[变化transform 1](#_Toc21108)

[www 1](#_Toc17604)

[简介 1](#_Toc15866)

[位移 1](#_Toc17815)

[旋转-简单 1](#_Toc27119)

[旋转-复杂1 2](#_Toc19597)

[旋转-复杂2 2](#_Toc29238)

[多次旋转 2](#_Toc9057)

[倾斜 3](#_Toc18779)

[简写调用链 3](#_Toc2861)

[动画animate 4](#_Toc18916)

[简介 4](#_Toc25321)

[兼容 4](#_Toc13132)

[5无素 4](#_Toc24318)

[公有属性详解 4](#_Toc30664)

[attributeName 4](#_Toc17012)

[attributeType 4](#_Toc10568)

[from to by values 4](#_Toc20090)

[begin, end 4](#_Toc1863)

[+++ svg动画的暂停 5](#_Toc869)

[dur 5](#_Toc31856)

[calcMode, keyTimes, keySplines 5](#_Toc26190)

[repeatCount, repeatDur 6](#_Toc4646)

[fill 6](#_Toc22776)

[accumulate, additive 6](#_Toc20834)

[restart 6](#_Toc15791)

[min, max 6](#_Toc916)

[其他遗漏参数 6](#_Toc12083)

[动画的暂停与播放 6](#_Toc7007)

[set 6](#_Toc5361)

[set 6](#_Toc3179)

[animate 6](#_Toc24090)

[基础属性 6](#_Toc21161)

[lg:平移 6](#_Toc16982)

[lg:呼吸+双重 7](#_Toc11443)

[lg:马儿入山影无踪 7](#_Toc7459)

[lg:相对变化之begin=id.end+1s 7](#_Toc27183)

[animateTransform 7](#_Toc24868)

[通过transform实现动画 7](#_Toc30312)

[animateMotion 7](#_Toc19177)

[沿着路径运动 7](#_Toc18540)

变化transform

|  |  |  |  |
| --- | --- | --- | --- |
| www | <http://www.w3cplus.com/svg/transforms-on-svg-elements.html> | | |
| 简介 | 同HTML元素一样，我们可以通过transform函数操作SVG元素。然而transform在SVG元素和HTML元素上的工作方式会有一些差别。  首先，[IE不支持SVG元素的CSS transform属性](https://connect.microsoft.com/IE/feedbackdetail/view/1046003/css-transforms-feature-detection-not-working-on-svg-elements)，但是如果只是应用一些2D变换，为了适配IE,我们可以使用[SVG的transform属性](https://developer.mozilla.org/en/docs/Web/SVG/Attribute/transform)。  SVG的transform属性中的所有函数的参数只能是纯数字，比如说，我们不能在translate函数中使用%单位（虽说在火狐浏览器中的CSS transform属性也不能使用—此处有链接—），rotate，skew角度只能使用deg单位，我们能在CSS transform属性中可以使用的所有其它单位在这里都不能使用。  注：火狐浏览器现在已经支持transform-origin上应用带有%的值，不过与chorme不同的是，火狐的%是相对于svg画布而不是元素自身。  而且蛋疼的是，JavaScript的特征检测会有问题（通过JavaScript读取在外部CSS文件中设置的transform属性会返回与其等价的矩阵），所以我们需要另外一种检测IE的方法，或者直接在HTML上书写transform属性。  SVG元素和HTML元素工作方式的差异，主要是由元素坐标系的不同造成的。无论是HTML元素还是SVG元素，都有一个自己的坐标系。对于HTML元素，初始的坐标原点在元素的中心。对于SVG元素，其坐标系原点是在SVG画板的（0，0）处（假设在SVG标签内祖先元素和自身都不存在任何变换）。如果SVG元素的中心点不在画板的(0,0)点，像rotate,scale或者skew这些变换的结果，都会与HTML元素上应用的结果大不一样。  为了更好地理解这些差异，让我们来看看transform函数是如何工作的。 | | |
| 位移 | 位移会在相同的方向上以相同的距离移动元素上的所有点，并且会保留元素上除了位置信息的所有其它信息。这种位移可以被解释成移动一个元素的坐标原点，所以位置是相对于那个坐标原点的任何元素都会被移动。这种位移之后的效果并不依赖于坐标系的位置。 | | |
| **Figure #1: translate transform: HTML 元素 (左边) vs SVG 元素 (右边)** | **Figure #1: translate transform: HTML 元素 (左边) vs SVG 元素 (右边)** | |
| **HTML 元素** | **SVG 元素** | |
| 上面的图片展示了当css3-transform-translate分别应用到HTML元素和SVG元素上的区别。  正如我们看到的，它们的区别在于各自坐标系的位置。HTML元素的坐标原点在自身50% 50%处，SVG元素的坐标原点在SVG画布0 0处，不过无论坐标原点处在什么位置，它们最后呈现的效果都是一样的。  对于HTML和SVG元素，我们都可以在 CSS transform 中使用3种2D的位移函数：translateX(tx),translateY(ty)和translate(tx[, ty])。  前两个分别作用在 X方向 和Y方向(相对于元素自身的坐标系)。需要注意的是，如果在translate之前存在另外的变换，X方向，Y方向就可能不再代表着水平方向，垂直方向。而第三个位移函数则同时在x,y方向上分别移动tx,ty个单位，ty是可选的，如果不明确指定，默认是0。 | | |
| SVG元素除了CSS tranform属性，还有SVG tranform属性。  在这个例子中，我们在tranform属性中只定义了translate，SVG属性中可以使用逗号分隔，或者空格分隔，其中1代表着1px，下面的两种为SVG元素应用位移的方式是等价的：  使用CSS transform控制:  rect {  /\* doesn't work in IE \*/  transform: translate(295px, 115px);  }  使用SVG transform属性:  <rect width='150' height='80' transform='translate(295 115)' />  注：SVG transform 和 CSS transform 将会被合并  连续的translate()将会被相加在一起，比如我们可以书写一个与translate(tx1 + tx2, ty1 + ty2)等价的链式写法translate(tx1, ty1) translate(tx2, ty2)，注意，这种等价关系只有当两个translate()之间没有任何其它的转换的情况下成立。从translate(tx, ty)返回到初始状态，应用translate(-tx, -ty)即可。 | | |
| 旋转-简单 | <**style**>  **svg**{  border: 1px solid red;  margin: 30px;  }  **rect**{  fill: #f08080;  stroke: #f08080;  stroke-width: 1;  } </**style**> <**svg** id="svg" width="500" height="500">  <**rect** x='0' y='0' width='150' height='100' transform='rotate(45)'/> </**svg**> | | svg元素的两种旋转方式 |
| <**style**>  **svg**{  border: 1px solid red;  margin: 30px;  }  **rect**{  fill: #f08080;  stroke: #f08080;  stroke-width: 1;  -webkit-transform: rotate(30deg);  } </**style**> <**svg** id="svg" width="500" height="500">  <**rect** x='0' y='0' width='150' height='100'/> </**svg**> | |
| 旋转-复杂1 | <**style**>  **svg**{  border: 1px solid red;  margin: 30px;  }  **rect**{  fill: #f08080;  stroke: #f08080;  stroke-width: 1;  -webkit-transform: rotate(30deg);**/\*0->30\*/**  } </**style**>  <**svg** id="svg" width="500" height="500">  <**rect** x='100' y='100' width='150' height='100'/>  <**path** d="M 0,0 l 500,500"></**path**>  <**path** d="M 175,0 v300"></**path**>  <**path** d="M 0,150 h300"></**path**> </**svg**> | | svg元素的旋转是基于svg画板原点(0,0) |
| <**svg** id="svg" width="500" height="500">  <**rect** x='100' y='100' width='150' height='100' transform='rotate(30)'/>  <**path** d="M 0,0 l 500,500"></**path**>  <**path** d="M 175,0 v300"></**path**>  <**path** d="M 0,150 h300"></**path**> </**svg**> | |
| 注：HTML元素的坐标原点在元素自身的50% 50%处，SVG元素的坐标原点则是在SVG画板的0 0处。 | | |
| 旋转-复杂2 | 我们可以在CSS transform中指定transform-origin属性来模拟SVG中的x y参数，长度单位是相对于元素坐标系而言的，百分比单位则是以元素自身为基准，完美！！不过也有一些需要注意的地方。  首先，transform-origin，rotate()中指定固定点，两者是不一样的，比如我们需要绕着元素的50% 50%点旋转SVG元素，下面是两种实现方式： | | |
| <**style**>  **svg**{  border: 1px solid red;  margin: 30px;  }  **rect**{  fill: #f08080;  stroke: #f08080;  stroke-width: 1;  -webkit-transform: rotate(30deg);  transform-origin: 50% 50%;  } </**style**> <**svg** id="svg" width="500" height="500">  <**rect** x='100' y='100' width='150' height='100'/>  <**path** d="M 0,0 l 500,500"></**path**>  <**path** d="M 175,0 v300"></**path**>  <**path** d="M 0,150 h300"></**path**> </**svg**> | |  |
| <**style**>  **svg**{  border: 1px solid red;  margin: 30px;  }  **rect**{  fill: #f08080;  stroke: #f08080;  stroke-width: 1;} </**style**> <**svg** id="svg" width="500" height="500">  <**rect** x='100' y='100' width='150' height='100' transform='rotate(30 175 150)'/>  **<!-- 旋转的度数，100+75,100+50 -->** <**path** d="M 0,0 l 500,500"></**path**>  <**path** d="M 175,0 v300"></**path**>  <**path** d="M 0,150 h300"></**path**> </**svg**> | |
| 多次旋转 | **/\*CSS\*/ rect** {  transform: rotate(45deg) rotate(-45deg);  transform-origin: 50% 50%; **/\* Chrome, Firefox behaves differently \*/** }  在CSS transform中，设置transform-origin为50% 50%，两次的rotate()会相互抵消，因为两次的旋转都是围绕坐标系原点50% 50%，而在SVG transform中，第一次指定了旋转的固定点在元素中心，而第二次并没有指定，所以默认以元素坐标系原点为固定点。如果想实现预期的效果可以指定第二个为:rotate(-45 140 105)而不是rotate(-45)。  我们能为SVG transform的每个roatate()指定不同的固定点，但是只能为CSS transform的每个rotate()指定一个transform-origin。如果想实现一个矩形先绕右下角的点旋转90°,再绕右上角旋转90°，对于SVG tranform来说很容易实现，为每个rotate()指定不同的固定点就可以了：  <**rect** x='0' y='80' width='150' height='80'  transform='rotate(90 150 160) rotate(90 150 80)'/> **<!-- bottom right:  x = x-offset + width = 0 + 150 = 150  y = y-offset + height = 80 + 80 = 160 top right:  x = x-offset + width = 0 + 150 = 150  y = y-offset = 80 -->** | | |
|  | 我们如何做到在CSS transform中实现相同的效果呢？第一步很简单，我们可以指定transform-origin为right bottom，但是第二步呢？如果只是简单地链式书写在第一个的后面， 元素会以right bottom为固定点再次旋转90°。  为了忽略transform-origin的位置，我们需要再添加3个变换，第 一个就是 translate(x, y) ，为了与第二次的旋转固定点对应，我们使用 translate(x, y) 使元素的坐标系原点和我们希望的固定点重合，第二个就是旋转，第三个是translate(-x, -y)(第一次变换的相反操作)。  **/\*CSS\*/ rect** {  **/\* doesn't work as intended in Firefox   \* % values are taken relative to the SVG, not the element  \* which actually seems to be correct \*/** transform-origin: right bottom; **/\* or 100% 100%, same thing \*/** transform:  rotate(90deg)  translate(0, -100%) **/\* go from bottom right to top right \*/** rotate(90deg)  translate(0, 100%); }    注：在火狐浏览器中transform-origin已经支持百分比单位了，但是它的行为与chorme不太一样，所以我们不建议您使用这个方法。 | | |
| 缩放 | 一个缩放函数的效果会依赖元素坐标系原点的位置，对于同一个元素，两个具有相同缩放因子的缩放函数，会因为不同的坐标系原点表现出不同的效果。   |  |  | | --- | --- | | Figure #7: scale transform: HTML 元素 (左边) vs SVG 元素 (右边) | Figure #7: scale transform: HTML 元素 (左边) vs SVG 元素 (右边) | | HTML 元素 | SVG 元素 |   上图是分别在HTML元素和SVG元素上应用缩放的不同结果，两者拥有完全相同的缩放因子，不同的是，HTML元素的坐标系原点在元素50% 50%处，而SVG元素的坐标原点则是SVG画板的0 0处。  在CSS transform中有3种2D的缩放函数： scale(sx[, sy])、 scaleX(sx)、scaleY(sy)。第一个函数，会同时在x和y方向上应用sx和sy缩放因子，sy是可选的，如果该函数只有一个参数，会默认是sx的值。其它两个函数是分别在两个方向进行缩放，scaleX(sx)、 scaleY(sy)分别等价于scale(sx,1) 、scale(1,sy)，如果在此之前存在其它的变换，那么相应的x和y方向将不再是水平和垂直的。  在SVG transform中，只有scale(sx[ sy])。同样，可以使用空格或者逗号分隔参数。 | | |
|  | 对于SVG元素来说，以下的两种缩放方法是等价的：  1.使用CSS transform:  /\*CSS\*/rect {  /\* doesn't work in IE \*/  transform: scale(2, 1.5);}  2.使用SVG transform:  <!-- HTML -->  <!-- works everywhere -->  <rect x='65' y='65' width='150' height='80' transform='scale(2 1.5)' />  在同一个元素上应用上面两种方法会产生相同的效果，就像上一个图片中的右边的区域。 | | |
|  | 实现像应用在HTML元素上一样的效果(以重心为中心进行缩放):  1.使用CSS transform中的transform-origin改变元素的坐标原点  2.使用translate() scale() translate()这样的组合:我们先将元素坐标系原点移动到元素的50% 50%位置，然后应用缩放，最后做与第一个位移相反的操作。  3.如果使用SVG transform属性，我们只能使用前面CSS transform中的第二种方式，类似的方法，下面是具体的代码。  1.使用CSS transform中的transform-origin()：  /\*CSS\*/  rect {  transform: scale(2, 1.5);  transform-origin: 50% 50%;  }  2.在CSS transform中链式调用：175 150 => 两次中心点之差  rect {  -webkit-transform: translate(175px,150px) scale(2,2) translate(-175px,-150px);  }  3.在SVG transform中链式调用:  <**rect** x='100' y='100' width='150' height='100' transform=' translate(175 150) scale(2 2) translate(-175 -150)'/>  <!-- works everywhere --> | | |
|  | 还有就是，两个连续的scale()如scale(sx1, sy1) scale(sx2, sy2),可以写成scale(sx1\*sx2, sy1\*sy2)。scale(1/sx1, 1/sy1)可以对scale(sx1,sy1)做相反的操作。如果所有的scale()参数的绝对值等于1，那么这个元素不产生任何的缩放。 | | |
| 倾斜 | 倾斜操作会将元素上所有点的坐标值在相应的方向上移动一段距离，这段距离取决于倾斜的角度而和每个点到倾斜轴的距离。也就是说，在一个方向上倾斜，另一个方向上的所有点相应的坐标值不会被改变。倾斜一个元素必定会扭曲这个元素，这点和旋转不太一样，倾斜一个矩形，这个矩形会变成不等边的平行四边形，倾斜一个圆，这个圆会变成椭圆。倾斜操作过后，角度(对于倾斜角α，矩形的直角将会变成90° ± α)或者长度都会发生变化，但是元素的原始区域会被保留。  另外，与位移，旋转不一样的是，倾斜不具有可加性，连续两次α1 α2的倾斜，不等于一次α1+α2的倾斜。  倾斜操作与旋转操作一样，元素的坐标原点都会影响最终的呈现效果，对同一个元素在相同的轴方向上做相同倾斜操作，会产生不同的结果： | | |
|  | html svg | | |
|  | 以上是分别对HTML元素和SVG元素做相同的倾斜操作后的图片，相同的角度，相同的倾斜轴，不同的是，它们的坐标原点不一样。HTML元素在其50% 50%处，SVG元素在SVG画布的0 0处。  方便起见，在这我们只关注元素上的一个点：右上角。两者右上角在垂直方向上始终保持不变，在水平方向上，HTML元素中向左平移，SVG元素中向右平移，右下角都向右平移。为什么会这样？  就像前面提到的，X轴方向的倾斜，y轴方向的坐标保持不变，X轴方向的坐标会增加或者减少d，这个d取决于倾斜角度和该点y轴上的坐标。如果倾斜角在[0°, 90°]之间，则d的符号与y坐标相同，如果在[-90°, 0°]之间，则相反。  在这个例子中，倾斜角是60°，所以在这里右上角y轴方向的坐标是导致两者不同的关键，在HTML元素中，由于坐标系原点在元素中央，所以右上角的y坐标为负值，在SVG元素中，坐标系原点在SVG画板的0 0处，所以右上角y坐标为正值。像上面所说的，这样分别加在两者右上角d的符号也是相反的，也就造成了HTML元素的右上角向左移动，SVG元素的右上角向右移动。  无论是CSS transform还是SVG transform，都有两个倾斜函数:skewX(angle)和skewY(angle)，前者的倾斜轴是X轴，后者的倾斜轴是y轴。  在CSS transform中，angle的单位可以是deg rad turn grad ,还能使用calc()(需要牢记的是，只有Blink内核的浏览器支持在calc()中计算角度)  在SVG transform中，倾斜角度都是没有单位的deg值。  也就是说有两种方法书写上一张图片中右边的效果：  使用CSS transform:  rect {  transform: skewX(60deg); /\* doesn't work in IE \*/}  使用SVG transform:  <!-- works everywhere --><rect x='65' y='65' width='150' height='80' transform='skewX(60)' />  如果我想实现像应用在HTML元素上一样的效果呢？就像旋转一样，存在3种方式： | | |
|  | 使用CSS transform的transform-origin(不建议使用):  rect {  transform: skewX(60deg);  transform-origin: 50% 50%;}  在CSS transform中链式调用：  rect {  transform: translate(140px, 105px)  skewX(60deg)  translate(-140px, -105px);}  在SVG transform中链式调用:  <!-- works everywhere --><rect x='65' y='65' width='150' height='80'  transform='translate(140 105) skewX(60) translate(-140 -105)' /> | | |
| 简写调用链 | 我们可以通过链式调用，在SVG元素上实现在HTML元素上进行rotate scale skew操作一样的效果，使用SVG transform也能做到兼容IE浏览器。然而，你不觉得这样的写法很丑陋么？难道就没有更加简便的方式？  如果我们可以将元素的坐标系原点移动到元素自身的50% 50%处，那么调用链会大大缩短，像下面这样：  <rect x='-75' y='-40' width='150' height='80'  transform='translate(140 105) rotate(45)'/>  <!-- 75 = 150/2, 40 = 80/2 --> | | |
|  | 如果使用SVG元素的[viewBox](http://sarasoueidan.com/blog/svg-coordinate-systems/" \t "http://www.w3cplus.com/svg/_blank)属性，第一个translate也能被精简掉。viewBox属性中有4个值，前两个指定了SVG画布的左上角在显示区域的位置， 后两个则是SVG画布显示区域的width和height，如果没有明确指定viewBox，画布的位置就是在显示区域的0 0处。  下面的两张图片能很好地展现viewBox='-140 -105 280 210'使用前和使用后的区别。  Figure #9: viewBox使用前 vs. viewBox使用后  viewBox使用前 vs. viewBox使用后。  回到刚才的话题，如果想将SVG画布的坐标原点0 0移动到矩形的中心点，我们可以像下面这样设置viewBox:  <svg viewBox='-140 -105 650 350'>  <rect x='-75' y='-40' width='150' height='80' transform='rotate(45)'/>  </svg>    将SVG画布的原点移动到元素的中心点，可以大大简化SVG图形的变换操作。 | | |
|  |  | | |
|  |  | | |

动画animate

简介

|  |  |
| --- | --- |
| www | <http://www.520ued.com/article/svg-animate>  <http://www.zhangxinxu.com/wordpress/2014/08/so-powerful-svg-smil-animation/> |
| 兼容 |  |
| 5无素 | <set>  <animate>  <animateColor>  <animateTransform>  <animateMotion> |
|  | begin dur ：动画的开始时间与结束时间。如果你想打开页面过几秒才开始动画，那么就定义begin就好了。 |

公有属性详解

|  |  |  |
| --- | --- | --- |
| attributeName | attributeName = <attributeName>  动画属性名：要变化的元素属性名称；  -可以是元素直接暴露的属性;例如，的text元素上的x, y或者font-size;  -可以是CSS属性;例如，透明度opacity. | |
| attributeType | attributeType = “CSS | XML | auto”  动画属性的类别，常见的两个值，XML和css，取决于属性是在xml里还是在style里;  attributeType支持三个固定参数，CSS/XML/auto. 用来表明attributeName属性值的列表。  x, y以及transform就属于XML,  opacity就属于CSS.  auto为默认值，自动判别的意思（实际上是先当成CSS处理，如果发现不认识，直接XML类别处理）。  因此，如果你不确信某属性是XML类别还是CSS类别的时候，我的建议是不设置attributeType值，直接让浏览器自己去判断，几乎无差错。  不知大家有没有和我一样的疑问：“既然浏览器酱可以自己判断属性类别，那这个属性还有什么意义吗？”我琢磨着，可能某些属性，XML能其作用，CSS也能其作用，例如font-size, 此时就需要明确下归属。 | |
| from to by values | 4个属性是一个家族的，解决的是：从哪里来、到哪里去  from = “<value>“ 动画的起始值。  to = “<value>“ 指定动画的结束值。  by = “<value>“ 动画的相对变化值。  values = “<list>“ 用分号分隔的一个或多个值，可以看出是动画的多个关键值点。 | |
|  | from, to, by, values虽然属于一个家族，但是相互之间还是有制约关系的。有以下一些规则：  ---from和to只能定义开始和终结两个时间点，第一帧和最后一帧并没有衔接起来，会导致看起来有突兀。属性by可以替代to的，to表示绝对值，by表示相对值；  from='50' to='80'，拿位移距离说明，则表示移动到80这个位置  from='50' by='80'，拿位移距离说明，则表示移动到130这个位置  ---如果动画的起始值与元素的默认值是一样的，from参数可以省略；  ---如果to,by,values都没设置，自然没动画效果；  ---不考虑values，to by两个参数至少需要有一个出现，否则动画效果没有；  ---如果to by同时出现，则by打酱油，只识别to；  ---如果任意（包括from）一个属性的值不合法，规范上说是没有动画效果。但是，据我测试，FireFox浏览器确实如此，但是Chrome特意做了写容错处理。例如，本来是数值的属性，写了个诸如a这个不合法的值，其会当作0来处理，动画效果依然存在。  ---values可以是一个值或多值。根据我在Chrome浏览器下的测试，是一个值的时候是没有动画效果。多值时候有动画效果。当values值设置并能识别时候，from, to, by的值都会被忽略。那values属性是干什么的呢？别看名字挺大众的，其还是有些功力的。我们实现动画，不可能就是单纯的从a位置到b位置，有时候，需要去c位置过渡下。此时，实际上有3个动画关键点。而from, to/by只能驾驭两个，此时就是values大显身手的时候了！  ---总结下，也就是from-to动画、from-by动画、to动画、by动画以及values动画 | |
| begin, end | begin指动画开始的时间，看上去很简单。设个时间，延迟嘛~~实际上非也非也，上面出现的beigin="3s"只是最简单最基本的表示。  begin的定义是分号分隔的一组值。看到没？是一组值，单值只是其中的情况之一。  例如，beigin="3s;5s"表示的是3s之后动画走一下，6s时候动画再走一下（如果之前动画没走完，会立即停止从头开始）。所以，如果一次动画时间为3s, 即dur="3s"，同时没有repeatCount属性时候，我们可以看到动画似乎连续执行了2次。 | |
|  | 时间值  既然这里提到了时间，就顺势讲简单一下SVG animation中的时间表示(也适用于dur, end属性)。常见单位有 "h"|"min"|"s"|"ms"  时间值支持的格式和规则相当复杂，例如我我规范上看到这个：1997-07-16T19:20:30.45+01:00. 以及洋洋洒洒N多看不懂的示意。尼玛，这个要通透我周末钓鱼时间都没了，关键是没有必要。所以，我们还是了解下最常见的基本使用。  上面的单位含义都是英文单位的缩写。例如h表示小时(hour).  时间值支持小数写法，因此，90s我们也可以使用1.5miu表示。时间值还支持hh:mm:ss这种写法，因此，90s我们也可以使用01:30表示。  还有一点，十进制的小数值是秒的浮点定义。什么意思呢？就是如果begin="1.5"没有单位，这里的小数点表示秒，也就是1.5s的意思。所以，上面N次出现的beigin="3s"也可以简写作beigin="3". 我测了下，FireFox和Chrome浏览器都是支持的。  begin的单值除了普通value，还有下面这些类别的value：offset-value | syncbase-value | event-value | repeat-value | accessKey-value | media-marker-value | wallclock-sync-value | "indefinite"  【1】offset-value表示偏移值，数值前面有+或-. 应该指相对于documentdocument的begin值而言。  【2】syncbase-value基于同步确定的值。语法为：[元素的id].begin/end +/- 时间值. 就是说借用其他元素的begin值再加加减减，这个可以准确实现两个独立元素的动画级联效果。  实现一个先横向走，再纵向走的效果：后面attributeName为y的元素的begin值是x.end. x.end中的x就是上面一个animate元素的id值，而end是动画元素都有的一个属性，动画结束的时间。因此，begin="x.end"意思就是，当id为x的元素动画结束的时候，我执行动画。非常类似于PowerPoint动画的“上一个动画之后”的选项。当然，我们还可以增加一些偏移值，例如begin="x.end-1s", 就表示id为x的元素动画结束前一秒开始纵向移动。  【3】event-value这个表示与事件相关联的值。类似于PowerPoint动画的“点击执行该动画”。语法是：[元素的id].[事件类型] +/- 时间值. 举个例子，点击下图的圆圈圈，马儿它就会自己跑！  代码的关键点就是上面红色高亮的begin="circle.click", 其中circle为circle元素（黑色圆）的id, click表示点击事件。含义一目了然，如果你想点击圆圈圈2秒钟后马儿才跑，很简单，偏移时间加上就可以了——begin="circle.click+2s".  主要注意的是，这类与事件关联的SVG需要内联在页面中，否则click什么的都是徒劳。  【4】repeat-value指重复多少次之后干嘛干嘛。语法为：[元素的id].repeat(整数) +/- 时间值. 举个例子，下面这个马儿会在水平运动2次之后，斜向运动；  begin="x.repeat(2)"指id为x的元素的动画重复2次后执行~~  【5】accessKey-value定义快捷键。即按下某个按键动画开始。语法为：accessKey(" character "). character表示快捷键所在的字符，举个例子，按下s键动画走起。  按下键盘上的字母"s", 理论上动画就会执行。但是，据我测试，我的Chrome浏览器（版本36）上是没有效果的，FireFox浏览器效果杠杠的！所以，如果您的浏览器没有效果，但是手上有火狐，可以复制下面这个地址去FireFox浏览器下感受下：<http://www.zhangxinxu.com/study/201408/horse-accesskey-value.svg>  【6】wallclock-sync-value指真实世界的时钟时间定义。时间语法是基于在ISO8601中定义的语法。例如上面提到的1997-07-16T19:20:30.45+01:00这个让人呵呵呵的时间表示。  【7】"indefinite"就是这个字符串值，表示“无限等待”。据说需要beginElement()方法触发 或者 指向该动画元素的超链接(SVG中的a元素)。 | |
|  | 【2】 | <**svg** width="320" height="200" xmlns="http://www.w3.org/2000/svg">  <**text** y="160" x="160">^m~  <**animate** id="x" attributeName="x" to="60" begin="0s" dur="3s" fill="freeze" />  <**animate** attributeName="y" to="100" begin="x.end" dur="3s" fill="freeze" />  </**text**> </**svg**> |
|  | 放一排的圆，让一个动完，才让下一个接着动。这也就是我们编程里的“同步”概念，在svg里就是 同步动画。同步动画的实现很简单，只需要有动画id，然后下一个动画的 begin 值 为上一个的 id.end, 比如  <animate id="a1"/>  <animate id="a2" begin="a1.end"/>  我们来看一下同步动画的实例：   |  |  |  | | --- | --- | --- | |  |  | [点击查看Demo](https://chengrang.com/demo/svg/demo7.html" \t "http://www.520ued.com/article/_blank) |   更为强大的是，begin的值可以是表达式，比如  <animate id="a2" begin="a1.end + 5s"/>  这样就实现了同步的同时，带有5s的偏移，也就是第一个结束5s之后，第二个动画才开始。OK，同步动画实现了. |
|  | **<!--相对变化-->** <**svg** width="800" height="400">  <**circle** cx="100" cy="100" r="0" style="fill:#d8210d">  <**animate** id="a1" attributeName="r" attributeType="XML" from="0" to="30" begin="0s" dur="2s" fill="freeze"></**animate**>  </**circle**>  <**circle** cx="150" cy="100" r="0" style="fill:#e61b64">  <**animate** id="a2" attributeName="r" attributeType="XML" from="0" to="30" begin="a1.end+1" dur=".2s" fill="freeze"></**animate**>  </**circle**>  <**circle** cx="200" cy="100" r="0" style="fill:#d7006f">  <**animate** id="a3" attributeName="r" attributeType="XML" from="0" to="30" begin="a2.end" dur=".2s" fill="freeze"></**animate**>  </**circle**>  <**circle** cx="250" cy="100" r="0" style="fill:#ee869a">  <**animate** id="a4" attributeName="r" attributeType="XML" from="0" to="30" begin="a3.end" dur=".2s" fill="freeze"></**animate**>  </**circle**>  <**circle** cx="300" cy="100" r="0" style="fill:#e5004f">  <**animate** id="a5" attributeName="r" attributeType="XML" from="0" to="30" begin="a4.end" dur=".2s" fill="freeze"></**animate**>  </**circle**>  <**circle** cx="350" cy="100" r="0" style="fill:#ea5520">  <**animate** id="a6" attributeName="r" attributeType="XML" from="0" to="30" begin="a5.end" dur=".2s" fill="freeze"></**animate**>  </**circle**>  <**circle** cx="400" cy="100" r="0" style="fill:#e61b64">  <**animate** id="a7" attributeName="r" attributeType="XML" from="0" to="30" begin="a6.end" dur=".2s" fill="freeze"></**animate**>  </**circle**> </**svg**> |
| 【3】 | <**svg** id="svg" width="320" height="200" xmlns="http://www.w3.org/2000/svg">  <**circle** id="circle" cx="100" cy="100" r="50"></**circle**>  <**text** y="160" x="160">马  <**animate** attributeName="x" to="60" begin="circle.click" dur="3s" />  </**text**> </**svg**> |
| 【4】 | <**svg** width="320" height="200" xmlns="http://www.w3.org/2000/svg">  <**text** font-family="microsoft yahei" font-size="120" y="160" x="160">马  <**animate** id="x" attributeName="x"  to="60" begin="0s" dur="3s" repeatCount="indefinite" />  <**animate** attributeName="y"  to="100" begin="x.repeat(2)" dur="3s" fill="freeze" />  </**text**> </**svg**> |
| 【5】 | <**svg** width="320" height="200" xmlns="http://www.w3.org/2000/svg">  <**text** font-family="microsoft yahei" font-size="120" y="160" x="160">马  <**animate** attributeName="x" to="60" begin="accessKey(s)" dur="3s" repeatCount="indefinite" />  </**text**> </**svg**> |
| 【7】 | <**svg** id="svg" width="320" height="200" xmlns="http://www.w3.org/2000/svg">  <**text** font-family="microsoft yahei" font-size="120" y="160" x="160">马  <**animate** id="animate"attributeName="x"  to="60" begin="indefinite" dur="3s"/>  </**text**> </**svg**> <**script** type="text/javascript">  // var animate = document.getElementsByTagName("animate")[0];  **var animate** = document.getElementById("animate");  **if** (**animate**) {  document.getElementById("svg").onclick = **function** () {  **animate**.beginElement();  };  } </**script**> |
| 上面代码是beginElement()方法触发的例子：意思很简单，点击我们的svg, 触发animate元素的beginElement()方法，前提是begin="indefinite".  由于牵扯JS, 文章页拘谨了，您可以狠狠地点击这里：[beginElement()方法触发SVG动画demo](http://www.zhangxinxu.com/study/201408/svg-animation-beginelement.html" \t "http://www.zhangxinxu.com/wordpress/2014/08/so-powerful-svg-smil-animation/_blank) |
| <**svg** width="320" height="200">  <**a xlink**:href="#animate">  <**text** x="10" y="20" fill="#cd0000" font-size="30">点击我</**text**>  </**a**>  <**text** font-size="120" y="160" x="160">马  <**animate** id="animate" attributeName="x"  to="60" begin="indefinite" dur="3s" repeatCount="indefinite" />  </**text**> </**svg**> |
| 从上面代码可以看出，动画触发条件很简单，只要a元素的xlink:href指向的我们的动画元素就可以了。如果上面SVG没效果，可以试试点击这里直接浏览器中访问：[horse-begin-link.svg](http://www.zhangxinxu.com/study/201408/horse-begin-link.svg" \t "http://www.zhangxinxu.com/wordpress/2014/08/so-powerful-svg-smil-animation/_blank)  最后，搞一段规范上出现的一段文字：  If no begin is specified, the default value is “0” – the animation begins when the document begins. If there is any error in the argument value syntax for begin, the default value for begin will be used.  意思是，没有begin或者begin参数解析异常，都当作0处理。  说到现在基本上都是begin属性，实际上end与begin除了名字和字面含义不一样，其值的种类与表意都是一模一样的，我这里就不重复啰嗦了。 |
| + svg动画的暂停 | <**svg** id="svg" width="320" height="200" xmlns="http://www.w3.org/2000/svg">  <**text** font-family="microsoft yahei" font-size="120" y="160" x="160">马  <**animate** id="animate" attributeName="x"  to="60" begin="0" dur="3s"/>  </**text**> </**svg**> <**script** type="text/javascript">  **var svg**=document.getElementById("svg");  **var isRun**=**true**;  **if** (animate) {  **svg**.onclick = **function** () {  **if**(**isRun**){  console.log("stop:::");  **svg**.pauseAnimations();//暂停  }**else**{  console.log("begin:::");  **svg**.unpauseAnimations();//开始  }  **isRun**=!**isRun**;  };  } </**script**> | |
| dur | dur属性值比begin简单了好几层楼，就后面两种：常规时间值 | "indefinite".  “常规时间值”就是3s之类的正常值；  "indefinite"指事件无限。试想下，动画时间无限，实际上就是动画压根不执行的意思。因此，设置为"indefinite"跟没有dur是一个意思，与dur解析异常一个意思。 | |
| calcMode, keyTimes, keySplines | 这几个参数是控制动画先快还是先慢类似这样作用的。 | |
| calcMode属性支持4个值：discrete | linear | paced | spline. 中文意思分别是：“离散”|“线性”|“踏步”|“样条”。  discrete：from值直接跳到to值。  linear：animateMotion元素以外元素的calcMode默认值。动画从头到尾的速率都是一致的。  paced：通过插值让动画的变化步调平稳均匀。仅支持线性数值区域内的值，这样点之间“距离”的概念才能被计算（如position, width, height等）。如果”paced“指定，任何keyTimes或keySplines值都会打酱油。  spline：插值定义贝塞尔曲线。spline点的定义在keyTimes属性中，每个时间间隔控制点由keySplines定义。 | |
| keyTimes = “<list>”  跟上面提到的<list>类似，都是分号分隔一组值。keyTimes总名字上看是关键时间点的意思，大致就是这个意思。前面提到过values也是多值，这里有一些约定的规则：首先，keyTimes值的数目要和values一致，如果是from/to/by动画，keyTimes就必须有两个值。然后对于linear和spline动画，第一个数字要是0, 最后一个是1。 最后，每个连续的时间值必须比它前面的值大或者相等。  paced模式下，keyTimes会被忽略；keyTimes定义错误，也会被忽略；dur为indefinite也会被忽略。 | |
| keySplines = “<list>” keySplines表示的是与keyTimes相关联的一组贝塞尔控制点（默认0 0 1 1）。每个控制点使用4个浮点值表示：x1 y1 x2 y2. 只有模式是spline时候这个参数才有用，也是分号分隔，值范围0~1，总是比keyTimes少一个值。  如果keySplines值不合法或个数不对，是没有动画效果的。 | |
|  | 叨叨这么多，规范的术语还真是拗口，不急，我们先感受例子，然后再给大家通俗解释：  如下4个SVG;其效果为……您可以狠狠地点击这里：[calcMode, keyTimes, keySplines属性demo](http://www.zhangxinxu.com/study/201408/svg-animation-calcmode.html" \t "http://www.zhangxinxu.com/wordpress/2014/08/so-powerful-svg-smil-animation/_blank);可以看到到4匹马上半途中你追我赶的经常场面； | |
| <**svg** width="320" height="100" xmlns="http://www.w3.org/2000/svg">  <**text** font-size="60" y="60" x="160">马  <**animate** attributeName="x" dur="5s" values="0; 20; 160" calcMode="linear" />  </**text**> </**svg**> <**svg** width="320" height="100" xmlns="http://www.w3.org/2000/svg">  <**text** font-size="60" y="60" x="160">马  <**animate** attributeName="x" dur="5s" values="0; 20; 160" calcMode="paced"/>  </**text**> </**svg**> <**svg** width="320" height="100" xmlns="http://www.w3.org/2000/svg">  <**text** font-size="60" y="60" x="160">马  <**animate** attributeName="x" dur="5s" values="0; 80; 160" keyTimes="0; .8; 1" calcMode="linear"/>  </**text**> </**svg**> <**svg** width="320" height="100" xmlns="http://www.w3.org/2000/svg">  <**text** font-size="60" y="60" x="160">马  <**animate** attributeName="x" dur="5s" values="0; 80; 160"  keyTimes="0; .8; 1" calcMode="spline" keySplines=".5 0 .5 1; 0 0 1 1" />  </**text**> </**svg**> | |
| 重要部分代码：  <animate attributeName="x" dur="5s" values="0; 20; 160" calcMode="linear" />  <animate attributeName="x" dur="5s" values="0; 20; 160" calcMode="paced"/>  <animate attributeName="x" dur="5s" values="0; 80; 160"  keyTimes="0; .8; 1" calcMode="linear"/>  <animate attributeName="x" dur="5s" values="0; 80; 160"  keyTimes="0; .8; 1" calcMode="spline" keySplines=".5 0 .5 1; 0 0 1 1" />   |  |  |  |  | | --- | --- | --- | --- | | IMG_256 | IMG_257 | IMG_258 | IMG_259 | | ="0 0 1 1"  default | =".5 0 .5 1" | ="0 .75 .25 1" | ="1 0 .25 .25" | | keySplines | | | |   拿最后一个SVG说事吧，实际上就是values, keyTimes, keySplines三个人之间事情。values确定动画的关键位置，keyTimes确定到这个关键点需要的时间，keySplines确定的是每个时间点段之间的贝塞尔曲线，也就是具体的缓动表现。我们平时CSS3写的transition动画效果，也是这么回事，这是values值就两个，所以，keyTimes只能是0;1, 贝塞尔曲线就只有一个，要不ease, 要不linear等。 | |
| repeatCount, repeatDur | repeatCount表示动画执行次数，想让动画一直循环播放就要用到，属取值可以是一个数字，也可以是“indefinite”表示无限循环。  repeatDur定义重复动画的总时间。可以是普通时间值或者"indefinite"（动画循环到电脑死机）。  例如这个：  <animate attributeName="x" to="60" dur="3s"  repeatCount="indefinite" repeatDur="10s" />  动画只玩执行完整3个 + 一个1/3个动画。因为repeat总时间就10s而已。 | |
| fill | 动画结束的时候又回到了第一帧，如果我们想让他停在最后一帧，就需要用到 fill 属性（animate的fill属性与svg的fill填充要区分一下）；  fill表示动画间隙的填充方式。  支持参数有：freeze | remove.  remove是默认值，表示动画结束直接回到开始的地方。  freeze“冻结”表示动画结束后像是被冻住了，元素保持了动画结束之后的状态。 | |
| accumulate, additive | accumulate是累积的意思。支持参数有：none | sum. 默认值是none. 如果值是sum表示动画结束时候的位置作为下次动画的起始位置。  additive控制动画是否附加。支持参数有：replace | sum. 默认值是replace. 如果值是sum表示动画的基础知识会附加到其他低优先级的动画上 | |
|  | 例子1：  <img ...>     <animateMotion begin="0" dur="5s" path="[some path]" additive="sum" fill="freeze" />     <animateMotion begin="5s" dur="5s" path="[some path]" additive="sum" fill="freeze" />     <animateMotion begin="10s" dur="5s" path="[some path]" additive="sum" fill="freeze" />  </img>  这里轮到第二个动画的时候，路径是从第一个动画路径结束地方开始的，于是，3个动画完美无缝连接起来了。 | |
|  | 例子2：  <**svg** xmlns="http://www.w3.org/2000/svg" width="320" height="200">  <**title**>马儿跑</**title**>  <**text** font-family="microsoft yahei" font-size="40" y="60" x="100">马  <**animateTransform** attributeName="transform" type="scale"  from="1" to="2"  dur="10s" repeatCount="indefinite" additive="sum"/>  <**animateTransform** attributeName="transform" type="rotate"  from="0 100 60" to="360 100 60"  dur="10s" fill="freeze" repeatCount="indefinite" additive="sum"/>  </**text**>  **<!-- transform="scale(1.82967 1.82967) rotate(298.68 100 60)"-->** </**svg**>  这里，两个动画同时都是transform，都要使用一个type属性，好在这个例子additive="sum"是累加的而不是replace替换。于是，我们就可以是实现一边旋转一边放大的效果  [horse-animate-sum.svg](http://www.zhangxinxu.com/study/201408/horse-animate-sum.svg" \t "http://www.zhangxinxu.com/wordpress/2014/08/so-powerful-svg-smil-animation/_blank) | |
| restart | restart这个属性诞生的背景如下：很多动画呢，其触发可能与事件相关，例如，点击某圆圈，马儿就跑。而且，似乎没点一次，马儿就跑一下。现在，存在这种情况，希望马儿只跑一次，之后在点击就没有反应。这种需求的出现迫使restart参数的出现。支持的参数有：always | whenNotActive | never.  always是默认值，表示总是，也就是点一次圈圈，马儿跑一下。whenNotActive表示动画正在进行的时候，是不能重启动画的。never表示动画是一波流。 | |
| min, max | min/max表示动画执行最短和最长时间。支持参数为时间值和"media"（媒介元素有效）, max还支持indefinite. | |
| 其他遗漏参数 | 本文前半部分的演示内容组织参考：[SVG Animation](http://tutorials.jenkov.com/svg/svg-animation.html) 本文后半部分的参数演示详解参考：[SMIL Animation](http://www.w3.org/TR/2001/REC-smil-animation-20010904/) | |
|  |  | |
|  |  | |

set

|  |  |
| --- | --- |
| set | set意思设置，此元素没有动画效果。你可能会疑问了，既然这个元素没有动画效果，怎么会是animation五大天团成员之一呢？  OK, 这样的，虽然set虽然不能触发连续的动画，但是，其还是可以实现基本的延迟功能。就是指：可以在特定时间之后修改某个属性值（也可以是CSS属性值）。  举个例子，下面这个A会在3秒之后从横坐标160的位置移动60这个位置 [set标签需写在text标签中] |
|  | <**svg** width="320" height="320" xmlns="http://www.w3.org/2000/svg">  <**g**>  <**text** font-family="microsoft yahei" font-size="120" y="160" x="160">  A  <**set** attributeName="x" attributeType="XML" to="60" begin="1s"/>  </**text**>  </**g**> </**svg**> |
|  |  |

animate

|  |  |
| --- | --- |
| 基础属性 |  |
| lg:平移 | <**svg** width="320" height="320" xmlns="http://www.w3.org/2000/svg">  <**g**>  <**text** font-family="microsoft yahei" font-size="120" y="160" x="160">A  <**animate** attributeName="x"   from="160" to="60"   begin="0s" dur="3s"   repeatCount="indefinite" />  </**text**>  </**g**> </**svg**> |
| <**svg** width="320" height="200" xmlns="http://www.w3.org/2000/svg">  <**text** font-family="microsoft yahei" font-size="120" y="150" x="160"> 马  <**animate** attributeName="x"  values="160;40;160"  dur="3s"  repeatCount="indefinite" />  </**text**> </**svg**> |
| lg:呼吸+双重 | 想让这个圆形呼吸的同时改变透明度，这就是svg在单个对象上面做多重动画了。我们在 attributeName=”r” 下面，增加一个animate元素，用来改变圆的透明度  <animate attributeName="fill-opacity" attributeType="CSS" values="1;0.6;1" begin="0s" dur="2s" repeatCount="indefinite"></animate> |
| <**svg** width="400" height="400">  <**circle** cx="200" cy="200" r="80" style="fill:#ff6600">  <**animate** attributeName="r" attributeType="XML"  values="50;80;50"  begin="0s" dur="2s"  repeatCount="indefinite">  </**animate**>  <**animate** attributeName="fill-opacity" attributeType="CSS"  values="1;0.6;1"  begin="0s" dur="2s"  repeatCount="indefinite">  </**animate**>  </**circle**> </**svg**> |
| lg:马儿入山影无踪 | <**svg** width="320" height="200" xmlns="http://www.w3.org/2000/svg">  <**text** font-family="microsoft yahei" font-size="120" y="160" x="160">马~~  <**animate** attributeName="x"  from="160" to="60" begin="0s" dur="3s" repeatCount="indefinite" />  <**animate** attributeName="opacity"  from="1" to="0" begin="0s" dur="3s" repeatCount="indefinite" />  </**text**> </**svg**> |
|  |  |
|  |

animateTransform

|  |  |
| --- | --- |
| 通过transform实现动画 | SVG的旋转动画需要用到另一个<animateTransform>元素，他的用法和animate并没有本质的区别，只不过属性需要换一批。  这里的transform变换与CSS3的transform变换，以及Snap.svg.js中的transform()方法都是一个路数。  ---其中from、to分别由三个值构成：第一个值是角度，从0到360度的旋转，第二个和第三个组成一个坐标，这个坐标决定了svg围绕哪个点来进行旋转，关于这方面的知识推荐新同学了解一下SVG的坐标系相关知识。transform的type可以包括translate，scale,skew等，和css相通，用法也与rotate大同小异。 |
| 五角星原地旋转: |
| <**svg** width="400" height="400">  <**polygon** points="122 59,72 205,194 114,49 114,171 205" style="fill: #ff6600;">  <**animateTransform** attributeName="transform" attributeType="XML"  type="rotate"  from="0 125 135" to="360 125 135"  begin="0s" dur="10s"  repeatCount="indefinite">  </**animateTransform**>  </**polygon**> </**svg**> |
| A原地旋转: |
| <**svg** width="320" height="320" xmlns="http://www.w3.org/2000/svg">  <**g**>  <**text** font-size="120" y="160" x="160">D</**text**>  <**animateTransform** attributeName="transform"  type="rotate"  from="0 160 160" to="360 160 160"  begin="0s" dur="10s"  repeatCount="indefinite"/>  </**g**> </**svg**> |
|  | A变换大小 |
|  | <**svg** width="320" height="320" xmlns="http://www.w3.org/2000/svg">  <**g**>  <**text** font-family="microsoft yahei" font-size="80" y="100" x="100">A</**text**>  <**animateTransform** attributeName="transform"  type="scale"  from="1" to="1.5"  begin="0s" dur="3s"  repeatCount="indefinite"/>  </**g**> </**svg**> |

animateMotion

|  |  |
| --- | --- |
| 沿着路径运动 | 可以用他来实现引导线动画，让图形沿着复杂的路径运动： **<!--五角星路径旋转-->** <**svg** width="800" height="800">  <**path** d="M100 100, A 120 120, -45 0 1, 300 300 A 120 120, -45 0 1, 100 100" stroke="#ccc" stroke-width="2" fill="none"></**path**>  <**polygon** points="-12 -69,-58 85,64 -14,-81 -14,41 85" style="fill: #ff6600;">  <**animateMotion** dur="6s" repeatCount="indefinite"  path="M100 100, A120 120, -45 0 1, 300 300 A120 120, -45 0 1, 100 100">  </**animateMotion**>  </**polygon**> </**svg**> |
| 如果期望五角星沿着路径转动的时候随着角度倾斜，就需要用到rotate="auto" |
| **<!--五角星路径旋转-->** <**svg** width="800" height="800">  <**path** d="M100 100, A 120 120, -45 0 1, 300 300 A 120 120, -45 0 1, 100 100"  stroke="#ccc" stroke-width="2" fill="none"></**path**>  <**polygon** points="-12 -69,-58 85,64 -14,-81 -14,41 85" style="fill: #ff6600;">  <**animateMotion** dur="6s" repeatCount="indefinite"  rotate="auto"  path="M100 100, A120 120, -45 0 1, 300 300 A120 120, -45 0 1, 100 100">  </**animateMotion**>  </**polygon**> </**svg**> |
| 字符沿着路径运行 |
| <**svg** width="360" height="200" xmlns="http://www.w3.org/2000/svg">  <**path** d="M10,80 q100,120 120,20 q140,-50 160,0"  stroke="#cd0000" stroke-width="2" fill="none" />  <**text** font-family="microsoft yahei" font-size="40" x="0" y="0" fill="#cd0000">M  <**animateMotion** path="M10,80 q100,120 120,20 q140,-50 160,0"  rotate="auto"  begin="0s" dur="3s"  repeatCount="indefinite"/>  </**text**> </**svg**> |
| mpath 的使用 |
| <**svg** width="120" height="120" viewBox="0 0 120 120" >  **<!-- 画路径 和 两个顶点 -->** <**path** d="M10,110 A120,120 -45 0,1 110 10 A120,120 -45 0,1 10,110"  stroke="orange" stroke-width="2"  fill="none" id="theMotionPath"/>  <**circle** cx="10" cy="110" r="3" fill="orange" />  <**circle** cx="110" cy="10" r="3" fill="orange" />   **<!-- Here is a red circle which will be moved along the motion path. -->** <**circle** cx="" cy="" r="5" fill="red">  <**animateMotion** dur="6s" repeatCount="indefinite">  <**mpath xlink**:href="#theMotionPath"/>  </**animateMotion**>  </**circle**> </**svg**> |
|  |  |
|  |  |
|  |  |

svg有滤镜filter标签，如果我想用filter给path图加个滤镜（比如外阴影），然后做让外阴影像呼吸灯似的渐变动画，是不是不能用animate做？这种效果有什么好的实现方案么

我研究出来了 可以用的！需要 animate标签用xlink:href属性指定到要修改标签的id上