### PPT/VS/Edge宿主的“面向2D的.svg文件宿客”（字符语法主导）

表 6‑23 PPT/VS/Edge宿主的“面向2D的.svg文件宿客”（字符语法主导）

|  |  |  |
| --- | --- | --- |
| ASP.Net MVC架构的视图V的创新研发：“客户端的Edge浏览器宿主的Blazor宿主的视图V宿客”的C#语言思维  （本表聚焦PPT/VS可以编辑的HTML可以引用的.svg文件） | ASP.Net MVC架构的控制C的创新研发：“互动的Kestrel/IIS宿主的控制C宿客”的C#语言思维 | ASP.Net MVC架构的模型M的创新研发：“服务端的SS/ChatGPT/Qwen宿主的模型M宿客”的C#语言思维 |
| 基于XML规范的多媒体应用有XHTML、SVG/XAML、X3D等，可形成媒体的整体解决方案。本节将系统叙述XML二维图形动画应用--XAML的基本原理，为二维图形动画的具体应用奠定原理基础。建议结合5.2节所述，进行上机实践探究如下。   1. 理解SVG/XAML是XML二维图形动画媒体应用。 2. 掌握SVG常用元素的功能和用法。 3. 掌握SVG布局容器。 4. 掌握SVG坐标系变换。 5. 理解SVG路径(二维形状定义标准)，路径与规则形状的相互关系。 6. 理解SVG形状填涂为图形的机制(图形格式)。笔画与填充，单色、渐变色、图案，透明度。 7. 理解SVG效果。 8. 掌握SVG动画。 9. 理解SVG屏蔽/剪切路径。 10. 掌握SVG与外部媒体引用。 11. 理解SVG资源与引用。 12. 掌握SVG超链接。 13. 了解XAML DOM，XAML结合Jscript、C#等编程。 | | |

1. **SVG：基于XML标准的2D应用**

二维图形及其动画是二维形象化信息传播的最基本媒体之一。二维图形动画编辑理论及其实现软件正日益发展并日趋标准化，XML在二维图形动画的具体应用--SVG语言则是描述二维图形动画的Web标准。

SVG 是Scalable Vector Graphics的首字母缩写，中文意译为可伸缩矢量图形，是一个描述二维图形动画的标记语言，当前最高版本是1.0。SVG二维图形动画技术拥有当前流行的二维图形动画（如Macromedia 公司的Flash、Microsoft公司的VML等）的基本功能，还具有可交换性、交互性、可扩展性、易维护性等优点，可完美结合HTML、脚本语言等实现强大的Web多媒体应用，奠定了现代教学技术领域中二维图形信息传播的基础。SVG功能强大，甚至具备网页技术平台的一些特点，但目前网页技术领域大多还是以HTML为技术平台，SVG则注重于结合HTML解决二维应用的问题。

SVG 和HTML在语法、功能等众多方面都具有相似性，只不过HTML偏重于文本理念，而SVG偏重于图形理念，将SVG与HTML对比理解，是掌握SVG的捷径。

该部分将主要以代码的方式叙述SVG，SVG功能强大、元素特性丰富，我们应该注重掌握SVG的基本原理，并结合Flash软件（参见第七章）、参考附录4中的SVG元素特性进行实践锻炼。

1. **SVG文档的基本结构**

SVG本质上是一种以XML为规范的标记语言，该标记语言编写并形成SVG文档。一个SVG文档往往对应一个计算机文件，文件扩展名是".svg" ，压缩文件的扩展名是".svgz"， MIME类型是 " image/svg+xml "。

下面是一个简单SVG文档的示例，反映了SVG文档的基本结构：

|  |
| --- |
| 1. <?xml version="1.0"?> **<!-- XML声明-->** 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> **<!--该语句声明和引用了SVG架构-->** 4. <svg width="400" height="300" xmlns="http://www.w3.org/2000/svg"> **<!--根元素<svg>及其名称空间-->** 5. <rect x="100" y="50" width="200" height="100"/> 6. </svg> |

SVG是一个Web标准，当前最高版本为1.0，是XML在二维图形动画领域的一个具体应用，所以SVG完全遵守XML规范，同时具有公共的预定义的SVG架构（而XML是自定义架构），掌握SVG的关键就是掌握SVG架构中预定义的元素及其特性的功能和用法，是该部分将要叙述的主要内容，SVG常见元素及其特性的详解还可参看关于SVG的附录4。

1. **SVG文档的编辑和浏览**
2. **SVG文档浏览**

和XML文档的浏览类似（参见**Error! Reference source not found.**）。不过 IE 6.0当前不能直接浏览SVG文档，必须安装SVG浏览器（如Adobe公司的SVG Viewer），IE（主浏览器）调用SVG浏览器（插件）浏览、解析、显示SVG文档。由于Web上一般以HTML文档为平台，所以实际上，SVG文档往往作为一个外部媒体被HTML文档引用（参见四、），IE浏览HTML文档时将调用SVG浏览器浏览被引用的SVG文档。

1. **SVG文档编辑**

从语言的角度看，SVG是一种遵守XML规范的标记语言。

从信息的角度看，SVG生成的是一种文本文档，SVG文档包含内容、样式和功能三种信息。

编辑SVG文档时，我们通过SVG元素直接生成文本、二维形状或引用外部媒体来获得文档内容；为文档内容进行布局、赋予样式（填涂和滤镜等）而获得屏幕显示效果；将图形的特性随时间变化而获得动画效果；为文档内容赋予超链接、结合脚本语言编程而获得交互功能；从而实现强大的SVG应用。

SVG文档本质上是文本文档，所以原则上可以使用任何文本编辑器（如Windows自带的记事本程序）来编辑SVG文档。但为了提高编辑效率，一般使用原生SVG文档编辑软件进行编辑（如美国Jasc公司的WebDraw），或者使用综合图形编辑软件进行编辑并输出为SVG格式（如Adobe公司的Illustrator， Macromedia公司的Flash动画也可转换为SVG，参见第七章）。

1. **SVG路径—二维形状定义标准**

SVG使用<path>(路径元素)来生成复杂的曲线或图形轮廓，通过进一步处理（如填充、笔画等）而获得SVG复杂图形效果。

注：数学意义上，基本形状元素等于生成相同基本形状的<path>。

路径（Path）是二维形状定义的标准。任何图形的轮廓（又称为形状）都可理解为是由一条或一组曲线进行描述的，而任何曲线又都是通过一组点进行定义的。path元素的实质就是定义了一组点，每两点之间通过不同的命令而进行不同方式的联接而形成复杂的曲线或闭合的图形轮廓，所以路径的机制是图形形状定义的根本，任何图形创作时图形的形状都必须和路径关联起来。

下面的示例使用<path>生成了一个圆角矩形(参见示例文件Svg\_RoundRect.svg)：

|  |
| --- |
| 1. <?xml version="1.0"?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg width="200" height="150"> 5. <path d="M130 50 C141.046 50 150 58.9544 150 70 L150 80 C150 91.0456 141.046 100 130 100 L70 100 C58.9544 100 50 91.0456 50 80 L50 70 C50 58.9544 58.9544 50 70 50 z"/> 6. </svg> |

示例文件Svg\_RoundRect.svg的浏览结果如图 6.3‑70所示：

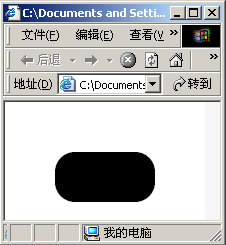


图 6.3‑70. 示例文件Svg\_RoundRect.svg的浏览结果

<path>拥有一个d="(path data)" 属性，属性值由命令和坐标数据组成。下面是关于d="(path data)"的句法说明:

1. 句法约定：使用了<path>来生成SVG图形时往往意味着大量的数据而使文件增大，为了尽量减小文件的大小，d="(path data)"句法设计为所有的命令用一个字符表示（例如：M或m是moveto的缩写）。大写字符意味着紧跟命令后面的坐标数据（即命令的参数）是绝对坐标值，小写字符则意味着紧跟命令后面的坐标数据（即命令的参数）是相对坐标值）。
2. M x y或m x y ："moveto"的缩写。在给定的坐标(x,y)处开始一个新的当前点，意味着一个新路径描述的开始，(x,y)成为路径描述的起始坐标。一个d="(path data)" 属性中的path data 必须以一个M x y或m x y开始。
3. Z或z："closepath"的缩写,该命令没有参数。自动从当前点画一条直线到起始点（M x y或m x y 命令中的(x,y)坐标）,从而结束路径的描述并获得闭合的形状。
4. L x y或l x y："lineto"的缩写。从当前点画一条直线到给定点(x,y)，命令结束后给定点(x,y)成为当前点。
5. C x1 y1 x2 y2 x y或c x1 y1 x2 y2 x y："cubic Bezier curve"的缩写。从当前点画一条bezier曲线到点(x,y)。该bezier曲线以(x1,y1)为曲线开始的控制点和(x2,y2)为曲线结束的控制点。该命令结束后，点(x,y)成为当前点。

注：贝齐尔曲线由三点组合定义而成的，其中只有一个点在曲线上，该点处曲线的曲率和方向则由另外两个点控制。从而获得任意曲线效果。

1. **SVG基本形状**

SVG基本形状基本形状是指数学规律上较简单的曲线或图形轮廓，如直线、矩形、圆等。基本形状是进一步处理（如填充、笔画等）而获得SVG图形效果的基础。SVG包含下列常用的基本形状元素:<rect>、<circle>、<ellipse>、<line>等，可参见关于SVG的附录4进行。使用基本形状元素可快捷获得规则形状，并减小文件。

思考：<path>与基本形状元素的关系。

1. **SVG文本**

在SVG文档中需要使用文本时，可以使用<text>生成文本并根据文本布局特性进行布局。SVG中的文本可以看作是具有文本布局特性的形状。下面的示例生成一行文本，可保存为text.svg：

|  |
| --- |
| 1. <?xml version="1.0" ?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg xmlns="http://www.w3.org/2000/svg" width="200" height="150"> 5. <text x="40px" y="70px">网页技术</text> 6. </svg> |

text.svg的浏览结果如图 6.3‑71所示：



图 6.3‑71. text.svg的浏览结果

每个<text>定义一个文本串 。SVG 1.0不支持文本的自动换行，要得到多行文本必须使用多重<text>。

注：SVG中的文本使用和其他图形元素一样的渲染方法，所以所有适用于形状元素的特点，也同样适用于text元素。

SVG支持CSS2.0中描述的所有文本布局特性，如文字书写方向，文本对齐方式等，此外，SVG还支持沿路径布局文本等更多的布局特性，可参见关于SVG的附录4进行。

图形软件主要用来编辑图形，因此文本排版功能远不及文本软件（如HTML），而主要强调文本的图形化视觉效果。

思考：基本形状、路径、文本的联系。

1. **SVG与CSS**

SVG主要使用CSS样式机制来样式化SVG文档中的形状元素的渲染效果，这和在HTML文档中使用CSS来样式化HTML文档的原理是相同的。和XHMTL文档使用CSS样式类似，SVG支持外部CSS样式样式引用、内部CSS样式单和在线样式及它们的继承和层叠机制，而且语法完全相同，只是因为SVG和HTML不是完全相同目的的应用而对CSS的属性要求有所不同而已，可参看关于CSS的附录3。

注：样式技术的使用需要SVG浏览器的支持，Adobe SVG View 3.0支持CSS2.0标准。SVG CSS属性与HTML属性范围有少许不同，如HTML中每一个元素都是矩形的，所以width，height是CSS属性赋值，而SVG是为二维图形设计的，每个元素不再是矩形，所以width，height不再作为CSS属性处理，而作为一些元素（如矩形rect的特性处理）。此外因为图形特点的需要，SVG中CSS有许多扩展，如支持Opacity、Mask、Filter等，可参看附录SVG参考的CSS部分。

下面是一个在线样式的示例，可保存为inlineCSS.svg：

|  |
| --- |
| 1. <?xml version="1.0" ?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg xmlns="http://www.w3.org/2000/svg" width="200" height="150"> 5. <text x="40" y="70" style="font-style:italic">网页技术</text> 6. </svg> |

inlineCSS.svg的浏览结果如图 6.3‑72所示：



图 6.3‑72. inlineCSS.svg的浏览结果

外部样式、内部样式、类、继承、层叠等功能请自行上机实验。

注：对于SVG的每一个CSS样式属性， 都设计有一个相对应的样式相关的特性，该样式相关的特性可实现与CSS属性相同的功能。例如SVG有一个CSS“font-style”属性用来定义文本的样式，就会有一个相应的同名的样式相关的特性(即font-style=”…”)可以用来指定文本的样式值而获得与CSS属性相同的样式结果。因为CSS支持类、继承和层叠的机制，所以使用CSS来样式化SVG图形元素比直接使用图形元素的样式相关的特性来样式化SVG图形元素具有更多的优越性。

1. **SVG形状的布局**

一个复杂的SVG文档将由许多SVG图形元素(如基本形状元素等)组成，这些元素必须有一个空间布局的规则，这是通过二维坐标系及其变换来实现的。

注：SVG以二维图形编辑为主，布局机制更为简捷，完全以坐标变换的方式来实现。与HTML文档布局和HTML文档中元素占位机制比较（即文本布局为主的机制），SVG文档中元素布局时没有元素内容，元素矩形的布局概念，所以也没有填充（padding）、边界（border）、边白（margin）的概念。但元素的父子关系和容器包容关系在SVG布局中仍然适用。

1. **SVG初始坐标系**

和HTML文档编辑相似，开始编辑一个SVG文档时，首先必须定义一个确定大小的矩形区域，这个矩形区域称为视区，编辑的SVG元素只有落在这个视区内才是可以看见的，同时，在这个视区内形成一个初始坐标系，从而开始为每一个SVG图形元素进行布局，可以对初始坐标系进行变换而获得新的布局坐标系。

SVG初始坐标系和HTML初始坐标系相似。该初始坐标系以“pixel”为长度单位，坐标原点位于视区的左上顶角，X轴正向朝右，Y轴正向朝下。但SVG文档一般作为外部对象引用而嵌入在HTML文档中，所以SVG视区的位置和大小将受到主文档（HTML文档）所提供的主视区的影响，主文档为SVG对象提供的主视区的位置默认是流式定位到嵌入的SVG对象时的位置，提供的主视区的大小由为SVG对象赋予的width和height CSS属性值决定。如图 6.3‑73是SVG嵌入在HTML文档中的视区和坐标系示意图。



XHTML初始坐标系

SVG嵌入在XHTML文档中，所以受到XHTML文档布局的影响，但拥有自己独立的视区和坐标系

XHTML文档的视区（默认属性）与浏览器的矩形区域等同。

(0,0)

SVG文档

(0,0)

x

x

(0,0)

y

y

图 6.3‑73. SVG嵌入在HTML文档中的视区和坐标系示意图

SVG文档以外部对象的方式嵌入在HTML文档中，所以SVG对象的位置和视区的大小将受到主文档（HTML文档）布局的影响，SVG文档默认的自定义视区大小的长、宽值都是100%，即等于主文档提供的视区大小，但SVG默认的100%的长宽视区大小可以通过为根元素<svg>的width和height 特性赋值而改变，当自定义的SVG的视区大于主文档提供的视区时，只有落在主文档提供的视区内的SVG内容是可见的。总之，因为元素本身长宽定义过大，或SVG自定义的视区过大等因素而超出了主文档提供的视区大小的SVG内容都将被裁剪。不过SVG浏览器一般提供平移SVG视区和推远拉近SVG视区的界面，从而提供了用户可以浏览到落在视区外的SVG内容的途径。

虽然嵌入在HTML文档中的SVG文档的视区位置和大小都受到主文档的控制，但SVG文档拥有自己独立的视区和坐标系，该SVG文档中所有元素的布局都是在SVG文档自己独立的视区和坐标系中进行的。

注：HTML文档布局采用以文本布局为主的机制，HTML文档直接在浏览器窗口中显示，并且默认视区的大小等于浏览器窗口的大小，默认采用的是流式定位的布局，这样当视区（即浏览器窗口）大小改变时文档内容也将重新流布而便于浏览。SVG文档布局以图形布局为特性，而且一般嵌入在HTML文档中，SVG文档不采用流式定位的布局，而是采用绝对定位，所以当SVG视区大小改变时SVG内容不会重新流布。

思考：SVG文档独立运行时的视区、坐标系与SVG浏览器矩形窗口的关系。

1. **SVG坐标系变换**

随着编辑新图形元素时布局的需要，可以以初始坐标系为起点不断进行变换，每次变换后获得的新坐标系称为当前坐标系，每一时刻当前坐标系只有一个，当前坐标系与其变换时所依据的坐标系呈父子关系，初始坐标系则是所有变换后的坐标系的根。坐标系变换是累积的，也就是说每次变换都是在父坐标系的基础上进行的，而父坐标系又是在自身父坐标系的基础上变换而来的，一直追溯到根坐标系—--初始坐标系。一个元素的编辑总是在当前坐标系上进行布局的，这样由于元素布局所依据的坐标系的变换而获得了元素的布局效果。

注：一些SVG元素（如<script>等非图形元素）没有坐标系变换的概念。

思考：SVG坐标系及其变换与HTML坐标系的异同。

SVG二维坐标系常用的变换有平移（translate）、缩放（scale）、旋转（rotate）三种，通过图形元素的”transform”特性指定。“transform”特性值是这三种变换的组合列表，各种变换按次序提供，并以逗号（或白空）分隔。如下所示。

|  |
| --- |
| 1. <g transform="translate(40,30) scale(2,1) rotate(45) > 2. <!—这里嵌套图形元素--> 3. </g> |

变换列表也可分开定义，上例的功能效果等同于如下的表示(<g>是将其所有子元素成组的意思，可参见“附录四 SVG常用元素参考”)。

|  |
| --- |
| 1. <g transform="translate(40,30)"> 2. <g transform="scale(2,1)"> 3. <g transform="rotate(45)"> 4. <!—这里嵌套图形元素--> 5. </g> 6. </g> 7. </g> |

各种变换类型的含义如下：

* 1. translate(<tx> [<ty>])指定一个以tx和ty值进行的坐标平移。tx和ty的默认值为0。
  2. scale(<sx> [<sy>])指定一个以sx值进行x轴的缩放和sy值进行y轴的缩放。如果<sy> 没有提供，默认sy=sx。
  3. rotate(<rotate-angle> [<cx> <cy>])指定一个绕点(<cx>, <cy>)进行<rotate-angle>角度值的旋转。如果<cx> 和<cy> 没有提供，默认围绕当前坐标系原点进行旋转。

下面的示例显示的是一个没有坐标变换的文档，SVG场景指定在初始坐标系中：

|  |
| --- |
| 1. <?xml version="1.0" ?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg xmlns="http://www.w3.org/2000/svg" width="200" height="150"> 5. <rect x="20" y="10" width="80" height="60"/> 6. </svg> |

下面的示例通过指定transform="scale(2,1) translate(30,50)"建立了一个新的用户坐标系。新的坐标系原点在父坐标系（这里是初始坐标系）的(30,50)处,并且新坐标系的x转进行了2倍的放大：

|  |
| --- |
| 1. <?xml version="1.0" ?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg xmlns="http://www.w3.org/2000/svg" width="200" height="150"> 5. <rect x="10" y="10" width="40" height="60" transform="scale(2,1) translate(30,50)"/> 6. </svg> |

图 6.3‑74是变换前和变换后的运行结果比较：

图 6.3‑74. 变换前和变换后的运行结果比较

新编辑的元素总是在当前坐标系上进行布局的，也就是说坐标系变换总是在任何与坐标相关的值处理之前应用，所以<rect x="10" y="10" width="40" height="60" transform="scale(2,1)"/>等同于<rect x="20" y="10" width="80" height="60"/>的效果。

注：平移、缩放、旋转等所有坐标变换都可以以矩阵的数学方式来表示，如transform=”translate(50,50)” 等同于transform="matrix(1 0 0 1 50 50)"。

1. **关于层**

CSS的形状是绝对定位。布局过程中当两个图形元素的内容的位置空间发生重叠时，文档中后渲染的图形元素的内容将覆盖在前面渲染的图形元素的内容之上。

注：SVG1.0没有层的概念，也就是说布局过程中当两个图形元素的内容的位置空间发生重叠时，后渲染的图形元素的内容总是覆盖在前面渲染的图形元素的内容之上。SVG 1.2将引入z-index属性以支持独立于渲染顺序的层覆盖机制。

1. **SVG形状的填涂**

SVG形状元素（包括path 、text和基本形状元素）生成各种形状后，再对形状进行填涂（默认布满重复整个形状），就可获得想要的图形效果。填涂可以分别对形状的笔画 (形状的的轮廓线)和填充（形状的内部）进行，填涂的内容可以是单色，渐变色（线性渐变或辐射渐变）或图案（矢量图或点阵图）。填涂的途径则是通过为SVG形状赋予CSS属性来实现的。

1. **SVG形状的填涂：笔画和填充**

一个SVG形状可分为笔画和填充两部分分别进行填涂处理，笔画部分是指形状的轮廓线，填充部分是指形状轮廓线包围的内部区域。下面是对形状的笔画和填充进行分别处理的一个示例，可保存为strokeAndFill.svg：

|  |
| --- |
| 1. **<**?xml version="1.0"?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg width="200" height="150"> 5. <ellipse cx="100" cy="70" rx="60" ry="40" 6. style="stroke:rgb(0,255,0); 7. stroke-width:10; 8. fill:rgb(192,192,255);"/> 9. </svg> |

strokeAndFill.svg的浏览结果如图 6.3‑75所示：



图 6.3‑75. strokeAndFill.svg的浏览结果

虽然笔画部分和填充部分一些特性不同（如笔画可以赋予宽度等，而填充不行），但两部分的填涂处理机制是完全一样的。

1. **SVG形状的填涂：单色**

下面是为形状的笔画和填充部分分别填涂单色的示例：

|  |
| --- |
| 1. <?xml version="1.0"?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg width="200" height="150"> 5. <ellipse cx="100" cy="70" rx="60" ry="40" 6. style="stroke:rgb(0,255,0); 7. stroke-width:10; 8. fill:rgb(192,192,255);"/> 9. </svg> |

1. **SVG形状的填涂：渐变色**

下面是为形状的笔画部分填涂单色，为形状的填充部分填涂渐变色的示例，可保存为gradient.svg：

|  |
| --- |
| 1. <?xml version="1.0"?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg width="200" height="150"> 5. <defs> 6. <linearGradient id="Gradient01"> 7. <stop offset="20%" stop-color="rgb(0,255,0)" /> 8. <stop offset="50%" stop-color="rgb(192,192,255)" /> 9. <stop offset="70%" stop-color="rgb(0,0,255)" /> 10. </linearGradient> 11. </defs> 12. <ellipse cx="100" cy="70" rx="60" ry="40" 13. style="stroke:rgb(0,255,0);stroke-width:10;fill:url(#Gradient01)"/> 14. </svg> |

gradient.svg的浏览结果如图 6.3‑76所示：

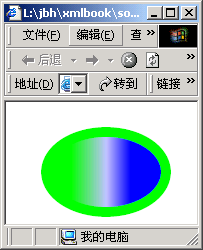


图 6.3‑76. gradient.css的浏览结果

示例中<defs>元素是一个容器元素，包含在<defs>元素中的所有内容不直接被渲染，而是等到被其他元素引用后才渲染。由于SVG浏览器在解析运行SVG文档时是按照文档中内容的顺序进行的，所以<defs> 元素（及其内容）应该放置在SVG文档元素集部分的前面，以便后面出现的元素引用。

1. **SVG形状的填涂：图案**

图案（Pattern）可以是矢量图案（如填充SVG代码），也可以是图像图案（Texture，又称为材图、纹理），下面是为图形的填充部分填涂图案的示例，可保存为pattern.svg：

|  |
| --- |
| 1. <?xml version="1.0" standalone="no"?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg width="200" height="150"> 5. <defs> 6. <pattern id="Argyle" width="50" height="50" patternUnits="userSpaceOnUse" patternContentUnits="userSpaceOnUse"> **<!--<pattern>定义了一个图案，可以被其它元素引用生成填涂效果。patternUnits等特性指定了图案填涂时的坐标系等特征，以获得相应的填涂方式-->** 7. <g> 8. <polygon style="stroke:rgb(112,97,66);stroke-width:1;stroke-opacity:1;stroke-miterlimit:30;fill:rgb(215,207,189); 9. fill-opacity:1" points="25,10 34,25 25,40 16,25"/> 10. <polygon style="stroke:rgb(112,97,66);stroke-width:1;stroke-opacity:1;stroke-miterlimit:30;fill:rgb(215,207,189); 11. fill-opacity:1" points="50,10 59,25 50,40 41,25"/> 12. <polygon style="stroke:rgb(112,97,66);stroke-width:1;stroke-opacity:1;stroke-miterlimit:30;fill:rgb(215,207,189); 13. fill-opacity:1" points="0,10 9,25 0,40 -9,25"/> 14. <polygon style="stroke:rgb(52,48,40);stroke-width:1;stroke-opacity:1;stroke-miterlimit:30;fill:rgb(172,152,112); 15. fill-opacity:1" points="11,0 21,25 11,50 1,25"/> 16. <polygon style="stroke:rgb(52,48,40);stroke-width:1;stroke-opacity:1;stroke-miterlimit:30;fill:rgb(172,152,112); 17. fill-opacity:1" points="39,0 49,25 39,50 29,25"/> 18. <polygon style="stroke:rgb(112,97,66);stroke-width:1;stroke-opacity:1;stroke-miterlimit:30;fill:rgb(112,97,66); 19. fill-opacity:1" points="20,0 30,0 25,9"/> 20. <polygon style="stroke:rgb(112,97,66);stroke-width:1;stroke-opacity:1;stroke-miterlimit:30;fill:rgb(112,97,66); 21. fill-opacity:1" points="20,50 25,41 30,50"/> 22. </g> 23. </pattern> 24. </defs> 25. <ellipse cx="100" cy="70" rx="60" ry="40" 26. style="fill:url(#Argyle);stroke:rgb(0,0,0);stroke-width:10"/>  **<!--<ellipse>的fill样式属性通过id引用了前面已定义的<pattern>-->** 27. </svg> |

pattern.svg的浏览结果如图 6.3‑77所示：

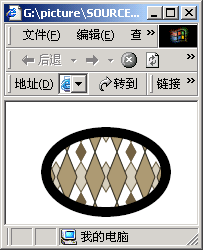


图 6.3‑77. pattern.svg的浏览结果

示例中<pattern>置于<defs>中，用来预定义一个图形对象，然后可以被其他图形元素的['fill'](file:///I:\private\books\computer\web\refSec\SVGRefer\painting.html#FillProperty) 和/或 ['stroke'](file:///I:\private\books\computer\web\refSec\SVGRefer\painting.html#StrokeProperty) 属性引用并在该图形元素的填涂区域以固定的间隔重复铺放而获得图案效果。

1. **SVG滤镜**

SVG支持通过滤镜元素来获得滤镜效果。SVG滤镜本质上是可应用于SVG图形而产生图形修饰效果的一系列图形操作代码。滤镜原理如图 6.3‑78所示:

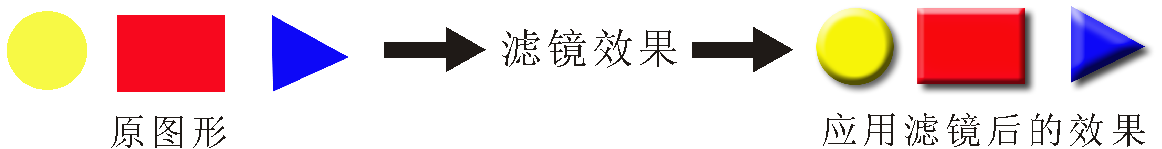


图 6.3‑78. 滤镜原理

下面是为SVG图形应用滤镜效果的一个示例，可保存为filter.svg：

|  |
| --- |
| 1. <?xml version="1.0"?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"   "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">   1. <svg width="200" height="150" xmlns="http://www.w3.org/2000/svg"> 2. <defs> 3. <filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="200" height="120"> **<!--<filter>嵌套 一些滤镜元元素定义滤镜-->** 4. <feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur"/> 5. <feOffset in="blur" dx="4" dy="4" result="offsetBlur"/> 6. <feSpecularLighting in="blur" surfaceScale="5" specularConstant=".75" specularExponent="20" lighting-color="#bbbbbb" result="specOut"> 7. <fePointLight x="-5000" y="-10000" z="20000"/> 8. </feSpecularLighting> 9. <feComposite in="specOut" in2="SourceAlpha" operator="in" result="specOut"/> 10. <feComposite in="SourceGraphic" in2="specOut" operator="arithmetic" k1="0" k2="1" k3="1" k4="0" result="litPaint"/> 11. <feMerge> 12. <feMergeNode in="offsetBlur"/> 13. <feMergeNode in="litPaint"/> 14. </feMerge> 15. </filter> 16. </defs> 17. <g filter="url(#MyFilter)" > **<!--<g>的filter属性通过id引用前面定义的滤镜元素, <filter>嵌套一些滤镜元素定义滤镜-->** 18. <path fill="#D90000" d="M60,80 C30,80 30,40 60,40 L140,40 C170,40 170,80 140,80 z" /> 19. <g fill="#FFFFFF" stroke="black" font-size="45" font-family="Verdana" > 20. <text x="52" y="76">SVG</text> 21. </g> 22. </g> 23. </svg> |

filter.svg的浏览结果如图 6.3‑79所示，获得了炫目的视觉效果：



图 6.3‑79. filter.svg的浏览结果

滤镜通过<filter>嵌套一个或一套<filterPrimitive>来定义，每个<filterPrimitive>执行一个基本图形效果操作（如模糊或光效等）,<filter>一般置于<defs>中，<filter>的内容从来不直接渲染，而是等到在欲使用滤镜效果的图形元素中通过”filter”属性引用后才渲染。

SVG滤镜的功能主要依靠于SVG浏览器所支持的大量预定义的<filterPrimitive>的功能，更多的<filterPrimitive>可参看SVG浏览器的功能说明。

注：滤镜是获得图形图像渲染效果的最高境界，一般只在点阵图处理时使用滤镜（如Adobe著名的图像处理软件photoshop的滤镜功能），使用滤镜处理后的点阵图文件往往不利于压缩处理，所以文件急剧增大而不利于网络传输，但SVG图形的滤镜效果是以代码的形式由客户端SVG浏览器实现的，对文件的网络传输几乎没有大的影响。

1. **SVG动画**

动画分为逐关键帧动画和插变帧动画。逐关键帧动画是随时间逐帧显示然后消失媒体内容的结果，插变帧动画是随时间改变图形元素属性的结果。在此主要叙述插变帧动画。

SVG支持随时间改变图形元素的特性或CSS属性而获得动画效果。动画一般通过动画元素来定义。SVG中常用的动画元素有<animate>,<animateMotion>,<animateTransform>等。SVG动画可以实现颜色变化、形状变化、布局变化等效果，由于手工编写代码编辑动画工作繁重，是不可取的，所以一般通过可视化操作软件来实现，所以在此只叙述SVG动画的基本原理和代码结构，具体的动画编辑操作参见第七章。SVG动画生成的基本步骤如下：

1. 定义一个动画元素

SVG动画一般是通过SVG动画元素来实现的，所以首先要定义一个动画元素。

1. 指明动画目标元素

动画元素操作动画目标元素使动画目标元素产生动画效果，默认的动画目标元素是该动画元素的父元素，然而可以通过动画元素的一个xlink:href="<uri>" 特性更灵活地指定动画目标元素。

1. 指明动画目标元素的动画目标特性或CSS属性

通过动画元素的一个attributeName=”attributeName”特性指定动画目标元素的动画目标特性或CSS属性。

1. 控制动画的计时

下面是通用于任何动画元素的控制计时的特性列表。这些动画元素的特性可用来控制动画的开始和结束时间，指定动画是否重复运行，指明动画一旦结束后是否保持动画结束时的状态等。

* 1. begin= Clock-value特性指定动画的开始时间。默认为0s。
  2. dur= Clock-value | "indefinite"指定动画的简单持续时间，默认为"indefinite",表示动画将无限持续。
  3. end = Clock-value指定动画的结束时间。
  4. repeatCount=numeric value | "indefinite"指定动画的重复次数.默认为"indefinite",表明动画无限重复。
  5. fill = "freeze" | "remove"指明动画一旦结束后是否保持动画结束时的状态。默认为”remove”,表明动画一旦结束后将不保持动画结束时的状态。

1. 定义动画的插值方式

SVG动画的实质是图形元素的特性或CSS属性随时间的变化，可通过动画元素的calcMode = "discrete | linear | paced | spline"特性来指定这些变化的插值方式来获得更丰富的动画效果。

* 1. calcMode="discrete"指定动画将无任何插值地从一个状态跳到另一个状态。
  2. calcMode="linear"（默认）指定动画从一个状态线性插值地跳到另一个状态。如果动画原理不可能支持线性插值（如可见性等非数值性的特性随时间的动画等），则使用calcMode="discrete"。
  3. calcMode="paced"指定一个平均的插值方式进行动画。是 'animateMotion'动画元素的默认插值方式。
  4. calcMode="spline"称为样条插值。是对动画具有最强控制功能的插值方式。可参见附录SVG参考部分。
  5. from ="<value>"特性指定动画的开始值。
  6. to = "<value>"特性指定动画的结束值。

1. 指定重复的动画结果是否累积

动画元素的accumulate = "none | sum" 特性指定如果一个动画是重复多次的时候，每次重复时特性或CSS属性值是否在前一次动画的基础上进行累积。

* 1. accumulate=sum指定要进行累积。
  2. accumulate=none 指定不要进行累积。默认值。

下面是对矩形的填充色进行动画的一个简单动画示例(示例文件svg\_animation.svg)：

|  |
| --- |
| 1. <?xml version="1.0" standalone="no"?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg> 5. <ellipse cx="100.5" cy="78.5" rx="57.5" ry="36.5" style="fill:rgb(202,255,195)；stroke:rgb(0,0,128)；stroke-width:1"> 6. <animate attributeName="fill" begin="0s" dur="3s" fill="freeze" 7. calcMode="linear" from="rgb(202,255,195)" to="rgb(152,44,54)" repeatCount="indefinite"> 8. </animate> 9. </ellipse> 10. </svg> |

注：使用动画元素获得动画是简单快捷的，使用脚本和DOM编程则是获得动画的另一种高级途径。脚本和DOM编程将在第四章详述。

1. **SVG与外部媒体引用**

SVG自身主要用来生成文本和图形，但在SVG文档中可以通过外部媒体引用的方式引用其他已保存为文件的媒体，如图像、视频、音频等。下面是通过<image>引用外部图像的一个示例(示例文件svg\_referencImage.svg)：

|  |
| --- |
| 1. <?xml version="1.0" standalone="no"?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg width="200" height="150"> 5. <image x="80" y="80" width="160" height="120" 6. xlink:href="art.jpg"/> 7. </svg> |

注：SVG主要用来生成矢量图形，实际上也可以用来表示点阵图形，但在SVG中直接以文本形式来表示点阵图不是SVG的优势，所以点阵图在SVG文档中一般采用外部媒体引用的方式来嵌入。对外部媒体的引用需要SVG浏览器的支持，点阵图方面一般都支持.jpg,. gif等文件格式。与Internet Explorer强大的外部媒体引用支持进行比较，Adobe SVG Viewer 3.0尚不支持音视频媒体的引用和对外部SVG文档的引用、目前也不支持外部对象的应用。所以要在SVG应用中使用音频，目前只能在嵌入了该SVG文档的HTML文档中使用音频而间接地为SVG应用获得音频效果。

1. **SVG与超链接**

作为Web二维图形应用，SVG支持超链接的功能，这通过<a>来实现，下面是一个示例(示例文件svg\_hyperLink.svg)：

|  |
| --- |
| 1. <?xml version="1.0"?> 2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" 3. "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"> 4. <svg width="200" height="150"> 5. <a xlink:href="http://www.w3.org" xlink:title="Web联盟"> **<!--SVG完全遵守XML规范，超链接属性使用xlink标准格式。-->** 6. <ellipse cx="93.9" cy="72.2" rx="50.1" ry="28.8" 7. style="fill:rgb(192,192,255);stroke:rgb(0,0,128);stroke-width:1"/> 8. </a> 9. </svg> |