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

坐标系统这一部分，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较高级的知识来介绍，但是我认为这部分知识非常重要，因为它影响你在实际开发应用中方方面面的理解。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

下面来看一下这节课的内容。

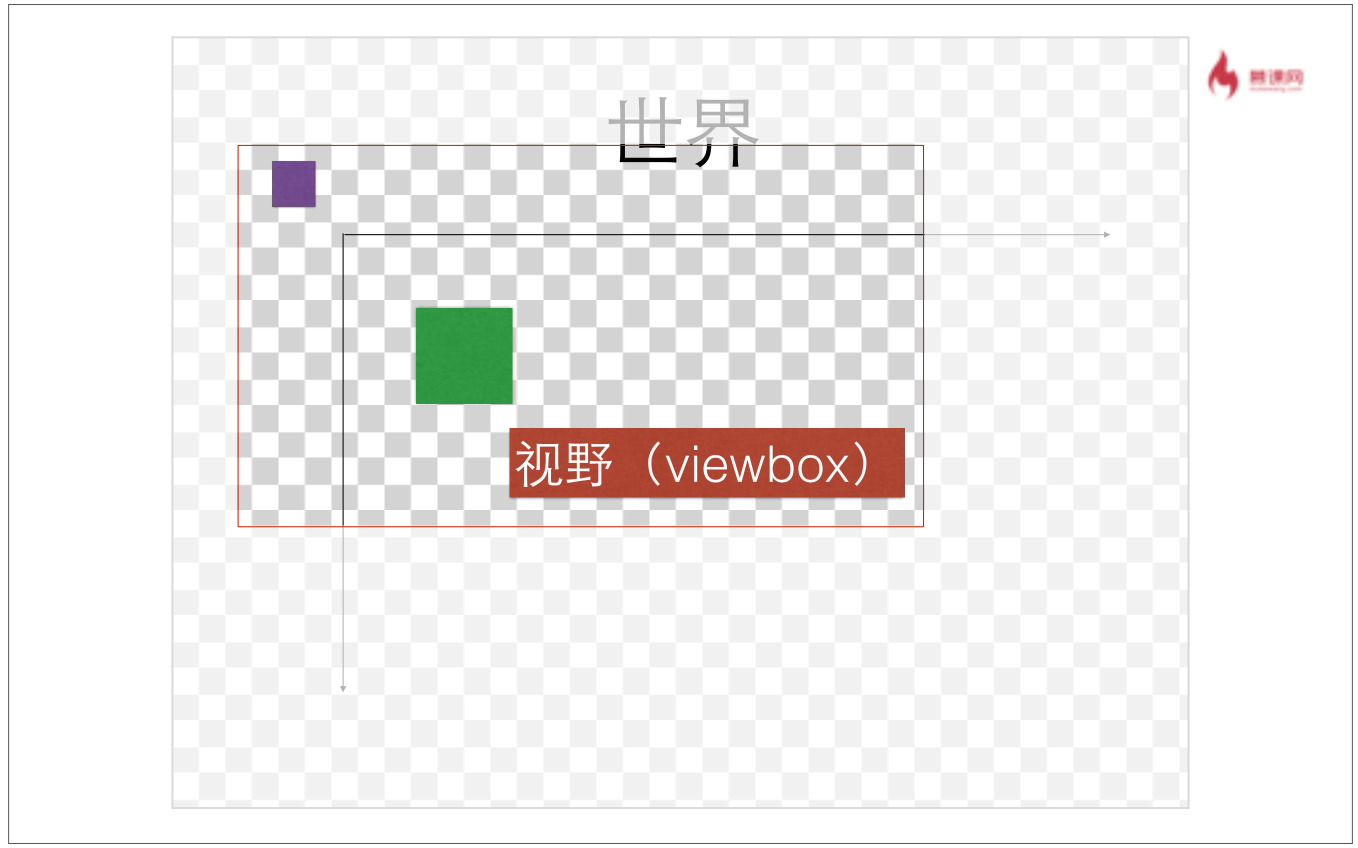
视野与世界

- 世界是无穷大的
- 视野是观察世界的一个矩形区域

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

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个无穷大的世界。

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，大部分绘图API都提供视野的控制方法，像在SVG中，Viewbox来控制视野。



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

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个无穷大的世界。

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，大部分绘图API都提供视野的控制方法，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

```
<svg xmlns="..."
     width="800" height="600"
     viewBox="0 0 400 300"
     preserveAspectRatio="xMidYMid meet">
     <!--SVG Content-->
</svg>
```

在 SVG 标签当中可以指定一个宽和高属性，来表示 SVG 文件渲染的区域大小。这个大小也可以使用样式表来定义。这个区域大小，就是视窗。视窗实际上就是浏览器开辟出来用于渲染 SVG 内容的一个区域。

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

而视野，也就是观看世界的矩形区域是哪一个，使用 viewBox 进行定义。

这里出现了视窗和视野，在理想情况下，视野和视窗有一样的尺寸，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小不一致，就存在如何控制这个填充的问题，填充的策略使用 preserveAspectRatio 进行指定。

锤子的故事

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

从前有一个画家

他很擅长画锤子

先画一个矩形作为锤头，再画一个矩形作为手柄。大功告成。

他很擅长画锤子

先画一个矩形作为锤头，再画一个矩形作为手柄。大功告成。

他很擅长画锤子

先画一个矩形作为锤头，再画一个矩形作为手柄。大功告成。

有一天他改行当程序员

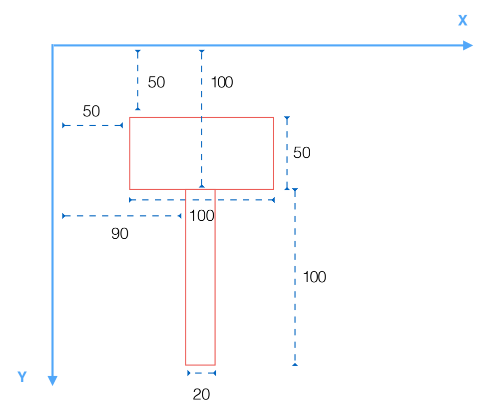
由于种种原因，改行当程序员。

老板说

“你用代码画一个锤子吧”

老板知道他是画锤子出生的，就说，“你用程序画一个锤子吧”

太简单了



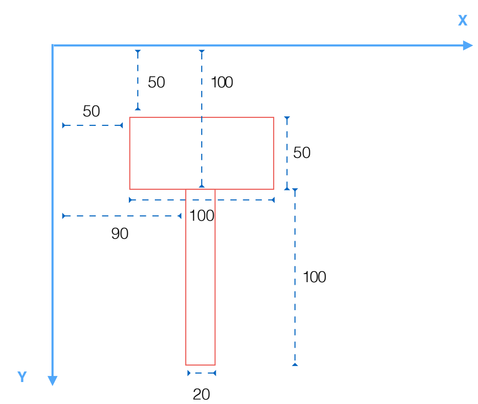

```
<svg xmlns="http://www.w3.org/2000/svg">
    <rect x="50" y="50"
        width="100" height="50"
        stroke="red" fill="none">
    </rect>
    <rect x="90" y="100"
        width="20" height="120"
        stroke="red" fill="none">
    </rect>
</svg>
```

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

老板又说

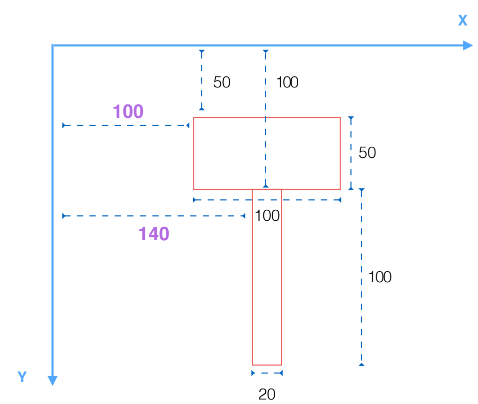
“锤子往右挪50像素吧”

没问题




```
<svg xmlns="http://www.w3.org/2000/svg">
    <rect x="50" y="50"
        width="100" height="50"
        stroke="red" fill="none">
    </rect>
    <rect x="90" y="100"
        width="20" height="120"
        stroke="red" fill="none">
    </rect>
</svg>
```

没问题



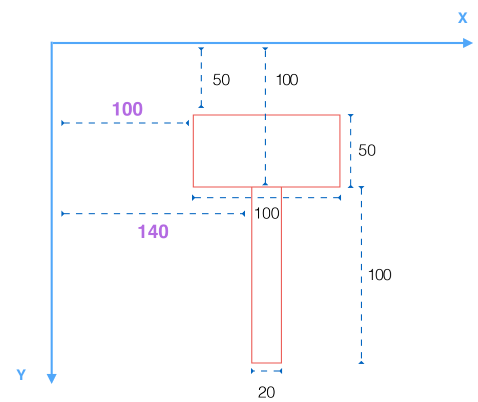

```
<svg xmlns="http://www.w3.org/2000/svg">
    <rect x="100" y="50"
        width="100" height="50"
        stroke="red" fill="none">
    </rect>
    <rect x="140" y="100"
        width="20" height="120"
        stroke="red" fill="none">
    </rect>
</svg>
```

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，大功告成。

老板还不满意

“我想要一把绿色的锤子”

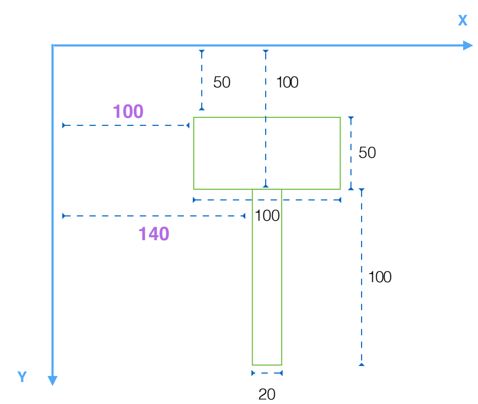
没问题2




```
<svg xmlns="http://www.w3.org/2000/svg">
    <rect x="100" y="50"
        width="100" height="50"
        stroke="red" fill="none">
    </rect>
    <rect x="140" y="100"
        width="20" height="120"
        stroke="red" fill="none">
    </rect>
</svg>
```

他又把两个矩形的 stroke 属性修改为 green

没问题2




```
<svg xmlns="http://www.w3.org/2000/svg">
    <rect x="100" y="50"
        width="100" height="50"
        stroke="green" fill="none">
    </rect>
    <rect x="140" y="100"
        width="20" height="120"
        stroke="green" fill="none">
    </rect>
</svg>
```

他又把两个矩形的 stroke 属性修改为 green

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

其实，无论是锤子也好，小米也好，瑞士军刀抑或苹果也好，老板都只是要针对图形的整体进行修改，而不是修改其细节。既然逻辑上它们是一个整体，那么，在代码里也应该是有一个整体的概念。有了整体的概念，就可以有整体的操作。

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

其实，无论是锤子也好，小米也好，瑞士军刀抑或苹果也好，老板都只是要针对图形的整体进行修改，而不是修改其细节。既然逻辑上它们是一个整体，那么，在代码里也应该是有一个整体的概念。有了整体的概念，就可以有整体的操作。

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

其实，无论是锤子也好，小米也好，瑞士军刀抑或苹果也好，老板都只是要针对图形的整体进行修改，而不是修改其细节。既然逻辑上它们是一个整体，那么，在代码里也应该是有一个整体的概念。有了整体的概念，就可以有整体的操作。

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

其实，无论是锤子也好，小米也好，瑞士军刀抑或苹果也好，老板都只是要针对图形的整体进行修改，而不是修改其细节。既然逻辑上它们是一个整体，那么，在代码里也应该是有一个整体的概念。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

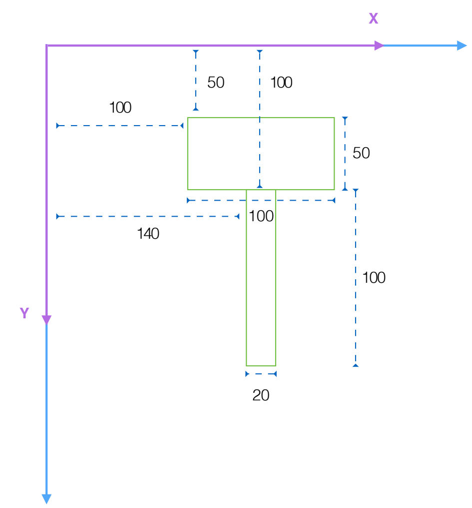
下面，就给大家讲一下 SVG 当中分组的概念。

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，就给大家讲一下 SVG 当中分组的概念。

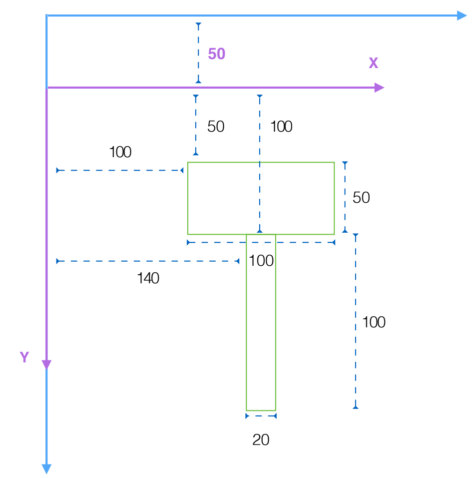
2.2. SVG 中的图形分组




```
<svg xmlns="...">
    <g stroke="green" fill="none">
        <rect x="100" y="50"
            width="100" height="50">
        </rect>
        <rect x="140" y="100"
            width="20" height="120">
        </rect>
    </g>
</svg>
```

还是拿充满情怀的锤子来说，画家修改了一下上面的代码，用一个 <g> 标签把两个锤子包起来了。然后，把描边和填充属性设置在 <g> 标签上。现在，这个代表了锤子的 <g> 标签就可以作为一个整体进行操作。

2.2. SVG 中的图形分组




```
<svg xmlns="...">
    <g stroke="green" fill="none"
        transform="translate(0, 50)">
        <rect x="100" y="50"
            width="100" height="50">
        </rect>
        <rect x="140" y="100"
            width="20" height="120">
        </rect>
    </g>
</svg>
```

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

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

那么你也许也注意到呢老师已经偷偷地把第四小节的标题给改了，之前打算讲自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常的支持和努力，老师呢现在决定买二送二，一次性给你讲明白四个坐标系。

好，下面开始上课。

Lesson 2 - SVG 中的坐标系统和坐标变换

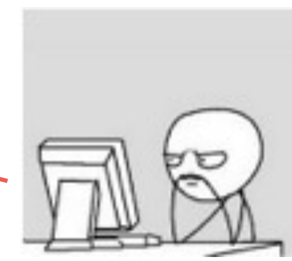
2.1. SVG 的世界、视野、视窗的概念

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



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

那么你也许也注意到呢老师已经偷偷地把第四小节的标题给改了，之前打算讲自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常的支持和努力，老师呢现在决定买二送二，一次性给你讲明白四个坐标系。

好，下面开始上课。

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

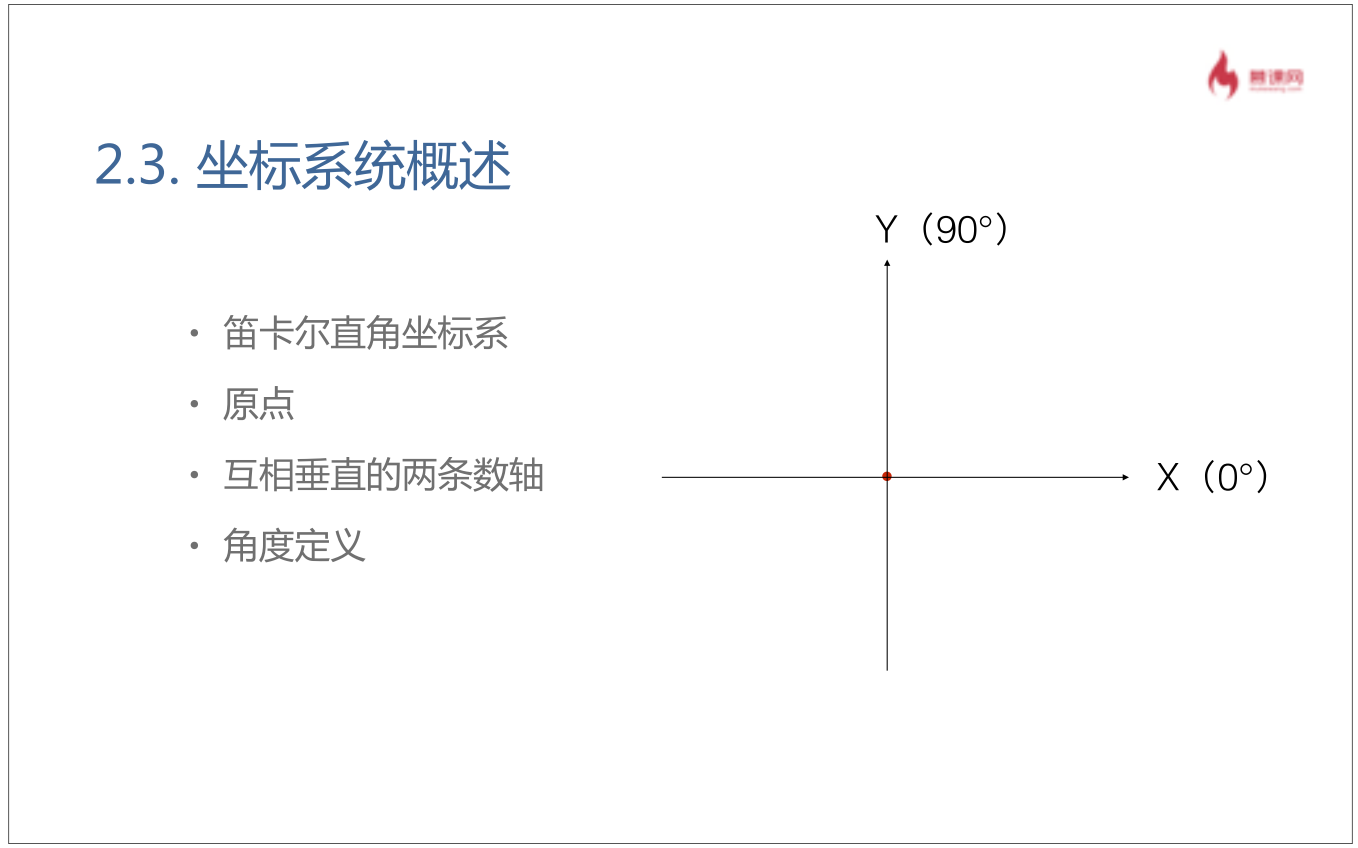
坐标系统的作用是为图形提供统一的定位基准，为了做到这点，笛卡尔直角坐标系定义了包括一个原点以及两条相互垂直的数轴。基于原点和数轴的定义，又可以定义角度的值的含义。

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 使用的坐标系统是笛卡尔直角坐标系统，本课讨论的坐标系统，无特殊说明，都是指笛卡尔直角坐标系。

坐标系统的作用是为图形提供统一的定位基准，为了做到这点，笛卡尔直角坐标系定义了包括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，又可以定义角度的值的含义。

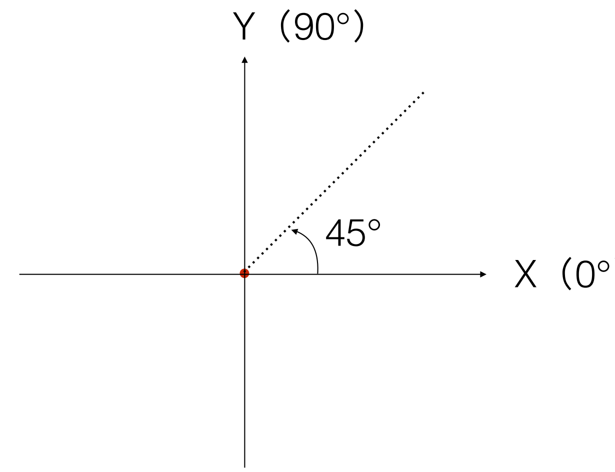


先来简单看看坐标系统的概述。SVG 使用的坐标系统是笛卡尔直角坐标系统，本课讨论的坐标系统，无特殊说明，都是指笛卡尔直角坐标系。

坐标系统的作用是为图形提供统一的定位基准，为了做到这点，笛卡尔直角坐标系定义了包括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，又可以定义角度的值的含义。

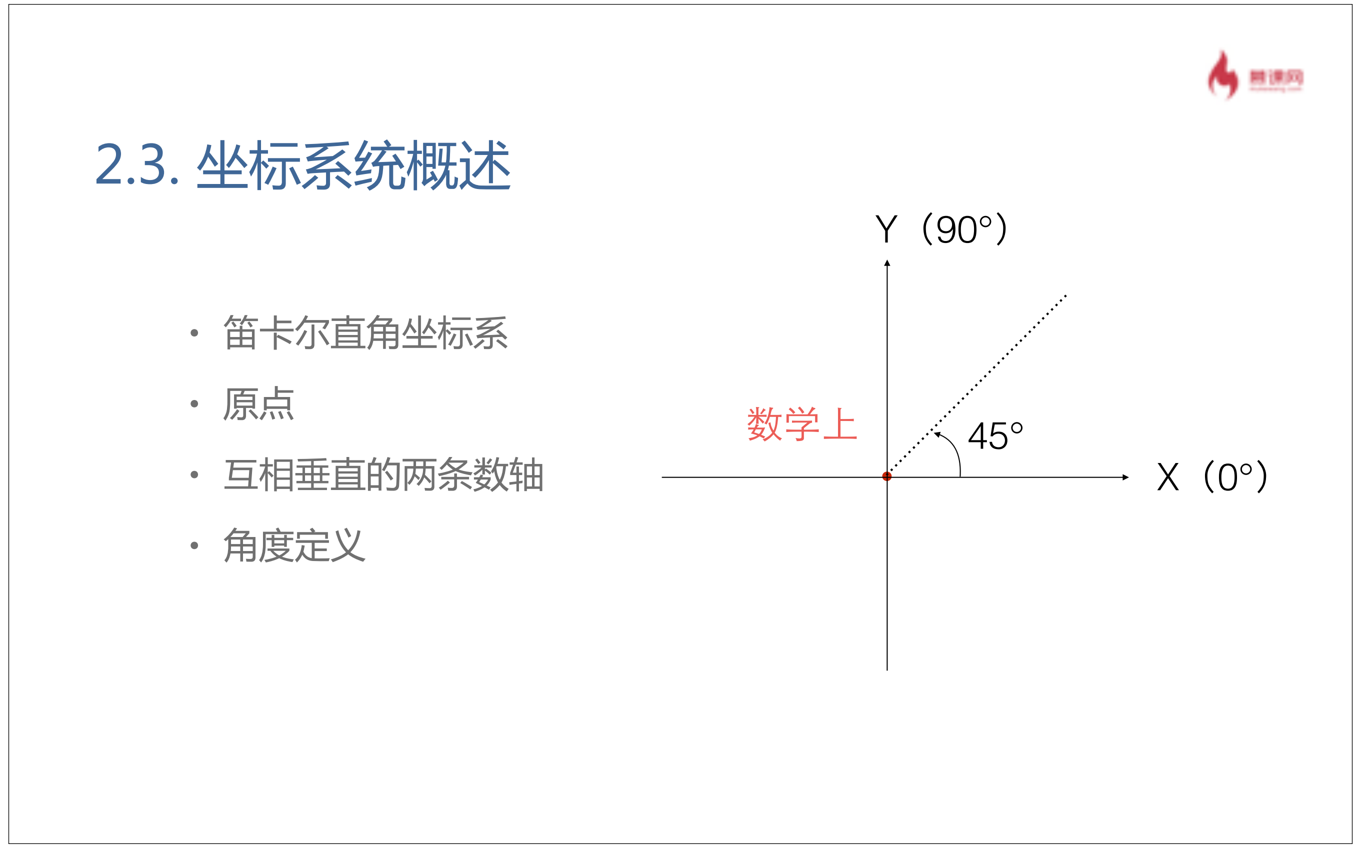
2.3. 坐标系统概述

- 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 使用的坐标系统是笛卡尔直角坐标系统，本课讨论的坐标系统，无特殊说明，都是指笛卡尔直角坐标系。

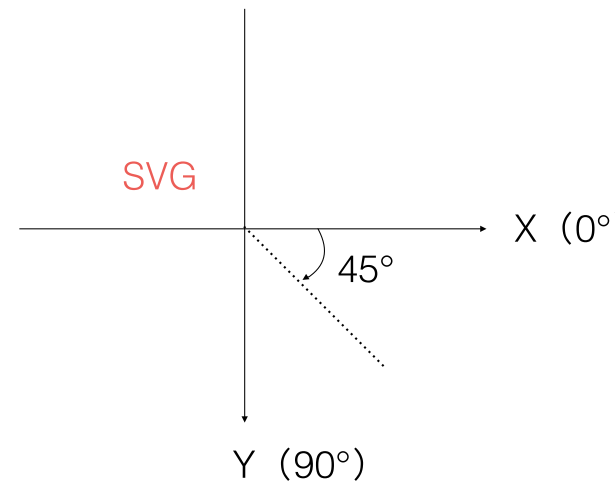
坐标系统的作用是为图形提供统一的定位基准，为了做到这点，笛卡尔直角坐标系定义了包括一个原点以及两条相互垂直的数轴。基于原点和数轴的定义，又可以定义角度的值的含义。



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

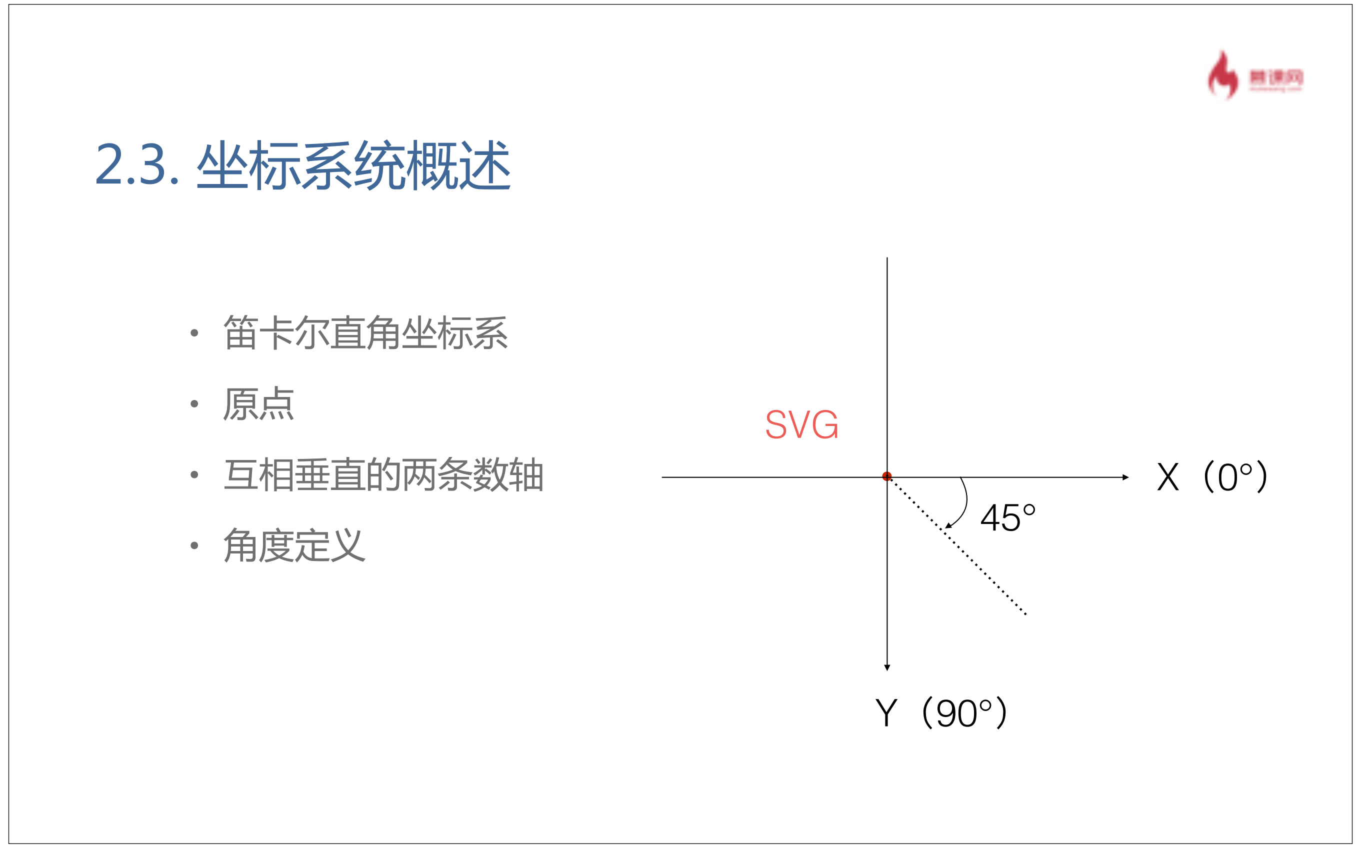
2.3. 坐标系统概述

- 笛卡尔直角坐标系
- 原点
- 互相垂直的两条数轴
- 角度定义



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

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



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

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

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

接下来给大家讲一下 SVG 里面四个需要大家记住坐标系。

这四个坐标系呢，分别叫用户坐标系、自身坐标系、前驱坐标系、参考坐标系。

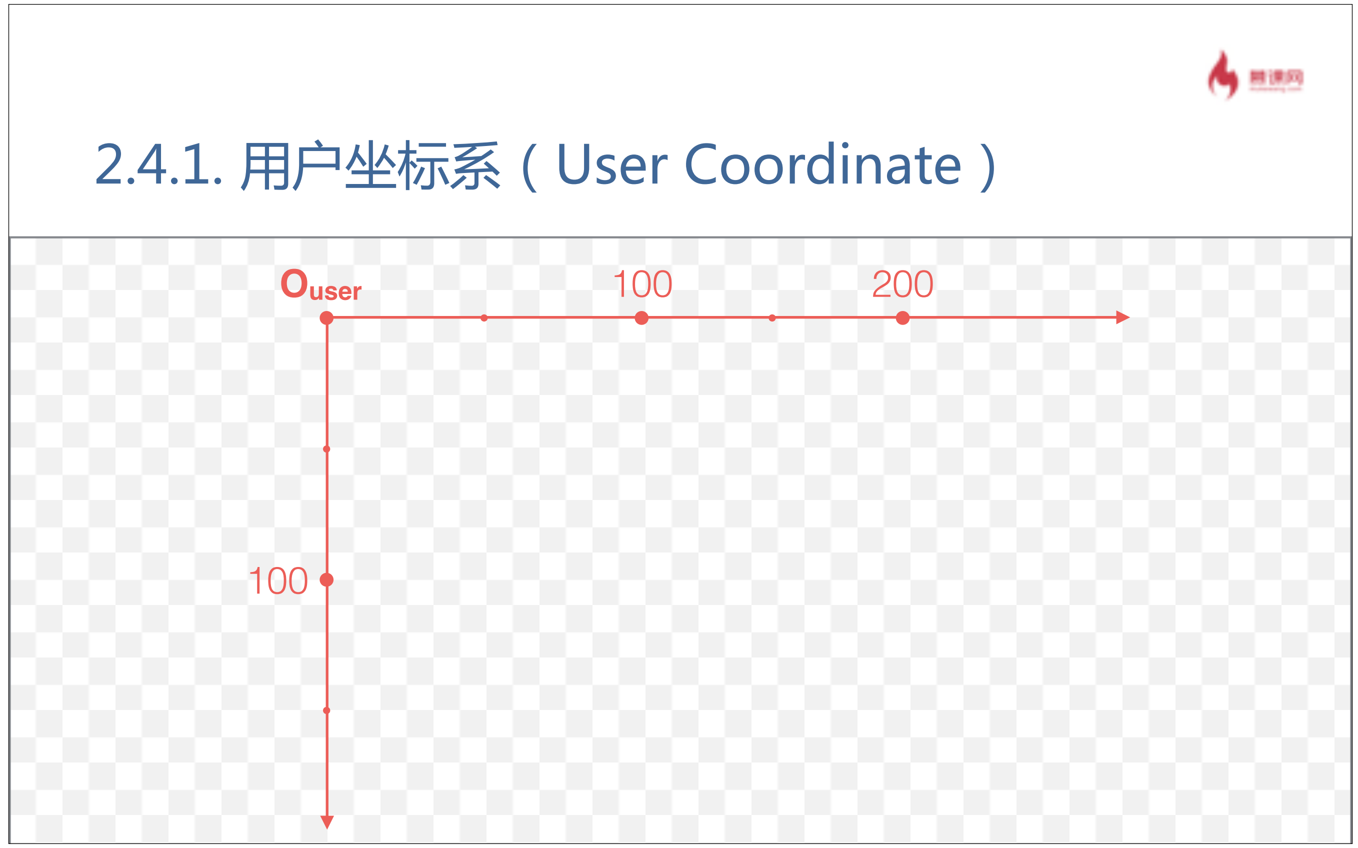
回顾一下世界的概念。

2.4.1. 用户坐标系（User Coordinate）

SVG 的世界是无限大的，世界有一个坐标系，这个坐标系就是用户坐标系。

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

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

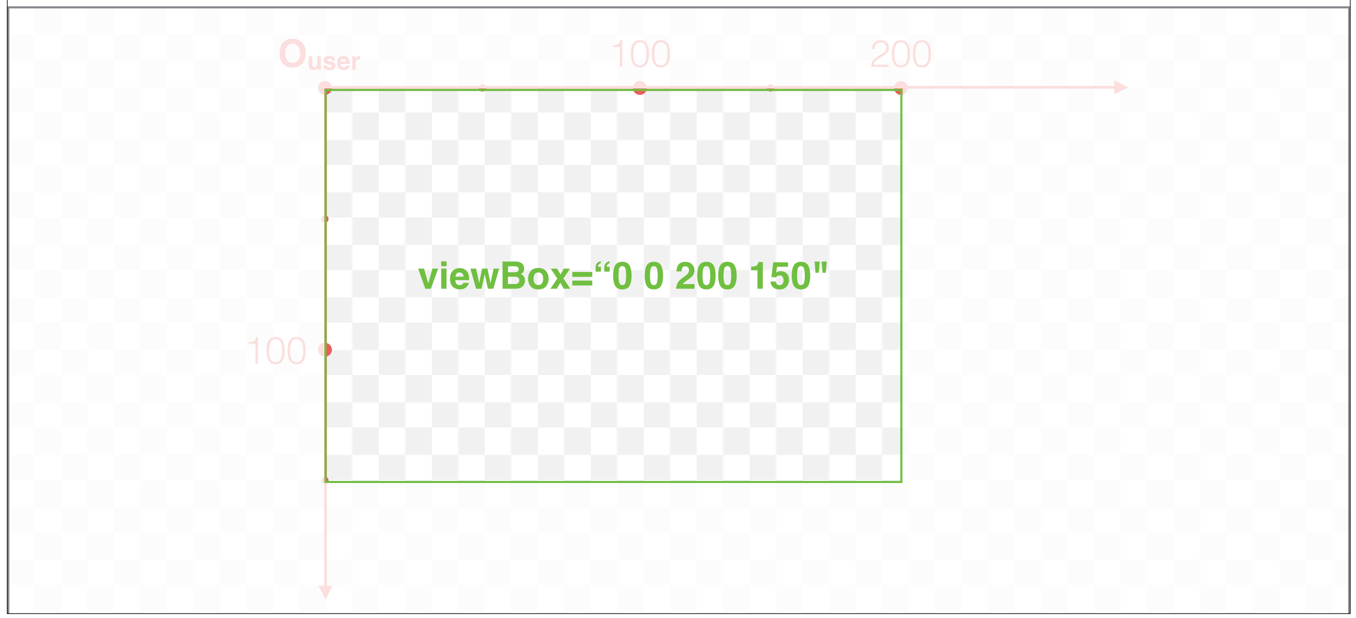


SVG 的世界是无限大的，世界有一个坐标系，这个坐标系就是用户坐标系。

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

用户坐标系是最原始的坐标系，其它产生的坐标系都从用户坐标系开始，所以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）



SVG 的世界是无限大的，世界有一个坐标系，这个坐标系就是用户坐标系。

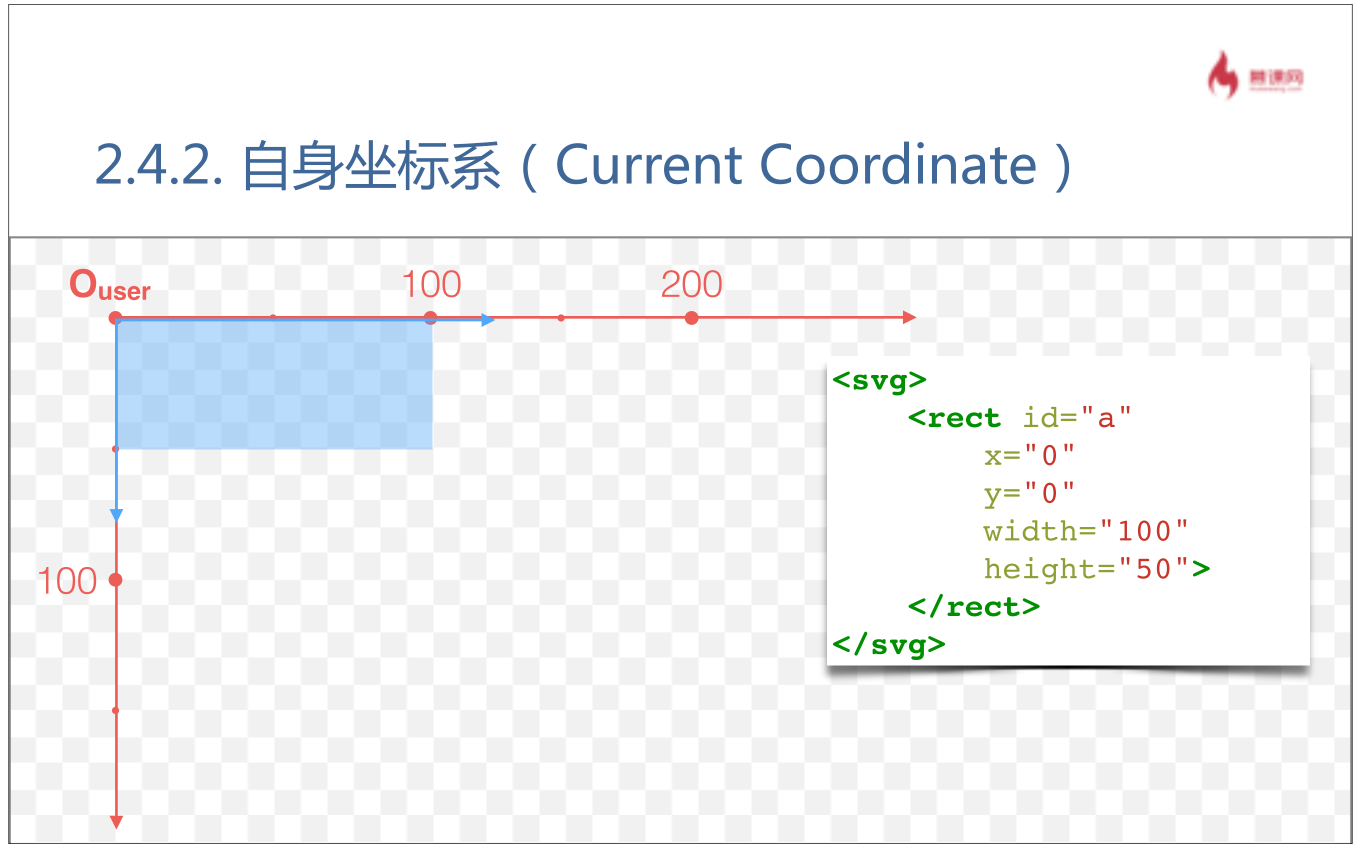
我们设置的 viewbox，也就是视野的大小，就说就是观察用户坐标系中的哪个区域。比如说，现在设置 viewbox 为 (0,0,200,150)

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

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

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

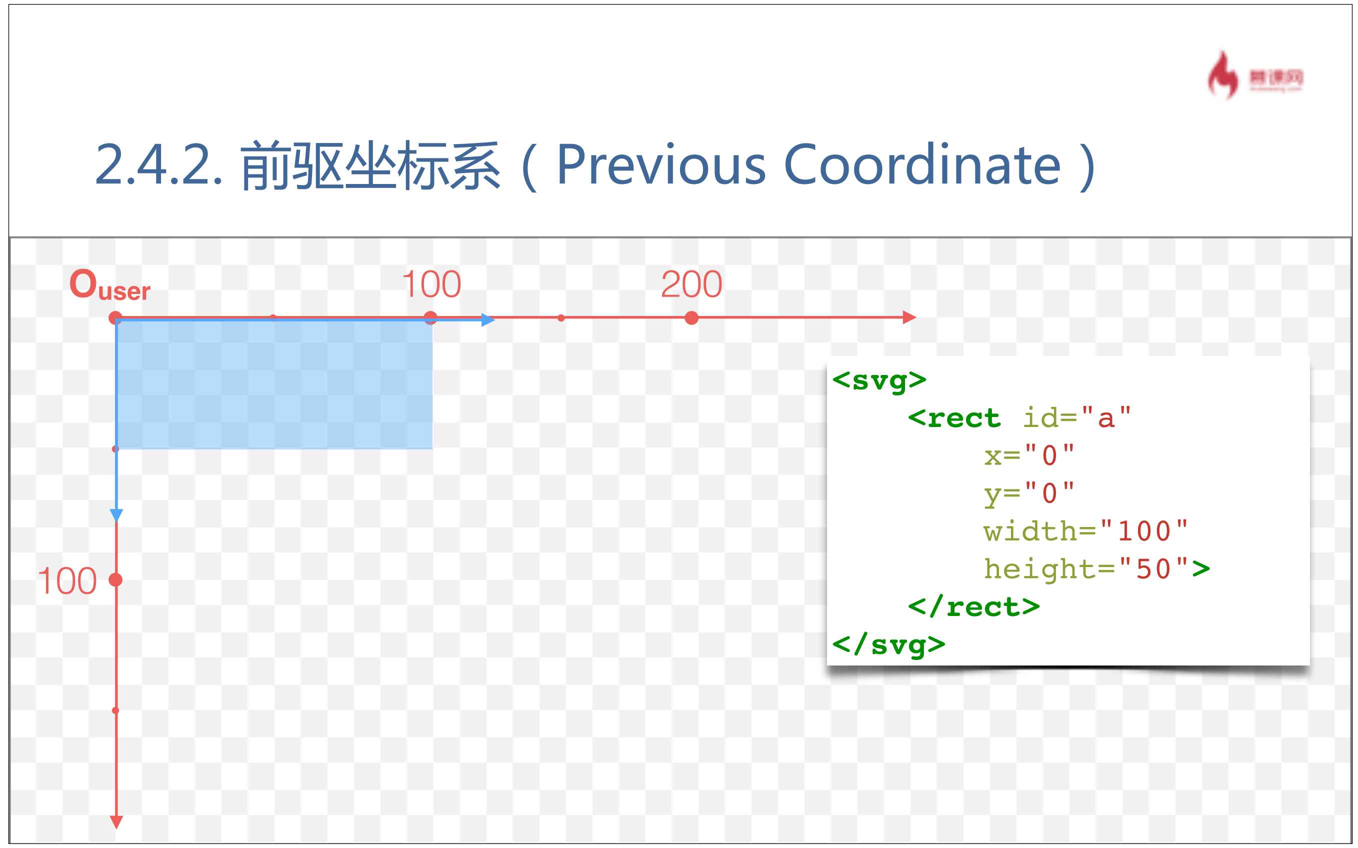
自身坐标系本身不太好理解，我们来结合前驱坐标系一同理解。



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

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

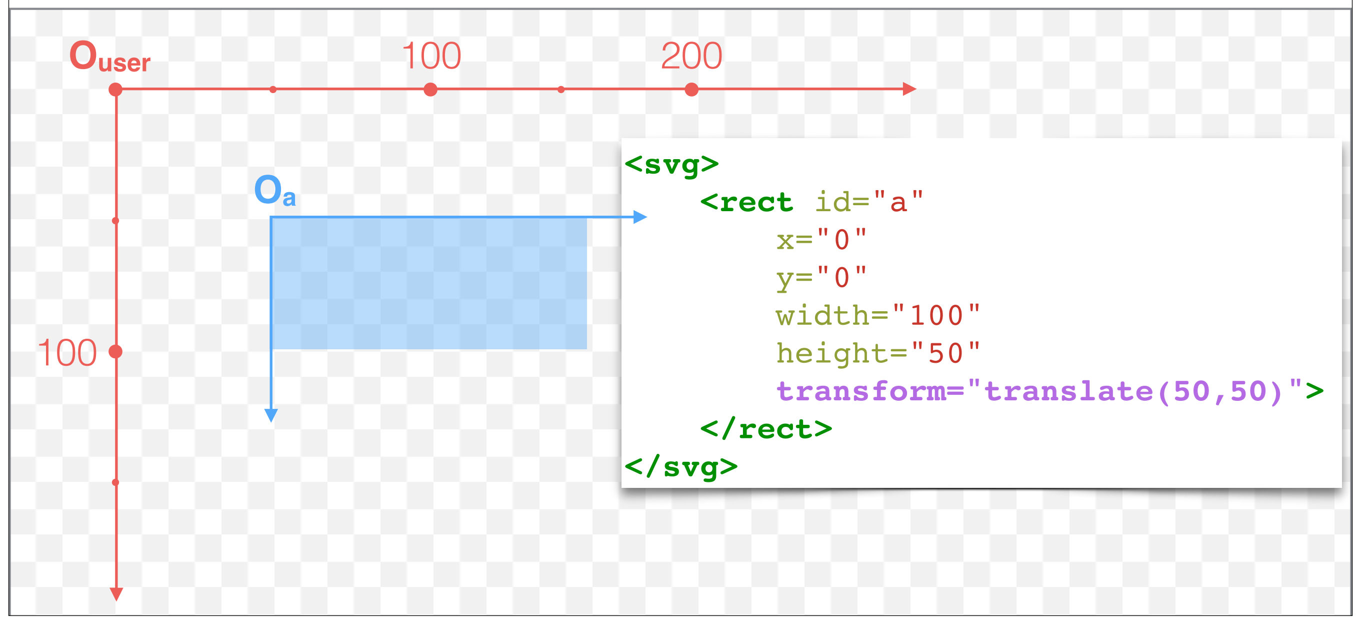
自身坐标系本身不太好理解，我们来结合前驱坐标系一同理解。



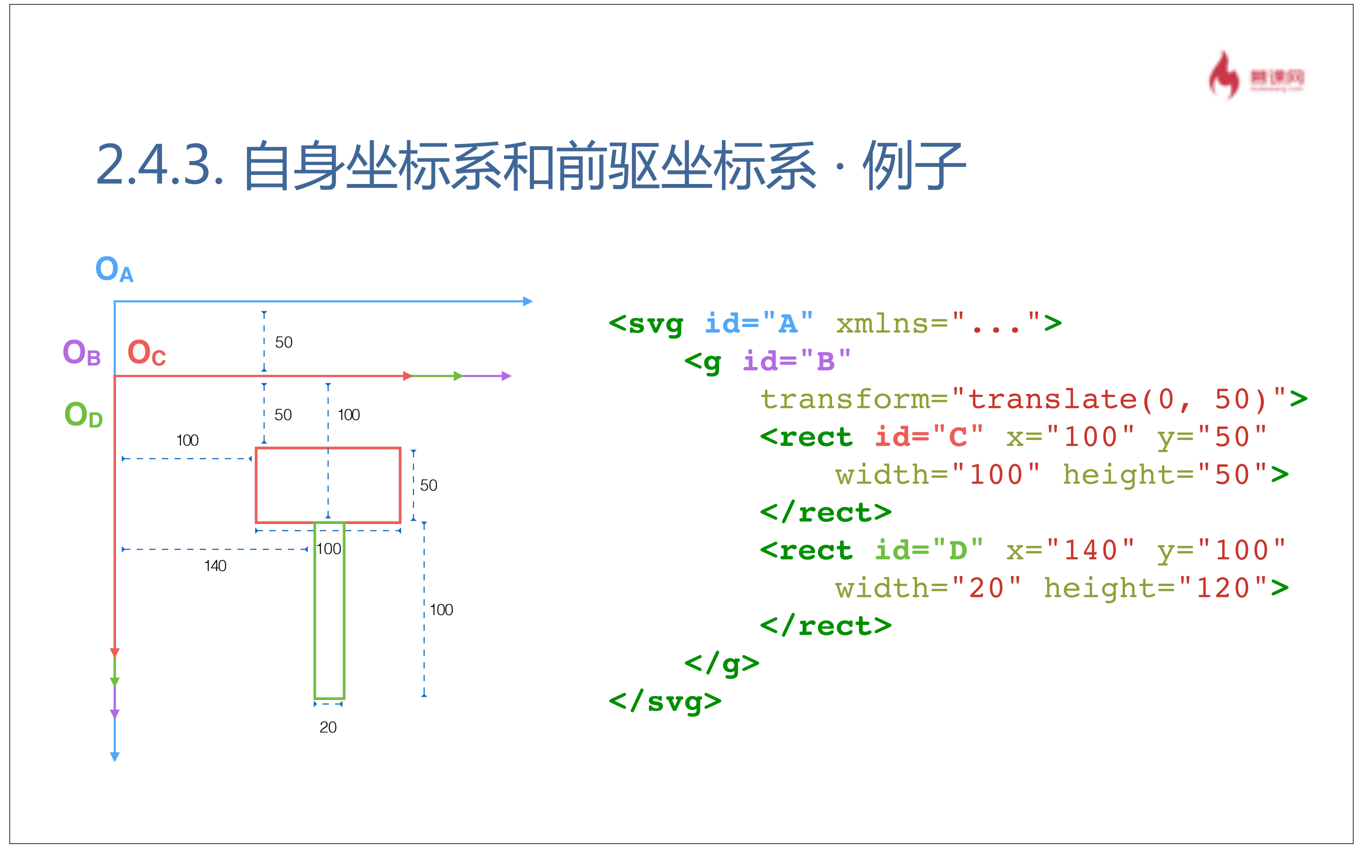
前驱坐标系，就是父容器的坐标系。现在矩形的父容器是 SVG 标签，那么它的前驱坐标系也就是世界坐标系。

坐标变换，就是前驱坐标系到自身坐标系的一个变换。现在给矩形加一个坐标变换。

2.4.3. 前驱坐标系（Previous Coordinate）



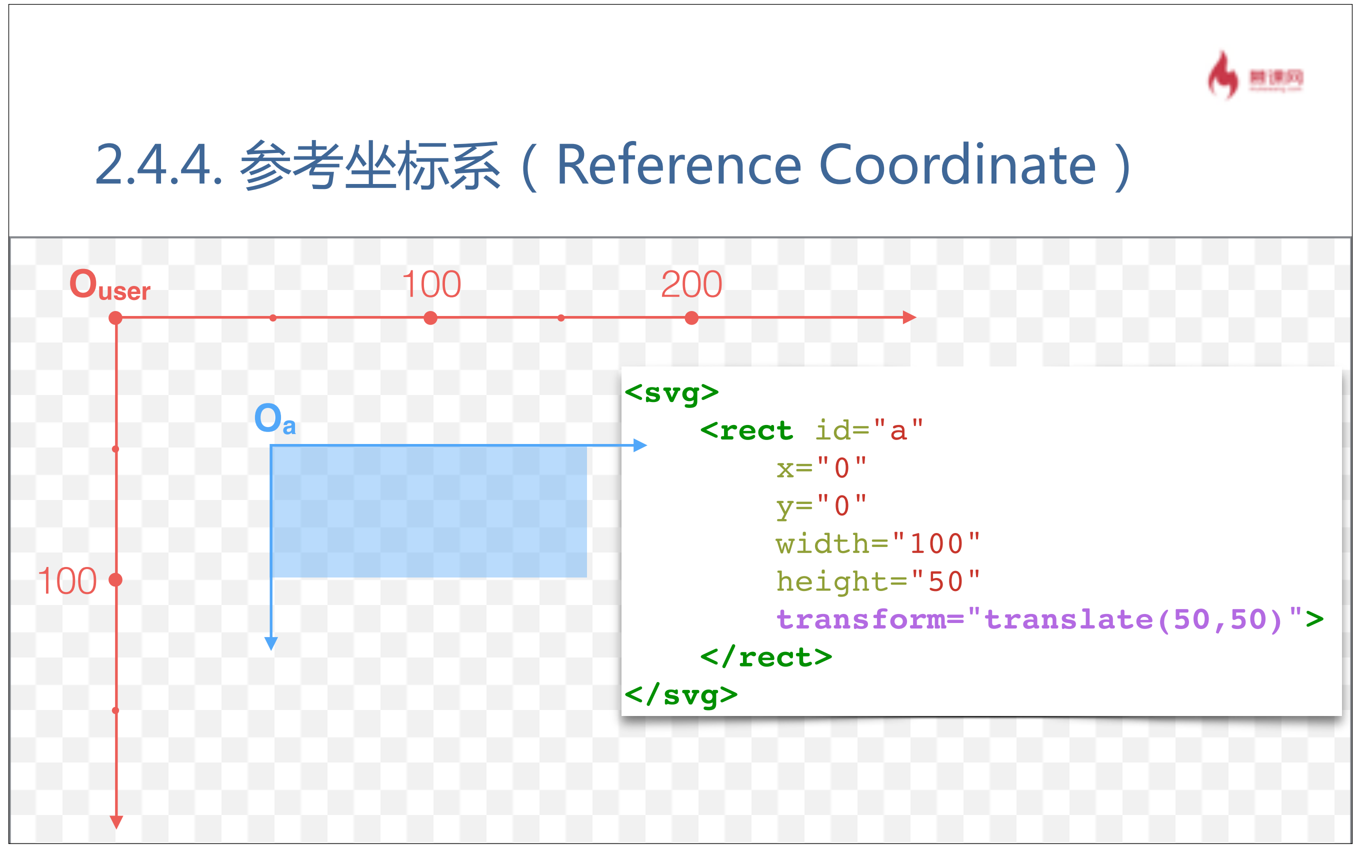
- 前驱坐标系，就是父容器的坐标系。现在矩形的父容器是 SVG 标签，那么它的前驱坐标系也就是世界坐标系。
- 坐标变换，就是前驱坐标系到自身坐标系的一个线性变换。现在给矩形加一个坐标变换。
- 大家观察一下，就会发现，矩形的坐标和宽高其实是没有改变的，x 和 y 依然是 0，而宽高依然是100x50。x、y和宽高是基于自身坐标系来定义的，定义了是多少就是多少。
- 变的是什么？变的是矩形的自身坐标系，它相对于他的前驱坐标系发生了一个变换。



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

- 世界坐标系是哪一个？OA
- B 的前驱坐标系是哪一个？OA
- C 和 D 的图形定义（坐标和宽高）是基于哪个坐标系的？OC 和 OD 两个自身坐标系
- C 和 D 的前驱坐标系是哪一个？OB
- OB 是哪个元素的自身坐标系？分组 B
- 分组 B 上设置的 transform 属性是什么意思？表示 B 的自身坐标系 OB 是从其前驱坐标系 OA 经过 transform 变换而来的。
- OB、OC和OD为什么重合？因为OB是OC和OD的前驱坐标系，而C和D上都没有定义transform属性，所以C和D的自身坐标系和前驱坐标系重合了。

好，整理了锤子的各种问题之后，再来看看最后一个坐标系。



参考坐标系，其实是任意的一个坐标系。使我们选区的用于观察某个图形情况的坐标系。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？我举一个例子大家就知道了。比如说我做图形编辑器，需要做一个对齐的功能。那么所谓对齐，一定是指在某个坐标系中对齐，因为图形的自身坐标系一般都是经过变换的。这个时候，就需要选取一个公共的参考坐标系，对图形进行观察，获得它们的坐标点，然后索引，对齐。

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。

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

希望这四个坐标系的概念同学们好好斟酌理解。

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

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

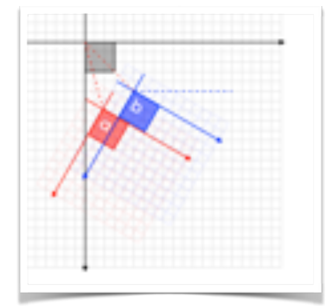
而参考坐标系，则是我在对某个图形进行观察、测量的时候，使用的一个坐标系。

Lesson 2 - SVG 中的坐标系统和坐标变换

好，刚刚说了前驱坐标系经过变换形成自身坐标系，现在就详细地介绍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面的内容，先来看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」是采用一定的数学方法将一个坐标系的坐标变换为另一个坐标系的坐标的过程。
- SVG 中，「坐标变换」是对一个坐标系到另一个坐标系的变换的描述

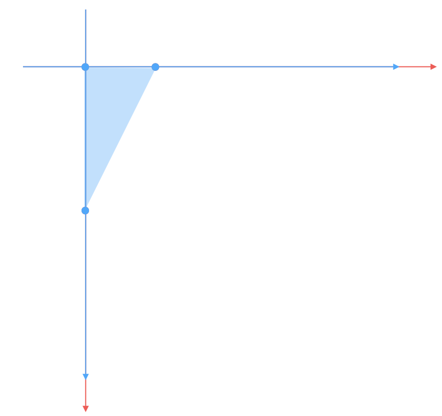
2.5.2. 线性变换

- 线性变换方程

  X’ = aX + cY + e
  Y’ = bX + dY + f

- 变换矩阵，记为 M

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$



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

线性变换的变换方程就是一个线性方程，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）

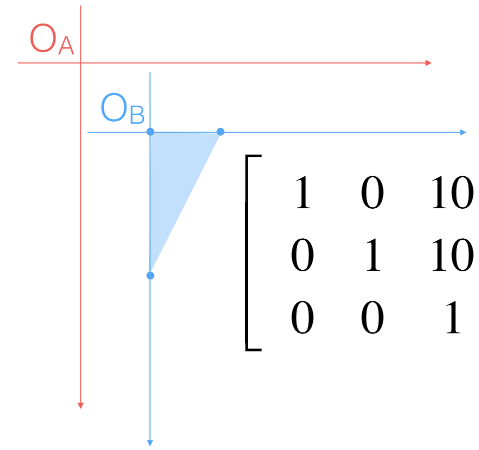
2.5.2. 线性变换

- 线性变换方程

X’ = aX + cY + e
Y’ = bX + dY + f

- 变换矩阵，记为 M

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$



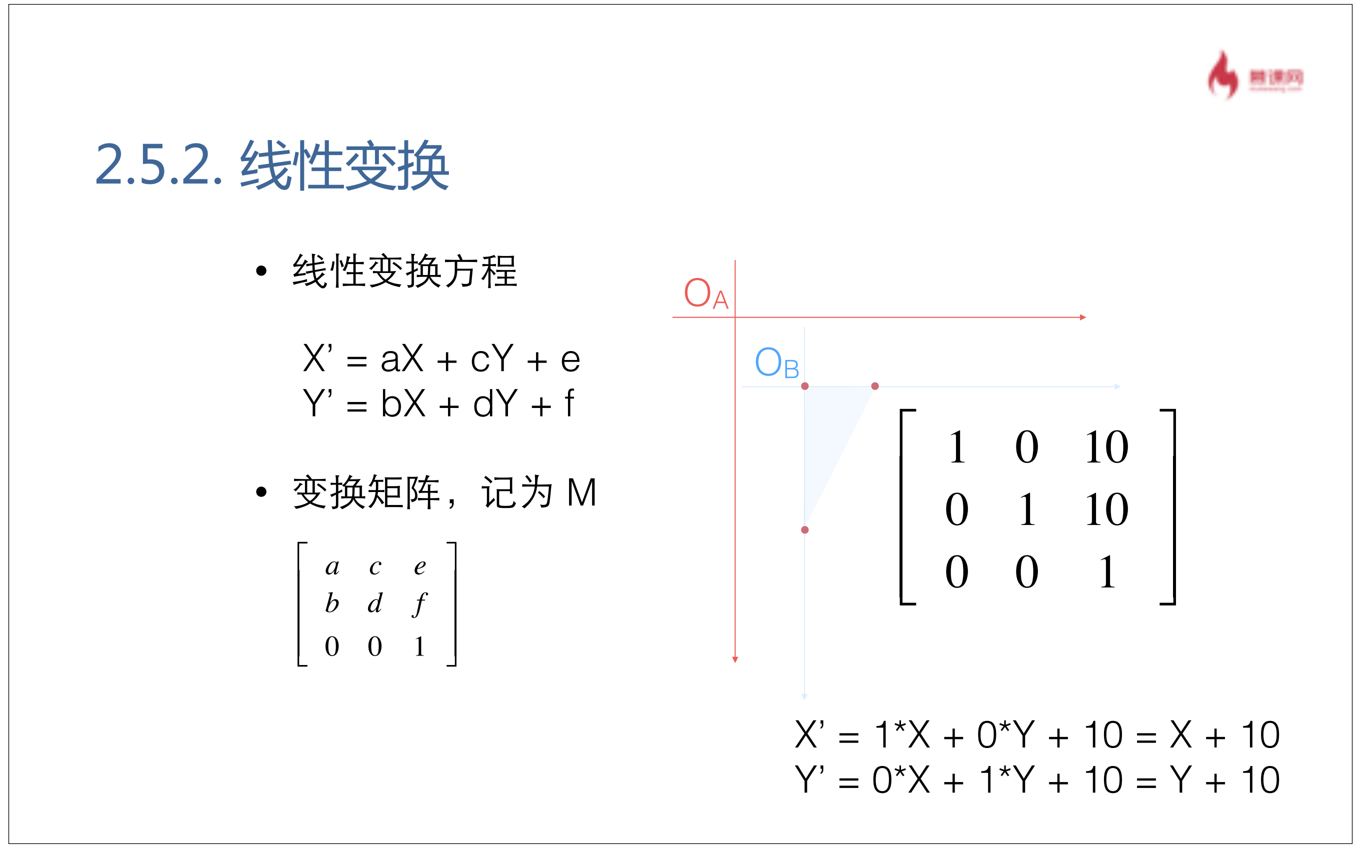
$$\begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix}$$

X’ = 1*X + 0*Y + 10 = X + 10
Y’ = 0*X + 1*Y + 10 = Y + 10

线性变换方程的意思是，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 OB 上每个点都是 OA 上每个点横纵坐标都加10而来。

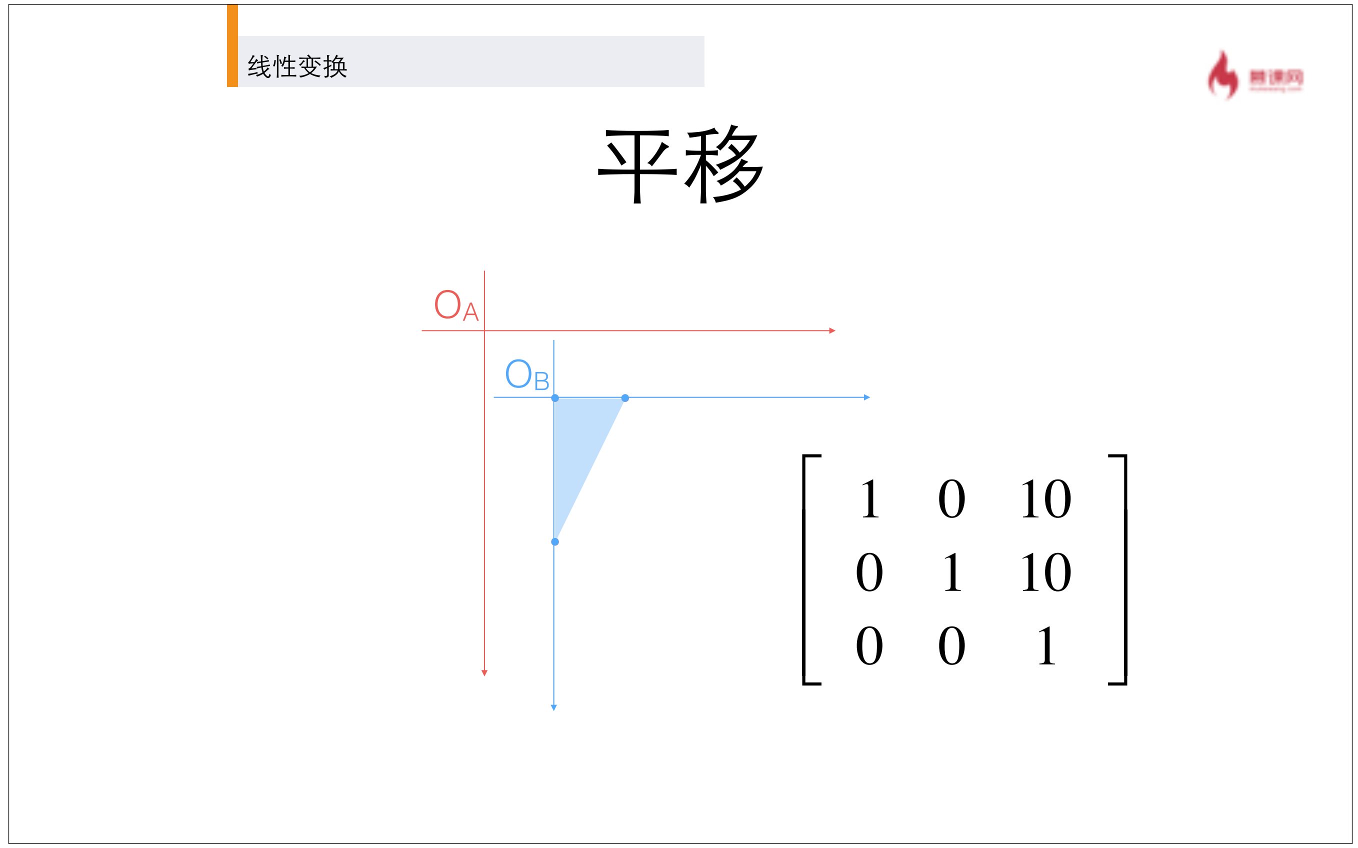
下面看几个常用的线性变换的例子。



线性变换方程的意思是，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 OB 上每个点都是 OA 上每个点横纵坐标都加10而来。

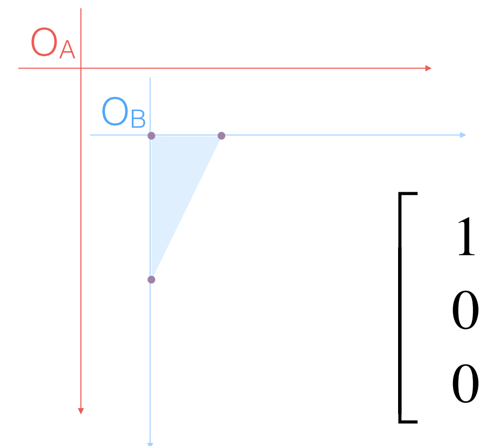
下面看几个常用的线性变换的例子。



平移比较简单，就是刚才看到的例子。

线性变换

平移



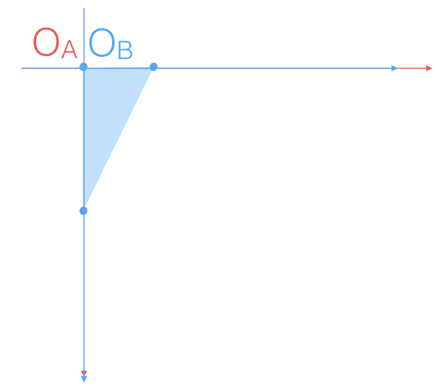
$$\begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix}$$

平移比较简单，就是刚才看到的例子。

线性变换

旋转

- 使用极坐标求变换矩阵



旋转可以通过极坐标的形式求出变换矩阵。

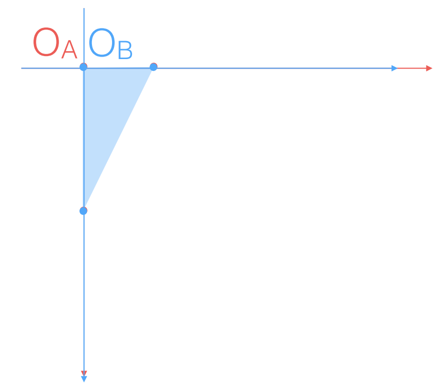
线性变换

旋转

- 使用极坐标求变换矩阵

  极坐标方程：

$$\begin{cases} X = r \cdot \cos(\alpha) \\ Y = r \bullet \sin(\alpha) \end{cases}$$



旋转可以通过极坐标的形式求出变换矩阵。

旋转

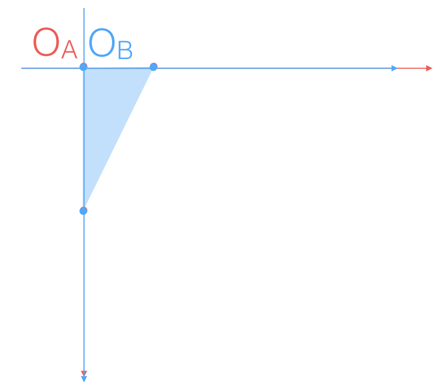
- 使用极坐标求变换矩阵

极坐标方程：

$$\begin{cases} X = r \cdot \cos(\alpha) \\ Y = r \bullet \sin(\alpha) \end{cases}$$

旋转 θ 度后：

$$\begin{cases} X' = r \cdot \cos(\alpha + \theta) \\ Y' = r \bullet \sin(\alpha + \theta) \end{cases}$$



旋转可以通过极坐标的形式求出变换矩阵。

旋转

- 使用极坐标求变换矩阵

极坐标方程:

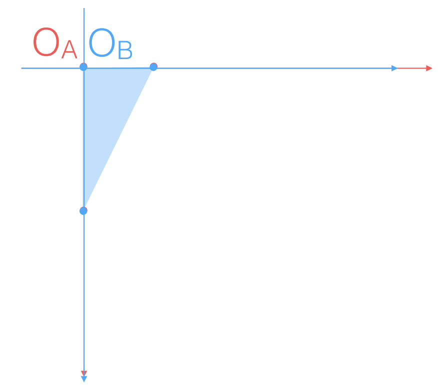
$$\begin{cases} X = r \cdot \cos(\alpha) \\ Y = r \bullet \sin(\alpha) \end{cases}$$

旋转 θ 度后:

$$\begin{cases} X' = r \cdot \cos(\alpha + \theta) \\ Y' = r \bullet \sin(\alpha + \theta) \end{cases}$$

展开:

$$\begin{cases} X' = r \cdot cos(\alpha)cos(\theta) - r \cdot sin(\alpha)sin(\theta) = cos(\theta)X - sin(\theta)Y \ + \ 0 \\ Y' = r \cdot cos(\alpha)sin(\theta) + r \cdot sin(\alpha)cos(\theta) = sin(\theta)X + cos(\theta)Y \ + \ 0 \end{cases}$$



旋转可以通过极坐标的形式求出变换矩阵。

旋转

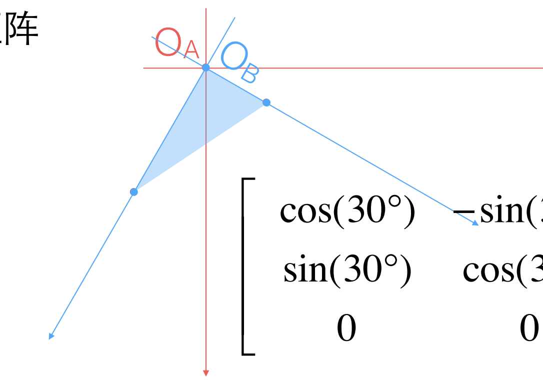
- 使用极坐标求变换矩阵

极坐标方程:

$$\begin{cases} X = r\cdot\cos(\alpha) \\ Y = r\bullet\sin(\alpha) \end{cases}$$

旋转 θ 度后:

$$\begin{cases} X' = r\cdot\cos(\alpha+\theta) \\ Y' = r\bullet\sin(\alpha+\theta) \end{cases}$$



$$\begin{bmatrix} \cos(30°) & -\sin(30°) & 0 \\ \sin(30°) & \cos(30°) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

展开:

$$\begin{cases} X' = r\cdot cos(\alpha)cos(\theta) - r\cdot sin(\alpha)sin(\theta) = cos(\theta)X - sin(\theta)Y \ + \ 0 \\ Y' = r\cdot cos(\alpha)sin(\theta) + r\cdot sin(\alpha)cos(\theta) = sin(\theta)X + cos(\theta)Y \ + \ 0 \end{cases}$$

旋转

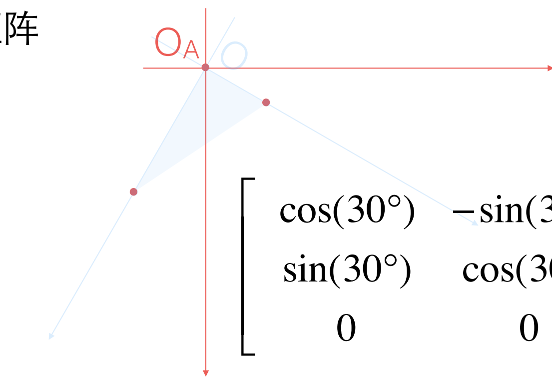
- 使用极坐标求变换矩阵

极坐标方程：

$$\begin{cases} X = r \cdot \cos(\alpha) \\ Y = r \bullet \sin(\alpha) \end{cases}$$

旋转 θ 度后：

$$\begin{cases} X' = r \cdot \cos(\alpha + \theta) \\ Y' = r \bullet \sin(\alpha + \theta) \end{cases}$$



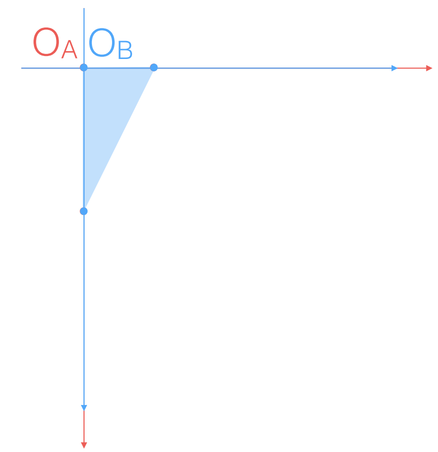
$$\begin{bmatrix} \cos(30°) & -\sin(30°) & 0 \\ \sin(30°) & \cos(30°) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

展开：

$$\begin{cases} X' = r \cdot cos(\alpha) cos(\theta) - r \cdot sin(\alpha) sin(\theta) = cos(\theta) X - sin(\theta) Y \;+\; 0 \\ Y' = r \cdot cos(\alpha) sin(\theta) + r \cdot sin(\alpha) cos(\theta) = sin(\theta) X + cos(\theta) Y \;+\; 0 \end{cases}$$

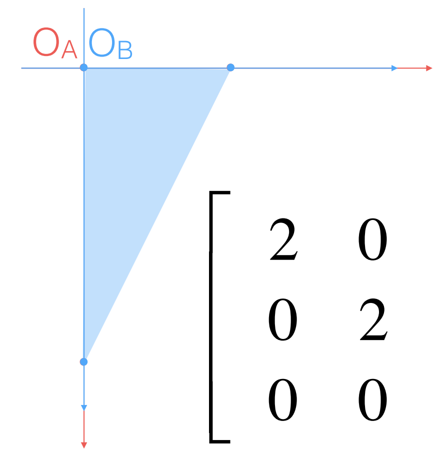
缩放

- a 和 c 直观控制缩放



缩放

- a 和 c 直观控制缩放



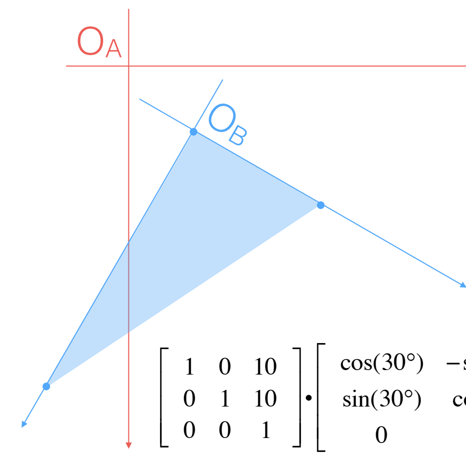
$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.5.3. 线性变换列表

- 表示一系列的变换，结果为变换的矩阵的乘积

$$M = M_n \cdot M_{n-1} \cdot \ldots \cdot M_2 \cdot M_1 \cdot M_0$$

- 后面的变换乘在前面



$$\begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(30°) & -\sin(30°) & 0 \\ \sin(30°) & \cos(30°) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_{translate} \cdot M_{rotate}$$

单个线性变换矩阵，可以表示所有的线性变换。但是，一般我们去描述一个线性变换可能更愿意分开一步步来描述。比如说，先旋转30度，再平移（10，10），那么已然可以有一个变换矩阵可以表示这个线性变换列表的结果，那就就是每一步变换矩阵的乘积。

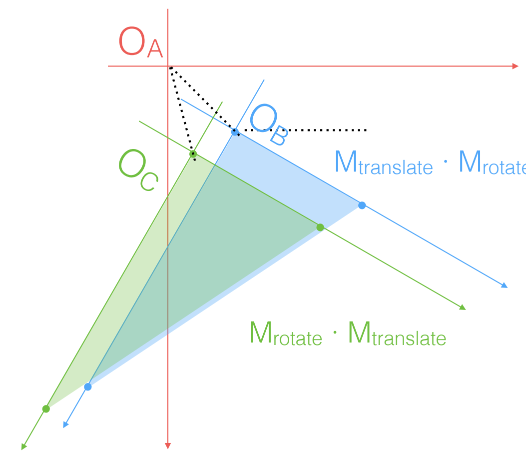
需要注意的是，最后面的变换，需要乘在前面。这是线性代数当中的一个结论，有兴趣的同学可以自行了解一下。

2.5.3. 线性变换列表

- 表示一系列的变换，结果为变换的矩阵的乘积

$$M = M_n \cdot M_{n-1} \cdot \ldots \cdot M_2 \cdot M_1 \cdot M_0$$

- 后面的变换乘在前面



2.5.4. transform属性

- 前驱坐标系：父容器的坐标系

- transform属性：定义前驱坐标系到自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。

刚刚的例子里头，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。这种行为，可以称为坐标观察。

坐标观察还可以解决很多问题，比如，交互的时候，我希望知道我点击的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法来配合坐标观察。