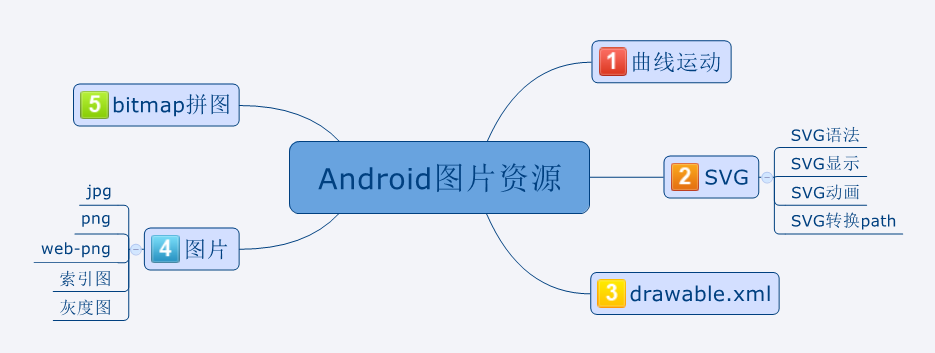
**Android图片资源优化**

1. **概述图**



1. **曲线运动**
2. 曲线运动是什么？

曲线运动其实就是增强版的属性动画（ObjectAnimator），从构造函数就可以看出，它只是新增了一个重载。所以它就是属性动画，但是提供了基于Path的复杂曲线运动的机制。

1. demo举例

**ImageView iv = findView(...);**

**Path path = new Path();**

**path.moveTo(xx, xx);**

**path.quadTo(xx, xx, xx, xx);**

**path.lineTo(xx, xx);**

**ObjectAnimator animator = ObjectAnimator.ofFloat(iv, View.X, View.Y, path);**

**animator.setDuration(3000);**

**//animator.setInterpolator(new FastOutLinearInInterpolator());**

**//animator.setInterpolator(new FastOutSlowInInterpolator());**

**//animator.setInterpolator(new LinearOutSlowInInterpolator());**

**animator.start();**

1. 效果

[演示视频](曲线运动.mp4)

1. 设置插值器

系统提供了三种基本的曲线，XML资源：

@interpolator/fast\_out\_linear\_in.xml

@interpolator/fast\_out\_slow\_in.xml

@interpolator/linear\_out\_slow\_in.xml

您可以用PathInterpolator对象作Animator.setInterpolator()方法的参数，如上。

1. **SVG**
2. 什么是SVG？

Scalable Vector Graphics，可缩放矢量图形。用数学语言描述二维矢量图形的一种图形格式。它由万维网联盟制定，是一个开放标准。

1. 与普通图片的区别？

* 用XML来定义图形
* 图片体积小，且和显示尺寸无关
* 可任意比例的缩放，且不会失真

1. 在Android显示静态SVG图片

* 新建Vector assets
* 像普通drawable一样使用

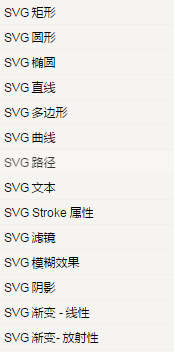
1. 效果

[演示视频](SVG显示.mp4)

1. SVG语法

SVG也算是一项独立的技术，它的全部语法可以在这个网站学习。

http://www.w3school.com.cn/svg/



但是对于Android开发者而言，不需要了解所有的语法，只要了解一部分命令符就可以了。因为我们如果用到SVG，也是用矢量图软件导出的，或者用Android Studio生成的。

矢量图软件有很多，最具代表性的就是Adobe illustrator，它和Photoshop并称矢量图和位图的两大神器。

Android开发者也可以用IDE生成。

先来看一下Android Studio自动生成的格式是什么样的？

**<vector xmlns:android="http://schemas.android.com/apk/res/android"**

**android:width="24dp"**

**android:height="24dp"**

**android:viewportWidth="24.0"**

**android:viewportHeight="24.0">**

**<path**

**android:fillColor="#FF000000"**

**android:pathData="M6,18c0,0.55 0.45,1 1,1h1v3.5c0,0.83 ...z"/>**

**</vector>**

根节点vector表示这是一个矢量资源图。

Width、height表示默认的显示高度，应用于wrap\_content属性。

viewportWidth、viewportHeight表示的是SVG描绘空间的大小，pathData中的数值都是根据这个计算的。将来图片在显示时的尺寸除以这个尺寸就是显示倍率，pathData中的数值会自动乘以这个倍率。

fillColor表示的是填充色，还有很多其它属性。

pathData表示的是矢量图形的数据，我们要学的语法也就在这里。

**M = moveto** 相当于Path的moveTo(),用于移动起始点

**L = lineto** 相当于Path的lineTo()，用于画线

**H = horizontal lineto** 用于画水平线

**V = vertical lineto** 用于画竖直线

**C = curveto** 相当于Path的cubicTo(),三次贝塞尔曲线

**S = smooth curveto** 同样三次贝塞尔曲线，更平滑

**Q = quadratic Belzier curve** 相当于Path的quadTo()，二次贝塞尔曲线

**T = smooth quadratic Belzier curveto** 同样二次贝塞尔曲线，更平滑

**A = elliptical Arc** 相当于Path的arcTo()，用于画弧

**Z = closepath** 相当于Path的closeTo(),关闭path

6、SVG动画

Android提供的animated-vector可以实现SVG的动画效果，其实就是两个SVG路径的切换。要求两个SVG的点数必须对应。

1. 实现

**<ImageView**

**android:layout\_width="200dp"**

**android:layout\_height="200dp"**

**android:src="@drawable/animvectordrawable"/>**

**<animated-vector xmlns:android="http://schemas.android.com/apk/res/android"**

**android:drawable="@drawable/vectordrawable" >**

**<target**

**android:name="rotationGroup"**

**android:animation="@anim/rotation" />**

**<target**

**android:name="v"**

**android:animation="@anim/path\_morph" />**

**</animated-vector>**

**<objectAnimator xmlns:android="http://schemas.android.com/apk/res/android"**

**android:duration="3000"**

**android:propertyName="rotation"**

**android:valueFrom="0"**

**android:valueTo="360" />**

**<set xmlns:android="http://schemas.android.com/apk/res/android">**

**<objectAnimator**

**android:duration="3000"**

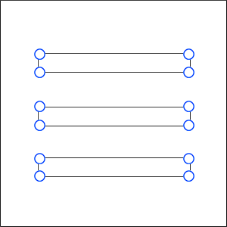
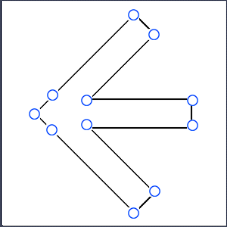
**android:propertyName="pathData"**

**android:valueFrom="M10,10 L60,10 L60,20 L10,20 Z M10,30 L60,30 L60,40 L10,40 Z M10,50 L60,50 L60,60 L10,60 Z"**

**android:valueTo="M5,35 L40,0 L47.072,7.072 L12.072,42.072 Z M10,30 L60,30 L60,40 L10,40 Z M12.072,27.928 L47.072,62.928 L40,70 L5,35 Z"**

**android:valueType="pathType" />**

**</set>**

** **

1. 效果

[演示视频](svg_anim.mp4)

1. SVG和Path的转换

目前Android中对SVG支持的api还不是很多，功能比较弱，SVG动画也不能进行控制。所以就产生了下面的PathView。

SVG的语法中常用的部分基本都能和Android中的Path函数吻合，所以很多人想到读取SVG信息，转换成Path信息，然后就可以进行更加丰富的操作了。

1. 效果

[演示视频](path_view.mp4)

1. 实现

看代码。

1. 目前的痛点

官方animated-vector提供的api很少，无法对SVG动画进行控制。

把SVG转换成Path的这种方式，解析很难。因为SVG标签很多，写法也很开放，同样的效果可能会有很多种写法，不同软件导出的SVG路径也不一样。Demo中的android.svg也是我从GIMP中导出后手动修改的。

V7中的DrawerArrowDrawable也是用Path来绘制的。

1. 总结

尽管SVG目前在Android中的功能还不是很多，但是已经可以看出它的强大之处，随着Google后续版本升级，SVG肯定会大方光彩。

当前我们可以使用SVG图片代替很多位图，因为本身SVG的体积就很小，而且只需要一套图。

也可以做一些比较酷炫的切换特效，比如上面的Demo。

也可以用开源框架转换成path作进一步的处理。

1. **Drawable.xml**

Android的drawable和SVG一样，也是xml，它提供了非常丰富的功能，用它提供的模板，我们可以用实现很多效果。其实很多项目中的切图都是可以用xml实现的。

这部分内容大多数都知道，所以就不再word中全写出来了，但是可能知道的不全，所以就用一个整合的demo来演示。一些细节的地方再作解释。

下面请看Demo。。。

**五、图片**

App的设计总是越来越复杂，效果越来越漂亮，这必然导致资源图片越来越多，app体积越来越大。所以App瘦身很重要的一点就是资源图的优化，在不影响图片质量的前提下，压缩图片大小。

**请看一组数据：**

QQ\_Android：**2242**张图片 **1.91MB**（2,012,157 字节）

Google Material：**2028**张图片 **908KB**（930018字节）

QQ\_Mail：**1723**张图片 **2.97MB**（2892991字节）

商户：**67**张图片 **2.5MB**（2216419字节）

 请允许我做一个悲伤的表情。。。。

1. **.9.png**

这个已经非常熟悉了，就不作介绍了。凡是能用.9做的，就用.9做，因为它和drawable.xml和SVG一样，都是自动拉伸，和分辨率无关的。

btn_tips_pressed.9 

**2、JPG**

jpg全名是JPEG。JPEG图片以 24 位颜色存储单个位图。JPEG 是与平台无关的格式，支持最高级别的压缩，不过，这种压缩是有损耗的。但是，文件大小是以牺牲图像质量为代价的。**压缩比率可以高达 100:1。（JPEG 格式可在 10:1 到 20:1 的比率下轻松地压缩文件，而图片质量不会下降。）JPEG压缩可以很好地处理写实摄影作品。但是，对于颜色较少、对比级别强烈、实心边框或纯色区域大的较简单的作品，JPEG压缩无法提供理想的结果。有时，压缩比率会低到 5:1，严重损失了图片完整性。这一损失产生的原因是，JPEG压缩方案可以很好地压缩类似的色调，但是 JPEG 压缩方案不能很好地处理亮度的强烈差异或处理纯色区域。**



**3、PNG**

可移植网络图形格式(Portable Network Graphic)，其设计目的是试图替代GIF和TIFF文件格式，同时增加一些GIF文件格式所不具备的特性。PNG用来存储灰度图像时，灰度图像的深度可多到16位，存储彩色图像时，彩色图像的深度可多到48位，并且还可存储多到16位的α通道数据。PNG使用从LZ77派生的无损数据压缩算法，一般应用于JAVA程序、网页或S60程序中，原因是它压缩比高，生成文件体积小。

特点：体积小、无损压缩、索引彩色模式、更优化的网络传输显示、支持透明通道



**4、WEB-PNG**

Photoshop在导出图片时，提供了两种方式，一种是“存储为”，一种是“存储为web所用格式”。

第二种方式就可以导出web-png。

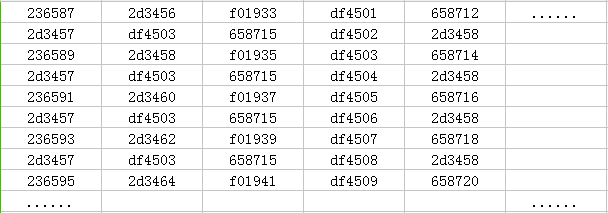
它们最大的区别就是第二种方式去除了所有图片数据本身之外的信息，比如拍摄时间、镜头参数、地理位置等，这样可以很大程度缩小图片体积（当然还有一些其它的差异我也没弄明白）。所以这个特性非常适合app的资源图。

**5、****索引图**

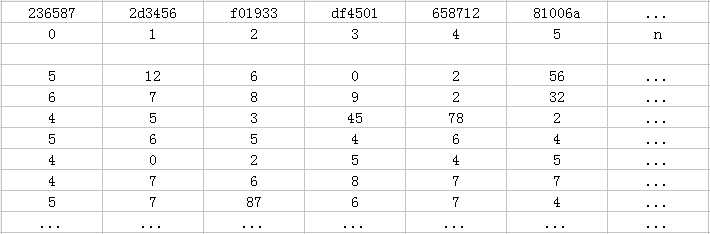
这是一种比较好的图片压缩方法，但是有局限性。

举例：

一个500 x500的图片，一个250000个像素，每一个像素由r、g、b三原色组成，那么它的图片数据是这样的：



而索引模式下是这样的：



优点：压缩图片体积，但却不影响图片质量。

缺点：颜色表最大256，不适用于色彩丰富的图片，适用于app图标类。

**6、****灰度图**

和索引图一样，灰度模式也可以压缩图片体积，但是也有局限性。

所谓灰度图就是所有颜色都是灰度色的图，那么什么是灰度色呢？

RGB色彩模式下，纯白：FFFFFF，纯黑：000000，纯红：FF0000，纯绿：00FF00，纯蓝：0000FF

R、G、B三种元素的不同比例混合出不同的彩色，R=G=B时，就是灰度色。



这张图是从纯白过渡到纯黑，这张图上的所有像素点都是灰度色。

既然灰度色的R=G=B，那就不需要用三个值来表示一个像素了，只需要一个值就够了。

808080 -> 80，F3F3F3 -> F3

这样就实现了图片数据的压缩。

优点：压缩图片体积，但却不影响图片质量。

缺点：只能适用于颜色全是灰度色的图片。如果图片中有彩色像素，经过灰度模式处理后，彩色就消失了。

PhotoShop导出不同格式图片的大小演示。

灰度模式之后还有位图模式、双色调模式，（三色调、四色调），这个就不研究了。

1. **Webp**

webp是谷歌2010年发布的一种新的图片格式，支持有损压缩或无损压缩。据官方称无损压缩的webp在体积上要比png小26%，而有损压缩要比同质量jpg小25%~34%。经本人测试，由腾讯智图处理的不同图片转换webp压缩率不太一样，在24%~83%之间都有。这样大的压缩率对于我们Web前端开发工程师诱惑力实在太大了。因为互联网很大一部分流量都来自图片，图片体积变小可以帮助我们让网页加载地更快，这对提升用户体验是有帮助的。

webp有损压缩使用了一种高级预测编码方法对图片进行编码，这种方法使用临近像素块中的值来预测当前块中的图像值，然后只需要对这种差异进行编码。

webp无损压缩使用已经可见的图像碎片来精确地重建一个个新的像素。这种压缩模式被称为“VP8L”，与LZ77压缩算法有一些共同特征。

那么，现在是否有产品使用webp呢？答案是肯定的，国外如Youtube、Gmail、Google Play、Chrome网上商店，国内如腾讯、淘宝、美团等都有webp的应用。科技博客 GigaOM 曾报道：YouTube 的视频略缩图采用 WebP 格式后，网页加载速度提升了 10%；谷歌的 Chrome 网上应用商店采用 WebP 格式图片后，每天可以节省几 TB 的带宽，页面平均加载时间大约减少 33%；Google+ 移动应用采用 WebP 图片格式后，每天节省了 50TB 数据存储空间。

WebP 的分析：

这里列举一个简单的测试：对比 PNG 原图、PNG 无损压缩、PNG 转 WebP（无损）、PNG 转 WebP（有损）的压缩效果。



可以得出结论：

1.PNG转WebP的压缩率要高于PNG原图压缩率，同样支持有损与无损压缩

2.转换后的WebP体积大幅减少，图片质量也得到保障（同时肉眼几乎无法看出差异）

3.转换后的WebP支持Alpha透明和24-bit颜色数，不存在PNG8色彩不够丰富和在浏览器中可能会出现毛边的问题

现在问题来了：WebP 的支持度和兼容性如何？

根据对目前国内浏览器占比与 WebP 的兼容性分析，大约有 50% 以上的国内用户可以直接体验到 WebP，如果你的网站以图片为主，或者你的产品基于 Chromium 内核，建议体验尝试。假如你打算在 App 中使用 WebP，除了 Android4.0 以上提供的原生支持外，其他版本以及 iOS 都可以直接使用官方提供的解析库（Android 、iOS ）。

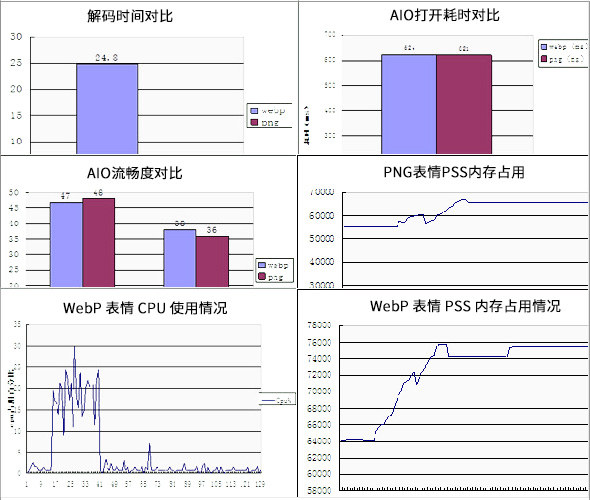
为了验证 WebP 图片格式的业务可行性，我们从流畅度、解码耗时、CPU 使用、内存占用几个维度进行的分析，在开发同学们的帮助下得到了非常宝贵的测试数据：

测试一：AndroidQQ 下 PNG 和 WebP 各指标对比。

测试环境：AndroidQQ、Galaxy Nexus、Android4.2.2 系统

测试对象：WebP、PNG

测试数据（部分）：



测试结论：

解码耗时：WebP 的解码时间是 PNG 格式的 4.4 倍（24.8ms）

流畅程度：两种格式下，AIO 滑动流畅度无明显差异

CPU使用：两种格式下，连续发送 15 个表情，CPU 使用均在 10%—26% 之间波动，两者无明显差异

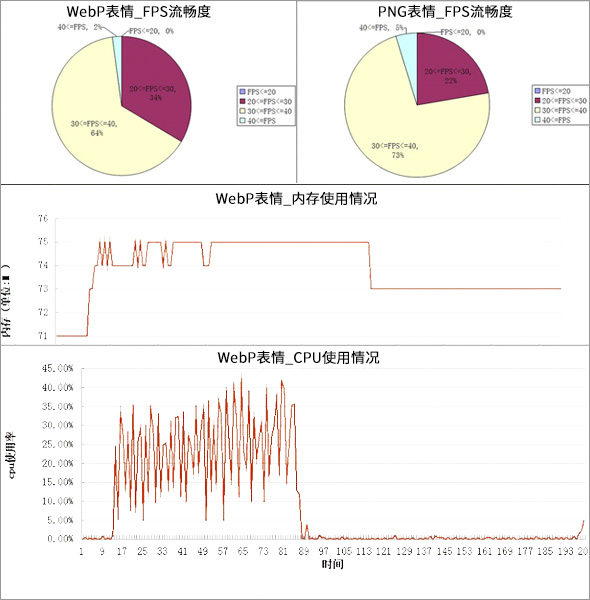
内存占用：两者格式下，连续发送 15 个表情，PSS 内存占用跨度均为 11M，无明显差异

测试二：iPhoneQQ 下 PNG 和 WebP 各指标对比。

测试环境：iPhoneQQ、iPhone4、iOS5.1.1 系统

测试对象：WebP、PNG

测试数据（部分）：



测试结论：

解码耗时：WebP 的解码时间是 PNG 的 5 倍 左右（64.1ms）

流畅程度：WebP 的 FPS 平均值会比 PNG 的平均值要小，但是比较稳定，跨度不大，性能也相差不大

CPU使用：总体上看，PNG 格式的表情使用 CPU 波动比较大。从平均值来看 WebP 格式表情占用的 CPU 会比 PNG 表情的占用率大

内存占用：WebP 格式表情，占用内存的跨度为 4M，波动比较明显。PNG 格式表情，占用内存的跨度为 5M，没有明显波动。停止发送表情后，40s 左右内存均有回降

可见除了 WebP 在解码时间与 PNG 有较明显差异（毫秒级别）之外，总体使用体验和 PNG 基本无差异。同时也需要明确，移动设备的发展迅猛，硬件升级快，上一年的表现也许在今年又有了明显的提升。所以，在 App 中使用 WebP 基本没有技术阻碍。

WebP图片转换

**iSparta、智图**

**演示**

1. **bitmap拼图**



上图的背景是一个黑白相间的块，如果要用一张整图作为背景，那体积太大了，而且在不同手机上方块还会有拉伸，如果每一种尺寸屏幕都提供一张，那体积就不得了了。

Android提供了拼图模式。

1. 准备一张单元图

tile

1. 编写drawable.xml

<bitmap xmlns:android="http://schemas.android.com/apk/res/android"

android:src="@drawable/tile"

android:tileMode="repeat" />

1. 使用

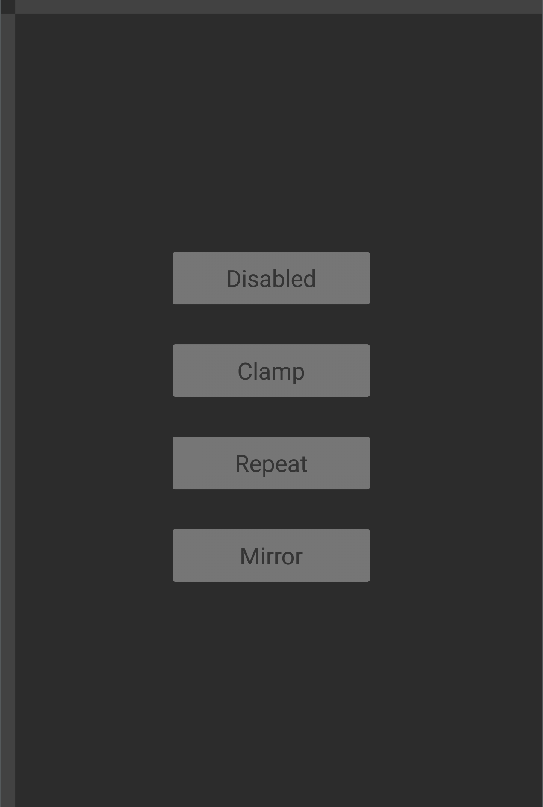
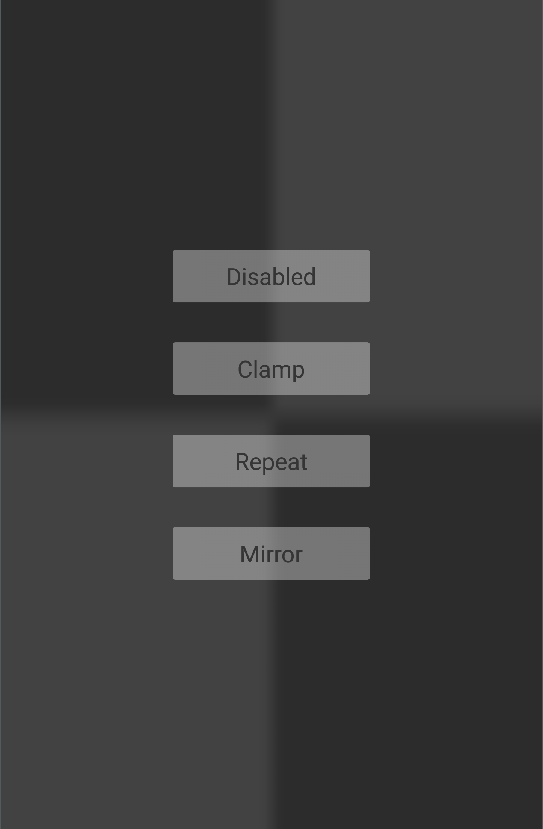
<View

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:background="@drawable/repeat" />

tileMode有四种：disabled、clamp、repeat、mirror





优点：

1）极大地减小了资源图的体积，而且显示效果很好。

2）模式可配，适合多种场景。

3）在任何尺寸屏幕上都不会拉伸。

**六、****总结**以上所有的分析都是为了一个最终目的，就是app资源图优化。不同的图片、格式都有各自的适用场景，对于Android开发者而言，如果选择优先顺序呢？

仅个人观点：

