# 一.动画

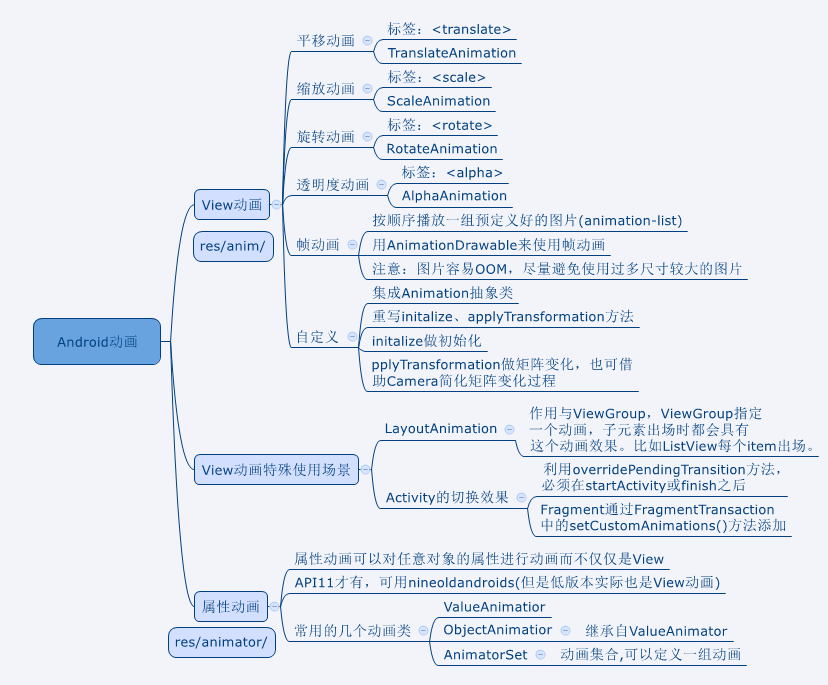
## 1.动画的种类

1.补间动画**(res/anim/)**Tween Animation或者View Animation

2.逐帧动画**(res/drawable/)**Frame Animation 或者Drawable Animation

3.属性动画**（res/animator/)**Property Animation,3.0引入的

4.过渡动画 **（res/transition/)** Transition,4.4引入的,5.0更加的丰富和易用



## 2.动画详解

### 2.1补间动画

#### 定义:

Tween动画是操作某个控件让其展现出旋转、渐变、移动、缩放的这么一种转换过程.

其中包括alpha(透明度)、scale(缩放)、translate(移动)、rotate(翻转)

对应java code:

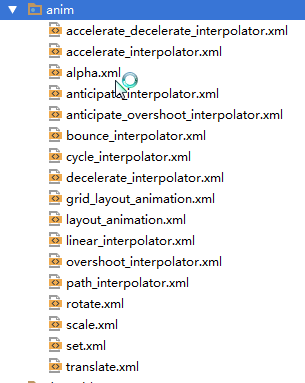
alpha->AlphaAnimation

scale->ScaleAnimation

translate->TranslateAnimation

rotate->RotateAnimation。

#### 种类:



##### 1. 插值器

accelerate\_decelerate\_interpolator.xml

accelerate\_interpolator.xml

anticipate\_interpolator.xml

anticipate\_overshoot\_interpolator.xml

bounce\_interpolator.xml

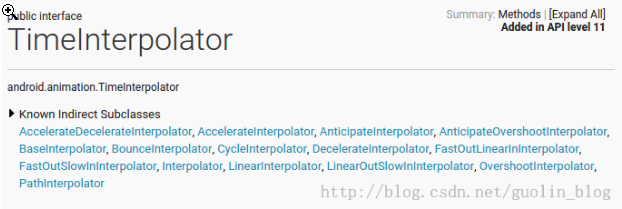
cycle\_interpolator.xml

decelerate\_interpolator.xml

linear\_interpolator.xml

overshoot\_interpolator.xml

path\_interpolator.xml





|  |
| --- |
| **1.AccelerateInterpolator  加速插值器**   加速的快慢度由参数fractor决定。  当fractor值为1.0f时，动画加速轨迹相当于一条y=x^2的抛物线。如下图：  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\7c2609e1ea234bb7b0a965cf88241ca7\5_hiuk_78883.png  当fractor不为1时，轨迹曲线是y=x^(2\*fractor)(0<x<=1)的曲线。  示例：当fractor为4时，插值器的加速轨迹曲线如下图：  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\9ba47935f9424b34a00c0b80a04b4f68\0_yea8_78883.png  如果你在使用AccelerateInterpolator时，想要那种一开始很慢，然后突然就很快的加速的动画效果的话。  就将fractor设置大点。  你可以到这里调试下你想要的抛物线效果：[http://www.wolframalpha.com/input/?i=x%5E%282\*3%29%280%3Cx%3C%3D1%29](http://www.wolframalpha.com/input/?i=x%5E%282*3%29%280%3Cx%3C%3D1%29)  Android提供的一个不同factor的加速插值器:  (1)accelerate\_cubic, factor为1.5  **2. DecelerateInterpolator 减速插值器**  根据getInterpolationa(float input);方法可以知道。  当fractor为1.0f。它减速的轨迹曲线为1-(1-x)^2。如下图：  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\ae0a73abf0e845c9924f58c3669c046d\5_mjhe_78883.png  当fractor增大到4时，曲线轨迹如下图：  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\202e2544a16546a3a1ca78aa667ce88d\2_ou2b_78883.png  **3.  AccelerateDecelerateInterpolator  加速减速插值器**  根据getInterpolation()方法可以得出其变化曲线如下：  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\3a74e33af73843fdb7e24676023d0bcd\9_mjvz_78883.png  **4. LinearInterpolator 线性插值器**  这可是最简单的插值器：  **5. BounceInterpolator 弹跳插值器**  根据getInterpolation()得到以下插值曲线图：  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\16a073b4f96745b1a90f9e3e4b528799\1_xris_78883.png  （这个插值器的作图函数我得记录下来啊）  plot Piecewise[{ {(1.1226x)^2\*8,0<x<0.3535}, {((1.1226x)-0.54719)^2\*8+0.7,0.3535<=x<0.7408}, {((1.1226x)-0.8526)^2\*8+0.9,0.7408<=x<0.9644}, {((1.1226x)-1.0435)^2\*8+0.95,0.9644<=x<=1}}]  **6.AnticipateInterpolator 回荡秋千插值器**  这个插值器的值变化过程，可以想像成荡秋千时的一个段过程。（此时秋千已经在比较上面的位置了，一放手就可以荡下来）。你开始用力推向更上面，然后秋千终将荡回下面。  tension值就好比推力的大小。  根据getInterpolation()方法。  当tension为默认值2.0f时，曲线图如下：  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\b5f2c850cfc64cbc89dd6b919008f834\5_s2ah_78883.png  当tension值为4.0f时，曲线图如下：  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\245cf7f0c3184273976efcac91536a38\9_5ha7_78883.png  **7. AnticipateOvershootInterpolator**  根据getInterpolation()方法，  可以得到当tension为默认值时，曲线图为：  plot Piecewise[{{0.5((2x)\*(2x)\*((2+1)\*2x-2)), 0<x<0.5}, {0.5\*(((2x-2)\*(2x-2)\*((2+1)\*(2x-2)+2))+2),0.5<=x<=1}}]  （不知道我的plot函数写对了没？）  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\5697a274c2ff4a0aba3df861763e5985\3_qsku_78883.png    **8. CycleInterpolator 正弦周期变化插值器**  当cycle时为1时，即变化一周时，曲线图如下：  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\dd48b2b3f6ad468d9b8a9824563b6953\5_25cz_78883.png  **9. OvershootInterpolator**  当tension为默认值2时，曲线图如下：  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\679450250df849278f14863462ca0868\6_vlji_78883.png  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\5fe89e58d4b7494a94a33ce9d9de9d21\7_yvzx_78883.png  当tension的值为4时，曲线图如下：  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\42615c802465477e89c5bae387710117\4_yznx_78883.png  C:\Users\zhiren.zhang\AppData\Local\YNote\data\zhirenzhangdashi@163.com\2fbc0521298849b08c3a5ccd9e0444fc\5_0xv5_78883.png  通过学习了解Android自带的这些Interpolator，我们可以很好的根据自己的使用场景使用这些Interpolator了。也可以很容易的写出我们自己的Interpolator。 |
| 1.xml插值器自定义的用法  xml的方式主要是修改现有插值器的属性。但是有些属性是不能修改的。属性如下:  <accelerateDecelerateInterpolator> 无可自定义的attribute。  <accelerateInterpolator> android:factor 浮点值,加速速率(默认值为1)。  <anticipateInterploator> android:tension 浮点值,起始点后拉的张力数(默认值为2)。  <anticipateOvershootInterpolator> android:tension 浮点值,起始点后拉的张力数(默认值为2)。 android:extraTension 浮点值,拉力的倍数(默认值为1.5)。  <bounceInterpolator> 无可自定义的attribute。  <cycleInterplolator> android:cycles 整形,循环的个数(默认为1)。  <decelerateInterpolator> android:factor 浮点值,减速的速率(默认为1)。  <linearInterpolator> 无可自定义的attribute。  <overshootInterpolator> android:tension 浮点值,超出终点后的张力(默认为2)。  在xml中定义  <?xml version="1.0" encoding="utf-8"?>  <overshootInterpolator xmlns:android="http://schemas.android.com/apk/res/android"  android:tension="8.0">  </overshootInterpolator >  然后在xml中调用  <scale xmlns:android="http://schemas.android.com/apk/res/android"  android:interpolator="@anim/my\_overshoot\_interpolator"  .../>  2.Java自定义插值器的(Java自定义插值器其实是xml自定义的升级，也就是说如果我们修改xml的属性还不能满足需求。那就可以选择通过Java来实现)方式。  我们可以实现Interpolator接口:因为上面所有的Interpolator都实现了Interpolator接口。这个接口定义了一个方法:float getInterpolation(float input) 此方法由系统调用input代表动画的时间,在0和1之间,也就是开始和结束之间。 线性(匀速)插值器定义如下:  public class CustomerInterpolator implements Interpolator {  @Override  public float getInterpolation(float input) {  return (float)(Math.cos((input + 1) \* Math.PI) / 2.0f) + 0.5f;  }  }  文／阿祥JOKER（简书作者） 原文链接：http://www.jianshu.com/p/551f84402752 著作权归作者所有，转载请联系作者获得授权，并标注“简书作者”。 |

##### 2.子类统一动画

grid\_layout\_animation.xml

layout\_animation.xml

##### 3. 视图动画

alpha.xml

rotate.xml

scale.xml

set.xml

translate.xml

#### 通用属性:

android:duration 动画从开始到结束持续的时长，单位为毫秒

android:detachWallpaper 设置是否在壁纸上运行，只对设置了壁纸背景的窗口动画(window animation)有效。设为true，则动画只在窗口运行，壁纸背景保持不变

android:fillAfter 设置为true时，动画执行完后，View会停留在动画的最后一帧；默认为false；如果是动画集，需在<set>标签中设置该属性才有效

android:fillBefore 设置为true时，动画执行完后，View回到动画执行前的状态，默认即为true

android:fillEnabled 设置为true时，android:fillBefore的值才有效，否则android:fillBefore会被忽略

android:repeatCount 设置动画重复执行的次数，默认为0，即不重复；可设为-1或infinite，表示无限重复

android:repeatMode 设置动画重复执行的模式，可设为以下两个值其中之一：

restart 动画重复执行时从起点开始，默认为该值

reverse 动画会反方向执行

android:startOffset 设置动画执行之前的等待时长，毫秒为单位；重复执行时，每次执行前同样也会等待一段时间

android:zAdjustment 表示被设置动画的内容在动画运行时在Z轴上的位置，取值为以下三个值之一：

normal 默认值，保持内容在Z轴上的位置不变

top 保持在Z周最上层

bottom 保持在Z轴最下层

android:interpolator 设置动画速率的变化，比如加速、减速、匀速等，需要指定Interpolator资源，后面再详细讲解

备注:1.无论我们补间动画的哪一种都已经具备了这种属性，也都可以设置使用这些属性中的一个或多个

2. <set>标签还有个android:shareInterpolator属性，设置为true时则可将interpolator应用到所有子元素中

#### xml的实际应用:

|  |
| --- |
| java代码中应用alpha.xml:  *//视图动画是依附在视图上的,而属性动画是改变view的属性*  Animation animation = **AnimationUtils**.*loadAnimation*(**this**, R.anim.***alpha***);  animation.setAnimationListener(**new** Animation.AnimationListener() {  @Override  **public void** onAnimationStart(Animation animation) {  }  @Override  **public void** onAnimationEnd(Animation animation) {  }  @Override  **public void** onAnimationRepeat(Animation animation) {  }  });  *//view.startAnimation(animation);顶替下面两句*  view.setAnimation(animation);  animation.start(); |

##### alpha.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>*  *<!--android:interpolator="@android:anim/accelerate\_interpolator" 插值器-->*  *<!--android:repeatCount="infinite"重复旋转-->*  *<!-- android:repeatMode="reverse"重复模式改为相反-->*  *<!--浮点型值:*  *fromAlpha 属性为动画起始时透明度*  *toAlpha   属性为动画结束时透明度*  *说明:*  *0.0表示完全透明*  *1.0表示完全不透明*  *以上值取0.0-1.0之间的float数据类型的数字*  *长整型值:*  *duration  属性为动画持续时间*  *说明:*  *时间以毫秒为单位-->*  <**alpha xmlns:android="http://schemas.android.com/apk/res/android"**  **android:duration="2000"**  **android:fromAlpha="0"**  **android:interpolator="@android:anim/accelerate\_interpolator"**  **android:repeatCount="infinite"**  **android:repeatMode="reverse"**  **android:toAlpha="1.0"**></**alpha**> |

##### rotate.xml

|  |
| --- |
| <rotate>可以实现旋转的动画效果，主要的属性如下：   * android:duration 动画从开始到结束持续的时长，单位为毫秒 * android:fromDegrees 旋转开始的角度 * android:toDegrees 旋转结束的角度 * android:pivotX 旋转中心点的X坐标，纯数字表示相对于View本身左边缘的像素偏移量；带"%"后缀时表示相对于View本身左边缘的百分比偏移量；带"%p"后缀时表示相对于父View左边缘的百分比偏移量 * android:pivotY 旋转中心点的Y坐标，纯数字表示相对于View本身顶部边缘的像素偏移量；带"%"后缀时表示相对于View本身顶部边缘的百分比偏移量；带"%p"后缀时表示相对于父View顶部边缘的百分比偏移量   以下示例代码旋转角度从0到360，即旋转了一圈，旋转的中心点都设为了50%，即是View本身中点的位置。  1. 在anim下新建rotate.xml:  *<?***xml version="1.0" encoding="utf-8"***?><!-- 动画——旋转*  *android:repeatCount="infinite"重复旋转*  *android:fromDegrees="0"*  *开始角度*  *android:toDegrees="360" 结束角度*  *android:pivotX="0"*  *android:pivotY="0"旋转中心*  *android:repeatMode="reverse"重复模式改为相反*  *-->*  <**rotate xmlns:android="http://schemas.android.com/apk/res/android"**  **android:duration="2000"**  **android:fromDegrees="0"**  **android:pivotX="50%"**  **android:pivotY="50%"**  **android:repeatCount="infinite"**  **android:repeatMode="reverse"**  **android:toDegrees="360"**>  </**rotate**> |

##### scale.xml

|  |
| --- |
| *<!-- 尺寸伸缩动画效果 scale*  *属性：interpolator 指定一个动画的插入器*  *在我试验过程中，使用android.res.anim中的资源时候发现*  *有三种动画插入器:*  *accelerate\_decelerate\_interpolator 加速-减速 动画插入器*  *accelerate\_interpolator 加速-动画插入器*  *decelerate\_interpolator 减速- 动画插入器*  *其他的属于特定的动画效果*  *浮点型值：*  *fromXScale 属性为动画起始时 X坐标上的伸缩尺寸*  *toXScale 属性为动画结束时 X坐标上的伸缩尺寸*  *fromYScale 属性为动画起始时Y坐标上的伸缩尺寸*  *toYScale 属性为动画结束时Y坐标上的伸缩尺寸*  *说明:*  *以上四种属性值*  *0.0表示收缩到没有*  *1.0表示正常无伸缩*  *值小于1.0表示收缩*  *值大于1.0表示放大*  *pivotX 属性为动画相对于物件的X坐标的开始位置*  *pivotY 属性为动画相对于物件的Y坐标的开始位置*  *说明:*  *以上两个属性值 从0%-100%中取值*  *50%为物件的X或Y方向坐标上的中点位置*  *长整型值：*  *duration 属性为动画持续时间*  *说明: 时间以毫秒为单位*  *布尔型值:*  *fillAfter 属性 当设置为true ，该动画转化在动画结束后被应用*  *-->*  <**scale xmlns:android="http://schemas.android.com/apk/res/android"**  **android:duration="2000"**  **android:fromXScale="1"**  **android:fromYScale="1"**  **android:pivotX="50%"**  **android:pivotY="50%"**  **android:repeatCount="infinite"**  **android:repeatMode="reverse"**  **android:toXScale="2"**  **android:toYScale="2"**  ></**scale**> |

##### set.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>*  *<!--若要实现多个效果共同体现在同一个组件上，将各种效果文件全部写到set里*  *标签还有个android:shareInterpolator属性，设置为true时则可将interpolator应用到所有子元素中*  *-->*  <**set xmlns:android="http://schemas.android.com/apk/res/android"**  **android:interpolator="@android:anim/accelerate\_interpolator"**  >  <**rotate xmlns:android="http://schemas.android.com/apk/res/android"**  **android:duration="2000"**  **android:fromDegrees="0"**  **android:pivotX="50%"**  **android:pivotY="50%"**  **android:repeatCount="infinite"**  **android:repeatMode="reverse"**  **android:toDegrees="360"** />  <**scale xmlns:android="http://schemas.android.com/apk/res/android"**  **android:duration="2000"**  **android:fromXScale="1"**  **android:fromYScale="1"**  **android:pivotX="50%"**  **android:pivotY="50%"**  **android:repeatCount="infinite"**  **android:repeatMode="reverse"**  **android:toXScale="2"**  **android:toYScale="2"**  />  <**translate xmlns:android="http://schemas.android.com/apk/res/android"**  **android:duration="2000"**  **android:fromXDelta="-200"**  **android:fromYDelta="0"**  **android:repeatCount="infinite"**  **android:repeatMode="reverse"**  **android:toXDelta="200"**  **android:toYDelta="0"** />  <**alpha xmlns:android="http://schemas.android.com/apk/res/android"**  **android:duration="2000"**  **android:fromAlpha="1"**  **android:repeatCount="infinite"**  **android:repeatMode="reverse"**  **android:toAlpha="0"**  />  </**set**> |

##### translate.xml

|  |
| --- |
| <translate>可以实现位置移动的动画效果，可以是垂直方向的移动，也可以是水平方向的移动。坐标的值可以有三种格式：从-100到100，以"%"结束，表示相对于View本身的百分比位置；如果以"%p"结束，表示相对于View的父View的百分比位置；如果没有任何后缀，表示相对于View本身具体的像素值。主要的属性如下：   * android:duration 动画从开始到结束持续的时长，单位为毫秒 * android:fromXDelta 起始位置的X坐标的偏移量 * android:toXDelta 结束位置的X坐标的偏移量 * android:fromYDelta 起始位置的Y坐标的偏移量 * android:toYDelta 结束位置的Y坐标的偏移量   看示例吧，以下代码实现的是从左到右的移动效果，起始位置为相对于控件本身-100%的位置，即在控件左边，与控件本身宽度一致的位置；结束位置为相对于父控件100%的位置，即会移出父控件右边缘的位置。  1. 在anim下新建translate.xml:  <**translate xmlns:android="http://schemas.android.com/apk/res/android"**  **android:duration="2000"**  **android:fromXDelta="-200"**  **android:fromYDelta="0"**  **android:repeatCount="infinite"**  **android:repeatMode="reverse"**  **android:toXDelta="200"**  **android:toYDelta="0"** /> |

##### layout\_animation.xml-为viewGroup添加进入统一动画

|  |
| --- |
| **定义和使用**  layout\_animation.xml:  <?xml version="1.0" encoding="utf-8"?> <layoutAnimation xmlns:android="http://schemas.android.com/apk/res/android" android:delay="1" android:animationOrder="normal" android:animation="@anim/slide\_in\_left"/>  其中的@anim/slide\_in\_left对应代码为：  这部分实现的效果是，让控件从左边50%的位置进入屏幕，同时透明度从0变到1；动画总时长为1000毫秒。  <?xml version="1.0" encoding="utf-8"?> <set xmlns:android="http://schemas.android.com/apk/res/android" android:duration="1000"> <translate android:fromXDelta="-50%p" android:toXDelta="0"/> <alpha android:fromAlpha="0.0" android:toAlpha="1.0"/> </set>  <ListView android:id="@+id/listview" android:layout\_width="match\_parent" android:layout\_height="match\_parent" android:layoutAnimation="@anim/layout\_animation" />  注意:动画仅在第一次创建时有用，后期加入的数据，将不会再有动画,这个不能解决  **属性含义:**   * **delay:**指每个Item的动画开始延时，取值是android:animation所指定动画时长的倍数，取值类型可以是float类型，也可以是百分数，默认是0.5;比如我们这里指定的动画是@anim/slide\_in\_left，而在slide\_in\_left.xml中指定android:duration=”1000”，即单次动画的时长是1000毫秒，而我们在这里的指定android:delay=”1”，即一个Item的动画会在上一个item动画完成后延时单次动画时长的一倍时间开始，即延时1000毫秒后开始。 * **animationOrder:**指viewGroup中的控件动画开始顺序，取值有normal(正序)、reverse(倒序)、random(随机) * **animation：**指定每个item入场所要应用的动画。仅能指定res/aim文件夹下的animation定义的动画，不可使用animator动画。   这里最难理解的参数应该是android:delay，它是指viewGroup中各个item开始动画的时间延迟，取值是Item动画时长的倍数。其中item动画是通过android:animation指定的。  **代码中使用:**  //代码设置通过加载XML动画设置文件来创建一个Animation对象； Animation animation= AnimationUtils.loadAnimation(this,R.anim.slide\_in\_left); //得到一个LayoutAnimationController对象； LayoutAnimationController controller = new LayoutAnimationController(animation); //设置控件显示的顺序； controller.setOrder(LayoutAnimationController.ORDER\_REVERSE); //设置控件显示间隔时间； controller.setDelay(0.3f); //为ListView设置LayoutAnimationController属性； mListView.setLayoutAnimation(controller); mListView.startLayoutAnimation(); |

##### grid\_layout\_animation.xml-为 grideView添加统一动画

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <gridLayoutAnimation xmlns:android="http://schemas.android.com/apk/res/android" android:rowDelay="75%" android:columnDelay="60%" android:directionPriority="none" android:direction="bottom\_to\_top|right\_to\_left" android:animation="@android:anim/slide\_in\_left"/>  这是一个写好了的gridLayoutAnimation的动画文件。其中各字段的意义如下：  - **rowDelay:**每一行动画开始的延迟。与LayoutAnimation一样，可以取百分数，也可以取浮点数。取值意义为，当前android:animation所指动画时长的倍数。  - **columnDelay：**每一列动画开始的延迟。取值类型及意义与rowDelay相同。  - **directionPriority：**方向优先级。取值为row,collumn,none，意义分别为：行优先，列优先，和无优先级（同时进行）;具体意义，后面会细讲  - \*\*direction：\*\*gridview动画方向。  取值有四个：left\_to\_right：列，从左向右开始动画  right\_to\_left ：列，从右向左开始动画  top\_to\_bottom：行，从上向下开始动画  bottom\_to\_top：行，从下向上开始动画  这四个值之间可以通过“|”连接，从而可以取多个值。很显然left\_to\_right和right\_to\_left是互斥的，top\_to\_bottom和bottom\_to\_top是互斥的。如果不指定 direction字段，默认值为left\_to\_right | top\_to\_bottom；即从上往下，从左往右。  - **animation:** gridview内部元素所使用的动画。  <GridView android:id="@+id/grid" android:layout\_width="match\_parent" android:layout\_height="match\_parent" android:columnWidth="60dp" android:gravity="center" android:horizontalSpacing="10dp" android:layoutAnimation="@anim/gride\_animation" android:numColumns="auto\_fit" android:stretchMode="columnWidth" android:verticalSpacing="10dp"/>  gridLayoutAnimation仅在gridview第一次创建时各个元素才会有出场动画，在创建成功以后，再向其中添加数据就不会再有动画。这一点与layoutAnimation相同。  Java代码实现:  GridLayoutAnimationController controller = new GridLayoutAnimationController(animation); controller.setColumnDelay(0.75f); controller.setRowDelay(0.5f); controller.setDirection(GridLayoutAnimationController.DIRECTION\_BOTTOM\_TO\_TOP|GridLayoutAnimationController.DIRECTION\_LEFT\_TO\_RIGHT); controller.setDirectionPriority(GridLayoutAnimationController.PRIORITY\_NONE); grid.setLayoutAnimation(controller); grid.startLayoutAnimation(); |

#### java代码的实际应用:

|  |
| --- |
| */\*\**  *\* 1.0 意味着 不透明*  *\* 0.0 意味着 透明*  *\*/*  AlphaAnimation aa = **new** AlphaAnimation(0, 1.0f);  aa.setDuration(2000); *//设置动画显示的时间*  aa.setRepeatCount(2); *//设置动画显示次数*  aa.setRepeatMode(Animation.***REVERSE***); *//设置动画显示模式*  *// 把动画设置到 iv上*  iv.startAnimation(aa); |
| ScaleAnimation sa = **new** ScaleAnimation(0.1f, 2.0f, 0.1f, 2.0f, Animation.***RELATIVE\_TO\_SELF***, 0.5f, Animation.***RELATIVE\_TO\_SELF***, 0.5f);  sa.setDuration(2000); *//设置动画显示的时间*  sa.setRepeatCount(2); *//设置动画显示次数*  sa.setRepeatMode(Animation.***REVERSE***); *//设置动画显示模式*  *// 把动画设置到 iv上*  iv.startAnimation(sa); |
| *//RotateAnimation ra = new RotateAnimation(0, 360);*  RotateAnimation ra = **new** RotateAnimation(0, 360, Animation.***RELATIVE\_TO\_SELF***, 0.5f, Animation.***RELATIVE\_TO\_SELF***, 0.5f);  ra.setDuration(2000); *//设置动画显示的时间*  ra.setRepeatCount(2); *//设置动画显示次数*  ra.setRepeatMode(Animation.***REVERSE***); *//设置动画显示模式*  *// 把动画设置到 iv上*  iv.startAnimation(ra); |
| TranslateAnimation ta = **new** TranslateAnimation(Animation.***RELATIVE\_TO\_PARENT***, -0.5f, Animation.***RELATIVE\_TO\_PARENT***, 0.5f, Animation.***RELATIVE\_TO\_PARENT***, -0.5f, Animation.***RELATIVE\_TO\_PARENT***, 0.5f);  ta.setDuration(2000); *//设置动画显示的时间*  ta.setRepeatCount(2); *//设置动画显示次数*  ta.setRepeatMode(Animation.***REVERSE***); *//设置动画显示模式*  *// 把动画设置到 iv上*  iv.startAnimation(ta); |
| AnimationSet set = **new** AnimationSet(**false**);  AlphaAnimation aa = **new** AlphaAnimation(0, 1.0f);  aa.setDuration(2000); *//设置动画显示的时间*  aa.setRepeatCount(2); *//设置动画显示次数*  aa.setRepeatMode(Animation.***REVERSE***); *//设置动画显示模式*  RotateAnimation ra = **new** RotateAnimation(0, 360, Animation.***RELATIVE\_TO\_SELF***, 0.5f, Animation.***RELATIVE\_TO\_SELF***, 0.5f);  ra.setDuration(2000); *//设置动画显示的时间*  ra.setRepeatCount(2); *//设置动画显示次数*  ra.setRepeatMode(Animation.***REVERSE***); *//设置动画显示模式*  ScaleAnimation sa = **new** ScaleAnimation(0.1f, 2.0f, 0.1f, 2.0f, Animation.***RELATIVE\_TO\_SELF***, 0.5f, Animation.***RELATIVE\_TO\_SELF***, 0.5f);  sa.setDuration(2000); *//设置动画显示的时间*  sa.setRepeatCount(2); *//设置动画显示次数*  sa.setRepeatMode(Animation.***REVERSE***); *//设置动画显示模式*  *//添加动画*  set.addAnimation(sa);  set.addAnimation(aa);  set.addAnimation(ra);  iv.startAnimation(set); |

### 2.2逐帧动画

<!-- 注意：rocket.xml文件位于res/drawable/目录下 -->

<?xml version="1.0" encoding="utf-8"?>

<animation-list xmlns:android="http://schemas.android.com/apk/res/android"

android:oneshot=["true" | "false"] >

<item

android:drawable="@[package:]drawable/drawable\_resource\_name"

android:duration="integer" />

</animation-list>

ImageView rocketImage = (ImageView) findViewById(R.id.rocket\_image);

rocketImage.setBackgroundResource(R.drawable.rocket\_thrust);

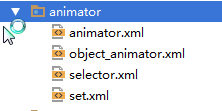
rocketAnimation = (AnimationDrawable) rocketImage.getBackground();

rocketAnimation.start();

AnimationDrawable的start()方法不能在Activity的onCreate方法中调运，因为AnimationDrawable还未完全附着到window上，所以最好的调运时机是onWindowFocusChanged()方法中。

### 2.3属性动画

#### 种类:



|  |  |
| --- | --- |
| ValueAnimator | <animator> 放置在res/animator/目录下 |
| TimeAnimator | [不支持/点我查看原因](http://developer.android.com/guide/topics/resources/animation-resource.html) |
| ObjectAnimator | <objectAnimator> 放置在res/animator/目录下 |
| AnimatorSet | <set> 放置在res/animator/目录下 |

Animator对应ValueAnimator, objectAnimator对应ObjectAnimator,set对应AnimatorSet

#### 属性:

通过**<animator>**标签可以很方便的对**ValuAnimiator**进行设置，可设置的属性如下：

* **android:duration** 动画从开始到结束持续的时长，单位为毫秒
* **android:startOffset** 设置动画执行之前的等待时长，单位为毫秒
* **android:repeatCount** 设置动画重复执行的次数，默认为0，即不重复；可设为-1或infinite，表示无限重复
* **android:repeatMode** 设置动画重复执行的模式，可设为以下两个值其中之一：
  + *restart* 动画重复执行时从起点开始，默认为该值
  + *reverse* 动画会反方向执行
* **android:valueFrom** 动画开始的值，可以为int值、float值或color值
* **android:valueTo** 动画结束的值，可以为int值、float值或color值
* **android:valueType** 动画值类型，若为color值，则无需设置该属性
  + *intType* 指定动画值，即以上两个value属性的值为整型
  + *floatType* 指定动画值，即以上两个value属性的值为浮点型，默认值
* **android:interpolator** 设置动画速率的变化，比如加速、减速、匀速等，需要指定

**<set>属性解释：**

|  |  |
| --- | --- |
| **xml属性** | **解释** |
| android:ordering | 控制子动画启动方式是先后有序的还是同时进行。sequentially:动画按照先后顺序；together(默认):动画同时启动； |

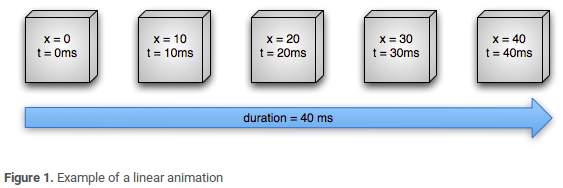
**<objectAnimator>属性解释：比< animator >仅仅多了一个**android:propertyName,其他的都类似

|  |  |
| --- | --- |
| **xml属性** | **解释** |
| android:propertyName | String类型，必须要设置的节点属性，代表要执行动画的属性（通过名字引用），辟如你可以指定了一个View的”alpha” 或者 “backgroundColor” ，这个objectAnimator元素没有对外说明target属性，所以你不能在XML中设置执行这个动画，必须通过调用 loadAnimator()方法加载你的XML动画资源，然后调用setTarget()应用到具备这个属性的目标对象上（譬如TextView）。 |
| android:valueTo | float、int或者color类型，必须要设置的节点属性，表明动画结束的点；如果是颜色的话，由6位十六进制的数字表示。 |
| android:valueFrom | 相对应valueTo，动画的起始点，如果没有指定，系统会通过属性的get方法获取，颜色也是6位十六进制的数字表示。 |
| android:duration | 动画的时长，int类型，以毫秒为单位，默认为300毫秒。 |
| android:startOffset | 动画延迟的时间，从调用start方法后开始计算，int型，毫秒为单位。 |
| android:repeatCount | 一个动画的重复次数，int型，”-1“表示无限循环，”1“表示动画在第一次执行完成后重复执行一次，也就是两次，默认为0，不重复执行。 |
| android:repeatMode | 重复模式：int型，当一个动画执行完的时候应该如何处理。该值必须是正数或者是-1，“reverse”会使得按照动画向相反的方向执行，可实现类似钟摆效果。“repeat”会使得动画每次都从头开始循环。 |
| android:valueType | 关键参数，如果该value是一个颜色，那么就不需要指定，因为动画框架会自动的处理颜色值。有intType和floatType（默认）两种：分别说明动画值为int和float型。 |

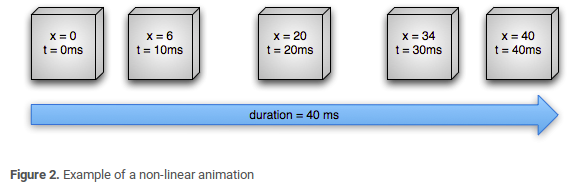
Android属性动画（注意最低兼容版本，不过可以使用开源项目来替代低版本问题）提供了以下属性：

* Duration：动画的持续时间；
* TimeInterpolation：定义动画变化速率的接口，所有插值器都必须实现此接口，如线性、非线性插值器；
* TypeEvaluator：用于定义属性值计算方式的接口，有int、float、color类型，根据属性的起始、结束值和插值一起计算出当前时间的属性值；
* Animation sets：动画集合，即可以同时对一个对象应用多个动画，这些动画可以同时播放也可以对不同动画设置不同的延迟；
* Frame refreash delay：多少时间刷新一次，即每隔多少时间计算一次属性值，默认为10ms，最终刷新时间还受系统进程调度与硬件的影响；
* Repeat Country and behavoir：重复次数与方式，如播放3次、5次、无限循环，可以让此动画一直重复，或播放完时向反向播放；

接下来先来看官方为了解释原理给出的两幅图（其实就是初中物理题，不解释）：



上面就是一个线性匀速动画，描述了一个Object的X属性运动动画，该对象的X坐标在40ms内从0移动到40，每10ms刷新一次，移动4次，每次移动为40/4=10pixel。



上面是一个非匀速动画，描述了一个Object的X属性运动动画，该对象的X坐标在40ms内从0移动到40，每10ms刷新一次，移动4次，但是速率不同，开始和结束的速度要比中间部分慢，即先加速后减速。

#### Xml实现方式和使用:

##### Animator:

<!-- res/animator/value\_animator.xml -->

<?xml version="1.0" encoding="utf-8"?>

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

android:duration="3000"

android:valueFrom="100"

android:valueTo="20"

android:valueType="intType" />

// 获取屏幕宽度

final int maxWidth = getWindowManager().getDefaultDisplay().getWidth();

//该动画还没有使用到任何view中,仅仅是启动了而已

ValueAnimator valueAnimator = (ValueAnimator) AnimatorInflater.loadAnimator(this, R.animator.value\_animator);

valueAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {

@Override

public void onAnimationUpdate(ValueAnimator animator) {

// 当前动画值，即为当前宽度比例值

int currentValue = (Integer) animator.getAnimatedValue();

// 根据比例更改目标view的宽度

view.getLayoutParams().width = maxWidth \* currentValue / 100;

view.requestLay});

valueAnimator.start();

##### set和objectAnimator:

<set

android:ordering=["together" | "sequentially"]>

<objectAnimator

android:propertyName="string"

android:duration="int"

android:valueFrom="float | int | color"

android:valueTo="float | int | color"

android:startOffset="int"

android:repeatCount="int"

android:repeatMode=["repeat" | "reverse"]

android:valueType=["intType" | "floatType"]/>

<animator

android:duration="int"

android:valueFrom="float | int | color"

android:valueTo="float | int | color"

android:startOffset="int"

android:repeatCount="int"

android:repeatMode=["repeat" | "reverse"]

android:valueType=["intType" | "floatType"]/>

<set>

</set>

</set>

AnimatorSet set = (AnimatorSet) AnimatorInflater.loadAnimator(myContext, R.animtor.property\_animator);

//将动画使用到view中,此时也可以设置监听

set.setTarget(myObject);

set.start();

#### Java代码实现方式:

通用:

ObjectAnimator mObjectAnimator= ObjectAnimator.ofInt(view, "customerDefineAnyThingName", 0, 1).setDuration(2000); mObjectAnimator.addUpdateListener(new AnimatorUpdateListener() { @Override public void onAnimationUpdate(ValueAnimator animation) { //int value = animation.getAnimatedValue(); 可以获取当前属性值 //view.postInvalidate(); 可以主动刷新 //view.setXXX(value); //view.setXXX(value); //......可以批量修改属性 } });

例子:

##### ObjectAnimator

1.如下是一个我在项目中的Y轴3D旋转动画实现实例：

ObjectAnimator.ofFloat(view, "rotationY", 0.0f, 360.0f).setDuration(1000).start();

 View有对应的什么属性呢?

2.PropertyValuesHolder：多属性动画同时工作管理类。有时候我们需要同时修改多个属性，那就可以用到此类，具体如下：

PropertyValuesHolder a1 = PropertyValuesHolder.ofFloat("alpha", 0f, 1f);

PropertyValuesHolder a2 = PropertyValuesHolder.ofFloat("translationY", 0, viewWidth);

......

ObjectAnimator.ofPropertyValuesHolder(view, a1, a2, ......).setDuration(1000).start();

##### ValueAnimator

ValueAnimator animator = ValueAnimator.ofFloat(0, mContentHeight); //定义动画

animator.setTarget(view); //设置作用目标

animator.setDuration(5000).start();

animator.addUpdateListener(new AnimatorUpdateListener() {

@Override

public void onAnimationUpdate(ValueAnimator animation){

float value = (float) animation.getAnimatedValue();

view.setXXX(value); //必须通过这里设置属性值才有效

view.mXXX = value; //不需要setXXX属性方法

}

});

##### AnimationSet

ObjectAnimator a1 = ObjectAnimator.ofFloat(view, "alpha", 1.0f, 0f);

ObjectAnimator a2 = ObjectAnimator.ofFloat(view, "translationY", 0f, viewWidth);

......

AnimatorSet animSet = new AnimatorSet();

animSet.setDuration(5000);

animSet.setInterpolator(new LinearInterpolator());

//animSet.playTogether(a1, a2, ...); //两个动画同时执行

animSet.play(a1).after(a2); //先后执行

......//其他组合方式

animSet.start();

##### Evaluators

类型估值器（TypeEvaluator）是针对于属性动画框架的，对于View动画框架是不需要类型估值器（TypeEvaluator）的。类型估值器（TypeEvaluator）的作用是根据属性值改变的百分比计算出改变后的属性值。由于属性动画实际上作用的是对象的属性，而属性的类型是不同的

相关类解释： Evaluators就是属性动画系统如何去计算一个属性值。它们通过Animator提供的动画的起始和结束值去计算一个动画的属性值。

● IntEvaluator：整数属性值。

● FloatEvaluator：浮点数属性值。

● ArgbEvaluator：十六进制color属性值。

● TypeEvaluator：用户自定义属性值接口，譬如对象属性值类型不是int、float、color类型，你必须实现这个接口去定义自己的数据类型。

既然说到这了，那就来个例子吧，譬如我们需要实现一个自定义属性类型和计算规则的属性动画，如下类型float[]：

ValueAnimator valueAnimator = new ValueAnimator();

valueAnimator.setDuration(5000);

valueAnimator.setObjectValues(new float[2]); //设置属性值类型

valueAnimator.setInterpolator(new LinearInterpolator());

valueAnimator.setEvaluator(new TypeEvaluator<float[]>()

{

@Override

public float[] evaluate(float fraction, float[] startValue,

float[] endValue)

{

//实现自定义规则计算的float[]类型的属性值

float[] temp = new float[2];

temp[0] = fraction \* 2;

temp[1] = (float)Math.random() \* 10 \* fraction;

return temp;

}

});

valueAnimator.start();

valueAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener()

{

@Override

public void onAnimationUpdate(ValueAnimator animation)

{

float[] xyPos = (float[]) animation.getAnimatedValue();

view.setHeight(xyPos[0]); //通过属性值设置View属性动画

view.setWidth(xyPos[1]); //通过属性值设置View属性动画

}

});

##### Interpolators

对于补间动画：时间插值器（TimeInterpolator）的作用是根据时间流逝的百分比计算出动画进度的百分比。有了动画进度的百分比，就可以很容易的计算出动画开始的关键帧与将要显示的帧之间的差异（通过Transformation类的对象表示），下面展示TranslateAnimation类中如何根据动画进度的百分比计算出动画开始的关键帧与将要显示的帧之间的差异：

@Override

protected void applyTransformation(float interpolatedTime, Transformation t) {

float dx = mFromXDelta;

float dy = mFromYDelta;

if (mFromXDelta != mToXDelta) {

dx = mFromXDelta + ((mToXDelta - mFromXDelta) \* interpolatedTime);

}

if (mFromYDelta != mToYDelta) {

dy = mFromYDelta + ((mToYDelta - mFromYDelta) \* interpolatedTime);

}

t.getMatrix().setTranslate(dx, dy);

}

上面源码中applyTransformation方法的第一个参数就是通过时间插值器（TimeInterpolator）获取的动画进度的百分比。  
然后根据帧之间的差异绘制出将要显示的帧，以此类推从而形成动画的效果。

对于属性动画：时间插值器（TimeInterpolator）的作用是根据时间流逝的百分比计算出动画进度的百分比（即属性值改变的百分比）。

相关类解释：

● AccelerateDecelerateInterolator：先加速后减速。

● AccelerateInterpolator：加速。

● DecelerateInterpolator：减速。

● AnticipateInterpolator：先向相反方向改变一段再加速播放。

● AnticipateOvershootInterpolator：先向相反方向改变，再加速播放，会超出目标值然后缓慢移动至目标值，类似于弹簧回弹。

● BounceInterpolator：快到目标值时值会跳跃。

● CycleIinterpolator：动画循环一定次数，值的改变为一正弦函数：Math.sin(2 \* mCycles \* Math.PI \* input)。

● LinearInterpolator：线性均匀改变。

● OvershottInterpolator：最后超出目标值然后缓慢改变到目标值。

● TimeInterpolator：一个允许自定义Interpolator的接口，以上都实现了该接口。

举个例子，就像系统提供的标准API一样，如下就是加速插值器的实现代码，我们自定义时也可以类似实现：

//开始很慢然后不断加速的插值器。

public class AccelerateInterpolator implements Interpolator {

private final float mFactor;

private final double mDoubleFactor;

public AccelerateInterpolator() {

mFactor = 1.0f;

mDoubleFactor = 2.0;

}

......

//input 0到1.0。表示动画当前点的值，0表示开头，1表示结尾。

//return 插值。值可以大于1超出目标值，也可以小于0突破低值。

@Override

public float getInterpolation(float input) {

//实现核心代码块

if (mFactor == 1.0f) {

return input \* input;

} else {

return (float)Math.pow(input, mDoubleFactor);

}

}

}

综上可以发现，我们可以使用现有系统提供标准的东东实现属性动画，也可以通过自定义继承相关接口实现自己的动画，只要实现上面提到的那些主要方法即可。

##### ViewPropertyAnimator

Java属性动画拓展之ViewPropertyAnimator动画

在Android API 12时，View中添加了animate方法，具体如下：

public class View implements Drawable.Callback, KeyEvent.Callback,

AccessibilityEventSource {

......

/\*\*

\* This method returns a ViewPropertyAnimator object, which can be used to animate

\* specific properties on this View.

\*

\* @return ViewPropertyAnimator The ViewPropertyAnimator associated with this View.

\*/

public ViewPropertyAnimator animate() {

if (mAnimator == null) {

mAnimator = new ViewPropertyAnimator(this);

}

return mAnimator;

}

......

}

可以看见通过View的animate()方法可以得到一个ViewPropertyAnimator的属性动画（有人说他没有继承Animator类，是的，他是成员关系，不是之前那种继承关系）。

ViewPropertyAnimator提供了一种非常方便的方法为View的部分属性设置动画（切记，是部分属性），它可以直接使用一个 Animator对象设置多个属性的动画；在多属性设置动画时，它比 上面的ObjectAnimator更加牛逼、高效，因为他会管理多个属性的invalidate方法统一调运触发，而不像上面分别调用，所以还会有一些 性能优化。如下就是一个例子：

myView.animate().x(0f).y(100f).start();

##### LayoutAnimator

Java属性动画拓展之LayoutAnimator容器布局动画

Property动画系统还提供了对ViewGroup中View添加时的动画功能，我们可以用LayoutTransition对 ViewGroup中的View进行动画设置显示。LayoutTransition的动画效果都是设置给ViewGroup，然后当被设置动画的 ViewGroup中添加删除View时体现出来。该类用于当前布局容器中有View添加、删除、隐藏、显示等时候定义布局容器自身的动画和View的动 画，也就是说当在一个LinerLayout中隐藏一个View的时候，我们可以自定义 整个由于LinerLayout隐藏View而改变的动画，同时还可以自定义被隐藏的View自己消失时候的动画等。

我们可以发现LayoutTransition类中主要有五种容器转换动画类型，具体如下：

● LayoutTransition.APPEARING：当View出现或者添加的时候View出现的动画。

● LayoutTransition.CHANGE\_APPEARING：当添加View导致布局容器改变的时候整个布局容器的动画。

● LayoutTransition.DISAPPEARING：当View消失或者隐藏的时候View消失的动画。

● LayoutTransition.CHANGE\_DISAPPEARING：当删除或者隐藏View导致布局容器改变的时候整个布局容器的动画。

● LayoutTransition.CHANGE：当不是由于View出现或消失造成对其他View位置造成改变的时候整个布局容器的动画。

XML方式使用系统提供的默认LayoutTransition动画：

我们可以通过如下方式使用系统提供的默认ViewGroup的LayoutTransition动画：

android:animateLayoutChanges=”true”

在ViewGroup添加如上xml属性默认是没有任何动画效果的，因为前面说了，该动画针对于ViewGroup内部东东发生改变时才有效，所以当我们设置如上属性然后调运ViewGroup的addView、removeView方法时就能看见系统默认的动画效果了。

还有一种就是通过如下方式设置：

android:layoutAnimation=”@anim/customer\_anim”

通过这种方式就能实现很多吊炸天的动画。

Java方式使用系统提供的默认LayoutTransition动画：

在使用LayoutTransition时，你可以自定义这几种事件类型的动画，也可以使用默认的动画，总之最终都是通过 setLayoutTransition(LayoutTransition lt)方法把这些动画以一个LayoutTransition对象设置给一个ViewGroup。

譬如实现如上Xml方式的默认系统LayoutTransition动画如下：

mTransitioner = new LayoutTransition();

mViewGroup.setLayoutTransition(mTransitioner);

稍微再高端一点吧，我们来自定义这几类事件的动画，分别实现他们，那么你可以像下面这么处理：

mTransitioner = new LayoutTransition();

......

ObjectAnimator anim = ObjectAnimator.ofFloat(this, "scaleX", 0, 1);

......//设置更多动画

mTransition.setAnimator(LayoutTransition.APPEARING, anim);

......//设置更多类型的动画 mViewGroup.setLayoutTransition(mTransitioner);

到此通过LayoutTransition你就能实现类似小米手机计算器切换普通型和科学型的炫酷动画了。

#### 其他说明:

视图动画是通过**AnimationUtils**类的**loadAnimation()**方法获取xml文件相对应的**Animation**类实例，而属性动画则是通过**AnimatorInflater**类的**loadAnimation()**方法获取相应的**Animator**类实例

### 2.4过渡动画

##### 种类

arc\_motion.xml

auto\_transition.xml

change\_bounds.xml

change\_cip\_bounds.xml

change\_image\_transform.xml

change\_scroll.xml

change\_transform.xml

path\_motion.xml

pattern\_path\_motion.xml

recolor.xml

targets.xml

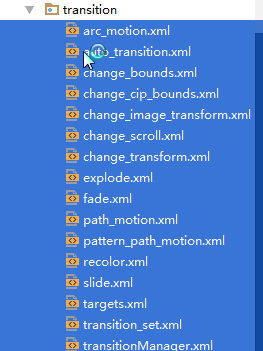
transition\_set.xml

transitionManager.xml

slide.xml

explode.xml

fade.xml



##### 内容变换

**第一步:**

当我们需要使用Transition进行Activity之间的动画变换时，首先需要设置：

|  |
| --- |
| //设置允许通过ActivityOptions.makeSceneTransitionAnimation发送或者接收Bundle getWindow().requestFeature(Window.FEATURE\_ACTIVITY\_TRANSITIONS); //设置使用TransitionManager进行动画，不设置的话系统会使用一个默认的TransitionManager getWindow().requestFeature(Window.FEATURE\_CONTENT\_TRANSITIONS); |

当然也可以使用xml文件来替代

|  |
| --- |
| <style name="MaterialAnimations" parent="@style/Theme.AppCompat.Light.NoActionBar"> ... //添加window开启Transtions动画属性 <item name="android:windowContentTransitions">true</item> //是否覆盖执行，其实可以理解成是否同步执行还是顺序执行 <item name="android:windowAllowEnterTransitionOverlap">false</item> <item name="android:windowAllowReturnTransitionOverlap">false</item> ... </style> |

*以下为常用的属性:*

*1<!-- 启用transtion动画 -->*

<**item name="android:windowContentTransitions"**>true</**item**>

*2<!-- 针对fragment控制 -->*

<**item name="android:fragmentEnterTransition"**>true</**item**>

<**item name="android:fragmentReturnTransition"**>true</**item**>

<**item name="android:fragmentExitTransition"**>true</**item**>

<**item name="android:fragmentReenterTransition"**>true</**item**>

<**item name="android:fragmentAllowEnterTransitionOverlap"**>true</**item**>

<**item name="android:fragmentAllowReturnTransitionOverlap"**>true</**item**>

<**item name="android:fragmentSharedElementEnterTransition"**>true</**item**>

<**item name="android:fragmentSharedElementReturnTransition"**>true</**item**>

*3<!-- 针对window控制content -->*

*同上(),会多一点点*

*4<!-- 针对activity控制 -->*

*<!-- activity执不执行动画，false不执行动画，设置什么都没用*

*matirial主题默认为true-->*

<**item name="android:windowActivityTransitions"**>true</**item**>

**第二步:**

Transition主要有退出(exit),进入(enter)，返回(return)，再次进入(reenter)四种变换，我们可以通过getWindow()然后进行设置

|  |
| --- |
| getWindow().setExitTransition(new Fade()); getWindow().setReenterTransition(new Explode()); getWindow().setEnterTransition(new Slide()); getWindow().setReturnTransition(new Fade()); |

注解:

1.ActivityA的退出变换(ExitTransition)决定了A调用B的时候，A中的View是如何播放动画的；

2.ActivityB的进入变换(EnterTransition)决定了A调用B的时候，B中的View是如何播放动画的；

3.ActivityB的返回变换(ReturnTransition)决定了在B返回A的时候，B中的View是如何播放动画的；

4.ActivityA的再次进入变换(ReenterTransition)决定了在B返回A的时候，A中的View是如何播放动画的。

然后我们可以在不同的状态变化中通过设置不同的效果，包括Fade(淡出)，Explode(分解)，以及Slide(滑动)；

官方提供了三种实践，Explode, Slide 和Fade(本来就有的,在api19,其他的都是新增加的)，分别是上下拉开，上下左右滑入滑出，以及淡出淡入等

* **explode(分解)** – 从屏幕中间进或出
* **slide(滑动)** - 从屏幕边缘进或出地
* **fade(淡出)** –通过改变屏幕上视图的不透明度达到添加或者移除视图的效果

注意:

如果我们没有对Activity设置进入或这退出动画，但是如果开启了Transition框架，则系统会默认为其设置Fade动画效果，如果设置了进入或退出，但是没有设置返回或者再次进入，则返回变换会使用设置的进入变换效果，再次进入会使用其退出的变换效果,(这几句话的意思就是进来的和出去的,或者出去的和进来的是同一个效果).

**第三步:**

|  |
| --- |
| *// 配置这个option必不可少 ActivityOptionsCompat options = ActivityOptionsCompat.makeSceneTransitionAnimation(activity,view,shareName); startActivity(intent,options.toBundle());* |

必须启用options.toBundle(),不然动画不起作用

##### 共享元素变换

可以是activity和fragment任何两者之间的

1.在你的主题中启用窗口内容过渡

2.在你的主题样式中指定共享元素的过渡

3.定义你的过渡动画为XML资源

4.使用android:transitionName属性给两个布局中的共享元素指定一个相同的名字（名字一定不要写错）

5.使用[ActivityOptions.makeSceneTransitionAnimation()](http://developer.android.com/reference/android/app/ActivityOptions.html#makeSceneTransitionAnimation%28android.app.Activity,) 方法

transition框架不仅可以实现在内容变换时的动画效果，同时还可以设置在两个界面中的共享元素变换。

在程序中使用共享元素的动画效果非常简单，只需要分别为两个需要共享的元素设置相同的transitionName，并在ActivityOptions. 中将需要共享的元素作为参数传递过去即可。

例如我们需要共享ActivityA和ActivityB中的两个Image图片，则只需要分别为两个Image在布局文件中设置相同的transitionName：

传递方法:通常的方法:

|  |
| --- |
| Intent intent = new Intent(MainActivityA.this, MainActivityB.class); ActivityOptions option = ActivityOptions.makeSceneTransitionAnimation(MainActivityA.this); startActivity(intent, option.toBundle()); |

如果我们只需要设置一个共享元素，那么在ActivityA中使用如下代码进行跳转：

|  |
| --- |
| Intent intent = new Intent(MainActivityA.this, MainActivityB.class); ActivityOptions option = ActivityOptions .makeSceneTransitionAnimation(MainActivityA.this,imageView,"share\_image"); startActivity(intent, option.toBundle()); |

如果我们需要多个共享的元素，可以通过Pair.create()(可变参数)进行创建，

|  |
| --- |
| ActivityOptions option = ActivityOptions.makeSceneTransitionAnimation(MainActivityA.this, Pair.create(imageView, "share\_image"), Pair.create(button, "share\_button")); |

注意:

需要注意的是Pair.create()中的两个参数，第一个参数为需要共享的View，第二个参数为共享的名称，即各自的transitionName；在使用的时候发现对于不同的view如Button，ImageView都需要将其强转为View，才能作为Pair.create()的第一个参数。

共享元素之fragment

|  |
| --- |
| // Transition for fragment1 Slide slideTransition = new Slide(Gravity.LEFT);  slideTransition.setDuration(getResources().getInteger(R.integer.anim\_duration\_long)); // Create fragment and define some of it transitions SharedElementFragment1 sharedElementFragment1 = SharedElementFragment1.newInstance(sample); // 下面这几个方法都是Fragment的方法，可见Google已经建议使用Fragment，进行共享元素跳转是相当好的体验 sharedElementFragment1.setReenterTransition(slideTransition);  sharedElementFragment1.setExitTransition(slideTransition); sharedElementFragment1.setSharedElementEnterTransition(new ChangeBounds());getSupportFragmentManager().beginTransaction() .replace(R.id.sample2\_content, sharedElementFragment1) .commit(); |

共享元素的效果种类:

1. changeBounds 对目标视图的大小进行动画 本来就有的api19,其他的都是5.0新增加的
2. changeClipBounds 对目标视图的剪裁大小进行动画
3. changeTransform 对目标视图进行缩放、旋转、位移动画
4. changeImageTransform 对目标图片进行缩放
5. [ChangeScroll](file:///D:\android_studio_sdk\docs\reference\android\transition\ChangeScroll.html) 滑动位置改变时

##### View动画(本界面)--Scenes

在一个界面内（Activity or Fragment）实现动画，就需要scenes

 scene = Scene.getSceneForLayout(container, R.layout.example, context);

其中container在Scene中是一个包含了所有view的ViewGroup。如果是在fragment中，Scene就是传入onCreateView()方法的参数。使用Transition的最简单方式就是使用TransitionManager处理，示例如下：

TransitionManager.go(scene);

如果在TransitionManager中不明确需要指定哪个Transition，就会默认使用AutoTransition

**AutoTransition**，事实证明，它只是TransitionSet的子类，只是它给自己定义了一个执行序列，分别是fading out、changing bounds、fading in。

View view = inflater.inflate(R.layout.example, container, false);

Scene scene = new Scene(container, (ViewGroup)view);

##### Reveal，揭示(展示)动画

##### 其他说明:

windowContentTransitions的作用使能的话就说明是要不要启用动画,这是副闸,控制单个fragment或者activity的.

windowActivityTransitions的作用是是的activity启用过渡功能,Material.theme默认已经启用了.控制所有的,这是总闸

## 3.动画说明:

1. 使用帧动画时，当图片数量较多且图片分辨率较大的时候容易出现OOM，需注意，尽量避免使用帧动画。
2. 使用无限循环的属性动画时，在Activity退出时即使停止，否则将导致Activity无法释放从而造成内存泄露。
3. View动画是对View的影像做动画，并不是真正的改变了View的状态，因此有时候会出现动画完成后View无法隐藏（setVisibility(View.GONE）失效），这时候调用view.clearAnimation()清理View动画即可解决。
4. 不要使用px，使用px会导致不同设备上有不同的效果。
5. View动画是对View的影像做动画，View的真实位置没有变动，也就导致点击View动画后的位置
6. 触摸事件不会响应，属性动画不存在这个问题。
7. 使用动画的过程中，使用硬件加速可以提高动画的流畅度。
8. 动画在3.0以下的系统存在兼容性问题，特殊场景可能无法正常工作，需做好适配工作。

## 4.第三方框架

### 1. NineOldAndroids:

https://github.com/JakeWharton/NineOldAndroids

|  |
| --- |
| shortHeight = getShortHeight();  longHeight = getLongHeight();  **if**(isOpen){  *//打开--->关闭*  isOpen = **false**;  **if**(shortHeight<longHeight){  valueAnimator = ValueAnimator.*ofInt*(longHeight,shortHeight);  }  }**else**{  *//关闭---->打开*  isOpen = **true**;  **if**(shortHeight<longHeight){  valueAnimator = ValueAnimator.*ofInt*(shortHeight,longHeight);  }  }  **if**(valueAnimator!=**null**){  valueAnimator.addUpdateListener(**new** AnimatorUpdateListener() {  @Override  **public void** onAnimationUpdate(ValueAnimator arg0) {  layoutParams.height = (Integer) arg0.getAnimatedValue();  des.setLayoutParams(layoutParams);  }  });  valueAnimator.addListener(**new** AnimatorListener() {  @Override  **public void** onAnimationStart(Animator arg0) {  }  @Override  **public void** onAnimationRepeat(Animator arg0) {  }  @Override  **public void** onAnimationEnd(Animator arg0) {  **if**(isOpen){  arrow.setBackgroundResource(R.drawable.arrow\_up);  }**else**{  arrow.setBackgroundResource(R.drawable.arrow\_down);  }  **final** ScrollView scrollView = (ScrollView) getScrollView();  *//触底滚动*  scrollView.post(**new** Runnable() {  @Override  **public void** run() {  scrollView.fullScroll(ScrollView.***FOCUS\_DOWN***);  }  });  }  @Override  **public void** onAnimationCancel(Animator arg0) {  }  });  valueAnimator.setDuration(300);  valueAnimator.start();  } |

### 2.其他第三方框架

学习动画过程中的网站:

https://github.com/hehonghui/android-tech-frontier 开发前线

http://www.tuicool.com/mobile 推酷

https://shapeshifter.design/ svg变换动画制作,需要配合svg制作网站使用,同时可以使用lotter

http://www.adobe.com/cn/products/illustrator.html矢量图工具网站

https://www.sketchapp.com/

https://github.com/alexjlockwood/ShapeShifter

//过度动画

http://www.jianshu.com/p/e497123652b5

https://github.com/toddway/MaterialTransitions 1115

https://github.com/lgvalle/Material-Animations 9000+

https://github.com/ImmortalZ/TransitionHelper 1300

https://github.com/asyl/ArcAnimator 1099

https://github.com/kaichunlin/android-transition 500+

https://github.com/Yellow5A5/ActSwitchAnimTool 500+

https://github.com/vikramkakkar/ImageTransition 500+

https://github.com/Cleveroad/BubbleAnimationLayout 500+

https://github.com/zhangke3016/ViewSpreadTranslationController 345

//基础动画知识

https://mp.weixin.qq.com/s?\_\_biz=MzI3OTU0MzI4MQ==&mid=2247484143&idx=2&sn=bd5eef0ce86e73abb79b3af2f69a415c&chksm=eb476e71dc30e76745880cdd5dbdf3025a63395828b6a8a60008556e7066e8690120e063843e&mpshare=1&scene=1&srcid=040589AEcKpS8HIjckk362iT#rd

//库

https://github.com/hitherejoe/animate 2444

https://github.com/JakeWharton/NineOldAndroids 9000+

https://github.com/Animatious/awesome-animation 1555

https://github.com/hehonghui/android-tech-frontier/tree/master/others/%E6%B7%B1%E5%85%A5%E6%B5%85%E5%87%BAAndroid%20%E6%96%B0%E7%89%B9%E6%80%A7-Transition-Part-1 讲解过度动画

https://github.com/willowtreeapps/spruce-android 列表动画 2600

https://github.com/florent37/ExpectAnim 跟随滚动动画 2000

https://github.com/daimajia/AndroidViewAnimations 给单个view设置动画 7000+

https://github.com/thunderrise/android-TNRAnimationHelper 给图片设置动画 768

https://github.com/geftimov/android-player 动画展示,设置view展示的顺序 830

https://github.com/takahirom/PreLollipopTransition 给5.0之前使用过度动画 1175

https://github.com/linfaxin/TransitionPlayer 实现给指定的图片以动画的形式进行变换 1184

https://github.com/airbnb/lottie-android 这个就牛B了. 学这个就可以了

https://www.lottiefiles.com/?page=6

//插值器

https://github.com/MartinRGB/RapidInterpolator 370

https://github.com/cimi-chen/EaseInterpolator 552

//效果

https://github.com/tslamic/FancyBackground 不得不说,这个挺好看的,在背景组合不同的资源一起动画展示 150

https://github.com/alexvasilkov/FoldableLayout 折叠布局,带有动画效果 1455 待看看是怎样的

https://github.com/UFreedom/FloatingView 浮起来的view,有动画效果

https://github.com/XunMengWinter/CircularAnim 1457

https://github.com/tumblr/Backboard 1400

https://github.com/florent37/ViewAnimator 1529

//svg --这个还需要另外看看

https://github.com/jaredrummler/AnimatedSvgView 实现给矢量图变换以动画的形式进行变换 1349

https://github.com/wnafee/vector-compat 1200 同上

https://github.com/ghuiii/AnimationWithoutCode 矢量图

https://github.com/glomadrian/RoadRunner 1051

https://github.com/nickbutcher/AnimatorDurationTile 1200 等待收录