--

[svg基础 1](#_Toc29401)

[www 1](#_Toc15661)

[svg形状小结 2](#_Toc4227)

[简述 2](#_Toc252)

[demo 2](#_Toc11217)

[polyline与polygon即fill 2](#_Toc17287)

[填充属性 fill 3](#_Toc19701)

[描边属性 stroke 3](#_Toc13157)

[stroke 3](#_Toc23194)

[stroke-width 3](#_Toc27995)

[stroke-linecap 3](#_Toc5160)

[stroke-linejoin 3](#_Toc23045)

[stroke-miterlimit 3](#_Toc4949)

[stroke-opacity  3](#_Toc22941)

[描边属性 4](#_Toc14158)

[stroke-dasharray 4](#_Toc14098)

[stroke-dashoffset 4](#_Toc11758)

[描边动画化 4](#_Toc25813)

[圆的描边效果 4](#_Toc13276)

[直线的描边效果 4](#_Toc22785)

[curve指令 path d="" 5](#_Toc10900)

[简要说明 5](#_Toc19400)

[help 5](#_Toc23449)

[直线 5](#_Toc29260)

[绘制直线 5](#_Toc14956)

[简化 5](#_Toc24916)

[简化+ 5](#_Toc17141)

[书写建议 5](#_Toc24179)

[曲线 5](#_Toc16518)

[目录 6](#_Toc23452)

[简要说明 6](#_Toc27037)

[二次 6](#_Toc30785)

[二次 6](#_Toc24336)

[二次+ QT 6](#_Toc31845)

[三次 6](#_Toc20812)

[三次 6](#_Toc9552)

[三次+ CS 6](#_Toc25801)

[二次+ 三次+ 6](#_Toc6336)

[各种三次 7](#_Toc30436)

[三次与三次+的比较 7](#_Toc15055)

[椭圆 7](#_Toc11961)

[椭圆A 7](#_Toc17386)

[x-axis-rotatio 7](#_Toc25021)

[flag flag 7](#_Toc8067)

[其他 7](#_Toc21024)

[text 7](#_Toc8560)

[简单展示 8](#_Toc15677)

[分类 8](#_Toc20236)

[svg样式控制的优先级 8](#_Toc8876)

[外部style中的样式优先级高于行内 8](#_Toc8854)

[外部.class级别高于标签 8](#_Toc12457)

[demo1 8](#_Toc26862)

[demo2 9](#_Toc13050)

[demo3 9](#_Toc26938)

[demo 9](#_Toc447)

[demo-图片的圆角显示(pattern、clip-path) 10](#_Toc23823)

[使用SVG中的Symbol元素制作Icon（gulp） 10](#_Toc6261)

[SVG基础 | SVG defs元素、symbol元素和use元素 10](#_Toc2676)

[defs 10](#_Toc25430)

[g 10](#_Toc14783)

[symbol 10](#_Toc2626)

[use 10](#_Toc30899)

[symbol---建立图标系统 11](#_Toc15333)

[demo2 11](#_Toc31599)

[demo2+ 11](#_Toc17250)

[demo3 12](#_Toc12807)

[snap.js 12](#_Toc17423)

[基础 12](#_Toc8128)

[snap. 13](#_Toc5182)

[snap.path. 13](#_Toc10561)

[d3.js 13](#_Toc14125)

svg基础

www

|  |  |
| --- | --- |
|  | SVG 的属性可在mozilla的开发者社区查到：  [https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute](https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute" \t "http://www.uedsc.com/_blank)  zhangxinxu  <http://www.zhangxinxu.com/wordpress/?s=SVG>  SVG系列教程目录  <http://www.htmleaf.com/ziliaoku/qianduanjiaocheng/201507082192.html>  当前位置：[主页](http://www.htmleaf.com/) > [资料库](http://www.htmleaf.com/ziliaoku/) > [前端教程](http://www.htmleaf.com/ziliaoku/qianduanjiaocheng/) > JavaScript系列教程栏目列表  <http://www.htmleaf.com/ziliaoku/qianduanjiaocheng/javascript-tutorial-list.html> |
|  | snap.svg  <http://www.zhangxinxu.com/GitHub/demo-Snap.svg/demo/basic/Element.add.php>  d3.js  SVG的用法  <http://www.webhek.com/svg>  在线制作  <https://icomoon.io/app/#/select>  本地制作  <http://blog.sina.com.cn/s/blog_ac8bb8070101c7uj.html>  <http://www.cnblogs.com/lhb25/archive/2013/01/08/raphael-js-reference.html>  选择算法  <http://www.webhek.com/pathfinding> |
|  | 小tip: SVG和Canvas分别实现图片圆角效果  <http://www.zhangxinxu.com/wordpress/2014/06/svg-canvas-image-border-radius/> |
|  | svg动画  <http://www.520ued.com/article/svg-animate> |
|  | [Canvas 与 SVG](https://segmentfault.com/a/1190000002435534)  <https://segmentfault.com/a/1190000002435534> |
|  | HTML5新特性之用SVG绘制微信logo  <http://www.jb51.net/html5/427270.html> |
|  | [利用SVG中path实现平滑曲线](http://blog.csdn.net/ouyang111222/article/details/51153638)  <http://blog.csdn.net/ouyang111222/article/details/51153638> |
|  | 为什么要用SVG？svg与iconfont、图片多维度对比  <https://isux.tencent.com/why-svg.html> |
|  | JavaScript交互式图表工具：JointJS  http://www.tuicool.com/articles/faY3Uv  http://www.shangxueba.com/jingyan/2262721.html |
|  | 关注一切UI技术，专注数据可视化  <http://www.cnblogs.com/twaver/>  [HTML5大数据可视化效果（一）彩虹爆炸图](http://www.cnblogs.com/twaver/p/4547924.html)  <http://www.cnblogs.com/twaver/p/4547924.html>  [无插件纯Web 3D机房，HTML5+WebGL倾力打造](http://www.cnblogs.com/twaver/p/4631332.html)  <http://www.cnblogs.com/twaver/p/4631332.html> |
|  | <https://isux.tencent.com/why-svg.html>  为什么要用SVG？svg与iconfont、图片多维度对比 |
|  |  |
|  |  |
|  | 理解SVG坐标系和变换：视窗,viewBox和preserveAspectRatio  <http://www.w3cplus.com/html5/svg-coordinate-systems.html>  理解SVG viewport,viewBox,preserveAspectRatio缩放  <http://www.zhangxinxu.com/wordpress/2014/08/svg-viewport-viewbox-preserveaspectratio/> |

svg形状小结

|  |  |  |
| --- | --- | --- |
|  | 下面是每个形状的使用说明(这里只介绍控制图形形状和位置的基本属性，填充等属性放到后面总结)。 |  |
| 简述 | 矩形 - rect元素  这个元素有6个控制位置和形状的属性，分别是：  x：矩形左上角的坐标(用户坐标系)的x值。  y：矩形左上角的坐标(用户坐标系)的y值。  width：矩形宽度。  height：矩形高度。  rx：实现圆角效果时，圆角沿x轴的半径。  ry：实现圆角效果时，圆角沿y轴的半径。  比如上面例子中，实现了圆角效果，你也可以通过设置rx,ry为不同的值实现椭圆角效果。 |  |
| 圆 - circle元素  这个元素的属性很简单，主要是定义圆心和半径：  r：圆的半径。  cx：圆心坐标x值。  cy：圆心坐标y值。 |  |
| 椭圆 - ellipse元素  这个是更加通用的圆形元素，你可以分别控制半长轴和半短轴的长度，来实现不同的椭圆，很容易想到，当两个半轴相等时，就是正圆形了。  rx：半长轴(x半径)。  ry：半短轴(y半径)。  cx：圆心坐标x值。  cy：圆心坐标y值。 |  |
| 直线 - line元素  直线需要定义起点与终点即可：  x1：起点x坐标。  y1：起点y坐标。  x2：终点x坐标。  y2：终点y坐标。 |  |
| 折线 - polyline元素  折线主要是要定义每条线段的端点即可，所以只需要一个点的集合作为参数：  points：一系列的用空格，逗号，换行符等分隔开的点。每个点必须有2个数字：x值和y值。所以下面3个点 (0,0), (1,1)和(2,2)可以写成："0 0, 1 1, 2 2"。 |  |
| 多边形 - polygon元素  这个元素就是比polyline元素多做一步，把最后一个点和第一个点连起来，形成闭合图形。参数是一样的。  points：一系列的用空格，逗号，换行符等分隔开的点。每个点必须有2个数字：x值和y值。所以下面3个点 (0,0), (1,1)和(2,2)可以写成："0 0, 1 1, 2 2"。 |  |
| 路径 - path元素  这个是最通用，最强力的元素了；使用这个元素你可以实现任何其他的图形，不仅包括上面这些基本形状，也可以实现像贝塞尔曲线那样的复杂形状；此外，使用path可以实现平滑的过渡线段，虽然也可以使用polyline来实现这种效果，但是需要提供的点很多，而且放大了效果也不好。这个元素控制位置和形状的只有一个参数：  d：一系列绘制指令和绘制参数(点)组合成。  绘制指令分为绝对坐标指令和相对坐标指令两种，这两种指令使用的字母是一样的，就是大小写不一样，绝对指令使用大写字母，坐标也是绝对坐标；相对指令使用对应的小写字母，点的坐标表示的都是偏移量。  SVG path绘制注意事项：  绘制带孔的图形时要注意：外层边的绘制需要是逆时针顺序的，里面的洞的边的顺序必须是顺时针的。只有这样绘制的图形填充效果才会正确。  M = moveto  L = lineto  H = horizontal lineto  V = vertical lineto  C = curveto  S = smooth curveto  Q = quadratic Belzier curve  T = smooth quadratic Belzier curveto  A = elliptical ArcZ = closepath  所有命令均允许小写字母。大写表示绝对定位，小写表示相对定位。 |  |
| demo | <**svg** width="200" height="250">  <**rect** x="10" y="10" width="30" height="30"  stroke="black" fill="transparent" stroke-width="5"/>  <**rect** x="60" y="10" rx="10" ry="10" width="30" height="30"  stroke="black" fill="transparent" stroke-width="5"/>  <**circle** cx="25" cy="75" r="20"  stroke="red" fill="transparent" stroke-width="5"/>  <**ellipse** cx="75" cy="75" rx="20" ry="5"  stroke="red" fill="transparent" stroke-width="5"/>  <**line** x1="10" x2="50" y1="110" y2="150"  stroke="orange" fill="transparent" stroke-width="5"/>  <**polyline** points="60 110 65 120 70 115 75 130 80 125 85 140 90 135 95 150 100 145"  stroke="orange" fill="transparent" stroke-width="5"/>  <**polygon** points="50 160 55 180 70 180 60 190 65 205 50 195 35 205 40 190 30 180 45 180"  stroke="green" fill="transparent" stroke-width="5"/>  <**path** d="M20,230 Q40,205 50,230 T90,230"  fill="none" stroke="blue" stroke-width="5"/> </**svg**> |  |
| polyline与polygon即fill | 图形和线条  在这篇文章的开头我提到你可以填充[图形](http://www.vanseodesign.com/web-design/visual-grammar-shapes/" \t "http://www.w3cplus.com/svg/_blank)或者线条，这听起来可能有点怪怪的。你怎么可能填充线条呢？毕竟，它是由线条组成的。  除了容易混淆的nonzero和evenodd算法，你可能不需要我来帮你理解像矩形或者圆形这样的基础图形的填充和描边。描边是图形的轮廓，填充是轮廓之内的所有东西。  既然这样的话，[那线条呢](http://www.vanseodesign.com/web-design/visual-grammar-lines/" \t "http://www.w3cplus.com/svg/_blank)？大部分线条不会使用填充，你看到的大多是线条的描边。线条没有什么内部填充，但是如果它是折线的话，可能就可以被填充了。折线可以看成是由每条有端点的线段组成的多边形。  例如，下边图中的折线和多边形看起来完全一样。  但是如果你添加一个描边，你会看到多边形把整个图形的轮廓都用线条完整地从起点到终点绘制出来了，而折线则没有。尽管它们的填充是一样的。   |  | | --- | | <**svg** width="300px" height="200px">  <**polyline** points="10 10, 50 50, 75 175, 175 150, 175 50, 225 75, 225 150, 300 150" fill="red"/> </**svg**> <**svg** width="300px" height="200px">  <**polygon** points="10 10, 50 50, 75 175, 175 150, 175 50, 225 75, 225 150, 300 150" fill="red"/> </**svg**> <**svg** width="300px" height="200px">  <**polyline** points="10 10, 50 50, 75 175, 175 150, 175 50, 225 75, 225 150, 300 150" fill="red"  stroke="#000" stroke-width="5"/> </**svg**> <**svg** width="300px" height="200px">  <**polygon** points="10 10, 50 50, 75 175, 175 150, 175 50, 225 75, 225 150, 300 150" fill="red"  stroke="#000" stroke-width="5"/> </**svg**> | |  | | |

填充属性 fill

|  |  |
| --- | --- |
|  | SVG基础——填充和描边  http://www.w3cplus.com/svg/svg-fill-stroke.html |
|  | 正如你所理解的，[填充属性](http://www.w3.org/TR/SVG/painting.html" \l "FillProperties" \t "http://www.w3cplus.com/svg/_blank)就是用你选定的颜色来填充你创建的图形（或线条）的内部区域。这是使用了填充的矩形和圆形：  <rect class="rectangle" width="100%" height="100%" fill="red" /><circle class="circle" cx="150" cy="150" r="100" fill="#039" />  你也可以在你的CSS中设置填充值，下面我在不同的class类中设置了填充。但是如果你喜欢的话，你也可以直接在SVG属性中设置。  .rectangle {fill: red;}  .circle {fill: #039;}  默认情况下的填充颜色是 black 或者 #000000 ，所以如果你不想要你的图形被填充的话，你需要显式地设置 fill 为 none 。  <rect class="rectangle" width="100%" height="100%" fill="none" /> |
|  | fill-rule 属性  关于图形（或线条）的内部区域，是由 fill-rule 属性决定的，它有 nonzero 和 evenodd 两个值：  fill-rule="nonzero"fill-rule="evenodd"  我并不打算假装我是完全了解这两种算法是如何工作的。下面是[官方定义](http://www.w3.org/TR/SVG/painting.html" \l "FillProperties" \t "http://www.w3cplus.com/svg/_blank)。还有来自规范文件的这两个算法分别的示例图像：  nonzero—这个规则通过从canvas上的某个点往任一方向绘制射线到无穷远，然后检查图形的线段和射线相交的点，来确定“内部区域”。从0开始计数，每次路径线段是从左到右穿过射线就加一，从右到左的就减一。通过计算交叉点，如果结果是0，则这个点在路径外边，不然，就是在里边。    evenodd—这个规则通过从canvas上某个点往任一方向绘制射线到无穷远，然后计算给定图形上线段路径和该射线交叉点的数量。如果这个数是奇数，那么该点在图形内部；如果是偶数，该点在图形外部。 |
|  | fill-opacity 属性  你还可以设置填充的 fill-opacity 属性。  fill-opacity="0.5"  它的取值应在0（透明）和1（不透明）之间，和CSS的[opacity属性](http://www.vanseodesign.com/css/transparency/" \t "http://www.w3cplus.com/svg/_blank)一样 |

描边属性 stroke

|  |  |
| --- | --- |
|  | stroke系列属性主要与“边”有关，比如stroke(边的颜色)，stroke-width(边的粗细)，stroke-linecap(边起始与终点的形状)，stroke-linejoin(边折角时的形状) 等等。 |
| stroke | stroke 表示描边颜色。这很有意思，名字不是stroke-color, 而就是单纯的stroke. 其值，官方称之为”paint“，我就不上梁小丑般翻译了。一般有如下类型值：none,currentColor, <color>. none表示没有颜色，<color>就是我们常规的颜色值。RGBA, HSBA都支持。currentColor略高深，我看了下[官方文档](http://www.w3.org/TR/SVG11/painting.html" \l "SpecifyingPaint)，个人理解为：共享父级但不越过SVG元素的XML中color(style中的)值；可以让路径绘制的文字直接继承父标签的color颜色值。  <path stroke="red" d="M5 20 l215 0" />  <path stroke="blue" d="M5 40 l215 0" />  <path stroke="black" d="M5 60 l215 0" /> |
| stroke-width | stroke-width 表示描边的粗细。  <path stroke-width="2" d="M5 20 l215 0" />  <path stroke-width="4" d="M5 40 l215 0" />  <path stroke-width="6" d="M5 60 l215 0" /> |
| stroke-linecap | stroke-linecap 表示描边端点表现方式。可用值有：butt, round, square, inherit. 表现如下图：  IMG_256  <path stroke-linecap="butt" d="M5 20 l215 0" />  <path stroke-linecap="round" d="M5 40 l215 0" />  <path stroke-linecap="square" d="M5 60 l215 0" /> |
| stroke-linejoin | stroke-linejoin 表示描边转角的表现方式。可用值有：miter, round, bevel, inherit. 表现如下图： IMG_257 |
| stroke-miterlimit | stroke-miterlimit表示描边相交（锐角）的表现方式。默认大小是4. 什么斜角转斜面的角度损耗之类的意思，值越大，损耗越小。具体干嘛的，我自己也不确定。大家可查查其他资料。  stroke-miterlimit 属性给 miter-length 和 stroke-width 之间的比率做了限制，它的比值范围应大于或等于1。当比值不在这个范围的时候， stroke 就会被转换成斜角（bevel）。 |
| stroke-opacity | stroke-opacity 表示描边透明度。默认是1. |
| 描边属性 | SVG 的描边动画重点与两个属性有关， stroke-dasharray 和 stroke-dashoffset  stroke-dasharray 表示虚线描边。可选值为：none, <dasharray>, inherit. 其中，none表示不是虚线；<dasharray>为一个逗号或空格分隔的数值列表。表示各个虚线端的长度。可以是固定的长度值，也可以是百分比值；inherit表继承。  stroke-dashoffset 表示虚线的起始偏移。可选值为：<percentage>, <length>, inherit. 百分比值，长度值，继承。 |
| stroke-dasharray | 说明  stroke-dasharray: 该属性控制路径中 虚线的长度 以及 虚线间的距离 。  我们知道虚线由一些间隔的实线组成，而该属性接受一串数字(可以理解为是数组)，然后下标为0、2、4…的数字表示这段路径实线的长度，下标1、3、5…的数字表示间隔(空白)的长度。  实现虚线  stroke-dasharray:1,1 或者省略的写法 stroke-dasharray:1  第一个数字表示实线长度 1，第二个表示间隔长度 1，后面则循环；如果要加大间隔而不加长实线，则 stroke-dasharray:1,5  注：  stroke-dasharray:1,2,3;可以写为  stroke-dasharray:1 2 3;等效于  stroke-dasharray:1,2,3,1,2,3;  demo:  <svg width="600px" height="300px">  <line x1="0" y1="20" x2="600" y2="20" stroke-dasharray="10" stroke="#000" stroke-width="3"/>  <line x1="0" y1="40" x2="600" y2="40" stroke-dasharray="1 1" stroke="#000" stroke-width="3"/>  <line x1="0" y1="60" x2="600" y2="60" stroke-dasharray="10 2" stroke="#000" stroke-width="3"/>  <line x1="0" y1="80" x2="600" y2="80" stroke-dasharray="5 10" stroke="#000" stroke-width="3"/>  <line x1="0" y1="100" x2="600" y2="100" stroke-dasharray="20 10 5 5 5 10" stroke="#000" stroke-width="3"/>  <line x1="0" y1="120" x2="600" y2="120" stroke-dasharray="8 2" stroke="#000" stroke-width="3"/>  <path d="m 0 140 h 600" stroke-dasharray="8 2" stroke="#000" stroke-width="3"></path>  <path d="m 0 160 h 600" stroke-dasharray="8,2" stroke="#000" stroke-width="3"></path> </svg> |
| stroke-dashoffset | 说明  该属性指定了虚线开始时的偏移长度；正数从路径起始点向前偏移，负数则向后  效果1  stroke-dasharray:15, 10, 5, 10;  stroke-dashoffset:0;  [IMG_256](http://www.uedsc.com/wp-content/uploads/2015/05/danshixian3.jpg)  效果2  stroke-dasharray:15, 10, 5, 10;  stroke-dashoffset:15;  [IMG_257](http://www.uedsc.com/wp-content/uploads/2015/05/danshixian4.jpg) |
| 描边动画化 | 0、我们做个假设，下面这条路径的总长度为180；  [IMG_256](http://www.uedsc.com/wp-content/uploads/2015/05/danshixian5.jpg)  1、我们设置这条路径的 storke-dasharray:180 ，于是这条路径就变成了由长度180的实线与长度180间隔组成；但是，它的总长度只有180，因此看上去，还是这样，但在实线的后面，跟着一段长度180的间隔；  [IMG_257](http://www.uedsc.com/wp-content/uploads/2015/05/danshixian6.jpg)  2、于是，当我们设置 stroke-dashoffset:100 , 我们将该虚线向前偏移了100：  那我们设置 stroke-dashoffset:180 ， 好吧，该路径就变成了空白。  3、于是，我们设置该路径为这样  stroke-dasharray: 180;  stroke-dashoffset: 180;  然后将 stroke-dashoffset 逐渐变成 0；  点击这里查看示例：[svg-animate](http://www.uedsc.com/wp-content/uploads/2015/05/svg-animate.html)  [F:\HTML\practice\svg\关于svg路径的动态效果\路径 逐步显示.html]  4、将 stroke-dashoffset 逐渐变成0的过程，可以通过CSS3来实现，也可以通过JavaScript来实现，这两种实现方式的优缺点嘛：  通过JavaScript，可以精确的得到每条路径的总长度(getTotalLength)，从而动画时长可以精确控制，缺点就是需要写一连串动画控制函数；  通过CSS3-animation，上述的优缺点正好相反，特别是，当页面中有多条路径进行一致的描边动画，它们会同步开始，但不能同步结束。  所以，通过JavaScript计算出所有页面中路径的长度，并设置 stroke-dasharray , stroke-dashoffset ，然后通过CSS3来进行执行动画，这应该是最好的解决方法。  **circle**.setAttribute('stroke-dasharray', perimeter \* percent + " " + perimeter \* (1- percent)); |
| 圆的描边效果 | .showLine {  -webkit-animation: lineTo 1200ms cubic-bezier(0.99, 0.01, 0.62, 0.94) 0.5s both;  **/\*name runTime timeFun layTime type\*/** stroke-dasharray: 565.48668;**/\* 2\*3.14\*90 \*//\* 虚线的长度 \*/** stroke-dashoffset: 565.48668; **/\* 缩进去 \*/** } .divWrap:hover .showLine{  stroke-dashoffset: 300.7876;  -webkit-transition: stroke-dashoffset 1s; } **@-webkit-keyframes** lineTo {  to {  stroke-dashoffset: 101.7876;  } } |
| 直线的描边效果 | <svg id="svgForStroke" width="800" height="200" xmlns="http://www.w3.org/2000/svg">  <line fill="none" stroke="#000000" stroke-width="5"  stroke-dasharray="400,400" stroke-dashoffset="400"  x1="0" y1="90" x2="400" y2="90"/> </svg>  stroke-dashoffset值由400变为0的过程，即可实现描边效果  <style>  line {  stroke-dasharray: 400;  stroke-dashoffset: 400;  animation: dash 5s linear infinite;  }   @keyframes dash {  to {  stroke-dashoffset: 0;  }  } </style> |
|  | 路径、线条的准确长度。可能需要借助JavaScript，类似下面的代码：  document.querySelectorAll( 'path' )[0].getTotalLength()  document.querySelector('line').getBoundingClientRect() |

|  |  |
| --- | --- |
|  | SVG属性和CSS  对于这个系列中的大部分示例，我都在不同的SVG元素上设置了SVG的属性作为其元素属性。之前的文章中我也展示了如何在你的CSS文件中设置属性值。你可以在CSS中设置更多的SVG属性，虽然不是所有的属性都可以，但是至少填充和描边是没有问题的。  你可以添加内联CSS如下：  <svg width="300px" height="300px">  <rect x="20" y="20" width="250px" height="125px"  style="fill: teal; stroke: 5px;" />  </svg>  你也可以直接在你的CSS文件中设置样式。  <svg width="300px" height="300px">  <rect class="example" x="20" y="20" width="250px" height="125px" />  </svg>  .example {  fill: teal;  stroke: red;  stroke-width: 5px  }  记住，把填充设置为和背景颜色不同的颜色，设置stroke和stroke-width，而不是border-color或者border-width。  很多其它的SVG属性都可以写成CSS属性，而不是把它们作为属性添加给某个SVG元素。W3C有在维护这样的一个列表——[可以用CSS样式来写的各个SVG属性](http://www.w3.org/TR/SVG/propidx.html" \t "http://www.w3cplus.com/svg/_blank)。 |
|  |  |

curve指令 path d=""

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 简要说明 | 标准的指令字母是10个，外加1个非标准的;  这个可以参见我翻译构建的[Snap.svg项目](https://github.com/zhangxinxu/demo-Snap.svg)中[Paper.path()](http://www.zhangxinxu.com/GitHub/demo-Snap.svg/demo/basic/Paper.path.php)页面中的表格：   |  |  |  | | --- | --- | --- | | 命令 | 名称 | 参数 | | M | moveto  移动到 | (x y)+ | | Z | closepath  关闭路径 | (none) | | L | lineto  画线到 | (x y)+ | | H | horizontal lineto  水平线到 | x+ | | V | vertical lineto  垂直线到 | y+ | | C | curveto  三次贝塞尔曲线到 | (x1 y1 x2 y2 x y)+ | | S | smooth curveto  光滑三次贝塞尔曲线到 | (x2 y2 x y)+ | | Q | quadratic Bézier curveto  二次贝塞尔曲线到 | (x1 y1 x y)+ | | T | smooth quadratic Bézier curveto  光滑二次贝塞尔曲线到 | (x y)+ | | A | elliptical arc  椭圆弧 | (rx ry x-axis-rotation large-arc-flag sweep-flag x y)+ | | R | [Catmull-Rom curveto](http://en.wikipedia.org/wiki/Catmull%E2%80%93Rom_spline" \l "Catmull.E2.80.93Rom_spline)\*  Catmull-Rom曲线 | x1 y1 (x y)+ | |
|  | 其中，Catmull-Rom曲线不是标准的SVG命令，我们这里不予以讨论!  如果指令字母是大写的，例如M, 则表示坐标位置是绝对位置；如果指令字母小写的，例如m, 则表示坐标位置是相对位置。  弧形命令A虽然曲线更规律，但其由于不确定性（指定椭圆长半径和短半径，以及上面的两个点的圆弧，可上半弧也可下半弧），因此，还需要额外的参数保证其弧线唯一，所以其比贝塞尔曲线看上去要复杂，不过大家不要怕。  SVG中path的元素，也就是路径绘制，属性名称是d, 具体值是由专门的“指令字母+坐标值”实现的，例如下面这个简单代码示意：  <path d="M10 10 L90 90" stroke="#000000" style="stroke-width: 5px;"></path> |
| help | <**style**>  **svg**{  border: 1px solid red;  }  **path**{  fill: white;  stroke: black;  stroke-width: 2;  } </**style**> |

直线

|  |  |
| --- | --- |
| 绘制直线 | [五种不同的直线指令](https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths" \t "http://www.w3cplus.com/svg/_blank)，你可以使用它们来创建路径:  moveto(M 或 m):移动到新的位置  lineto(L 或 l):从当前坐标画一条直线到一个新坐标  horizontal lineto(H 或 h):画一条水平线到新坐标  vertical lineto(V 或 v):画一条垂直线到新坐标  closepath(Z 或 z):关闭当前路径 |
| <**svg** width="600" height="400">  **<!--使用相对-->** <**path** d="M 50 50 l 0 300 l 200 0 l 0 -300 l -200 0"/> </**svg**>  该路径首先移动到一个起点上，然后垂直向下绘制一条直线，再一条往右，一条往上，最后一条往左，回到起点结束路径。    我们只使用了moveto和lineto指令来创建矩形的轮廓。现在我们把其它的三条指令也加入进来： |
| 简化 | <**path** d="M 50 50 l 0 300 l 200 0 l 0 -300 Z"/>  用一个closepath指令(Z)替换了最后一个lineto指令(l)。因为没有指定坐标，所以使用 Z 和 z 效果是一样的;  正如你所看到的，它和前面的示例效果完全一样，但是closepath使得我们可以不需要指定最后一组坐标。 |
| 简化+ | <**path** d="M 50 50 v 300 h 200 v -300 Z"/>  使用了vertical lineto (v) 和hosizontal lineto (h) 指令替换了中间的三个lineto指令。这两个指令只需要x和y两个坐标值中的一个，另一个值它自己会确定为0。  虽然它还是有很多坐标点，但已经比原来简洁了。当然你不会总是在绘制[水平线或者垂直线](http://www.vanseodesign.com/web-design/visual-grammar-lines/" \t "http://www.w3cplus.com/svg/_blank)这两种线条，但是当你需要绘制这两种线条时，比起普通的lineto指令，你可能最好还是选择vertical lineto (v) 和hosizontal lineto (h) 这两个指令。 |
| 书写建议 | 1. 样式、空格和逗号   对于使用空格，你应该有一些调节措施。你可以通过移动或者添加空格，使得各个指令以及整个路径的数据阅读起来更简明。  <svg width="600" height="400">  <path d="M50 50 v300 h200 v-300 Z"/>  </svg>  在这里，我去掉了一些指令和它们的第一个值之间的空格。我还在最后一个坐标之后、下一条指令之前添加了额外的空格。  虽然逗号不是必要的，你也可以在每条指令后边使用他们来分隔x和y坐标。你不能在指令之间使用逗号，但是你可以在坐标之间使用。  <svg width="600" height="400">  <path d="M50,50 v300 h200 v-300 Z"/>  </svg>  我在路径起点的x和y值之间添加了一个逗号。我可以保留它们之间的空格，但是考虑到坐标之间没有空格的话更容易阅读。在这方面你可以灵活一些。  如果它和前面的指令是一样的，你也可以跳过这条指令。下面是第一个示例中使用了lineto指令的路径，lineto指令一条接一条，所以我把实际的指令删除了。希望这些间隔和逗号可以使得不同组的坐标更容易查看。  <svg width="600" height="400">  <path d="M50,50 l 0,300 200,0 0,-300 Z"/></svg>  这是有效的，但是我发现它阅读起来更混乱了。我比较喜欢坐标一个一个分开来的指令，但是对于你来说，怎样阅读起来才最简明，是你自己决定的。  <svg width="600" height="400">  <path d="M50 50  v300  h200  v-300  Z"/>  </svg>  这可能是最容易阅读的，但是很明显坐标占据了大部分的空间。而且有一点大家要记住，在WordPress中内联SVG的唯一解决方案是，所有的代码都要写在一行上。 |

曲线

|  |  |
| --- | --- |
| 目录 | 曲线指令一共有三组，和直线指令一样，指令字母大写表示坐标位置是绝对坐标，指令字母小写表示坐标位置是相对坐标。  三次贝塞尔曲线指令 (C, c, S, s)  二次贝塞尔曲线指令 (Q, q, T, t)  椭圆弧线 (A, a) |
|  |  |
| 简要说明 | Canvas, 以及CSS3动画函数也有贝塞尔曲线的，但他们的[用法形式上与SVG是不同的](http://www.zhangxinxu.com/wordpress/?p=3626)（访问该地址可大致感受其异同）。 |

二次

|  |  |
| --- | --- |
| 二次 |  |
| <**svg** id="svg" width="500" height="100">  <**path** d="M0 0 Q50 50 100 0" stroke="#000000" fill="none" style="stroke-width: 2px;"></**path**> </**svg**> |
| 二次+ QT |  |
| <**svg** id="svg" width="500" height="100">  <**path** d="M0 50 Q50 0 100 50 T200 50" stroke="#000000" fill="none" style="stroke-width: 3px;"></**path**> </**svg**> |

三次

|  |  |
| --- | --- |
| 三次 |  |
|  | <svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org/2000/svg">  <path d="M10 10 C 20 20, 40 20, 50 10" style="fill:white;stroke:red;stroke-width:2"/> </svg>  M后面的起点，加C后面3个点，构成了贝赛尔曲线的4个点。  三种写法：  <**path d="M20,20 C90,40 130,40 180,20"**></**path**> <**path d="M20 20 C90 40, 130 40, 180 20"**></**path**> <**path d="M20 20 C90 40 130 40 180 20"**></**path**> |
| 三次+ CS |  |
|  | <**style**>  **svg**{  border: 1px solid red;  }  **path**{  fill: none;  stroke: red;  stroke-width: 2;  }  .desLine{  stroke: blue;  stroke-width: 1;  }  .assistLine{  stroke: black;  stroke-width: 3;  } </**style**>  <**svg** id="svg" width="500" height="200">  <**desc**>三次贝塞尔平滑曲线</**desc**>  <**defs**></**defs**>  <**path** d="M 0 50  C25 0 75 0 100 50  S177 100 200 50" ></**path**>  <**path** d="M 0 0 v200 h25 v-200 h25 v200 h25 v-200 h25 v200 h25 v-200 h25 v200 h25 v-200 h25 v200 h25 v-200 h25 v200 h25 v-200 h25 v200 h25 v-200 h25 v200 h25 v-200 h25 v200 h25 v-200 h25 v200 h25 v-200 h25 v200" class="desLine"></**path**>  <**path** d="M 0 50 h300" class="desLine"></**path**>  <**path** d="M 0 0 h300" class="desLine"></**path**>  <**path** d="M 0 100 h300" class="desLine"></**path**>  <**path** d="M0,50 l25,-50 M 75 0 l 25 50 M 175 100 L 200 50" class="assistLine"></**path**> </**svg**> |
| 二次+ 三次+ | <**svg** id="svg" width="300" height="100">  <**desc**>二次贝塞尔平滑曲线</**desc**><**defs**></**defs**>  <**path** d="M20 10 Q140 40 180 20 T280 30"></**path**>  <**text** x="120" y="60">小蛮腰</**text**>  <**text** x="200" y="40">小翘臀</**text**> </**svg**>   <**svg** id="svg" width="500" height="100">  <**desc**>三次贝塞尔平滑曲线</**desc**>  <**defs**></**defs**>  <**path** d="M20 20 C90 40 130 40 180 20 S250 60 280 20"></**path**>  <**text** x="90" y="60">A杯罩</**text**>  <**text** x="230" y="60">赘肉小肚子</**text**> </**svg**> |
| 各种三次 | <svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org/2000/svg">  <path d="M10 10 C 20 20, 40 20, 50 10" style="fill:white;stroke:red;stroke-width:2"/>  <path d="M70 10 C 70 20, 120 20, 120 10" style="fill:white;stroke:red;stroke-width:2"/>  <path d="M130 10 C 120 20, 180 20, 170 10" style="fill:white;stroke:red;stroke-width:2"/>  <path d="M10 60 C 20 80, 40 80, 50 60" style="fill:white;stroke:red;stroke-width:2"/>  <path d="M70 60 C 70 80, 110 80, 110 60" style="fill:white;stroke:red;stroke-width:2"/>  <path d="M130 60 C 120 80, 180 80, 170 60" style="fill:white;stroke:red;stroke-width:2"/>  <path d="M10 110 C 20 140, 40 140, 50 110" style="fill:white;stroke:red;stroke-width:2"/>  <path d="M70 110 C 70 140, 110 140, 110 110" style="fill:white;stroke:red;stroke-width:2"/>  <path d="M130 110 C 120 140, 180 140, 170 110" style="fill:white;stroke:red;stroke-width:2"/> </svg> |

三次与三次+的比较

|  |  |
| --- | --- |
| CC | <**svg** width="800" height="500">  <**path** d="M100,200 C100,100 400,100 400,200 c100,200 400,100 300,0"/> </**svg**> |
| 添加了第二条贝塞尔曲线来连接第一条。你可以看到他们之间的衔接不是很平滑。  通常，在连接曲线时，第二条曲线的起点控制点是第一条曲线终点控制点的对称点。  幸运的是，你不必弄清楚如何计算控制点的反射。S 和 s 指令是让浏览器帮你计算对称点的快捷方式，使用(S s)指令创建对称的控制点很容易。  在这里我用了一个相对定位的指令(s)快捷地创建了第二条曲线，来匹配使用相对坐标的曲线。你可以看到两条曲线连接的交点处相比之前平滑了很多。 |
| CS | <**svg** width="800" height="500">  <**path** d="M100,200  C100,100 400,100 400,200  s100,100 300,0"/> </**svg**> |

椭圆

|  |  |  |
| --- | --- | --- |
| 椭圆A | 椭圆弧线(A 或 a)。下面这个弧线指令绘制了一段椭圆，它是曲线指令里边需要最多参数的，格式如下：  A rx ry x-axis-rotation large-arc-flag sweep-flag x y  --------------------------------------------------------  rx 是弧线在x轴方向的半径  ry 是弧线在y轴方向的半径  x-axis-rotation 是与x轴夹角的度数，是此段弧所在的x轴与水平方向的夹角，即x轴的逆时针旋转角度，负数代表顺时针转动的角度。 large-arc-flag ：值为0表示使用较小的弧线，值为1表示使用较大的弧线。  sweep-flag：使用弧线(值为0)，使用它的[轴对称](http://www.vanseodesign.com/web-design/symmetry-asymmetry/" \t "http://www.w3cplus.com/svg/_blank)弧线(值为1)，  1代表从起点到终点弧线绕中心顺时针方向，0 代表逆时针方向。 x,y 是弧终端坐标。 | |
| x-axis-rotatio | x-axis-rotation代表旋转的角度，体会下面例子中圆弧的不同：  <svg width="320px" height="320px"> <path d="M10 315 L 110 215 A 30 50 0 0 1 162.55 162.45 L 172.55 152.45 A 30 50 -45 0 1 215.1 109.9 L 315 10" stroke="black" fill="green" stroke-width="2" fill-opacity="0.5"/> </svg>    从图中可以看到椭圆旋转参数的不同导致绘制的圆弧方向不同，当然这个参数对正圆来说没有影响。 | |
| flag flag | 椭圆弧绘制出的弧线是椭圆的一个片段。[若rx和ry的值相同，则绘制出圆](http://www.vanseodesign.com/css/shapes-module/" \t "http://www.w3cplus.com/svg/_blank)。  给定起点和终点坐标，以及rx ry x-axis-rotation的值，绘制出的弧线有四种可能。  large-arc-flag和sweep-flag控制了圆弧的大小和走向。   |  |  |  | | --- | --- | --- | |  | 取大/小 | 圆心指向 | | 0 | 小 | 顺时针 | | 1 | 大 | 逆时针 | | |
| <path d="M250,100 A120,80 0 0,0 250,200"  fill="none" stroke="red" stroke-width="5"/> <path d="M250,100 A120,80 0 1,1 250,200"  fill="none" stroke="green" stroke-width="5"/> <path d="M250,100 A120,80 0 1,0 250,200"  fill="none" stroke="purple" stroke-width="5"/> <path d="M250,100 A120,80 0 0,1 250,200"  fill="none" stroke="blue" stroke-width="5"/> |  |
| <svg width="325px" height="325px"> <path d="M80 80 A 45 45, 0, 0, 0, 125 125 L 125 80 Z" fill="green"/> <path d="M230 80 A 45 45, 0, 1, 0, 275 125 L 275 80 Z" fill="red"/> <path d="M80 230 A 45 45, 0, 0, 1, 125 275 L 125 230 Z" fill="purple"/> <path d="M230 230 A 45 45, 0, 1, 1, 275 275 L 275 230 Z" fill="blue"/> </svg> |  |
| 路径画一个圆 | <**style**>  .circlePath{  stroke-linecap:round;  stroke-width:20;  stroke:red;  } </**style**> <**path** class='circlePath' d="m 150 160 a 100 100 0 1 1 150 159" /> <**path** class='circlePath' d="m 140 170 a 100 100 0 1 0 150 159" /> | |

其他

text

|  |  |  |
| --- | --- | --- |
|  | Text-svg元素详解  http://www.360doc.com/content/07/0906/21/39836\_724448.shtml | |
|  | html5中的svg包含的文本text上下左右居中的方法  http://blog.mn886.net/chenjianhua/show/2c2c9729ecec484da77e4d9c29714cf1/index.html | |
| 简单展示 | | <**style**>  **text**{  font-family: **"Microsoft YaHei"**, **"微软雅黑"**, **"Segoe UI"**, **"Lucida Grande"**, Helvetica, Arial, sans-serif, FreeSans, Arimo;  color: lightgreen;**/\*无效\*/** font-size: 30px;  } </**style**>  <**text** x="0" y="30" fill="red">I love SVG</**text**>  cursor="pointer" | |
| 分类 | | <!--简单文字--> <**text x="0" y="20" fill="red"**>I love SVG</**text**>  <!--变换：旋转--><!-- （旋转多少度，旋转点）--> <**text x="0" y="20" fill="red" transform="rotate(45 0,200)"**>I love SVG</**text**> <**path d="M 0 200 l 200 -200" fill="none" stroke-width="1" stroke="blue"**></**path**> <**path d="M 1 200 v-200" fill="none" stroke-width="1" stroke="blue"**></**path**>  <!--路径上的文字：--> <**defs**>  <**path id="path1" d="M0,222 a1,1 0 0,0 100,0"**/> </**defs**> <**text x="10" y="0" style="fill:red;"**><!-- x:x方向起始位置;y:无效-->  <**textPath xlink:href="#path1"**>I love SVG I love SVG I love SVG I love SVG </**textPath**> </**text**>   <!--元素可以安排任何分小组与<tspan> 元素的数量。每个<tspan> 元素可以包含不同的格式和位置。几行文本(与 <tspan> 元素):--> <**text x="0" y="100" style="fill:red;"**>  Several lines:  <**tspan x="20" y="125"**>First line</**tspan**>  <**tspan x="20" y="150"**>Second line</**tspan**> </**text**>  <!--作为链接文本（ <a> 元素）：-->   <**a xlink:href="http://www.w3schools.com/svg/" target="\_blank"**>  <**text x="0" y="200" fill="red"**>I love SVG</**text**> </**a**> | |

svg样式控制的优先级

|  |  |
| --- | --- |
| 外部style中的样式优先级高于行内 | <**style**>  **svg**{  border: 1px solid red;  }  **path**{  fill: none;  stroke: red;  stroke-width: 2;  }  <**svg** width="1000" height="600" id="svgId">  <**path** stroke-linecap="square" stroke-width="20" stroke="red" d="m150 160 a 100 100 0 0 1 150 159" /> |
| 外部.class级别高于标签 | <**style**>  **path**{  fill: none;  stroke: red;  stroke-width: 2;  }  .circlePath{  stroke-linecap:square;  stroke-width:20;  stroke:red;  } </**style**> <**path** class='circlePath' d="m150 160 a 100 100 0 0 1 150 159" /> |

|  |  |
| --- | --- |
|  | 打破盒子模式的限制，使用Clip-Path创建响应式图形  <http://www.w3cplus.com/css3/creating-responsive-shapes-with-clip-path.html>  CSS和SVG中的剪切——clip-path属性和<clipPath>元素  <http://www.w3cplus.com/css3/css-svg-clipping.html> |
|  |  |
|  | CSS中的剪切——clip-path属性  clip-path属性是[CSS Masking模块](http://www.w3.org/TR/2014/WD-css-masking-1-20140213/" \t "http://www.w3cplus.com/css3/_blank)的一部分。自从2000年以来，剪切都只是SVG中的一部分，现在将这个功能引入到CSS的Msking模块中，所以现在可以对HTML元素和SVG元素进行剪切。  clip-path属性是指定一个应用到元素上的剪切路径。应用在SVG中<clipPath>元素上的属性值可以完全运用在clip-path的属性上。你还可以使用CSS Shapes模块中的[基本形状](http://dev.w3.org/csswg/css-shapes-2/" \l "ltbasic-shapegt" \t "http://www.w3cplus.com/css3/_blank)来定义剪切路径。这些形状你可以使用形状函数来创建。这些形状态函数包括polygon()、circle()、inset()(用来定义嵌入的矩形)和ellipse()。  SVG中的剪切——<clipPath>元素  在SVG中使用<clipPath>元素来定义剪切元素的剪切路径。如果不想使用CSS的clip-path来定义元素的剪切路径，可以使用SVG中的clip-path属性。 |
| demo1 | 使用clip-path属性将一个剪切路径运用在一个元素上非常的简单：  /\* SVG中的clipPath的使用 \*/  .element {  clip-path: url(#svgClipPathID);  }  /\* 使用CSS中的基本图形函数 \*/  .element {  clip-path: polygon(...); /\* 或者其他的图形函数 \*/  }  例如，我们使用polygon()函数定义一个多边形的剪切路径，并且把这个路径应用到一个图像上: |
| <**style**>  .clip-svg {  width: 0;  height: 0;  }  .clip-polygon {  -webkit-clip-path: polygon(25% 0%, 75% 0%, 100% 50%, 75% 100%, 25% 100%, 0% 50%);  clip-path: polygon(25% 0%, 75% 0%, 100% 50%, 75% 100%, 25% 100%, 0% 50%);  -webkit-clip-path: url(**"#clip-svg-demo"**);  clip-path: url(**"#clip-svg-demo"**);**/\*本人浏览器真正起作用\*/** } </**style**> <**svg** class="clip-svg">  <**defs**>  <**clipPath** id="clip-svg-demo" clipPathUnits="objectBoundingBox">  <**polygon** points="0.25 0, 0.75 0, 1 0.5, 0.75 1, 0.25 1, 0 0.5"/>  </**clipPath**>  </**defs**> </**svg**> <**img** src="images/1.jpg" class="clip-polygon"> |
| 基本图形函数允许我们创建一定数量的图形，其中最复杂的就是多边形。如果你想创建一个更为复杂的图形，而且图形看起来不是用直线画出来的，这个时候你就需要使用SVG的<clipPath>元素。正如<clipPath>元素名称所暗示的一样，你可以使用这个元素绘制任意路径的图形。这也意味着，你可以使用<clipPath>元素绘制出任意图形来做为一个剪切路径。 |
| demo2 | 使用SVG的clipPath定义一个路径，将定义好的路径通过clip-path运用到图片上: |
| <**svg** height="0" width="0">  <**defs**>  <**clipPath** id="svgPath">  <**path** fill="#FFFFFF" stroke="#000000" stroke-width="1.5794" stroke-miterlimit="10"  d="M215,100.3c97.8-32.6,90.5-71.9,336-77.6 c92.4-2.1,98.1,81.6,121.8,116.4c101.7,149.9,53.5,155.9,14.7,178c-96.4,54.9,5.4,269-257,115.1c-57-33.5-203,46.3-263.7,20.1 c-33.5-14.5-132.5-45.5-95-111.1C125.9,246.6,98.6,139.1,215,100.3z"></**path**>  </**clipPath**>  </**defs**> </**svg**> <**style**>  **img** {  -webkit-clip-path: url(#svgPath);  clip-path: url(#svgPath);  } </**style**> <**img** src="http://www.w3cplus.com/sites/default/files/blogs/2014/1407/flowers.jpg" alt="" /> |
| demo3 | 使用文本做为剪切路径：  <svg height="0" width="0">  <defs>  <clipPath id="svgTextPath">  <text x="0" y="300" textLength="800px" lengthAdjust="spacing" font-family="Vollkorn" font-size="230px" font-weight="700" font-style="italic"> Blossom </text>  </clipPath>  </defs>  </svg>  SVG中的<text>最酷的特点就是可以使用自定义字体，就像HTML文本。在我们的示例中使用了Google Web Fonts中的[Vollkorn](http://www.google.com/fonts/specimen/Vollkorn" \t "http://www.w3cplus.com/css3/_blank)字体。使用textLength属性，将文本的宽度设置的和图片宽度一样， 并且通过x和y来定位文本。注意，x和y坐标确认了文本左下角的位置（也就是文本的基线baseline）。  使用上面的文本路径剪切图来的图片效果如下： |
|  | 使用了多个<circle>形状，他们大小不同，位置不同。  <svg height="0" width="0">  <defs>  <clipPath id="svgPath">  <circle stroke="#000000" stroke-miterlimit="10" cx="50" cy="50" r="40" />  <circle stroke="#000000" stroke-miterlimit="10" cx="193.949" cy="235" r="74.576"/>  <circle stroke="#000000" stroke-miterlimit="10" cx="426.576" cy="108.305" r="47.034"/>  <circle stroke="#000000" stroke-miterlimit="10" cx="346.915" cy="255.763" r="43.644"/>  <circle stroke="#000000" stroke-miterlimit="10" cx="255.39" cy="82.882" r="35.17"/>  <!-- more circles... -->  </clipPath>  </defs>  </svg>  此时，图像只会在圆中显示，圆外就不会显示： |
|  | 正如我们这篇文章中介绍的，你可以使用clip-path属性应用在SVG元素绘制的路径。在上面演示的示例，剪切路径都是应用在HTML中的<img>元素上。  在接下来的示例中，将演示的是一个剪切路径用于<svg>的根元素。同样是樱花图片，使用下面SVG的中的<image>来引用。 |
| demo | SVG中的<image>元素用来引用一个完整的SVG或像素图像。如果你在<image>中引用的SVG图像，设置的width和height属性，将会用来设置SVG视窗的大小。如果你引用的是像素图像（我们这里的例子就是这样做的），图像将会自动缩放到指定的width和height。所以我们要确认好他们的长宽比例，避免图像扭曲。  当你创建一个SVG时，你可以在<svg>元素上指定其宽度和高度的大小，用于创建一个窗口。任何超过这个窗口的内容都将不会显示出来。你可以通过<clipPath>元素定制一个窗口。  使用clip-path将定义好的<clipPath>运用到<svg>元素上： |
| <**style**>  **svg** {  -webkit-clip-path: url(#theSVGPath);  clip-path: url(#theSVGPath);  } </**style**> <**svg** height="500" width="800">  <**image xlink**:href="http://www.w3cplus.com/sites/default/files/blogs/2014/1407/flowers.jpg" x="0" y="0" width="800" height="500"/>  <**defs**>  <**clipPath** id="theSVGPath">  <**rect** x="0" y="0" stroke="#000000" stroke-miterlimit="10" width="108" height="500"/>  <**rect** x="121.5" y="25.5" stroke="#000000" stroke-miterlimit="10" width="55" height="455"/>  <**rect** x="192.5" y="9.5" stroke="#000000" stroke-miterlimit="10" width="60" height="484"/>  <**rect** x="271.5" y="44.5" stroke="#000000" stroke-miterlimit="10" width="63" height="416"/>  <**rect** x="349.5" y="25.5" stroke="#000000" stroke-miterlimit="10" width="208" height="447"/>  <**rect** x="574.5" y="44.5" stroke="#000000" stroke-miterlimit="10" width="60" height="446"/>  <**rect** x="644.5" y="9.5" stroke="#000000" stroke-miterlimit="10" width="68" height="471"/>  <**rect** x="736.5" y="18.5" stroke="#000000" stroke-miterlimit="10" width="49" height="462"/>  </**clipPath**>  </**defs**> </**svg**> |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| demo-图片的圆角显示(pattern、clip-path) | <**svg** width="100" height="100">  <**desc**>SVG圆角效果</**desc**>  <**defs**>  <**pattern** id="raduisImage" patternUnits="userSpaceOnUse" width="100" height="100">  <**image xlink**:href="images/1.jpg" x="0" y="0" width="100" height="100" />  </**pattern**>  </**defs**>  <**circle** cx="10" cy="50" r="50" fill="url(#raduisImage)"></**circle**>  <**circle** cx="50" cy="50" r="50" fill="url(#raduisImage)"></**circle**>  <**circle** cx="90" cy="50" r="50" fill="url(#raduisImage)"></**circle**> </**svg**>  <**svg** width="100" height="100">  <**defs**>  <**clipPath** id="svgPath" clipPathUnits="objectBoundingBox">  <**circle** stroke="#000000" stroke-miterlimit="10" cx=".5" cy=".5" r=".5"/>  </**clipPath**>  </**defs**>  <**image** x="0" y="0" width="100" height="100" preserveAspectRatio="none"  **xlink**:href="images/1.jpg"  clip-path="url(#svgPath)" /> </**svg**> |
|  |
|  |  |
|  |  |

使用SVG中的Symbol元素制作Icon（gulp）

|  |  |
| --- | --- |
|  | https://isux.tencent.com/16292.html |
|  | F:\HTML\practice\svg\svgicon  >gulp webserver  http://localhost:8080/ |
|  | 未来必热：SVG Sprite技术介绍  <http://www.zhangxinxu.com/wordpress/2014/07/introduce-svg-sprite-technology/>  symbol + use => SVG Sprite |
|  | 使用SVG symbols建立图标系统  <http://www.w3cplus.com/svg/how-to-create-an-icon-system-using-svg-symbols.html>  F:\HTML\practice\svg\svgicon\syl test\test2.html |

SVG基础 | SVG defs元素、symbol元素和use元素

|  |  |
| --- | --- |
| defs  g | SVG的<defs>元素用于预定义一个元素使其能够在SVG图像中重复使用。例如你可以将一些图形制作为一个组，并用<defs>元素来定义它，然后你就可以在SVG图像中将它当做简单图形来重复使用。  在<defs>元素中定义的图形不会直接显示在SVG图像上。要显示它们需要使用<use>元素来引入它们。如下面的代码所示：  <svg xmlns="http://www.w3.org/2000/svg">   <defs>  <g id="shape">  <rect x="50" y="50" width="50" height="50"/>  <circle cx="50" cy="50" r="50"/>  </g>  </defs>  <use xlink:href="#shape" x="50" y="50"/>  <use xlink:href="#shape" x="200" y="50"/> </svg>    要引用<g>元素，必须在<g>元素上设置一个ID，通过ID来引用它。<use>元素通过xlink:href属性来引入<g>元素。注意在ID前面要添加一个#。  在<use>元素中，通过x和y属性来指定重用图形的显示位置。注意在<g>元素中的图形的定位点都是0,0，在使用<use>元素来引用它的时候，它的定位点被转换为<use>元素x和y属性指定的位置。  上面SVG图像中绿色的圆点并不是示例代码的一部分。它们是用来显示2个<use>元素的x和y坐标的。  可以将下面的元素放入到<defs>元素中使用：  ===任何图形元素，如：rect，line等  ===g  ===symbol |
| symbol | SVG <symbol>元素用于定义可重复使用的符号。嵌入在<symbol>元素中的图形是不会被直接显示的，除非你使用<use>元素来引用它。  下面是一个使用<symbol>元素的简单例子：   |  | | --- | | <svg xmlns="http://www.w3.org/2000/svg" width="500" height="100">  <symbol id="shape2">  <circle cx="25" cy="25" r="25" style="fill:#bf55ec;"/>  </symbol>  <use xlink:href="#shape2" x="50" y="25"/> </svg> | |  |   <symbol>元素需要一个ID号，以便以被<use>元素引用。  **一个<symbol>元素可以有preserveAspectRatio和viewBox属性。而<g>元素不能拥有这些属性。因此相比于在<defs>元素中使用<g>的方式来复用图形，使用<symbol>元素也许是一个更好的选择。** |
| use | SVG <use>元素可以在SVG图像中多次重用一个预定义的SVG图形，包括<g>元素和<symbol>元素。  被重用的图形可以在定义<defs>的内部（图形将不可见直到使用use来引用它）或外部。 |
| 在<defs>元素中定义的<g>元素：要引用<g>元素，必须给它一个ID号，通过ID号来引用它。   |  | | --- | | <svg>  <defs>  <g id="shape">  <rect x="50" y="50" width="50" height="50"/>  <circle cx="50" cy="50" r="50"/>  </g>  </defs>  <use xlink:href="#shape" x="50" y="50"/>  <use xlink:href="#shape" x="200" y="50"/> </svg> | |
| 我们也可以引用不在<defs>元素中的图形。<use>元素可以引用SCG图像中的任何元素，只要这个元素有一个唯一的ID号；   |  | | --- | | <svg width="500" height="110">  <g id="shape3">  <rect x="0" y="0" width="50" height="50"/>  </g>  <use xlink:href="#shape3" x="100" y="50" style="fill: #00ff00;"/>  <use xlink:href="#shape3" x="200" y="50" style="stroke: #00ff00; fill: none;"/> </svg> | |  | | 在<g>元素中定义了一个<rect>元素。然后通过<use>元素来引用这个<g>元素。因为没有定义在<defs>或<symbol>元素中，所以它是可见的。  为引用的图形设置CSS样式（<g></g>标签中的原始图形没有设置样式，它将是默认的样式（黑色））。 | |
| 可以引用svg文件中定义的模块id：  <svg class="icon">  <use xlink:href="/assets/svg-symbols.svg#film"></use> </svg> |

symbol---建立图标系统

|  |  |  |
| --- | --- | --- |
|  | symbol结合viewBox的简单使用，viewBox代表模块的总视口，模块中内容的大小决定能观察到的东西；  一般作为图标按钮的背景；  use进行等比例缩放处理，象征着用户的显示器是小屏幕还是大屏幕，但是在大小屏幕上看到的内容是一样的；  viewBox进行截断性处理，决定用户能看到总体内容中的多少，或者总体内容的分布情况 | |
| <**style** type="text/css">  **svg**{  border: 1px solid red;  }  **use**.tip-1 {  fill: #4BC0A5;  color: #A4DFD1;  } </**style**> <**svg** style="display: none;">  <**symbol** id="tip" viewBox="0 0 30 30">  <**rect** x="0" y="0" width="25" height="25"></**rect**>  </**symbol**> </**svg**> <**svg** height="100px" width="100px">  <**use xlink**:href="#tip" class="tip-1"/> </**svg**> | |
|  | |
| demo2 | 不同使用情况下的对比:当显示器长宽比与原图存在差异时，保证不改变原画的长宽比的情况下完整展示； | |
| <**svg** display="none">  <**symbol** viewBox="0 0 24 24" id="heart">  <**path** fill="#E86C60" d="M17,0c-1.9,0-3.7,0.8-5,2.1C10.7,0.8,8.9,0,7,0C3.1,0,0,3.1,0,7c0,6.4,10.9,15.4,11.4,15.8 c0.2,0.2,0.4,0.2,0.6,0.2s0.4-0.1,0.6-0.2C13.1,22.4,24,13.4,24,7C24,3.1,20.9,0,17,0z"></**path**>  </**symbol**> </**svg**> | |
| **<!--svg的默认大小：width="300" height="150" ???-->** <**svg**>  <**use xlink**:href="#heart"/> </**svg**> |  |
| <**svg** width="300" height="150">  <**use xlink**:href="#heart"/> </**svg**> |  |
| <**svg** width="30" height="150">  <**use xlink**:href="#heart"/> </**svg**> |  |
| <**svg** width="300" height="30">  <**use xlink**:href="#heart"/> </**svg**> |  |
| demo2+ | 怎么给SVG添加样式呢？给SVG <use>元素添加样式需要一点技巧。这是因为SVG 图标引用这种方式有自己单独的DOM结构（也就是shadow DOM），CSS选择器并不能获取到，所以假设我们有如下的图标：  <svg class="icon">  <use xlink:href="img/myicons.svg#heart"/>  </svg>  然后这样添加样式：  .icon path {  fill: #000000;  }  是没有办法工作的。  如何解决这个问题呢？比如说你想要改变图标的填充颜色。  首先，确保fill属性不是svg文件中内联定义的。  所以，如果图标 sprite如下：  <svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">  <symbol viewBox="0 0 24 24" id="heart">  <title>Heart</title>  <path **fill="#E86C60"** d="M17,0c-1.9,0-3.7,0.8-5,2.1C10.7,0.8,8.9,0,7,0C3.1,0,0,3.1,0,7c0,6.4,10.9,15.4,11.4,15.8 c0.2,0.2,0.4,0.2,0.6,0.2s0.4-0.1,0.6-0.2C13.1,22.4,24,13.4,24,7C24,3.1,20.9,0,17,0z"></path>  </symbol>  <symbol viewBox="0 0 32 32" id="arrow">  <title>Arrow</title>  <path **fill="#0f0f0f"** d="M16,0C7.2,0,0,7.2,0,16s7.2,16,16,16s16-7.2,16-16S24.8,0,16,0z M22.8,13.6l-6,8C16.6,21.9,16.3,22,16,22 s-0.6-0.1-0.8-0.4l-6-8c-0.2-0.3-0.3-0.7-0.1-1S9.6,12,10,12h12c0.4,0,0.7,0.2,0.9,0.6S23,13.3,22.8,13.6z"></path>  </symbol>  </svg>  把内联fill属性删除；  然后在你的css代码中，加入：  .icon {  fill: #00000; /\* this will be your icons default color \*/  }  因为你没有在SVG中给path元素指定fill，所以它们会继承父级元素，即SVG的fill属性，这样你可以直接使用CSS选择器修改这个属性。  现在，如果我们只想改变一个图标的fill，怎么办呢？我们给这个图标再单独指定一个class即可：  <svg class="icon my-class-name">  <use xlink:href="img/myicons.svg#heart"></use>  <svg>  然后使用CSS改变它的fill值：  .my-class-name {  fill: red;  } | |
| demo3 | symbol viewBox定义，通过use使用不同的样式进行展示  （可以引用同HTML文件中不同svg标签中定义的模块id） | |
| <**style** type="text/css">  **svg**{  border: 1px solid red;  }  **path** {  fill: inherit;**/\*重要：与path中的 fill="" 相互作用；默认情况下位黑色\*/** }  **use**.tip-1 {  fill: #4BC0A5;  color: #A4DFD1;  }  **use**.tip-2 {  fill: #0099CC;  color: #7FCBE5;  }  **use**.tip-3 {  fill: #5F5EC0;  color: #AEAFDD;  } </**style**>  <**svg** style="display: none;">  <**symbol** id="tip" viewBox="0 0 23 30">  <**path** fill="#aaa"  d="M22.63,18.261c-0.398-3.044-2.608-6.61-4.072-9.359c-1.74-3.271-3.492-5.994-5.089-8.62l0,0 c-1.599,2.623-3.75,6.117-5.487,9.385c0.391,0.718,0.495,1.011,0.889,1.816c0.143,0.294,0.535,1.111,0.696,1.43 c0.062-0.114,0.582-1.052,0.643-1.162c0.278-0.506,0.54-0.981,0.791-1.451c0.823-1.547,1.649-2.971,2.469-4.33 c0.817,1.359,1.646,2.783,2.468,4.33c0.249,0.47,0.513,0.946,0.791,1.453c1.203,2.187,2.698,4.906,2.96,6.895 c0.292,2.237-0.259,4.312-1.556,5.839c-1.171,1.376-2.824,2.179-4.663,2.263c-1.841-0.084-3.493-0.887-4.665-2.263 c-0.16-0.192-0.311-0.391-0.448-0.599c-0.543,0.221-1.127,0.346-1.735,0.365c-0.56-0.019-1.095-0.127-1.599-0.313 c1.448,3.406,4.667,5.66,8.447,5.78C19.086,29.537,23.469,24.645,22.63,18.261z"/>  <**path** fill="#ddd"  d="M6.177,11.659c0.212,0.367,0.424,0.747,0.635,1.136c0.164,0.303,0.333,0.606,0.512,0.927 c0.683,1.225,1.618,2.898,1.755,3.937c0.144,1.073-0.111,2.056-0.716,2.769c-0.543,0.641-1.315,1.014-2.186,1.067 c-0.87-0.054-1.643-0.43-2.186-1.067c-0.604-0.713-0.858-1.695-0.715-2.771c0.137-1.036,1.072-2.712,1.755-3.936 c0.18-0.32,0.349-0.623,0.513-0.927C5.752,12.404,5.964,12.026,6.177,11.659 M6.177,5.966L6.177,5.966 c-1.02,1.649-2.138,3.363-3.247,5.419c-0.932,1.728-2.344,3.967-2.598,5.88c-0.535,4.014,2.261,7.09,5.846,7.203 c3.583-0.113,6.379-3.189,5.845-7.203c-0.255-1.912-1.666-4.152-2.598-5.88C8.314,9.329,7.196,7.617,6.177,5.966L6.177,5.966z"/>  </**symbol**> </**svg**> <**svg** height="90px" width="69px">  <**use xlink**:href="#tip"/>  **<!--在style中path样式的作用下，显示黑色-->** </**svg**> <**svg** height="90px" width="69px">  <**use xlink**:href="#tip" class="tip-1"/> </**svg**> <**svg** height="90px" width="69px">  <**use xlink**:href="#tip" class="tip-2" x="30" y="30"/>  **<!--默认情况下-->** </**svg**> <**svg** height="90px" width="69px">  <**use xlink**:href="#tip" class="tip-3"/> </**svg**> | |
|  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |

snap.js

|  |  |
| --- | --- |
|  | snap.js  <http://www.zhangxinxu.com/wordpress/2014/01/snap-svg-study-demo-docs/> |
|  | <https://segmentfault.com/a/1190000006975681> |
|  |  |

基础

|  |  |
| --- | --- |
| append | Element.append(el)  当前元素内部后置给定元素。也就是el要塞到Element的里面，同时是最后面。  el 类型可以是Element或者Set. 表示要附加的元素。  ---将第二个svg中的某个元素，提取出来放到第一个svg中、 |
|  | var svg = Snap("#svgId"); var svg2 = Snap("#svgId2"); var c1=svg2.select("#c3"); svg.append(c1) |
| prepend | Element.prepend(el)  当前元素内部前置给定元素。也就是el要塞到Element的里面，同时是最前面。 |
|  |  |
| before | Element.before(el)  给定元素插入到当前元素的前面。也就是el到Element的前面。类似于z-index |
|  | var c1 = svg.paper.circle(50, 50, 40).attr({  fill: "#f00" // 红色  });  var c2 = svg.paper.circle(100, 50, 40).attr({  fill: "#00f" // 蓝色  });  c1.before(c2); |
| inserrAfter | c1.insertAfter(c2);  ---执行之后，c1在眼前，c2在眼后：1拉到上层【在视觉上，上层的覆盖下层的】 |
| insertBefore | c2.insertBefore(c1);  ---执行之后，c1在眼前，c2在眼后：2拉到下层 |
| aspx | Element.asPX(attr, [value])  以px值(非%, em等)返回元素给定的属性。  ---只能获取，不能设置 |
|  | svg.select("#c1").asPX("cx"); //100 |
| attr | Element.attr(...)  获得或设置元素给定的属性值。 |
|  | svg.selectAll("circle").forEach(function (n,i) {  n.attr("cx","0") })  n.attr({cx:0}); |
| clone | var c1 = svg.paper.circle(50, 50, 40);  c1.clone().attr({cx: 150});  ---会在svg中实时创建一个新的元素，在设置新的属性之前会表现与原元素一致 |
| data | var c = svg.paper.circle(50, 50, 40).attr({  fill: "#f00"  }).data("author", "zhangxinxu").click(function() {  svg.paper.text(10, 10, this.data("author"));  });  ---数据存储 |
| removeData | Element.removeData([key])  移除元素上存储的对应关键字数据。如果关键字未提供，则移除元素上存储的所有数据。 |
| click | svg.select(".c1").click(function() {  this.animate({  fill: "#00f" // 蓝色  }, 1000);  this.attr("fill","red");  }); |
| marker | Element.marker(x, y, width, height, refX, refY)  利用当前元素创建一个<marker>元素。为了创建这个标记，你需要指定边界矩阵以及参考点。  SVG <marker>元素被用来表示线或路径的开始、中间以及结束。例如，你可以使用一个圆或正方形标记作为路径的开始，然后一个三角标记作为路径的结束。 |
|  |  |
| 事件 | click  drag  hover |
|  |  |
|  |  |
| innerSVG | Element.innerSVG()  返回元素内容的SVG代码。近似于HTML中的innerHTML方法。  var svg = Snap("#svg");  svg.paper.circle(50, 50, 40);  console.log(svg.innerSVG());// 结果是：  // <desc>Created with Snap</desc><defs/><circle cx="50" cy="50" r="40"/> |
|  |  |
| inAnim | Element.inAnim()  返回可能操纵当前元素的一组动画。 |
| toString | Paper.toString()  返回元素的SVG代码，等同于HTML中的outerHTML方法。 |
|  | <svg id="svg" width="0" height="0"></svg>  var svg = Snap("#svg");  console.log(svg.paper.toString());// 结果是：  // <svg id="svg" width="0" height="0" xmlns="http://www.w3.org/2000/svg"><desc>Created with Snap</desc><defs></defs></svg> |
| toDefs |  |
|  | lement.pattern(x, y, width, height)  利用当前元素创建一个<pattern>元素。为了创建这个图案，你需要指定图案矩阵。  <pattern>容易让人困惑，但同时也很强大。在SVG中，<pattern>被用来填充图形，填充的图案可以来自SVG图片，或者位图图片。  SVG图案填充很像我们在photoshop中使用的图案填充，如"tiles"(拼贴)等。 |
|  |  |
| getBBox | Element.getBBox()  返回给定元素的边界框描述。 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

snap.

|  |  |
| --- | --- |
| getElementByPoint | Snap.getElementByPoint(x, y)  返回给定点最上面一个元素。 |
| parse  add | Snap.parse(svg)  解析SVG片段字符串为SVG片段。  var fragment = Snap.parse('<circle cx="50" cy="50" r="40"></circle>');  Snap("#svg").add(fragment); |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

snap.path.

|  |  |
| --- | --- |
|  |  |
| getSubpath | Element.getSubpath(from, to)  返回给定元素上指定起始结束长度上的子路径（仅适用于path元素）。  var svg = Snap("#svg");  var c = svg.paper.path("M10 10L90 90").attr({  stroke: "#000",  strokeWidth: 5  });  console.log(c.getSubpath(20, 80));  // 结果是：  // M24.139,24.139C35.874,35.874,52.807,52.807,66.565,66.565 |
| getTotalLength | Element.getTotalLength()  以像素为单位，返回路径的长度（仅适用于path元素）。  var svg = Snap("#svg");  var c = svg.paper.path("M10 10L90 90").attr({  stroke: "#000",  strokeWidth: 5  });  var c=svg.select(".path1");  console.log(c.getTotalLength());  // 结果是：  // 113.1370849609375 |
|  | console.log(Snap.path.getTotalLength("M10 10L90 90"));  // 结果是：  // 113.1370849609375 |
|  | 创建矩阵对象  var myMatrix = new Snap.Matrix();  myMatrix.scale(4,2);                 // 横纵缩放  myMatrix.translate(100,0);          // 平移  myMatrix.rotate(45);              // 旋转 |
|  |  |

d3.js

|  |  |
| --- | --- |
|  | http://www.sxt.cn/blogv2hander!showbloglistblogtype.action?uid=2839&blogSelfTypeid=7965&pageNum=4 |