

个人资料



学问积年而成



访问： 215492次
积分： 2562
等级： 5
排名： 第10259名

原创： 56篇
转载： 0篇
译文： 0篇
评论： 327条

文章搜索

文章分类

- android动效篇 (13)
 - 性能优化 (0)
 - 反编译 (0)
- android常见问题 (2)
- 基础总结 (20)
- javascript (5)
- javaweb (11)
- JDBC (3)
- AJAX (1)
- 心路历程 (0)

文章存档

- 2015年07月 (2)
- 2015年06月 (1)
- 2015年05月 (1)
- 2015年04月 (4)
- 2015年03月 (6)

展开

[攒课--我的学习我做主](#) [【hot】直播技术精选](#) [主流编程语言图谱之二](#)

[置顶] Paint、Canvas、Matrix使用讲解(一、Paint)

标签： [Paint](#) [Canvas](#) [Matrix](#) [安卓绘制](#) [android动画](#)

2015-03-18 17:56 10470人阅读 评论(24) 收藏 举报

分类：

[android动效篇 \(12\)](#)

版权声明：本文为博主原创文章，未经博主允许不得转载。

[目录\(?\)](#) [\[+\]](#)

我正在参加 CSDN 2015 博客之星评选 感恩分享活动，如果觉得文章还不错，请投个票鼓励下吧：<http://vote.blog.csdn.net/blogstar2015/candidate?username=tianjian4592>

好了，前面主要讲了Animation, Animator 的使用，以及桌面火箭效果和水波纹效果，分别使用android框架和自己绘制实现，俗话说，工欲善其事，必先利其器，接下来几篇文章主要讲绘制中我们需要常使用的一些利器：
Paint：画笔
Canvas：画布
Matrix：变换矩阵
绘制动效确实就像拿着笔在画布上面画画一样，而Paint就是我们拿着的笔，Canvas就是使用的画布；

一、Paint（画笔）

根据我们要画的类型，我们可以选择不同的笔，比如大气磅礴的山水画，我们可以选择大头的毛笔；细腻入微的肖像画我们可以选择尖头的铅笔。并且根据我们想要的效果，我们在绘画的时候，还会选择不同的颜料或不同颜色的笔；
那么在程序中，Paint 就足以满足以上所有的需要，我们可以根据我们自己的需要去自行设置我们画笔的属性，首先来看看都能设置哪些属性：

Paint 有三个构造函数，分别是：

Paint() 创建一个画笔对象；
Paint(int flags):在构造的时候可以传入一些定义好的属性，eg: Paint.ANTI_ALIAS_FLAG 一一用于绘制时抗锯齿
Paint(Paint paint):使用构造函数中Paint的属性生成一个新的Paint

```
[html] 1
01. private void initPaint() {
02.     // 构建Paint时直接加上锯齿属性
03.     mColorPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
04.     // 直接使用mColorPaint的属性构建mBitmapPaint
05.     mBitmapPaint = new Paint(mColorPaint);
06. }
```

方法常用的主要有以下一些：

setARGB (int a, int r, int g, int b):用于设置画笔颜色，A 代表 alpha（透明度），R 代表Red（红色），G 代表Green（绿色），B 代表Blue（蓝色），色值采用16进制，取值在 0 — 255 之间，0 (0x00) 即 完全没有，255 (0xff) 代表满值；
setAlpha(int a): 用于设置Paint 的透明度；
setColor(int color):同样设置颜色，如果是常用色，可以使用Color 类中定义好的一些色值，eg: Color.WHITE
setColorFilter(ColorFilter filter):设置颜色过滤器，可以通过颜色过滤器过滤掉对应的色值，比如去掉照片中的背景色

阅读排行

- Android使用SVG矢量图: (32894)
- 一个绚丽的loading动效分析 (27408)
- Canvas开篇之drawBitmap: (17901)
- 自定义view实现水波纹效果 (12301)
- Android Paint之 setXfer (11566)
- Paint、Canvas、Matrix (10450)
- android动效开篇 (7670)
- Canvas之translate、scale (7395)
- 关于引入第三方jar包引发 (7372)
- Path特效之PathMeasure (6842)

评论排行

- 一个绚丽的loading动效分析 (170)
- 自定义view实现水波纹效果 (32)
- Paint、Canvas、Matrix (24)
- Android使用SVG矢量图: (18)
- Android Paint之 setXfer (18)
- Canvas开篇之drawBitmap: (15)
- Canvas之translate、scale (10)
- Animation & Property Animation (9)
- android动效开篇 (8)
- Android 漂浮类动效的分 (7)

最新评论

- 关于引入第三方jar包引发的java.SDadmin: 不错，搞定了
- 一个绚丽的loading动效分析与实肖天师: 稍作修改移至GitHub，详见<https://github.com/cpxiao/LeafProgressView>
- 一个绚丽的loading动效分析与实rainberda: 膜拜大神
- 自定义view实现水波纹效果One_Sun: 博主您好，我现在有个需求，刚好用到了您的正余弦波浪动效的绘制demo,但是现在遇到个问题，解决了好久...
- 一个绚丽的loading动效分析与实kunhdd: 拜读 >.< 楼主好帅。
- 一个绚丽的loading动效分析与实我太认真: 感觉这个叶子的数量和进度条的速度有关系。这样更真实点。
- 一个绚丽的loading动效分析与实zhang_ying_xian: 感觉这篇帖子讲的没有头绪，直接看看不懂，还是要先把源代码敲一遍，才知道在讲啥，博主这个讲解方式有待改...
- 一个绚丽的loading动效分析与实西西弗斯丶: 分析能力不错，程序员就要这种思维~!
- 一个绚丽的loading动效分析与实seeyou_boy: 屌屌的文章，为啥没人评论？怒赞！！!
- Canvas之translate、scale、rotate code_river: 感谢楼主分享

推荐文章

- * Chromium扩展（Extension）机制简要介绍和学习计划
- * Android官方开发文档Training系列课程中文版：APP的内存管

ColorFilter有以下几个子类可用：

- ColorMatrixColorFilter
- LightingColorFilter
- PorterDuffColorFilter

1. ColorMatrixColorFilter:通过颜色矩阵（ColorMatrix）对图像中的色值进行改变

在Android中，图片是以一个个 RGBA 的像素点的形式加载到内存中的，所以如果需要改变图片的颜色，就需要针对这一个个像素点的RGBA的值进行修改，其实主要是RGB，A是透明度；

修改图片 RGBA 的值需要ColorMatrix类的支持，它定义了一个 4*5 的float[]类型的矩阵，矩阵中每一行表示RGBA 中的一个参数。

颜色矩阵M是以一维数组m=[a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t]的方式进行存储的：

$$M = \begin{bmatrix} a & b & c & d & e \\ f & g & h & i & j \\ k & l & m & n & o \\ p & q & r & s & t \end{bmatrix}$$

颜色矩阵

而对于一张图像来说，展示的颜色效果取决于图像的RGBA（红色、绿色、蓝色、透明度）值。而图像的 RGBA 值则存储在一个5*1的颜色分量矩阵C中，由颜色分量矩阵C可以控制图像的颜色效果。颜色分量矩阵如图所示：

$$C = \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix}$$

所以为了改变图像的显示效果，只需要改变 4*5 的颜色矩阵ColorMatrix，然后通过

$$C1 = M * C = \begin{bmatrix} a & b & c & d & e \\ f & g & h & i & j \\ k & l & m & n & o \\ p & q & r & s & t \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix} = \begin{bmatrix} aR + bG + cB + dA + e \\ fR + gG + hB + iA + j \\ kR + lG + mB + nA + o \\ pR + qG + rB + sA + t \end{bmatrix} = \begin{bmatrix} R1 \\ G1 \\ B1 \\ A1 \end{bmatrix}$$

即可得到新的图像显示矩阵；

由此可见，通过颜色矩阵 ColorMatrix 修改了原图像的 RGBA 值，从而达到了改变图片颜色效果的目的。并且，通过如上图所示的运算可知，颜色矩阵 ColorMatrix 的第一行参数abcde决定了图像红色成分，第二行参数fghij决定了图像的绿色成分，第三行参数klmno决定了图像的蓝色成分，第四行参数pqrst决定了图像的透明度，第五列参数ejot是颜色的偏移量。

通常，改变颜色分量时可以通过修改第5列的颜色偏移量来实现，如上面所示的颜色矩阵，通过计算后可以得知该颜色矩阵的作用是使图像红色分量和绿色分量均增加100，这样的效果就是图片泛黄（因为红色与绿色混合后得到黄色）：

$$M1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 100 \\ 0 & 1 & 0 & 0 & 100 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

除此之外，也可以通过直接对颜色值乘以某一系数而达到改变颜色分量的目的。如下图所示的颜色矩阵，将绿色分量放大了2倍，这样的效果就是图片泛绿色：

$$M2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

基于此，我们利用ColorFilter 和 ColorMatrixColorFilter类和 Paint 的setColorFilter 就可以改变图片的展示效果（颜色，饱和度，对比度等），从而得到类似市面上图像软件中的黑白老照片、泛黄旧照片、羞涩的青春...特效；

- 理
- * 程序员，别了校园入了江湖
 - * RxJava 合并组合两个（或多个）Observable数据源
 - * 探索Android软键盘的疑难杂症



(调节效果)



(原图效果)

[html] C P

```
01. // 通过外层传入的 A、R、G、B值生成对应的ColorMatrix，然后重新绘制图像
02. public void setArgb(float alpha, float red, float green, float blue) {
03.     mRedFilter = red;
04.     mGreenFilter = green;
05.     mBlueFilter = blue;
06.     mAlphaFilter = alpha;
07.     mColorMatrix = new ColorMatrix(new float[] {
08.         mRedFilter, 0, 0, 0, 0,
09.         0, mGreenFilter, 0, 0, 0,
10.         0, 0, mBlueFilter, 0, 0,
11.         0, 0, 0, mAlphaFilter, 0,
12.     });
13.     mPaint.setColorFilter(new ColorMatrixColorFilter(mColorMatrix));
14.
15.     postInvalidate();
16. }
```

在Activity中拖动seekbar时，动态改变对应的色值参数，然后设置给里面的View:

[html] C P

```
01. @Override
02. public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
03.     float filter = (float) progress / 100;
04.     if (seekBar == mRedSeekBar) {
05.         mRedSeek = filter;
06.     } else if (seekBar == mGreenSeekBar) {
07.         mGreenSeek = filter;
08.     } else if (seekBar == mBlueSeekBar) {
09.         mBlueSeek = filter;
10.     } else if (seekBar == mAlphaSeekBar) {
11.         mAlphaSeek = filter;
12.     }
13.     mColorFilterView.setArgb(mAlphaSeek, mRedSeek, mGreenSeek, mBlueSeek);
14. }
```

我们再专门把绿色位置扩大为两倍，看下效果:

[html] C P

```
01. private static final float[] mMatrixFloats = new float[] {
02.     1, 0, 0, 0, 0,
03.     0, 2, 0, 0, 0,
04.     0, 0, 1, 0, 0,
05.     0, 0, 0, 1, 0
06. };
```



可以看到这个图片绿的敞亮

我们继续修改影响绿色的这一排矩阵值，看会发生什么变化，这次我们只是把2改大一点，改为10看看：

```
[html] C P
01. private static final float[] mMatrixFloats = new float[] {
02.     1, 0, 0, 0, 0,
03.     0, 10, 0, 0, 0,
04.     0, 0, 1, 0, 0,
05.     0, 0, 0, 1, 0
06. };
```



这次我们已经觉得图片绿的过头了

我们再将影响绿色的矩阵值改小，大家已经可以猜测下图片的效果了，那就是绿色会成倍的减弱，减弱到一定的层度就会显得发灰：

而最后面的颜色增量也是一样，可以根据数值的大小，正负一定程度上增强和减弱对应的色值；

上面的demo只是通过动态的改变上面颜色矩阵里的a、g、s、m 值，进而改变图像对应的色值分量，达到改变图片显示效果的目的，我们还可以通过改变矩阵的其他数据来调节出各种有意思的效果：

修改影响红色的这一排矩阵：

```
[html] C P
01. private static final float[] mMatrixFloats = new float[] {
02.     2, 0.5f, 0.5f, 0.5f, 20,
03.     0, 1, 0, 0, 0,
04.     0, 0, 1, 0, 0,
05.     0, 0, 0, 1, 0
06. };
```



继续改：

```
[html] C P
01. private static final float[] mMatrixFloats = new float[] {
02.     0.22f, 0.5f, 0.1f, 0, 0,
03.     0.22f, 0.5f, 0.1f, 0, 0,
04.     0.22f, 0.5f, 0.1f, 0, 0,
05.     0, 0, 0, 1, 0
06. };
```



这下图片变灰了

再改:

```
[html] C ?
01. private static final float[] mMatrixFloats = new float[] {
02.     -1, 0, 0, 1, 0,
03.     0, -1, 0, 1, 0,
04.     0, 0, -1, 1, 0,
05.     0, 0, 0, 1, 0
06. };
```



将RGB均反相，但如果只这样会变成黑色，然后再增强一下；

再改:

```
[html] C ?
01. private static final float[] mMatrixFloats = new float[] {
02.     0, 0, 1, 0, 0,
03.     1, 0, 0, 0, 0,
04.     0, 2, 0, 0, 0,
05.     0, 0, 0, 1, 0
06. };
```



根据上面的矩阵，我们相当于去除了图片原先的RGB，然后原先的蓝色转变为红色，原先的红色转变为绿色，原先色绿色转变为蓝色，并变为2倍，经过以上变化，则形成了上面最终的效果；
所以通过修改颜色矩阵值，可以调出各种图片展示效果，如：高对比度、低对比度、高饱和度、低饱和度、负片等等，大家可以自己体验体验；

2. LightingColorFilter:

```
[html] C ?
01. /**
02.  * Create a colorfilter that multiplies the RGB channels by one color, and then adds a second c
03.  * pinning the result for each component to [0..255]. The alpha components of the mul and add i
04.  * are ignored.
05.  */
```

```

06. public LightingColorFilter(int mul, int add) {
07.     native_instance = native_CreateLightingFilter(mul, add);
08.     nativeColorFilter = nCreateLightingFilter(native_instance, mul, add);
09. }

```

从源码上

看, LightingColorFilter 只有一个构造方法, 然后传入 mul 和 add , 这两个 分别是16进制色值, 定义如下:

```

[html] C ⌘
01. private static final int MUL_COLOR = 0xff00ffff;
02. private static final int ADD_COLOR = 0x000000ff;

```

最终颜色的计算方式为: $(MUL_COLOR * 原色值 + ADD_COLOR) \% 255$

所以上面则能够过滤掉红色色值, 并增强蓝色色值, 另外从源码的注释上也可以看到LightingColorFilter 是不计算

3. PorterDuffColorFilter: 可以类比于PS里的 图层混合模式

```

[html] C ⌘
01. /**
02.  * Create a colorfilter that uses the specified color and porter-duff mode.
03.  *
04.  * @param srcColor      The source color used with the specified
05.  *                      porter-duff mode
06.  * @param mode          The porter-duff mode that is applied
07.  */
08. public PorterDuffColorFilter(int srcColor, PorterDuff.Mode mode) {
09.     native_instance = native_CreatePorterDuffFilter(srcColor, mode.nativeInt);
10.     nativeColorFilter = nCreatePorterDuffFilter(native_instance, srcColor, mode.nativeInt);
11. }

```

其实就是用 一个色值和要绘制的图片进行叠加, 然后可以选择叠加的模式

(PorterDuff.Mode), PorterDuff.Mode 以枚举的形式定义, 共有如下一些:

```

[html] C ⌘
01. // these value must match their native equivalents. See SkPorterDuff.h
02. public enum Mode {
03.     /** [0, 0] */
04.     CLEAR      (0),
05.     /** [Sa, Sc] */
06.     SRC        (1),
07.     /** [Da, Dc] */
08.     DST        (2),
09.     /** [Sa + (1 - Sa)*Da, Rc = Sc + (1 - Sa)*Dc] */
10.     SRC_OVER   (3),
11.     /** [Sa + (1 - Sa)*Da, Rc = Dc + (1 - Da)*Sc] */
12.     DST_OVER   (4),
13.     /** [Sa * Da, Sc * Da] */
14.     SRC_IN     (5),
15.     /** [Sa * Da, Sa * Dc] */
16.     DST_IN     (6),
17.     /** [Sa * (1 - Da), Sc * (1 - Da)] */
18.     SRC_OUT    (7),
19.     /** [Da * (1 - Sa), Dc * (1 - Sa)] */
20.     DST_OUT    (8),
21.     /** [Da, Sc * Da + (1 - Sa) * Dc] */
22.     SRC_ATOP   (9),
23.     /** [Sa, Sa * Dc + Sc * (1 - Da)] */
24.     DST_ATOP   (10),
25.     /** [Sa + Da - 2 * Sa * Da, Sc * (1 - Da) + (1 - Sa) * Dc] */
26.     XOR        (11),
27.     /** [Sa + Da - Sa*Da,
28.         Sc*(1 - Da) + Dc*(1 - Sa) + min(Sc, Dc)] */
29.     DARKEN     (12),
30.     /** [Sa + Da - Sa*Da,
31.         Sc*(1 - Da) + Dc*(1 - Sa) + max(Sc, Dc)] */
32.     LIGHTEN    (13),
33.     /** [Sa * Da, Sc * Dc] */
34.     MULTIPLY    (14),
35.     /** [Sa + Da - Sa * Da, Sc + Dc - Sc * Dc] */
36.     SCREEN     (15),
37.     /** Saturate(S + D) */
38.     ADD        (16),

```


39. OVERLAY (17);

以上公式中, Sa 代表 Source alpha (源透明度 —— 传入颜色的透明度), Da 代表 Destination alpha (目标 alpha), Sc 代表 Source Color (源颜色), Dc 代表 Destination Color (目标色); 每一种模式的计算方法分别如上;
我们简单测试下, 还是用前面例子的图:



```
[html] C P
01. mBitDuffPaint.setColorFilter(new PorterDuffColorFilter(DUFF_COLOR, PorterDuff.Mode.DARKEN));
    --- 变暗
```

我们看一下效果:



这时得出的图片效果和PS里对两个图层使用变暗的叠加效果是一样的, PS里的效果如下:



我们继续调几个模式看看效果, 变亮效果:

```
[html] C P
01. mBitDuffPaint.setColorFilter(new PorterDuffColorFilter(DUFF_COLOR, PorterDuff.Mode.LIGHTEN));
```

onDraw 中:

```
[html] C P
01. canvas.drawBitmap(mBitmap, mSrcRect, mDestRect, mBitDuffPaint);
```

程序 和 PS 展示效果分别为:



我上面只是使用蓝色 和图片进行了混合，大家可以使用其他颜色再调的玩玩，其中具体的叠加规律大家也可以自己稍加总结；

好了，颜色过滤器就讲这么多；

setDither(boolean dither):防抖动，这个属性的需求场景主要出现在绘制渐变色彩或含渐变的图片时，android对成为RGB565 格式的，这种格式占用内存小，但因为如此，就会出现讨厌的“色带”情景，让人感觉过渡的不是那么防抖动，它会将原始颜色的过渡处根据两边的色值进行一些改变，从而让颜色过渡更加的柔和，让人觉得是平滑的。
setFilterBitmap(boolean filter):如果该项设置为true，则图像在动画进行中会滤掉对Bitmap图像的优化操作，加
setFlags(int flags):可以用来给Paint设置里面定义好的一些属性，如抗锯齿，防抖动等；
setMaskFilter(MaskFilter maskFilter):设置绘制时图片边缘效果，可以有模糊和浮雕；
MaskFilter类可以为Paint分配边缘效果。

对MaskFilter的扩展可以对一个Paint边缘的alpha通道应用转换。Android包含了下面几种MaskFilter

BlurMaskFilter 指定了一个模糊的样式和半径来处理Paint的边缘；

EmbossMaskFilter 指定了光源的方向和环境光强度来添加浮雕效果；

setPathEffect(PathEffect effect):是用来控制绘制轮廓(线条)的方式：

这个类本身并没有做什么特殊的处理，只是继承了Object，通常我们使用的是它的几个子类，就是上面图片中所显示的。在使用的时候，通常是

PathEffect pe = new 一个具体的子类；

然后使用Paint的setPathEffect(PathEffect pe)方法即可。

CornerPathEffect:

这个类的作用就是将Path的各个连接线段之间的夹角用一种更平滑的方式连接，类似于圆弧与切线的效果；

一般的，通过CornerPathEffect(float radius)指定一个具体的圆弧半径来实例化一个CornerPathEffect；

DashPathEffect:

这个类的作用就是将Path的线段虚线化：

构造函数为DashPathEffect(float[] intervals, float phase)，其中intervals为虚线的ON和OFF数组，该数组的length必须大于等于2，phase为绘制时的偏移量。

DiscretePathEffect:

这个类的作用是打散Path的线段，使得在原来路径的基础上发生打散效果。

一般的，通过构造DiscretePathEffect(float segmentLength, float deviation)来构造一个实例，其中，segmentLength指定最大的段长，deviation指定偏离量。

PathDashPathEffect:

这个类的作用是使用Path图形来填充当前的路径，其构造函数为PathDashPathEffect (Path shape, float advance, float phase, PathDashPathEffect.Style style)。

shape则是指填充图形，advance指每个图形间的间距，phase为绘制时的偏移量，style为该类的枚举值，有三种情况：Style. ROTATE、Style. MORPH和Style. TRANSLATE；

其中ROTATE的情况下，线段连接处的图形转换以旋转到与下一段移动方向相一致的角度进行转转，MORPH时图形会以发生拉伸或压缩等变形的情况与下一段相连接，TRANSLATE时，图形会以位置平移的方式与下一段相连接。

ComposePathEffect:

组合效果，这个类需要两个PathEffect参数来构造一个实

例，ComposePathEffect (PathEffect outerpe, PathEffect innerpe)，表现时，会首先将innerpe表现出来，

然后再在innerpe的基础上去增加outerpe的效果：

SumPathEffect：

叠加效果，这个类也需要两个PathEffect作为参数SumPathEffect(PathEffect first, PathEffect second)，但与ComposePathEffect不同的是，在表现时，会分别对两个参数的效果各自独立进行表现，然后将两个效果简单的重叠在一起显示出来；

关于参数phase

在存在phase参数的两个类里，如果phase参数的值不停发生改变，那么所绘制的图形也会随着偏移量而不断的发生变动，这个时候，看起来这条线就像动起来了一样：

举个栗子：

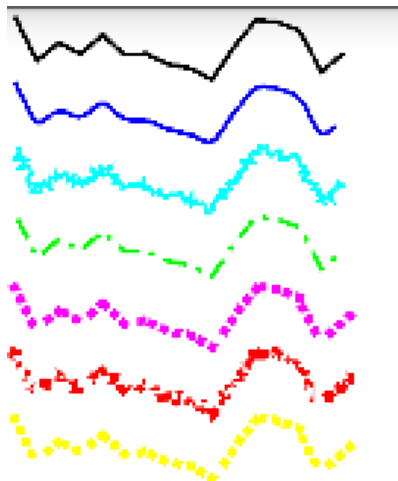
```
[html]
01. public class PathEffectView extends View {
02.
03.     private float mPhase;
04.     private PathEffect[] mEffects = new PathEffect[7];
05.     private int[] mColors;
06.     private Paint mPaint;
07.     private Path mPath;
08.
09.     public PathEffectView(Context context) {
10.         super(context);
11.         mPaint = new Paint();
12.         mPaint.setStyle(Paint.Style.STROKE);
13.         mPaint.setStrokeWidth(4);
14.         // 创建,并初始化Path
15.         mPath = new Path();
16.         mPath.moveTo(0, 0);
17.         for (int i = 1; i <= 15; i++)
18.         {
19.             // 生成15个点,随机生成它们的坐标,并将它们连成一条Path
20.             mPath.lineTo(i * 20, (float) Math.random() * 60);
21.         }
22.         // 初始化七个颜色
23.         mColors = new int[] {
24.             Color.BLACK, Color.BLUE, Color.CYAN,
25.             Color.GREEN, Color.MAGENTA, Color.RED, Color.YELLOW
26.         };
27.     }
28.
29.     protected void onDraw(Canvas canvas)
30.     {
31.         // 将背景填充成白色
32.         canvas.drawColor(Color.WHITE);
33.         // -----下面开始初始化7种路径的效果
34.         // 使用路径效果——原始效果
35.         mEffects[0] = null;
36.         // 使用CornerPathEffect路径效果--参数为圆角半径
37.         mEffects[1] = new CornerPathEffect(10);
38.         // 初始化DiscretePathEffect —— segmentLength指定最大的段长, deviation指定偏离量
39.         mEffects[2] = new DiscretePathEffect(3.0f, 5.0f);
40.         // 初始化DashPathEffect —— intervals为虚线的ON和OFF数组, offset为绘制时的偏移量
41.         mEffects[3] = new DashPathEffect(new float[] {
42.             20, 10, 5, 10
43.         }, mPhase);
44.         // 初始化PathDashPathEffect
45.         Path p = new Path();
46.         p.addRect(0, 0, 8, 8, Path.Direction.CCW);
47.         // shape则是指填充图形, advance指每个图形间的间距, phase为绘制时的偏移量, style为该类自由的枚举值
48.         mEffects[4] = new PathDashPathEffect(p, 12, mPhase, PathDashPathEffect.Style.ROTATE);
49.         // 组合效果
50.         mEffects[5] = new ComposePathEffect(mEffects[2], mEffects[4]);
51.         // 叠加效果
52.         mEffects[6] = new SumPathEffect(mEffects[4], mEffects[3]);
53.         // 将画布移到8,8处开始绘制
54.         canvas.translate(8, 8);
55.         // 依次使用7种不同路径效果,7种不同的颜色来绘制路径
56.         for (int i = 0; i < mEffects.length; i++)
57.         {
58.             mPaint.setPathEffect(mEffects[i]);
59.             mPaint.setColor(mColors[i]);
60.             canvas.drawPath(mPath, mPaint);
61.             canvas.translate(0, 60);
62.         }
63.     }
64. }
```

```

63.         // 改变phase值,形成动画效果
64.         mPhase += 1;
65.         invalidate();
66.     }
67. }

```

效果为:



```
setShader(Shader shader);
```

设置图像效果,使用Shader可以绘制出各种渐变效果;

Shader下面有五个子类可用:

BitmapShader :位图图像渲染

LinearGradient:线性渲染

RadialGradient:环形渲染

SweepGradient:扫描渐变渲染/梯度渲染

ComposeGradient:组合渲染,可以和其他几个子类组合起来使用

这几个类中LinearGradient、RadialGradient、SweepGradient均是可以将颜色进行处理,形成柔和的过渡,也可以称为渐变,而BitmapShader 则是直接使用位图进行渲染,就类似于贴图,在贴图的过程中根据需要自然就可以选择相应的模式,有三种模式可供选择,分别是:

枚举:

```
enum Shader.TileMode
```

定义了平铺的3种模式:

static final Shader.TileMode CLAMP: 边缘拉伸,即使用边缘的最后一个像素进行延展;

static final Shader.TileMode MIRROR: 在水平方向和垂直方向交替景象,两个相邻图像间没有缝隙,从名称上看就像照镜子一样;

Static final Shader.TillMode REPETA: 在水平方向和垂直方向重复摆放,两个相邻图像间有缝隙缝隙;

Shader在应用中也是非常的广泛,不少大产品中的亮点设计也与之相关,接下来我们分别看个小例子:

1. LinearGradient (线性渲染——也叫线性渐变)

我们在PS上如果要拉出一条线性渐变,只需要先把渐变模式选为线性,然后拉出一条渐变线,同理,在android中,只需要一个起始点和一个终点即可确定一条渐变线,那么颜色如何渐变过去呢,往下看,LinearGradient有两个构造方法:

```

[html] C P
01.     /** Create a shader that draws a linear gradient along a line.
02.     @param x0      The x-coordinate for the start of the gradient line
03.     @param y0      The y-coordinate for the start of the gradient line
04.     @param x1      The x-coordinate for the end of the gradient line
05.     @param y1      The y-coordinate for the end of the gradient line
06.     @param color0   The color at the start of the gradient line.
07.     @param color1  The color at the end of the gradient line.
08.     @param tile     The Shader tiling mode
09.     */
10.     public LinearGradient(float x0, float y0, float x1, float y1, int color0, int color1,
11.                           TileMode tile) {--省略--}

```

这个构造方法就是传入起点和终点作为渐变线，然后color0作为起始点的颜色值，color1作为终点的颜色值，然后生成的效果就是从起始点到终点从color0均匀的渐变到color1，注意是均匀：
我们来看下效果：

```
[html] C P
01. // 定义几种初始颜色
02. private static final int COLOR_BLUE = 0xff0000ff;
03. private static final int COLOR_GREEN = 0xff00ff00;
04. private static final int COLOR_RED = 0xffff0000;
05.
06. // 定义起始和最终颜色
07. private int mStartColor = COLOR_BLUE;
08. private int mToColor = COLOR_RED;
```

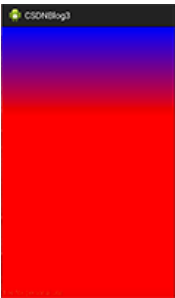
然后创建一个从屏幕上中心点到屏幕1/3高度的下中心点的渐变：

```
[html] C P
01. new LinearGradient(mHalfWidth, 0, mHalfWidth, mTotalHeight/3, mStartColor, mToColor, Shader.Ti:
```

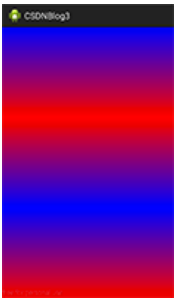
绘制一个全屏矩形：

```
[html] C P
01. canvas.drawRect(0, 0, mTotalWidth, mTotalHeight,
02. mGradientPaint);
```

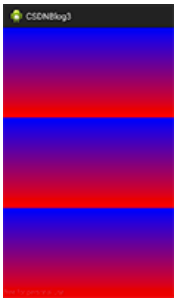
看下三种模式下的效果：



(Shader.TileMode.CLAMP)



(Shader.TileMode.MIRROR)

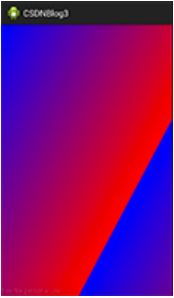
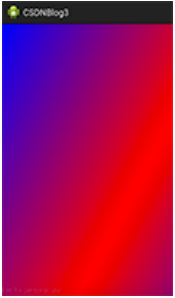
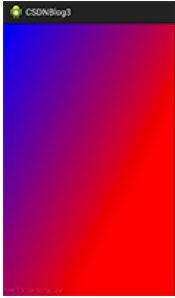


(Shader.TileMode.REPEAT)

接下来改下渐变起始点和终点为左上角和1/3屏幕高度的右下角：

```
[html] C P
01. new LinearGradient(0, 0, mTotalWidth, mTotalHeight/3, mStartColor, mToColor, Shader.TileMode.Cl
```

还是绘制同样的矩形，看下三种模式下的效果：



(Shader.TileMode.CLAMP) (Shader.TileMode.MIRROR)
(Shader.TileMode.REPEAT)

仔细看三种模式下的渐变，可能有些人会觉得奇怪，因为感觉图一次的结尾处并不是1/3高度处，原因是啥呢？是因为渐变线是斜角，大家可以看REPEAT模式下的图，一次结尾处在屏幕右边的位置刚好是屏幕 1/3 处，看到这儿应该清楚了吧；

渐变ok了，但有时候我们的需求不止于此，还需要多种颜色渐变，并且还得控制渐变的大小，比如需要红绿蓝三种颜色渐变，并且还要 红色占比 10%，绿色占比 60%，蓝色占比 30%，这个时候该怎么做呢？我们来看 LinearGradient 的第二个构造函数：

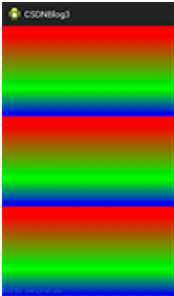
```
[html] C #
01.      /** Create a shader that draws a linear gradient along a line.
02.      @param x0      The x-coordinate for the start of the gradient line
03.      @param y0      The y-coordinate for the start of the gradient line
04.      @param x1      The x-coordinate for the end of the gradient line
05.      @param y1      The y-coordinate for the end of the gradient line
06.      @param colors  The colors to be distributed along the gradient line
07.      @param positions May be null. The relative positions [0..1] of
08.                     each corresponding color in the colors array. If this is null,
09.                     the the colors are distributed evenly along the gradient line.
10.      @param tile    The Shader tiling mode
11.      */
12.      public LinearGradient(float x0, float y0, float x1, float y1, int colors[], float positions[],
13.                             TileMode tile) {---省略---
```

其中 colors 代表要使用的颜色色值数组，positions 代表与颜色对应的相应的位置数组，位置数组里的取值从 0 —1，分别对应从开始到结束，这样里面的每一个值就对应了距离起始点的距离比例，并且使用该方式，则默认为有多种颜色，所以颜色数组长度必须大于等于2，并且两数组长度必须一致，这么说可能还不是特清晰，我们来完成上面的需求—需要红黄蓝三种颜色渐变，并且还要 红色占比 10%，黄色占比 60%，蓝色占比 30%：

好，咱们按照上面的说法，把对应颜色的位置按照比例设置上，看下效果：

```
[html] C #
01.      // 颜色数组
02.      private int[] mColors = new int[] {
03.          COLOR_RED, COLOR_GREEN, COLOR_BLUE
04.      };
05.      // 与颜色数组对应的位置数组
06.      private float[] mPositions = new float[] {
07.          0.1f, 0.7f, 1f
08.      };
```

我们就以 REPEAT 模式为例，看看效果：



(Shader.TileMode.REPEAT)

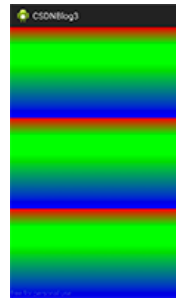
我们发现根本和我们想要的比例不一样，并不是 红：绿：蓝 = 1：6：3，反而 接近于 4：4.5：1.5，为什么？

那是因为 positions 里面代表的是对应颜色的比例位置，红色在 10% 处，绿色在 70% 处，蓝色在 100% 处，那么 红色和 绿色之间则有 60%的位置用于渐变，那么平均下来红色和绿色分别可以占据 30% 的位置（当然中间是渐变的，只是估算大概位置），同理可以计算出 绿 和 蓝 渐变的过程中各占 15% 的位置，综合起来则是 4:4.5:1.5；

这么计算下来，我们可以根据需求可以算出 红 和 绿 的距离为20%，绿和蓝的距离为 60%，这样可以的大致算出颜色的位置应该为一红：绿：绿：蓝 = 0f, 0.2f, 0.4f, 1f;

```
[html] C P
01. // 颜色数组
02. private int[] mColors = new int[] {
03.     COLOR_RED, COLOR_GREEN, COLOR_GREEN, COLOR_BLUE
04. };
05. // 与颜色数组对应的位置数组
06. private float[] mPositions = new float[] {
07.     0f, 0.2f, 0.4f, 1f
08. };
```

展示效果如下：



(Shader.TileMode.REPEAT)

好了，LinearGradient 就说这么多；

2. 接下来我们来看 RadialGradient （环形渲染）：

熟悉了 LinearGradient （线性渲染）之后，RadialGradient （环形渲染）就变得很简单了，看源码可知同样有两个可用的构造函数：

```
[html] C P
01. <span style="font-family:SimSun;font-size:14px;">public RadialGradient(float x, float y, float radius,int color0, int color1, TileMode
</span>
```

该函数里的 x,y分别代表 RadialGradient （环形渲染）所基于的中心点，color0 代表中心点的颜色，color1 代表结尾处的颜色，tile 同样代表Shader.TileMode （平铺模式）；

```
[html] C P
01. public RadialGradient(float x, float y, float radius,int colors[], float positions[], TileMode
```

该方法里与上面不同的就是单一颜色变成了颜色数组，并与之对应一个位置比例数组，对应关系和上面 LinearGradient （线性渲染）一样；

那么使用RadialGradient （环形渲染）我们可以做些什么呢，举个比较实用的小栗子，用于给大家开脑洞：

eg: 用于做柔和的渐变背景：

在产品的设计中，为了美观，一般背景都不会是纯色，要么加上一些光泽，要么是渐变，要么加上一些其他的元素，而渐变的时候，线性渐变相对会显得比较硬，所以设计师在做界面的时候很可能会选择环形渐变这样的方式，如下图：



对于这样的效果，如果直接切图的话在xxhdpi (1920 * 1080)这样的分辨率上估计少则会有几十K，并且为了控制在小分辨率手机上运行时的内存大小，则还需要切对应小型分辨率的图，这样加下来，则会影响到上百K的包大

小，有人可能会说，这么点大小实在不值一提，产品用户量小的话确实影响不大，等产品用户量大到一定数量，这点包大小可能就会影响到产品各方面的数据，而这些数据影响到的也直接是真金白银，言归正传，我们要做的就是保证产品优势的条件下按照射鸡湿的要求进行实现，其实有条经验挺好使的，那就是：

看到一个效果之后，我们可以先别急着去做，可以先问射鸡湿在软件上的实现方式，射鸡湿们一般就实用PS，AE这样的设计软件，PS里面用图层进行叠图，叠图的时候又可能选择不同的叠加模式，比如做渐变会选择渐变模式，然后直接拉渐变线形成效果，通过知道他们的实现方式，很大程度上就能解决我们程序上的实现问题，因为里面绝大部分的方式android里面都有可用的api去进行实现，即使没有或存在版本兼容的问题，也相当于给我们提供了一种思路，让我们去想其他的解决方案：

好，我们还是回到上面的需求，要实现上面的效果假定提出了三个方案：

1. 直接由UX进行切图，包大小增大150K；
2. 使用纯色背景，然后在背景上放一张光的图片，进行叠加产生效果 一毕竟是两张图，叠加效果稍有不佳，并且同样增大包大小；
3. 纯代码实现对应设计效果，该方式有以上两个优点，即不需要切图，并且界面展示效果佳；

既然如此，我们就直接上第三种实现方式，通过与射鸡湿沟通，可以知道几个关键数据，首先渐变的起始点在图上最亮的地方，我们使用屏幕高的比例大概取14/15 处，然后中间共有四种颜色的渐变，我们可以直接让射鸡湿提供 对应的色值 和大概的位置，有了这些数据我们做出来的效果也就自然而然和她们在PS上做的效果保持高度的一致性，好了，直接上代码：

```
[html] C P
01. // 颜色数组
02. private static final int[] COLOR_BLUES = new int[] {
03.     0xff1db6ff, 0xff1db6ff, 0xff0b58c2, 0xff002a6d
04. };
05.
06. // 颜色对应的位置数组
07. private static final float[] COLOR_LOCATIONS = new float[] {
08.     0, 0.15f, 0.65f, 1f
09. };

[html] C P
01. mRadialGradient = new RadialGradient(mHalfWidth, mTotalHeight * SCREEN_HEIGHT_FRACTION,
02.     mTotalHeight * SCREEN_HEIGHT_FRACTION,
03.     mCurrentColors, COLOR_LOCATIONS, Shader.TileMode.MIRROR);
04. mLightGradientPaint.setShader(mRadialGradient);
```

同样还是画矩形：

```
[html] C P
01. canvas.drawRect(0, 0, mTotalWidth, mTotalHeight, mLightGradientPaint);
```

效果就不再贴了，如上；

这个时候，有些人可能就会说了，这样的需求比较简单，如果是这样的需求该如何做呢，页面是渐变的，然后在扫描，根据扫描的情况，前台展示的界面还需要动态的过渡到其他的渐变色（当前页面是渐变的，但总体偏蓝，需要过渡到偏红，而每时每刻单个界面也是渐变的），好了这个问题这里不说了，有兴趣的可以思考下，后面再抽空专门讲；

当然用RadialGradient 还可以做手点击即产生震荡展开的水波纹效果，大家可以考虑下，注意不要忽略了这三种不同的模式；

3. SweepGradient:扫描渐变渲染/梯度渲染：

直接看android 给我们提供的构造方法，一如即往，还是两个：

```
[html] C P
01. public SweepGradient(float cx, float cy, int color0, int color1) {}

[html] C P
01. public SweepGradient(float cx, float cy,
02.     int colors[], float positions[]) {}
```


从方法设计上基本和上面保持一致，中心点位置，然后就是起始点和终点颜色，如果需要多种颜色均匀或不均匀渐变，则采用传入数组的方式，颜色和比例进行对应，我们来写个小例子，还是用上面的几种颜色：

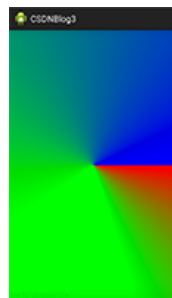
```
[html] C #
01. // 定义几种初始颜色
02. private static final int COLOR_RED = 0xffff0000;
03. private static final int COLOR_GREEN = 0xff00ff00;
04. private static final int COLOR_BLUE = 0xff0000ff;
05.
06. // 颜色数组
07. private int[] mColors = new int[] {
08.     COLOR_RED, COLOR_GREEN, COLOR_GREEN, COLOR_BLUE
09. };
10. // 与颜色数组对应的位置数组
11. private float[] mPositions = new float[] {
12.     0f, 0.2f, 0.4f, 1f
13. };

[html] C #
01. mGradientPaint.setShader(new SweepGradient(mHalfWidth, mHalfHeight, mColors, mPositions));
```

还是绘制全屏矩形：

```
[html] C #
01. canvas.drawRect(0, 0, mTotalWidth, mTotalHeight, mGradientPaint);
```

我们来看下效果：



大家可以看到，比例还是等于之前咱们算的 1:6:3，说明这几种渲染模式都是同样的计算规则，好了 SweepGradient (扫描渐变渲染/梯度渲染) 就说这么多；

大家还可以自己将绘制的矩形改为圆形，然后再让它旋转起来，咱们一起来看最后一种单独的渲染模式

4. BitmapShader (位图渲染)：

我们还是先看android给我们提供了什么方法可用：

```
[html] C #
01. public BitmapShader(Bitmap bitmap, TileMode tileX, TileMode tileY) {
```

分别是一张位图 和 横向，竖向的平铺模式；

我们假设现在背景是一个墙面，墙面一般都是同样的砖块，这时候我们就可以直接用一小部分砖块直接无缝生成整个界面，而不需切整个背景图；



如此同样的也可以一定减少包大小，优化运行时内存；

当然BitmapShader的用途还可以很多，这里就举这个栗子当作给大家开脑洞吧；

最后咱们一起来看看ComposeGradient (组合渲染)

5、ComposeGradient（组合渲染）：

顾名思义，也就是可以把多种渲染模式组合到一起进行绘制：

android 给我们提供了两个方法使用：

```
[html] C 2
01. public ComposeShader(Shader shaderA, Shader shaderB, PorterDuff.Mode mode) {}

[html] C 2
01. public ComposeShader(Shader shaderA, Shader shaderB, Xfermode mode) {
```

第一个我们在前面PorterDuffColorFilter里讲过，叠加模式（PorterDuff.Mode），第二个是后面即将要讲的图像的混合模式，这两种模式其实是非常类似的，另外两个参数也就是 两种渲染方式：

以叠加模式（PorterDuff.Mode）为例，讲个小例子：

QQ 或 很多应用里面都有圆形头像，怎么做呢？用shader做的话就非常简单，我们只需要绘制一个圆，只是把头像的bitmap作为BitmapShader 传到paint里即可；

原图如下：



好，接下来我们首先需要让他显示成圆形：

我们只需要以下简单的几步：

1. 创建位图
2. 将位图scale到目标大小
3. 创建BitmapShader并设置给paint
4. 使用该paint绘制目标大小的圆

```
[html] C 2
01. //创建位图
02. mBitmap = ((BitmapDrawable) getResources().getDrawable(R.drawable.touxiang)).getBitmap();
03. //将图scale成我们想要的大小
04. mBitmap = Bitmap.createScaledBitmap(mBitmap, mTotalWidth, mTotalWidth, false);
05.
06. // 创建位图渲染
07. BitmapShader bitmapShader = new BitmapShader(mBitmap, TileMode.REPEAT, TileMode.REPEAT);
08. // 将位图渲染设置给paint
09. mBitmapPaint.setShader(bitmapShader);
```

onDraw中：

```
[html] C 2
01. canvas.drawCircle(mHalfWidth, mHalfWidth, mHalfWidth, mBitmapPaint);
```

此时效果如下：



当然，我们的目标肯定不止于此，帅气逼人的脸庞必须主体突出，有虚有实，好，咱们让图片显示出外围虚化的效果，把帅脸显出来，该怎么做呢，其实只需要加一个以中心为全透明，外围为纯白的环形渐变（RadialGradient），代码上只需要做如下改变：

```
[html] C 2
01. // 创建位图渲染
02. BitmapShader bitmapShader = new BitmapShader(mBitmap, TileMode.REPEAT, TileMode.REPEAT);
03. // 创建环形渐变
04. RadialGradient radialGradient = new RadialGradient(mHalfWidth, mHalfWidth, mHalfWidth,
05.         Color.TRANSPARENT, Color.WHITE, TileMode.MIRROR);
06. // 创建组合渐变，由于直接按原样绘制就好，所以选择Mode.SRC_OVER
07. ComposeShader composeShader = new ComposeShader(bitmapShader, radialGradient,
08.         PorterDuff.Mode.SRC_OVER);
09. // 将组合渐变设置给paint
10. mBitmapPaint.setShader(composeShader);
```

我们再来看下效果：



当然，我们直接均匀渐变，会让渐变盖住一部分脸，让脸显得朦胧，如果我们需要不影响脸的展示该怎么做呢？创建颜色和位置数组把中心脸的部分露出来，我们再改改：先加两个数组记录颜色和位置：

```
[html] C 2
01. private static final int[] mColors = new int[] {
02.         Color.TRANSPARENT, Color.TRANSPARENT, Color.WHITE
03.     };
04. private static final float[] mPositions = new float[] {
05.         0, 0.6f, 1f
06.     };
```

然后修改环形渐变：

```
[html] C 2
01. // 创建环形渐变
02. RadialGradient radialGradient = new RadialGradient(mHalfWidth, mHalfWidth, mHalfWidth,
03.         mColors, mPositions, TileMode.MIRROR);
```

再看下效果：



好了，这篇文章就先写到这儿吧，实在写不动了，估计大家看的也够累，后面的重点主要有两个：

1. setXfermode(Xfermode xfermode) —— 图像混合模式

2. 文字的绘制

这两块后面都单独用一篇文章写吧，大家可以先看看后面都还有哪些方法：

`setShadowLayer(float radius, float dx, float dy, int color)`: 在图形下面设置阴影层，产生阴影效果，radius为阴影的角度，dx和dy为阴影在x轴和y轴上的距离，color为阴影的颜色；

`setStyle(Paint.Style style)`: 设置画笔的样式，为FILL, FILL_OR_STROKE, 或STROKE；

`setStrokeCap(Paint.Cap cap)`: 当画笔样式为STROKE或FILL_OR_STROKE时，设置笔刷的图形样式，如圆形样式；

Cap. ROUND, 或方形样式Cap. SQUARE

`setStrokeJoin(Paint.Join join)`: 设置绘制时各图形的结合方式，如平滑效果等；

`setStrokeWidth(float width)`: 当画笔样式为STROKE或FILL_OR_STROKE时，设置笔刷的粗细度；

`setXfermode(Xfermode xfermode)`: 设置图形重叠时的处理方式，如合并，取交集或并集；

Text文本绘制相关：

`setFakeBoldText(boolean fakeBoldText)`: 模拟实现粗体文字，设置在小字体上效果会非常差；

`setSubpixelText(boolean subpixelText)`: 设置该项为true，它可以保证在绘制斜线的时候使用抗锯齿效果来平滑该斜线的外观；

`setTextAlign(Paint.Align align)`: 设置绘制文字的对齐方向；

`setTextScaleX(float scaleX)`: 设置绘制文字x轴的缩放比例，可以实现文字的拉伸的效果；

`setTextSize(float textSize)`: 设置绘制文字的字号大小；

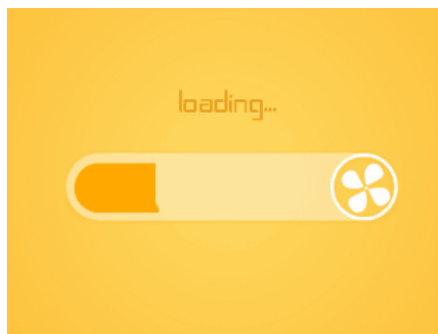
`setTextSkewX(float skewX)`: 设置斜体文字，skewX为倾斜弧度；

`setTypeface(Typeface typeface)`: 设置Typeface对象，即字体风格，包括粗体，斜体以及衬线体，非衬线体等；

`setUnderlineText(boolean underlineText)`: 设置带有下划线的文字效果；

`setStrikeThruText(boolean strikeThruText)`: 设置带有删除线的效果；

前两天看到一个比较酷炫的loading效果，下一篇文章跟大家分享一下：



我正在参加 CSDN 2015 博客之星评选 感恩分享活动，如果觉得文章还不错，请投个票鼓励下吧：<http://vote.blog.csdn.net/blogstar2015/candidate?username=tianjian4592>

顶

19

踩

0

[上一篇](#) 自定义view实现水波纹效果

下一篇 一个绚丽的loading动效分析与实现！

我的同类文章

android动效篇（12）

• Path特效之PathMeasure...

2015-07-26

阅读 6823

• Path相关方法讲解(一)

2015-06-15

阅读 5748

• Android 漂浮类动效的分...

2015-04-22

阅读 6034

• Android Paint之 setXferm...

2015-04-07

阅读 11541

• 一个绚丽的loading动效分...

2015-03-23

阅读 27358

• Path相关方法讲解(二)

2015-07-19

阅读 2937

• Canvas之translate、scal...

2015-05-07

阅读 7367

• Canvas开篇之drawBitma...

2015-04-14

阅读 17836

• Android使用SVG矢量图打...

2015-03-31

阅读 32832

• 自定义view实现水波纹效果

2015-03-12

猜你在找

Android之动画全讲	Android应用自定义View绘制方法手册
Android开发一屏幕适配	Android自定义绘制
Android高级界面控件难点精讲	一个炫字都不够手把手带你打造3D自定义view
Android事件处理重难点快速掌握	Android - 打造3D自定义view
Android5.0新特征详解(Material Design入门篇)	一个炫字都不够手把手带你打造3D自定义view

访问被拒绝---您没有访问此网站的权限，具体请咨询流程与信息管

查看评论

20楼 Mr_immortalZ 2016-05-03 15:41 发表



酷，总结得很棒！

19楼 迎风嘘嘘手插兜 2016-02-25 16:20 发表



画的圆比bitmap的内接圆还要小，只要加上内边距立马见光死。什么原因啊？

18楼 mr_gang1992 2016-01-21 17:03 发表



大神能不能加你为好友，，向请教你一些问题。。。方便留下QQ吗？？

Re: 学问积年而成 2016-01-21 17:40 发表



回复mr_gang1992: 1935400187

17楼 li-Struggle 2016-01-16 17:16 发表



写的好，谢谢楼主分享。

16楼 InsthyD 2015-10-29 21:12 发表



楼主。请教下那个用RadialGradient渲染靠外0.2白色边的圆形头像那个例子为啥mPositions =0, 0.6f, 1f 要三个啊。直接Color.TRANSPARENT, Color.WHITE : 0.6f, 1f不行吗。

15楼 shouhouhuakai 2015-10-27 16:00 发表



写的真好！

14楼 Encode_myself 2015-09-17 10:52 发表



厉害厉害

13楼 赵凯强 2015-09-17 10:22 发表



setFilterBitmap(boolean filter):如果该项设置为true，则图像在动画进行中会滤掉对Bitmap图像的优化操作，加快显示速度

这句话是错误的，设置为true，则图像在动画中时，会对图像进行滤波处理，从而优化图像的显示结果，避免锯齿和断层的出现，但是由于需要进行滤波计算，所以计算量大，并不会加快显示速度

12楼 虾虾的人生 2015-08-04 14:57发表



very nice !

11楼 sinat_28947593 2015-07-13 11:59发表



大神厉害！

10楼 liuhua_xing 2015-07-10 10:38发表



大神

9楼 tangzejin921 2015-04-22 11:15发表



好

8楼 Tiger_老虎 2015-04-13 19:09发表



楼主你这么耐心的写，而且那么详细，这么做到的，写这个要多久呀

7楼 HunterHubi 2015-04-13 15:03发表



写得非常好，谢谢博主分享

6楼 amaya25366707 2015-04-10 11:17发表



LZ高手! 求膜拜 \(*o*)/~

5楼 taothreeyears 2015-03-26 10:27发表



楼主高手

4楼 一叶飘舟 2015-03-24 22:49发表



果然是高手啊

3楼 IT_xiao小巫 2015-03-24 17:29发表



楼主对这部分api很熟悉啊，学习了

2楼 yongxinzhenxi 2015-03-23 23:58发表



楼主，我们做朋友吧~~~

Re: 学问积年而成 2015-03-24 10:57发表



回复yongxinzhenxi: 哈哈，这个可以有！

1楼 -琥珀川- 2015-03-23 21:02发表



阅

Re: 学问积年而成 2015-03-24 10:58发表



回复-琥珀川-: 复！

Re: -琥珀川- 2015-09-04 09:22发表



回复学问积年而成: 大神你好

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场


核心技术类目

- 全部主题
- Hadoop
- AWS
- 移动游戏
- Java
- Android
- iOS
- Swift
- 智能硬件
- Docker
- OpenStack
- VPN
- Spark
- ERP
- IE10
- Eclipse
- CRM
- JavaScript
- 数据库
- Ubuntu
- NFC
- WAP
- jQuery
- BI
- HTML5
- Spring
- Apache
- .NET
- API
- HTML
- SDK
- IIS
- Fedora
- XML
- LBS
- Unity
- Splashtop
- UML
- components
- Windows Mobile
- Rails
- QEMU
- KDE
- Cassandra
- CloudStack
- FTC
- coremail
- OPhone
- CouchBase
- 云计算
- iOS6
- Rackspace
- Web App

[SpringSide](#)[Maemo](#)[Compuware](#)[大数据](#)[aptech](#)[Perl](#)[Tornado](#)[Ruby](#)[Hibernate](#)[ThinkPHP](#)

[HBase](#)[Pure](#)[Solr](#)[Angular](#)[Cloud Foundry](#)[Redis](#)[Scala](#)[Django](#)[Bootstrap](#)

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [银行汇款帐号](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#) | [合作伙伴](#) | [论坛反馈](#)

[网站客服](#) [杂志客服](#) [微博客服](#) [webmaster@csdn.net](#) 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持
京 ICP 证 09002463 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved 

0