# Android 动画分类

Android动画可以分为两类，最初的传统动画和Android3.0 之后出现的属性动画；

传统动画又包括帧动画（Frame Animation）和补间动画（Tweened Animation）。总之，有帧动画、补间动画、属性动画。

## 1.1 帧动画

帧动画是最容易实现的一种动画，这种动画更多的依赖于完善的UI资源，他的原理就是将一张张单独的图片连贯的进行播放。从而在视觉上产生一种动画的效果；有点类似于某些软件制作gif动画的方式。

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <animation-list xmlns:android="http://schemas.android.com/apk/res/android">  <item  android:drawable="@drawable/a\_0"  android:duration="100" />  <item  android:drawable="@drawable/a\_1"  android:duration="100" />  <item  android:drawable="@drawable/a\_2"  android:duration="100" />  </animation-list> |
| ImageView animationImg1 = (ImageView) findViewById(R.id.animation1);  animationImg1.setImageResource(R.drawable.frame\_anim1);  AnimationDrawable animationDrawable1 = (AnimationDrawable) animationImg1.getDrawable();  animationDrawable1.start(); |

android：oneshot="false" ，这个oneshot 的含义就是动画执行一次（true）还是循环执行多次。

//缺点：

需不需要特殊回收。

## 1.2 补间动画

补间动画又可以分为四种形式，分别是 alpha（淡入淡出），translate（位移），scale（缩放大小），rotate（旋转）。

补间动画的实现，一般会采用xml 文件的形式；代码会更容易书写和阅读，同时也更容易复用。

首先，在res/anim/ 文件夹下定义如下的动画实现方式

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <scale xmlns:android="http://schemas.android.com/apk/res/android"  android:duration="1000"  android:fromXScale="0.0"  android:fromYScale="0.0"  android:pivotX="50%"  android:pivotY="50%"  android:toXScale="1.0"  android:toYScale="1.0"/> |
| Animation animation = AnimationUtils.loadAnimation(mContext, R.anim.alpha\_anim);  img = (ImageView) findViewById(R.id.img);  img.startAnimation(animation); |

## 1.3属性动画

顾名思义它是对于对象属性的动画。因此，所有补间动画的内容，都可以通过属性动画实现。

private void RotateAnimation() {

ObjectAnimator anim = ObjectAnimator.ofFloat(myView, "rotation", 0f, 360f);

anim.setDuration(1000);

anim.start();

}

private void AlpahAnimation() {

ObjectAnimator anim = ObjectAnimator.ofFloat(myView, "alpha", 1.0f, 0.8f, 0.6f, 0.4f, 0.2f, 0.0f);

anim.setRepeatCount(-1);

anim.setRepeatMode(ObjectAnimator.REVERSE);

anim.setDuration(2000);

anim.start();

}

# 2.共享元素动画

它是Android 5.0新加入的一种过度动画，主要用于2个activity之间，可以共享某些控件，实现无缝转场的动画。应用场景之一就是现在的以图片、视频为主的内容流APP。

如何使用？

1.为共享元素指定统一的 transition name

2.启动 Activity 时带上共享元素参数

ShareElement即两个Activity（或Fragment）之间切换时的共享元素。

|  |
| --- |
| public class ContactsActivity extends Activity {  @Override  protected void onCreate(@Nullable Bundle savedInstanceState) {  /\*\*  \*1、打开FEATURE\_CONTENT\_TRANSITIONS开关(可选)，这个开关默认是打开的  \*/  requestWindowFeature(Window.FEATURE\_CONTENT\_TRANSITIONS);  /\*\*  \*2、设置除ShareElement外其它View的退出方式(左边滑出)  \*/  getWindow().setExitTransition(new Slide(Gravity.LEFT));  super.onCreate(savedInstanceState);  ...  }    @Override  public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {  ...  /\*\*  \*3、设置两个Activity的共享元素的TransitionName，  \*两个Activity的共享元素必须设置同样的TransitionName  \*/  ViewCompat.setTransitionName(avatarImg,"avatar:"+item.name);  ViewCompat.setTransitionName(nameTxt,"name:"+item.name);  }    private void gotoDetailActivity(Contacts contacts, final View avatarImg, final View nameTxt) {  Intent intent = new Intent(ContactActivity.this,DetailActivity.class);  Pair<View,String> pair1 = new  Pair<>((View)avatarImg,ViewCompat.getTransitionName(avatarImg));  Pair<View,String> pair2 = new  Pair<>((View)nameTxt,ViewCompat.getTransitionName(nameTxt));  /\*\*  \*4、生成带有共享元素的Bundle，这样系统才会知道这几个元素需要做动画  \*/  ActivityOptionsCompat activityOptionsCompat = ActivityOptionsCompat.makeSceneTransitionAnimation(ContactActivity.this, pair1, pair2);  ActivityCompat.startActivity(ContactActivity.this,intent,activityOptionsCompat.toBundle());  }  } |
| public class DetailActivity extends Activity {  @Override  protected void onCreate(@Nullable Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_detail);  ImageView avatarImg = findViewById(R.id.avatar);  TextView nameTxt = findViewById(R.id.name);  Contacts item = getIntent().getParcelableExtra(ContactsActivity.KEY\_CONTACTS);  /\*\*  \* 1、设置相同的TransitionName  \*/  ViewCompat.setTransitionName(avatarImg,"avatar:"+item.name);  ViewCompat.setTransitionName(nameTxt,"name:"+item.name);  /\*\*  \* 2、设置WindowTransition,除指定的ShareElement外，其它所有View都会执行这个Transition动画  \*/  getWindow().setEnterTransition(new Fade());  getWindow().setExitTransition(new Fade());  /\*\*  \* 3、设置ShareElementTransition,指定的ShareElement会执行这个Transiton动画  \*/  TransitionSet transitionSet = new TransitionSet();  transitionSet.addTransition(new ChangeBounds());  transitionSet.addTransition(new ChangeTransform());  transitionSet.addTarget(avatarImg);  transitionSet.addTarget(nameTxt);  getWindow().setSharedElementEnterTransition(transitionSet);  getWindow().setSharedElementExitTransition(transitionSet);  }  } |

# 3.HenCoder 属性动画

动画：你就是把内容的两个状态进行平滑的过渡。

简单的线程控制 --> 复杂的变化（加速、减速）、监听器等等。

3.1 所以，我们可以使用线程中按时间依次设置控件的属性。当然，这个是很麻烦。不过本质就是如此。android系统也是帮我们做好了动画的api。

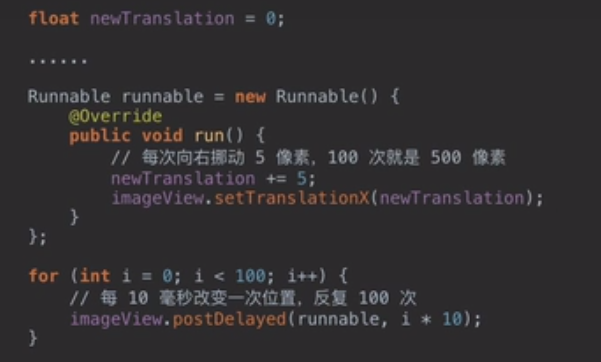
ViewPropertyAnimator

view.animate().setXxx();

设置时长

设置速度模型：插值器

设置监听器



3.2 使用ObjectAnimator 做属性动画，本质是是会调用属性名对应的setter方法。





在自定义控件中，要注意的。使用属性动画的时候。

ObjectAnimator

如果给自定义控件添加属性动画，要添加setter方法。

每个时间点，它的时间完成度会转化为动画完成来实现的。

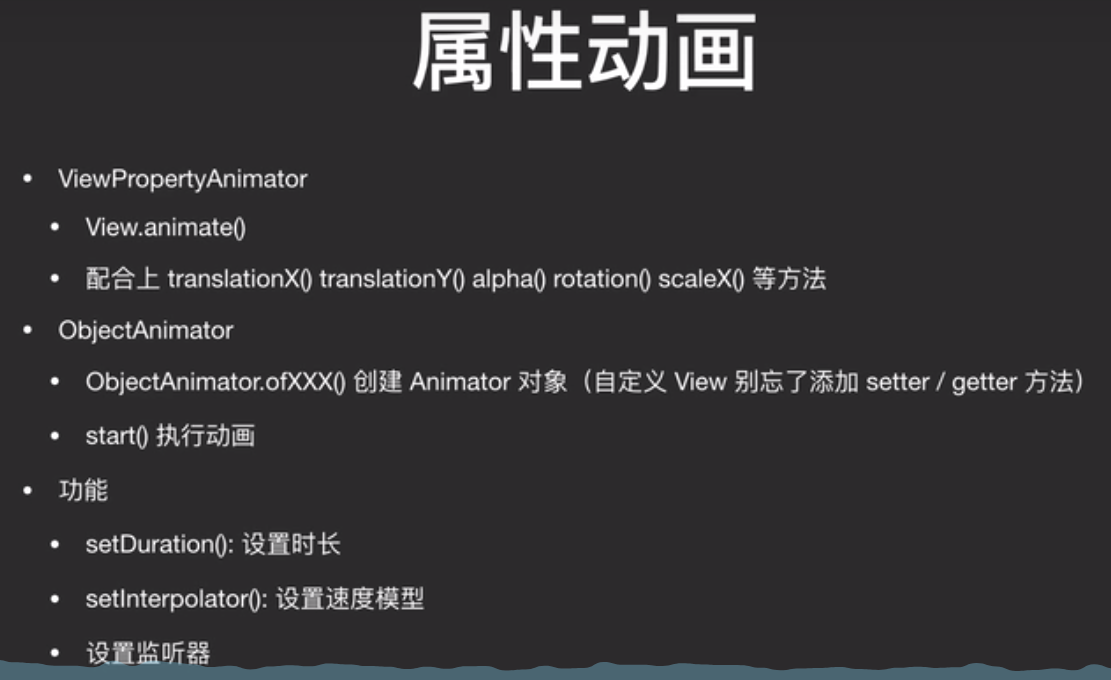
TypeEvaluator: 动画完成度 --> 属性值。

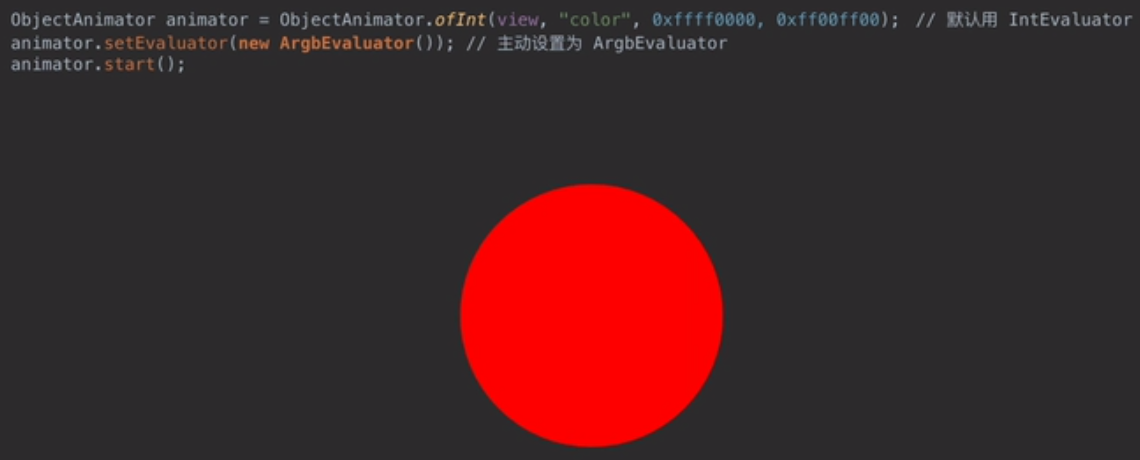




invalidate(); 将当前的显示标记为失效，然后到下一帧的时候，就会使用新的值，进行重绘。

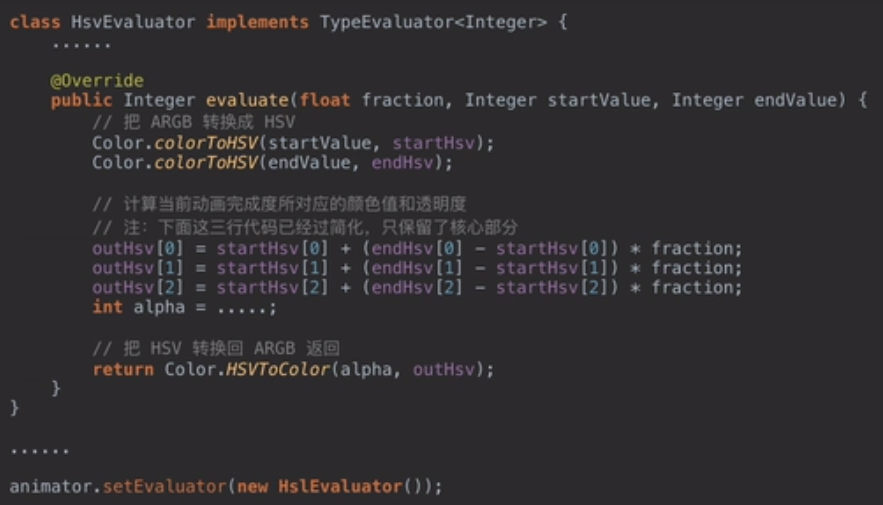
小结





每个时间点，它的时间完成度会转化为动画完成来实现的。

TypeEvaluator: 动画完成度 --> 属性值。





# ViewPropertyAnimator

使用方式：View.animate() 后跟 translationX() 等方法，动画会自动执行。

view.animate().translationX(500);



从图中可以看到， View 的每个方法都对应了 ViewPropertyAnimator 的两个方法，其中一个是带有 -By 后缀的，例如，View.setTranslationX() 对应了

ViewPropertyAnimator.translationX() 和 ViewPropertyAnimator.translationXBy() 这两个方法。其中带有 -By() 后缀的是增量版本的方法，例如，translationX(100) 表示用动画把 View 的 translationX 值渐变为 100，而 translationXBy(100) 则表示用动画把 View 的 translationX 值渐变地增加100。

# 5.ObjectAnimator

使用方式：

如果是自定义控件，需要添加 setter / getter 方法；

用 ObjectAnimator.ofXXX() 创建 ObjectAnimator 对象；

用 start() 方法执行动画。

|  |
| --- |
| public class SportsView extends View {    float progress = 0;    ......    // 创建 getter 方法  public float getProgress() {  return progress;  }  // 创建 setter 方法  public void setProgress(float progress) {  this.progress = progress;  invalidate();  }    @Override  public void onDraw(Canvas canvas) {  super.onDraw(canvas);    ......    canvas.drawArc(arcRectF, 135, progress \* 2.7f, false, paint);    ......  }  }  ......  // 创建 ObjectAnimator 对象  ObjectAnimator animator = ObjectAnimator.ofFloat(view, "progress", 0, 65);  // 执行动画  animator.start(); |

1. setDuration(int duration) 设置动画时长

单位是毫秒。

// imageView1: 500 毫秒

imageView1.animate()

.translationX(500)

.setDuration(500);

// imageView2: 2 秒

ObjectAnimator animator = ObjectAnimator.ofFloat(

imageView2, "translationX", 500);

animator.setDuration(2000);

animator.start();

2. setInterpolator(Interpolator interpolator) 设置 Interpolator 插值器

视频里已经说了， Interpolator 其实就是速度设置器。你在参数里填入不同的 Interpolator ，动画就会以不同的速度模型来执行。

// imageView1: 线性 Interpolator，匀速

imageView1.animate()

.translationX(500)

.setInterpolator(new LinearInterpolator());

// imageView: 带施法前摇和回弹的 Interpolator

ObjectAnimator animator = ObjectAnimator.ofFloat(

imageView2, "translationX", 500);

animator.setInterpolator(new AnticipateOvershootInterpolator());

animator.start();

AccelerateDecelerateInterpolator 先加速再减速。这是默认的 Interpolator。

LinearInterpolator匀速。

AccelerateInterpolator持续加速。

DecelerateInterpolator持续减速直到 0。

AnticipateInterpolator先回拉一下再进行正常动画轨迹。效果看起来有点像投掷物体或跳跃等动作前的蓄力。

3. 设置监听器

给动画设置监听器，可以在关键时刻得到反馈，从而及时做出合适的操作，例如在动画的属性更新时同步更新其他数据，或者在动画结束后回收资源等。

设置监听器的方法， ViewPropertyAnimator 和 ObjectAnimator 略微不一样： ViewPropertyAnimator 用的是 setListener() 和 setUpdateListener() 方法，可以设置一个监听器，要移除监听器时通过 set[Update]Listener(null) 填 null 值来移除；而 ObjectAnimator 则是用 addListener() 和 addUpdateListener() 来添加一个或多个监听器，移除监听器则是通过 remove[Update]Listener() 来指定移除对象。

另外，由于 ObjectAnimator 支持使用 pause() 方法暂停，所以它还多了一个 addPauseListener() / removePauseListener() 的支持；而 ViewPropertyAnimator 则独有 withStartAction() 和 withEndAction() 方法，可以设置一次性的动画开始或结束的监听。

3.1 ViewPropertyAnimator.setListener() / ObjectAnimator.addListener()

这两个方法的名称不一样，可以设置的监听器数量也不一样，但它们的参数类型都是 AnimatorListener，所以本质上其实都是一样的。 AnimatorListener 共有 4 个回调方法：

3.1.1 onAnimationStart(Animator animation)

当动画开始执行时，这个方法被调用。

3.1.2 onAnimationEnd(Animator animation)

当动画结束时，这个方法被调用。

3.1.3 onAnimationCancel(Animator animation)

当动画被通过 cancel() 方法取消时，这个方法被调用。

需要说明一下的是，就算动画被取消，onAnimationEnd() 也会被调用。所以当动画被取消时，如果设置了 AnimatorListener，那么 onAnimationCancel() 和 onAnimationEnd() 都会被调用。onAnimationCancel() 会先于 onAnimationEnd() 被调用。

3.1.4 onAnimationRepeat(Animator animation)

当动画通过 setRepeatMode() / setRepeatCount() 或 repeat() 方法重复执行时，这个方法被调用。

由于 ViewPropertyAnimator 不支持重复，所以这个方法对 ViewPropertyAnimator 相当于无效。

3.2 ViewPropertyAnimator.setUpdateListener() / ObjectAnimator.addUpdateListener()

和上面 3.1 的两个方法一样，这两个方法虽然名称和可设置的监听器数量不一样，但本质其实都一样的，它们的参数都是 AnimatorUpdateListener。它只有一个回调方法：onAnimationUpdate(ValueAnimator animation)。

3.2.1 onAnimationUpdate(ValueAnimator animation)

当动画的属性更新时（不严谨的说，即每过 10 毫秒，动画的完成度更新时），这个方法被调用。

方法的参数是一个 ValueAnimator，ValueAnimator 是 ObjectAnimator 的父类，也是 ViewPropertyAnimator 的内部实现，所以这个参数其实就是 ViewPropertyAnimator 内部的那个 ValueAnimator，或者对于 ObjectAnimator 来说就是它自己本身。

ValueAnimator 有很多方法可以用，它可以查看当前的动画完成度、当前的属性值等等。不过 ValueAnimator 是下一期才讲的内容，所以这期就不多说了。

3.3 ObjectAnimator.addPauseListener()

由于 ObjectAnimator.pause() 是下期的内容，所以这个方法在这期就不讲了。当然，如果你有兴趣的话，现在就了解一下也可以。

4. TypeEvaluator 估值器

ArgbEvaluator如视频中的例子，TypeEvaluator 最经典的用法是使用 ArgbEvaluator 来做颜色渐变的动画。

ObjectAnimator animator = ObjectAnimator.ofInt(view, "color", 0xffff0000, 0xff00ff00);

animator.setEvaluator(new ArgbEvaluator());

animator.start();

另外，在 Android 5.0 （API 21） 加入了新的方法 ofArgb()，所以如果你的 minSdk 大于或者等于 21（哈哈哈哈哈哈哈哈），你可以直接用下面这种方式：

ObjectAnimator animator = ObjectAnimator.ofArgb(view, "color", 0xffff0000, 0xff00ff00);

animator.start();

1. ofObject()

借助于 TypeEvaluator，属性动画就可以通过 ofObject() 来对不限定类型的属性做动画了。方式很简单：

为目标属性写一个自定义的 TypeEvaluator

使用 ofObject() 来创建 Animator，并把自定义的 TypeEvaluator 作为参数填入。

1. 插值器和估值器关系

属性动画是对属性做动画，属性要实现动画。

首先由插值器根据时间流逝的百分比计算出当前属性值改变的百分比，然后由插值器将这个百分比返回。这个时候插值器的工作就完成了。

插值器算好属性变化百分比之后，由估值器根据当前属性改变的百分比来计算改变后的属性值，根据这个属性值，我们就可以对View设置当前的属性值了。