数据可视化 SVG

刘军　liujun

目录
content

邂逅SVG

- **什么是SVG？维基百科介绍：**
  - SVG 全称为（Scalable Vector Graphics），即可缩放矢量图形。（矢量定义：既有大小又有方向的量。在物理学中称作矢量，如一个带箭头线段：长度表示大小，箭头表示方向；在数学中称作向量。在计算机中，矢量图可无限放大而不变形）
  - SVG 是一种基于XML格式的矢量图，主要用于定义二维图形，支持交互和动画。
  - SVG 规范是万维网联盟(W3C) 自 1998 年以来开发的标准。
  - SVG 图像可在不损失质量的情况下按比例缩放，并支持压缩。
  - 基于XML的SVG可轻松的用文本编辑器或矢量图形编辑器创建和编辑，并可以直接在浏览器显示。
- **SVG对浏览器的兼容性：**

Chrome	Edge *	Safari	Firefox	Opera	IE
	[3] 12-18	3.1	2		6-8
4-103	79-103	3.2-15.5	3-103	10-89	[2 3] 9-10
104	104	15.6	104	90	[2 3] 11
105-107		16.0-TP	105-106		

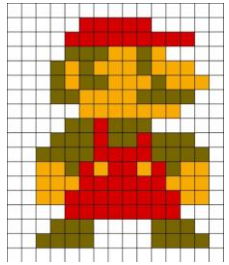
Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Bai Brow
						2.1-2.3			
						[1] 3-4.3			
	3.2-15.5	4-17.0		12-12.1		4.4-4.4.4			
104	15.6	18.0	all	64	12.12	104	101	10.4	7.1
	16.0								

SVG 历史

- SVG1.x 版本
  - SVG 是 W3C SVG工作组于 1998 年开始开发，而 SVG 1.0于 2001 年 9 月 4 日成为W3C 推荐的标准。
  - SVG 1.1 于 2003 年 1 月 14 日成为 W3C 推荐的标准。 该版本增加了模块化规范的内容。除此之外，1.1 和 1.0 几乎没有区别。
  - SVG Tiny 1.2 于 2008 年 12 月 22 日成为 W3C 推荐标准，主要是为性能低的小设备生成图形，但是后来被 SVG 2 所弃用了。
  - SVG 1.1 第二版 于 2011 年 8 月 16 日发布，这次只是更新了勘误表和说明，并没有增加新功能 。
- SVG 2.0 版本（推荐）
  - SVG 2.0于2016 年 9 月 15 日成为W3C 候选推荐标准，最新草案于2020年5月26日发布。
  - SVG2.x Change From SVG1.x（ https://www.w3.org/TR/SVG/changes.html ），比如：
    - ✓ Removed the **baseProfile** and **version** attributes from the **'svg'** element.
    - ✓ Added the ability to use **'auto'** for the width and height attributes on **'image'** .
    - ✓ Added **'lang'** attribute on **'desc'** and **'title'** elements.
    - ✓ Removed the **'xlink:type'** , **'xlink:role'** , **'xlink:arcrole'** , **'xlink:show'** and **'xlink:actuate'** attributes.
    - ✓ Deprecated the **'xlink:href'** attribute in favor of using **'href'** without a namespace.

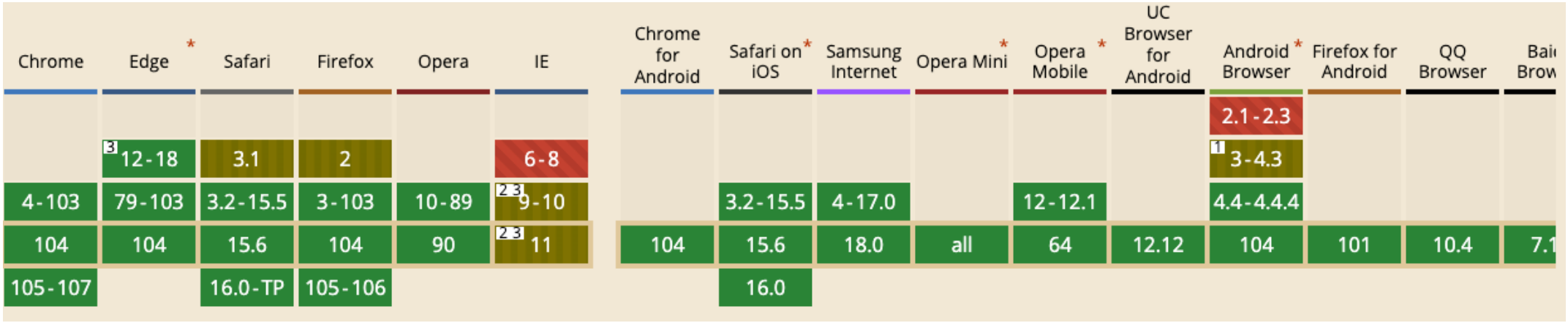
SVG 优点

- **扩展好**：矢量图像在浏览器中放大缩小不会失真，可被许多设备和浏览器中使用。而光栅图像（PNG 、JPG）放大缩小会失真。
  - 矢量图像是基于矢量的点、线、形状和数学公式来构建的图形，该图形是没有像素的，放大缩小是不会失真的。
  - 光栅图像是由像素点构建的图像——微小的彩色方块，大量像素点可以形成高清图像，比如照片。图像像素越多，质量越高。
- **灵活**：SVG是W3C开发的标准，可结合其它的语言和技术一起使用，包括 CSS、JavaScript、 HTML 和 SMIL 。SVG图像可以直接使用JS和CSS进行操作，使用时非常方便和灵活，因为SVG也是可集成到 DOM 中的。
- **可以动画**：SVG 图像可以使用 JS 、 CSS 和 SMIL 进行动画处理。对于 Web 开发人员来说非常的友好。
- **轻量级**：与其它格式相比，SVG 图像的尺寸非常小。根据图像的不同，PNG 图像质量可能是 SVG 图像的 50 倍。
- **可打印**：SVG 图像可以以任何分辨率打印，而不会损失图像质量。
- **利于SEO**：SVG 图像被搜索引擎索引。因此，SVG 图像非常适合 SEO（搜索引擎优化）目的。
- **可压缩**：与其它图像格式一样，SVG 文件支持压缩。
- **易于编辑**：只需一个文本编辑器就可以创建 SVG 图像。设计师通常会使用 Adobe Illustrator （AI）等矢量图形工具创建和编辑。



SVG 缺点

- **不适和高清图片制作**
  - SVG 格式非常适合用于徽标和图标（ICON）等 2D 图形，但不适用于高清图片，不适合进行像素级操作。
  - SVG 的图像无法显示与标准图像格式一样多的细节，因为它们是使用点和路径而不是像素来渲染的。
- **SVG 图像变得复杂时，加载会比较慢**
- **不完全扩平台**
  - 尽管 SVG 自 1998 年以来就已经存在，并得到了大多数现代浏览器（桌面和移动设备）的支持，但它不适用于 IE8 及更低版本的旧版浏览器。根据caniuse的数据，大约还有 5% 的用户在使用不支持 SVG 的浏览器。



SVG 应用场景

- **SVG应用场景有哪些？ 下面是一些保证 SVG 优于其他图像格式的应用场景：**
  - SVG 非常适合显示矢量徽标（Logo）、图标（ICON）和其他几何设计。
  - SVG 适合应用在需适配多种尺寸的屏幕上展示，因为SVG的扩展性更好。
  - 当需要创建简单的动画时，SVG 是一种理想的格式。
    - ✓ SVG 可以与 JS 交互来制作线条动画、过渡和其他复杂的动画。
    - ✓ SVG 可以与 CSS 动画交互，也可以使用自己内置的 SMIL 动画。
  - SVG 也非常适合制作各种图表（条形图、折线图、饼图、散点图等等），以及大屏可视化页面开发。

SVG 和 Canvas的区别

- 可扩展性：
  - SVG 是基于矢量的点、线、形状和数学公式来构建的图形，该图形是没有像素的，放大缩小不会失真。
  - Canvas 是由一个个像素点构成的图形，放大会使图形变得颗粒状和像素化（模糊）。
  - SVG可以在任何分辨率下以高质量的打印。Canvas 不适合在任意分辨率下打印。
- 渲染能力：
  - 当 SVG 很复杂时，它的渲染就会变得很慢，因为在很大程度上去使用 DOM 时，渲染会变得很慢。
  - Canvas 提供了高性能的渲染和更快的图形处理能力，例如：适合制作H5小游戏。
  - 当图像中具有大量元素时，SVG 文件的大小会增长得更快（导致DOM变得复杂），而Canvas并不会增加太多。
- 灵活度：
  - SVG 可以通过JavaScript 和 CSS 进行修改，用SVG来创建动画和制作特效非常方便。
  - Canvas只能通过JavaScript进行修改，创建动画得一帧帧重绘。
- 使用场景：
  - Canvas 主要用于游戏开发、绘制图形、复杂照片的合成，以及对图片进行像素级别的操作，如：取色器、复古照片。
  - SVG 非常适合显示矢量徽标（Logo）、图标（ICON）和其他几何设计。

初体验SVG

- **如何绘制一个SVG矢量图？ 绘制SVG矢量图常用有4种方式：**
  - 方式一：在一个单独的svg文件中绘制，svg文件可直接在浏览器预览或嵌入到HTML中使用（推荐）
  - 方式二：直接在HTML文件中使用svg元素来绘制（推荐）
  - 方式三：直接使用JavaScript代码来生成svg矢量图。
  - 方式四：使用AI（Adobe Illustractor）矢量绘图工具来绘制矢量图，并导出为svg文件（推荐）

```
<script>
  let xmlns = "http://www.w3.org/2000/svg";
  let boxWidth = 200;
  let boxHeight = 200;

  let svgEl = document.createElementNS(xmlns, "svg");
  let rectEl = document.createElementNS(xmlns, "rect");
  svgEl.setAttributeNS(null, "version", "1.1");
  svgEl.setAttributeNS(null, "width", boxWidth);
  svgEl.setAttributeNS(null, "height", boxHeight);

  rectEl.setAttributeNS(null, "width", 100);
  rectEl.setAttributeNS(null, "height", 40);
  rectEl.setAttributeNS(null, "fill", "red");

  svgEl.style.background = "#cdcd";

  svgEl.appendChild(rectEl);
  document.body.appendChild(svgEl);
</script>
```

- **SVG 初体验**
  - 第一步：新建一个 svg 文件，在文件第一行编写XML文件声明
  - 第二步：编写 一个<svg>元素，并给该元素添加如下属性：
    - ~~version~~： 指定使用svg的版本，值为 1.0 和 1.1，并没有 2.0。
    - ~~baseProfile~~：SVG 2 之前，version 和 baseProfile 属性用来验证和识别 SVG 版本。而SVG2后不推荐使用这两个属性了。
    - width / height ： 指定svg画布（视口）的宽和高，默认值分别为300和150，默认使用px单位。
    - xmlns：给svg元素帮定一个命名空间（http://www.w3.org/2000/svg）

      意味着这个 <svg> 标签和它的子元素都属于该命名空间下。
  - 第三步：在< svg >元素中添加 图形（比如：< rect >） 元素
  - 第四步：在浏览器直接预览 或 嵌入到 HTML中预览（嵌入HTML有6种方案）

The **version** attribute is used to indicate what specification a SVG document conforms to. It is only allowed on the root <svg> element. It is purely advisory and has no influence on rendering or processing.

While it is specified to accept any number, the only two valid choices are currently `1.0` and `1.1`.

XML 和 DTD 声明

- 由于 SVG 是一个 XML 文件格式。在编写XML文档时，通常是推荐编写XML声明。因为在 XML 1.0 中，XML 声明是可选的，推荐写但不是强制性。然而，在 XML 1.1 中，声明是强制性的，如果没有声明，则自动暗示该文档是 XML 1.0 文档。所以这里建议大家在编写SVG文件时也编写一个 XML 声明。
- SVG的XML声明格式：<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  - version 指定版本（必填）
  - encoding 指定XML文档的编码（可选，默认是UTF-8）
  - standalone：指定当前 XML 文档是否依赖于外部标记声明（可选，使用该属性时，需和DTD声明一起用才有意义）。
    - ✓ 默认为no：代表依赖外部标记声明
    - ✓ yes：代表依赖内部默认的标记声明
- SVG的文档类型声明（DTD），让解析器验证XML文件是否符合该规范，与HTML5文件的DTD声明类似。
  - XML中内部 DTD 声明（可选）
  - XML中外部 DTD 声明（可选）

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"

"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

SVG文档结构

- **SVG1.1文档结构：** https://www.w3.org/TR/SVG11/struct.html
  - 第一行：包含一个 XML 声明。由于 SVG 文件是一个 XML 格式的，它应包含一个 XML 声明。
  - 第二行：定义文档类型声明 (DTD)，这里依赖外部的 SVG1.1 文档类型，让解析器验证XML文件是否符合该规范。

    <!DOCTYPE svg PUBLIC “-//W3C//DTD SVG 1.1//EN” "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
- **SVG2.0文档结构：** https://www.w3.org/TR/SVG2/struct.html#Namespace
  - SVG2 version和baseProfile属性已删除，也不推荐写文档类型声明（DTD）。其中<desc>元素是用来描述该文件的。

SVG 1.1

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="4cm" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Four separate rectangles
  </desc>
    <rect x="0.5cm" y="0.5cm" width="2cm" height="1cm"/>
    <rect x="0.5cm" y="2cm" width="1cm" height="1.5cm"/>
    <rect x="3cm" y="0.5cm" width="1.5cm" height="2cm"/>
    <rect x="3.5cm" y="3cm" width="1cm" height="0.5cm"/>

  <!-- Show outline of canvas using 'rect' element -->
  <rect x=".01cm" y=".01cm" width="4.98cm" height="3.98cm"
        fill="none" stroke="blue" stroke-width=".02cm" />

</svg>
```

SVG2

```
<?xml version="1.0" standalone="no"?>
<svg width="5cm" height="4cm" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Four separate rectangles
  </desc>
    <rect x="0.5cm" y="0.5cm" width="2cm" height="1cm"/>
    <rect x="0.5cm" y="2cm" width="1cm" height="1.5cm"/>
    <rect x="3cm" y="0.5cm" width="1.5cm" height="2cm"/>
    <rect x="3.5cm" y="3cm" width="1cm" height="0.5cm"/>

  <!-- Show outline of viewport using 'rect' element -->
  <rect x=".01cm" y=".01cm" width="4.98cm" height="3.98cm"
        fill="none" stroke="blue" stroke-width=".02cm" />

</svg>
```

JS 创建 SVG

- 使用JS脚本来创建SVG时，创建的元素都是需要添加命名空间的。
  - 比如：创建<svg>或者<rect>元素都需要添加命名空间（http://www.w3.org/2000/svg）
  - 对于元素上的属性如果不带前缀的，命名空间赋值为null。

```
<image width="100" height="100" xlink:href="flower.png"></image>
```

- 因为在XML1.1命名空间规范中建议，不带前缀的属性（带前缀xlink:href）命名空间的名称是没有值的，这时命名空间的值必须使用null值。

```
var SVG_NS = 'http://www.w3.org/2000/svg';
var XLink_NS = 'http://www.w3.org/1999/xlink';
var image = document.createElementNS(SVG_NS, 'image');
image.setAttributeNS(null, 'width', '100');
image.setAttributeNS(null, 'height', '100');
image.setAttributeNS(XLink_NS, 'xlink:href', 'flower.png');
```

- 创建 SVG 常用的 DOM2 API：
  - createElementNS（ns，elname）：创建SVG元素
  - setAttributeNS（ns，attrname，value）：给SVG元素添加属性
  - getAttributeNS（ns，attrname）：获取SVG元素上的属性
  - hasAttributeNS（ns, attrname）： 判断SVG元素上是否存在某个属性
  - removeAttributeNS（ns，attname）：删除SVG元素上的某个属性
  - 更多的API：https://developer.mozilla.org/zh-CN/docs/Web/SVG/Namespaces_Crash_Course

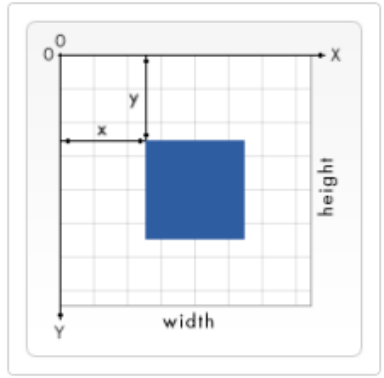
HTML 引入 SVG

- 方式一：img元素
  - 作为一张图片使用，不支持交互，只兼容ie9以上
- 方式二：CSS背景
  - 作为一张背景图片使用，不支持交互
- 方式三：直接在HTML文件引用源文件
  - 作为HTML 的DOM元素，支持交互，只兼容ie9以上
- 方式四：object元素（了解）。
  - 支持交互式 svg，能拿到object的引用，为 SVG 设置动画、更改其样式表等
- 方式五：iframe元素（了解）。
  - 支持交互式 svg，能拿到iframe的引用，为 SVG 设置动画、更改其样式表等
- 方式六：embed元素（了解）。
  - 支持交互式 svg，能拿到embed的引用，为 SVG 设置动画、更改其样式表等，对旧版浏览器有更好的支持。

```
<!-- 1.在object元素上-->
<object data="./svg/rect.svg" type="image/svg+xml"></object>
<!-- 2.使用iframe -->
<iframe src="./svg/rect.svg"></iframe>
<!-- 3.背景图片引入 -->
<embed src="./svg/rect.svg" type="image/svg+xml" />
```

SVG Grid 和 坐标系

- **SVG 使用的坐标系统（网格系统）和 Canvas的差不多。坐标系是 以左上角为 (0,0) 坐标原点，坐标以像素为单位，x 轴正方向是向右，y 轴正方向是向下。**



- **SVG Grid（坐标系）**
  - <svg>元素默认宽为 300px, 高为 150px。如右图所示，<svg>元素默认被网格所覆盖。
  - 通常来说网格中的一个单元相当于 svg 元素中的一像素。
  - 基本上在 SVG 文档中的 1 个像素对应输出设备（比如显示屏）上的 1 个像素（除非缩放）。
  - <svg>元素和其它元素一样也是有一个坐标空间的，其原点位于元素的左上角，被称为初始视口坐标系
  - <svg>的 transform 属性可以用来移动、旋转、缩放SVG中的某个元素，如<svg>中某个元素用了变形，该元素内部会建立一个新的坐标系统，该元素默认后续所有变化都是基于新创建的坐标系统。

SVG 坐标系单位

- SVG坐标系统，在没有明确指定单位时，默认以像素为单位。
- 比如：<rect x="0" y="0" width="100" height="100" />
  - 定义一个矩形，即从左上角开始，向右延展 100px，向下延展 100px，形成一个 100*100 大的矩形。
- 当然我们也可以手动指明坐标系的单位，比如：

```
<!-- mm单位 cm单位 px单位-->
<svg
  version="1.1"
  width="200mm"
  height="200mm"
  xmlns="https://www.w3.org/2000/svg"
>
  <rect width="100mm" height="100mm" fill="pink"></rect>
  <circle r="3cm" cy="75px" cx="150" fill="green"></circle>
</svg>
```

以下是可用于 SVG 元素的单位列表：

单元	描述
em	默认字体大小 - 通常是字符的高度。
ex	人物身高x
px	像素
pt	点（1 / 72 英寸）
pc	皮卡斯（1 / 6 英寸）
cm	厘米
mm	毫米
in	英寸

视口-viewport

- **视口（viewport)**
  - 视口是 SVG 可见的区域（也可以说是SVG画布大小）。可以将视口视为可看到特定场景的窗口。
  - 可以使用<svg>元素的width和height属性指定视口的大小。
  - 一旦设置了最外层 SVG 元素的宽度和高度，浏览器就会建立初始视口坐标系和初始用户坐标系。

```
<!-- the viewport will be 800px by 600px -->
<svg width="800" height="600">
    <!-- SVG content drawn onto the SVG canvas -->
</svg>
```

- **视口坐标系**
  - 视口坐标系是在视口上建立的坐标系，原点在视口左上角的点(0, 0)，x轴正向向右，y轴正向下。
  - 初始视口坐标系中的一个单位等于视口中的一个像素，该坐标系类似于 HTML 元素的坐标系。
- **用户坐标系**（ 也称为当前坐标系或正在使用的用户空间，后面绘图都是参照该坐标系 ）
  - 用户坐标系是建立在 SVG 视口上的坐标系。该坐标系最初与视口坐标系相同——它的原点位于视口的左上角。
  - 使用viewBox属性，可以修改初始用户坐标系，使其不再与视口坐标系相同。

```
<!-- The viewbox in this example is equal to the viewport, but it can
be different -->
<svg width="800" height="600" viewbox="0 0 800 600">
    <!-- SVG content drawn onto the SVG canvas -->
</svg>
```

- **为什么要有两个坐标系?**
  - 因为SVG是矢量图，支持任意缩放。在用户坐标系统绘制的图形，最终会参照视口坐标系来进行等比例缩放。

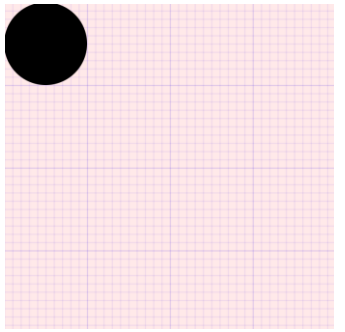
视图框-viewBox

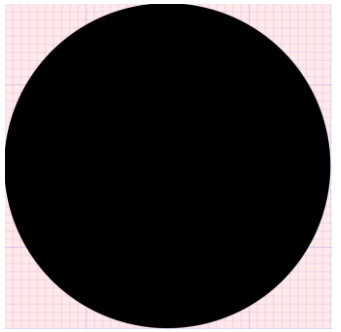
- **视图框（viewBox）**
  - viewport是 SVG 画布的大小，而 viewBox 是用来定义用户坐标系中的位置和尺寸 （该区域通常会被缩放填充视口）。
  - viewBox 也可理解为是用来指定用户坐标系大小。因为SVG的图形都是绘制到该区域中。用户坐标系可以比视口坐标系更小或更大，也可以在视口内完全或部分可见。
  - 一旦创建了视口坐标系（<svg>使用width和height），浏览器就会创建一个与其相同的默认用户坐标系。
  - 我们可以使用 viewBox 属性指定用户坐标系的大小。
    - ✓ 如果用户坐标系与视口坐标系具有相同的高宽比，它将viewBox区域拉伸以填充视口区域。
    - ✓ 如果用户坐标系和视口坐标系没有相同的宽高比，可用 preserveAspectRatio 属性来指定整个用户坐标系统是否在视口内可见。
- **viewBox语法**
  - viewBox = <min-x> <min-y> <width> <height>，比如：viewBox =' 0 0 100 100'
    - ✓ <min-x>和<min-y> 确定视图框的左上角坐标（不是修改用户坐标系的原点，绘图还是从原来的 0, 0 开始）
    - ✓ <width> <height>确定该视图框的宽度和高度。
      - ➢ 宽度和高度不必与父<svg>元素上设置的宽度和高度相同。
      - ➢ 宽度和高度负值无效，为 0 是禁用元素的显示。

```
<!-- The viewbox in this example is equal to the viewport, but it can
be different -->
<svg width="800" height="600" viewbox="0 0 800 600">
    <!-- SVG content drawn onto the SVG canvas -->
</svg>
```

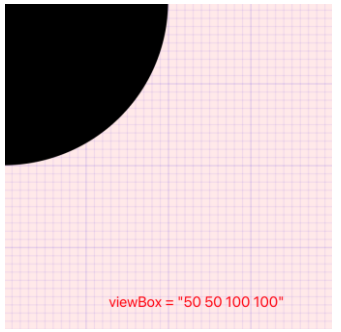
视图框-viewBox-相同的宽高比

- **看一个 viewport 和 viewBox 有相同的宽高比的例子：**
  - ✓ 在viewBox属性上设置视图框为视口大小的一半。
  - ✓ 暂时不改变这个视图框左上角，将<min-x>将<min-y>设置为零。
  - ✓ 视图框的宽度和高度将是视口宽度和高度的一半。





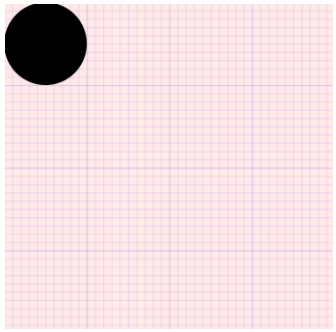
```
<svg width="400" height="400" viewBox="0 0 100 100">
  <circle cx="50" cy="50" r="50"></circle>
</svg>
```

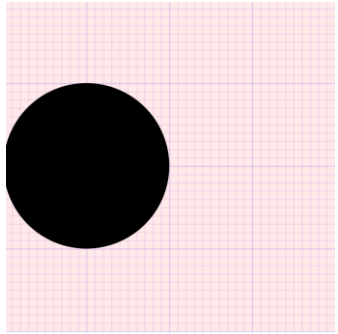


- **那么，viewbox="0 0 100 100" 具体做什么的呢?**
  - 指定画布可显示的区域，用户坐标系从 (0, 0) 的左上点到 (100, 100) 的点，默认单位是px。
  - 然后将 SVG 图形绘制到该 viewBox 区域。
  - viewBox区域等比例被放大（放大不会失真）以填充整个视口。
  - 用户坐标系映射到视口坐标系，因此——在这种情况下——1个用户单位等于4个视口单位。
  - 在 SVG 画布上绘制的任何内容都将相对于该用户坐标系进行绘制 。

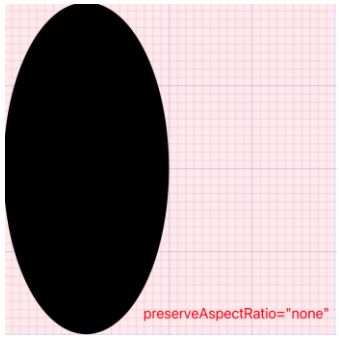
视图框-viewBox-不同的宽高比

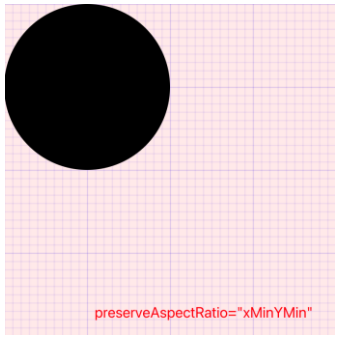
- **在400*400的视口中，viewbox="0 0 200 100" 具体做什么的呢?**
  - 保留视图框viewBox的宽高比，但视图框viewBox不会拉伸以覆盖整个视口区域。
  - 视图框viewBox在视口内垂直和水平居中 。





- **想改变视口内的视框位置怎么办?**
  - 给<svg>添加preserveAspectRatio属性，该属性允许强制统一缩放视图框viewBox
    - ✓ preserveAspectRatio= "none"，强制拉伸图形以填充整个视口。
    - ✓ preserveAspectRatio= “xMinYMin”，图形在视口的最小x和y轴上显示
    - ✓ .....






```
<body>
  <svg
    version="1.1"
    xmlns="https://www.w3.org/2000/svg"
    width="400"
    height="400"
    viewBox="0, 0, 200, 100"
    preserveAspectRatio="none"
  >
    <circle r="50" cy="50" cx="50" fill="rgba(255,0,0,0.2)"></circle>
  </svg>
```

绘制-矩形（rect）

- **SVG的基本形状**
  - 在SVG画布中，如果要想插入一个形状，可以在文档中创建一个对应的元素。
  - 不同的元素对应着不同的形状，并且可以使用不同的属性来定义图形的大小和位置。
  - SVG所支持的基本形状有：矩形、圆形、椭圆、线条、折线、多边形、路径。

- **下面用SVG来绘制一个矩形（rect）**
  - <rect>元素会在屏幕上绘制
  - <rect>元素有6 个基本属性可以控制它在屏幕上的位置和形状。
- **<rect>元素6 个基本属性**
  - x： 矩形左上角的 x 轴位置
  - y： 矩形左上角的 y轴位置
  - width： 矩形的宽度
  - height： 矩形的高度
  - rx： 圆角的 x 轴方位的半径
  - ry： 圆角的 y 轴方位的半径 。

```
<rect x="10" y="10" width="30" height="30"/>
<rect x="60" y="10" rx="10" ry="10" width="30" height="30"/>
```

绘制-圆形（circle）

- **下面用SVG来绘制一个圆形（circle）**
  - <circle>元素会在屏幕上绘制一个圆形。
  - <circle>元素有 3 个基本属性用来设置圆形。
- **<circle>元素3 个基本属性**
  - r： 圆的半径
  - cx： 圆心的 x轴位置
  - cy： 圆心的 y轴位置

```
<circle cx="25" cy="75" r="20"/>
```

绘制-椭圆（ellipse）

- **下面用SVG来绘制一个椭圆（ellipse）**
  - <ellipse>元素是 < circle > 元素更通用的形式，它可以分别缩放圆的 x 半径和 y 半径 。
  - x 半径和 y 半径， 数学家通常称之为长轴半径和短轴半径
  - < ellipse >元素有 4 个基本属性用来设置椭圆。
- **<ellipse>元素4 个基本属性**
  - rx :椭圆的 x轴半径
  - ry :椭圆的 y轴半径
  - cx :椭圆中心的 x轴位置
  - cy :椭圆中心的 y轴位置

```
<ellipse cx="75" cy="75" rx="20" ry="5"/>
```

绘制-线条(line)

- **下面用SVG来绘制一条直线（line）**
  - <line>元素是绘制直线。它取两个点的位置作为属性，指定这条线的起点和终点位置。
  - 需描边才能显示，不支持填充和Canvas线条一样。
  - < line >元素有 4 个基本属性用来设置线条。
- **<line>元素4 个基本属性**
  - x1 :起点的 x 轴位置
  - y1 :起点的 y轴位置
  - x2 :终点的 x轴位置
  - y2 :终点的 y轴位置

```
<line x1="10" x2="50" y1="110" y2="150"/>
```

绘制-折线(polyline)

- **下面用SVG来绘制一条直线（polyline）**
  - <polyline>元素是一组连接在一起的直线。因为它可以有很多的点，折线的的所有点位置都放在一个 points 属性。
  - 默认会填充黑色
  - <polyline >元素有 1 个基本属性用来设置折线所有点位置。
- **<polyline>元素1 个基本属性**
  - points : 点集数列。每个数字用空白、逗号、终止命令符或者换行符分隔开。
  - 每个点必须包含 2 个数字，一个是 x 坐标，一个是 y 坐标。
  - 所以点列表 (0,0)，(1,1) 和 (2,2) 可以写成这样： “0 0, 1 1, 2 2” 。
    - ✓ 支持格式： “0 0, 1 1, 2 2” 或 “0 , 0 , 1 , 1, 2 , 2” 或 “0 0 1 1 2 2”

```
<polyline points="60 110, 65 120, 70 115, 75 130, 80 125, 85 140, 90 135,
95 150, 100 145"/>
```

绘制-多边型（polygon）

- **下面用SVG来绘制多边形（polygon）**
  - <polygon>元素是和折线很像，它们都是由连接一组点集的直线构成。不同的是，< polygon > 的路径在最后一个点处自动回到第一个点。需要注意的是，矩形也是一种多边形，也可以用多边形创建一个矩形。
  - 默认会填充黑色
  - < polygon >元素有 1 个基本属性用来设置多边形线所有点位置。
- **<polygon>元素1 个基本属性**
  - points :点集数列。每个数字用空白符、逗号、终止命令或者换行符分隔开。
  - 每个点必须包含 2 个数字，一个是 x 坐标，一个是 y 坐标。
  - 所以点列表 (0,0), (1,1) 和 (2,2) 推荐写成这样：“0 0, 1 1, 2 2”。
  - 路径绘制完后闭合图形，所以最终的直线将从位置 (2,2) 连接到位置 (0,0)。

```
<polygon points="50 160, 55 180, 70 180, 60 190, 65 205, 50 195, 35 205,
40 190, 30 180, 45 180"/>
```

绘制-路径（path）

- **下面用SVG来绘制路径（path）**
  - <path>元素可能是 SVG 中最常见的形状。你可以用 < path >元素绘制矩形（直角矩形或圆角矩形）、圆形、椭圆、折线形、多边形，以及一些其他的形状，例如贝塞尔曲线、2 次曲线等曲线。
  - 默认会填充黑色，默认路径不会闭合
  - < path >元素有 1 个基本属性用来设置路径点的位置。

```
<path
    d="M 120 10, 190 10, 190 50"
    stroke="orange"
    fill="transparent"
></path>
```

- **<path>元素1 个基本属性**
  - d :一个点集数列，以及其它关于如何绘制路径的信息，必须M命令开头。
    - ✓ 所以点列表 (0,0), (1,1) 和 (2,2) 推荐写成这样：“M 0 0, 1 1, 2 2”。
    - ✓ 支持格式：“M 0 0, 1 1, 2 2”或“M0 0, 1 1, 2 2”或“M 0，0，1， 1, 2， 2”或“M 0 0 1 1 2 2”

SVG 路径 和 命令

- **SVG路径（path） 和 命名**
  - <path>元素是 SVG基本形状中最强大的一个。 你可以用它创建线条，曲线，弧形等等。
  - <path> 元素的形状是通过属性d定义的，属性d的值是一个 “命令 + 参数 ” 的序列。
  - 每一个命令都用一个关键字母来表示，比如，字母“M”表示的是“Move to”命令，当解析器读到这个命令时，它就知道你是打算移动到某个点。跟在命令字母后面的，是你需要移动到的那个点的 x 和 y 轴坐标。比如移动到 (10,10) 这个点的命令，应该写成“M 10 10”命令。这一段字符结束后，解析器就会去读下一段命令。每一个命令都有两种表示方式，一种是用大写字母，表示采用绝对定位。另一种是用小写字母，表示采用相对定位（例如：从上一个点开始，向上移动 10px，向左移动 7px）。
  - 属性 d采用的是用户坐标系统，**不需标明单位。**

- **d属性值支持的命令**
  - 直线命令
    - ✓ M / m：Move To
    - ✓ L / l ： Line To
    - ✓ Z / z：Close Path
    - ✓ H / h：horizontal
    - ✓ V / v：vertical
  - 曲线命令
    - ✓ C ： 三次贝塞尔曲线
    - ✓ S：简写三次贝塞尔曲线
    - ✓ Q：二次贝塞尔曲线
    - ✓ T：简写二次贝塞尔曲线

```
<path
  d="M 10 10, L 100 10, L 100 50"
  stroke="orange"
  fill="transparent"
></path>
```

绘制-图片

- 在SVG中绘制一张图片
  - 在<image>元素的 href 属性引入图片URL


```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="4cm" version="1.1"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink=
"http://www.w3.org/1999/xlink">
  <image xlink:href="firefox.jpg" x="0" y="0" height="50px"
width="50px"/>
</svg>
```

- 注意事项
  - image 元素没设置 x , y 值，它们自动被设置为 0。
  - image 元素没设置 height 、width 时，默认为图片大小。
  - width 、height 等于 0，将不会呈现这个图像。
  - 需在 href 属性上引用外部的图像，不是src属性。
  - href属性兼容性： https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/href#browser_compatibility

> **Note:** SVG 2 removed the need for the `xlink` namespace, so instead of `xlink:href` you should use `href`. If you need to support earlier browser versions, the deprecated `xlink:href` attribute can be used as a fallback in addition to the `href` attribute, e.g. `<use href="some-id" xlink:href="some-id" x="5" y="5" />`.

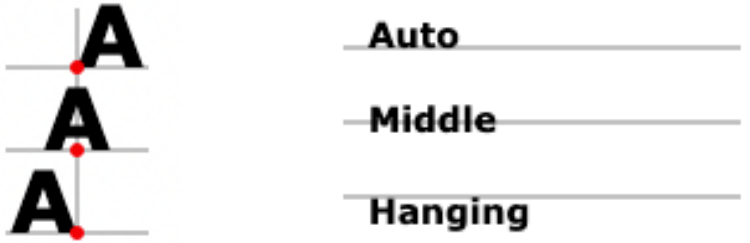
绘制-文字

- **下面我们来看一下如何在SVG画布中绘制文字。**
  - <text>元素是用来在SVG画布中绘制文字用的。

```
<text x="10" y="10">Hello World!</text>
```

```
<text>
  <tspan font-weight="bold" fill="red">This is bold and red</tspan>
</text>
```

- **<text>元素的基本属性**
  - x 和 y 属性决定了文本在用户坐标系中显示的位置。
  - text-anchor 文本流方向属性，可以有 start、middle、end 或 inherit 值，默认值 start
  - dominant-baseline 基线对齐属性 : 有 auto 、middle 或 hanging 值, 默认值：auto



- **<text>元素的字体属性**
  - 文本的一个至关重要的部分是它显示的字体。SVG 提供了一些属性，类似于CSS 。下列的属性可以被设置为一个 SVG 属性或一个 CSS 属性:
    - ✓ font-family、font-style、font-weight、font-variant、font-stretch、font-size、font-size-adjust、kerning、letter-spacing、word-spacing和text-decoration。
- **其它文本相关的元素:**
  - **<tspan>** 元素用来标记大块文本的子部分，它必须是一个text元素或别的tspan元素的子元素。
    - ✓ x 和 y 属性决定了文本在视口坐标系中显示的位置。
    - ✓ alignment-baseline 基线对齐属性：auto 、baseline、middle、hanging、top、bottom ... ,默认是 auto

元素的组合（g）

- **元素的组合**
  - <g>元素是用来组合元素的容器。
  - 添加到g元素上的变换会应用到其所有的子元素上。
  - 添加到g元素的属性大部分会被其所有的子元素继承。
  - g元素也可以用来定义复杂的对象，之后可以通过<use>元素来引用它们

```
<!-- 分组 -->
<g stroke="red" fill="transparent">
  <circle r="30" cx="50" cy="100"></circle>
  <circle r="30" cx="90" cy="100"></circle>
  <circle r="30" cx="130" cy="100"></circle>
  <circle r="30" cx="170" cy="100"></circle>
</g>
```

- **<g>元素的属性（该元素只包含全局属性）**
  - 核心属性：id
  - 样式属性：class 、style
  - Presentation Attributes（也可说是 CSS 属性，这些属性可写在CSS中，也可作为元素的属性用）：
    - ✓ cursor, display, fill, fill-opacity, opacity,...
    - ✓ stroke, stroke-dasharray, stroke-dashoffset, stroke-linecap, stroke-linejoin
    - ✓ 更多表示属性：https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/Presentation
  - 事件属性：onchange, onclick, ondblclick, ondrag...
  - 动画属性：transform
  - 更多：https://developer.mozilla.org/en-US/docs/Web/SVG/Element/g

图形元素的复用（defs）

- SVG 是允许我们定义一些可复用元素的。
  - 即把可复用的元素定义在< defs >元素里面，然后通过<use>元素来引用和显示。
  - 这样可以增加 SVG 内容的易读性、复用性和利于无障碍开发。

- < defs >元素，定义可复用元素。
  - 例如：定义基本图形、组合图形、渐变、滤镜、样式等等。
  - 在< defs >元素中定义的图形元素是不会直接显示的。
  - 可在视口任意地方用<use>来呈现在defs中定义的元素。
  - <defs>元素没有专有属性，使用时通常也不需添加任何属性。

- <defs>定义的元素的坐标系参照哪个？用户坐标系

```
<svg width="300" height="150" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <style>
      rect {
        fill: red;
      }
    </style>
    <linearGradient id="Gradiend1">
      <stop offset="0%" stop-color="red"></stop>
      <stop offset="50%" stop-color="green"></stop>
      <stop offset="100%" stop-color="blue"></stop>
    </linearGradient>
    <rect id="rectangle" x="0" y="0" width="30" height="30"></rect
  </defs>
  <use href="#rectangle"></use>
  <use x="100" y="100" href="#rectangle"></use>
</svg>
<!-- 也支持在外部SVG引用 -->
```

引入元素（use）

- **<use>元素从 SVG 文档中获取节点，并将获取到的节点复制到指定的地方。**
  - <use>等同于深度克隆DOM节点，克隆到use元素所在的位置。
  - 克隆的节点是不可见的，当给<use>元素应用CSS样式时须小心。因为克隆的 DOM 不能保证都会继承 <use>元素上的CSS属性，但是CSS可继承的属性是会继承的。
- **<use>元素的属性**
  - href： 需要复制元素/片段的 URL 或 ID（支持跨SVG引用）。默认值：无
  - ~~xlink:href： （SVG2.0已弃用）~~ 需要复制的元素/片段的 URL 或 ID 。默认值：无
  - x / y：元素的 x / y 坐标（相对复制元素的位置）。 默认值：0
  - width / height：元素的宽和高（在引入svg或symbol元素才起作用）。 默认值：0

> **Note:** `width`, and `height` have no effect on `use` elements, unless the element referenced has a viewBox - i.e. they only have an effect when `use` refers to a `svg` or `symbol` element.

> **Note:** Starting with SVG2, `x`, `y`, `width`, and `height` are *Geometry Properties*, meaning those attributes can also be used as CSS properties for that element.

```
<use
  x="0"
  y="0"
  xlink:href="#myCircle"
  href="#myCircle"
  fill="url('#myGradient')"
/>
```

图形元素复用（symbols）

- **<symbol>元素和<defs>元素类似，也是用于定义可复用元素，然后通过<use>元素来引用显示。**
  - 在<symbol>元素中定义的图形元素默认也是不会显示在界面上。
  - <symbol>元素常见的应用场景是用来定义各种小图标，比如：icon、logo、徽章等
- **<symbol>元素的属性**
  - viewBox：定义当前 <symbol> 的视图框。
  - x / y ： symbol元素的 x / y坐标。 ； 默认值：0
  - width / height：symbol元素的宽度。 默认值：0
- **<symbol>和<defs> 的区别**
  - <defs>元素没有专有属性，而<symbol>元素提供了更多的属性
    - ✓ 比如： viewBox、 preserveAspectRatio 、x、y、width、height等。
  - <symbol>元素有自己用户坐标系，可以用于制作SVG精灵图。
  - <symbol>元素定义的图形增加了结构和语义性，提高文档的可访问性。
- **SVG ICON文件-合并成SVG精灵图：**https://www.zhangxinxu.com/sp/svgo

```
<body>
  <svg style="display: none" width="200" height="200">
    <!-- 第一个图标 -->
    <symbol id="hy-circle" viewBox="0 0 200 200">…
    </symbol>
    <!-- 第一个图标 -->
    <symbol id="hy-rect" viewBox="0 0 200 200">…
    </symbol>
  </svg>

  <svg width="50" height="50">
    <use x="0" y="0" href="#hy-circle" />
  </svg>
  <svg width="50" height="50">
    <use x="0" y="0" href="#hy-rect" />
  </svg>
</body>
```

填充和描边

- **如果想要给SVG中的元素上色，一般有两种方案可以实现：**
  - 第一种：直接使用元素的属性，比如：填充（fill）属性、描边（stroke）属性等。
  - 第二种：直接编写CSS样式，因为SVG也是HTML中的元素，也支持用CSS的方式来编写样式。

- **第一种：直接使用元素的属性（填充和描边）**
  - 在SVG中，绝大多数元素的上色都可通过 fill和stroke 两个属性来搞定。
    - ✓ fill属性：设置对象填充颜色。支持：颜色名、十六进制值、rgb、 rgba 。
    - ✓ stroke属性：设置绘制对象的边框颜色。支持：颜色名、十六进制值、rgb、 rgba 。

```
<path
    d="M 10 10, L 100 10, L 100 50"
    stroke="orange"
    fill="transparent"
></path>
```

第一种：填充属性（fill）

- fill 填充属性，专门用来给SVG中的元素填充颜色。
  - fill = “color” 。支持：颜色名、十六进制值、rgb、 rgba 、 currentColor （继承自身或父亲字体color）。
- 如下图给矩形填充颜色：fill= “pink”

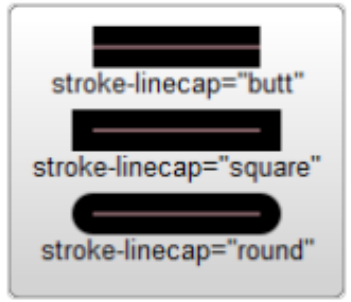
```
<rect x="0" y="0" width="100" height="50" fill="pink"></rect>
```

- 控制填充色的不透明
  - fill-opacity = ” number ” ， 该属性专门用来控制填充色的不透明，值为 0 到 1。

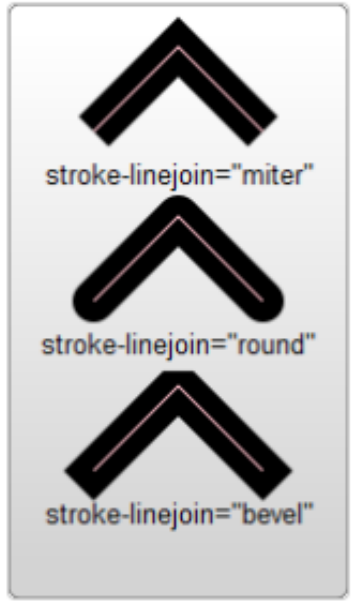
```
<rect
    x="0"
    y="0"
    width="100"
    height="50"
    fill="pink"
    fill-opacity="0.3"
></rect>
```

第一种：描边属性（stroke）

- **stroke 描边属性**
  - stroke = “color”： 指定元素边框填充颜色。
  - stroke-opacity = “number”：控制元素边框填充颜色的透明度。
  - stroke-width = “number”：指定边框的宽度。注意，边框是以路径为中心线绘制的。
  - stroke-linecap = “butt | square | round”：控制边框端点的样式。
  - stroke-linejoin = “miter | round | bevel”:控制两条线段连接处样式
  - stroke-dasharray = “number [, number , ...]”：将虚线类型应用在边框上。
    - ✓ 该值必须是用逗号分割的数字组成的数列，空格会被忽略。比如 3，5 ：
      - ➢ 第一个表示填色区域长度为 3
      - ➢ 第二个表示非填色区域长度为 5
  - stroke-dashoffset：指定在dasharray模式下路径的偏移量。
    - ✓ 值为number类型，除了可以正值，也可以取负值。




```
<rect
  x="10"
  y="10"
  width="100"
  height="50"
  fill="rgba(0,0,255,0.5)"
  stroke="pink"
  stroke-width="5"
  stroke-opacity="1"
  stroke-linejoin="round"
  stroke-dasharray="20,10"
  stroke-linecap="round"
></rect>
```



第二种：CSS 样式

- **直接编写CSS样式实现填充和描边**
  - 除了定义元素的属性外，你也可以通过CSS来实现填充和描边（CSS样式可写在defs中，也可写在HTML头部或外部等）。
  - 语法和 HTML 里使用 CSS 一样，需要注意的是：需要把 background-color、border 改成 fill 和 stroke
  - 不是所有的属性都能用 CSS 来设置，上色和填充的部分是可以用 CSS 来设置。
    - ✓ 比如，fill，stroke，stroke-dasharray 等可以用CSS设置；比如，路径的命令则不能用 CSS 设置。
- **哪些属性可以使用CSS设置，哪些不能呢?**
  - SVG规范中将属性区分成 Presentation Attributes 和 Attributes 属性。
    - ✓ Presentation Attributes 属性(支持CSS和元素用)：https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/Presentation
    - ✓ Attributes 属性（只能在元素用）： https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute
    - ✓ 提示：这些属性是不需要去记的，用多了就记住了，在忘记时测试一下就知道了。
- **CSS给SVG中的元素填充、描边和上色，支持如下4种编写方式：**
  - 方式一：内联（行内） CSS 样式，写在元素的style属性上
  - 方式二：内嵌（内部） CSS 样式，写在 <defs>中的 <style>标签中
  - 方式三：内嵌（内部） CSS 样式，写在<head>中的<style>标签中
  - 方式四：外部 CSS 样式文件，写在 .css 文件中
- **CSS样式优先级别：**内联的style > defs中的style > 外部 / head内部 > 属性 fill

```
<rect x="10" height="180" y="10" width="180" style="stroke: black; fill: red;"/>
```

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css" href="style.css"?>

<svg width="200" height="150" xmlns="http://www.w3.org/2000/svg"
version="1.1">
  <rect height="10" width="10" id="MyRect"/>
</svg>
```