属性动画分析与使用

## 引言：

在讲述属性动画之前，先引入一个问题：如何以动画的形式，让Button的宽度在1000ms内利由40dp增长到100dp，如图1.1所示：



图1.1 目标缩放动效

首先，我们会想到利用View Animation中的ScaleAnimation来实现，但是，ScaleAnimation所做的是一个整体的缩放，虽然能改变Button的宽度，同时也会将字体进行拉伸，其动画效果就会如下图1.2所示：



图1.2 利用View Animation实现的缩放效果

其次，虽然ScaleAnimation好像改变了View的宽度，但其实不过是Android系统将变形后的Button在屏幕上绘画出来，实际的Button宽度、位置等属性信息并未发生变化，在动效结束的时候，为了保持最终状态，必须手动设置最终状态。那么如何实现上述动效，同时保证动画结束的时可以不用手动设置最终状态呢？这就是属性动画所要解决的问题。

## 1. 属性动画概述

#### 1.1 Android动画简介与分类

Android动画大致可分为三种：绘图动画（Drawable Animation）、视图动画（View Animation）、属性动画（Property Animation）。绘图动画主要是用来设置一些动态改变的背景，用途比较局限，不在分析范围；View Animation能实现缩放、平移、旋转、透明度渐变等基本动画，用途相对广泛，但视图动画仍具有一定的局限性，比如：视图动画只是不断的在屏幕上重绘View，但是View自身的属性并未发生改变，比如平移操作，虽然看到View平移，但是View自身的位置属性并未发生改变，动画结束时必须手动设置最终位置、或者其他状态。其次，View动画也只能用于缩放、旋转、平移、改变透明度，但是对于View其他的属性束手无策，属性动画不仅兼容视图动画，还能解决许多视图动画无法解决的问题，可以看做更加广义的动画。

#### 1.2 属性动画概述：

顾名思义，属性动画是面向对象属性的动画，可以将对象的任何属性时定义为动画，不再局限于View的平移（Translate）、旋转（Rotate）、缩放（Scale）、不透明度（Alpha）。属性动画不再局限于View，将动画的范围扩展到任何对象，无论对象是否在前台显示，都可以实时改变其属性值，因此是更加广义的动画。同时，由于属性动画可以直接操作对象的属性，能够实时改变对象的属性值，因此，在动画结束时可以不用设置对象的最终状态。

属性动画的用法大致与视图动画类似，必须定义相应的属性，比如：变化区间、持续时长、采用的插值器类型等。如果不加定义，则采用默认值，比如：默认持续时长是300ms，刷新频率是10ms等。

#### 1.3属性动画框架与原理：

属性动画框架在Android3.0中引入，是一个健壮的框架，它几乎可以把任何对象变成动画。属性动画的目标与宗旨是：能够根据时间的推移来实时改变任何对象的属性值，而不用关心该对象是否要绘制在屏幕上。属性动画框架主要包含几个部分：动画执行类、动画监听机类、计算估值器、动画插值器

* 动画执行类：

[ValueAnimator](http://developer.android.com/reference/android/animation/ValueAnimator.html)、[ObjectAnimator](http://developer.android.com/reference/android/animation/ObjectAnimator.html)、AnimatorSet。[ValueAnimator](http://developer.android.com/reference/android/animation/ValueAnimator.html)是最常用的属性动画时间引擎，ValueAnimator本身不作用于任何对象，主要用于对一个值做动画，根据当前动画进度提供计算值，至于如何利用这个值，则完全不关心。[ObjectAnimator](http://developer.android.com/reference/android/animation/ObjectAnimator.html)：顾名思义，是面向对象的[ValueAnimator](http://developer.android.com/reference/android/animation/ValueAnimator.html)，需要指定特定的操作对象及相应属性，这样Android系统便可以自动更新对象的属性。AnimatorSet是属性动画集合，主要用于整合多个动画，比如先后执行、并发执行等。

* TypeEvaluator 类型估值：

主要用于实时计算动画操作属性的值。常用的有[IntEvaluator](http://developer.android.com/reference/android/animation/IntEvaluator.html)、[FloatEvaluator](http://developer.android.com/reference/android/animation/FloatEvaluator.html)。[IntEvaluator](http://developer.android.com/reference/android/animation/IntEvaluator.html)用于Int型属性的计算，比如旋转的度数、移动的距离，[FloatEvaluator](http://developer.android.com/reference/android/animation/FloatEvaluator.html)用于Float型属性的计算，比如Alpha值（0-1之间）。

* TimeInterpolator 时间插值器：

用作动画的执行函数模型，线性、加速、减速等

|  |  |
| --- | --- |
| [AccelerateDecelerateInterpolator](http://developer.android.com/reference/android/view/animation/AccelerateDecelerateInterpolator.html) | [AccelerateInterpolator](http://developer.android.com/reference/android/view/animation/AccelerateInterpolator.html) |
| [AnticipateInterpolator](http://developer.android.com/reference/android/view/animation/AnticipateInterpolator.html) | [AnticipateOvershootInterpolator](http://developer.android.com/reference/android/view/animation/AnticipateOvershootInterpolator.html) |
| [BounceInterpolator](http://developer.android.com/reference/android/view/animation/BounceInterpolator.html) | [CycleInterpolator](http://developer.android.com/reference/android/view/animation/CycleInterpolator.html) |
| [DecelerateInterpolator](http://developer.android.com/reference/android/view/animation/DecelerateInterpolator.html) | [LinearInterpolator](http://developer.android.com/reference/android/view/animation/LinearInterpolator.html) |
| [OvershootInterpolator](http://developer.android.com/reference/android/view/animation/OvershootInterpolator.html) |  |

* Animation Listeners 动画监听机制：

Animation Listeners主要用于监听动画的执行状态，并根据需要，执行相应的的操作。同视图动画不同，属性动画存在两种类型监听：

* 第一种：[Animator.AnimatorListener](http://developer.android.com/reference/android/animation/Animator.AnimatorListener.html)（用于“宏观”监听）开始、结束、重复、取消

[onAnimationStart()](http://developer.android.com/reference/android/animation/Animator.AnimatorListener.html#onAnimationStart(android.animation.Animator)) – 开始时调用

[onAnimationEnd()](http://developer.android.com/reference/android/animation/Animator.AnimatorListener.html#onAnimationEnd(android.animation.Animator)) –动画结束时调用

[onAnimationRepeat()](http://developer.android.com/reference/android/animation/Animator.AnimatorListener.html#onAnimationRepeat(android.animation.Animator)) – 动画重复时调用

[onAnimationCancel()](http://developer.android.com/reference/android/animation/Animator.AnimatorListener.html#onAnimationCancel(android.animation.Animator)) – 动画取消时调用

* 第二种：[ValueAnimator.AnimatorUpdateListener](http://developer.android.com/reference/android/animation/ValueAnimator.AnimatorUpdateListener.html)（用于进度监听）

设置监听后，动画的每一帧更新是都会调用[onAnimationUpdate()](http://developer.android.com/reference/android/animation/ValueAnimator.AnimatorUpdateListener.html#onAnimationUpdate(android.animation.ValueAnimator)) ，之后，可以通过调用 [ValueAnimator](http://developer.android.com/reference/android/animation/ValueAnimator.html)的 [getAnimatedValue()](http://developer.android.com/reference/android/animation/ValueAnimator.html#getAnimatedValue()) 方法获取当前动画的进度、属性值。

属性动画工作原理：

创建属性动画时，每个对象都包含一个ValueAnimator，并通过监控机制AnimatorUpdateListner实时监控动画的执行，并可以计算出运行了多场时间、当前相应属性的值等，比如（X），之后，可以根据需求更新相应的对象。最直观的就是利用[ObjectAnimator](http://developer.android.com/reference/android/animation/ObjectAnimator.html)做动画时，目标对象需要提供相应属性的getter和setter方法，系统会根据属性的初始值和最终值，以动画的效果多次去调用set方法，随着时间的推移，set的值会越来越接近最终值。

#### 1.4属性动画与视图动画的异同

表1：属性动画与视图动画对照表

|  |  |  |
| --- | --- | --- |
| 对照点 | 属性动画 | 视图动画 |
| 可操作对象类型 | 任何Object | 仅限于View |
| 能否实时监听动画进度 | YES | NO |
| 能否实时改变对象属性 | YES | NO |
| 动画类型限制 | 无限制 | 仅限平移、缩放、透明、旋转 |
| 实现复杂度（代码量） | 复杂，代码量大 | 简单，代码量少 |
| 播放完毕后是否需校正 | NO | YES |
| 支持XML加载的方式 | YES | YES |
| 支持布局动画 | YES | NO |

## 2. 属性动画的用法

从属性动画框架可以知：想要合理的使用属性动画，必须正确的利用动画的几大元素：动画执行类、动画监听、估值器、插值器，合理的属性动画工具能够帮助我们更加方便的达成目标。

#### 2.1 利用ValueAnimator实现属性动画

ValueAnimator本身不作用于任何对象，它可以对一个值做动画，然后我们可以监听其动画过程，在动画过程中修改我们的对象的属性值。ValueAnimator的限制较小，不过需要自己通过监听来实时改变对象的属性值。使用方式：

第一步：通过 ValueAnimator静态方法获取对象[ofInt()](http://developer.android.com/reference/android/animation/ValueAnimator.html#ofInt(int...)), [ofFloat()](http://developer.android.com/reference/android/animation/ValueAnimator.html#ofFloat(float...)), or [ofObject()](http://developer.android.com/reference/android/animation/ValueAnimator.html#ofObject(android.animation.TypeEvaluator, java.lang.Object...)).，并设置动画持续时长。

ValueAnimator animation = ValueAnimator.ofFloat(0f, 1f);  
animation.setDuration(1000);

第二步：设置实时监听。[ValueAnimator.AnimatorUpdateListener](http://developer.android.com/reference/android/animation/ValueAnimator.AnimatorUpdateListener.html)并实现其[onAnimationUpdate()](http://developer.android.com/reference/android/animation/ValueAnimator.AnimatorUpdateListener.html#onAnimationUpdate(android.animation.ValueAnimator)) 。在该函数中，可以获取动画进度、以及属性的实时值，但该实时值并未直接作用于任何对象，如果想要作用于某个对象，必须用于将属性值实时设置到对象。 在[onAnimationUpdate()](http://developer.android.com/reference/android/animation/ValueAnimator.AnimatorUpdateListener.html#onAnimationUpdate(android.animation.ValueAnimator)) 内部，可以通过 [getAnimatedValue()](http://developer.android.com/reference/android/animation/ValueAnimator.html#getAnimatedValue()). 获取当前计算到的值，之后通过对象的set等方法，实时更新对象，比如 [setAlpha()](http://developer.android.com/reference/android/view/View.html#setAlpha(float)) 等：

    animation.addUpdateListener(new AnimatorUpdateListener() {

        @Override

        public void onAnimationUpdate(ValueAnimator animator) {

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

//计算当前进度占整个动画过程的比例，浮点型，0-1之间  然后再设给Object（比如View）

targetView.setAlpha(currentValue );

 }  });

第三步：执行动画，之后[ValueAnimator](http://developer.android.com/reference/android/animation/ValueAnimator.html) 就开始利用估值器已经插值器计算属性值，并实时利用监听接口更新对象。

animation.start();

#### 2.2 利用ObjectAnimator实现属性动画

ObjectAnimator是[ValueAnimator](http://developer.android.com/reference/android/animation/ValueAnimator.html) 的子类，顾名思义ObjectAnimator与具体的对象绑定，也就说在ObjectAnimator创建之初，动画所要操作的对象、及属性都是可以设定的。动画执行过程中，Android系统会自动调用对象的setter方法更新Object相应的属性，省去了设置 [ValueAnimator.AnimatorUpdateListener](http://developer.android.com/reference/android/animation/ValueAnimator.AnimatorUpdateListener.html)的麻烦，实现起来相对简单。

第一步：利用ObjectAnimator静态方法构造对象，同时要制定target对象、以及要操作的属性，比如：不透明度（alpha）：

ObjectAnimator anim = ObjectAnimator.ofFloat(foo, "alpha", 0f, 1f);  
anim.setDuration(1000);

第二步：执行动画

anim.start();

但是，由于ObjectAnimator会利用对象的setter方法自动更新相应的属性，该对象必须具有setter方法,如果实现动画的时候没有传递初始值，那么还要提供getter方法，否则无法更新。例如：属性是color，则必须具有setterColor/getterColor方法，否则动画无效。如果不存在相应属性的操作方法，则可以通过wrapper 方式间接为其提供getter和setter方法。例如：View没有直接设置宽度的方法width，可以利用wrapper实现setWidth，getWidth：

public class ViewWrapper {

    private View mTarget;

    public ViewWrapper(View target) {

        mTarget = target;

}

public int getWidth() {

        return mTarget.getLayoutParams().width;

    }

public void setWidth(int width) {

        mTarget.getLayoutParams().width = width;

        mTarget.requestLayout();

    }  }

之后便可使用该封装类进行属性动画：

private void performAnimate(View view) {

    ViewWrapper wrapper = new ViewWrapper(view);

    ObjectAnimator.ofInt(wrapper, "width", 500).setDuration(5000).start();

}

#### 2.3 利用AnimatorSet实现属性动画

开发中，经常会遇到这种情形：播放一个动画依赖另一个动画是否播放完成。在属性动画中框架中，可以利用AnimatorSet构造属性动画集，来整合多个动画的播放，控制动画的并发、顺序逻辑等。AnimatorSet是Animator的子类，不仅可以作为动画使用，而且可以实现嵌套。使用方法：

ObjectAnimator anim1= ObjectAnimator.ofFloat(foo, "alpha" …);

ObjectAnimator anim2= ObjectAnimator.ofFloat(foo, "x" …);

ObjectAnimator anim3= ObjectAnimator.ofFloat(foo, "y" … );

ObjectAnimator anim4= ObjectAnimator.ofFloat(foo, …);

ObjectAnimator anim5= ObjectAnimator.ofFloat(foo, …);

AnimatorSet bouncer = new AnimatorSet();  
bouncer.play(anim1).before(anim2);  
bouncer.play(anim2).with(anim3);  
bouncer.play(anim3).with(anim4);  
bouncer.play(bounceBackAnim).after(stretchAnim2);

//集合与元素的嵌套使用  
AnimatorSet animatorSet = new AnimatorSet();  
animatorSet.play(bouncer).before(anim5);  
animatorSet.start();  
}

#### 2.4 利用[LayoutTransition](http://developer.android.com/reference/android/animation/LayoutTransition.html) 实现布局属性动画：

属性动画允许布局通过动画来显示变化，比如新View元素添加到LinearLayout中时，新元素可以定义显示动画，旧元素可以定义移动动画，布局动画更像是childViews属性动画集。布局属性动画主要包含以下几种元素：

* + APPEARING  新添加的View本身的出现动画；
  + DISAPPEARING View消失动画；
  + CHANGE\_APPEARING 由于新增了其他View而需要改变位置的动画；
  + CHANGE\_DISAPPEARING 由于移除了其他View而需要改变位置的动画。

private void setupTransition(LayoutTransition transition) {

customDisappearingAnim = ObjectAnimator.ofFloat(null, "rotationX", 0f, 90f)

customAppearingAnim = ObjectAnimator.ofFloat(null, "rotationY", 90f, 0f)

…

transition.setAnimator(LayoutTransition.DISAPPEARING, customDisappearingAnim);

transition.setAnimator(LayoutTransition.CHANGE\_APPEARING, customChangingAppearingAnim);

transition.setAnimator(LayoutTransition. APPEARING , customAnimCB);

transition.setAnimator(LayoutTransition.CHANGE\_APPEARING, customChangingAppearingAnim);

}

2.5 利用XML实现属性动画。

同视图动画类似，属性动画也可利用XML定义，不过放置的位置与视图动画不同，属性动画的XML文件放在res/animator目录下。XML属性动画的使用分两步

第一步：定义相应动画

<set android:ordering="sequentially">  
    <set>  
        <objectAnimator  
            android:propertyName="x"  
            android:duration="500"  
            android:valueTo="400"  
            android:valueType="intType"/>  
        <objectAnimator  
            android:propertyName="y"  
            android:duration="500"  
            android:valueTo="300"  
            android:valueType="intType"/>  
    </set>  
    <objectAnimator  
        android:propertyName="alpha"  
        android:duration="500"  
        android:valueTo="1f"/>  
</set>

第二步：在Java代码中利用AnimatorInflater加载，渲染既可

AnimatorSet set = (AnimatorSet) AnimatorInflater.loadAnimator(myContext, R.anim.property\_animator);  
set.setTarget(myObject);  
set.start();

## 3. 属性动画总结

属性动画与视图动画最大的区别是：属性动画能够实时改变Object的属性，而视图动画无法做到，并且属性动画应用的范围更广，限制更小。很多场合下，如果视图动画无法满足需求，不放考虑属性动画，然而功能的强大导致了实现的复杂度提升，所以如果视图动画足够满足需求，可以不用视图动画。