widget 英[ˈwɪdʒɪt]

美[ˈwɪdʒɪt]

n. 小器具，装饰品，窗口小部件;

# TASK

修改SearchView的下划线颜色

公共控件UI定制原理

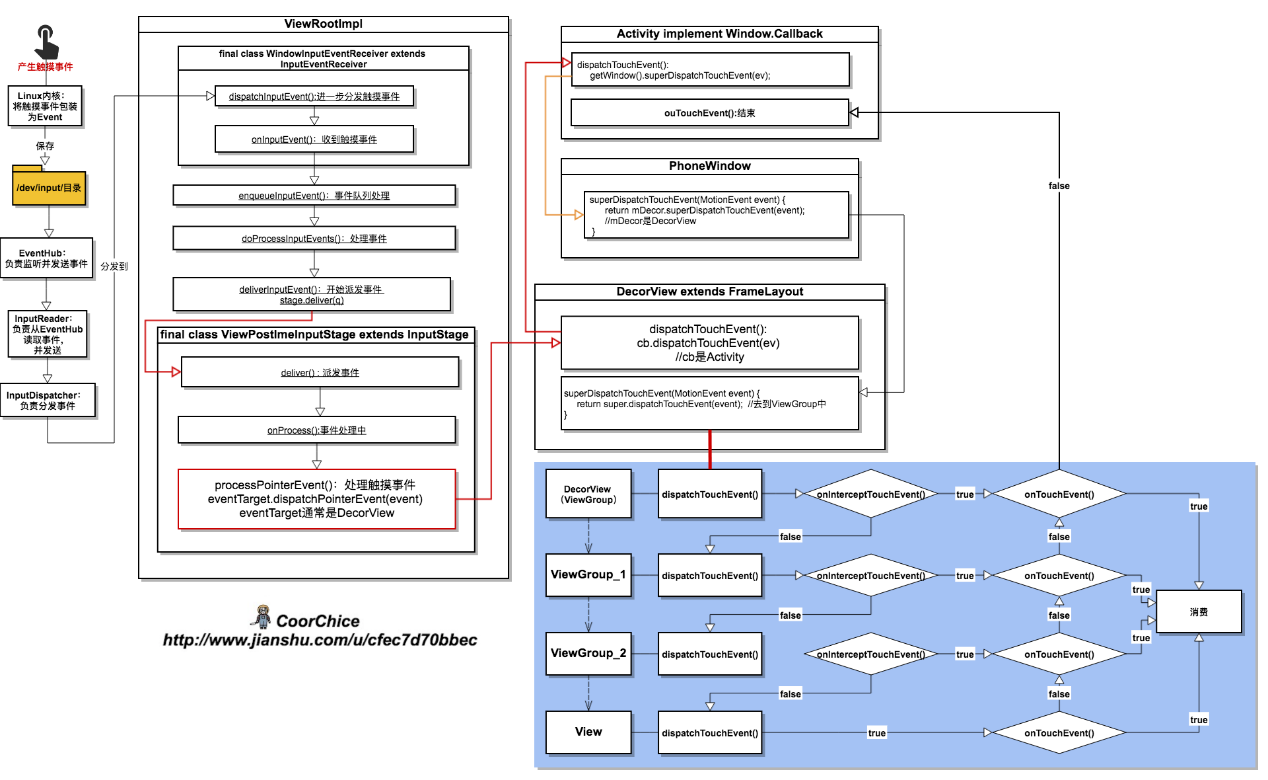
<https://blog.csdn.net/weixin_36039900/article/details/79409800>

<https://blog.csdn.net/ngl272/article/details/70199992>

# ViewRootImpl

ViewRootImpl就是ViewRoot，继承了ViewParent，但本身并不是一个View，可以看作是View树的管理者。而这里的成员变量mView就是DecorView，它指向的对象跟Window和Activity的mDecor指向的对象是同一个对象。所有的View组成了一个View树，每一个View都是树中的一个节点，如下图所示：

// frameworks/base/core/java/android/view/ViewRootImpl.java:447



## **WindowInputReceiver**

它继承了**InputEventReceiver**，并且是**ViewRootImpl**的一个内部类

final class WindowInputEventReceiver extends InputEventReceiver｛

@Override

public void onInputEvent(InputEvent event) ｛

...

enqueueInputEvent(event, this, 0, true);

...

｝

｝

当一个输入事件产生时（这里我们认为是触摸事件），会回调**InputEventReceiver.onInputEvent()**。从名字也可以看出，它是接收输入事件的。然后进一步调用 **ViewRootImpl.enqueueInputEvent()** 将输入事件加入单链表队列。

## **enqueueInputEvent**

void enqueueInputEvent(InputEvent event,

InputEventReceiver receiver,

int flags,

boolean processImmediately) {

...

mPendingInputEventTail = q;

//进行队列操作后，

//将ViewRootImpl的mPendingInputEventTail复制为新的触摸事件。

...

if (processImmediately) {

doProcessInputEvents();

//立即处理事件

} else {

scheduleProcessInputEvents();

//走一遍Handler延迟处理事件

}

}

上面这个方法主要对触摸事件进行队列操作(即排了个序)，然后再根据processImmediately参数，决定是立即处理，还是延后处理。

下面看看是如何进行处理的。

void doProcessInputEvents() {

while (mPendingInputEventHead != null) {

...

deliverInputEvent(q);

//进一步派发事件处理

...

}

}

在这个方法中有一个while{}循环体。它会将事件队列循环处理，直到队列中没有数据为

private void deliverInputEvent(QueuedInputEvent q) {

...

InputStage stage;

if (q.shouldSendToSynthesizer()) {

stage = mSyntheticInputStage;

} else {

stage = q.shouldSkipIme() ? mFirstPostImeInputStage : mFirstInputStage;

}

//上面决定将事件派发到那个InputStage中处理

if (stage != null) {

stage.deliver(q);

//派发事件到InputStage中处理

} else {

finishInputEvent(q);

}

}

# **InputStage**

**InputStage** 是在setView()的时候创建的，也就是在**Activity**的onResume()阶段

这些**InputStage**也相当于是单链表结构，一个套一个，比如调用了mFirstPostImeInputStage.deliver(q)。那么SyntheticInputStage, ViewPostImeInputStage, NativePostImeInputStage, EarlyPostImeInputStage都将能够处理这个触摸事件。这里我们主要看看ViewPostImeInputStage是如何处理的。

CharSequence counterSuffix = attrs.getTitle();  
mSyntheticInputStage = **new** SyntheticInputStage();  
InputStage viewPostImeStage = **new** ViewPostImeInputStage(mSyntheticInputStage);  
InputStage nativePostImeStage = **new** NativePostImeInputStage(viewPostImeStage,  
 **"aq:native-post-ime:"** + counterSuffix);  
InputStage earlyPostImeStage = **new** EarlyPostImeInputStage(nativePostImeStage);  
InputStage imeStage = **new** ImeInputStage(earlyPostImeStage,  
 **"aq:ime:"** + counterSuffix);  
InputStage viewPreImeStage = **new** ViewPreImeInputStage(imeStage);  
InputStage nativePreImeStage = **new** NativePreImeInputStage(viewPreImeStage,  
 **"aq:native-pre-ime:"** + counterSuffix);  
  
mFirstInputStage = nativePreImeStage;  
mFirstPostImeInputStage = earlyPostImeStage;

SyntheticInputStage<- viewPostImeStage<- nativePostImeStage <- **earlyPostImeStage** <- imeStage <- viewPreImeStage <- nativePreImeStage

## ViewPostImeInputStage

 合成的。

## setView

http://www.jcodecraeer.com/a/anzhuokaifa/androidkaifa/2016/0302/4025.html

<https://www.jianshu.com/p/b7cef3b3e703>

# Android 矢量图

## Why SVG

**可缩放矢量图形**（英语：**Scalable Vector Graphics**，**SVG**）是一种基于可扩展标记语言（XML），用于描述二维矢量图形的图形格式。SVG由W3C制定，是一个开放标准。——摘自[维基百科](https://link.jianshu.com?t=https:/zh.wikipedia.org/wiki/%E5%8F%AF%E7%B8%AE%E6%94%BE%E5%90%91%E9%87%8F%E5%9C%96%E5%BD%A2)

.svg格式相对于.jpg、.png甚至.webp具有较多优势，我认为核心有两点：

* **省时间**。图像与分辨率无关，收放自如，适配安卓机坑爹的分辨率真是一劳永逸；
* **省空间**。体积小，一般复杂图像也能在数KB搞定，图标更不在话下。

## VectorDrawable

VectorDrawable是Google从Android 5.0开始引入的一个新的Drawable子类，能够加载矢量图

Android中的VectorDrawable只支持SVG的部分属性，相当于阉割版。

它虽然是个类，但是一般通过配置xml再设置到要使用的控件上。在Android工程中，在资源文件夹res/drawable/的目录下（没有则需新建），通过<vector></vector>标签描述，例如svg\_ic\_arrow\_right.xml：

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

android:width="8dp"

android:height="8dp"

android:viewportHeight="24.0"

android:viewportWidth="24.0">

<path

android:fillColor="#ffffff"

android:pathData="M12,4l-1.41,1.41L16.17,11H4v2h12.17l-5.58,5.59L12,20l8,-8z"/>

</vector>

作者：woxingxiao  
链接：https://www.jianshu.com/p/0555b8c1d26a  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

何形式的转载都请联系作者获得授权并注明出处。

## 获得矢量图

### 图片资源编译工具Metro Studio

模板相当的多！支持SVG/PNG!



inker在线绘制矢量图的网站[http://app.inker.co](https://link.jianshu.com/?t=http://app.inker.co/)

svg收费的！

矢量编辑软件。比如：安装illustrator／sketch／Coreldraw/InDesign等。

Illustrator：要注册，试用一下，

Sketch只支持苹果的

### [iconfont](https://link.jianshu.com/?t=http://iconfont.cn)

前言部分已经墙裂推荐过，感觉我已经离不开它了=￣ω￣=

第一步，搜索你要的资源名字，中英文一般都会有结果。比如“arrow”

https://www.jianshu.com/p/0555b8c1d26a

### 标准svg 转->android svg

文件里好多标签Android是不认识的。不过没关系，有三种解决办法：

#### svg2VectorDrawable

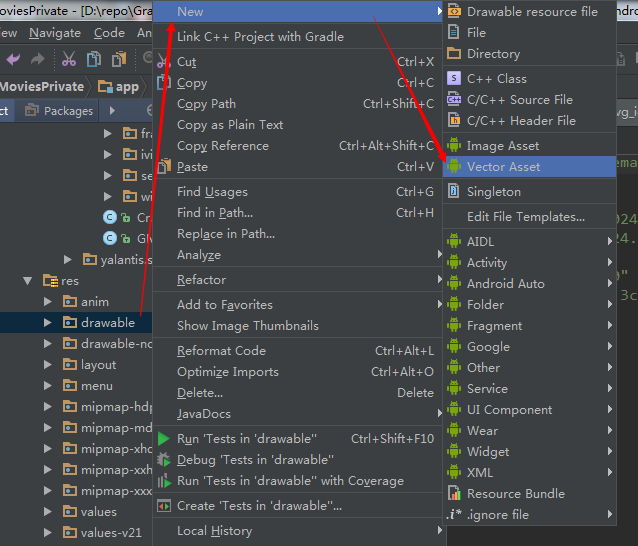
* 大招。先放大招，大招之下，后两种可以自动忽略。经大神[@天之界线2010](https://www.jianshu.com/u/denPHP)在评论区力荐的[svgtoandroid](https://link.jianshu.com?t=https:/github.com/misakuo/svgtoandroid)插件，用过之后果然神清气爽。安装：File -> Setting -> Plugins -> Browser repositories -> 搜“svg2VectorDrawable” -> 安装并重启Android Studio，再次进来后顶部工具栏会多一个图标：

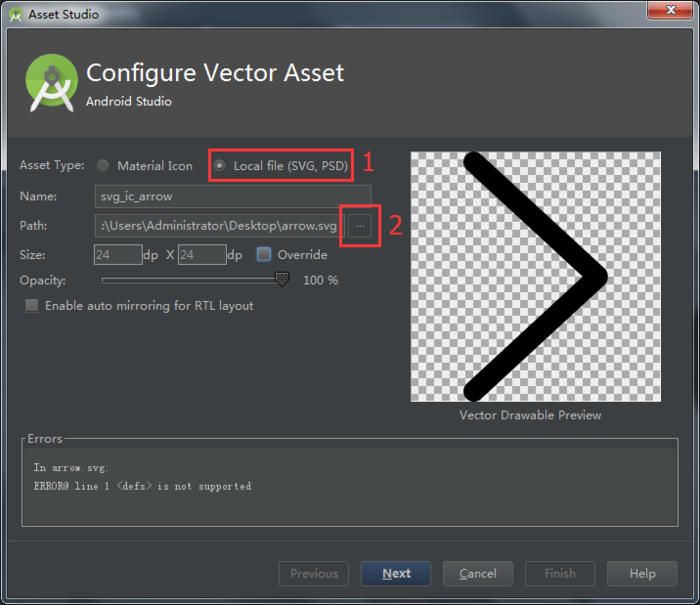
#### 手动编辑xml

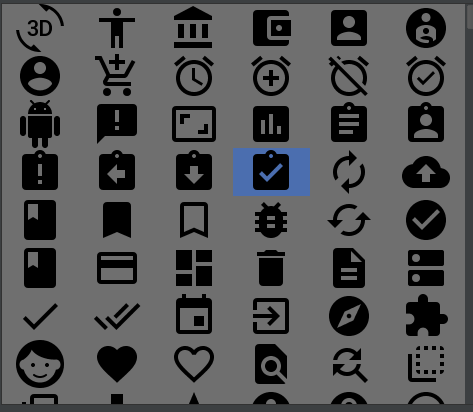
新建一个<vector></vector>标签的xml文件，通过观察文件内容，很容易获取到关键信息。width height自然对应<vector/>中宽高，viewBox后两位数字是分别对应<vector/>中的viewportWidth和viewportHeight，往下<path/>中的d的数据的对应<vector/>中<path/>中的pathData。fillColor自己手动设置。

#### 自带工具android studio

提供了丰富的图片资源，可以右键module,new->vector asset选择，如下







这个会主动展平层级结构

### 常见问题

#### 无法预览

有些时候，可以预览，有些时候不行，为啥呢？

但是可以运行

#### 小数点太长了

无法正常渲染

保持为整数就好了吧。。。

## vector 属性

android:name 定义该drawable的名字

android:width 定义该 drawable 的内部(intrinsic)宽度,支持所有 Android 系统支持的尺寸，通常使用 dp

android:height 定义该 drawable 的内部(intrinsic)高度,支持所有 Android 系统支持的尺寸，通常使用 dp

android:viewportWidth 定义矢量图视图的宽度，视图就是矢量图 path 路径数据所绘制的虚拟画布

android:viewportHeight 定义矢量图视图的高度，视图就是矢量图 path 路径数据所绘制的虚拟画布

android:tint 定义该 drawable 的 tint 颜色。默认是没有 tint 颜色的

android:tintMode 定义 tint 颜色的 Porter-Duff blending 模式，默认值为 src\_in

android:autoMirrored 设置当系统为 RTL (right-to-left) 布局的时候，是否自动镜像该图片。比如 阿拉伯语。

android:alpha 该图片的透明度属性

<group>里面可以定义多了<path>，这样可以方便管理多个<path>

## vector 标签 pathData 语法

Sdf

**[javascript]** [view plain](https://blog.csdn.net/qq1028850792/article/details/80287021) [copy](https://blog.csdn.net/qq1028850792/article/details/80287021)

1. <!--上面的手掌对应的代码实现-->
2. <vector xmlns:android="http://schemas.android.com/apk/res/android"
3. android:width="24dp"
4. android:height="24dp"
5. android:viewportWidth="24"
6. android:viewportHeight="24">
7. <path
8. android:fillColor="#000000"
9. android:pathData="
10. M22,23 q0,4 -4,4 h-7 q-2,0 -3,-1 T1,16 q-0.6,-0.8 0,-2 t5,3
11. q1,1 2,0 T8,4 q0,-1 0.9,-1.1 t1.1,1 1.5,9 q0.25,0.5 0.5,0.5
12. t0.5,-0.5 0,-11 q0.2,-1 1.1,-1.1 t1.1,1.1 1,11 q0.25,0.5 0.5,0.5
13. t0.5,-0.5 0.5,-9 q0.2,-1 1,-1 t1,1 0.5,9 q0.25,0.5 0.5,0.5
14. t0.5,-0.5 1.2,-6.5 q0.3,-1 1,-1 t0.8,1 -0.8,6 T22,23"/>
15. </vector>

### 基本语法

pathData 的指令基本都是由字母跟若干数字组成，数字之间可以用空格或者逗号隔开 (其实逗号会被忽略掉，加上逗号只是一些习惯的问题)。一般来说指令字母分为大小写两种,**大写的字母是基于原点的坐标系(偏移量)，即绝对位置；小写字母是基于当前点坐标系(偏移量)，即相对位置。**

**每个命令都有大小写形式，大写代表后面的参数是绝对坐标，小写表示相对坐标，相对于上一个点的位置。参数之间用空格或逗号隔开。**

### 移动M

move to 移动绘制点，作用相当于把画笔落在哪一点。

**M x,y (m dx, dy)** 移动虚拟画笔到对应的点，但是并不绘制。一开始的时候默认是在(0,0)。

### 直线L

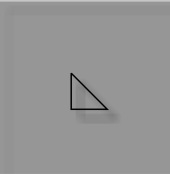
line to 直线，就是一条直线，**注意**，只是直线，直线是没有宽度的，所以你什么也看不到。

android:strokeColor="#333330" android:strokeWidth="10" 设置颜色和线宽

* **L x,y (l dx, dy)** 从当前点划一条直线到对应的点。
* **H x (h dx)** 从当前点绘制水平线，相当于l x,0
* **V y (v dy)** 从当前点绘制垂直线，相当于l 0,y

**[javascript]** [view plain](https://blog.csdn.net/qq1028850792/article/details/80287021) [copy](https://blog.csdn.net/qq1028850792/article/details/80287021)

1. <vector xmlns:android="http://schemas.android.com/apk/res/android"
2. android:width="24dp"
3. android:height="24dp"
4. android:viewportWidth="24"
5. android:viewportHeight="24">
6. <path
7. android:fillColor="#0000"
8. android:strokeColor="#000"
9. android:strokeWidth="0.2"
10. android:pathData=" M10,10 L10,15 L15,15 L10,10"/>
11. </vector>



将上述代码 android:pathData=" M10,10 L10,15 L15,15 L10,10" 替换成以下代码效果相同

android:pathData="M10,10 l 0,5 l 5,0 l-5,-5"

android:pathData="M10,10 V 15 H 15 L10,10"

android:pathData="M10,10 v 5 h 5 l-5,-5"

### 闭合Z

* **Z(或z)** 从结束点绘制一条直线到开始点，闭合路径

上面的图形型也可以由以下代码绘制  
android:pathData="M10,10 v 5 h 5 z"

### 弧线

ellipse 圆弧

* **A rx,ry x-axis-rotation large-arc-flag,sweepflag x,y**
* **a rx,ry x-axis-rotation large-arc-flag,sweepflag dx,dy**

rx ry 椭圆半径  
x-axis-rotation x轴旋转角度  
large-arc-flag 为0时表示取小弧度，1时取大弧度（要长的还是短的）  
sweep-flag 0取逆时针方向，1取顺时针方向  
x,y (dx,dy) 终点的位置

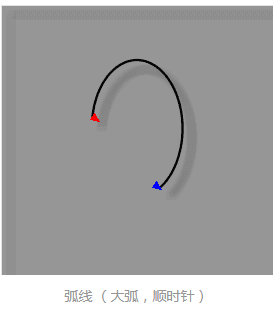
### 画圆

A（圆的x轴半径，y轴半径，x轴旋转角度，0/1-小/大弧度，0/1-逆/顺时针，圆的终点位置x点，圆的终点位置y点）

**Android 绘图和shape圆形**

**https://blog.csdn.net/linghu\_java/article/details/46403817**

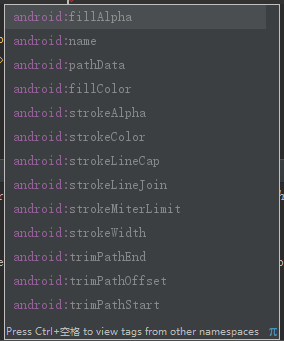
这个弧线的指令比起直线就相对复杂得多了，7个参数容易搞混了。来看个例子  
android:pathData="M8,10 a4,6 0 1,1 6 6"



[Android vector 标签 pathData 详解](https://www.jianshu.com/p/a3cb1e23c2c4)

https://www.cnblogs.com/yuhanghzsd/p/5466846.html

## path其他android属性



android:name 定义该 path 的名字，这样在其他地方可以通过名字来引用这个路径

android:pathData 和 SVG 中 d 元素一样的路径信息。

android:fillColor 定义填充路径的颜色，如果没有定义则不填充路径

android:strokeColor 定义如何绘制路径边框，如果没有定义则不显示边框

android:strokeWidth 定义路径边框的粗细尺寸

android:strokeAlpha 定义路径边框的透明度

android:fillAlpha 定义填充路径颜色的透明度

android:trimPathStart 从路径起始位置截断路径的比率，取值范围从 0 到1

android:trimPathEnd 从路径结束位置截断路径的比率，取值范围从 0 到1

android:trimPathOffset 设置路径截取的范围 Shift trim region (allows showed region to include the start and end), in the range from 0 to 1.

android:strokeLineCap 设置路径线帽的形状，取值为 butt, round, square.

android:strokeLineJoin 设置路径交界处的连接方式，取值为 miter,round,bevel.

android:strokeMiterLimit 设置斜角的上限，Sets the Miter limit for a stroked path.

## <group>

Fg

android:name 定义 group 的名字

android:rotation 定义该 group 的路径旋转多少度，这样图片就被旋转了，注意写数字的时候别晕了。

android:pivotX 定义缩放和旋转该 group 时候的 X 参考点。该值相对于 vector 的 viewport 值来指定的。

android:pivotY 定义缩放和旋转该 group 时候的 Y 参考点。该值相对于 vector 的 viewport 值来指定的。

android:scaleX 定义 X 轴的缩放倍数

android:scaleY 定义 Y 轴的缩放倍数

android:translateX 定义移动 X 轴的位移。相对于 vector 的 viewport 值来指定的。

android:translateY 定义移动 Y 轴的位移。相对于 vector 的 viewport 值来指定的。

<**vector xmlns:android="http://schemas.android.com/apk/res/android"  
 android:width="24dp"  
 android:height="24dp"  
 android:viewportWidth="72"  
 android:viewportHeight="72"**>  
 <**group  
 android:translateX="0.0"  
 android:translateY="42.0"**>  
 <**path  
 android:fillColor="?attr/wifi\_signal\_color"  
 android:pathData="M18.0,8.0l-1.0,0.0L17.0,6.0c0.0,-2.76 -2.24,-5.0 -5.0,-5.0S7.0,3.24 7.0,6.0l0.0,2.0L6.0,8.0c-1.1,0.0 -2.0,0.9 -2.0,2.0l0.0,10.0c0.0,1.0 0.9,2.0 2.0,2.0l12.0,0.0c1.1,0.0 2.0,-0.9 2.0,-2.0L20.0,10.0c0.0,-1.1 -0.9,-2.0 -2.0,-2.0zm-6.0,9.0c-1.1,0.0 -2.0,-0.9 -2.0,-2.0s0.9,-2.0 2.0,-2.0 2.0,0.9 2.0,2.0 -0.9,2.0 -2.0,2.0zm3.1,-9.0L8.9,8.0L8.9,6.0c0.0,-1.71 1.39,-3.1 3.1,-3.1 1.71,0.0 3.1,1.39 3.1,3.1l0.0,2.0z"**/>  
 </**group**>  
</**vector**>

## 性能问题

矢量图会有性能问题么，相比于png

<https://blog.csdn.net/a1961613299/article/details/50512729>

理论上讲，SVG的效率可能会不如PNG好，这是因为它需要运行时的计算和对应平台的渲染绘制。而且对于PNG来说的另一优势是在开启硬件加速的设备上，绘制Bitmap一个非常快速的过程。可以想象，让SVG不比PNG慢将是一件很有挑战的事情。

[微信的矢量化解决方案——WeChatSVGCode](https://blog.csdn.net/a1961613299/article/details/50512729)

# View

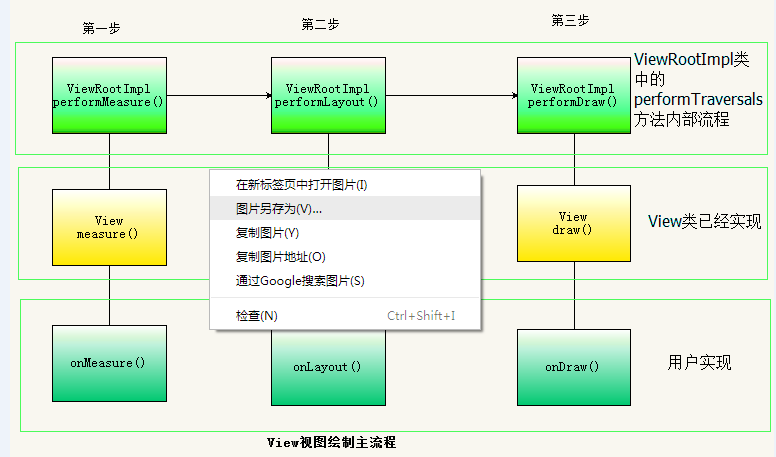
有个大家族，已经经历多代，族人角色可以这样定义：

* 成员：View
* 有子女的成员：ViewGroup
* 辈分最高的长老：DecorView

### 这里先给出Android系统View的绘制流程：

依次执行View类里面的如下三个方法：

* measure(int ,int) :测量View的大小
* layout(int ,int ,int ,int) ：设置子View的位置
* draw(Canvas) ：绘制View内容到Canvas画布上



### ViewRootImpl的主要作用如下（此处不多讲，如有意图，看源码）：

* A：链接WindowManager和DecorView的纽带，更广一点可以说是Window和View之间的纽带。
* B：完成View的绘制过程，包括measure、layout、draw过程。
* C：向DecorView分发收到的用户发起的event事件，如**按键**，**触屏**等事件。

在RootViewImpl中的函数通道是各种策略（**InputStage**）的组合，各策略负责的任务不同，如SyntheticInputStage、ViewPostImeInputStage、NativePostImeInputStage等等，这些策略以链表结构结构起来，当一个策略者没有消费事件时，就传递个下一个策略者。其中**触摸和按键事件由ViewPostImeInputStage**处理。

**DecorView则是PhoneWindow类的一个内部类，继承于FrameLayout，由此可知它是一个ViewGroup。**

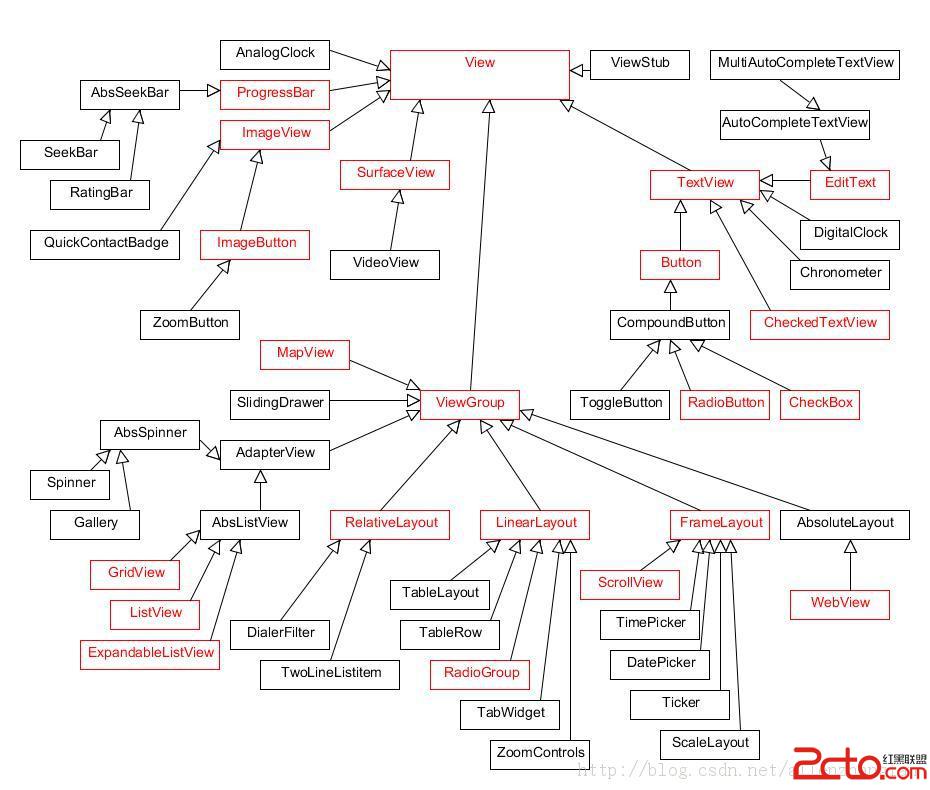
那么，DecroView到底充当了什么样的角色呢？

其实，**DecorView是整个ViewTree的最顶层View**，它是一个FrameLayout布局，代表了整个应用的界面。在该布局下面，有**标题view**和**内容view**这两个子元素。

作者：我爱大灿灿  
链接：https://www.jianshu.com/p/75a267486b44  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

作者：我爱大灿灿  
链接：https://www.jianshu.com/p/75a267486b44  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

## view及其子类的关系



# animated-selector

android5.0开始支持的动效了

android:state\_checked

android:state\_enabled=

<https://blog.csdn.net/qq_33689414/article/details/51712297>

# nine-patch

# tint

Tint的存在一定程度上减少了我们对图片的需求以及apk的大小，我们拿ImageView来说吧，假如它的背景图有两种，一种是默认情况下需要显示的是背景图片1，另一种是是在触摸模式下单击时需要显示的是背景图片2。一般情况下背景图片1和背景图片2之间除了颜色不一样，其他都一样的。我们之前的做法也许会找UI要这样两张颜色不一样的图片，但是如果我们使用Tint的话，一张矢量图是能适配所有的颜色

我们仅需要设置tint这个属性即可，android:tint=”@color”来改变颜色

## 改变EditText的背景色以及光标的颜色

https://blog.csdn.net/qq\_20785431/article/details/52106012

## tintMode

https://blog.csdn.net/zenglanjing/article/details/54603242

Android Tint的使用及源码分析

<https://blog.csdn.net/u013085697/article/details/70055621>

# EditText

**public** EditText(Context context, AttributeSet attrs) {  
 **this**(context, attrs, com.android.internal.R.attr.***editTextStyle***);  
}

# **水波纹效果**

**<ripple>是API 21才有的新Tag,正是实现水波纹效果的;**

 其中<ripple android:color="#FF21272B" .... />这个是指定水波纹的颜色.

    而<item />里面的东西,我们都很熟悉,就是普通的定义一个带圆角的背景.

ripple\_bg.xml:

**[html]** [view plain](https://blog.csdn.net/cxc19890214/article/details/51605719) [copy](https://blog.csdn.net/cxc19890214/article/details/51605719)

1. **<?xml** version="1.0" encoding="utf-8"**?>**
2. **<ripple** xmlns:android="http://schemas.android.com/apk/res/android"
3. android:color="#FF21272B"**>**
4. **<item>**
5. **<shape** android:shape="rectangle"**>**
6. **<solid** android:color="#FFFFFF" **/>**
7. **<corners** android:radius="4dp" **/>**
8. **</shape>**
9. **</item>**
10. **<item** android:drawable="@drawable/rounded\_corners" **/>**
11. **</ripple>**

# Switch

## switchStyle

**public** Switch(Context context, AttributeSet attrs) {  
 **this**(context, attrs, com.android.internal.R.attr.***switchStyle***);  
}

## Switch\_track

**final** TypedArray a = context.obtainStyledAttributes(  
 attrs, com.android.internal.R.styleable.***Switch***, defStyleAttr, defStyleRes);

**mTrackDrawable** = a.getDrawable(com.android.internal.R.styleable.***Switch\_track***);

# Spinner

<**style name="Widget.Spinner"**>  
 <**item name="background"**>@drawable/btn\_dropdown</**item**>  
 <**item name="clickable"**>true</**item**>  
 <**item name="spinnerMode"**>dialog</**item**>  
  
 <**item name="dropDownSelector"**>@drawable/list\_selector\_background</**item**>  
 <**item name="popupBackground"**>@drawable/spinner\_dropdown\_background</**item**>  
 <**item name="dropDownVerticalOffset"**>-10dip</**item**>  
 <**item name="dropDownHorizontalOffset"**>0dip</**item**>  
 <**item name="dropDownWidth"**>wrap\_content</**item**>  
 <**item name="popupPromptView"**>@layout/simple\_dropdown\_hint</**item**>  
 <**item name="gravity"**>center</**item**>  
</**style**>

<**style name="Widget.Material.Spinner" parent="Widget.Spinner.DropDown"**>  
 <**item name="background"**>@drawable/spinner\_background\_material</**item**>  
 <**item name="dropDownSelector"**>?attr/listChoiceBackgroundIndicator</**item**>  
 <**item name="popupBackground"**>@drawable/popup\_background\_material</**item**>  
 <**item name="popupElevation"**>@dimen/floating\_window\_z</**item**>  
 <**item name="popupAnimationStyle"**>@empty</**item**>  
 <**item name="popupEnterTransition"**>@transition/popup\_window\_enter</**item**>  
 <**item name="popupExitTransition"**>@transition/popup\_window\_exit</**item**>  
 <**item name="dropDownVerticalOffset"**>0dip</**item**>  
 <**item name="dropDownHorizontalOffset"**>0dip</**item**>  
 <**item name="overlapAnchor"**>true</**item**>  
 <**item name="dropDownWidth"**>wrap\_content</**item**>  
 <**item name="popupPromptView"**>@layout/simple\_dropdown\_hint</**item**>  
 <**item name="gravity"**>start|center\_vertical</**item**>  
 <**item name="disableChildrenWhenDisabled"**>true</**item**>  
</**style**>

## popupPromptView

<**item name="popupPromptView"**>@layout/simple\_dropdown\_hint</**item**>

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

android:id="@android:id/text1"

android:textAppearance="?android:attr/dropDownHintAppearance"

android:singleLine="true"

android:layout\_marginStart="3dip"

android:layout\_marginTop="3dip"

android:layout\_marginEnd="3dip"

android:layout\_marginBottom="3dip"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

[android](https://www.pocketdigi.com/tag/android).R.layout.simple\_spinner\_item

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

android:id="@android:id/text1"

style="?android:attr/spinnerItemStyle"

android:singleLine="true"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:ellipsize="marquee"

android:textAlignment="inherit"/>

spinnerItemStyle

# checkbox

<item name="listChoiceIndicatorMultiple">@drawable/btn\_check\_material\_anim</item>

btn\_check\_material\_anim.xml

btn\_checkbox\_checked\_mtrl.xml

# Alert

# Android动画深入分析

Android的动画可以分成三种，view动画，帧动画，还有属性动画，其实帧动画也是属于view动画的一种，，只不过他和传统的平移之类的动画不太一样的是表现形式上有点不一样，view动画是通过对场景的不断图像交换而产生的动画效果，而帧动画就是播放一大段图片，很显然，图片多了会OOM，属性动画通过动态的改变对象的属性达到动画效果，也是api11的新特性，在低版本无法使用属性动画，但是我们依旧有一些兼容库，OK，我们还是继续来看下详细的类别

## View动画

View动画的作用是view，他支持四种动画，平移，缩放，旋转和透明，帧动画算是特殊的第五种了，我们接下来就一个个的去分析

### View动画的种类

数学含义应该是几何变化、图形变化：位置和像素值的变化

view动画的变换效果对应着Animation的四个子类，分别是TranslateAnimation,ScaleAnimation,RotateAnimation,AlphaAnimation,这四种动画可以通过xml来定义，也可以代码来实现，如图



要使用view动画，首先要创建XML文件，这个文件的路径为res/anim/filename.xml，他的描述语法是固定的，我们来看下:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <set xmlns:android="http://schemas.android.com/apk/res/android"  android:shareInterpolator="true">  <alpha  android:fromAlpha="10.0"  android:toAlpha="10.0" />  <scale  android:fromXScale="10dp"  android:fromYScale="10dp"  android:pivotX="10"  android:pivotY="10"  android:toXScale="10dp"  android:toYScale="10dp" />  <translate  android:fromXDelta="10"  android:fromYDelta="10"  android:toXDelta="10"  android:toYDelta="10" />  <rotate  android:fromDegrees="10"  android:pivotX="10"  android:pivotY="10"  android:toDegrees="10" />  </set> |

从上面的语法可以看出，view的动画既可以是单一的动画，也可以组合在一起，而set标签就是组合动画，对应着AnimationSet,他包含很多个类。我们来看下他的两个属性

**android:interpolator**表示动画集合所使用的插值器，插值器影响动画的速度，比如非匀速动画就需要插值器来制作动画的过程，这个属性可以不指定，默认为加速减速插值器，这个会在后面详细描述

**android:shareInterpolator**表示集合中的动画是否和集合共享同一个插值器，如果集合不指定插值器，那么子动画就需要单独的去指定所需要的插值器了

**< translate>**标签表示平移动画，对应的TranslateAnimation类，他可以使用一个view在水平和竖直方向完成平移，我们来看下他的属性

android:fromXDelta表示x的起始值，比如0

android:fromYDelta表示y的结束值，比如100

android:toXDelta表示x的起始值

android:toYDelta表示y的结束值

**< scale>**标签表示的是缩放动画，对应的ScaleAnimation，他可以使view具有放大，缩小的动画效果，他的一系列属性的含义如下

android:fromXScale水平方向缩放的起始值，比如0.5

android:fromYScale竖直方向缩放的起始值

android:pivotX缩放轴点的x坐标，它会影响缩放的效果

android:pivotY缩放轴点的y坐标，它会影响缩放的效果

android:toXScale水平方向缩放的结束值，比如1.2

android:toYScale竖直方向缩放的起始值

在< scale>中提到轴点的这个概念，我举个例子，默认情况下轴点事view的中心点，这个时候水平缩放的话会导致view向左右两个方向进行缩放，但是如果把轴点设置在view的右边界，那么view只会向左缩放，反之，xxx

**rotate**标签是旋转标签，对应RotateAnimation，他可以让view旋转，他的属性如下

android:fromDegrees旋转开始的角度，比如0

android:toDegrees旋转结束的角度，比如180

android:pivotX旋转轴点的x

android:pivotY旋转轴点的y

在旋转中也有轴的概念，他也会影响到旋转的效果，轴点扮演者旋转轴的角色，view围绕着轴点进行旋转，默认情况下在view的中心，考虑到一种情况，view围绕自己的中心，和围绕左上角进行90度显然是不同的轨迹

**< alpha>**表示透明动画。对应的AlphaAnimation，我们看下属性

android:fromAlpha表示透明度的起始值，比如0.1

android:toAlpha表示透明度的结束值，比如1

上面都只是很简单的介绍了XML格式，具体的使用方法还是看文档，我们还有一些常用的属性如下

**android:duration**动画的时间

**android:fillAfter**动画结束之后是否停留在结束的位置

下面我们再来一个实际的例子

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <set xmlns:android="http://schemas.android.com/apk/res/android"  android:fillAfter="true"  android:zAdjustment="normal">  <translate  android:duration="2000"  android:fromXDelta="0"  android:fromYDelta="0"  android:toXDelta="100"  android:toYDelta="100" />  <rotate  android:duration="4000"  android:fromDegrees="0"  android:toDegrees="90" />  </set> |

|  |
| --- |
| private void test1() {  Animation animation = AnimationUtils.loadAnimation(this, R.anim.test1);  mTestAn1View.startAnimation(animation);  } |

除了在XML中定义之外，我们还可以通过代码来使用，如下

|  |
| --- |
| private void test2() {  AlphaAnimation animation = new AlphaAnimation(0,1);  animation.setDuration(4\*1000);  mTestAn1View.startAnimation(animation);  } |

我们还可以对动画进行监听，来看下代码

|  |
| --- |
| alpha.setAnimationListener(new Animation.AnimationListener() {  @Override  public void onAnimationStart(Animation animation) {  Log.i("TAG", "动画开始");  }  @Override  public void onAnimationEnd(Animation animation) {  Log.i("TAG", "动画结束");  }  @Override  public void onAnimationRepeat(Animation animation) {  Log.i("TAG", "重复动画");  }  }); |

### 自定义View动画

除了系统提供的四种动画以外，我们还可以自定义动画，自定义是一种简单又复杂的工作，说他简单，是因为派生出来只需要继承animation这个抽象类，这里也不打算去讲讲，直接看APIdemo里面3D效果的源码来分析:

|  |
| --- |
| public class Rotate3dAnimation extends Animation {  private final float mFromDegrees;  private final float mToDegrees;  private final float mCenterX;  private final float mCenterY;  private final float mDepthZ;  private final boolean mReverse;  private Camera mCamera;  /\*\*  \* Creates a new 3D rotation on the Y axis. The rotation is defined by its  \* start angle and its end angle. Both angles are in degrees. The rotation  \* is performed around a center point on the 2D space, definied by a pair  \* of X and Y coordinates, called centerX and centerY. When the animation  \* starts, a translation on the Z axis (depth) is performed. The length  \* of the translation can be specified, as well as whether the translation  \* should be reversed in time.  \*  \* @param fromDegrees the start angle of the 3D rotation  \* @param toDegrees the end angle of the 3D rotation  \* @param centerX the X center of the 3D rotation  \* @param centerY the Y center of the 3D rotation  \* @param reverse true if the translation should be reversed, false otherwise  \*/  public Rotate3dAnimation(float fromDegrees, float toDegrees,  float centerX, float centerY, float depthZ, boolean reverse) {  mFromDegrees = fromDegrees;  mToDegrees = toDegrees;  mCenterX = centerX;  mCenterY = centerY;  mDepthZ = depthZ;  mReverse = reverse;  }  @Override  public void initialize(int width, int height, int parentWidth, int parentHeight) {  super.initialize(width, height, parentWidth, parentHeight);  mCamera = new Camera();  }  @Override  protected void applyTransformation(float interpolatedTime, Transformation t) {  final float fromDegrees = mFromDegrees;  float degrees = fromDegrees + ((mToDegrees - fromDegrees) \* interpolatedTime);  final float centerX = mCenterX;  final float centerY = mCenterY;  final Camera camera = mCamera;  final Matrix matrix = t.getMatrix();  camera.save();  if (mReverse) {  camera.translate(0.0f, 0.0f, mDepthZ \* interpolatedTime);  } else {  camera.translate(0.0f, 0.0f, mDepthZ \* (1.0f - interpolatedTime));  }  camera.rotateY(degrees);  camera.getMatrix(matrix);  camera.restore();  matrix.preTranslate(-centerX, -centerY);  matrix.postTranslate(centerX, centerY);  }  } |

### 帧动画

帧动画就是顺序的播放一组图片，系统提供了一个AnimationDrawable来实现帧动画，帧动画的使用比较简单，我们看下在xml中如何定义

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <animation-list xmlns:android="http://schemas.android.com/apk/res/android"  android:oneshot="false">  <item  android:drawable="@drawable/ic\_dashboard\_black\_24dp"  android:duration="200" />  <item  android:drawable="@drawable/ic\_dashboard\_black\_24dp"  android:duration="200" />  <item  android:drawable="@drawable/ic\_dashboard\_black\_24dp"  android:duration="200" />  </animation-list> |

然后将上述的XML作为view的背景并通过drawable来播放就可以了

|  |
| --- |
| iv\_icon.setBackgroundResource(R.drawable.animation\_test2);  AnimationDrawable ad = (AnimationDrawable) iv\_icon.getBackground();  ad.start(); |

帧动画比较简单，但是容易OOM，这个要注意

## View动画的特殊使用场景

动画的应用场景很多，我们来看下，比如activity的切换动画，又比如viewgroup的子view切换动画等，一起来看下吧

### LayoutAnimation

LayoutAnimation作用于**ViewGroup**，为ViewGroup指定一个动画，这样他的子元素出场的时候就会具有这种动画了，这种效果常常在listview上，我们时常会看到一种特殊的listview，他的每一个item都有一个动画，其实这并不是什么高深的技术，它使用的就是LayoutAnimation，我们来看下具体的实现。

#### 定义LayoutAnimation

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <layoutAnimation xmlns:android="http://schemas.android.com/apk/res/android"  android:animation="@anim/animation"  android:animationOrder="normal"  android:delay="0.5">  </layoutAnimation> |

他们的含义分别是

**android:delay**表示子元素开始动画的延迟，假设子元素入场动画的周期为300ms，那么0.5表示每一个子元素都需要延迟150ms才能播放入场动画，总体来说，第一个子元素延迟150ms，第二个子元素300ms以此类推

**android:animationOrder**表示子元素动画的顺序，有三种模式，normal，random,reverse,其中第一个表示顺序执行，第二个表示随机，第三个表示倒叙执行

**android:animation**为子元素指定动画，如下

#### 为子元素指定的入场动画

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <set xmlns:android="http://schemas.android.com/apk/res/android"  android:duration="300"  android:shareInterpolator="true">  <alpha  android:fromAlpha="0.1"  android:toAlpha="1.0" />  <translate  android:fromXDelta="500"  android:fromYDelta="0" />  </set> |

#### 为ViewGroup指定layoutanimation属性

对于listview来说，这样item就具有出场动画了

|  |
| --- |
| <ListView  android:id="@+id/mListView"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:layoutAnimation="@anim/anim\_layout" /> |

除了在XML中实现，我们还可以通过LayoutAnimationController来实现

|  |
| --- |
| Animation animation = AnimationUtils.loadAnimation(this,R.anim.anim\_layout);  LayoutAnimationController controller = new LayoutAnimationController(animation);  controller.setDelay(0.5f);  controller.setOrder(LayoutAnimationController.ORDER\_NORMAL);  mListView.setLayoutAnimation(controller); |

### Activity的切换效果

activity有默认的切换效果，但是这个效果我们是可以自定义的，主要是利用overridePendingTransition(int enterAnim, int exitAnim)这个方法，这个方法必须在startactivity或者finish之后调用才是有效的，里面的两个参数也很简单，就是进出的动画，让我们启动一个activity的时候，我们可以这样做

|  |
| --- |
| startActivity(new Intent(MainActivity.this,OneActivity.class));  //这是activity的跳转动画  overridePendingTransition(R.anim.ch7\_animation\_enter,R.anim.ch7\_animation\_back); |

当activity退出的时候我们也可以为他指定一个切换效果

|  |
| --- |
| @Override  public void onBackPressed() {  super.onBackPressed();  overridePendingTransition(R.anim.ch7\_animation\_enter,R.anim.ch7\_animation\_back);  } |

使用它只需要在后面调用，这个也是注意的地方，其他地方调用无效.

在fragment中也是可以的，使用的方法是通过FragmentTransaction中的setCustomAnimations去使用.

## 属性动画

属性动画是API11加入的，和view动画不同的是，他对作用对象进行了扩展，属性动画可以对任何对象做动画，甚至没有对象也是可以的，除了作用对象进行了扩展以外，属性动画的效果也得到了加强，不再像view动画一样只支持四种，属性动画有ValueAnimator,ObjectAnimator,AnimatorSet;

### 使用属性动画

属性动画可以对任意**对象的属性**进行动画而不仅仅是view，动画默认的时间间隔是300ms，默认帧率是10，可以达到的效果是： 在一个时间间隔内完成对象从一个属性值到另一个属性值的改变。

为何叫属性动画，这里有个公式化的解释：ObjectAnimator.of$1(mView, "$2",value…); mView对mView的set$2($1 arg)方法执行动画，这里的方法改变的View的属性值。比如ObjectAnimator.ofFloat(iv\_icon, "translationY",50)，对iv\_icon的setTranslationY（float arg）方法执行平移动画，平移量为50。不用担心属性值填错，ide会自动纠错。属性动画要求对象的属性有set get方法。

但是属性动画是从API11才开始有的，所以这个比较有限制性。当然，网上还是有很多的就兼容库，这个我们就不多说，我们举几个例子：

(1)改变一个对象的translationY属性，让其沿着Y轴向上平移一个时间，该动画在默认的时间完成，我们来看下怎么去用,[demo](http://img.blog.csdn.net/20170422184616225?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvcXFfMjY3ODcxMTU=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast):

ObjectAnimator.ofFloat(mOV,"translationY",50).start();

(2)改变一个对象的背景颜色值，典型的就是改变view的背景，下面的动画是让view的背景从0xffff8080到0xff8080ff,动画会无限循环和反转,[demo](http://img.blog.csdn.net/20170422184728352?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)

|  |
| --- |
| ValueAnimator valueAnimator = ObjectAnimator.ofInt(mVV,"backgroundColor",0xFFFF8080, 0xFF8080FF);  valueAnimator.setDuration(3\*1000);  valueAnimator.setEvaluator(new ArgbEvaluator());  valueAnimator.setRepeatCount(ValueAnimator.INFINITE);  valueAnimator.setRepeatMode(ValueAnimator.REVERSE);  valueAnimator.start(); |

(3)动画集合，10s内对view的旋转，平移，缩放和透明度进行改变，注意此处是AnimatorSet,不是AnimationSet ,[demo](http://img.blog.csdn.net/20170422184811248?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvcXFfMjY3ODcxMTU=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast)

|  |
| --- |
| AnimatorSet set = new AnimatorSet();  set.playTogether(  ObjectAnimator.ofFloat(mSV,"rotationX",0,360)  ,ObjectAnimator.ofFloat(mSV,"rotationY",0,90)  ,ObjectAnimator.ofFloat(mSV,"rotation",0,-90)  ,ObjectAnimator.ofFloat(mSV,"translationX",90)  ,ObjectAnimator.ofFloat(mSV,"scaleX",0,5f)  ,ObjectAnimator.ofFloat(mSV,"alpha",0,0.25f,1));  set.setDuration(10\*1000);  set.start(); |

属性动画还可以用XML来表示的，在res/animator目录下

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <set xmlns:android="http://schemas.android.com/apk/res/android">  <objectAnimator  android:duration="1000"  android:propertyName="@string/app\_name"  android:repeatCount="infinite"  android:repeatMode="restart"  android:startOffset="15"  android:valueFrom="0.1dp"  android:valueTo="1.0"  android:valueType="colorType" />  <animator  android:duration="1000"  android:repeatCount="infinite"  android:repeatMode="restart"  android:startOffset="15"  android:valueFrom="0.5dp"  android:valueTo="1.0dp"  android:valueType="colorType" />  </set> |

属性动画的各个参数是比较好理解的，我们简单来说下他们之间的含义

android:propertyName:表示属性动画作用对象的属性的名称

android:duration：表示动画的时长

android:valueFrom：表示属性的起始值

android:valueTo：表示属性的结束值

android:startOffset：表示动画的延迟时间，当动画开始后，需要延迟多少毫秒才会真正的播放

android:repeatCount：表示动画的重复次数

android:repeatMode：表示动画的重复模式

android:valueType:表示propertyName有两个属性有int和float两个可选项，分别表示属性的类型，和浮点型，另外，如果所制定的是颜色类型，那么就不需要指定propertyName，系统会自动对颜色类型进行处理

对于一个动画来说，有两个属性这里要特别注意一下，一个是count另一个就是mode

下面我们给出一个具体的事例来分析，我们通过XML定义属性动画并且作用在view上

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <set xmlns:android="http://schemas.android.com/apk/res/android">  <objectAnimator  android:duration="300"  android:propertyName="x"  android:valueTo="200"  android:valueType="intType" />  <objectAnimator  android:duration="300"  android:propertyName="y"  android:valueTo="300"  android:valueType="intType" />  </set> |

那我们该如何使用呢？

|  |
| --- |
| AnimatorSet sets = (AnimatorSet) AnimatorInflater.loadAnimator(MainActivity.this, R.animator.property\_animator);  sets.setTarget(iv\_icon);  sets.start(); |

在实际开发当中我还是建议使用**代码来实现属性动画**，这是因为用代码会比较简单，比如一个view需要从左边移动到右边，但是如果是XML，是不知道屏幕的宽高的

### 理解插值器和估值器

TimeInterpolator中文翻译是时间插值器的意思，他的作用是根据时间流逝的百分比来计算出当前属性值改变的百分比，系统预设的是LinearInterpolator(线性加速器，匀速加速器)，加速和减速插值器，TypeEvaluator的中文翻译是类型估值算法，也叫估值器，他的作用是根据当前属性变化的百分比来计算变化后的属性值，系统也预设了针对整型属性，浮点型，和color颜色值，定义的其实上是一个时刻-属性值的函数映射关系。

属性动画中的插值器和估值器很重要，他们实现非匀速动画的重要手段，可能说的有点苦涩，我们去通过实际的例子就能很好的理解了。如图，他表示的是一个匀速动画，采用了线性插值器和整形估值算法，在40ms内，x从0-40的变换。



由于动画的默认刷新率为10ms/帧，所有该动画将分5帧进行，我们来考虑一下第三帧，当时间为20ms的时候，百分比为0.5，意味着时间过去了一半，那x改变了多少？其实x也是0.5，为什么因为他使用了线性的插值器也就是匀速动画。

很显然，线性插值器的返回值和输入值是一样的，因为都是0.5，这个估算我们可以看他的源码



IntEvaluator.java中，对于我们的例子而言分别是0.5 0 40 ，所有返回给我们的是20，这就是（x= 20 t = 20ms）的由来。



插值器和估值器除了系统提供给我们的外，我们还可自定义，实现起来也很简单，因为他们都只是一个接口，而且内部只有一个方法。

### 属性动画的监听

属性动画提供了监听器用于监听动画的播放过程，主要有两个接口AnimationUpdateListener和AnimationListener

AnimationListener监听了开始，结束，取消已经重复。

AnimatorListenerAdapter监听整个动画过程。

### 对任意属性做动画

这里最一个需求，就是给buttion设置一个动画，让他的宽度从当前的变成500px，这个可以用view动画来搞定，但是你仔细想想，view不能对宽高变化，所有我么可以使用属性动画，我们来试试

ObjectAnimator.ofInt(mBtn,"width",300).setDuration(10\*1000).start();

不work，为何呢？我们 先看原理。

#### 属性动画的使用条件

属性动画要求动画作用在对象提供的get/set方法，属性动画根据外界传递的该属性的初始值和最终值，以动画效果多次去set，每次传递的set方法的值都不一样，确切来说是随着时间的时间推移，所传递的值越来越接近最终值，总结一下，我们对object的属性abc做动画，如果想让动画生效，要同时满足两个条件：

(1)object必须要提供set方法，如果动画的时候没有传递初始值，那么我们还要提供get方法，因为系统要去取abc的属性(如果这条不满意，程序直接Crash)

(2)object的set方法对abc所做的改变必须通过某种方法反应，比如带来UI的改变(如果这条不满足，动画无效果但是不会Crash)

以上条件缺一不可，那么为什么我们对button的width属性做动画没有效果，这是因为button内部虽然提供了get/set方法，但是这个set方法并不是改变视图大小，他是textview新添加的方法，view是没有这个setWidth方法的，由于button继承了textview，所有button也就有了set方法，下面看一下这个get/set的源码





从上述的源码可以看出，get的确是获取view的宽度，而set是textview的专属方法，他的作用不是设置view的宽度，而是设置textview的最大宽度和最小宽度，这个和textview的宽度不死一个东西，具体来说，textview的宽度对应XML中的android:layout\_width，而textview还有一个属性android:width，这个就对应了setwidth，总之textview和button的set/get干的不是同一件事，通过set无法改变控件的宽度，所以对width做属性动画没有效果，对于属性动画的两个条件来说，本例中的动画只满足了第一个条件，我们有三个解决办法：

* 给你的对象增加set/get方法，前提是你有权限的话
* 用这个类来包装原始对象，间接提供get/set方法
* 采用ValueAnimator，监听动画过程自己去实现

#### 解决办法

我们来具体的实现下这三个解决办法

**1.给你的对象增加set/get方法，前提是你有权限的话**

这个的意思很好理解，如果你有权限的话，加个set/get方法就搞定了，但是很多时候我们没有权限去这么做，比如本文开头所提到的问题，你无法给button加上一个合乎要求的setwidth方法，因为这个是Android SDK内部实现的，这个方法很简单，但是往往是不可行的，这里就不对其进行更多的分析了

**2.用这个类来包装原始对象，间接提供get/set方法**

这是一个很有用的解决方法，是笔者最喜欢用的，因为用起来很方便，也好理解，下面我将一个具体的实现来介绍，为了达到这个效果我们写了一个包装类去提供方法，这样也就完美的实现了

|  |
| --- |
| public class AnimViewWraper {  private View view;  public AnimViewWraper(View view) {  this.view = view;  }  public int getWidth() {  return view.getLayoutParams().width;  }  public void setWidth(int width) {  view.getLayoutParams().width = width;  view.requestLayout();  }  } |

ObjectAnimator.ofInt(new AnimViewWraper(mBtn),"width",300).setDuration(10\*1000).start();

**3.采用ValueAnimator，监听动画过程自己去实现**

首先说下什么是ValueAnimator，ValueAnimator本身不作用于任何对象，也就是说直接使用它没有任何的效果，他可以对一个值做动画，然后我们监听这个过程，在过程中修改我们对象的属性值，这样就相当于我们的对象做了动画，下面我们用例子来说明：

|  |
| --- |
| private void performAnimator(final View target, final int start, final int end) {  ValueAnimator valueAnimator = ValueAnimator.ofInt(1, 100);  valueAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {  //持有一个IntEvaluator对象，方便下面估值的时候使用  private IntEvaluator mEvaluator = new IntEvaluator();  @Override  public void onAnimationUpdate(ValueAnimator animation) {  //获得当前动画的进度值，整形1-100之间  int currentValue = (int) animation.getAnimatedValue();  //获得当前进度占整个动画之间的比例，浮点0-1之间  float fraction = animation.getAnimatedFraction();  //直接使用整形估值器，通过比例计算宽度，然后再设置给按钮  target.getLayoutParams().width = mEvaluator.evaluate(fraction, start, end);  target.requestLayout();  }  });  valueAnimator.setDuration(5000).start();  } |

上面的代码的效果和刚才的viewwrapper是一样的，关于ValueAnimator还要再说一下，拿上来的例子来说，他会在5s内将一个数1变成100，然后动画的每一帧会回调的每一帧onAnimationUpdate方法，在这个方法里，我们可以获取当前的值和占用的比例我们可以计算出宽度是多少，比如时间过去了一半，当前值是50，比例是0.5，假设起始值为100，最终是500px，那么500-100=400，所有这个时候乘以0.5=200，这些都是内部实现，我们不用自己写，直接用。

### 属性动画的源码分析

属性动画他要求作用的对象必须有set方法根据传递的最终值去不断的更改然后set，而且每次的值都不一样，就这样根据时间的推移形成动画，那我们具体来看下源码是怎么操作的，首先我们找一个入口，可以看他的start方法。  属性动画需要运行在Lopper线程中，上述代码终会调用AnimationHandler的start，这个并不是真正的handler，他是一个Runnable，看一下他的代码，通过代码我们发现，很快的调用了JNI层，不过JNI层最终还是调回来的，他的run方法被调用，这个Runnable涉及和底层的调用，我们来看下他的doAnimationFrame方法 }

当动画的下一帧到来的时候，PropertyValuesHolder中的setAnimatedValue方法将新的属性值设置给对象，调用其set方法，下面是源码，通过反射调用

在初始化的时候，如果属性的初始值没有提供，则get方法就会调用，请看PropertyValuesHolder中的setupValue



## 使用动画的注意事项

通过动画可以实现一些绚丽的效果，但是在使用过程中也发生了一些问题

1.OOM问题

这个问题主要还是帧动画中，当图片过多的时候就OOM了，这个在实际的开发中尤其注意，尽量避免使用帧动画

2.内存泄漏

在属性动画中有一类无限循环的动画，如果你在activity退出后不停止的话，可能就会存在这个问题了

3.兼容性问题

动画在3.0以下的系统上有缺陷，最好做好适配工作

4.View动画的问题

view动画死对view的影像做动画，并不是真正的改变view的状态，因此有时候会出现完成后view无法隐藏的现象，即setVisibility(View.GONE),这个时候只要调用clearAnimation清除动画即可

5.不要使用PX

在进行动画的过程，要尽量使用dp，使用px会导致适配问题

6.动画元素的交互

将view移动后，在3.0以前的系统，不管是view动画还是属性动画，新位置都无法调用单机事件，同时老位置却可以，从3.0之后，但是事件就是懂后的，但是view动画仍然在原位置。

7.硬件加速

使用动画的过程，建议开启硬件加速，这样会提高动画的流畅性

## QA 如何改为最大呢

**Theme.DeviceDefault.Settings.Dialog.Alert**

<**style name="Theme.DeviceDefault.Settings.Dialog.Alert" parent="Theme.Material.Settings.Dialog.Alert"**>  
 *<!-- Color palette -->* <**item name="colorPrimary"**>@color/primary\_device\_default\_settings</**item**>  
 <**item name="colorPrimaryDark"**>@color/primary\_dark\_device\_default\_settings</**item**>  
 <**item name="colorSecondary"**>@color/secondary\_device\_default\_settings</**item**>  
 <**item name="colorAccent"**>@color/accent\_device\_default\_light</**item**>  
</**style**>

<**style name="Theme.Material.Settings.Dialog.Alert" parent="Theme.Material.Settings.Dialog.BaseAlert"** />

**style name="Theme.Material.Settings.Dialog.BaseAlert" parent="Theme.Material.Light.Dialog.BaseAlert"**>  
 <**item name="colorPrimary"**>@color/primary\_material\_settings</**item**>  
 <**item name="colorPrimaryDark"**>@color/primary\_dark\_material\_settings</**item**>  
 <**item name="colorSecondary"**>@color/secondary\_material\_settings</**item**>  
</**style**>

<**style name="Theme.Material.Light.Dialog" parent="Theme.Material.Light.BaseDialog"**/>

<**style name="Theme.Material.Light.BaseDialog"**>  
 <**item name="colorBackground"**>?attr/colorBackgroundFloating</**item**>  
  
 <**item name="windowFrame"**>@null</**item**>

<**item name="windowTitleStyle"**>@style/DialogWindowTitle.Material.Light</**item**>  
 <**item name="windowTitleBackgroundStyle"**>@style/DialogWindowTitleBackground.Material.Light</**item**>  
 <**item name="windowBackground"**>@drawable/dialog\_background\_material</**item**>  
 <**item name="windowElevation"**>@dimen/floating\_window\_z</**item**>  
 <**item name="windowIsFloating"**>true</**item**>  
 <**item name="windowContentOverlay"**>@null</**item**>  
 <**item name="windowAnimationStyle"**>@style/Animation.Material.Dialog</**item**>  
 <**item name="windowSoftInputMode"**>stateUnspecified|adjustPan</**item**>  
 <**item name="windowActionBar"**>false</**item**>  
 <**item name="windowActionModeOverlay"**>true</**item**>  
 <**item name="windowCloseOnTouchOutside"**>@bool/config\_closeDialogWhenTouchOutside</**item**>  
  
 <**item name="colorBackgroundCacheHint"**>@null</**item**>  
  
 <**item name="listPreferredItemPaddingLeft"**>?attr/dialogPreferredPadding</**item**>  
 <**item name="listPreferredItemPaddingRight"**>?attr/dialogPreferredPadding</**item**>  
 <**item name="listPreferredItemPaddingStart"**>?attr/dialogPreferredPadding</**item**>  
 <**item name="listPreferredItemPaddingEnd"**>?attr/dialogPreferredPadding</**item**>  
  
 <**item name="listDivider"**>@null</**item**>  
  
 <**item name="preferencePanelStyle"**>@style/PreferencePanel.Dialog</**item**>  
</**style**>

http://blog.supenta.com/2014/07/02/how-to-style-alertdialogs-like-a-pro/