SVG  系列课程之



SVG中的坐标系统与坐标变换

@techird

Laydies and gentelment, 欢迎回来观看慕课⺴⽹网 SVG 系列课程。上⼀一节课给⼤大家⼤大致了解了⼀一些 SVG 的基本知识，那么这节课，就带⼤大家进⼊入 SVG 的坐标系统世界。

!

坐标系统这⼀一部分，特别是坐标变换，可能在很多介绍 SVG 的教程上都会把它作为⽐比较⾼高级的知识来介绍，但是我认为这部分知识⾮非常重要，因为它影响你在实际开发应⽤用

中⽅方⽅方⾯面⾯面的理解。学习好本节课，会让你在图形定位上不再迷迷糊糊的。

Lesson  2  -  SVG  中的坐标系统和坐标变换



2.1.  SVG  的世界、视野、视窗的概念 

2.2.  SVG  中的图形分组 

2.3.  坐标系统概述 

2.4.  自身坐标系和参照坐标系 

2.5.  坐标变换

下⾯面来看⼀一下这节课的内容。

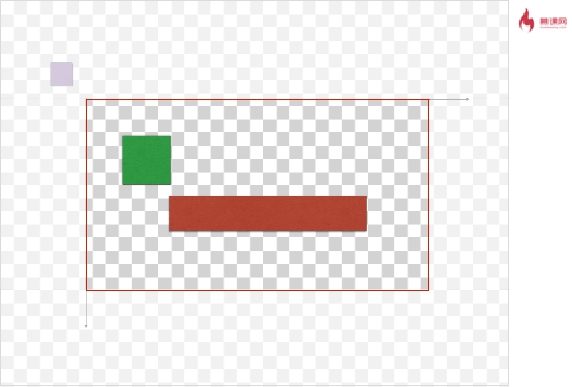
视野与世界



·  世界是⽆无穷⼤大的

·  视野是观察世界的⼀一个矩形区域

世界



视野（viewbox）

2D绘图中很多⼈人会有⼀一个误区，就是我绘图的区域是⼀一个矩形区域。⽆无论新建⼀一个画布还是创建了⼀一个容器，⼼心⾥里都想象⾥里⾯面有⼀一个矩形区域。

!

其实，在SVG当中，矩形区域只是视野，是我们看到的部分。实际上你能绘制的区域是⼀一个⽆无穷⼤大的世界。

!

世界是客观地，只要定义了世界的内容，那么内容就是确定的。视野是主观地，⼤大部分绘图API都提供视野的控制⽅方法，像在SVG中，Viewbox来控制视野。

世界



视野（viewbox）

2D绘图中很多⼈人会有⼀一个误区，就是我绘图的区域是⼀一个矩形区域。⽆无论新建⼀一个画布还是创建了⼀一个容器，⼼心⾥里都想象⾥里⾯面有⼀一个矩形区域。

!

其实，在SVG当中，矩形区域只是视野，是我们看到的部分。实际上你能绘制的区域是⼀一个⽆无穷⼤大的世界。

!

世界是客观地，只要定义了世界的内容，那么内容就是确定的。视野是主观地，⼤大部分绘图API都提供视野的控制⽅方法，像在SVG中，Viewbox来控制视野。

2.1. SVG 的世界、视野、视窗的概念



·  width, height - 控制视窗

·  SVG 代码 - 定义世界

·  viewBox, preserveAspectRatio - 控制视野

在 SVG 标签当中可以指定⼀一个宽和⾼高属性，来表⽰示 SVG ⽂文件渲染的区域⼤大⼩小。这个⼤大⼩小也可以使⽤用样式表来定义。这个区域⼤大⼩小，就是视窗。视窗实际上就是浏览器开辟

出来⽤用于渲染 SVG 内容的⼀一个区域。

!

在 SVG 当中，⾥里⾯面的内容就是对 SVG 世界的定义，这个 SVG ⽂文件⾥里⾯面有多少个矩形多少条曲线，在哪⾥里，什么颜⾊色，都是在定义世界。

!

⽽而视野，也就是观看世界的矩形区域是哪⼀一个，使⽤用 viewBox 进⾏行定义。

!

这⾥里出现了视窗和视野，在理想情况下，视野和视窗有⼀一样的尺⼨寸，那浏览器就可以地把视野完美地填充到视窗内。可是如果视窗和视野⼤大⼩小不⼀一致，就存在如何控制这个

填充的问题，填充的策略使⽤用 preserveAspectRatio 进⾏行指定。

!

锤⼦子的故事



了解完世界、视窗和视野的概念，在继续深⼊入坐标系的问题之前，我想先给⼤大家讲⼀一个关于锤⼦子的故事。

从前有⼀一个画家



他很擅⻓长画锤⼦子



先画⼀一个矩形作为锤头，再画⼀一个矩形作为⼿手柄。⼤大功告成。

他很擅⻓长画锤⼦子



先画⼀一个矩形作为锤头，再画⼀一个矩形作为⼿手柄。⼤大功告成。

他很擅⻓长画锤⼦子



先画⼀一个矩形作为锤头，再画⼀一个矩形作为⼿手柄。⼤大功告成。

有⼀一天他改⾏行当程序员



由于种种原因，改⾏行当程序员。

⽼老板说



“你用代码画一个锤子吧”

⽼老板知道他是画锤⼦子出⽣生的，就说，“你⽤用程序画⼀一个锤⼦子吧”

太简单了



**X**

**<svg**  xmlns="http://www.w3.org/2000/svg"**>**!

50

100

**<rect**  x="50"  y="50"  !

50

90

100

50

100

width="100"  height="50"  !

stroke="red"  fill="none"**>**!

**</rect>**!

**<rect**  x="90"  y="100"  !

width="20"  height="120"  !

stroke="red"  fill="none"**>**!

**</rect>**!

**</svg>**!

**Y**

20

画家学习了慕课⺴⽹网系列的SVG课程，于是他算好坐标，使⽤用 SVG 画了两个矩形。⼀一切看起来都是美好的。

⽼老板⼜又说



“锤子往右挪50像素吧”

没问题



**X**

**<svg**  xmlns="http://www.w3.org/2000/svg"**>**!

50

100

**<rect**  x="50"  y="50"  !

50

90

100

50

100

width="100"  height="50"  !

stroke="red"  fill="none"**>**!

**</rect>**!

**<rect**  x="90"  y="100"  !

width="20"  height="120"  !

stroke="red"  fill="none"**>**!

**</rect>**!

**</svg>**!

**Y**

20

没问题



**X**

**<svg**  xmlns="http://www.w3.org/2000/svg"**>**!

50

100

**<rect**  x="**100**"  y="50"  !

**100**

**140**

100

50

100

width="100"  height="50"  !

stroke="red"  fill="none"**>**!

**</rect>**!

**<rect**  x="**140**"  y="100"  !

width="20"  height="120"  !

stroke="red"  fill="none"**>**!

**</rect>**!

**</svg>**!

**Y**

20

他把两个矩形的 X 坐标都加了 50，修改了两个矩形标签的 x 值，⼤大功告成。

⽼老板还不满意



“我想要一把绿色的锤子”

没问题2



**X**

**<svg**  xmlns="http://www.w3.org/2000/svg"**>**!

50

100

**<rect**  x="**100**"  y="50"  !

**100**

**140**

100

50

100

width="100"  height="50"  !

stroke="red"  fill="none"**>**!

**</rect>**!

**<rect**  x="**140**"  y="100"  !

width="20"  height="120"  !

stroke="red"  fill="none"**>**!

**</rect>**!

**</svg>**!

**Y**

20

他⼜又把两个矩形的 stroke 属性修改为 green

没问题2



**X**

**<svg**  xmlns="http://www.w3.org/2000/svg"**>**!

50

100

**<rect**  x="**100**"  y="50"  !

**100**

**140**

100

50

100

width="100"  height="50"  !

stroke="**green**"  fill="none"**>**!

**</rect>**!

**<rect**  x="**140**"  y="100"  !

width="20"  height="120"  !

stroke="**green**"  fill="none"**>**!

**</rect>**!

**</svg>**!

**Y**

20

他⼜又把两个矩形的 stroke 属性修改为 green

同学们，到了这⾥里⼤大家思考⼀一下，真的没有问题吗？⽼老板这次是让画的锤⼦子，那么假如说某天它让画家画⼀一把瑞⼠士军⼑刀，怎么办？



!

其实，⽆无论是锤⼦子也好，⼩小⽶米也好，瑞⼠士军⼑刀抑或苹果也好，⽼老板都只是要针对图形的整体进⾏行修改，⽽而不是修改其细节。既然逻辑上它们是⼀一个整体，那么，在代码⾥里也

应该是有⼀一个整体的概念。有了整体的概念，就可以有整体的操作。

？？



？

同学们，到了这⾥里⼤大家思考⼀一下，真的没有问题吗？⽼老板这次是让画的锤⼦子，那么假如说某天它让画家画⼀一把瑞⼠士军⼑刀，怎么办？

!

其实，⽆无论是锤⼦子也好，⼩小⽶米也好，瑞⼠士军⼑刀抑或苹果也好，⽼老板都只是要针对图形的整体进⾏行修改，⽽而不是修改其细节。既然逻辑上它们是⼀一个整体，那么，在代码⾥里也

应该是有⼀一个整体的概念。有了整体的概念，就可以有整体的操作。

？？



？

同学们，到了这⾥里⼤大家思考⼀一下，真的没有问题吗？⽼老板这次是让画的锤⼦子，那么假如说某天它让画家画⼀一把瑞⼠士军⼑刀，怎么办？

!

其实，⽆无论是锤⼦子也好，⼩小⽶米也好，瑞⼠士军⼑刀抑或苹果也好，⽼老板都只是要针对图形的整体进⾏行修改，⽽而不是修改其细节。既然逻辑上它们是⼀一个整体，那么，在代码⾥里也

应该是有⼀一个整体的概念。有了整体的概念，就可以有整体的操作。

？？



？

同学们，到了这⾥里⼤大家思考⼀一下，真的没有问题吗？⽼老板这次是让画的锤⼦子，那么假如说某天它让画家画⼀一把瑞⼠士军⼑刀，怎么办？

!

其实，⽆无论是锤⼦子也好，⼩小⽶米也好，瑞⼠士军⼑刀抑或苹果也好，⽼老板都只是要针对图形的整体进⾏行修改，⽽而不是修改其细节。既然逻辑上它们是⼀一个整体，那么，在代码⾥里也

应该是有⼀一个整体的概念。有了整体的概念，就可以有整体的操作。

Lesson  2  -  SVG  中的坐标系统和坐标变换



2.1.  SVG  的世界、视野、视窗的概念 

2.2.  SVG  中的图形分组 

2.3.  坐标系统概述 

2.4.  自身坐标系和参照坐标系 

2.5.  坐标变换

下⾯面，就给⼤大家讲⼀一下 SVG 当中分组的概念。

2.2.  SVG  中的图形分组



·   <g>标签来创建分组 

·   属性继承 

·   transform  属性定义坐标变换 

·   可以嵌套使用

下⾯面，就给⼤大家讲⼀一下 SVG 当中分组的概念。

2.2.  SVG  中的图形分组



**X**

**<svg**  xmlns="..."**>**!

100

50

100

50

**<g**  stroke="green"  fill="none"**>**  !

**<rect**  x="100"  y="50"  !

width="100"  height="50"**>**!

**</rect>**!

**Y**

140

100

100

**<rect**  x="140"  y="100"  !

width="20"  height="120"**>**!

**</rect>**!

**</g>**!

**</svg>**!

20

还是拿充满情怀的锤⼦子来说，画家修改了⼀一下上⾯面的代码，⽤用⼀一个 <g> 标签把两个锤⼦子包起来了。然后，把描边和填充属性设置在 <g> 标签上。现在，这个代表了锤⼦子的

<g> 标签就可以作为⼀一个整体进⾏行操作。

2.2.  SVG  中的图形分组



**<svg**  xmlns="..."**>**!

**50**

**X**

**<g**  stroke="green"  fill="none"!

**transform="translate(0,  50)">**  !

100

50

100

50

**<rect**  x="100"  y="50"  !

width="100"  height="50"**>**!

**</rect>**!

**<rect**  x="140"  y="100"  !

**Y**

140

100

100

width="20"  height="120"**>**!

**</rect>**!

**</g>**!

**</svg>**!

20

现在，画家紧张地给锤⼦子加了⼀一个 transform 属性，锤⼦子往下挪了！哟⻄西！看来成功了。咦？⽼老师你什么时候画了两个坐标系？咳咳，这⼀一课的⾼高潮要来了。请允许我介

绍，坐标系统！

Lesson  2  -  SVG  中的坐标系统和坐标变换



2.1.  SVG  的世界、视野、视窗的概念 

2.2.  SVG  中的图形分组 

2.3.  坐标系统概述 

2.4.  四个坐标系 

2.5.  坐标变换

这节课给⼤大家介绍 SVG 中坐标系统的相关知识。前⾯面两节课呢兜兜转转讲了差不多半个⼩小时，也没开始讲坐标系统的事⼉儿，那么⼤大家也别⼼心急，接下来⼤大家就知道，前⾯面的

铺垫还是有意义的。

!

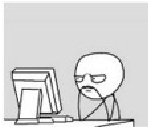
那么你也许也注意到呢⽼老师已经偷偷地把第四⼩小节的标题给改了，之前打算讲⾃自⾝身坐标系和参考坐标系这两个坐标系，那么因为同学们都⾮非常的⽀支持和努⼒力，⽼老师呢现在决

定买⼆二送⼆二，⼀一次性给你讲明⽩白四个坐标系。

!

好，下⾯面开始上课。

Lesson  2  -  SVG  中的坐标系统和坐标变换



2.1.  SVG  的世界、视野、视窗的概念 

2.2.  SVG  中的图形分组 

2.3.  坐标系统概述 

2.4.  四个坐标系 

2.5.  坐标变换

这节课给⼤大家介绍 SVG 中坐标系统的相关知识。前⾯面两节课呢兜兜转转讲了差不多半个⼩小时，也没开始讲坐标系统的事⼉儿，那么⼤大家也别⼼心急，接下来⼤大家就知道，前⾯面的

铺垫还是有意义的。

!

那么你也许也注意到呢⽼老师已经偷偷地把第四⼩小节的标题给改了，之前打算讲⾃自⾝身坐标系和参考坐标系这两个坐标系，那么因为同学们都⾮非常的⽀支持和努⼒力，⽼老师呢现在决

定买⼆二送⼆二，⼀一次性给你讲明⽩白四个坐标系。

!

好，下⾯面开始上课。

2.3.  坐标系统概述



·   笛卡尔直角坐标系 

·   原点 

·   互相垂直的两条数轴 

·   角度定义

先来简单看看坐标系统的概述。SVG 使⽤用的坐标系统是笛卡尔直⾓角坐标系统，本课讨论的坐标系统，⽆无特殊说明，都是指笛卡尔直⾓角坐标系。

坐标系统的作⽤用是为图形提供统⼀一的定位基准，为了做到这点，笛卡尔直⾓角坐标系定义了包括⼀一个原点以及两条相互垂直的数轴。基于原点和数轴的定义，⼜又可以定义⾓角度

的值的含义。

2.3.  坐标系统概述



·   笛卡尔直角坐标系 

·   原点 

·   互相垂直的两条数轴 

·   角度定义

先来简单看看坐标系统的概述。SVG 使⽤用的坐标系统是笛卡尔直⾓角坐标系统，本课讨论的坐标系统，⽆无特殊说明，都是指笛卡尔直⾓角坐标系。

坐标系统的作⽤用是为图形提供统⼀一的定位基准，为了做到这点，笛卡尔直⾓角坐标系定义了包括⼀一个原点以及两条相互垂直的数轴。基于原点和数轴的定义，⼜又可以定义⾓角度

的值的含义。

2.3.  坐标系统概述



Y（90°）

·   笛卡尔直角坐标系 

·   原点 

·   互相垂直的两条数轴 

X（0°）

·   角度定义

先来简单看看坐标系统的概述。SVG 使⽤用的坐标系统是笛卡尔直⾓角坐标系统，本课讨论的坐标系统，⽆无特殊说明，都是指笛卡尔直⾓角坐标系。

坐标系统的作⽤用是为图形提供统⼀一的定位基准，为了做到这点，笛卡尔直⾓角坐标系定义了包括⼀一个原点以及两条相互垂直的数轴。基于原点和数轴的定义，⼜又可以定义⾓角度

的值的含义。

2.3.  坐标系统概述



Y（90°）

·   笛卡尔直角坐标系 

·   原点 

·   互相垂直的两条数轴 

45°

X（0°）

·   角度定义

先来简单看看坐标系统的概述。SVG 使⽤用的坐标系统是笛卡尔直⾓角坐标系统，本课讨论的坐标系统，⽆无特殊说明，都是指笛卡尔直⾓角坐标系。

坐标系统的作⽤用是为图形提供统⼀一的定位基准，为了做到这点，笛卡尔直⾓角坐标系定义了包括⼀一个原点以及两条相互垂直的数轴。基于原点和数轴的定义，⼜又可以定义⾓角度

的值的含义。

2.3.  坐标系统概述



Y（90°）

·   笛卡尔直角坐标系 

·   原点 

·   互相垂直的两条数轴 

数学上

45°

X（0°）

·   角度定义

在数学上呢，x轴⽔水平向右，y轴竖直向上，这是我们周知的。⽽而相应的，⾓角度的扫描⽅方向，也就是正⾓角的⽅方向，是逆时针的。

2.3.  坐标系统概述



·   笛卡尔直角坐标系 

·   原点 

·   互相垂直的两条数轴 

·   角度定义

SVG

45°

X（0°）

Y（90°）

但是由于 SVG 的阅读媒介⼀一般是屏幕，出于对⼈人类阅读习惯的考虑，⼤大多数屏幕上使⽤用的笛卡尔坐标系都是 Y 轴朝下的。这种情况下，⾓角度的正⽅方向是顺时针⽅方向。

!

其实⾓角度的⽅方向在笛卡尔坐标系中是有统⼀一描述的，就是从 X 轴正⽅方向到 Y 轴正⽅方向的直⾓角旋转⽅方向为正⽅方向。

2.3.  坐标系统概述



·   笛卡尔直角坐标系 

·   原点 

·   互相垂直的两条数轴 

·   角度定义

SVG

45°

X（0°）

Y（90°）

但是由于 SVG 的阅读媒介⼀一般是屏幕，出于对⼈人类阅读习惯的考虑，⼤大多数屏幕上使⽤用的笛卡尔坐标系都是 Y 轴朝下的。这种情况下，⾓角度的正⽅方向是顺时针⽅方向。

!

其实⾓角度的⽅方向在笛卡尔坐标系中是有统⼀一描述的，就是从 X 轴正⽅方向到 Y 轴正⽅方向的直⾓角旋转⽅方向为正⽅方向。

Lesson  2  -  SVG  中的坐标系统和坐标变换



2.1.  SVG  的世界、视野、视窗的概念 

2.2.  SVG  中的图形分组 

2.3.  坐标系统概述 

2.4.  四个坐标系 

2.5.  坐标变换

接下来给⼤大家讲⼀一下 SVG ⾥里⾯面四个需要⼤大家记住坐标系。

2.4.  四个坐标系



·   用户坐标系（User  Coordinate） 

‣  世界的坐标系 

·   自身坐标系（Current  Coordinate） 

‣  每个图形元素或分组独立与生俱来 

·   前驱坐标系（Previous  Coordinate） 

‣  父容器的坐标系 

·   参考坐标系（Reference  Coordinate） 

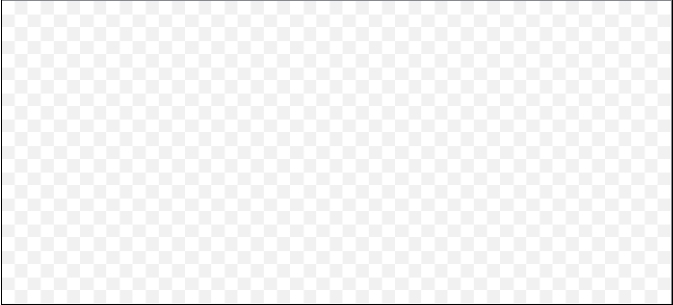
‣  使用其它坐标系来考究自身的情况时使用

这四个坐标系呢，分别叫⽤用户坐标系、⾃自⾝身坐标系、前驱坐标系、参考坐标系。

!

回顾⼀一下世界的概念。

2.4.1.  用户坐标系（User  Coordinate）



SVG 的世界是⽆无限⼤大的，世界有⼀一个坐标系，这个坐标系就是⽤用户坐标系。

我们设置的 viewbox，也就是视野的⼤大⼩小，就说就是观察⽤用户坐标系中的哪个区域。⽐比如说，现在设置 viewbox 为 (0,0,200,150)

⽤用户坐标系是最原始的坐标系，其它产⽣生的坐标系都从⽤用户坐标系开始，所以⽤用户坐标系也可以称之为原始坐标系（initial coordinate）

2.4.1.  用户坐标系（User  Coordinate）



**Ouser** 100 200

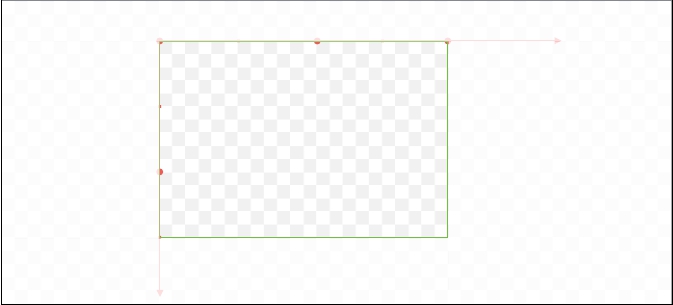
100

SVG 的世界是⽆无限⼤大的，世界有⼀一个坐标系，这个坐标系就是⽤用户坐标系。

我们设置的 viewbox，也就是视野的⼤大⼩小，就说就是观察⽤用户坐标系中的哪个区域。⽐比如说，现在设置 viewbox 为 (0,0,200,150)

⽤用户坐标系是最原始的坐标系，其它产⽣生的坐标系都从⽤用户坐标系开始，所以⽤用户坐标系也可以称之为原始坐标系（initial coordinate）

2.4.1.  用户坐标系（User  Coordinate）



**Ouser** 100 200

**viewBox=“0 0 200 150"**

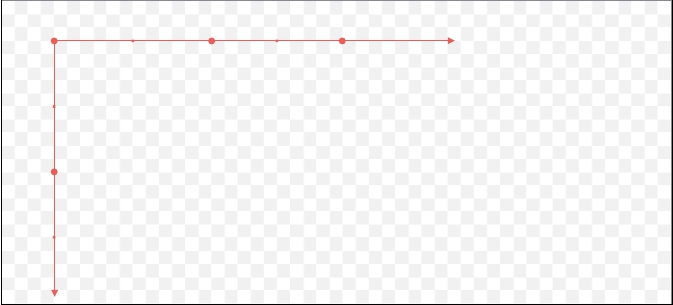
100

SVG 的世界是⽆无限⼤大的，世界有⼀一个坐标系，这个坐标系就是⽤用户坐标系。

我们设置的 viewbox，也就是视野的⼤大⼩小，就说就是观察⽤用户坐标系中的哪个区域。⽐比如说，现在设置 viewbox 为 (0,0,200,150)

⽤用户坐标系是最原始的坐标系，其它产⽣生的坐标系都从⽤用户坐标系开始，所以⽤用户坐标系也可以称之为原始坐标系（initial coordinate）

2.4.2.  自身坐标系（Current  Coordinate）



**Ouser** 100 200

100

什么是⾃自⾝身坐标系呢？⾃自⾝身坐标系就是每个图形或者是分组与⽣生俱来的⼀一个坐标系。我们来看⼀一个例⼦子，现在我绘制了⼀一个矩形。

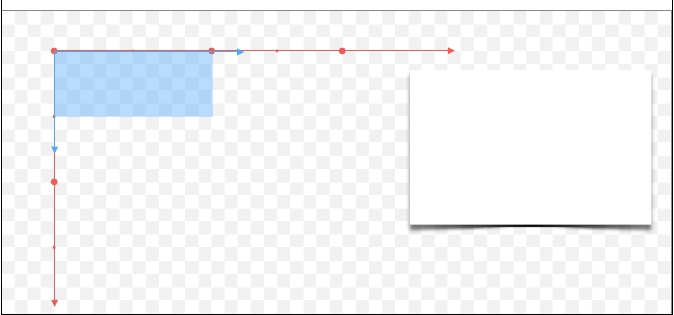
!

那么这个矩形就会⾃自⾝身带着⼀一个坐标系，成为这个矩形的⾃自⾝身坐标系。这个坐标系⽤用于给矩形定义⾃自⼰己的形状。⽐比如 x, y 坐标以及宽⾼高，都是基于⾃自⾝身坐标系进⾏行定义的。

!

⾃自⾝身坐标系本⾝身不太好理解，我们来结合前驱坐标系⼀一同理解。

2.4.2.  自身坐标系（Current  Coordinate）



**Ouser** 100 200

**<svg>**

**<rect**  id="a"

x="0"

y="0"

width="100"

100

height="50"**>**

**</rect>**

**</svg>**

什么是⾃自⾝身坐标系呢？⾃自⾝身坐标系就是每个图形或者是分组与⽣生俱来的⼀一个坐标系。我们来看⼀一个例⼦子，现在我绘制了⼀一个矩形。

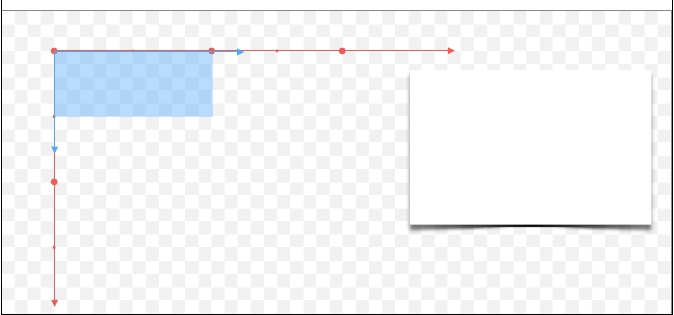
!

那么这个矩形就会⾃自⾝身带着⼀一个坐标系，成为这个矩形的⾃自⾝身坐标系。这个坐标系⽤用于给矩形定义⾃自⼰己的形状。⽐比如 x, y 坐标以及宽⾼高，都是基于⾃自⾝身坐标系进⾏行定义的。

!

⾃自⾝身坐标系本⾝身不太好理解，我们来结合前驱坐标系⼀一同理解。

2.4.2.  前驱坐标系（Previous  Coordinate）



**Ouser** 100 200

**<svg>**!

**<rect**  id="a"!

x="0"  !

y="0"  !

width="100"  !

100

height="50"**>**!

**</rect>**!

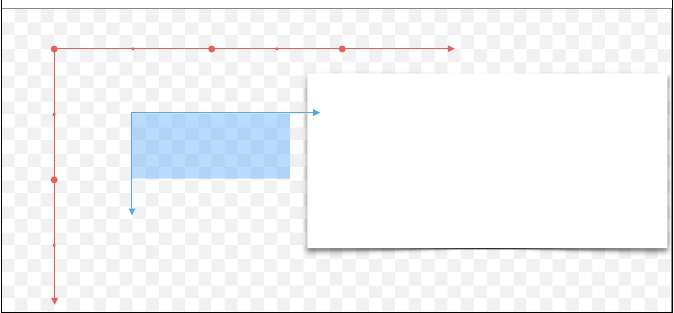
**</svg>**

前驱坐标系，就是⽗父容器的坐标系。现在矩形的⽗父容器是 SVG 标签，那么它的前驱坐标系也就是世界坐标系。

!

坐标变换，就是前驱坐标系到⾃自⾝身坐标系的⼀一个变换。现在给矩形加⼀一个坐标变换。

2.4.3.  前驱坐标系（Previous  Coordinate）



**Ouser** 100 200

100

**Oa**

**<svg>**!

**<rect**  id="a"!

x="0"  !

y="0"  !

width="100"  !

height="50"!

!  !  !  !**transform="translate(50,50)">**!

**</rect>**!

**</svg>**

- 前驱坐标系，就是⽗父容器的坐标系。现在矩形的⽗父容器是 SVG 标签，那么它的前驱坐标系也就是世界坐标系。

- 坐标变换，就是前驱坐标系到⾃自⾝身坐标系的⼀一个线性变换。现在给矩形加⼀一个坐标变换。

- ⼤大家观察⼀一下，就会发现，矩形的坐标和宽⾼高其实是没有改变的，x 和 y 依然是 0，⽽而宽⾼高依然是100x50。x、y和宽⾼高是基于⾃自⾝身坐标系来定义的，定义了是多少就是多少。

- 变的是什么？变的是矩形的⾃自⾝身坐标系，它相对于他的前驱坐标系发⽣生了⼀一个变换。

2.4.3.  自身坐标系和前驱坐标系  ·∙  例子



**OA**

**OB   OC**

50

**<svg  id="A"**  xmlns="..."**>**!

**<g  id="B"**!

**OD**

100

50

100

transform="translate(0,  50)"**>**  !

**<rect  id="C"**  x="100"  y="50"  !

50

width="100"  height="50"**>**!

**</rect>**!

140

100

100

**<rect  id="D"**  x="140"  y="100"  !

width="20"  height="120"**>**!

**</rect>**!

**</g>**!

**</svg>**!

20

还是以锤⼦子为例。可以看到，SVG 标签 A ⾥里嵌套了⼀一个分组 B，⽽而 B 呢⾥里⾯面有两个矩形，C 和 D。那么在这⾥里，问⼤大家⼏几个问题。

-   世界坐标系是哪⼀一个？OA

-   B 的前驱坐标系是哪⼀一个？OA

-   C 和 D 的图形定义（坐标和宽⾼高）是基于哪个坐标系的？OC 和 OD 两个⾃自⾝身坐标系

-   C 和 D 的前驱坐标系是哪⼀一个？OB

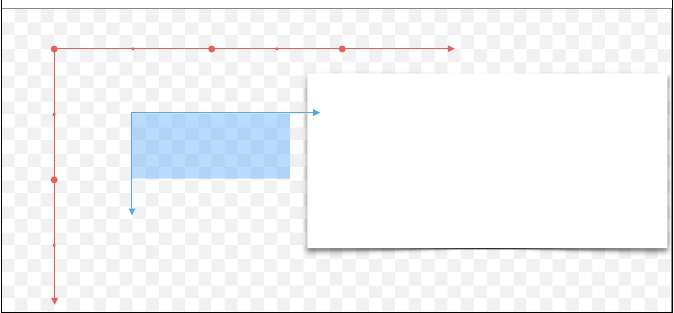
-   OB 是哪个元素的⾃自⾝身坐标系？分组 B

-   分组 B 上设置的 transform 属性是什么意思？表⽰示 B 的⾃自⾝身坐标系 OB 是从其前驱坐标系 OA 经过 transform 变换⽽而来的。

-   OB、OC和OD为什么重合？因为OB是OC和OD的前驱坐标系，⽽而C和D上都没有定义transform属性，所以C和D的⾃自⾝身坐标系和前驱坐标系重合了。

好，整理了锤⼦子的各种问题之后，再来看看最后⼀一个坐标系。

2.4.4.  参考坐标系（Reference  Coordinate）



**Ouser** 100 200

100

**Oa**

**<svg>**!

**<rect**  id="a"!

x="0"  !

y="0"  !

width="100"  !

height="50"!

!  !  !  !**transform="translate(50,50)">**!

**</rect>**!

**</svg>**

参考坐标系，其实是任意的⼀一个坐标系。使我们选区的⽤用于观察某个图形情况的坐标系。⽐比如说还是图中的矩形，我选取世界坐标系作为参考坐标系来观察这个矩形的时候，矩形的坐标是多少？没错，就是 50 x 50。宽⾼高是

多少？100 x 50。

!

那么这种观察在实际当中有什么意义呢？我举⼀一个例⼦子⼤大家就知道了。⽐比如说我做图形编辑器，需要做⼀一个对⻬齐的功能。那么所谓对⻬齐，⼀一定是指在某个坐标系中对⻬齐，因为图形的⾃自⾝身坐标系⼀一般都是经过变换的。这个时

候，就需要选取⼀一个公共的参考坐标系，对图形进⾏行观察，获得它们的坐标点，然后索引，对⻬齐。

2.4.  四个坐标系



·   用户坐标系（User  Coordinate） 

‣  世界的坐标系 

·   自身坐标系（Current  Coordinate） 

‣  每个图形元素或分组独立与生俱来 

·   前驱坐标系（Previous  Coordinate） 

‣  父容器的坐标系 

·   参考坐标系（Reference  Coordinate） 

‣  使用其它坐标系来考究自身的情况时使用

好，现在回过头来看看这四个坐标系。

!

⽤用户坐标系，也叫做原始坐标系，是指世界上的⼀一个坐标系。视野的定义是基于⽤用户坐标系描述的。

!

希望这四个坐标系的概念同学们好好斟酌理解。

!

每⼀一个图形或者分组都会产⽣生⼀一个⾃自⾝身坐标系，⾃自⾝身坐标系⽤用于定义⾃自⾝身的⼀一些图形属性。

!

⽗父容器的坐标叫做前驱坐标系，前驱坐标系经过元素的 transform 属性进⾏行变换之后形成了图形的⾃自⾝身坐标系。

!

⽽而参考坐标系，则是我在对某个图形进⾏行观察、测量的时候，使⽤用的⼀一个坐标系。

Lesson  2  -  SVG  中的坐标系统和坐标变换



2.1.  SVG  的世界、视野、视窗的概念 

2.2.  SVG  中的图形分组 

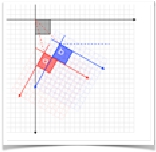
2.3.  坐标系统概述 

2.4.  四个坐标系 

2.5.  坐标变换

好，刚刚说了前驱坐标系经过变换形成⾃自⾝身坐标系，现在就详细地介绍⼀一下坐标变换的概念。

2.5.  坐标变换



·  定义

·  线性变换

·  线性变换列表

·  transform 属性

坐标变换主要讲述四个⽅方⾯面的内容，先来看⼀一下坐标变换的定义。

2.5.1.  坐标变换定义



·  数学上，「坐标变换」是采⽤用⼀一定

的数学⽅方法将⼀一个坐标系的坐标变

换为另⼀一个坐标系的坐标的过程。

·  SVG 中，「坐标变换」是对⼀一个坐

标系到另⼀一个坐标系的变换的描述

2.5.2.  线性变换

*b* *d*⎢

*f* ⎥



·  线性变换⽅方程

X’ = aX + cY + e

Y’ = bX + dY + f

·  变换矩阵，记为 M

⎡*a* *c* *e*   ⎤

⎢ ⎥

⎣ ⎦

⎢   0 0 1   ⎥

在 2D 平⾯面上，⼀一般我们都使⽤用线性变换来满⾜足我们的变换需求。SVG 当中也使⽤用线性变换来表⽰示两个坐标系之间的变换。

!

线性变换的变换⽅方程就是⼀一个线性⽅方程，⽅方程的六个参数 a, b, c, d, e, f，可以记为矩阵的形式。这个矩阵，就称为线性变换矩阵。（切）

2.5.2.  线性变换



·  线性变换⽅方程

⎢*b* *d*

*f* ⎥

0 1 10  ⎥

X’ = aX + cY + e

Y’ = bX + dY + f

·  变换矩阵，记为 M

⎡*a* *c* *e*   ⎤

⎢ ⎥

⎣ ⎦

⎢   0 0 1   ⎥

OA

OB

⎡  1    0    10  ⎤

⎢                      ⎥

⎢

⎢  0    0     1   ⎥

⎣ ⎦

X’ = 1\*X + 0\*Y + 10 = X + 10

Y’ = 0\*X + 1\*Y + 10 = Y + 10

线性变换⽅方程的意思是，原坐标系上的每个点，经过线性运算之后，得到新坐标系上的每个点。

!

图上的变换矩阵就表⽰示 OB 上每个点都是 OA 上每个点横纵坐标都加10⽽而来。

!

下⾯面看⼏几个常⽤用的线性变换的例⼦子。

2.5.2.  线性变换



·  线性变换⽅方程

⎢*b* *d*

*f* ⎥

0 1 10  ⎥

X’ = aX + cY + e

Y’ = bX + dY + f

·  变换矩阵，记为 M

⎡*a* *c* *e*   ⎤

⎢ ⎥

⎣ ⎦

⎢   0 0 1   ⎥

OA

OB

⎡  1    0    10  ⎤

⎢                      ⎥

⎢

⎢  0    0     1   ⎥

⎣ ⎦

X’ = 1\*X + 0\*Y + 10 = X + 10

Y’ = 0\*X + 1\*Y + 10 = Y + 10

线性变换⽅方程的意思是，原坐标系上的每个点，经过线性运算之后，得到新坐标系上的每个点。

!

图上的变换矩阵就表⽰示 OB 上每个点都是 OA 上每个点横纵坐标都加10⽽而来。

!

下⾯面看⼏几个常⽤用的线性变换的例⼦子。

线性变换

0 1   10  ⎥



平移

OA

OB

⎡  1 0   10  ⎤

⎢ ⎥

⎢

⎢  0 0 1   ⎥

⎣ ⎦

平移⽐比较简单，就是刚才看到的例⼦子。

线性变换

0 1   10  ⎥



平移

OA

OB

⎡  1 0   10  ⎤

⎢ ⎥

⎢

⎢  0 0 1   ⎥

⎣ ⎦

平移⽐比较简单，就是刚才看到的例⼦子。

线性变换



旋转

旋转可以通过极坐标的形式求出变换矩阵。

·  使⽤用极坐标求变换矩阵

OA OB

线性变换



旋转

旋转可以通过极坐标的形式求出变换矩阵。

⎩*Y*  =*r* i sin(α)

·  使⽤用极坐标求变换矩阵

极坐标⽅方程：

⎧*X* =*r* ⋅ cos(α)

⎨

OA OB

线性变换



旋转

旋转可以通过极坐标的形式求出变换矩阵。

⎩*Y*  =*r* i sin(α)

⎩*Y* ' =*r* i sin(α + θ )

·  使⽤用极坐标求变换矩阵

极坐标⽅方程：

⎧*X* =*r* ⋅ cos(α)

⎨

旋转 θ 度后：

⎧*X* ' =*r* ⋅ cos(α + θ )

⎨

OA OB

线性变换



旋转

·  使⽤用极坐标求变换矩阵

OA OB

极坐标⽅方程：

⎩*Y*  =*r* i sin(α)

⎩*Y* ' =*r* i sin(α + θ )

⎧⎪*X* ' =*r* ⋅*cos*(α )*cos*(θ ) −*r* ⋅*sin*(α )*sin*(θ ) =*cos*(θ )*X* −*sin*(θ )*Y*   +   0

⎪⎩*Y* ' =*r* ⋅*cos*(α )*sin*(θ ) +*r* ⋅*sin*(α )*cos*(θ ) =*sin*(θ )*X* +*cos*(θ )*Y*   +   0

⎧*X* =*r* ⋅ cos(α)

⎨

旋转 θ 度后：

⎧*X* ' =*r* ⋅ cos(α + θ )

⎨

展开：

⎨

旋转可以通过极坐标的形式求出变换矩阵。

线性变换



旋转

·  使⽤用极坐标求变换矩阵

OA  O

⎩*Y*  =*r* i sin(α)

⎩*Y* ' =*r* i sin(α + θ )

⎢   sin(30°)

cos(30°)

1   ⎥⎦

⎣

极坐标⽅方程：

⎧*X* =*r* ⋅ cos(α)

⎨

旋转 θ 度后：

⎧*X* ' =*r* ⋅ cos(α + θ )

⎨

B

⎡   cos(30°)    − sin(30°)    0   ⎤

⎢                                                      ⎥

0   ⎥

⎢          0                    0

展开：

⎧⎪*X* ' =*r* ⋅*cos*(α )*cos*(θ ) −*r* ⋅*sin*(α )*sin*(θ ) =*cos*(θ )*X* −*sin*(θ )*Y*   +   0

⎪⎩*Y* ' =*r* ⋅*cos*(α )*sin*(θ ) +*r* ⋅*sin*(α )*cos*(θ ) =*sin*(θ )*X* +*cos*(θ )*Y*   +   0

⎨

线性变换



旋转

·  使⽤用极坐标求变换矩阵

⎩*Y*  =*r* i sin(α)

⎩*Y* ' =*r* i sin(α + θ )

⎢   sin(30°)

cos(30°)

1   ⎥⎦

⎣

极坐标⽅方程：

⎧*X* =*r* ⋅ cos(α)

⎨

旋转 θ 度后：

⎧*X* ' =*r* ⋅ cos(α + θ )

⎨

OA  O

⎡   cos(30°)    − sin(30°)    0   ⎤

⎢                                                      ⎥

0   ⎥

⎢          0                    0

展开：

⎧⎪*X* ' =*r* ⋅*cos*(α )*cos*(θ ) −*r* ⋅*sin*(α )*sin*(θ ) =*cos*(θ )*X* −*sin*(θ )*Y*   +   0

⎪⎩*Y* ' =*r* ⋅*cos*(α )*sin*(θ ) +*r* ⋅*sin*(α )*cos*(θ ) =*sin*(θ )*X* +*cos*(θ )*Y*   +   0

⎨

线性变换



缩放

·  a 和 c 直观控制缩放

OA OB

线性变换



缩放

·  a 和 c 直观控制缩放

0 2 0  ⎥

OA OB

⎡  2    0    0  ⎤

⎢                     ⎥

⎢

⎢⎣  0    0    1  ⎥⎦

2.5.3.  线性变换列表



·  表⽰示⼀一系列的变换，

结果为变换的矩阵的

乘积

OA

OB

M = Mn · Mn-1 · ...  · M2 · M1 · M0

⎥ i ⎢

⎡  1 0 10   ⎤   ⎡  cos(30°) − sin(30°) 0   ⎤

⎢  0 1 10   ⎥   ⎢   sin(30°) cos(30°) 0   ⎥

·  后⾯面的变换乘在前⾯面

⎢ ⎥

⎥   ⎣ ⎦⎢

⎣  0 0 1   ⎦   ⎢ 0 0 1   ⎥

Mtranslate · Mrotate

单个线性变换矩阵，可以表⽰示所有的线性变换。但是，⼀一般我们去描述⼀一个线性变换可能更愿意分开⼀一步步来描述。⽐比如说，先旋转30度，再平移（10，10），那么已然可

以有⼀一个变换矩阵可以表⽰示这个线性变换列表的结果，那就就是每⼀一步变换矩阵的乘积。

!

需要注意的是，最后⾯面的变换，需要乘在前⾯面。这是线性代数当中的⼀一个结论，有兴趣的同学可以⾃自⾏行了解⼀一下。

!

2.5.3.  线性变换列表



·  表⽰示⼀一系列的变换，

结果为变换的矩阵的

乘积

M = Mn · Mn-1 · ...  · M2 · M1 · M0

OA

OC

OB

Mtranslate · Mrotate

·  后⾯面的变换乘在前⾯面

Mrotate · Mtranslate

2.5.4.  transform属性



·  前驱坐标系：⽗父容器的坐标系

·  transform属性：定义前驱坐标系到⾃自⾝身坐标系的线性变换

·  语法：

‣   rotate(<deg>)\*

‣   translate(<x>,<y>)\*

‣   scale(<sx>,<sy>)\*

‣   matrix(<a>,<b>,<c>,<d>,<e>,<f>)\*

SVG 当中提供了 transform 属性为我们来定义线性变换列表。

2.6.  坐标观察



·  getBBox()

‣   获得当前元素所占的矩形区域

·  getCTM()

‣   获得视窗坐标系到当前元素⾃自⾝身坐标系的变换矩阵

·  getScreenCTM()

‣   获得浏览器坐标系到当前元素⾃自⾝身坐标系的变换矩阵

·  getTransformToElement()

‣   获得从指定元素的⾃自⾝身坐标系到当前元素的⾃自⾝身坐标系的变换矩阵

刚刚的例⼦子⾥里头，我提到了观察某个坐标系中的元素在参考坐标系中的坐标。这种⾏行为，可以称为坐标观察。

!

坐标观察还可以解决很多问题，⽐比如，交互的时候，我希望知道我点击的⿏鼠标位置在指定的坐标系中是哪个位置。

!

SVG 所有元素都提供了四个⽅方法来配合坐标观察。