前言

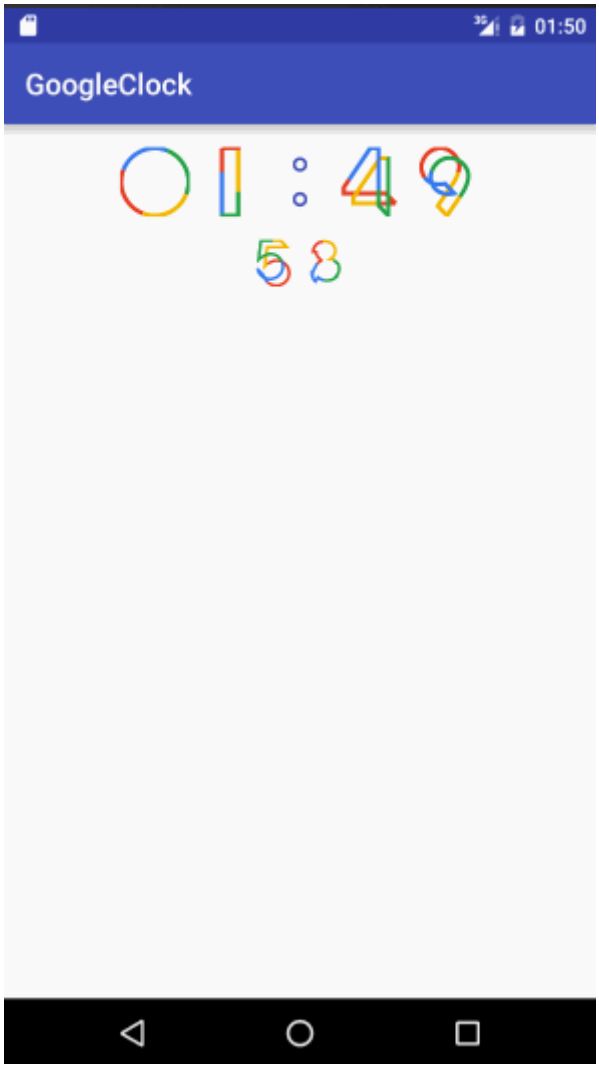
不管是开发 Android 已久的老司机，还是刚刚上车的新司机，都肯定会对一件事情深恶痛绝：图片适配（尤其是在美工不给力的条件下）！为什么 Android 手机要有这么多不同的分辨率？ 为什么我的图片在这台手机上显示地好好的完全符合设计图的要求结果换到另一台手机上就变形了？Oh my god !

以前为了解决图片在不同的分辨率的屏幕上显示不一致的问题，通常我们会采取两种方式：一是根据不同的分辨率建立不同的资源包，然后每个要用到的图片资源都需要做成多种尺寸的以适配不同的分辨率；二是干脆直接放一个分辨率比较大的图片，然后钦定 ImageView 的大小，强行把图片塞进去——由于图片的分辨率比较大，所以在大部分机型上也还是能看的。但是这两种方式都有一些问题，首先不可避免的，都极容易导致 apk 包的体积变大，这个问题在 app 体量比较小的时候可能不会对产品造成什么干扰，但是当体量逐渐变大，这将是个极为让人头疼的问题。另外，用第一种方法完成过图片适配的兄弟都懂，把不同尺寸的图片更名为符合规范的相同名字并放到它们合适的文件夹里面去简直是一种折磨。。。

这样的痛苦体验什么时候才是个头?

拥抱SVG吧。

PS：我用 SVG 动画写了一个类似于 Google 2016 I/O 大会上的那个时钟的东西，应该是一个比较好的学习 SVG 在 Android 上的使用的资源，大家可以去关注一下点点 star 提提 issue 哈，地址是：GoogleClock，贴个截图：



一、什么是 SVG ?

SVG是指可伸缩矢量图形 (Scalable Vector Graphics)，它不同于传统的位图，不是通过存储图像中每一点的像素值来保存与使用图形，而是通过 XML 文件来定义一个图形，通过一些特定的语法和规则来绘制出我们所需的图像——同样是使用一张图片，SVG 的方式是事先定义好怎么去画这个图，然后等要用的时候再把它去画出来，而使用传统的位图的话就是已经有了画出来的图，然后要用的时候直接把画好的图拿出来用。这样一来的话我们就很容易可以分析出它们两种方式之间的优劣之处：

- SVG 是在要用图的时候再把图画出来，所以理所当然的在图片显示的时候会花费更多的时间消耗更多的资源。
- 同样由于上一个原因， SVG 并不太适合层次过于复杂细节过于繁多的图片。
- 位图是事先已经画好的图片，所以适应性必然没有 SVG 好，同一张图片在不同分辨率下显示会有差异。
- SVG 的文件里存储了绘制图片的相关信息，所以我们能够对图片的线条有一个非常清晰的感知，这在做动画的时候特别有用。
- SVG 没有存储任何图像的像素信息，所以 SVG 的文件体积远小于传统的位图文件。
- SVG 的文件画出来的图像是矢量图，所以不会存在失真的问题，理论上支持任何级别的缩放。

从上面的分析大家可以发现，在 SVG 的众多优点下， 它的缺点几乎可以忽略不计了（当然，前提是它所耗费的资源不会对用户体验造成较大的影响）——这也正是本文标题 “拥抱SVG” 的由来。.

二、SVG in Android

既然 SVG 这么好，那么为什么当前并没有多少 Android 应用是使用了 SVG 图片的呢？因为市场。Android对于 SVG 的支持是从 Android L 开始的，它的 SDK 里面加入了 VectorDrawable，AnimatedVectorDrawable 等类帮助我们构建 SVG 图形以及动画，并且你可以在 xml 文件里面直接使用 <vector> 标签绘制 SVG 图像以及 <animated-vector> 标签为 SVG 图像分配动画。

但是请注意，目前并没有官方的 support 包来帮助我们对运行着 Android L 之前的系统的设备做兼容，而在一些开源社区里一些人做的兼容包也都还没有那种比较完美的解决方案，总是会有一些问题——这意味着，如果不想将就的话，就只能等到市场上基本都是 Android L 或以上的设备的时候，才有可能在生产中大规模的全面的用 SVG 替换位图了。而目前，2016年秋，据友盟统计，Android L 及以上的设备的市场占有率仅有 43.27%，一半都不到——但是很显然，距离 SVG 在 Android 上发力的时候没有多远了，毕竟目前在售卖的 Android 手机已经基本上都是搭载的 Android L 及以上的系统了，只待老设备被淘汰。

三、SVG 的使用

3.1，获得一个 SVG 文件

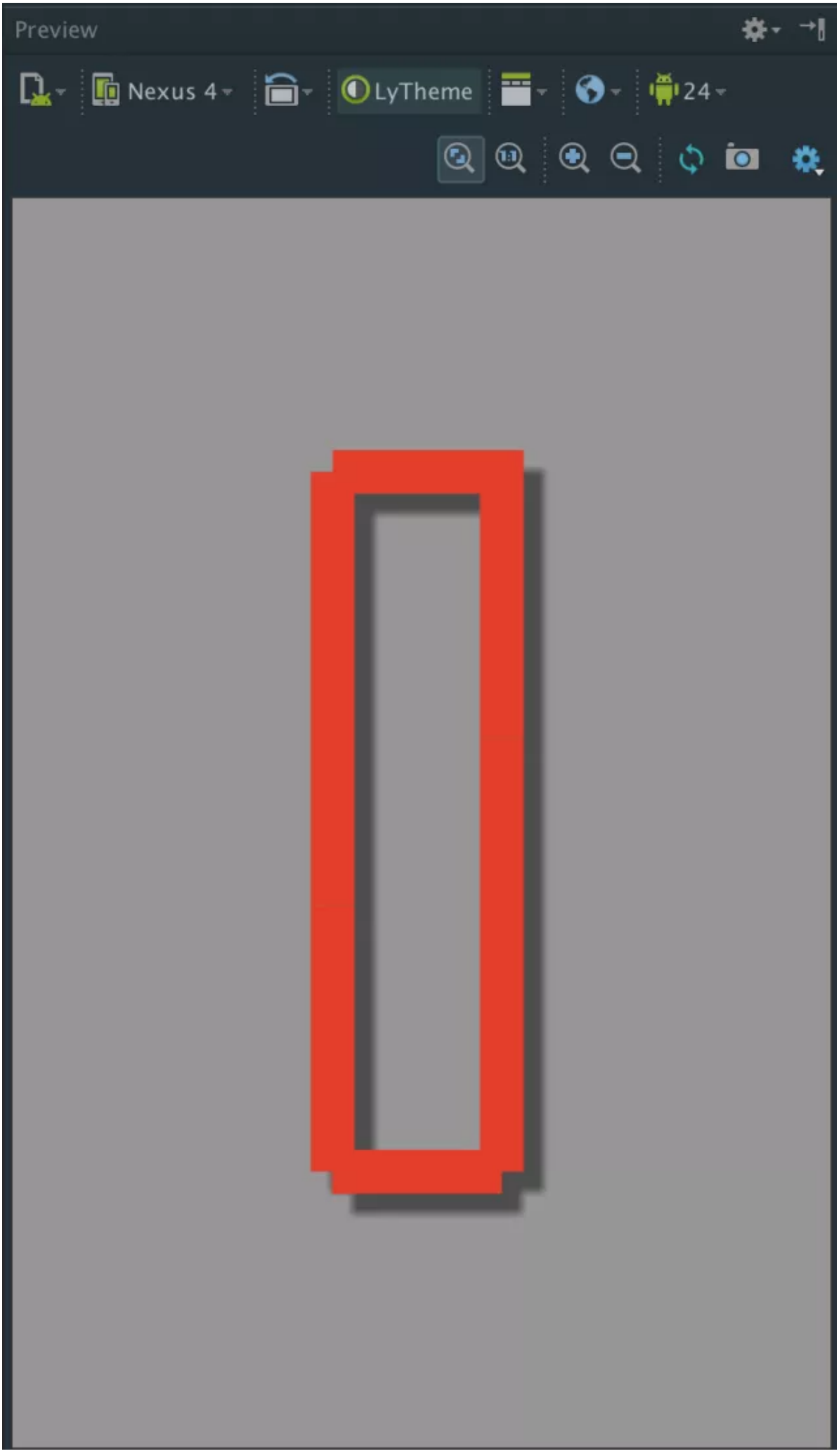
要使用 SVG，那么首先我们肯定得有一个 SVG 文件。我们一般都有两种方式来获得一个 SVG 文件：自己写一个 SVG 文件，或者通过 AI 或一些网站作图之后导出它的 SVG 文件。

3.1.1，自己写一个 SVG 文件（Android 中）

前面说过，SVG 文件里面存储的是如何去绘制目标图片的相关信息，所以理论上我们是可以从 0 开始写一个我们自己的 SVG 文件的——只要知道它绘制文件的规则，一切皆可绘制。我们先来看一下一个简单的 SVG 文件：

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="132dp"
    android:height="132dp"
    android:viewportHeight="132.0"
    android:viewportWidth="132.0">
  <path
    android:pathData="M50,2 L80.813,2 L80.813,130 L50,130 L50,2 Z"
    android:strokeColor="#e33e2b"
    android:strokeWidth="8" />
</vector>
```

这个文件绘制出来的图形是这样的(没错，在编写 SVG 的 XML 文件的时候 Android Studio 是可以预览的，很强大)：

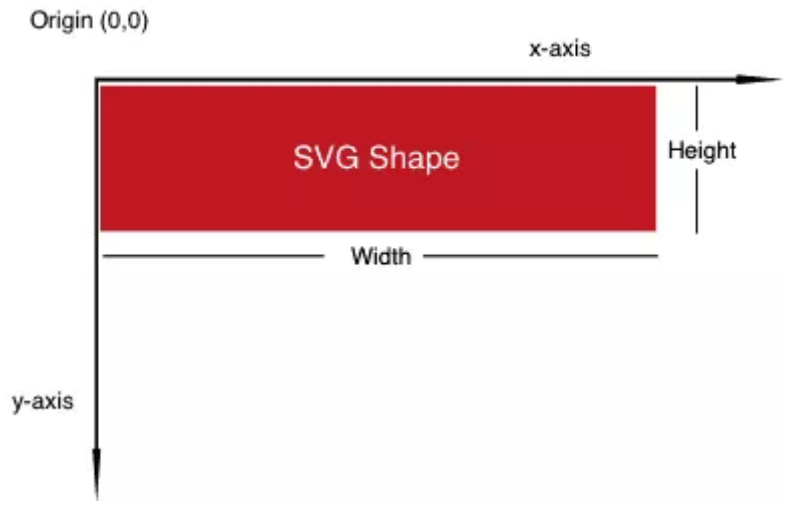

Preview
Nexus 4
LyTheme
24

可以看到，总的来说 SVG 文件在 Android 中的载体是一个 标签，而绘制图片的工作是在 这个子标签里面做的。我们先来看一下这两个标签里面最常使用的一些属性。

vector：

属性	参数类型	默认值	描述
width	dimen	必填属性	图形的实际宽度，可在使用时根据需要再次定义
height	dimen	必填属性	图形的实际长度，可在使用时根据需要再次定义
viewportHeight	float	必填属性	定义画布的尺寸
viewportWidth	float	必填属性	定义画布的尺寸

平时在 标签里面常用的属性基本上也就上面这四个了，但是关于这四个属性我想再多讲一些东西，因为我发现目前网上有很多文章对这个的描述都有些问题，容易对刚接触这个的同学产生误导。首先如果对应到一张具体的图片，XXX.png来讲的话，上表中的 width 和 height 就相当于 XXX.png 的实际的宽度和长度，但是不同的是我们可以在使用它的时候再任意的定义它的新的宽度和长度，甚至比例和原先不一样都没关系，图形并不会因此而失真。而 viewportHeight 和 viewportWidth 这两个属性则是用来确定 XXX.png 上的坐标系的单位长度的。比方说，像上面的代码的话，viewportHeight 为 132，heigth 为 132dp ，意思就是图片的长度被分成了 132 份，每一份的长度为 1dp ——这样一来，坐标系的单位长度就有了，也就有了根据坐标来绘制图形的基础。另外，在 Android 上 SVG 的画布上的坐标系是这样的：



接下来看一下 标签的常用属性：

属性	参数类型	默认值	描述
pathData	String	无	画图的核心所在，有一定的语法，根据它来绘制目标图形
strokeColor	color	透明	画笔的颜色
name	String	无	这一条path的name，在其他地方可以根据name来找到这一条path
strokeWidth	float	0.0	画笔的宽度
fillColor	color	透明	用颜色填充绘制过的区域，如果图形是闭合的就直接填充，如果图形不是闭合的那么就将图形的起点和终点相连使其闭合然后填充

基本上知道了上面这些属性就可以通过赋给 pathData 一些值来绘制出一些比较常规的图形了。接下来我讲一下在 pathData 里面绘制图形的一些基本语法：

- M = moveto(M X,Y)：将画笔移动到指定的坐标位置，但未发生绘制
- L = lineto(L X,Y)：画直线到指定的坐标位置
- H = horizontal lineto(H X)：画水平线到指定的X轴坐标
- V = vertical lineto(V Y)：画垂直线到指定的Y轴坐标
- C = curveto(C X1,Y1,X2,Y2,ENDX,ENDY)：三次贝塞曲线
- S = smooth curveto(S X2,Y2,ENDX,ENDY)：三次贝塞曲线
- Q = quadratic Belzier curveto(Q X,Y,ENDX,ENDY)：二次贝塞曲线
- T = smooth quadratic Belzier curveto(T ENDX,ENDY)：映射前面路径后的终点
- A = elliptical Arc(A RX,RY,XROTATION,FLAG1,FLAG2,X,Y)：弧线
- Z = closepath()：关闭路径

另，上述所有指令大小写均可。大写表示绝对定位，参照全局坐标系；小写表示相对定位，参照父容器坐标系。指令和数据间的空格可以省略。同一指令出现多次可以只用一个。

现在我们可以稍微的解读一下上面的那个例子里面的 pathData 是什么意思了。上面的数据是：

M50,2 L80.813,2 L80.813,130 L50,130 L50,2 Z//大家可以看到，L 指令的后面只有一个坐标，这样的意思就是画的时候以上一笔的终点为起点

根据我们的语法，他的意思就是：

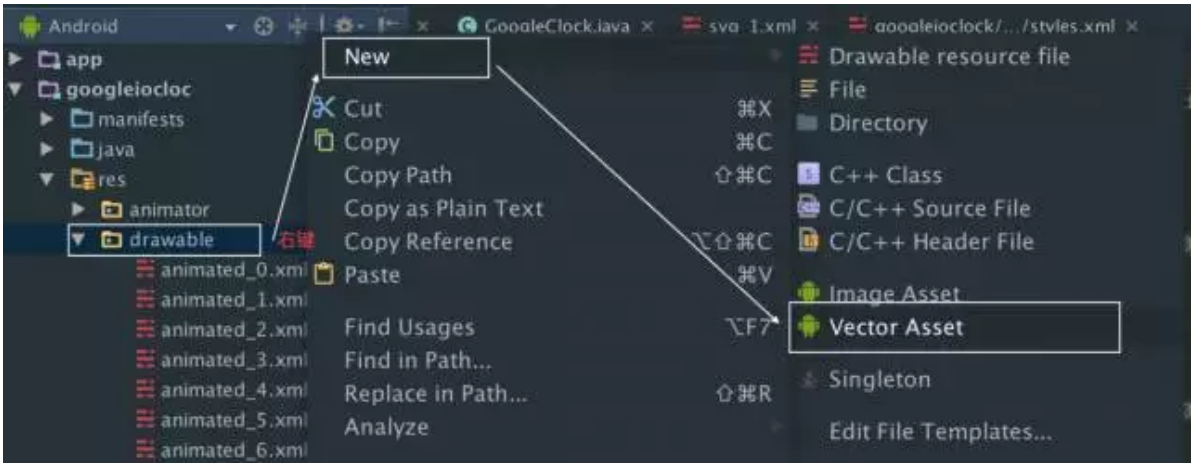
- M 50,2 ： 将画笔移到(50,2) 坐标处。
- L 80.813,2 ： 从 (50,2) 画直线到 (80.813,2)。
- L 80.813,130 ： 从 (80.813,2) 画直线到 (80.813,130)。
- L 50,130 ： 从 (80.813,130) 画直线到 (50,130)。
- L 50,2 ： 从 (50,130) 画直线到 (50,2)。
- Z ： 结束这一笔。

我们不难发现，绘制的过程其实很简单，最后的结果是形成了一个闭合的矩形——和我们的成型的图像是一致的。

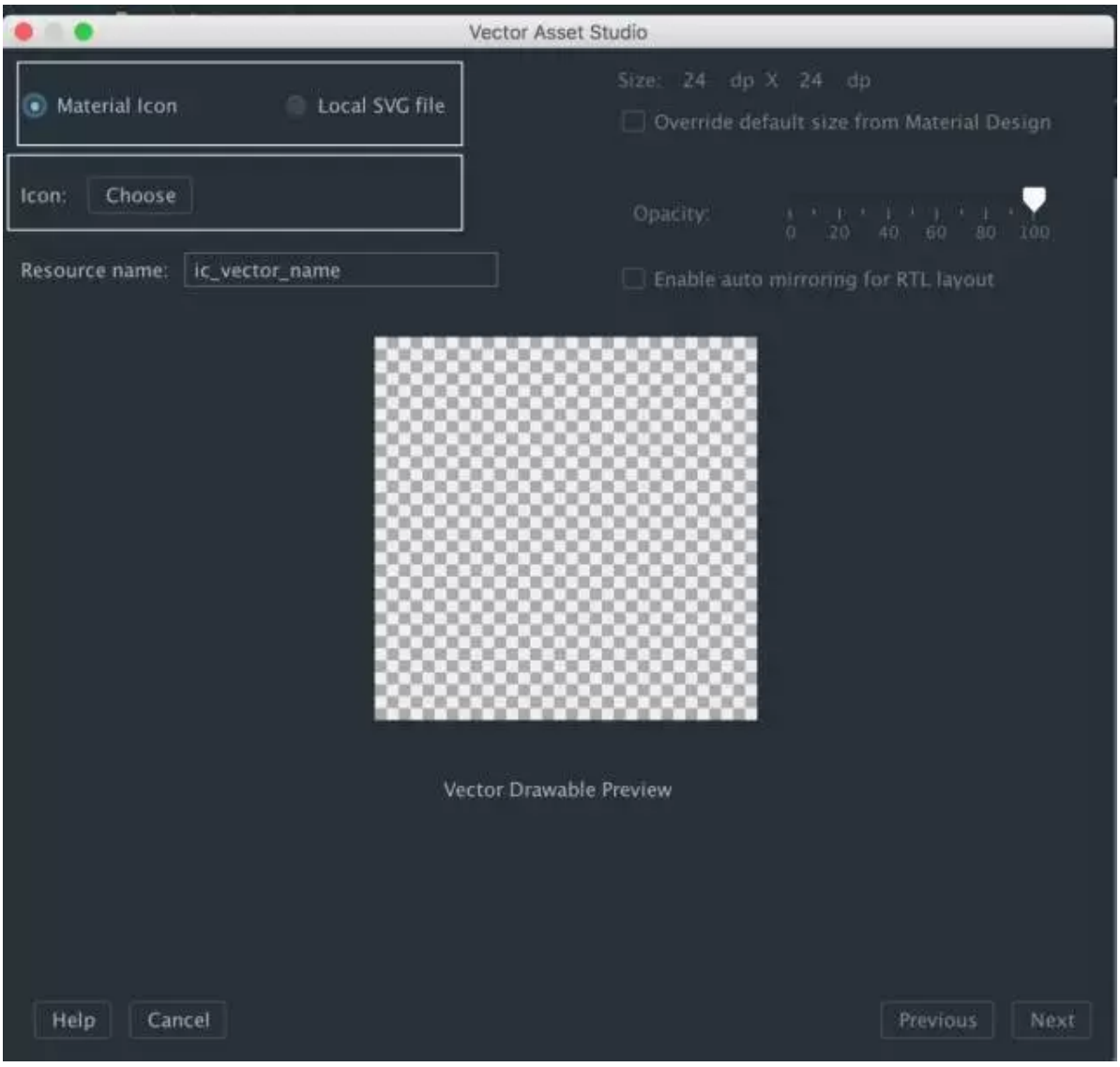
在这里我就不介绍更多的指令的使用了，具体的大家可以参照这篇文章：Android vector标签 PathData 画图超详解。不过其实我认为，根本没有必要去把这些指令完全的记忆下来，因为在实际的工作中我们几乎是不会去手写 SVG 文件的，我们需要掌握的指令其实只有两个： M 和 Z 。因为它们象征着画笔的一笔的开始和结束，在涉及一些和路径轨迹相关的操作的时候我们可以合理的控制 M 和 Z 来很快速的达到我们的目的。

3.1.2，获取一个 SVG 文件（Android 中）

在希望加入 SVG 文件的地方右键一下，然后 new -> Vector Asset：



点一下 Vector Asset，会弹出另一个新的弹窗，它是长这样的：



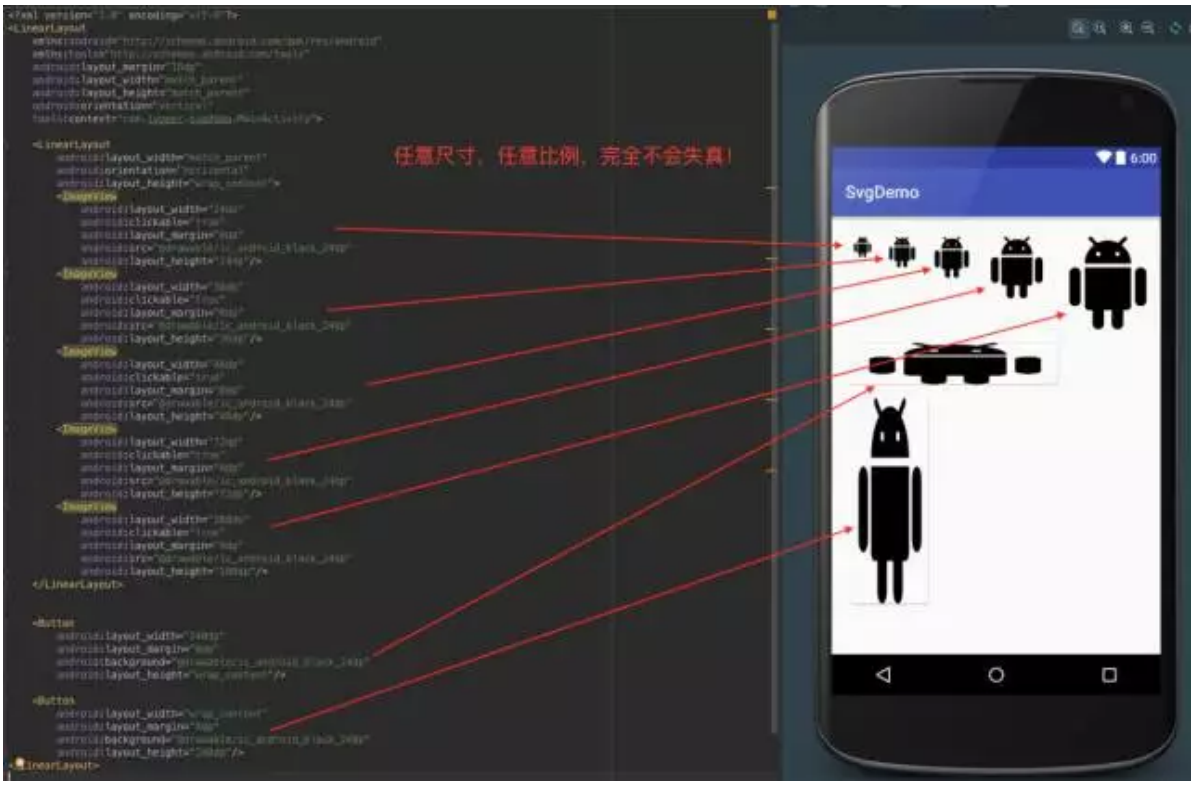
图中有两个白色框框起来的部位，上面那个是一个单选框，可以很清晰的知道选第一个是选取 Material Icon （没错，就是这么贴心，AS 自带所有 MD 图标的 SVG 文件），第二个是选取本地 SVG 文件，选好第一个框之后再在下面的白框里面选择具体的哪一个文件——这样一来，我们的目的就完成了，我们成功的在 AS 里面得到了一个 SVG 的 XML 文件。

在这里可能很多同学会有一个问题：刚才说了如何将本地的 SVG 文件导入到 AS 中，那么我们又怎么获得本地的 SVG 文件呢？讲道理，这个问题不应该是我们考虑的，这个东西应当是美工把制作好的图给我们，然后我们直接使用就可以了。但是有的时候我们自己做小东西并没有专业的美工怎么办呢？要么你可以自己去学学 AI 或者 GIMP 等软件的使用方法， 用它们来制作图形然后导成 SVG ，当然这样的话学习成本有点高——不过没关系，我们还有低配版的实现方式：Method Draw。这是一个在线制作矢量图的网站，可以很方便的将在上面制作的图形导出成 SVG 文件，学习成本相当低，而且能完成我们大部分的需求，总之我觉得还挺好用的。

3.2，开始使用吧！

如果前面的工作都完成了的话，我们应当是已经有了一个 SVG 的 XML 文件，接下来理所当然的是如何在我们需要的地方使用它了。那么怎么使用呢？

我想说的是，接下来你完全可以把它当成我们引入项目的一张图片来用。比如：



上图是直接在布局文件中直接引用 SVG 的 XML 文件，代码中的 ic_android_black_24dp 就是已经写好的 SVG 文件。可以看到，直接把它当做一张普通的图片来使用就可以了。当然，我们也可以在 Java 代码里面来使用 SVG 文件，像下面这样：

```
mImageView.setImageDrawable(getDrawable(R.drawable.ic_android_black_24dp));
```

到这里我们就已经可以在 Android 中使用 SVG 来作为图片资源了，这样一来不仅 Apk 包的体积得到了大大的减小，我们的图片也具有了任意拉伸而不失真的特性，而且我们也再也不用非常痛苦的去搞图片改名称分包了。

结语

总的来讲，我认为 SVG 是有在大部分应用场景下取代传统的位图成为一种更优的图片的解决方案的潜力的，至少在 Android 中是这样。但是由于目前搭载着 Android L 之前的机子还很多，所有很多的 Android 开发人员就将学习 SVG 相关的知识无限期的推迟了，但其实，已经是时候了。