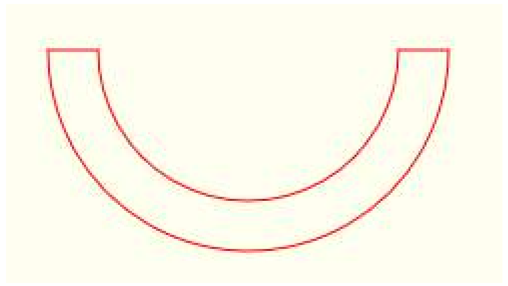
```
var arc={
cx:150, cy:150,
    innerRadius:75, outerRadius:100,
    startAngle:0, endAngle:Math.PI
}

function isPointInArc(x,y,arc){
    var dx=x-arc.cx;
    var dy=y-arc.cy;
    var dxy=dx*dx+dy*dy;
    var rrOuter=arc.outerRadius*arc.outerRadius;
    var rrInner=arc.innerRadius*arc.innerRadius;
    if(dxy<rrInner || dxy>rrOuter){return(false);}
    var angle=(Math.atan2(dy,dx)+PI2)%PI2;
    return(angle>=arc.startAngle && angle<=arc.endAngle);
}
```

第11.10节：点（X，Y）是否在扇形内？

测试点 [x,y] 是否在扇形内。



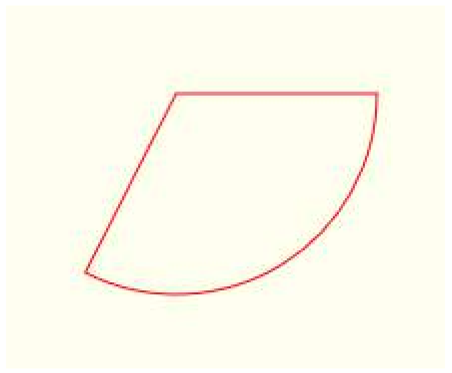
```
// 扇形对象：{cx:,cy:,radius:,startAngle:,endAngle:}
// var wedge={
//     cx:150, cy:150,   // 中心点
//      radius:100,
//     startAngle:0, endAngle:Math.PI
// }
// 如果点 x,y 在闭合扇形内则返回 true

function isPointInWedge(x,y,wedge){
    var PI2=Math.PI*2;
    var dx=x-wedge.cx;
    var dy=y-wedge.cy;
    var rr=wedge.radius*wedge.radius;
    if(dx*dx+dy*dy>rr){return(false);}
    var angle=(Math.atan2(dy,dx)+PI2)%PI2;
    return(angle>=wedge.startAngle && angle<=wedge.endAngle);
}
```



```
var arc={
    cx:150, cy:150,
    innerRadius:75, outerRadius:100,
    startAngle:0, endAngle:Math.PI
}

function isPointInArc(x,y,arc){
    var dx=x-arc.cx;
    var dy=y-arc.cy;
    var dxy=dx*dx+dy*dy;
    var rrOuter=arc.outerRadius*arc.outerRadius;
    var rrInner=arc.innerRadius*arc.innerRadius;
    if(dxy<rrInner || dxy>rrOuter){return(false);}
    var angle=(Math.atan2(dy,dx)+PI2)%PI2;
    return(angle>=arc.startAngle && angle<=arc.endAngle);
}
```

Section 11.10: Is an X,Y point inside a wedge?

Tests if the [x,y] point is inside a wedge.



```
// wedge objects: {cx:,cy:,radius:,startAngle:,endAngle:}
// var wedge={
//     cx:150, cy:150,  // centerpoint
//     radius:100,
//     startAngle:0, endAngle:Math.PI
// }
// Return true if the x,y point is inside the closed wedge

function isPointInWedge(x,y,wedge){
    var PI2=Math.PI*2;
    var dx=x-wedge.cx;
    var dy=y-wedge.cy;
    var rr=wedge.radius*wedge.radius;
    if(dx*dx+dy*dy>rr){return(false);}
    var angle=(Math.atan2(dy,dx)+PI2)%PI2;
    return(angle>=wedge.startAngle && angle<=wedge.endAngle);
}
```

第11.11节：点 X,Y 是否在圆内？

测试点 [x,y] 是否在圆内。

```
// 圆对象: {cx:,cy:,radius:,startAngle:,endAngle:}
// var circle={
//     cx:150, cy:150,  // 中心点
//     radius:100,
// }
// 如果点 x,y 在圆内则返回 true

function isPointInCircle(x,y,circle){
    var dx=x-circle.cx;
    var dy=y-circle.cy;
    return(dx*dx+dy*dy<circle.radius*circle.radius);
}
```

第11.12节：点（X,Y）是否在矩形内？

测试一个[x,y]点是否在矩形内。

```
// 矩形对象：{x:, y:, width:, height: }
// var rect={x:10, y:15, width:25, height:20}
// 如果x,y点在矩形内则返回true

function isPointInRectangle(x,y,rect){
    return(x>rect.x && x<rect.x+rect.width && y>rect.y && y<rect.y+rect.height);
}
```

Section 11.11: Is an X,Y point inside a circle?

Tests if an [x,y] point is inside a circle.

```
// circle objects: {cx:,cy:,radius:,startAngle:,endAngle:}
// var circle={
//     cx:150, cy:150,  // centerpoint
//     radius:100,
// }
// Return true if the x,y point is inside the circle

function isPointInCircle(x,y,circle){
    var dx=x-circle.cx;
    var dy=y-circle.cy;
    return(dx*dx+dy*dy<circle.radius*circle.radius);
}
```

Section 11.12: Is an X,Y point inside a rectangle?

Tests if an [x,y] point is inside a rectangle.

```
// rectangle objects: {x:, y:, width:, height: }
// var rect={x:10, y:15, width:25, height:20}
// Return true if the x,y point is inside the rectangle

function isPointInRectangle(x,y,rect){
    return(x>rect.x && x<rect.x+rect.width && y>rect.y && y<rect.y+rect.height);
}
```

第12章：清除屏幕

第12.1节：矩形

您可以使用clearRect方法来清除画布上的任意矩形区域。

```
// 清除整个画布
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

> 注意：clearRect 依赖于变换矩阵。

为了解决这个问题，可以在清除画布之前重置变换矩阵。

```
ctx.save();                                          // 保存当前上下文状态
ctx.setTransform(1, 0, 0, 1, 0, 0);                  // 重置变换矩阵
ctx.clearRect(0, 0, canvas.width, canvas.height);    // 清除画布
ctx.restore();                                       // 恢复上下文状态，包括
                                                     // 变换矩阵
```

> 注意：ctx.save 和 ctx.restore 仅在你希望保留画布2D上下文状态时才需要。在某些情况下，save和restor e可能较慢，通常如果不必要应避免使用。

第12.2节：使用渐变清除画布

你可能不想使用使所有像素透明的 clearRect，而是想要一个背景。

使用渐变清除画布的方法

```
// 创建背景渐变一次
var bgGrad = ctx.createLinearGradient(0,0,0,canvas.height);
bgGrad.addColorStop(0,"#0FF");
bgGrad.addColorStop(1,"#08F");

// 每次需要清除画布时
ctx.fillStyle = bgGrad;
ctx.fillRect(0,0,canvas.width,canvas.height);
```

这大约是 clearRect 0.004ms 的一半速度 0.008ms，但这几百万分之一秒的差异不会对任何实时动画产生负面影响。（时间会根据设备、分辨率、浏览器及浏览器配置有较大差异。时间仅供比较）

第12.3节：使用复合操作清除画布

使用复合操作清除画布。这将独立于变换清除画布，但速度不如 clearRect() 快。

```
ctx.globalCompositeOperation = 'copy';
```

接下来绘制的任何内容都会清除之前的内容。

Chapter 12: Clearing the screen

Section 12.1: Rectangles

You can use the `clearRect` method to clear any rectangular section of the canvas.

```
// Clear the entire canvas
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

> **Note:** `clearRect` is dependent on the transformation matrix.

To deal with this, it's possible to reset the transformation matrix before you clear the canvas.

```
ctx.save();                                          // Save the current context state
ctx.setTransform(1, 0, 0, 1, 0, 0);                  // Reset the transformation matrix
ctx.clearRect(0, 0, canvas.width, canvas.height);    // Clear the canvas
ctx.restore();                                       // Revert context state including
                                                     // transformation matrix
```

> **Note:** `ctx.save` and `ctx.restore` are only requiered if you wish to keep the canvas 2D context state. In some situations save and restore can be be slow and generally should be avoided if not required.

Section 12.2: Clear canvas with gradient

Rather than use `clearRect` which makes all pixels transparent you may want a background.

To clear with a gradient

```
// create the background gradient once
var bgGrad = ctx.createLinearGradient(0,0,0,canvas.height);
bgGrad.addColorStop(0,"#0FF");
bgGrad.addColorStop(1,"#08F");

// Every time you need to clear the canvas
ctx.fillStyle = bgGrad;
ctx.fillRect(0,0,canvas.width,canvas.height);
```

This is about half as quick `0.008ms` as clearRect `0.004ms` but the 4millions of a second should not negatively impact any realtime animation. (Times will vary considerably depending on device, resolution, browser, and browser configuration. Times are for comparison only)

Section 12.3: Clear canvas using composite operation

Clear the canvas using compositing operation. This will clear the canvas independent of transforms but is not as fast as `clearRect()`.

```
ctx.globalCompositeOperation = 'copy';
```

anything drawn next will clear previous content.

第12.4节：原始图像数据

可以使用`putImageData`直接写入渲染后的图像数据。通过创建新的图像数据然后将其赋值给画布，您将清空整个屏幕。

```
var imageData = ctx.createImageData(canvas.width, canvas.height);
ctx.putImageData(imageData, 0, 0);
```

注意：`putImageData` 不受应用于上下文的任何变换影响。它将数据直接写入渲染的像素区域。

第12.5节：复杂形状

可以通过更改globalCompositeOperation属性来清除复杂形状区域。

```
// 所有绘制的像素将变为透明
ctx.globalCompositeOperation = 'destination-out';

// 清除一个三角形区域
ctx.globalAlpha = 1;    // 确保alpha为1
ctx.fillStyle = '#000'; // 确保当前填充样式没有任何透明度
ctx.beginPath();
ctx.moveTo(10, 0);
ctx.lineTo(0, 10);
ctx.lineTo(20, 10);
ctx.fill();

// 重新开始正常绘制
ctx.globalCompositeOperation = 'source-over';
```

Section 12.4: Raw image data

It's possible to write directly to the rendered image data using `putImageData`. By creating new image data then assigning it to the canvas, you will clear the entire screen.

```
var imageData = ctx.createImageData(canvas.width, canvas.height);
ctx.putImageData(imageData, 0, 0);
```

Note: `putImageData` is not affected by any transformations applied to the context. It will write data directly to the rendered pixel region.

Section 12.5: Complex shapes

It's possible to clear complex shaped regions by changing the `globalCompositeOperation` property.

```
// All pixels being drawn will be transparent
ctx.globalCompositeOperation = 'destination-out';

// Clear a triangular section
ctx.globalAlpha = 1;    // ensure alpha is 1
ctx.fillStyle = '#000'; // ensure the current fillStyle does not have any transparency
ctx.beginPath();
ctx.moveTo(10, 0);
ctx.lineTo(0, 10);
ctx.lineTo(20, 10);
ctx.fill();

// Begin drawing normally again
ctx.globalCompositeOperation = 'source-over';
```

第13章：响应式设计

第13.1节：创建响应式全屏画布

用于通过JavaScript创建和移除响应窗口大小变化事件的全屏画布的起始代码。

```
var canvas;    // 全局画布引用
var ctx;       // 全局2D上下文引用
// 创建画布
function createCanvas () {
    const canvas = document.createElement("canvas");
    canvas.style.position = "absolute"; // 设置样式
    canvas.style.left     = "0px";      // 定位到左上角
   canvas.style.top       = "0px";
canvas.style.zIndex   = 1;
    document.body.appendChild(canvas);  // 添加到文档中
    return canvas;
}
// 调整画布大小。如果画布不存在则创建画布
function sizeCanvas () {
    if (canvas === undefined) {         // 检查全局画布引用
        canvas = createCanvas();        // 创建新的画布元素
        ctx = canvas.getContext("2d");  // 获取2D上下文
    }
canvas.width  = innerWidth;          // 设置画布分辨率以填充页面
    canvas.height = innerHeight;
}
// 移除画布
function removeCanvas () {
    if (canvas !== undefined) {              // 确保有东西可以移除
        removeEventListener("resize", sizeCanvas); // 移除窗口大小调整事件
        document.body.removeChild(canvas);   // 从DOM中移除画布
        ctx = undefined;                     // 取消对上下文的引用
canvas = undefined;                 // 取消对画布的引用
     }
}

// 添加窗口大小调整监听器
addEventListener("resize", sizeCanvas);
// 调用 sizeCanvas 来创建并设置画布分辨率
sizeCanvas();
// 现在 ctx 和 canvas 可供使用。
```

如果不再需要画布，可以调用 `removeCanvas()` 来移除它

[此示例的演示](#) 在 jsfiddle

第13.2节：调整大小（或滚动）后的鼠标坐标

画布应用程序通常高度依赖用户与鼠标的交互，但当窗口调整大小时，画布依赖的鼠标事件坐标很可能会发生变化，因为调整大小会导致画布相对于窗口的位置发生偏移。因此，响应式设计要求在窗口调整大小时重新计算画布的偏移位置——并且在窗口滚动时也要重新计算。

此代码监听窗口调整大小事件，并重新计算鼠标事件处理程序中使用的偏移量：

```
// 保存当前画布相对于窗口的偏移位置的变量
```

Chapter 13: Responsive Design

Section 13.1: Creating a responsive full page canvas

Starter code to create and remove a full page canvas that responds to resize events via JavaScript.

```
var canvas;    // Global canvas reference
var ctx;       // Global 2D context reference
// Creates a canvas
function createCanvas () {
    const canvas = document.createElement("canvas");
    canvas.style.position = "absolute"; // Set the style
    canvas.style.left     = "0px";      // Position in top left
    canvas.style.top      = "0px";
    canvas.style.zIndex   = 1;
    document.body.appendChild(canvas);  // Add to document
    return canvas;
}
// Resizes canvas. Will create a canvas if it does not exist
function sizeCanvas () {
    if (canvas === undefined) {         // Check for global canvas reference
        canvas = createCanvas();        // Create a new canvas element
        ctx = canvas.getContext("2d");  // Get the 2D context
    }
    canvas.width  = innerWidth;         // Set the canvas resolution to fill the page
    canvas.height = innerHeight;
}
// Removes the canvas
function removeCanvas () {
    if (canvas !== undefined) {              // Make sure there is something to remove
        removeEventListener("resize", sizeCanvas); // Remove resize event
        document.body.removeChild(canvas);   // Remove the canvas from the DOM
        ctx = undefined;                     // Dereference the context
        canvas = undefined;                  // Dereference the canvas
     }
}

// Add the resize listener
addEventListener("resize", sizeCanvas);
// Call sizeCanvas to create and set the canvas resolution
sizeCanvas();
// ctx and canvas are now available for use.
```

If you no longer need the canvas you can remove it by calling `removeCanvas()`

[A demo of this example](#) at jsfiddle

Section 13.2: Mouse coordinates after resizing (or scrolling)

Canvas apps often rely heavily on user interaction with the mouse, but when the window is resized, the mouse event coordinates that canvas relies on are likely changed because resizing causes the canvas to be offset in a different position relative to the window. Thus, responsive design requires that the canvas offset position be recalculated when the window is resized -- and also recalculated when the window is scrolled.

This code listens for window resizing events and recalculates the offsets used in mouse event handlers:

```
// variables holding the current canvas offset position
//    relative to the window
```

```
var offsetX,offsetY;

// 重新计算画布偏移量的函数
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
offsetY=BB.top;
}

// 监听窗口调整大小（和滚动）事件
// 然后重新计算画布偏移量
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }

// 在鼠标处理程序中使用偏移量的示例
function handleMouseUp(e){
    // 使用 offsetX 和 offsetY 获取正确的鼠标位置
mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // ...
}
```

第13.3节：无调整大小事件的响应式画布动画

窗口大小调整事件可能会响应用户输入设备的移动而触发。当你调整画布大小时，画布会自动被清空，你必须重新渲染内容。对于动画来说，你需要通过主循环函数每帧执行此操作，该函数由requestAnimationFrame调用，尽力保持渲染与显示硬件的同步。

调整大小事件的问题在于，当使用鼠标调整窗口大小时，事件触发的频率可能远快于浏览器标准的60fps速率。当调整大小事件退出时，画布的后备缓冲区会以与显示设备不同步的方式呈现给DOM，这可能导致剪切和其他负面效果。此外，还有大量不必要的内存分配和释放，这会在垃圾回收（GC）稍后清理时进一步影响动画。

防抖调整大小事件

处理调整大小事件高频触发的常用方法是对调整大小事件进行防抖处理。

```
 // 假设canvas在作用域内
addEventListener.("resize", debouncedResize );

 // 防抖超时句柄
 var debounceTimeoutHandle;

 // 防抖时间，单位毫秒（千分之一秒）
 const DEBOUNCE_TIME = 100;

 // 调整大小函数
 function debouncedResize () {
clearTimeout(debounceTimeoutHandle);  // 清除任何待处理的防抖事件

     // 调度画布大小调整
debounceTimeoutHandle = setTimeout(resizeCanvas, DEBOUNCE_TIME);
 }
```

```
var offsetX,offsetY;

// a function to recalculate the canvas offsets
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
}

// listen for window resizing (and scrolling) events
//     and then recalculate the canvas offsets
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }

// example usage of the offsets in a mouse handler
function handleMouseUp(e){
    // use offsetX & offsetY to get the correct mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // ...
}
```

Section 13.3: Responsive canvas animations without resize events

The window resize events can fire in response to the movement of the user's input device. When you resize a canvas it is automatically cleared and you are forced to re-render the content. For animations you do this every frame via the main loop function called by `requestAnimationFrame` which does its best to keep the rendering in sync with the display hardware.

The problem with the resize event is that when the mouse is used to resize the window the events can be trigger many times quicker than the standard 60fps rate of the browser. When the resize event exits the canvas back buffer is presented to the DOM out of sync with the display device, which can cause shearing and other negative effects. There is also a lot of needless memory allocation and release that can further impact the animation when GC cleans up some time afterwards.

Debounced resize event

A common way to deal with the high firing rates of the resize event is to debounce the resize event.

```
// Assume canvas is in scope
addEventListener.("resize", debouncedResize );

// debounce timeout handle
var debounceTimeoutHandle;

// The debounce time in ms (1/1000th second)
const DEBOUNCE_TIME = 100;

// Resize function
function debouncedResize () {
    clearTimeout(debounceTimeoutHandle);  // Clears any pending debounce events

    // Schedule a canvas resize
    debounceTimeoutHandle = setTimeout(resizeCanvas, DEBOUNCE_TIME);
}
```

```
// 画布大小调整函数
function resizeCanvas () { ... 调整大小并重绘 ... }
```

上述示例将画布的大小调整延迟到调整事件发生后100毫秒。如果在这段时间内触发了更多的调整事件，现有的调整超时将被取消并重新调度。这样有效地消耗了大部分调整事件。

它仍然存在一些问题，最显著的是调整大小与看到调整后画布之间的延迟。减少防抖时间可以改善这一点，但调整仍然与显示设备不同步。动画主循环仍然会在不合适的画布上渲染。

更多代码可以减少问题！但更多代码也会带来新的问题。

简单且最佳的调整大小

尝试了许多不同的方法来平滑画布的调整大小，从极其复杂到干脆忽略问题（反正谁在乎呢？），我最终回归了一个可靠的朋友。

K.I.S.S.\ 是大多数程序员应该知道的原则（（Keep It Simple Stupid）这里的“stupid”指的是我自己，没能多年前就想到它。）事实证明，最好的解决方案往往是最简单的。

只需在主动画循环中调整画布大小。它会与显示设备保持同步，无需多余的渲染，并且在保持全帧率的同时，将资源管理降到最低。
你也不需要为窗口添加调整大小事件或任何额外的调整大小函数。

你可以在通常清除画布的地方添加调整大小的代码，通过检查画布大小是否与窗口大小匹配。如果不匹配，则调整大小
。

```
// 假设画布元素在作用域内，变量名为 canvas

// 来自 requestAnimationFrame 的标准主循环函数回调
function mainLoop(time) {

    // 检查画布大小是否与窗口大小匹配
    if (canvas.width !== innerWidth || canvas.height !== innerHeight) {
        canvas.width = innerWidth;     // 调整画布大小
        canvas.height = innerHeight;   // 同时清除画布
    } else {
ctx.clearRect(0, 0, canvas.width, canvas.height); // 如果未调整大小则清除画布
    }

    // 动画代码照常执行。



requestAnimationFrame(mainLoop);
}
```

```
// canvas resize function
function resizeCanvas () { ... resize and redraw ... }
```

The above example delays the resizing of the canvas until 100ms after the resize event. If in that time further resize events are triggered the existing resize timeout is canceled and a new one scheduled. This effectively consumes most of the resize events.

It still has some problems, the most notable is the delay between resizing and seeing the resized canvas. Reducing the debounce time improves this but the resize is still out of sync with the display device. You also still have the animation main loop rendering to an ill fitting canvas.

More code can reduce the problems! More code also creates its own new problems.

Simple and the best resize

Having tried many differing ways to smooth out the resizing of the canvas, from the absurdly complex, to just ignoring the problem (who cares anyways?) I fell back to a trusty friend.

K.I.S.S. is something most programmers should be aware of ((**K**eep **I**t **S**imple **S**tupid) *The stupid refers to me for not having thought of it years ago.*) and it turns out the best solution is the simplest of all.

Just resize the canvas from within the main animation loop. It stays in sync with the display device, there is no needless rendering, and the resource management is at the minimum possible while maintaining full frame rate. Nor do you need to add a resize event to the window or any additional resize functions.

You add the resize where you would normally clear the canvas by checking if the canvas size matches the window size. If not resize it.

```
// Assumes canvas element is in scope as canvas

// Standard main loop function callback from requestAnimationFrame
function mainLoop(time) {

    // Check if the canvas size matches the window size
    if (canvas.width !== innerWidth || canvas.height !== innerHeight) {
        canvas.width = innerWidth;    // resize canvas
        canvas.height = innerHeight;  // also clears the canvas
    } else {
        ctx.clearRect(0, 0, canvas.width, canvas.height); // clear if not resized
    }

    // Animation code as normal.



    requestAnimationFrame(mainLoop);
}
```

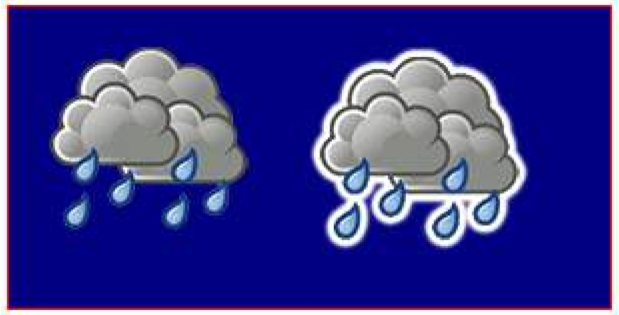
第14章：阴影

第14.1节：使用阴影的贴纸效果

此代码为图像添加向外扩散的阴影，以创建图像的“贴纸”版本。

注意事项：

- 除了作为ImageObject之外，“img”参数也可以是Canvas元素。这允许你对自定义绘图进行贴纸化。如果你在Canvas参数上绘制文本，也可以对该文本进行贴纸化。
- 完全不透明的图像不会有贴纸效果，因为该效果是绘制在由透明像素包围的不透明像素簇周围的。



```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.style.background='navy';
canvas.style.border='1px solid red;';

// 始终(!) 等待图像完全加载后再尝试使用drawImage绘制！
var img=new Image();
img.onload=start;
// 在这里放置你的 img.src ...
img.src='http://i.stack.imgur.com/bXaB6.png';
function start(){
ctx.drawImage(img,20,20);
    var sticker=stickerEffect(img,5);
    ctx.drawImage(sticker, 150,20);
}

function stickerEffect(img,grow){
    var canvas1=document.createElement("canvas");
    var ctx1=canvas1.getContext("2d");
    var canvas2=document.createElement("canvas");
    var ctx2=canvas2.getContext("2d");
    canvas1.width=canvas2.width=img.width+grow*2;
    canvas1.height=canvas2.height=img.height+grow*2;
    ctx1.drawImage(img,grow,grow);
ctx2.shadowColor='white';
    ctx2.shadowBlur=2;
    for(var i=0;i<grow;i++){
        ctx2.drawImage(canvas1,0,0);
        ctx1.drawImage(canvas2,0,0);
    }
ctx2.shadowColor='rgba(0,0,0,0)';
    ctx2.drawImage(img,grow,grow);
    return(canvas2);
```

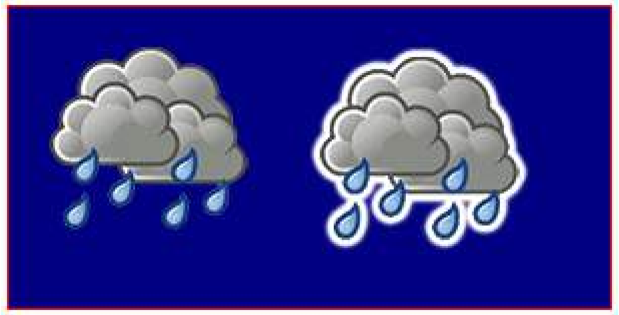
Chapter 14: Shadows

Section 14.1: Sticker effect using shadows

This code adds outwardly increasing shadows to an image to create a "sticker" version of the image.

Notes:

- In addition to being an ImageObject, the "img" argument can also be a Canvas element. This allows you to stickerize your own custom drawings. If you draw text on the Canvas argument, you can also stickerize that text.
- Fully opaque images will have no sticker effect because the effect is drawn around clusters of opaque pixels that are bordered by transparent pixels.



```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.style.background='navy';
canvas.style.border='1px solid red;';

// Always(!) wait for your images to fully load before trying to drawImage them!
var img=new Image();
img.onload=start;
// put your img.src here...
img.src='http://i.stack.imgur.com/bXaB6.png';
function start(){
    ctx.drawImage(img,20,20);
    var sticker=stickerEffect(img,5);
    ctx.drawImage(sticker, 150,20);
}

function stickerEffect(img,grow){
    var canvas1=document.createElement("canvas");
    var ctx1=canvas1.getContext("2d");
    var canvas2=document.createElement("canvas");
    var ctx2=canvas2.getContext("2d");
    canvas1.width=canvas2.width=img.width+grow*2;
    canvas1.height=canvas2.height=img.height+grow*2;
    ctx1.drawImage(img,grow,grow);
    ctx2.shadowColor='white';
    ctx2.shadowBlur=2;
    for(var i=0;i<grow;i++){
        ctx2.drawImage(canvas1,0,0);
        ctx1.drawImage(canvas2,0,0);
    }
    ctx2.shadowColor='rgba(0,0,0,0)';
    ctx2.drawImage(img,grow,grow);
    return(canvas2);
}
```

第14.2节：如何停止进一步的阴影效果

一旦开启阴影效果，画布上的每一次新绘制都会带有阴影。

通过将context.shadowColor设置为透明颜色来关闭进一步的阴影效果。

```
// 开始投影
context.shadowColor='black';

... 渲染 一些带阴影的绘图 ...

// 关闭投影。
context.shadowColor='rgba(0,0,0,0)';
```

第14.3节：投影计算开销大——缓存那个阴影！

警告！请谨慎使用阴影！

应用阴影开销大，如果在动画循环中应用阴影，开销会成倍增加。

相反，缓存图像（或其他绘图）的带阴影版本：

- 在应用程序开始时，在第二个仅内存中的画布中创建图像的带阴影版本：var memoryCanvas = document.createElement('canvas') ...
- 每当需要带阴影的版本时，从内存画布将预先带阴影的图像绘制到可见画布上：context.drawImage(memoryCanvas,x,y)

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
canvas.style.border='1px solid red;';
document.body.appendChild(canvas);

// 始终(!) 使用 "img.onload" 给你的图片足够时间
// 完全加载后再尝试将其绘制到画布上！
var img=new Image();
img.onload=start;
// 在这里放置你自己的 img.src
img.src="http://i.stack.imgur.com/hYFNe.png";
function start(){
ctx.drawImage(img,0,20);
    var cached=cacheShadowedImage(img,'black',5,3,3);
    for(var i=0;i<5;i++){
```

```
}
```

Section 14.2: How to stop further shadowing

Once shadowing is turned on, every new drawing to the canvas will be shadowed.

Turn off further shadowing by setting `context.shadowColor` to a transparent color.

```
// start shadowing
context.shadowColor='black';

... render some shadowed drawings ...

// turn off shadowing.
context.shadowColor='rgba(0,0,0,0)';
```

Section 14.3: Shadowing is computationally expensive -- Cache that shadow!

Warning! Apply shadows sparingly!

Applying shadowing is expensive and is multiplicatively expensive if you apply shadowing inside an animation loop.

Instead, cache a shadowed version of your image (or other drawing):

- At the start of your app, create a shadowed version of your image in a second in-memory-only Canvas: `var memoryCanvas = document.createElement('canvas') ...`
- Whenever you need the shadowed version, draw that pre-shadowed image from the in-memory canvas to the visible canvas: `context.drawImage(memoryCanvas,x,y)`

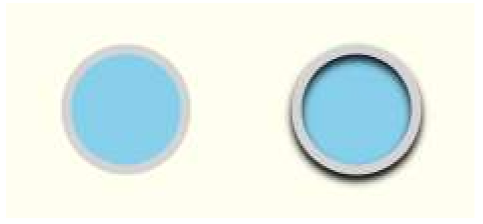
```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
canvas.style.border='1px solid red;';
document.body.appendChild(canvas);

// Always(!) use "img.onload" to give your image time to
//     fully load before you try drawing it to the Canvas!
var img=new Image();
img.onload=start;
// Put your own img.src here
img.src="http://i.stack.imgur.com/hYFNe.png";
function start(){
    ctx.drawImage(img,0,20);
    var cached=cacheShadowedImage(img,'black',5,3,3);
    for(var i=0;i<5;i++){
```

```
ctx.drawImage(cached,i*(img.width+10),80);
    }
}

function cacheShadowedImage(img,shadowcolor,blur){
    var c=document.createElement('canvas');
    var cctx=c.getContext('2d');
c.width=img.width+blur*2+2;
c.height=img.height+blur*2+2;
    cctx.shadowColor=shadowcolor;
    cctx.shadowBlur=blur;
cctx.drawImage(img,blur+1,blur+1);
    return(c);
}
```

第14.4节：通过阴影增加视觉深度

传统使用阴影是为了给二维图形赋予三维深度的错觉。

此示例展示了带阴影和不带阴影的同一个“按钮”



```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

ctx.fillStyle='skyblue';
ctx.strokeStyle='lightgray';
ctx.lineWidth=5;

// 无阴影
ctx.beginPath();
ctx.arc(60,60,30,0,Math.PI*2);
ctx.closePath();
ctx.fill();
ctx.stroke();

// 带阴影
ctx.shadowColor='black';
ctx.shadowBlur=4;
ctx.shadowOffsetY=3;
ctx.beginPath();
ctx.arc(175,60,30,0,Math.PI*2);
ctx.closePath();
ctx.fill();
ctx.stroke();
// 停止阴影效果
ctx.shadowColor='rgba(0,0,0,0)';
```

第14.5节：内阴影

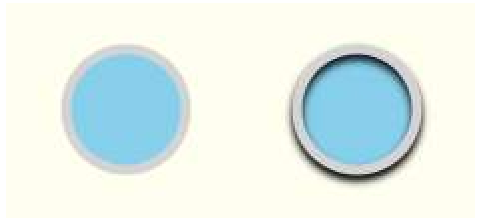
Canvas没有CSS的inner-shadow效果。

```
        ctx.drawImage(cached,i*(img.width+10),80);
    }
}

function cacheShadowedImage(img,shadowcolor,blur){
    var c=document.createElement('canvas');
    var cctx=c.getContext('2d');
    c.width=img.width+blur*2+2;
    c.height=img.height+blur*2+2;
    cctx.shadowColor=shadowcolor;
    cctx.shadowBlur=blur;
    cctx.drawImage(img,blur+1,blur+1);
    return(c);
}
```

Section 14.4: Add visual depth with shadows

The traditional use of shadowing is to give 2-dimensional drawings the illusion of 3D depth.

This example shows the same "button" with and without shadowing



```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

ctx.fillStyle='skyblue';
ctx.strokeStyle='lightgray';
ctx.lineWidth=5;

// without shadow
ctx.beginPath();
ctx.arc(60,60,30,0,Math.PI*2);
ctx.closePath();
ctx.fill();
ctx.stroke();

// with shadow
ctx.shadowColor='black';
ctx.shadowBlur=4;
ctx.shadowOffsetY=3;
ctx.beginPath();
ctx.arc(175,60,30,0,Math.PI*2);
ctx.closePath();
ctx.fill();
ctx.stroke();
// stop the shadowing
ctx.shadowColor='rgba(0,0,0,0)';
```

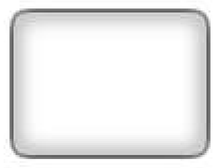
Section 14.5: Inner shadows

Canvas does not have CSS's `inner-shadow`.

- Canvas会对填充形状的外部投影阴影。
- Canvas会对描边形状的内外两侧投影阴影。

但使用合成操作很容易创建内阴影效果。

带内阴影的描边



要创建带内阴影的描边，使用destination-in合成模式，该模式使现有内容仅在被新内容覆盖的地方保留。未被新内容覆盖的现有内容将被擦除。

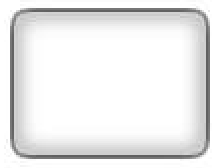
1. 给形状描边并添加阴影。阴影将从描边向外和向内延伸。我们必须
   去除外阴影——只保留所需的内阴影。
2. 将合成模式设置为destination-in，这样只有被新绘制内容覆盖的现有描边阴影才会保留。
   任何新的绘制内容。
3. 填充形状。这会使描边和内阴影保留，而外阴影被擦除。不过，
   不完全是这样！由于描边一半在填充形状内部，一半在外部，描边的外部半部分也会被擦除。解决方法是将c
   ontext.lineWidth加倍，这样加倍描边的一半仍在填充形状内部。

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

// 绘制一个不透明形状——这里我们使用一个圆角矩形
defineRoundedRect(30,30,100,75,10);

// 设置阴影
ctx.shadowColor='black';
ctx.shadowBlur=10;

// 描边带阴影的圆角矩形
ctx.lineWidth=4;
ctx.stroke();

// 设置合成模式以擦除描边外的所有内容
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// 始终清理——将合成模式恢复为默认
ctx.globalCompositeOperation='source-over';


function defineRoundedRect(x,y,width,height,radius) {
    ctx.beginPath();
ctx.moveTo(x + 半径, y);
ctx.lineTo(x + 宽度 - 半径, y);
    ctx.quadraticCurveTo(x + 宽度, y, x + 宽度, y + 半径);
    ctx.lineTo(x + 宽度, y + 高度 - 半径);
ctx.quadraticCurveTo(x + 宽度, y + 高度, x + 宽度 - 半径, y + 高度);
    ctx.lineTo(x + 半径, y + 高度);
ctx.quadraticCurveTo(x, y + 高度, x, y + 高度 - 半径);
```

- Canvas will shadow the outside of a filled shape.
- Canvas will shadow both inside and outside a stroked shape.

But it's easy to create inner-shadows using compositing.

Strokes with an inner-shadow

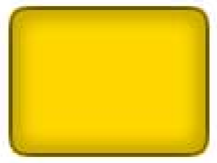


To create strokes with an inner-shadow, use `destination-in` compositing which causes existing content to remain only where existing content is overlapped by new content. Existing content that is not overlapped by new content is erased.

1. **Stroke a shape with a shadow.** The shadow will extend both outward and inward from the stroke. We must get rid of the outer-shadow -- leaving just the desired inner-shadow.
2. **Set compositing to `destination-in`** which keeps the existing stroked shadow only where it is overlapped by any new drawings.
3. **Fill the shape.** This causes the stroke and inner-shadow to remain while the outer shadow is erased. *Well, not exactly! Since a stroke is half-inside and half-outside the filled shape, the outside half of the stroke will be erased also. The fix is to double the `context.lineWidth` so half of the double-sized stroke is still inside the filled shape.*

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

// draw an opaque shape -- here we use a rounded rectangle
defineRoundedRect(30,30,100,75,10);

// set shadowing
ctx.shadowColor='black';
ctx.shadowBlur=10;

// stroke the shadowed rounded rectangle
ctx.lineWidth=4;
ctx.stroke();

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// always clean up -- set compsiting back to default
ctx.globalCompositeOperation='source-over';


function defineRoundedRect(x,y,width,height,radius) {
    ctx.beginPath();
    ctx.moveTo(x + radius, y);
    ctx.lineTo(x + width - radius, y);
    ctx.quadraticCurveTo(x + width, y, x + width, y + radius);
    ctx.lineTo(x + width, y + height - radius);
    ctx.quadraticCurveTo(x + width, y + height, x + width - radius, y + height);
    ctx.lineTo(x + radius, y + height);
    ctx.quadraticCurveTo(x, y + height, x, y + height - radius);
ctx.lineTo(x, y + 半径);
    ctx.quadraticCurveTo(x, y, x + 半径, y);
    ctx.closePath();
}
```

带内阴影的描边填充



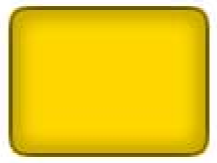
要创建带内阴影的填充，请按照上述步骤#1-3操作，但进一步使用 destination-over 合成，这会导致新内容绘制在现有内容下方。

4. 将合成模式设置为 destination-over，使填充绘制在现有内阴影的下方。
5. 通过将 context.shadowColor 设置为透明色来关闭阴影效果。
6. **用所需颜色填充形状。形状将在现有内阴影下方被填充。**

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

// 绘制一个不透明形状——这里我们使用一个圆角矩形
defineRoundedRect(30,30,100,75,10);

// 设置阴影
ctx.shadowColor='black';
ctx.shadowBlur=10;

// 描边带阴影的圆角矩形
ctx.lineWidth=4;
ctx.stroke();

// 停止阴影处理
ctx.shadowColor='rgba(0,0,0,0)';

// 设置合成模式以擦除描边外的所有内容
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// 设置合成模式以擦除描边外的所有内容
ctx.globalCompositeOperation='destination-over';
ctx.fillStyle='gold';
ctx.fill();

// 始终清理——将合成模式恢复为默认
ctx.globalCompositeOperation='source-over';

function defineRoundedRect(x,y,width,height,radius) {
    ctx.beginPath();
ctx.moveTo(x + 半径, y);
ctx.lineTo(x + 宽度 - 半径, y);
    ctx.quadraticCurveTo(x + 宽度, y, x + 宽度, y + 半径);
    ctx.lineTo(x + 宽度, y + 高度 - 半径);
ctx.quadraticCurveTo(x + 宽度, y + 高度, x + 宽度 - 半径, y + 高度);
    ctx.lineTo(x + 半径, y + 高度);
ctx.quadraticCurveTo(x, y + height, x, y + height - radius);
    ctx.lineTo(x, y + radius);
ctx.quadraticCurveTo(x, y, x + radius, y);
```

```
    ctx.lineTo(x, y + radius);
    ctx.quadraticCurveTo(x, y, x + radius, y);
    ctx.closePath();
}
```

Stroked Fills with an inner-shadow

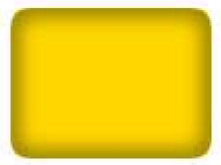


To create fills with an inner-shadow, follow steps #1-3 above but further use `destination-over` compositing which causes new content to be drawn **under existing content**.

4. **Set compositing to `destination-over`** which causes the fill to be drawn **under** the existing inner-shadow.
5. **Turn off shadowing** by setting `context.shadowColor` to a transparent color.
6. **Fill the shape** with the desired color. The shape will be filled underneath the existing inner-shadow.

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

// draw an opaque shape -- here we use a rounded rectangle
defineRoundedRect(30,30,100,75,10);

// set shadowing
ctx.shadowColor='black';
ctx.shadowBlur=10;

// stroke the shadowed rounded rectangle
ctx.lineWidth=4;
ctx.stroke();

// stop shadowing
ctx.shadowColor='rgba(0,0,0,0)';

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-over';
ctx.fillStyle='gold';
ctx.fill();

// always clean up -- set compsiting back to default
ctx.globalCompositeOperation='source-over';

function defineRoundedRect(x,y,width,height,radius) {
    ctx.beginPath();
    ctx.moveTo(x + radius, y);
    ctx.lineTo(x + width - radius, y);
    ctx.quadraticCurveTo(x + width, y, x + width, y + radius);
    ctx.lineTo(x + width, y + height - radius);
    ctx.quadraticCurveTo(x + width, y + height, x + width - radius, y + height);
    ctx.lineTo(x + radius, y + height);
    ctx.quadraticCurveTo(x, y + height, x, y + height - radius);
    ctx.lineTo(x, y + radius);
    ctx.quadraticCurveTo(x, y, x + radius, y);
ctx.closePath();
}
```

无描边的填充带内阴影

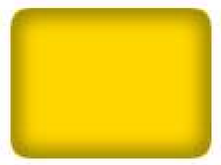


要绘制带内阴影但无描边的填充形状，可以将描边绘制在画布外，并使用 shadowOffsetX 用于将阴影推回画布上。

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

// 定义一个不透明形状——这里我们使用圆角矩形
defineRoundedRect(30-500,30,100,75,10);

// 设置阴影
ctx.shadowColor='black';
ctx.shadowBlur=10;
ctx.shadowOffsetX=500;

// 描边带阴影的圆角矩形
ctx.lineWidth=4;
ctx.stroke();

// 停止阴影处理
ctx.shadowColor='rgba(0,0,0,0)';

// 重新定义一个不透明形状——这里我们使用圆角矩形
defineRoundedRect(30,30,100,75,10);

// 设置合成模式以擦除描边外的所有内容
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// 设置合成模式以擦除描边外的所有内容
ctx.globalCompositeOperation='destination-over';
ctx.fillStyle='gold';
ctx.fill();

// 始终清理——将合成模式恢复为默认
ctx.globalCompositeOperation='source-over';

function defineRoundedRect(x,y,width,height,radius) {
    ctx.beginPath();
ctx.moveTo(x + 半径, y);
ctx.lineTo(x + 宽度 - 半径, y);
    ctx.quadraticCurveTo(x + 宽度, y, x + 宽度, y + 半径);
    ctx.lineTo(x + 宽度, y + 高度 - 半径);
ctx.quadraticCurveTo(x + 宽度, y + 高度, x + 宽度 - 半径, y + 高度);
    ctx.lineTo(x + 半径, y + 高度);
ctx.quadraticCurveTo(x, y + height, x, y + height - radius);
    ctx.lineTo(x, y + radius);
ctx.quadraticCurveTo(x, y, x + radius, y);
    ctx.closePath();
```

```
    ctx.closePath();
}
```

Non-stroked Fills with an inner-shadow



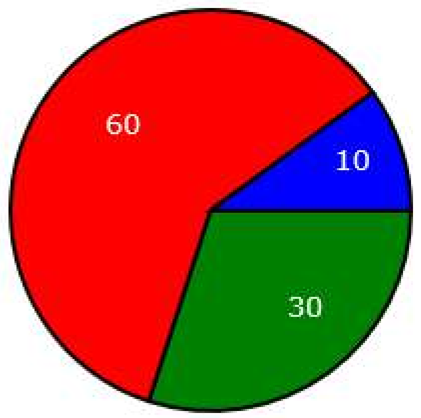
To draw a filled shape with an inner-shadow, but with no stroke, you can draw the stroke off-canvas and use `shadowOffsetX` to push the shadow back onto the canvas.

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

// define an opaque shape -- here we use a rounded rectangle
defineRoundedRect(30-500,30,100,75,10);

// set shadowing
ctx.shadowColor='black';
ctx.shadowBlur=10;
ctx.shadowOffsetX=500;

// stroke the shadowed rounded rectangle
ctx.lineWidth=4;
ctx.stroke();

// stop shadowing
ctx.shadowColor='rgba(0,0,0,0)';

// redefine an opaque shape -- here we use a rounded rectangle
defineRoundedRect(30,30,100,75,10);

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-over';
ctx.fillStyle='gold';
ctx.fill();

// always clean up -- set compsiting back to default
ctx.globalCompositeOperation='source-over';

function defineRoundedRect(x,y,width,height,radius) {
    ctx.beginPath();
    ctx.moveTo(x + radius, y);
    ctx.lineTo(x + width - radius, y);
    ctx.quadraticCurveTo(x + width, y, x + width, y + radius);
    ctx.lineTo(x + width, y + height - radius);
    ctx.quadraticCurveTo(x + width, y + height, x + width - radius, y + height);
    ctx.lineTo(x + radius, y + height);
    ctx.quadraticCurveTo(x, y + height, x, y + height - radius);
    ctx.lineTo(x, y + radius);
    ctx.quadraticCurveTo(x, y, x + radius, y);
    ctx.closePath();
}
```

```
}
```

第15章：图表与示意图

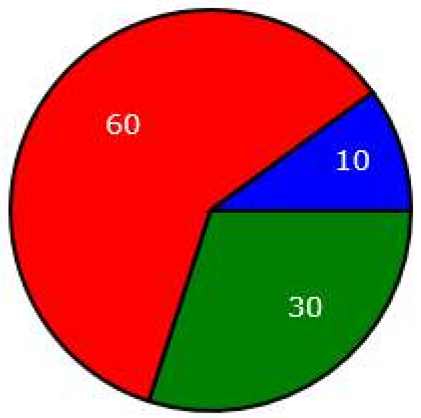
第15.1节：带演示的饼图



```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    var canvas = document.getElementById("canvas");
    var ctx = canvas.getContext("2d");
    ctx.lineWidth = 2;
    ctx.font = '14px verdana';

    var PI2 = Math.PI * 2;
    var myColor = ["Green", "Red", "Blue"];
    var myData = [30, 60, 10];
    var cx = 150;
    var cy = 150;
    var radius = 100;

pieChart(myData, myColor);

    function pieChart(data, colors) {
      var total = 0;
      for (var i = 0; i < data.length; i++) {
        total += data[i];
      }

      var sweeps = []
      for (var i = 0; i < data.length; i++) {
        sweeps.push(data[i] / total * PI2);
      }

      var accumAngle = 0;
      for (var i = 0; i < sweeps.length; i++) {
        drawWedge(accumAngle, accumAngle + sweeps[i], colors[i], data[i]);
        accumAngle += sweeps[i];
      }
```

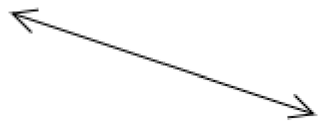
Chapter 15: Charts & Diagrams

Section 15.1: Pie Chart with Demo



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    var canvas = document.getElementById("canvas");
    var ctx = canvas.getContext("2d");
    ctx.lineWidth = 2;
    ctx.font = '14px verdana';

    var PI2 = Math.PI * 2;
    var myColor = ["Green", "Red", "Blue"];
    var myData = [30, 60, 10];
    var cx = 150;
    var cy = 150;
    var radius = 100;

    pieChart(myData, myColor);

    function pieChart(data, colors) {
      var total = 0;
      for (var i = 0; i < data.length; i++) {
        total += data[i];
      }

      var sweeps = []
      for (var i = 0; i < data.length; i++) {
        sweeps.push(data[i] / total * PI2);
      }

      var accumAngle = 0;
      for (var i = 0; i < sweeps.length; i++) {
        drawWedge(accumAngle, accumAngle + sweeps[i], colors[i], data[i]);
        accumAngle += sweeps[i];
      }
    }

    function drawWedge(startAngle, endAngle, fill, label) {
      // 绘制扇形
ctx.beginPath();
ctx.moveTo(cx, cy);
ctx.arc(cx, cy, radius, startAngle, endAngle, false);
      ctx.closePath();
ctx.fillStyle = fill;
      ctx.strokeStyle = 'black';
      ctx.fill();
ctx.stroke();

      // 绘制标签
      var midAngle = startAngle + (endAngle - startAngle) / 2;
      var labelRadius = radius * .65;
      var x = cx + (labelRadius) * Math.cos(midAngle);
      var y = cy + (labelRadius) * Math.sin(midAngle);
      ctx.fillStyle = 'white';
      ctx.fillText(label, x, y);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

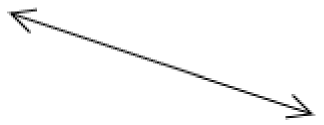
第15.2节：带箭头的线



```
// 用法：
drawLineWithArrows(50,50,150,50,5,8,true,true);

// x0,y0：线条的起点
// x1,y1：线条的终点
// width：箭头垂直于线条方向的延伸距离
// height：箭头从终点向后延伸的距离
// arrowStart：true/false，指示是否在起点绘制箭头
// arrowEnd：true/false，指示是否在终点绘制箭头

function drawLineWithArrows(x0,y0,x1,y1,aWidth,aLength,arrowStart,arrowEnd){
    var dx=x1-x0;
    var dy=y1-y0;
    var angle=Math.atan2(dy,dx);
    var length=Math.sqrt(dx*dx+dy*dy);
    //
ctx.translate(x0,y0);
    ctx.rotate(angle);
    ctx.beginPath();
    ctx.moveTo(0,0);
    ctx.lineTo(length,0);
```

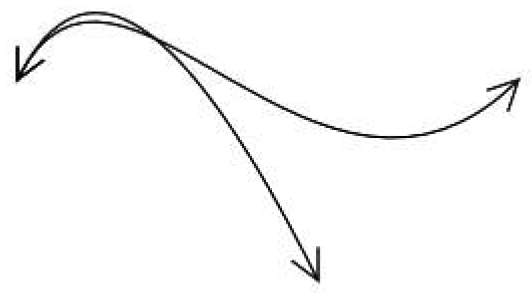
```
    }

    function drawWedge(startAngle, endAngle, fill, label) {
      // draw the wedge
      ctx.beginPath();
      ctx.moveTo(cx, cy);
      ctx.arc(cx, cy, radius, startAngle, endAngle, false);
      ctx.closePath();
      ctx.fillStyle = fill;
      ctx.strokeStyle = 'black';
      ctx.fill();
      ctx.stroke();

      // draw the label
      var midAngle = startAngle + (endAngle - startAngle) / 2;
      var labelRadius = radius * .65;
      var x = cx + (labelRadius) * Math.cos(midAngle);
      var y = cy + (labelRadius) * Math.sin(midAngle);
      ctx.fillStyle = 'white';
      ctx.fillText(label, x, y);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

Section 15.2: Line with arrowheads



```
// Usage:
drawLineWithArrows(50,50,150,50,5,8,true,true);

// x0,y0: the line's starting point
// x1,y1: the line's ending point
// width: the distance the arrowhead perpendicularly extends away from the line
// height: the distance the arrowhead extends backward from the endpoint
// arrowStart: true/false directing to draw arrowhead at the line's starting point
// arrowEnd: true/false directing to draw arrowhead at the line's ending point

function drawLineWithArrows(x0,y0,x1,y1,aWidth,aLength,arrowStart,arrowEnd){
    var dx=x1-x0;
    var dy=y1-y0;
    var angle=Math.atan2(dy,dx);
    var length=Math.sqrt(dx*dx+dy*dy);
    //
    ctx.translate(x0,y0);
    ctx.rotate(angle);
    ctx.beginPath();
    ctx.moveTo(0,0);
    ctx.lineTo(length,0);
    if(arrowStart){
ctx.moveTo(aLength,-aWidth);
        ctx.lineTo(0,0);
ctx.lineTo(aLength,aWidth);
    }
    if(arrowEnd){
ctx.moveTo(length-aLength,-aWidth);
        ctx.lineTo(length,0);
ctx.lineTo(length-aLength,aWidth);
    }
    //
ctx.stroke();
ctx.setTransform(1,0,0,1,0,0);
}
```

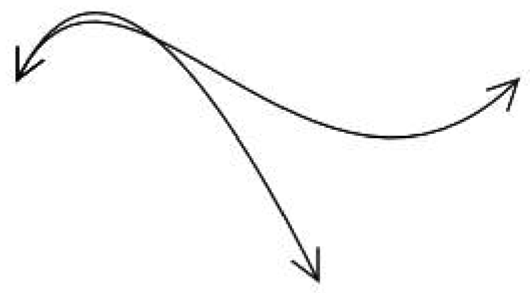
第15.3节：带箭头的三次和二次贝塞尔曲线



```
// 用法：
var  p0={x:50,y:100};
var  p1={x:100,y:0};
var  p2={x:200,y:200};
var  p3={x:300,y:100};

cubicCurveArrowHeads(p0, p1, p2, p3, 15, true, true);

quadraticCurveArrowHeads(p0, p1, p2, 15, true, true);

// 或者使用默认值，两端均为 true，带箭头
cubicCurveArrowHeads(p0, p1, p2, p3, 15);

quadraticCurveArrowHeads(p0, p1, p2, 15);



// 绘制三次和二次贝塞尔曲线
function bezWithArrowheads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow) {
    var x, y, norm, ex, ey;
    function pointsToNormalisedVec(p,pp){
        var len;
norm.y = pp.x - p.x;
norm.x = -(pp.y - p.y);
        len = Math.sqrt(norm.x * norm.x + norm.y * norm.y);
        norm.x /= len;
norm.y /= len;
        return norm;
    }
```

```
    if(arrowStart){
        ctx.moveTo(aLength,-aWidth);
        ctx.lineTo(0,0);
        ctx.lineTo(aLength,aWidth);
    }
    if(arrowEnd){
        ctx.moveTo(length-aLength,-aWidth);
        ctx.lineTo(length,0);
        ctx.lineTo(length-aLength,aWidth);
    }
    //
    ctx.stroke();
    ctx.setTransform(1,0,0,1,0,0);
}
```

Section 15.3: Cubic & Quadratic Bezier curve with arrowheads



```
// Usage:
var p0={x:50,y:100};
var p1={x:100,y:0};
var p2={x:200,y:200};
var p3={x:300,y:100};

cubicCurveArrowHeads(p0, p1, p2, p3, 15, true, true);

quadraticCurveArrowHeads(p0, p1, p2, 15, true, true);

// or use defaults true for both ends with arrow heads
cubicCurveArrowHeads(p0, p1, p2, p3, 15);

quadraticCurveArrowHeads(p0, p1, p2, 15);



// draws both cubic and quadratic bezier
function bezWithArrowheads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow) {
    var x, y, norm, ex, ey;
    function pointsToNormalisedVec(p,pp){
        var len;
        norm.y = pp.x - p.x;
        norm.x = -(pp.y - p.y);
        len = Math.sqrt(norm.x * norm.x + norm.y * norm.y);
        norm.x /= len;
        norm.y /= len;
        return norm;
    }
    var arrowWidth = arrowLength / 2;
    norm = {};
    // 如果未包含参数，默认两个箭头均为真
hasStartArrow = hasStartArrow === undefined || hasStartArrow === null ? true : hasStartArrow;

ctx.beginPath();
    ctx.moveTo(p0.x, p0.y);
    if (p3 === undefined) {
ctx.quadraticCurveTo(p1.x, p1.y, p2.x, p2.y);
        ex = p2.x;  // 获取终点
ey = p2.y;
norm = pointsToNormalisedVec(p1,p2);
    } else {
ctx.bezierCurveTo(p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)
        ex = p3.x; // 获取终点
ey = p3.y;
norm = pointsToNormalisedVec(p2,p3);
    }
    if (hasEndArrow) {
x = 箭头宽度 * 归一化.x + 箭头长度 * -归一化.y;
        y = 箭头宽度 * 归一化.y + 箭头长度 * 归一化.x;
        ctx.moveTo(ex + x, ey + y);
ctx.lineTo(ex, ey);
x = arrowWidth * -norm.x + arrowLength * -norm.y;
        y = arrowWidth * -norm.y + arrowLength * norm.x;
        ctx.lineTo(ex + x, ey + y);
    }
    if (hasStartArrow) {
norm = pointsToNormalisedVec(p0,p1);
        x = arrowWidth * norm.x - arrowLength * -norm.y;
        y = arrowWidth * norm.y - arrowLength * norm.x;
        ctx.moveTo(p0.x + x, p0.y + y);
ctx.lineTo(p0.x, p0.y);
        x = arrowWidth * -norm.x - arrowLength * -norm.y;
        y = arrowWidth * -norm.y - arrowLength * norm.x;
        ctx.lineTo(p0.x + x, p0.y + y);
    }

ctx.stroke();
}

function cubicCurveArrowHeads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow) {
    bezWithArrowheads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow);
}
function quadraticCurveArrowHeads(p0, p1, p2, arrowLength, hasStartArrow, hasEndArrow) {
    bezWithArrowheads(p0, p1, p2, undefined, arrowLength, hasStartArrow, hasEndArrow);
}
```

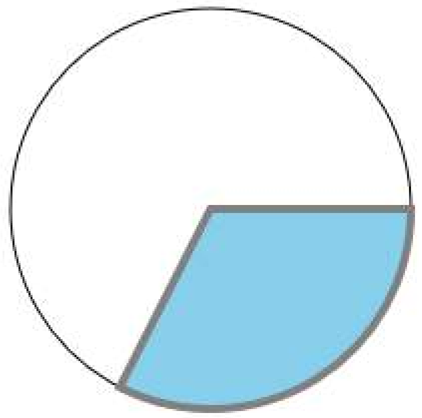
第15.4节：楔形

代码仅绘制楔形……此处绘制的圆仅用于透视参考。

```
    var arrowWidth = arrowLength / 2;
    norm = {};
    // defaults to true for both arrows if arguments not included
    hasStartArrow = hasStartArrow === undefined || hasStartArrow === null ? true : hasStartArrow;
    hasEndArrow = hasEndArrow === undefined || hasEndArrow === null ? true : hasEndArrow;
    ctx.beginPath();
    ctx.moveTo(p0.x, p0.y);
    if (p3 === undefined) {
        ctx.quadraticCurveTo(p1.x, p1.y, p2.x, p2.y);
        ex = p2.x;  // get end point
        ey = p2.y;
        norm = pointsToNormalisedVec(p1,p2);
    } else {
        ctx.bezierCurveTo(p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)
        ex = p3.x; // get end point
        ey = p3.y;
        norm = pointsToNormalisedVec(p2,p3);
    }
    if (hasEndArrow) {
        x = arrowWidth * norm.x + arrowLength * -norm.y;
        y = arrowWidth * norm.y + arrowLength * norm.x;
        ctx.moveTo(ex + x, ey + y);
        ctx.lineTo(ex, ey);
        x = arrowWidth * -norm.x + arrowLength * -norm.y;
        y = arrowWidth * -norm.y + arrowLength * norm.x;
        ctx.lineTo(ex + x, ey + y);
    }
    if (hasStartArrow) {
        norm = pointsToNormalisedVec(p0,p1);
        x = arrowWidth * norm.x - arrowLength * -norm.y;
        y = arrowWidth * norm.y - arrowLength * norm.x;
        ctx.moveTo(p0.x + x, p0.y + y);
        ctx.lineTo(p0.x, p0.y);
        x = arrowWidth * -norm.x - arrowLength * -norm.y;
        y = arrowWidth * -norm.y - arrowLength * norm.x;
        ctx.lineTo(p0.x + x, p0.y + y);
    }

    ctx.stroke();
}

function cubicCurveArrowHeads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow) {
    bezWithArrowheads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow);
}
function quadraticCurveArrowHeads(p0, p1, p2, arrowLength, hasStartArrow, hasEndArrow) {
    bezWithArrowheads(p0, p1, p2, undefined, arrowLength, hasStartArrow, hasEndArrow);
}
```

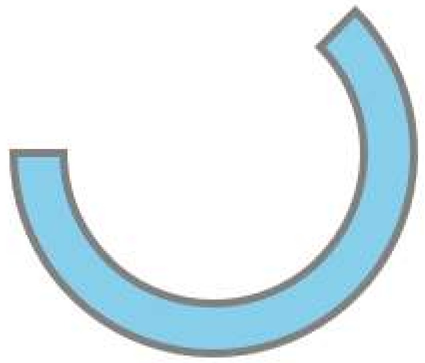
Section 15.4: Wedge

The code draws only the wedge ... circle drawn here for perspective only.

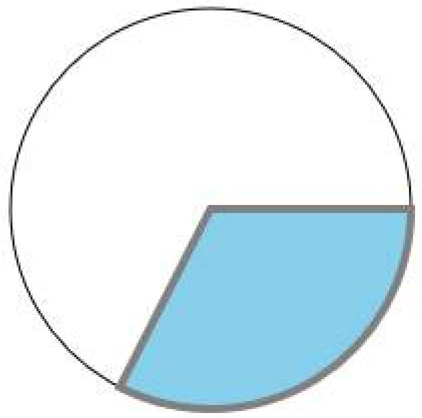


```
// 用法
var wedge={
cx:150, cy:150,
    radius:100,
    startAngle:0,
    endAngle:Math.PI*.65
}

drawWedge(wedge,'skyblue','gray',4);

function drawWedge(w,fill,stroke,strokewidth){
    ctx.beginPath();
ctx.moveTo(w.cx, w.cy);
ctx.arc(w.cx, w.cy, w.radius, w.startAngle, w.endAngle);
    ctx.closePath();
ctx.fillStyle=fill;
    ctx.fill();
ctx.strokeStyle=stroke;
    ctx.lineWidth=strokewidth;
    ctx.stroke();
}
```

第15.5节：同时填充和描边的弧形

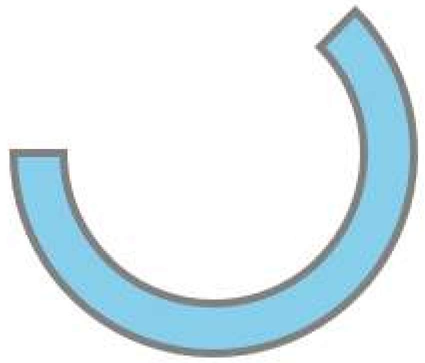


```
// 用法：
var arc={
cx:150, cy:150,
内半径:75, 外半径:100,
   起始角度:-Math.PI/4, 结束角度:Math.PI
}
```



```
// Usage
var wedge={
    cx:150, cy:150,
    radius:100,
    startAngle:0,
    endAngle:Math.PI*.65
}

drawWedge(wedge,'skyblue','gray',4);

function drawWedge(w,fill,stroke,strokewidth){
    ctx.beginPath();
    ctx.moveTo(w.cx, w.cy);
    ctx.arc(w.cx, w.cy, w.radius, w.startAngle, w.endAngle);
    ctx.closePath();
    ctx.fillStyle=fill;
    ctx.fill();
    ctx.strokeStyle=stroke;
    ctx.lineWidth=strokewidth;
    ctx.stroke();
}
```

Section 15.5: Arc with both fill and stroke



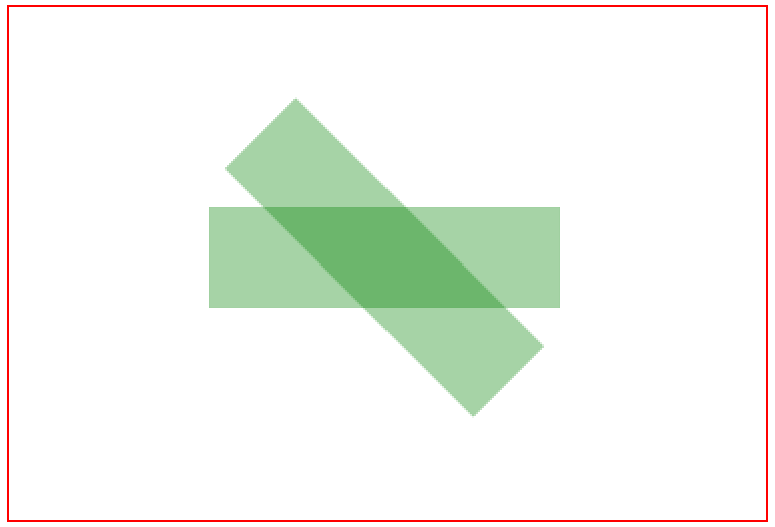
```
// Usage:
var arc={
    cx:150, cy:150,
    innerRadius:75, outerRadius:100,
    startAngle:-Math.PI/4, endAngle:Math.PI
}
```

```
drawArc(arc,'skyblue','gray',4);

function drawArc(a,fill,stroke,strokewidth){
    ctx.beginPath();
ctx.arc(a.cx,a.cy,a.innerRadius,a.startAngle,a.endAngle);
    ctx.arc(a.cx,a.cy,a.outerRadius,a.endAngle,a.startAngle,true);
    ctx.closePath();
ctx.fillStyle=fill;
    ctx.strokeStyle=stroke;
    ctx.lineWidth=strokewidth
    ctx.fill();
ctx.stroke();
}
```

```
drawArc(arc,'skyblue','gray',4);

function drawArc(a,fill,stroke,strokewidth){
    ctx.beginPath();
    ctx.arc(a.cx,a.cy,a.innerRadius,a.startAngle,a.endAngle);
    ctx.arc(a.cx,a.cy,a.outerRadius,a.endAngle,a.startAngle,true);
    ctx.closePath();
    ctx.fillStyle=fill;
    ctx.strokeStyle=stroke;
    ctx.lineWidth=strokewidth
    ctx.fill();
    ctx.stroke();
}
```

第16章：变换

第16.1节：围绕中心点旋转图像或路径



以下第1-5步允许将任何图像或路径形状移动到画布上的任意位置，并旋转到任意角度，而不改变图像/路径形状的原始点坐标。

1. 将画布的[0,0]原点移动到形状的中心点

```
context.translate( shapeCenterX, shapeCenterY );
```

2. 将画布旋转到所需角度（以弧度为单位）

```
context.rotate( radianAngle );
```

3. 将画布原点移回左上角

```
context.translate( -shapeCenterX, -shapeCenterY );
```

4. 使用图像或路径形状的原始坐标绘制

```
context.fillRect( shapeX, shapeY, shapeWidth, shapeHeight );
```

5. 始终清理！将变换状态恢复到步骤#1之前的状态

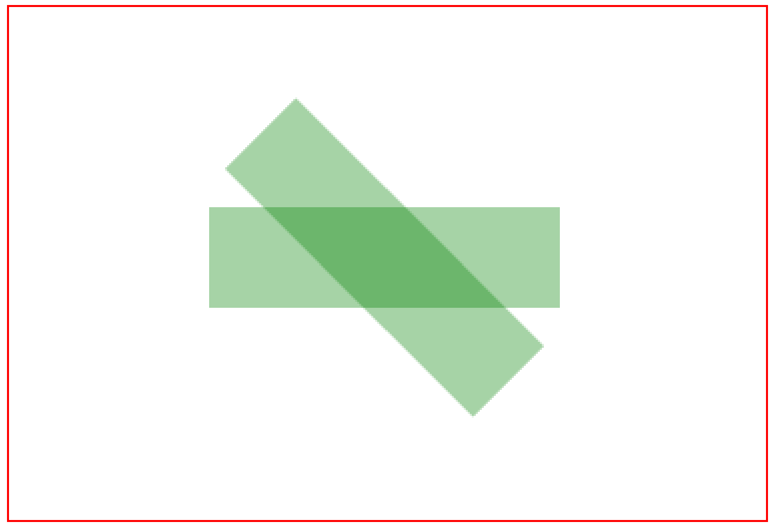
- 步骤#5，选项#1： 按相反顺序撤销每个变换

```
    // 撤销 #3
context.translate( shapeCenterX, shapeCenterY );
    // 撤销 #2
context.rotate( -radianAngle );
    // 撤销 #1
context.translate( -shapeCenterX, shapeCenterY );
```

- *步骤#5，选项#2：* 如果画布在开始步骤#1之前处于未变换状态（默认状态），你可以通过重置所有变换为默认状态来撤销步骤#1-3的效果

Chapter 16: Transformations

Section 16.1: Rotate an Image or Path around it's centerpoint



Steps#1-5 below allow any image or path-shape to be both moved anywhere on the canvas and rotated to any angle without changing any of the image/path-shape's original point coordinates.

1. Move the canvas [0,0] origin to the shape's center point

```
context.translate( shapeCenterX, shapeCenterY );
```

2. Rotate the canvas by the desired angle (in radians)

```
context.rotate( radianAngle );
```

3. Move the canvas origin back to the top-left corner

```
 context.translate( -shapeCenterX, -shapeCenterY );
```

4. Draw the image or path-shape using it's original coordinates.

```
 context.fillRect( shapeX, shapeY, shapeWidth, shapeHeight );
```

5. Always clean up! Set the transformation state back to where it was before #1

- *Step#5, Option#1:* Undo every transformation in the reverse order

```
    // undo #3
    context.translate( shapeCenterX, shapeCenterY );
    // undo #2
    context.rotate( -radianAngle );
    // undo #1
    context.translate( -shapeCenterX, shapeCenterY );
```

- *Step#5, Option#2:* If the canvas was in an untransformed state (the default) before beginning step#1, you can undo the effects of steps#1-3 by resetting all transformations to their default state

```
    // 将变换设置为默认状态（==未应用任何变换）
context.setTransform(1,0,0,1,0,0)
```

示例代码演示：

```
// 画布引用及画布样式设置
var canvas=document.createElement("canvas");
canvas.style.border='1px solid red';
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;
var ctx=canvas.getContext("2d");
ctx.fillStyle='green';
ctx.globalAlpha=0.35;

// 定义一个要旋转的矩形
var rect={ x:100, y:100, width:175, height:50 };

// 绘制未旋转的矩形
ctx.fillRect( rect.x, rect.y, rect.width, rect.height );

// 绘制旋转45度（即PI/4弧度）的矩形
ctx.translate( rect.x+rect.width/2, rect.y+rect.height/2 );
ctx.rotate( Math.PI/4 );
ctx.translate( -rect.x-rect.width/2, -rect.y-rect.height/2 );
ctx.fillRect( rect.x, rect.y, rect.width, rect.height );
```

第16.2节：快速绘制多个平移、缩放和旋转的图像

有许多情况下你需要绘制一个旋转、缩放和平移的图像。旋转应围绕图像的中心进行。这是在二维画布上实现这一效果的最快方法。这些函数非常适合二维游戏，期望每秒60帧渲染几百甚至超过1000张图像。（取决于硬件）

```
// 假设画布上下文为 ctx 并且在作用域内
function drawImageRST(image, x, y, scale, rotation){
    ctx.setTransform(scale, 0, 0, scale, x, y); // 设置缩放和平移
    ctx.rotate(rotation);                       // 添加旋转
ctx.drawImage(image, -image.width / 2, -image.height / 2); // 绘制图像，偏移其宽度和高度的一半

}
```

一个变体还可以包含 alpha 值，这对于粒子系统非常有用。

```
function drawImageRST_Alpha(image, x, y, scale, rotation, alpha){
    ctx.setTransform(scale, 0, 0, scale, x, y); // 设置缩放和平移
    ctx.rotate(rotation);                       // 添加旋转
ctx.globalAlpha = alpha;
ctx.drawImage(image, -image.width / 2, -image.height / 2); // 绘制图像，偏移其宽度和高度的一半

}
```

需要注意的是，这两个函数都会使画布上下文处于一个随机状态。虽然函数本身不会受到影响，但其他渲染可能会受到影响。当你完成图像渲染后，可能需要恢复默认的变换

```
    // set transformation to the default state (==no transformation applied)
    context.setTransform(1,0,0,1,0,0)
```

Example code demo:

```
// canvas references & canvas styling
var canvas=document.createElement("canvas");
canvas.style.border='1px solid red';
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;
var ctx=canvas.getContext("2d");
ctx.fillStyle='green';
ctx.globalAlpha=0.35;

// define a rectangle to rotate
var rect={ x:100, y:100, width:175, height:50 };

// draw the rectangle unrotated
ctx.fillRect( rect.x, rect.y, rect.width, rect.height );

// draw the rectangle rotated by 45 degrees (==PI/4 radians)
ctx.translate( rect.x+rect.width/2, rect.y+rect.height/2 );
ctx.rotate( Math.PI/4 );
ctx.translate( -rect.x-rect.width/2, -rect.y-rect.height/2 );
ctx.fillRect( rect.x, rect.y, rect.width, rect.height );
```

Section 16.2: Drawing many translated, scaled, and rotated images quickly

There are many situation where you want to draw an image that is rotated, scaled, and translated. The rotation should occur around the center of the image. This is the quickest way to do so on the 2D canvas. These functions a well suited to 2D games where the expectation is to render a few hundred even up to a 1000+ images every 60th of a second. (dependent on the hardware)

```
// assumes that the canvas context is in ctx and in scope
function drawImageRST(image, x, y, scale, rotation){
    ctx.setTransform(scale, 0, 0, scale, x, y); // set the scale and translation
    ctx.rotate(rotation);                       // add the rotation
    ctx.drawImage(image, -image.width / 2, -image.height / 2); // draw the image offset by half its
width and height
}
```

A variant can also include the alpha value which is useful for particle systems.

```
function drawImageRST_Alpha(image, x, y, scale, rotation, alpha){
    ctx.setTransform(scale, 0, 0, scale, x, y); // set the scale and translation
    ctx.rotate(rotation);                       // add the rotation
    ctx.globalAlpha = alpha;
    ctx.drawImage(image, -image.width / 2, -image.height / 2); // draw the image offset by half its
width and height
}
```

It is important to note that both functions leave the canvas context in a random state. Though the functions will not be affected other rendering my be. When you are done rendering images you may need to restore the default transform

```
ctx.setTransform(1, 0, 0, 1, 0, 0); // 将上下文变换重置为默认值
```

如果你先使用带 alpha 的版本（第二个示例），然后再使用标准版本，你必须确保全局 alpha 状态被恢复

```
ctx.globalAlpha = 1;
```

使用上述函数渲染一些粒子和几张图片的示例

```
// 假设 particles 包含一个粒子数组
for(var i = 0; i < particles.length; i++){
    var p = particles[i];
drawImageRST_Alpha(p.image, p.x, p.y, p.scale, p.rot, p.alpha);
    // 循环中无需重置 alpha
}
// 你需要重置 alpha，因为它可以是任意值
ctx.globalAlpha = 1;

drawImageRST(myImage, 100, 100, 1, 0.5);  // 在 100,100 处绘制一张图片
// 无需重置变换
drawImageRST(myImage, 200, 200, 1, -0.5); // 在 200,200 处绘制一张图片
ctx.setTransform(1,0,0,1,0,0);            // 重置变换
```

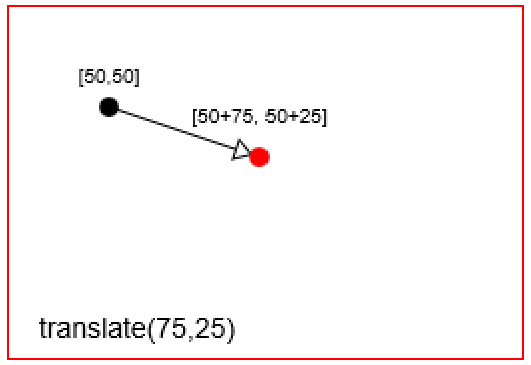
第16.3节：变换简介

变换通过移动、旋转和缩放来改变给定点的起始位置。

- 翻译： 将一个点移动 distanceX 和 distanceY 的距离。
- 旋转：绕其旋转点以弧度角旋转一个点。Html Canvas中默认的旋转点是画布的左上角原点[x=0,y=0]。但你可以通过平移来重新定位旋转点。
- 缩放：从其缩放点按缩放因子X和缩放因子Y缩放一个点的位置。Html Canvas中默认的缩放点是画布的左上角原点[x=0,y=0]。但你可以通过平移来重新定位缩放点。

你也可以通过直接设置画布的变换矩阵context.transform来实现较少见的变换，比如剪切（倾斜）。

使用context.translate(75,25)平移（即移动）一个点



使用context.rotate(Math.PI/8)旋转一个点

```
ctx.setTransform(1, 0, 0, 1, 0, 0); // set the context transform back to the default
```

If you use the alpha version (second example) and then the standard version you will have to ensure that the global alpha state is restored

```
ctx.globalAlpha = 1;
```

An example of using the above functions to render some particles and the a few images

```
// assume particles to contain an array of particles
for(var i = 0; i < particles.length; i++){
    var p = particles[i];
    drawImageRST_Alpha(p.image, p.x, p.y, p.scale, p.rot, p.alpha);
    // no need to rest the alpha in the loop
}
// you need to reset the alpha as it can be any value
ctx.globalAlpha = 1;

drawImageRST(myImage, 100, 100, 1, 0.5);  // draw an image at 100,100
// no need to reset the transform
drawImageRST(myImage, 200, 200, 1, -0.5); // draw an image at 200,200
ctx.setTransform(1,0,0,1,0,0);            // reset the transform
```

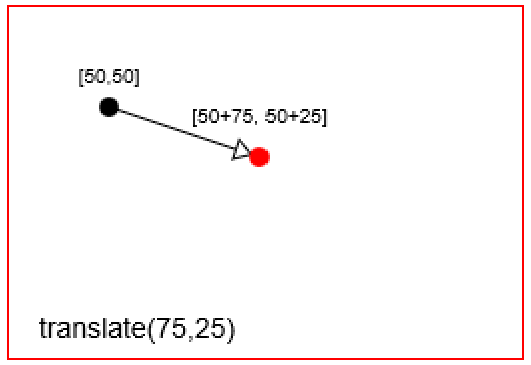
Section 16.3: Introduction to Transformations

Transformations alter a given point's starting position by moving, rotating & scaling that point.

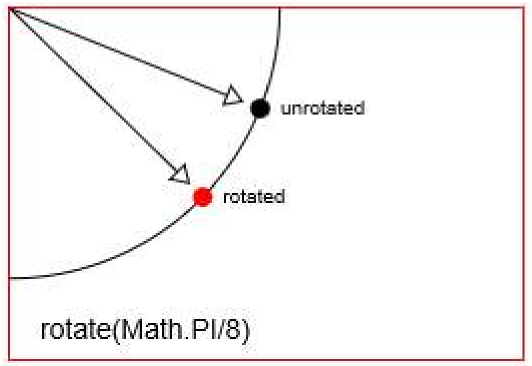
- **Translation:** Moves a point by a `distanceX` and `distanceY`.
- **Rotation:** Rotates a point by a `radian angle` around it's rotation point. The default rotation point in Html Canvas is the top-left origin [x=0,y=0] of the Canvas. But you can reposition the rotation point using translations.
- **Scaling:** Scales a point's position by a `scalingFactorX` and `scalingFactorY` from it's scaling point. The default scaling point in Html Canvas is the top-left origin [x=0,y=0] of the Canvas. But you can reposition the scaling point using translations.

You can also do less common transformations, like shearing (skewing), by directly setting the transformation matrix of the canvas using `context.transform`.

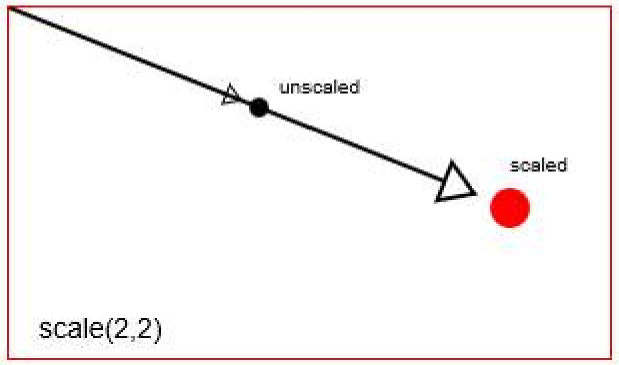
Translate (==move) a point with `context.translate(75,25)`



Rotate a point with `context.rotate(Math.PI/8)`



使用context.scale(2,2)缩放一个点



画布实际上是通过改变整个画布的坐标系统来实现变换的。

- context.translate会将画布的[0,0]原点从左上角移动到一个新位置。
- context.rotate会围绕原点旋转整个画布坐标系统。
- context.scale会围绕原点缩放整个画布坐标系统。可以把它理解为放大画布上每个x,y的大小：每个x*=scaleX，每个y*=scaleY。

画布变换是持久的。所有新的绘图都会继续被变换，直到你将画布的变换重置回默认状态（即完全未变换）。你可以通过以下方式重置回默认状态：

```
// 将上下文变换重置为默认（未变换）状态
context.setTransform(1,0,0,1,0,0);
```

第16.4节：用于跟踪平移、旋转和缩放形状的变换矩阵

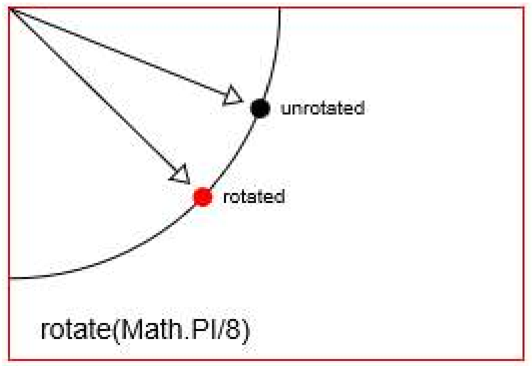
Canvas允许你使用context.translate、context.rotate和context.scale来绘制你所需位置和大小的形状。

Canvas本身使用变换矩阵来高效地跟踪变换。

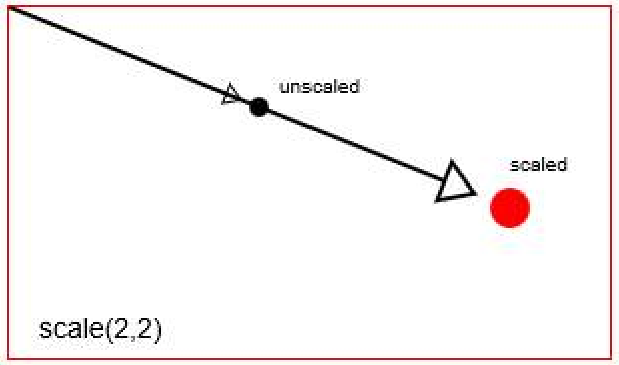
- 你可以使用context.transform来更改Canvas的矩阵
- 你可以使用单独的`translate`、rotate和scale命令来更改Canvas的矩阵
- 你可以使用`context.setTransform`完全覆盖Canvas的矩阵，
- *但你无法读取Canvas的内部变换矩阵——它是只写的。*

为什么要使用变换矩阵？

变换矩阵允许你将多个单独的平移、旋转和缩放合并成一个易于重新应用的矩阵。



Scale a point with `context.scale(2,2)`



Canvas actually achieves transformations by altering the canvas' entire coordinate system.

- `context.translate` will move the canvas [0,0] origin from the top left corner to a new location.
- `context.rotate` will rotate the entire canvas coordinate system around the origin.
- `context.scale` will scale the entire canvas coordinate system around the origin. Think of this as increasing the size of every x,y on the canvas: `every x*=scaleX` and `every y*=scaleY`.

Canvas transformations are persistent. All New drawings will continue to be transformed until you reset the canvas' transformation back to it's default state (==totally untransformed). You can reset back to default with:

```
// reset context transformations to the default (untransformed) state
context.setTransform(1,0,0,1,0,0);
```

Section 16.4: A Transformation Matrix to track translated, rotated & scaled shape(s)

Canvas allows you to `context.translate`, `context.rotate` and `context.scale` in order to draw your shape in the position & size you require.

Canvas itself uses a transformation matrix to efficiently track transformations.

- You can change Canvas's matrix with `context.transform`
- You can change Canvas's matrix with individual `translate`, `rotate` & `scale` commands
- You can completely overwrite Canvas's matrix with `context.setTransform`,
- *But you can't read Canvas's internal transformation matrix -- it's write-only.*

Why use a transformation matrix?

A transformation matrix allows you to aggregate many individual translations, rotations & scalings into a single, easily reapplied matrix.

在复杂动画中，你可能会对一个形状应用数十个（甚至数百个）变换。通过使用
变换矩阵，你可以用一行代码（几乎）瞬间重新应用这些数十个变换。

一些示例用法：

- **测试鼠标是否在你已平移、旋转和缩放的形状内**

  有一个内置的context.isPointInPath方法，用于测试一个点（例如鼠标）是否在路径形状内，但与使用矩阵测试相比，这个内置测试非常慢。

  高效测试鼠标是否在形状内，涉及将浏览器报告的鼠标位置以与形状相同的方式进行变换。然后你可以像形状未被变换一样进行碰撞检测。

- **重绘一个经过大量平移、旋转和缩放的形状。**

  你可以不用多次调用.translate、.rotate、.scale来逐个重新应用变换，而是用一行代码应用所有聚合的变换。

- **测试经过平移、旋转和缩放的形状的碰撞**

  你可以使用几何和三角函数计算构成变换后形状的点，但使用变换矩阵计算这些点更快。

一个变换矩阵“类”

此代码对应原生的context.translate、context.rotate、context.scale变换命令。
与原生画布矩阵不同，此矩阵是可读且可重用的。

方法：

- translate、rotate、scale对应上下文变换命令，允许你将变换输入矩阵中。该矩阵高效地保存了聚合的变换。
- setContextTransform接收一个上下文，并将该上下文的矩阵设置为此变换矩阵。这样可以高效地将存储在此矩阵中的所有变换重新应用到上下文中。
- resetContextTransform将上下文的变换重置为默认状态（即未变换状态）。
- getTransformedPoint接收一个未变换的坐标点，并将其转换为变换后的点。
- getScreenPoint接收一个变换后的坐标点，并将其转换为未变换的点。
- getMatrix以矩阵数组的形式返回聚合的变换。

代码：

```
var TransformationMatrix=( function(){
    // 私有
    var self;
    var m=[1,0,0,1,0,0];
    var reset=function(){ var m=[1,0,0,1,0,0]; }
    var multiply=function(mat){
        var m0=m[0]*mat[0]+m[2]*mat[1];
```

During complex animations you might apply dozens (or hundreds) of transformations to a shape. By using a transformation matrix you can (almost) instantly reapply those dozens of transformations with a single line of code.

Some Example uses:

- **Test if the mouse is inside a shape that you have translated, rotated & scaled**

  There is a built-in `context.isPointInPath` that tests if a point (eg the mouse) is inside a path-shape, but this built-in test is very slow compared to testing using a matrix.

  Efficiently testing if the mouse is inside a shape involves taking the mouse position reported by the browser and transforming it in the same way that the shape was transformed. Then you can apply hit-testing as if the shape was not transformed.

- **Redraw a shape that has been extensively translated, rotated & scaled.**

  Instead of reapplying individual transformations with multiple `.translate, .rotate, .scale` you can apply all the aggregated transformations in a single line of code.

- **Collision test shapes that have been translated, rotated & scaled**

  You can use geometry & trigonometry to calculate the points that make up transformed shapes, but it's faster to use a transformation matrix to calculate those points.

A Transformation Matrix "Class"

This code mirrors the native `context.translate`, `context.rotate`, `context.scale` transformation commands. Unlike the native canvas matrix, this matrix is readable and reusable.

Methods:

- `translate`, `rotate`, `scale` mirror the context transformation commands and allow you to feed transformations into the matrix. The matrix efficiently holds the aggregated transformations.
- `setContextTransform` takes a context and sets that context's matrix equal to this transformation matrix. This efficiently reapplies all transformations stored in this matrix to the context.
- `resetContextTransform` resets the context's transformation to it's default state (==untransformed).
- `getTransformedPoint` takes an untransformed coordinate point and converts it into a transformed point.
- `getScreenPoint` takes a transformed coordinate point and converts it into an untransformed point.
- `getMatrix` returns the aggregated transformations in the form of a matrix array.

Code:

```
var TransformationMatrix=( function(){
    // private
    var self;
    var m=[1,0,0,1,0,0];
    var reset=function(){ var m=[1,0,0,1,0,0]; }
    var multiply=function(mat){
        var m0=m[0]*mat[0]+m[2]*mat[1];
        var m1=m[1]*mat[0]+m[3]*mat[1];
        var m2=m[0]*mat[2]+m[2]*mat[3];
        var m3=m[1]*mat[2]+m[3]*mat[3];
        var m4=m[0]*mat[4]+m[2]*mat[5]+m[4];
        var m5=m[1]*mat[4]+m[3]*mat[5]+m[5];
        m=[m0,m1,m2,m3,m4,m5];
    }
    var screenPoint=function(transformedX,transformedY){
        // 反转
        var d =1/(m[0]*m[3]-m[1]*m[2]);
            im=[ m[3]*d, -m[1]*d, -m[2]*d, m[0]*d, d*(m[2]*m[5]-m[3]*m[4]), d*(m[1]*m[4]-m[0]*m[5]) ];
        // 点
        return({
x:transformedX*im[0]+transformedY*im[2]+im[4],
            y:transformedX*im[1]+transformedY*im[3]+im[5]
        });
    }
    var transformedPoint=function(screenX,screenY){
        return({
x:screenX*m[0] + screenY*m[2] + m[4],
            y:screenX*m[1] + screenY*m[3] + m[5]
        });
    }
    // 公共
    function TransformationMatrix(){
        self=this;
    }
    // 共享方法
TransformationMatrix.prototype.translate=function(x,y){
        var mat=[ 1, 0, 0, 1, x, y ];
multiply(mat);
    };
TransformationMatrix.prototype.rotate=function(rAngle){
        var c = Math.cos(rAngle);
        var s = Math.sin(rAngle);
        var mat=[ c, s, -s, c, 0, 0 ];
        multiply(mat);
    };
TransformationMatrix.prototype.scale=function(x,y){
        var mat=[ x, 0, 0, y, 0, 0 ];
multiply(mat);
    };
TransformationMatrix.prototype.skew=function(radianX,radianY){
        var mat=[ 1, Math.tan(radianY), Math.tan(radianX), 1, 0, 0 ];
        multiply(mat);
    };
TransformationMatrix.prototype.reset=function(){
        reset();
    }
TransformationMatrix.prototype.setContextTransform=function(ctx){
        ctx.setTransform(m[0],m[1],m[2],m[3],m[4],m[5]);
    }
TransformationMatrix.prototype.resetContextTransform=function(ctx){
        ctx.setTransform(1,0,0,1,0,0);
    }
TransformationMatrix.prototype.getTransformedPoint=function(screenX,screenY){
        return(transformedPoint(screenX,screenY));
    }
TransformationMatrix.prototype.getScreenPoint=function(transformedX,transformedY){
        return(screenPoint(transformedX,transformedY));
    }
TransformationMatrix.prototype.getMatrix=function(){
```

```
        var m1=m[1]*mat[0]+m[3]*mat[1];
        var m2=m[0]*mat[2]+m[2]*mat[3];
        var m3=m[1]*mat[2]+m[3]*mat[3];
        var m4=m[0]*mat[4]+m[2]*mat[5]+m[4];
        var m5=m[1]*mat[4]+m[3]*mat[5]+m[5];
        m=[m0,m1,m2,m3,m4,m5];
    }
    var screenPoint=function(transformedX,transformedY){
        // invert
        var d =1/(m[0]*m[3]-m[1]*m[2]);
        im=[ m[3]*d, -m[1]*d, -m[2]*d, m[0]*d, d*(m[2]*m[5]-m[3]*m[4]), d*(m[1]*m[4]-m[0]*m[5]) ];
        // point
        return({
            x:transformedX*im[0]+transformedY*im[2]+im[4],
            y:transformedX*im[1]+transformedY*im[3]+im[5]
        });
    }
    var transformedPoint=function(screenX,screenY){
        return({
            x:screenX*m[0] + screenY*m[2] + m[4],
            y:screenX*m[1] + screenY*m[3] + m[5]
        });
    }
    // public
    function TransformationMatrix(){
        self=this;
    }
    // shared methods
    TransformationMatrix.prototype.translate=function(x,y){
        var mat=[ 1, 0, 0, 1, x, y ];
        multiply(mat);
    };
    TransformationMatrix.prototype.rotate=function(rAngle){
        var c = Math.cos(rAngle);
        var s = Math.sin(rAngle);
        var mat=[ c, s, -s, c, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.scale=function(x,y){
        var mat=[ x, 0, 0, y, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.skew=function(radianX,radianY){
        var mat=[ 1, Math.tan(radianY), Math.tan(radianX), 1, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.reset=function(){
        reset();
    }
    TransformationMatrix.prototype.setContextTransform=function(ctx){
        ctx.setTransform(m[0],m[1],m[2],m[3],m[4],m[5]);
    }
    TransformationMatrix.prototype.resetContextTransform=function(ctx){
        ctx.setTransform(1,0,0,1,0,0);
    }
    TransformationMatrix.prototype.getTransformedPoint=function(screenX,screenY){
        return(transformedPoint(screenX,screenY));
    }
    TransformationMatrix.prototype.getScreenPoint=function(transformedX,transformedY){
        return(screenPoint(transformedX,transformedY));
    }
    TransformationMatrix.prototype.getMatrix=function(){
            var clone=[m[0],m[1],m[2],m[3],m[4],m[5]];
            return(clone);
        }
        // return public
        return(TransformationMatrix);
    })();
```

演示：

此演示使用上述的变换矩阵"类"来：

- 跟踪（即保存）一个矩形的变换矩阵。
- 在不使用上下文变换命令的情况下重绘变换后的矩形。
- 测试鼠标是否点击在变换后的矩形内部。

代码：

```
<!doctype html>
<html>
<head>
<style>
body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;
    function reOffset(){
        var BB=canvas.getBoundingClientRect();
        offsetX=BB.left;
        offsetY=BB.top;
    }
    var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
    window.onresize=function(e){ reOffset(); }

    // 变换矩阵"类"

    var TransformationMatrix=( function(){
        // 私有
        var self;
        var m=[1,0,0,1,0,0];
        var reset=function(){ var m=[1,0,0,1,0,0]; }
        var multiply=function(mat){
            var m0=m[0]*mat[0]+m[2]*mat[1];
            var m1=m[1]*mat[0]+m[3]*mat[1];
            var m2=m[0]*mat[2]+m[2]*mat[3];
            var m3=m[1]*mat[2]+m[3]*mat[3];
            var m4=m[0]*mat[4]+m[2]*mat[5]+m[4];
            var m5=m[1]*mat[4]+m[3]*mat[5]+m[5];
            m=[m0,m1,m2,m3,m4,m5];
        }
        var screenPoint=function(transformedX,transformedY){
            // 反转
```

```
            var clone=[m[0],m[1],m[2],m[3],m[4],m[5]];
            return(clone);
        }
        // return public
        return(TransformationMatrix);
    })();
```

Demo:

This demo uses the Transformation Matrix "Class" above to:

- Track (==save) a rectangle's transformation matrix.
- Redraw the transformed rectangle without using context transformation commands.
- Test if the mouse has clicked inside the transformed rectangle.

Code:

```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;
    function reOffset(){
        var BB=canvas.getBoundingClientRect();
        offsetX=BB.left;
        offsetY=BB.top;
    }
    var offsetX,offsetY;
    reOffset();
    window.onscroll=function(e){ reOffset(); }
    window.onresize=function(e){ reOffset(); }

    // Transformation Matrix "Class"

    var TransformationMatrix=( function(){
        // private
        var self;
        var m=[1,0,0,1,0,0];
        var reset=function(){ var m=[1,0,0,1,0,0]; }
        var multiply=function(mat){
            var m0=m[0]*mat[0]+m[2]*mat[1];
            var m1=m[1]*mat[0]+m[3]*mat[1];
            var m2=m[0]*mat[2]+m[2]*mat[3];
            var m3=m[1]*mat[2]+m[3]*mat[3];
            var m4=m[0]*mat[4]+m[2]*mat[5]+m[4];
            var m5=m[1]*mat[4]+m[3]*mat[5]+m[5];
            m=[m0,m1,m2,m3,m4,m5];
        }
        var screenPoint=function(transformedX,transformedY){
            // invert
            var d =1/(m[0]*m[3]-m[1]*m[2]);
im=[ m[3]*d, -m[1]*d, -m[2]*d, m[0]*d, d*(m[2]*m[5]-m[3]*m[4]), d*(m[1]*m[4]-m[0]*m[5])
];
            // 点
            return({
x:transformedX*im[0]+transformedY*im[2]+im[4],
                y:transformedX*im[1]+transformedY*im[3]+im[5]
            });
        }
        var transformedPoint=function(screenX,screenY){
            return({
x:screenX*m[0] + screenY*m[2] + m[4],
                y:screenX*m[1] + screenY*m[3] + m[5]
            });
        }
        // 公共
        function TransformationMatrix(){
            self=this;
        }
        // 共享方法
TransformationMatrix.prototype.translate=function(x,y){
            var mat=[ 1, 0, 0, 1, x, y ];
multiply(mat);
        };
TransformationMatrix.prototype.rotate=function(rAngle){
            var c = Math.cos(rAngle);
            var s = Math.sin(rAngle);
            var mat=[ c, s, -s, c, 0, 0 ];
            multiply(mat);
        };
TransformationMatrix.prototype.scale=function(x,y){
            var mat=[ x, 0, 0, y, 0, 0 ];
multiply(mat);
        };
TransformationMatrix.prototype.skew=function(radianX,radianY){
            var mat=[ 1, Math.tan(radianY), Math.tan(radianX), 1, 0, 0 ];
            multiply(mat);
        };
TransformationMatrix.prototype.reset=function(){
            reset();
        }
TransformationMatrix.prototype.setContextTransform=function(ctx){
            ctx.setTransform(m[0],m[1],m[2],m[3],m[4],m[5]);
        }
TransformationMatrix.prototype.resetContextTransform=function(ctx){
            ctx.setTransform(1,0,0,1,0,0);
        }
TransformationMatrix.prototype.getTransformedPoint=function(screenX,screenY){
            return(transformedPoint(screenX,screenY));
        }
TransformationMatrix.prototype.getScreenPoint=function(transformedX,transformedY){
            return(screenPoint(transformedX,transformedY));
        }
TransformationMatrix.prototype.getMatrix=function(){
            var clone=[m[0],m[1],m[2],m[3],m[4],m[5]];
            return(clone);
        }
        // return public
        return(TransformationMatrix);
    })();

    // DEMO starts here
```

```
            var d =1/(m[0]*m[3]-m[1]*m[2]);
            im=[ m[3]*d, -m[1]*d, -m[2]*d, m[0]*d, d*(m[2]*m[5]-m[3]*m[4]), d*(m[1]*m[4]-m[0]*m[5])
];
            // point
            return({
                x:transformedX*im[0]+transformedY*im[2]+im[4],
                y:transformedX*im[1]+transformedY*im[3]+im[5]
            });
        }
        var transformedPoint=function(screenX,screenY){
            return({
                x:screenX*m[0] + screenY*m[2] + m[4],
                y:screenX*m[1] + screenY*m[3] + m[5]
            });
        }
        // public
        function TransformationMatrix(){
            self=this;
        }
        // shared methods
        TransformationMatrix.prototype.translate=function(x,y){
            var mat=[ 1, 0, 0, 1, x, y ];
            multiply(mat);
        };
        TransformationMatrix.prototype.rotate=function(rAngle){
            var c = Math.cos(rAngle);
            var s = Math.sin(rAngle);
            var mat=[ c, s, -s, c, 0, 0 ];
            multiply(mat);
        };
        TransformationMatrix.prototype.scale=function(x,y){
            var mat=[ x, 0, 0, y, 0, 0 ];
            multiply(mat);
        };
        TransformationMatrix.prototype.skew=function(radianX,radianY){
            var mat=[ 1, Math.tan(radianY), Math.tan(radianX), 1, 0, 0 ];
            multiply(mat);
        };
        TransformationMatrix.prototype.reset=function(){
            reset();
        }
        TransformationMatrix.prototype.setContextTransform=function(ctx){
            ctx.setTransform(m[0],m[1],m[2],m[3],m[4],m[5]);
        }
        TransformationMatrix.prototype.resetContextTransform=function(ctx){
            ctx.setTransform(1,0,0,1,0,0);
        }
        TransformationMatrix.prototype.getTransformedPoint=function(screenX,screenY){
            return(transformedPoint(screenX,screenY));
        }
        TransformationMatrix.prototype.getScreenPoint=function(transformedX,transformedY){
            return(screenPoint(transformedX,transformedY));
        }
        TransformationMatrix.prototype.getMatrix=function(){
            var clone=[m[0],m[1],m[2],m[3],m[4],m[5]];
            return(clone);
        }
        // return public
        return(TransformationMatrix);
    })();

    // DEMO starts here
```

```
     // 创建一个矩形并添加一个变换矩阵
     // 用于跟踪其平移、旋转和缩放
     var rect={x:30,y:30,w:50,h:35,matrix:new TransformationMatrix()};

     // 以黑色绘制未变换的矩形
ctx.strokeRect(rect.x, rect.y, rect.w, rect.h);
     // 演示：标签
ctx.font='11px arial';
ctx.fillText('未变换的矩形',rect.x,rect.y-10);

     // 变换画布并以红色绘制变换后的矩形
ctx.translate(100,0);
     ctx.scale(2,2);
ctx.rotate(Math.PI/12);
     // 绘制变换后的矩形
ctx.strokeStyle='红色';
    ctx.strokeRect(rect.x, rect.\y, rect.\w, rect.\h);
     ctx.font='6像素 Arial';
     // 演示：标签
ctx.fillText('相同矩形：平移、旋转和缩放',rect.\x,rect.\y-6);
     // 重置上下文为未变换状态
ctx.setTransform(1,0,0,1,0,0);

     // 记录变换矩阵
     var m=rect.matrix;
m.translate(100,0);
m.scale(2,2);
m.rotate(Math.PI/12);

     // 使用矩形保存的变换矩阵重新定位、
     // 调整大小并重绘矩形
ctx.strokeStyle='蓝色';
    drawTransformedRect(rect);

     // 演示：说明
ctx.font='14px arial';
ctx.fillText('演示：点击蓝色矩形内',30,200);

     // 根据保存的变换矩阵重绘矩形
     function drawTransformedRect(r){
         // 使用矩形保存的矩阵设置上下文变换矩阵
m.setContextTransform(ctx);
         // 绘制矩形（无需更改位置或大小！）
ctx.strokeRect( r.x, r.y, r.w, r.h );
         // 重置上下文变换为默认（即未变换状态）；
m.resetContextTransform(ctx);
     }

     // 是变换后矩形中的点吗？
     function isPointInTransformedRect(r,transformedX,transformedY){
         var p=r.matrix.getScreenPoint(transformedX,transformedY);
         var x=p.x;
         var y=p.y;
         return(x>r.x && x<r.x+r.w && y>r.y && y<r.y+r.h);
     }

     // 监听鼠标按下事件
canvas.onmousedown=handleMouseDown;
     function handleMouseDown(e){
         // 告诉浏览器我们正在处理此事件
e.preventDefault();
e.stopPropagation();
```

```
    // create a rect and add a transformation matrix
    // to track it's translations, rotations & scalings
    var rect={x:30,y:30,w:50,h:35,matrix:new TransformationMatrix()};

    // draw the untransformed rect in black
    ctx.strokeRect(rect.x, rect.y, rect.w, rect.h);
    // Demo: label
    ctx.font='11px arial';
    ctx.fillText('Untransformed Rect',rect.x,rect.y-10);

    // transform the canvas & draw the transformed rect in red
    ctx.translate(100,0);
    ctx.scale(2,2);
    ctx.rotate(Math.PI/12);
    // draw the transformed rect
    ctx.strokeStyle='red';
    ctx.strokeRect(rect.x, rect.y, rect.w, rect.h);
    ctx.font='6px arial';
    // Demo: label
    ctx.fillText('Same Rect: Translated, rotated & scaled',rect.x,rect.y-6);
    // reset the context to untransformed state
    ctx.setTransform(1,0,0,1,0,0);

    // record the transformations in the matrix
    var m=rect.matrix;
    m.translate(100,0);
    m.scale(2,2);
    m.rotate(Math.PI/12);

    // use the rect's saved transformation matrix to reposition,
    //     resize & redraw the rect
    ctx.strokeStyle='blue';
    drawTransformedRect(rect);

    // Demo: instructions
    ctx.font='14px arial';
    ctx.fillText('Demo: click inside the blue rect',30,200);

    // redraw a rect based on it's saved transformation matrix
    function drawTransformedRect(r){
        // set the context transformation matrix using the rect's saved matrix
        m.setContextTransform(ctx);
        // draw the rect (no position or size changes needed!)
        ctx.strokeRect( r.x, r.y, r.w, r.h );
        // reset the context transformation to default (==untransformed);
        m.resetContextTransform(ctx);
    }

    // is the point in the transformed rectangle?
    function isPointInTransformedRect(r,transformedX,transformedY){
        var p=r.matrix.getScreenPoint(transformedX,transformedY);
        var x=p.x;
        var y=p.y;
        return(x>r.x && x<r.x+r.w && y>r.y && y<r.y+r.h);
    }

    // listen for mousedown events
    canvas.onmousedown=handleMouseDown;
    function handleMouseDown(e){
        // tell the browser we're handling this event
        e.preventDefault();
        e.stopPropagation();
        // 获取鼠标位置
mouseX=parseInt(e.clientX-offsetX);
        mouseY=parseInt(e.clientY-offsetY);
        // 鼠标是否在变换后的矩形内？
        if(isPointInTransformedRect(rect,mouseX,mouseY)){
            alert('你点击了变换后的矩形');
        }
    }

    // 演示：在不使用上下文变换命令的情况下重绘变换后的矩
    形
    function drawTransformedRect(r,color){
        var m=r.matrix;
        var tl=m.getTransformedPoint(r.x,r.y);
        var tr=m.getTransformedPoint(r.x+r.w,r.y);
        var br=m.getTransformedPoint(r.x+r.w,r.y+r.h);
        var bl=m.getTransformedPoint(r.x,r.y+r.h);
        ctx.beginPath();
ctx.moveTo(tl.x,tl.y);
        ctx.lineTo(tr.x,tr.y);
        ctx.lineTo(br.x,br.y);
        ctx.lineTo(bl.x,bl.y);
        ctx.closePath();
ctx.strokeStyle=color;
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=250></canvas>
</body>
</html>
```

```
        // get mouse position
        mouseX=parseInt(e.clientX-offsetX);
        mouseY=parseInt(e.clientY-offsetY);
        // is the mouse inside the transformed rect?
        if(isPointInTransformedRect(rect,mouseX,mouseY)){
            alert('You clicked in the transformed Rect');
        }
    }

    // Demo: redraw transformed rect without using
    //       context transformation commands
    function drawTransformedRect(r,color){
        var m=r.matrix;
        var tl=m.getTransformedPoint(r.x,r.y);
        var tr=m.getTransformedPoint(r.x+r.w,r.y);
        var br=m.getTransformedPoint(r.x+r.w,r.y+r.h);
        var bl=m.getTransformedPoint(r.x,r.y+r.h);
        ctx.beginPath();
        ctx.moveTo(tl.x,tl.y);
        ctx.lineTo(tr.x,tr.y);
        ctx.lineTo(br.x,br.y);
        ctx.lineTo(bl.x,bl.y);
        ctx.closePath();
        ctx.strokeStyle=color;
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=250></canvas>
</body>
</html>
```

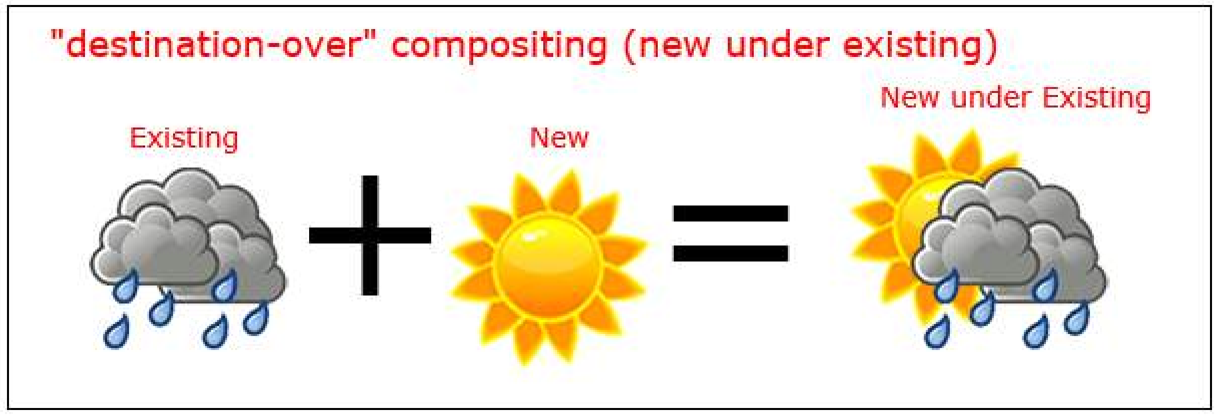
第17章：合成

第17.1节：使用“destination-over”在现有图形后绘制

```
context.globalCompositeOperation = "destination-over"
```

“destination-over”合成将新绘制内容放置在现有绘制内容的下方。

```
context.drawImage(rainy,0,0);
context.globalCompositeOperation='destination-over';  // sunny 在 rainy 下面
context.drawImage(sunny,0,0);
```



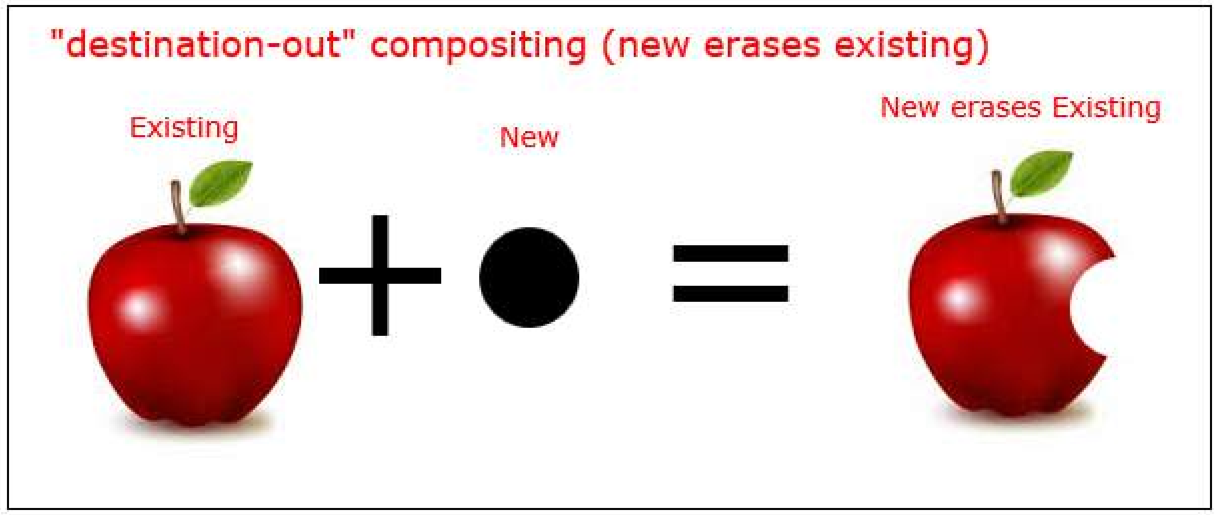
第17.2节：使用“destination-out”擦除现有图形

```
context.globalCompositeOperation = "destination-out"
```

“destination-out”合成使用新图形来擦除现有绘制内容。

新图形实际上并未被绘制——它只是用作“曲奇刀”来擦除现有像素。

```
context.drawImage(apple,0,0);
context.globalCompositeOperation = 'destination-out';   // 咬痕擦除
context.drawImage(bitemark,100,40);
```



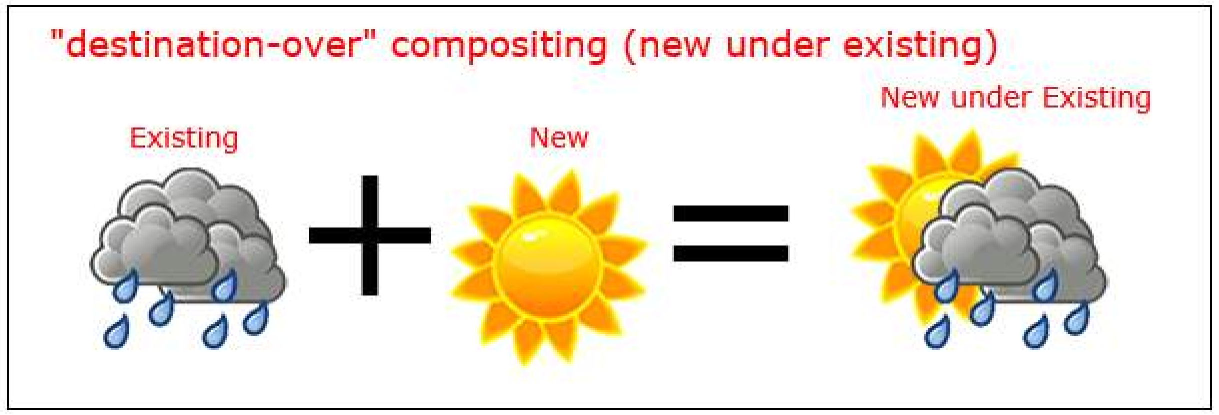
Chapter 17: Compositing

Section 17.1: Draw behind existing shapes with "destination-over"

```
context.globalCompositeOperation = "destination-over"
```

"destination-over" compositing places new drawing *under existing drawings*.

```
context.drawImage(rainy,0,0);
context.globalCompositeOperation='destination-over';  // sunny UNDER rainy
context.drawImage(sunny,0,0);
```



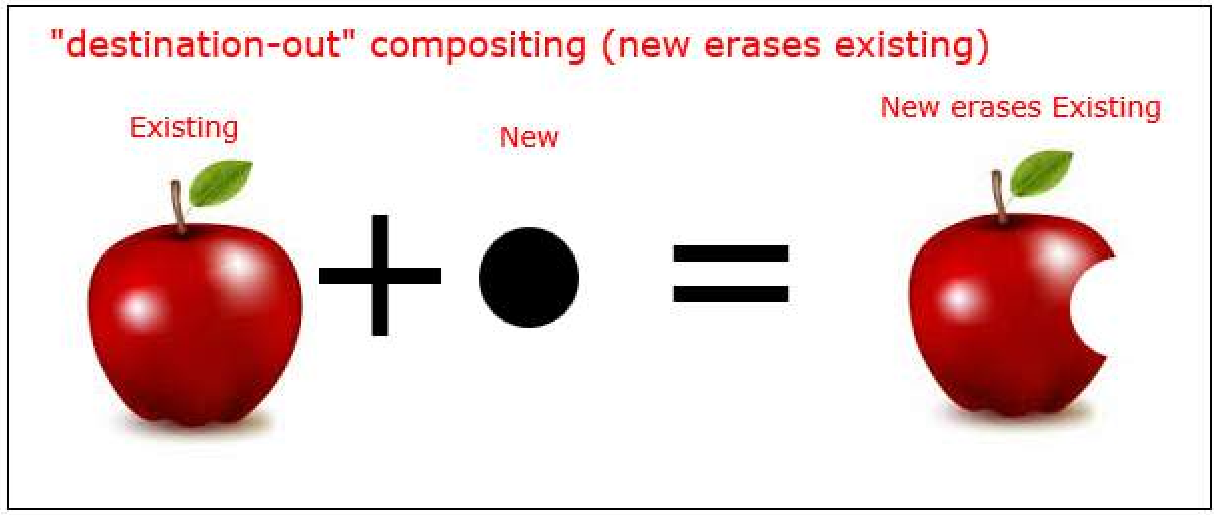
Section 17.2: Erase existing shapes with "destination-out"

```
context.globalCompositeOperation = "destination-out"
```

"destination-out" compositing uses new shapes to erase existing drawings.

The new shape is not actually drawn -- it is just used as a "cookie-cutter" to erase existing pixels.

```
context.drawImage(apple,0,0);
context.globalCompositeOperation = 'destination-out';   // bitemark erases
context.drawImage(bitemark,100,40);
```

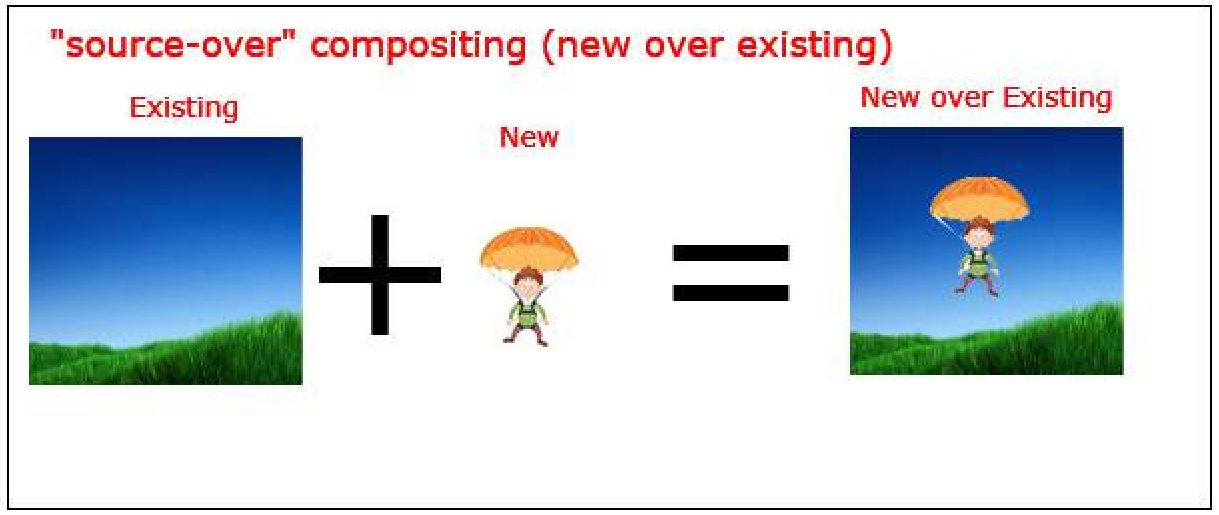


第17.3节：默认合成：新形状绘制在现有形状之上

```
context.globalCompositeOperation = "source-over"
```

“source-over”合成[默认]，将所有新绘制内容置于任何现有绘制内容之上。

```
context.globalCompositeOperation='source-over';  // 默认值
context.drawImage(background,0,0);
context.drawImage(parachuter,0,0);
```



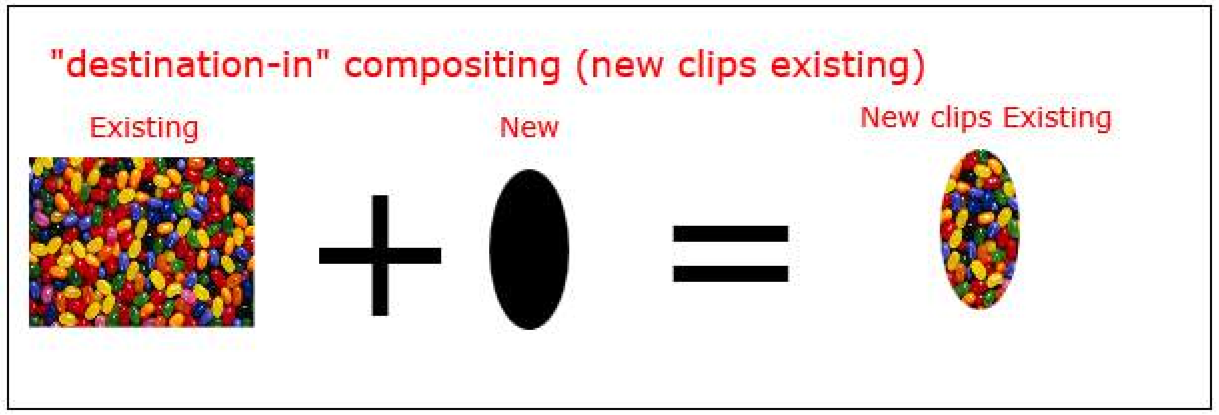
第17.4节：使用“destination-in”在形状内裁剪图像

```
context.globalCompositeOperation = "destination-in"
```

“destination-in”合成将在新形状内裁剪现有绘制内容。

注意：现有绘制内容中落在新绘制内容外的部分将被擦除。

```
context.drawImage(picture,0,0);
context.globalCompositeOperation='destination-in';  // 图片裁剪在椭圆内
context.drawImage(oval,0,0);
```



第17.5节：使用“source-in”在形状内裁剪图像

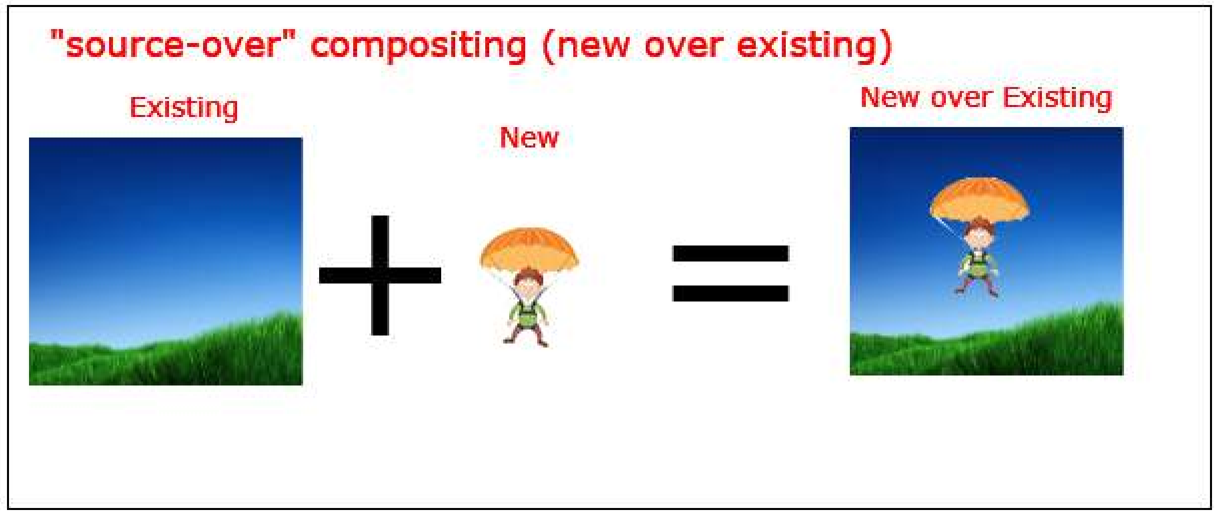
```
context.globalCompositeOperation = "source-in";
```

Section 17.3: Default compositing: New shapes are drawn over Existing shapes

```
context.globalCompositeOperation = "source-over"
```

"source-over" compositing **[default]**, places all new drawings over any existing drawings.

```
context.globalCompositeOperation='source-over';  // the default
context.drawImage(background,0,0);
context.drawImage(parachuter,0,0);
```



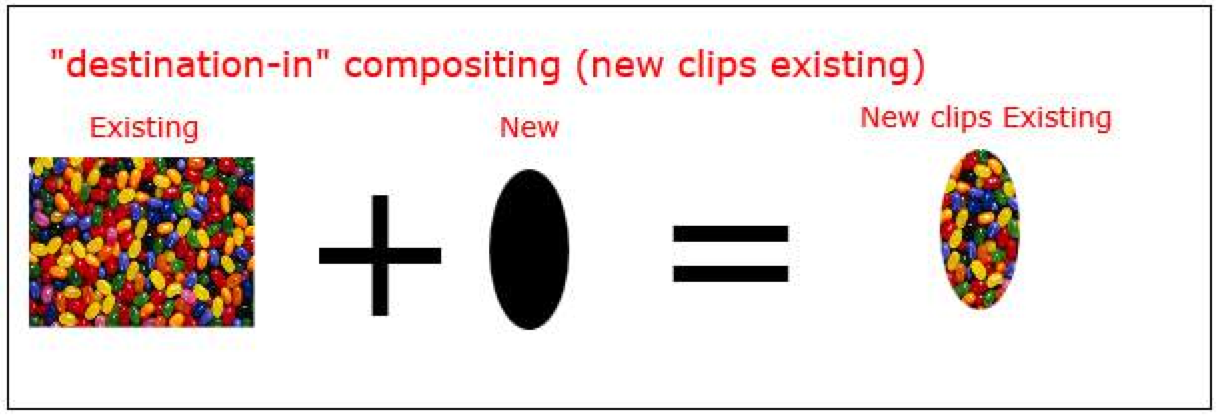
Section 17.4: Clip images inside shapes with "destination-in"

```
context.globalCompositeOperation = "destination-in"
```

"destination-in" compositing clips existing drawings inside a new shape.

Note: Any part of the existing drawing that falls outside the new drawing is erased.

```
context.drawImage(picture,0,0);
context.globalCompositeOperation='destination-in';  // picture clipped inside oval
context.drawImage(oval,0,0);
```



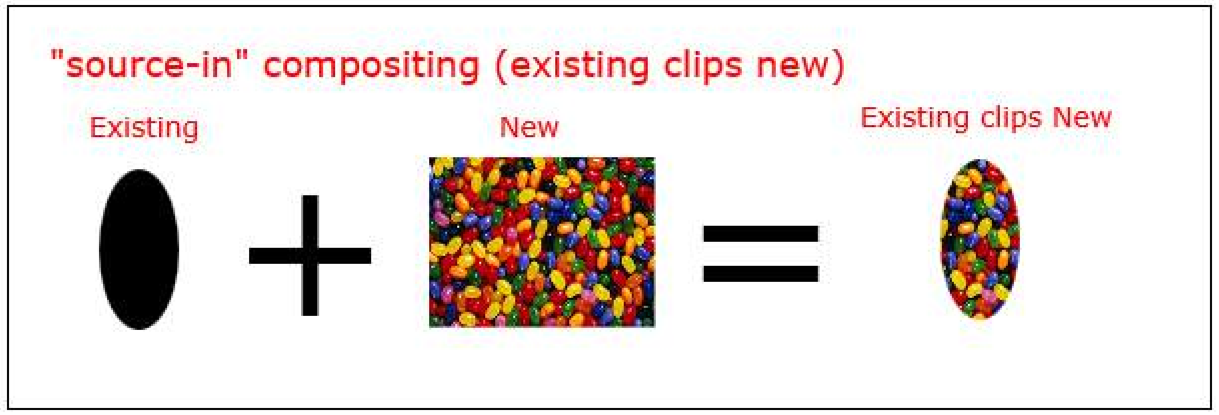
Section 17.5: Clip images inside shapes with "source-in"

```
context.globalCompositeOperation = "source-in";
```

source-in 合成时将新绘制内容裁剪在已有形状内。

注意：新绘制内容中落在已有绘制内容外的部分会被擦除。

```
context.drawImage(oval,0,0);
context.globalCompositeOperation='source-in';  // 图片裁剪在椭圆内
context.drawImage(picture,0,0);
```

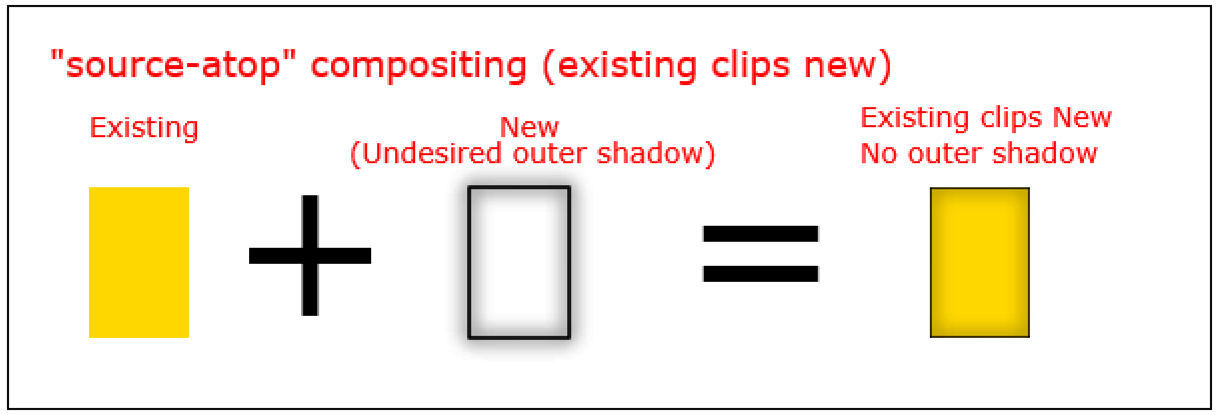


第17.6节：使用“source-atop”实现内阴影

```
context.globalCompositeOperation = 'source-atop'
```

source-atop 合成时将新图像裁剪在已有形状内。

```
// 镀金矩形
ctx.fillStyle='gold';
ctx.fillRect(100,100,100,75);
// 阴影
ctx.shadowColor='black';
ctx.shadowBlur=10;
// 限制新绘制覆盖现有像素
ctx.globalCompositeOperation='source-atop';
// 带阴影的描边
// "source-atop" 剪裁掉不需要的外部阴影
ctx.strokeRect(100,100,100,75);
ctx.strokeRect(100,100,100,75);
```



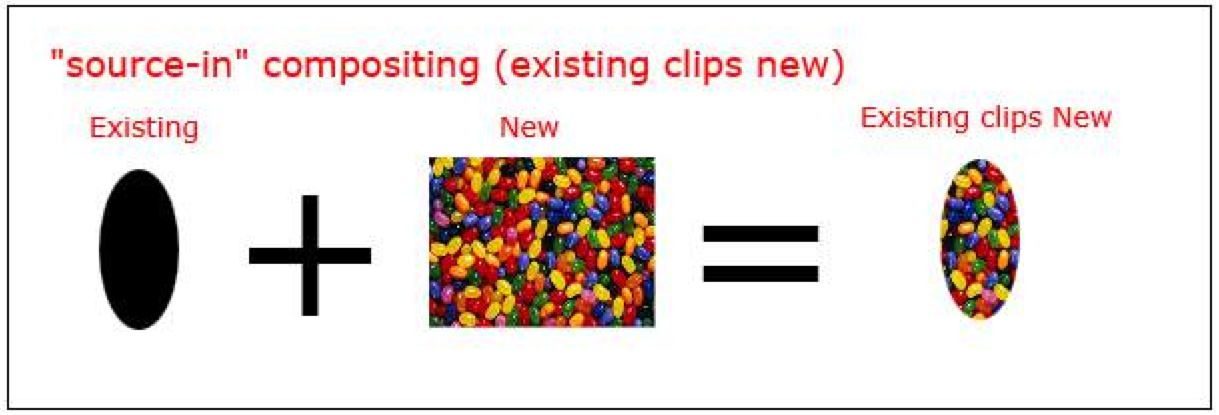
第17.7节：使用 "globalAlpha" 改变不透明度

```
context.globalAlpha=0.50
```

source-in compositing clips new drawings inside an existing shape.

Note: Any part of the new drawing that falls outside the existing drawing is erased.

```
context.drawImage(oval,0,0);
context.globalCompositeOperation='source-in';  // picture clipped inside oval
context.drawImage(picture,0,0);
```

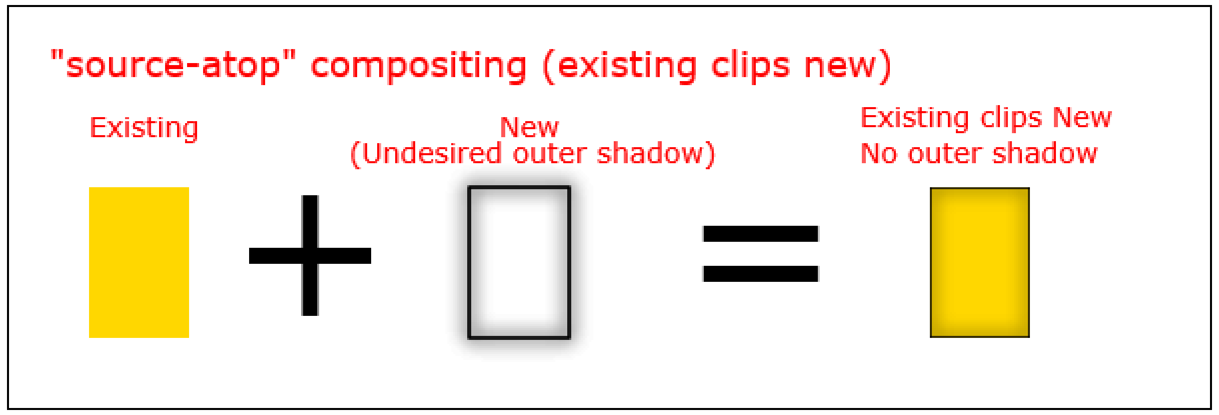


Section 17.6: Inner shadows with "source-atop"

```
context.globalCompositeOperation = 'source-atop'
```

source-atop compositing clips new image inside an existing shape.

```
// gold filled rect
ctx.fillStyle='gold';
ctx.fillRect(100,100,100,75);
// shadow
ctx.shadowColor='black';
ctx.shadowBlur=10;
// restrict new draw to cover existing pixels
ctx.globalCompositeOperation='source-atop';
// shadowed stroke
// "source-atop" clips off the undesired outer shadow
ctx.strokeRect(100,100,100,75);
ctx.strokeRect(100,100,100,75);
```



Section 17.7: Change opacity with "globalAlpha"

```
context.globalAlpha=0.50
```

你可以通过将globalAlpha设置为0.00（完全透明）到1.00（完全不透明）之间的值来改变新绘制内容的不透明度。

默认的globalAlpha值是1.00（完全不透明）。

现有的绘图不受globalAlpha的影响。

```
// 绘制一个不透明的矩形
context.fillRect(10,10,50,50);

// 将透明度改为50%——所有新的绘图将具有50%的不透明度
context.globalAlpha=0.50;

// 绘制一个半透明的矩形
context.fillRect(100,10,50,50);
```

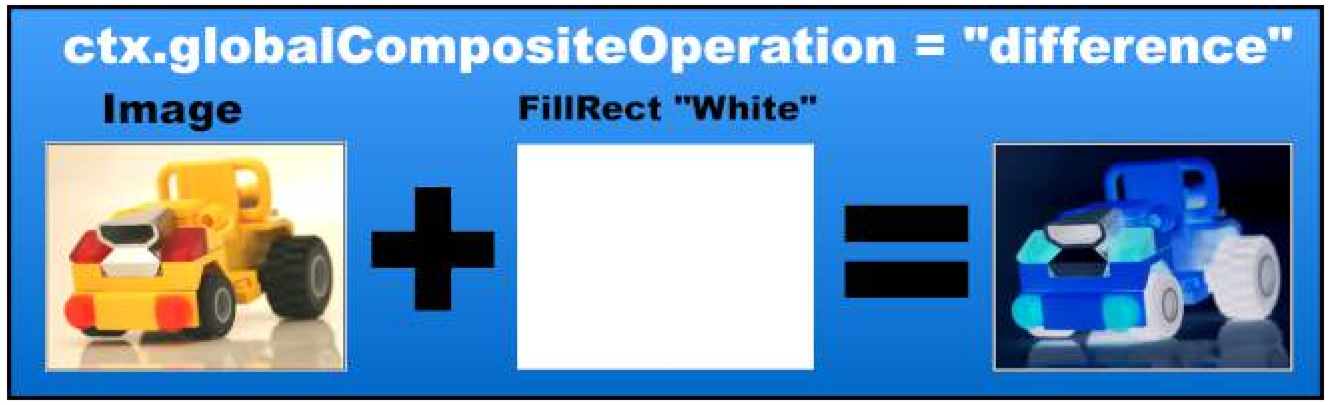
第17.8节：使用“difference”反转或取反图像

使用复合操作在图像上渲染一个白色矩形

```
ctx.globalCompositeOperation = 'difference';
```

效果的强度可以通过 alpha 设置进行控制

```
// 渲染图像
ctx.globalCompositeOperation='source-atop';
ctx.drawImage(image, 0, 0);

// 设置合成操作
ctx.globalCompositeOperation='difference';
ctx.fillStyle = "white";
ctx.globalAlpha = alpha;  // alpha 0 = 无效果 1 = 完全效果
ctx.fillRect(0, 0, image.width, image.height);
```



第17.9节：带“颜色”的黑白

通过以下方式去除图像颜色

```
ctx.globalCompositeOperation = 'color';
```

效果的强度可以通过 alpha 设置进行控制

```
// 渲染图像
ctx.globalCompositeOperation='source-atop';
```

You can change the opacity of new drawings by setting the `globalAlpha` to a value between 0.00 (fully transparent) and 1.00 (fully opaque).

The default `globalAlpha` is 1.00 (fully opaque).

Existing drawings are not affected by `globalAlpha`.

```
// draw an opaque rectangle
context.fillRect(10,10,50,50);

// change alpha to 50% -- all new drawings will have 50% opacity
context.globalAlpha=0.50;

// draw a semi-transparent rectangle
context.fillRect(100,10,50,50);
```

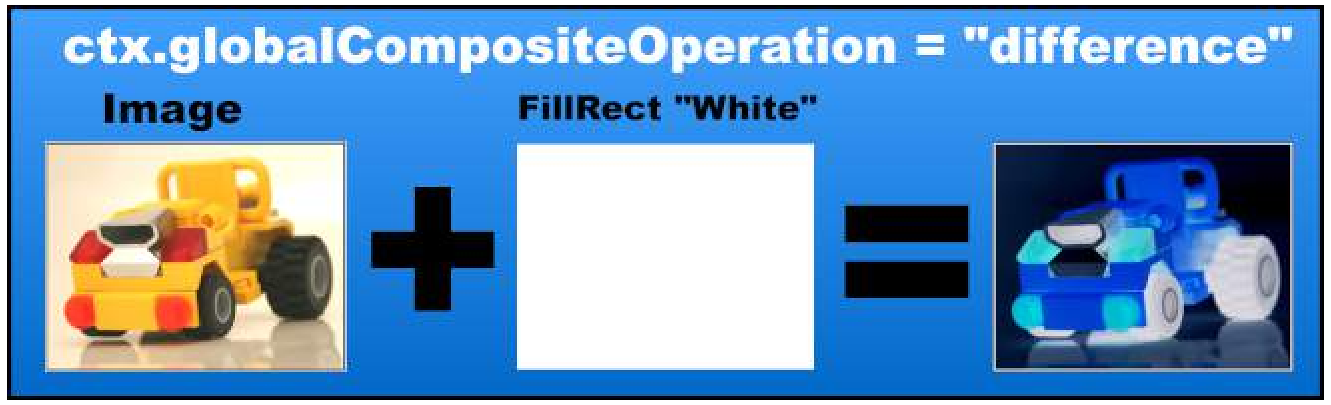
Section 17.8: Invert or Negate image with "difference"

Render a white rectangle over an image with the composite operation

```
ctx.globalCompositeOperation = 'difference';
```

The amount of the effect can be controlled with the alpha setting

```
// Render the image
ctx.globalCompositeOperation='source-atop';
ctx.drawImage(image, 0, 0);

// set the composite operation
ctx.globalCompositeOperation='difference';
ctx.fillStyle = "white";
ctx.globalAlpha = alpha;  // alpha 0 = no effect 1 = full effect
ctx.fillRect(0, 0, image.width, image.height);
```



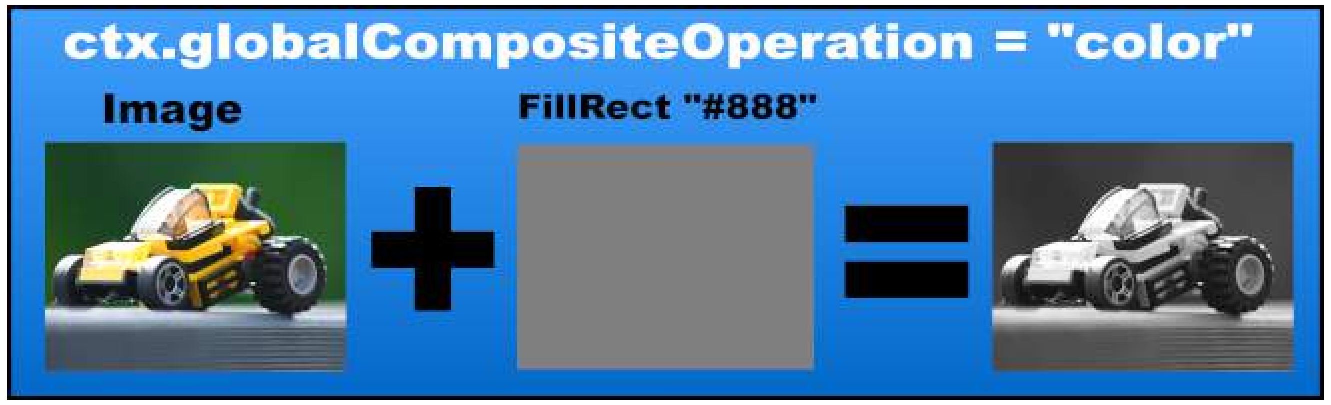
Section 17.9: Black & White with "color"

Remove color from an image via

```
ctx.globalCompositeOperation = 'color';
```

The amount of the effect can be controlled with the alpha setting

```
// Render the image
ctx.globalCompositeOperation='source-atop';
ctx.drawImage(image, 0, 0);

// 设置合成操作
ctx.globalCompositeOperation='color';
ctx.fillStyle = "white";
ctx.globalAlpha = alpha;  // alpha 0 = 无效果 1 = 完全效果
ctx.fillRect(0, 0, image.width, image.height);
```



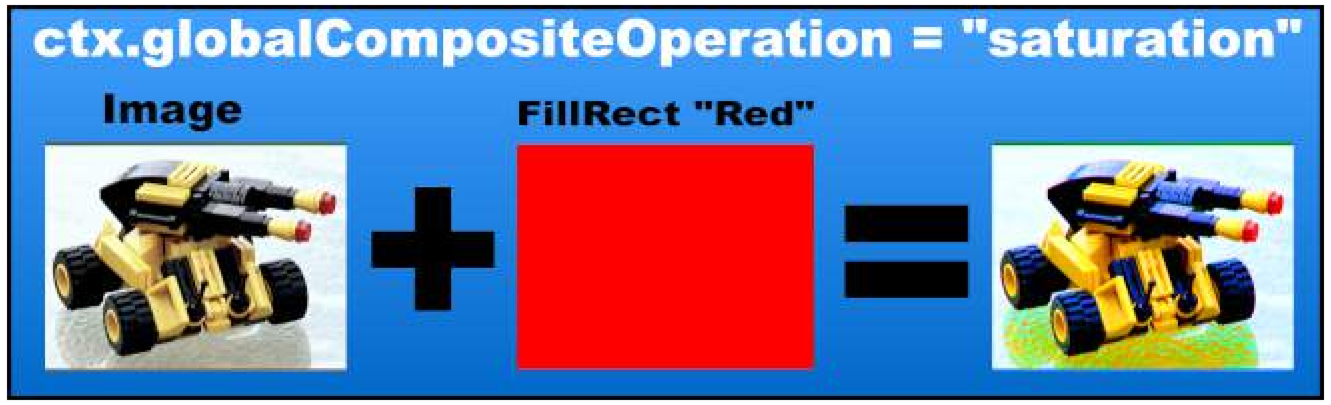
第17.10节：使用“饱和度”增加颜色对比度

使用以下方法增加图像的饱和度水平

```
ctx.globalCompositeOperation = 'saturation';
```

效果的强度可以通过alpha设置或填充叠加中的饱和度量来控制

```
// 渲染图像
ctx.globalCompositeOperation='source-atop';
ctx.drawImage(image, 0, 0);

// 设置合成操作
ctx.globalCompositeOperation ='saturation';
ctx.fillStyle = "red";
ctx.globalAlpha = alpha;  // alpha 0 = 无效果 1 = 完全效果
ctx.fillRect(0, 0, image.width, image.height);
```



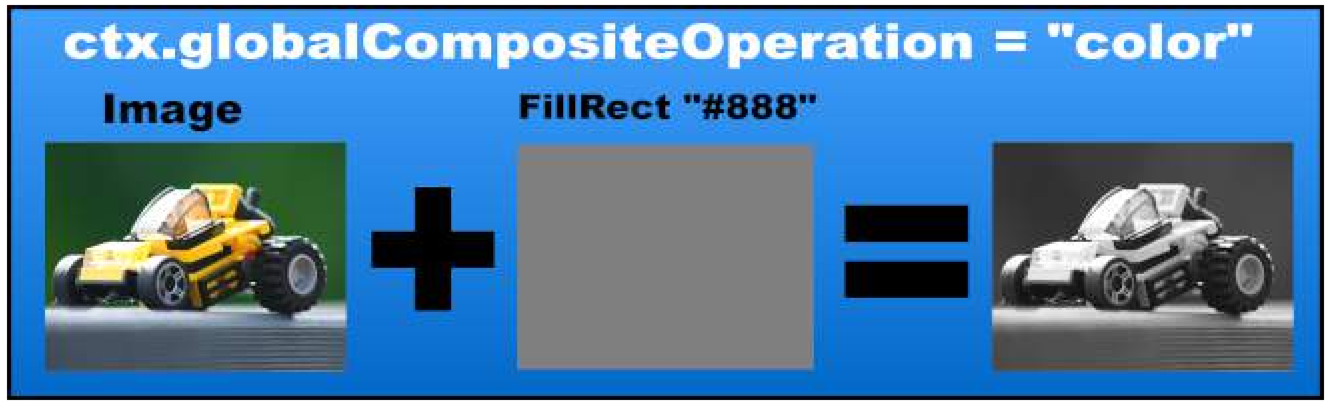
第17.11节：使用“亮度”实现棕褐色效果

使用以下方法创建彩色棕褐色效果

```
ctx.globalCompositeOperation = 'luminosity';
```

在这种情况下，棕褐色先被渲染，然后是图像。

```
ctx.drawImage(image, 0, 0);

// set the composite operation
ctx.globalCompositeOperation='color';
ctx.fillStyle = "white";
ctx.globalAlpha = alpha;  // alpha 0 = no effect 1 = full effect
ctx.fillRect(0, 0, image.width, image.height);
```



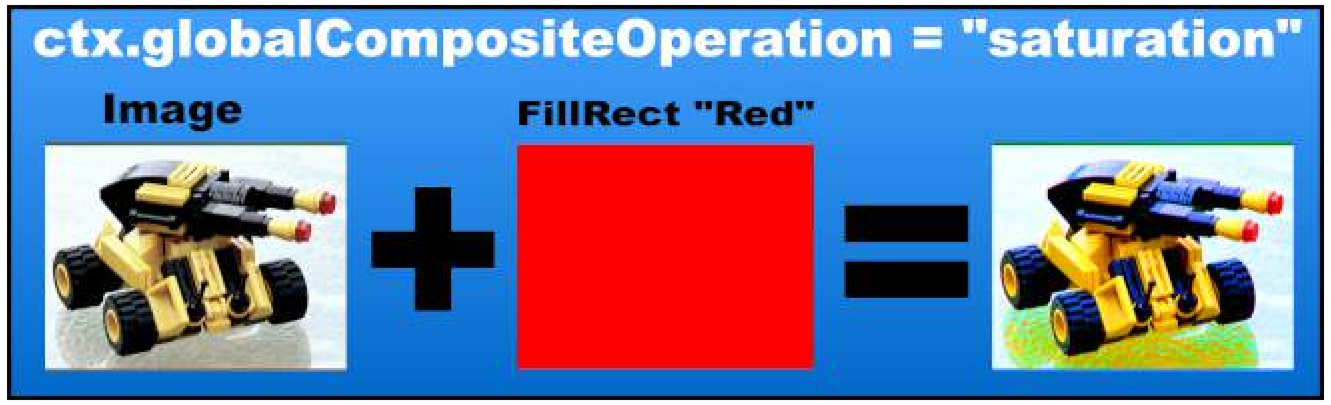
Section 17.10: Increase the color contrast with "saturation"

Increase the saturation level of an image with

```
ctx.globalCompositeOperation = 'saturation';
```

The amount of the effect can be controlled with the alpha setting or the amount of saturation in the fill overlay

```
// Render the image
ctx.globalCompositeOperation='source-atop';
ctx.drawImage(image, 0, 0);

// set the composite operation
ctx.globalCompositeOperation ='saturation';
ctx.fillStyle = "red";
ctx.globalAlpha = alpha;  // alpha 0 = no effect 1 = full effect
ctx.fillRect(0, 0, image.width, image.height);
```



Section 17.11: Sepia FX with "luminosity"

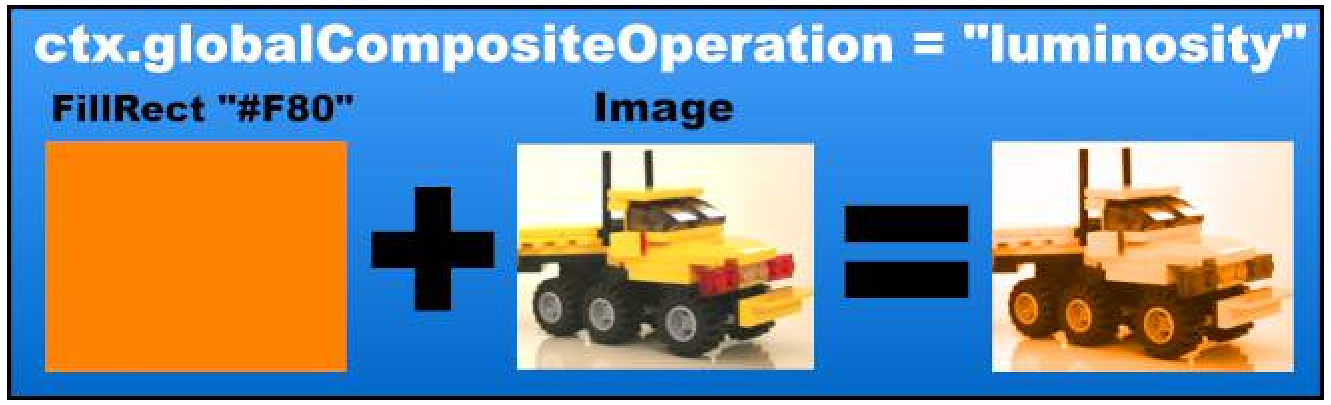
Create a colored sepia FX with

```
ctx.globalCompositeOperation = 'luminosity';
```

In this case the sepia colour is rendered first the the image.

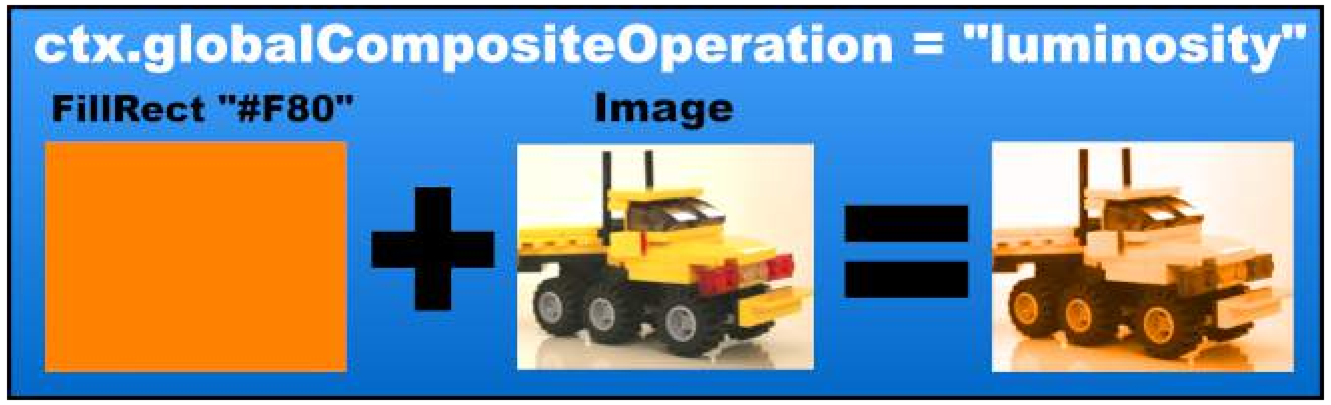
效果的强度可以通过alpha设置或填充叠加中的饱和度量来控制

```
// 渲染图像
ctx.globalCompositeOperation='source-atop';
ctx.fillStyle = "#F80";  // 复古效果的颜色
ctx.fillRect(0, 0, image.width, image.height);

// 设置合成操作
ctx.globalCompositeOperation ='luminosity';

ctx.globalAlpha = alpha;  // alpha 0 = 无效果 1 = 完全效果
ctx.drawImage(image, 0, 0);
```



The amount of the effect can be controlled with the alpha setting or the amount of saturation in the fill overlay

```
// Render the image
ctx.globalCompositeOperation='source-atop';
ctx.fillStyle = "#F80";  // the color of the sepia FX
ctx.fillRect(0, 0, image.width, image.height);

// set the composite operation
ctx.globalCompositeOperation ='luminosity';

ctx.globalAlpha = alpha;  // alpha 0 = no effect 1 = full effect
ctx.drawImage(image, 0, 0);
```



第18章：使用 "getImageData" 和 "putImageData" 进行像素操作

第18.1节："context.getImageData" 介绍

Html5 画布（Canvas）使您能够获取并更改画布上任意像素的颜色。

您可以使用画布的像素操作来：

- 为图像创建颜色选择器或在色轮上选择颜色。
- 创建复杂的图像滤镜，如模糊和边缘检测。
- 在像素级别重新着色图像的任何部分（如果使用HSL，甚至可以在保留重要的光照和饱和度的情况下重新着色图像，这样结果看起来不会像是随意涂抹的油漆）。
  注意：画布现在支持混合合成（Blend Compositing），在某些情况下也可以重新着色图像。
- “敲出”图像中人物/物体周围的背景，创建油漆桶工具以检测并洪
- 泛填充图像的部分区域（例如，将用户点击的花瓣颜色从绿色改为黄色）。

- 检查图像内容（例如人脸识别）。

常见问题：

- 出于安全原因，如果您绘制的图像来源于与网页不同的域，getImageData 将被禁用。

- getImageData 是一个相对耗费资源的方法，因为它会创建一个大型的像素数据数组，并且不使用GPU来辅助处理。注意：Canvas 现在支持混合合成，可以执行部分与getImageData相同的像素操作。

- 对于.png图像，getImageData 可能不会报告与原始.png文件完全相同的颜色，因为浏览器在将图像绘制到画布时允许进行伽马校正和预乘Alpha处理。

获取像素颜色

使用getImageData获取画布内容的全部或部分像素颜色。

`getImageData` 方法返回一个`imageData`对象

该`imageData`对象有一个`.data`属性，包含像素颜色信息。

data属性是一个Uint8ClampedArray，包含所有请求像素的红色、绿色、蓝色和Alpha（不透明度）颜色数据。

```
// 确定要获取哪些像素（此处获取画布上的所有像素）
var x=0;
var y=0;
var width=canvas.width;
var height=canvas.height;

// 获取 imageData 对象
var imageData = context.getImageData(x,y,width,height);

// 从 imageData 对象中提取像素颜色数据数组
var pixelDataArray = imageData.data;
```

你可以这样获取任意 [x,y] 像素在data数组中的位置：

Chapter 18: Pixel Manipulation with "getImageData" and "putImageData"

Section 18.1: Introduction to "context.getImageData"

Html5 Canvas gives you the ability to fetch and change the color of any pixel on the canvas.

You can use Canvas's pixel manipulation to:

- Create a color-picker for an image or select a color on a color-wheel.
- Create complex image filters like blurring and edge detection.
- Recolor any part of an image at the pixel level (if you use HSL you can even recolor an image while retaining the important Lighting & Saturation so the result doesn't look like someone slapped paint on the image).
  Note: Canvas now has Blend Compositing that can also recolor an image in some cases.
- "Knockout" the background around a person/item in an image,
- Create a paint-bucket tool to detect and Floodfill part of an image (eg, change the color of a user-clicked flower petal from green to yellow).
- Examine an image for content (eg. facial recognition).

Common issues:

- For security reasons, `getImageData` is disabled if you have drawn an image originating on a different domain than the web page itself.
- `getImageData` is a relatively expensive method because it creates a large pixel-data array and because it does not use the GPU to assist its efforts. Note: Canvas now has blend compositing that can do some of the same pixel manipulation that `getImageData` does.
- For .png images, `getImageData` might not report the exact same colors as in the original .png file because the browser is allowed to do gamma-correction and alpha-premultiplication when drawing images on the canvas.

Getting pixel colors

Use `getImageData` to fetch the pixel colors for all or part of your canvas content.

The `getImageData` method returns an `imageData` object

The `imageData` object has a `.data` property that contains the pixel color information.

The `data` property is a `Uint8ClampedArray` containing the Red, Green, Blue & Alpha (opacity) color data for all requested pixels.

```
// determine which pixels to fetch (this fetches all pixels on the canvas)
var x=0;
var y=0;
var width=canvas.width;
var height=canvas.height;

// Fetch the imageData object
var imageData = context.getImageData(x,y,width,height);

// Pull the pixel color data array from the imageData object
var pixelDataArray = imageData.data;
```

You can get position of any [x,y] pixel within `data` array like this:

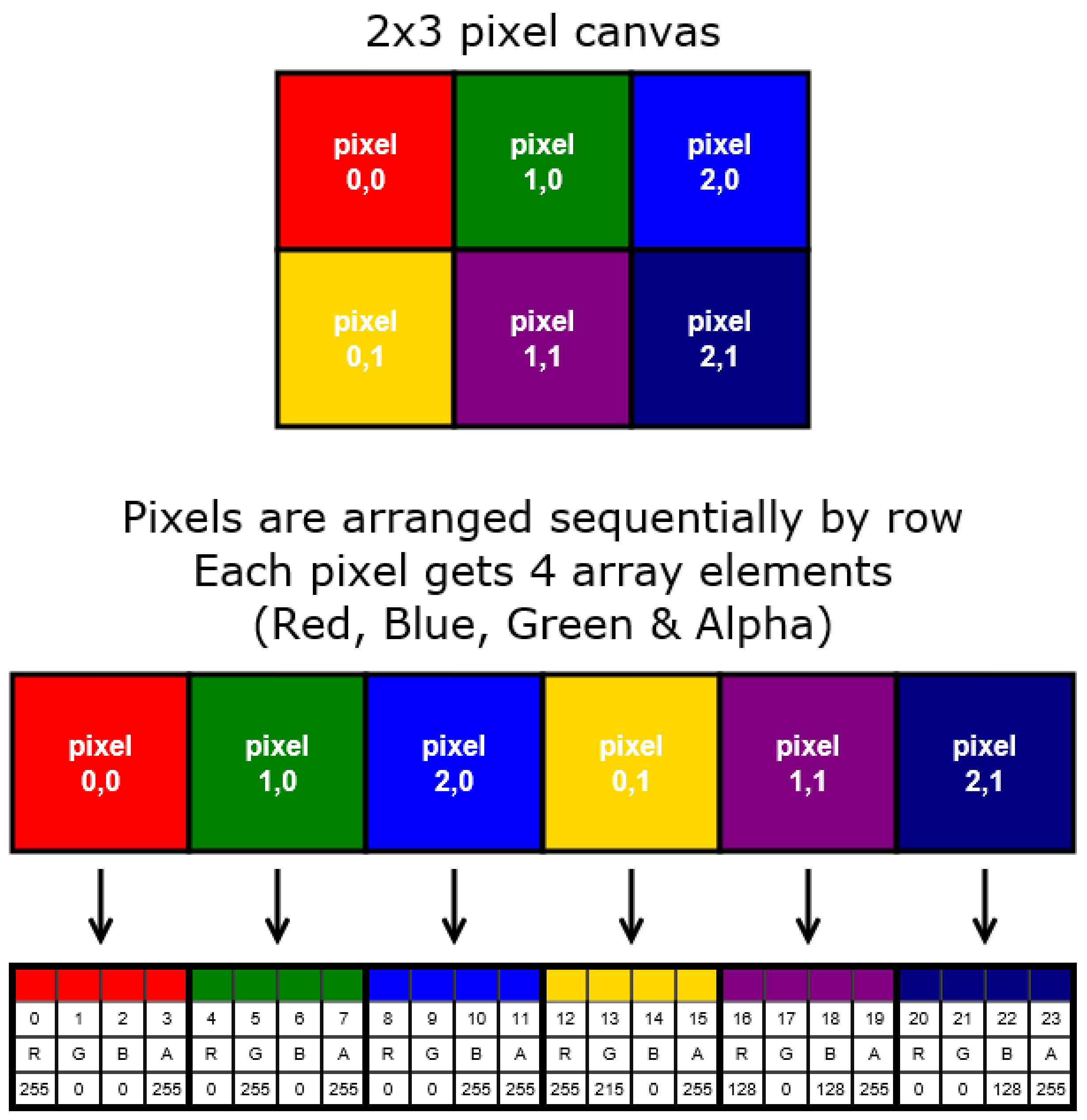
```
// 像素 [x,y] 在 data[] 数组中的位置
var n = y * canvas.width + x;
```

然后你可以这样获取该像素的红色、绿色、蓝色和透明度值：

```
// 像素 [x,y] 的 RGBA 信息
var red=data[n];
var green=data[n+1];
var blue=data[n+2];
var alpha=data[n+3];
```

展示像素数据数组结构的示意图

下面展示了 context.getImageData 在一个小型 2x3 像素画布上的示例：



🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟦	🟦	🟦	🟦	🟨	🟨	🟨	🟨	🟪	🟪	🟪	🟪	🟦	🟦	🟦	🟦
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
R	G	B	A	R	G	B	A	R	G	B	A	R	G	B	A	R	G	B	A	R	G	B	A
255	0	0	255	0	255	0	255	0	0	255	255	255	215	0	255	128	0	128	255	0	0	128	255

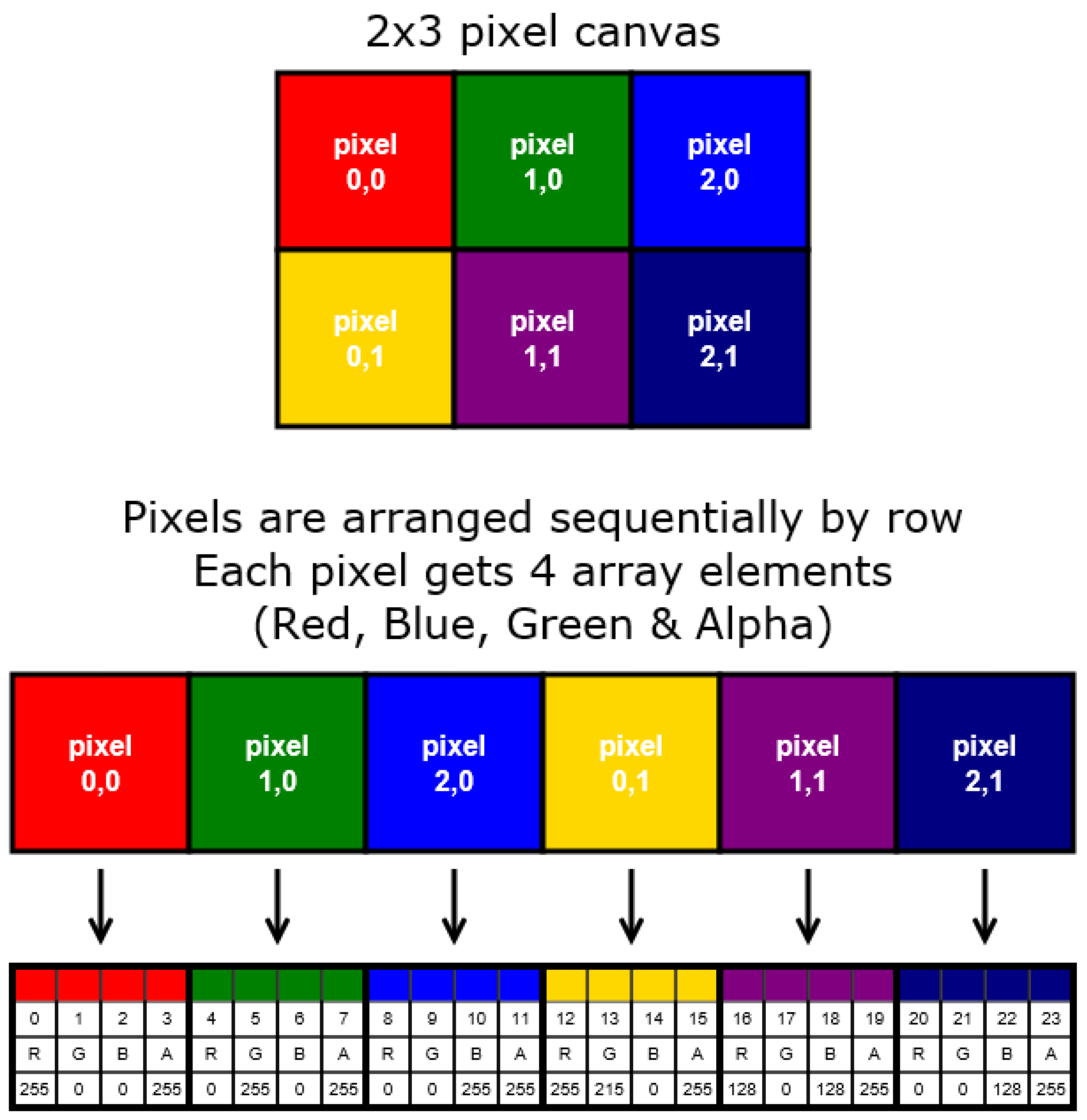
```
// the data[] array position for pixel [x,y]
var n = y * canvas.width + x;
```

And then you can fetch that pixel's red, green, blue & alpha values like this:

```
// the RGBA info for pixel [x,y]
var red=data[n];
var green=data[n+1];
var blue=data[n+2];
var alpha=data[n+3];
```

An Illustration showing how the pixel data array is structured

`context.getImageData` is illustrated below for a small 2x3 pixel sized canvas:



🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟦	🟦	🟦	🟦	🟨	🟨	🟨	🟨	🟪	🟪	🟪	🟪	🟦	🟦	🟦	🟦
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
R	G	B	A	R	G	B	A	R	G	B	A	R	G	B	A	R	G	B	A	R	G	B	A
255	0	0	255	0	255	0	255	0	0	255	255	255	215	0	255	128	0	128	255	0	0	128	255

鸣谢

非常感谢所有来自 Stack Overflow Documentation 的人员帮助提供这些内容，
更多更改可发送至 web@petercv.com 以发布或更新新内容

almcd	第2章
bjanes	第12章
Blindman67	第1、2、3、4、6、7、9、10、11、12、13、15、16和17章
凯伊多	第1、4、9和12章
markE	第1、2、3、4、5、6、7、8、10、11、13、14、15、16、17和18章
迈克·C	第12章
罗嫩·内斯	第12章
斯宾塞·维乔雷克	第1章
user2314737	第1章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

almcd	Chapter 2
bjanes	Chapter 12
Blindman67	Chapters 1, 2, 3, 4, 6, 7, 9, 10, 11, 12, 13, 15, 16 and 17
Kaiido	Chapters 1, 4, 9 and 12
markE	Chapters 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 13, 14, 15, 16, 17 and 18
Mike C	Chapter 12
Ronen Ness	Chapter 12
Spencer Wieczorek	Chapter 1
user2314737	Chapter 1