**Android**动画知识总结

Android中，动画可以分为三种：补间动画（**View Animation**）、属性动画（**Property Animation**）和帧动画（**Drawable Animation**）

一、补间动画（**View Animation**）

View Animation相对简单，不过只能支持简单的缩放、平移、旋转、透明度基本的动画，且有一定的局限性。比如：你希望View有一个颜色的切换动画；你希望可以使用3D旋转动画；你希望当动画停止时，View的位置就是当前的位置；这些View Animation都无法做到。简单来说，补间动画只能用来View。

它有四种典型的变换效果，分别是：

（1）平移动画

（2）缩放动画

（3）旋转动画

（4）透明度动画

这四种动画效果可以通过java代码的方式动态的创建，也可以通过xml文件来创建。

**1.** 首先看一下通过**java**代码的创建方式：

**1.1** 平移动画

平移动画是通过TranslateAnimation类来实现的，常用的构造函数有以下两个：

1. public TranslateAnimation(float fromXDelta, float toXDelta, float fromYDelta, float toYDelta){}
2. public TranslateAnimation(int fromXType, float fromXValue, int toXType, float toXValue,

int fromYType, float fromYValue, int toYType, float toYValue) {}

先说第一个构造函数，参数 fromXDelta、toXDelta 代表x方向平移的起始值和结束值，单位为像素，若 toXDelta 减 fromXDelta 大于0，则View右移，否则左移。fromYDelta、toYDelta是同样的道理，差值大于0 View下移，否则上移。

要实现一个View右移100像素可以这么做：

TranslateAnimation translateAnimation = new TranslateAnimation(0f, 0f, 0f, 100f);

translateAnimation.setDuration(2000); //动画的持续时间，单位毫秒

translateAnimation.setFillAfter(true); //参数为true表示动画结束后View停留在结束为止

view.startAnimation(translateAnimation); //开始动画

再看第二个构造函数，在x方向上，fromXType、toXType有三种类型：Animation.ABSOLUTE、Animation.RELATIVE\_TO\_SELF、Animation.RELATIVE\_TO\_PARENT，分别代表绝对像素、相对于自身平移、相对于父View平移。fromXValue、toXValue，当type为Animation.ABSOLUTE时，这个两个值为具体的像素值，当type为Animation.RELATIVE\_TO\_SELF或Animation.RELATIVE\_TO\_PARENT，这个两个值为比例值，取值范围是[0f, 1.0f]， y方向上同理。

具体的用法如下：

TranslateAnimation translateAnimation = new TranslateAnimation(Animation.RELATIVE\_TO\_SELF, 0f,

Animation.RELATIVE\_TO\_SELF, 0f, Animation.RELATIVE\_TO\_SELF, 0f, Animation.RELATIVE\_TO\_SELF, -1.0f);

translateAnimation.setDuration(2000);

translateAnimation.setFillAfter(true);

view.startAnimation(translateAnimation);

此时type都是Animation.RELATIVE\_TO\_SELF，toYValue的值是-1.0f，即100%，此时则view向上平移自身高度的距离，即就是常见的隐藏title的效果。当type为Animation.RELATIVE\_TO\_PARENT时，则view向上平移父view高度距离。

**1.2** 缩放动画

缩放动画是通过ScaleAnimation类实现的，常用构造函数如下：

public ScaleAnimation(float fromX, float toX, float fromY, float toY,

int pivotXType, float pivotXValue, int pivotYType, float pivotYValue) {}

x方向上，参数fromX、toX分别代表view在水平方向缩放的起始比例和结束比例，都是大于等于0的浮点数。pivotXType代表缩放类型，有三种Animation.ABSOLUTE,、Animation.RELATIVE\_TO\_SELF、Animation.RELATIVE\_TO\_PARENT，pivotXValue代表缩放的中心点，可以是具体的像素值，可以是比比例值，比例值范围是[0f, 1.0f]，比例值是常用的，y方向上同理。

具体用法如下：

ScaleAnimation scaleAnimation = new ScaleAnimation(1.0f, 2f, 1.0f, 0.5f,

Animation.RELATIVE\_TO\_SELF, 0.5f, Animation.RELATIVE\_TO\_SELF, 0.5f);

scaleAnimation.setDuration(2000);

scaleAnimation.setFillAfter(true);

view.startAnimation(scaleAnimation);

实现了view相对于自身中心，在x方向拉伸为原来2倍，在y方向缩小为原来0.5倍。

**1.3** 旋转动画

旋转动画是通过RotateAnimation实现的，常用构造函数如下：

public RotateAnimation(float fromDegrees, float toDegrees, int pivotXType, float pivotXValue,

int pivotYType, float pivotYValue) {}

参数fromDegrees、toDegrees代表旋转的开始角度和结束角度，pivotXValue、pivotYValue代

表旋转的中心位置，可以是绝对的像素值，也可以是比例值，比例值范围是[0f, 1.0f]，pivotXType、pivotYType代表旋转类型，有三种Animation.ABSOLUTE,、Animation.RELATIVE\_TO\_SELF、Animation.RELATIVE\_TO\_PARENT

具体用法如下：

RotateAnimation rotateAnimation = new RotateAnimation(0f, 360f, Animation.RELATIVE\_TO\_SELF, 0.5f, Animation.RELATIVE\_TO\_SELF, 0.5f);

rotateAnimation.setDuration(2000);

rotateAnimation.setFillAfter(true);

view.startAnimation(rotateAnimation);

实现了view相对自身中心，瞬时间旋转360度。

**1.4** 透明度动画

透明度动画通过AlphaAnimation类实现，构造函数如下：

public AlphaAnimation(float fromAlpha, float toAlpha) { }

参数fromAlpha、toAlpha代表透明度的起始值和结束值，0f代表完全透明，1.0f则无透明度。

具体用法如下：

AlphaAnimation alphaAnimation = new AlphaAnimation(1, 0);

alphaAnimation.setDuration(2000);

alphaAnimation.setFillAfter(true);

view.startAnimation(alphaAnimation);

实现了view从无透明度到完全透明的变化。

**1.5 View**的组合动画

View的组合动画通过AnimationSet类实现的，具体用法如下：

AnimationSet animationSet = new AnimationSet(true);

RotateAnimation rotateAnimation = new RotateAnimation(0f, 360f, Animation.RELATIVE\_TO\_SELF, 0.5f, Animation.RELATIVE\_TO\_SELF, 0.5f);

AlphaAnimation alphaAnimation = new AlphaAnimation(1, 0);

animationSet.addAnimation(rotateAnimation);

animationSet.addAnimation(alphaAnimation);

animationSet.setFillAfter(true);

animationSet.setDuration(2000);

view.startAnimation(animationSet);

AnimationSet的参数为true表示组合动画公用一个插值器，什么是插值器呢？就是动画速度的变化规律，常用的插值器如下：

LinearInterpolator：匀速

AccelerateInterpolator：加速

AccelerateDecelerateInterpolator：先加速再减速

DecelerateInterpolator：减速

BounceInterpolator：阻尼下落，即反弹数次后停止

可通过如下方式使用

RotateAnimation rotateAnimation = new RotateAnimation(0f, 360f, Animation.RELATIVE\_TO\_SELF, 0.5f, Animation.RELATIVE\_TO\_SELF, 0.5f);

rotateAnimation.setDuration(2000);

rotateAnimation.setFillAfter(true);

rotateAnimation.setInterpolator(new AccelerateInterpolator());

view.startAnimation(rotateAnimation);

最后，还可以通过setRepeatCount()、setRepeatMode()来设置动画重复的次数、和重复模式，重复模式包括Animation.RESTART、Animation.REVERSE，即重新开始和逆序播放。

如果要监听动画的执行情况，则可以通过如下接口：

public static interface AnimationListener {

void onAnimationStart(Animation animation); // 开始

void onAnimationEnd(Animation animation); // 结束

void onAnimationRepeat(Animation animation); // 重复

}

**2.** 通过**xml**方式实现

xml文件需要放到res目录下的anim文件夹。

**2.1** 平移动画

通过<translate >标签实现。

实现View向上平移自身高度距离，即title隐藏效果：

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

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

android:duration="2000"

android:fillAfter="true"

android:fromXDelta="0%"

android:fromYDelta="0%"

android:toXDelta="0%"

android:toYDelta="-100%" />

实现View向上平移50像素

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

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

android:duration="2000"

android:fillAfter="true"

android:fromXDelta="0"

android:fromYDelta="0"

android:toXDelta="0"

android:toYDelta="-50" />

**2.2** 缩放动画

通过<scale >标签实现。

实现View相对于自身中心，在x方向拉伸为原来2倍，在y方向缩小为原来0.5倍：

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

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

android:duration="2000"

android:fromXScale="1.0"

android:fromYScale="1.0"

android:pivotX="50%"

android:pivotY="50%"

android:toXScale="2.0"

android:toYScale="0.5" />

**2.3** 旋转动画

通过<scale >标签实现。

实现View相对自身中心，瞬时间旋转360度，同时逆序重复两次：

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

android:duration="2000"

android:fromDegrees="0"

android:pivotX="50%"

android:pivotY="50%"

android:repeatCount="2"

android:repeatMode="reverse"

android:toDegrees="360" />

**2.4** 透明度动画

通过<alpha>标签实现。

实现了View透明度从1.0f到0f的变化，同时是加速变化的：

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

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

android:duration="2000"

android:fromAlpha="1.0"

android:interpolator="@android:anim/accelerate\_interpolator"

android:toAlpha="0" />

**2.5 View**的组合动画

通过<set>标签实现，但不能控制次序，只能同时发生，测试中发现，此时repeatCount属相无效。

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

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

android:duration="5000"

android:fillAfter="true"

android:interpolator="@android:anim/linear\_interpolator"

android:shareInterpolator="true">

<alpha

android:fromAlpha="1.0"

android:toAlpha="0" />

<rotate

android:fromDegrees="0"

android:pivotX="50%"

android:pivotY="50%"

android:toDegrees="360" />

<scale

android:fromXScale="1.0"

android:fromYScale="1.0"

android:pivotX="50%"

android:pivotY="50%"

android:toXScale="1.5"

android:toYScale="0.5" />

<translate

android:fromXDelta="0%"

android:fromYDelta="0%"

android:toXDelta="0%"

android:toYDelta="-100%" />

</set>

通过xml方式实现时，需要先通过loadAnimation()加载xml文件，如下：

Animation animation = AnimationUtils.loadAnimation(context, R.anim.set\_anim);

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);

补间动画在Android最初的版本就有了，在功能和可扩展方面都有相当大的局限性，例如平移效果只能改变View的显示效果而已，并不能改变View真正的位置，举个例子，如果将一个有点击事件的Button从屏幕左上角移动到屏幕右上角，点击右上角按钮，发现并不能响应点击事件，此时再点击屏幕左上角竟然有响应。还有补间动画只能作用于View，如果我们要对一个非View的对象进行动画操作，那就无能为力了，正式因为种种功能上的缺陷，Android在3.0版本中引入了属性动画来进一步完善Android的动画机制。

二、属性动画（**Property Animation**）

有了属性动画，我们除了最基本的对View进行平移、缩放、旋转、透明度操作外，还可以将动画作用于指定的对象上，例如将一个Point对象从(0, 0)位置移动到(100, 100)位置。但是呢，有一点要注意，属性动画只能在**Android3.0**即以上版本使用，如果要兼容Android3.0以下版本，可以考虑使用大神JakeWharton的动画库：http://nineoldandroids.com/，但是这个库在Android3.0以下的属性动画其实还是传统的补间动画哦！

属性动画中核心的两个类就是ValueAnimator和ObjectAnimator。

**1. ValueAnimator**

是用来计算动画对应属性初始值和结束值之间的过渡的，类似于一个数值发生器，例如我们要实现数值0到数值2再到数值0在2000毫秒的过渡，可以这样做：

ValueAnimator animator = ValueAnimator.ofFloat(0f, 2f, 0f);

//设置监听器

animator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {

@Override

public void onAnimationUpdate(ValueAnimator animation) {

float curValue = (float) animation.getAnimatedValue(); //当前值

float fraction = animation.getAnimatedFraction(); //当前已过渡完成的比例

}

});

animator.setDuration(2000); // 动画时长

animator.start();

ValueAnimator.ofFloat()是实现浮点数的平滑过渡，如果需要整数的平滑过渡则可以使用ValueAnimator.ofInt()，用法基本一致。除了这两个外还有 ValueAnimator.ofArgb()、 ValueAnimator.ofObject()，其中ValueAnimator.ofArgb()可以用来进行颜色值的过渡，ValueAnimator.ofObject()可以用来实现对象的过渡效果，这是就需要我们自行定义扩展了。

ValueAnimator.ofFloat()是如何实现过渡效果的呢？其实就是通过一个FloatEvaluator 来完成的，不断的计算当前的值：

public class FloatEvaluator implements TypeEvaluator {

public Object evaluate(float fraction, Object startValue, Object endValue) {

float startFloat = ((Number) startValue).floatValue();

return startFloat + fraction \* (((Number) endValue).floatValue() - startFloat);

}

}

如果我们要从点Point(0, 0)过渡到点Point(100, 100)，同样也需要实现一个TypeEvaluator来计算当前的过渡属性值：

public class PointEvaluator implements TypeEvaluator {

@Override

public Object evaluate(float fraction, Object startValue, Object endValue) {

Point startPoint = (Point) startValue;

Point endPoint = (Point) endValue;

float x = startPoint.x + fraction \* (endPoint.x - startPoint.x);

float y = startPoint.y + fraction \* (endPoint.y - startPoint.y);

return new Point((int) x, (int) y);

}

}

很简单，fraction代表已经过渡完成的比例，根据当前完成的比例、开始点和结束点计算出当前点的值。有了PointEvaluator就可以实现我们自己的Point过渡效果了：

Point startPoint = new Point(0, 0);

Point endPoint = new Point(100, 100);

ValueAnimator animator = ValueAnimator.ofObject(new PointEvaluator(), startPoint, endPoint);

animator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {

@Override

public void onAnimationUpdate(ValueAnimator animation) {

Point currPoint = (Point) animation.getAnimatedValue();

}

});

这样就通过TypeEvaluator实现了一个我们自定义的估值器。

**2. ValueAnimator**

只是对值进行了一个平滑的动画过渡，ObjectAnimator才是实现对任意对象的属性进行动画操作的，ObjectAnimator是ValueAnimator的子类。先看一下如何通过ObjectAnimator实现传统补间动画的四种效果：

**2.1** 平移动画

要将一个View右移出屏幕，再移动回来，可以这样做：

float translationX = view.getTranslationX();

ObjectAnimator animator = ObjectAnimator.ofFloat(view, "translationX", translationX, 600f, translationX);

animator.setDuration(3000);

animator.start();

通过ofFloat()方法我们创建了一个ObjectAnimator对象，ofFloat()方法的首个参数view就是要进行平移操作的对象，因为我们要对view进行平移操作，所以第二个参数传入translationX，代表view的平移属性，之后的参数是一个是可变长度的，个数根据你的需求控制。所以核心的参数就是第二个，根据这个属性类型来区分对View进行何种动画操作。

**2.2** 缩放动画

同样的道理，实现View的缩放效果可以这样做：

ObjectAnimator animator = ObjectAnimator.ofFloat(view, "scaleX", 1f, 2f, 1f);

animator.setDuration(5000);

animator.start();

我们将ofFloat()方法第二个参数换成了scaleX，表示在x方向对view进行缩放操作，这样我们就实现了view拉伸两倍再还原的效果。

**2.3** 旋转动画

例如，要将一个View旋转360度可以这样做，只要将ofFloat()第二个参数写成rotation，起始、结束角度分别为0和360：

ObjectAnimator animator = ObjectAnimator.ofFloat(view, "rotation", 0f, 360f);

animator.setDuration(3000);

animator.start();

**2.4** 透明度动画

只要将ofFloat()第二个参数写成alpha，则实现了View透明度从1到0在到0的变化：

ObjectAnimator animator = ObjectAnimator.ofFloat(view, "alpha", 1f, 0f, 1f);

animator.setDuration(5000);

animator.start();

**2.5** 颜色动画

使用属性动画改变一个View的背景颜色也是可以的，如下代码可以实现View背景色从蓝到红的变化：

ObjectAnimator animator = ObjectAnimator.ofFloat(view, "backgroundColor", Color.BLUE, Color.RED);

animator.setEvaluator(new ArgbEvaluator());

animator.setDuration(5000);

animator.start();

**2.6** 组合动画

和补间动画类似，属性动画同样可以将单个动画进行组合，而且功能更强大，需要通过AnimatorSet类来实现，通过调用其play()方法得到一个AnimatorSet.Builder对象，Builder对象有以下四个方法：

（1）after(Animator anim) 将现有动画插入到传入的动画之后执行；

（2）after(long delay) 将现有动画延迟指定毫秒后执行；

（3）before(Animator anim) 将现有动画插入到传入的动画之前执行；

（4）with(Animator anim) 将现有动画和传入的动画同时执行 。

我们要实现一个View从屏幕右侧移入屏幕，然后旋转360度同时有透明度变化，最后在水平方向拉伸两倍后还原的效果可以这样么做：

AnimatorSet animatorSet = new AnimatorSet();

ObjectAnimator alpha = ObjectAnimator.ofFloat(view, "alpha", 1f, 0f, 1f);

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

ObjectAnimator translation = ObjectAnimator.ofFloat(view, "translationX", 600f, 0f);

ObjectAnimator scale = ObjectAnimator.ofFloat(view, "scaleX", 1f, 2f, 1f);

animatorSet.play(alpha).with(rotation).after(translation).before(scale);

animatorSet.setDuration(5000);

animatorSet.start();

当然还可以setRepeatCount()、setRepeatMode()设置动画的重复次数以及重复模式，重复模式有ValueAnimator.RESTART、ValueAnimator.REVERSE两种。

**3.** 除了通过代码来编写属性动画外，还可以使用**xml**的方式，**xml**文件需要放到**res**目录下的**animator**文件夹。

可用的标签有以下三种：

• <animator> 代表ValueAnimator

• <objectAnimator> 代表ObjectAnimator

• <set> 代表AnimatorSet

好了，看几个例子：

要实现0到50平滑过渡的效果，可以这么做：

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

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

android:duration="2000"

android:valueFrom="0"

android:valueTo="50"

android:valueType="floatType" />

要将一个View旋转360度，可以这么做：

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

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

android:duration="2000"

android:propertyName="rotation"

android:valueFrom="0"

android:valueTo="360"

android:valueType="floatType" />

要将一个View透明度从1变为0，可以这样编写：

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

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

android:duration="2000"

android:propertyName="alpha"

android:valueFrom="1"

android:valueTo="0"

android:valueType="floatType" />

平移和缩放动画都是类似的。再看一下<set>标签的用法：

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

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

android:ordering="sequentially">

<objectAnimator

android:duration="1500"

android:propertyName="translationX"

android:valueFrom="-500"

android:valueTo="0"

android:valueType="floatType" />

<set android:ordering="together">

<objectAnimator

android:duration="2000"

android:propertyName="rotation"

android:valueFrom="0"

android:valueTo="360"

android:valueType="floatType" />

<set android:ordering="sequentially">

<objectAnimator

android:duration="1000"

android:propertyName="alpha"

android:valueFrom="1"

android:valueTo="0"

android:valueType="floatType" />

<objectAnimator

android:duration="1000"

android:propertyName="alpha"

android:valueFrom="0"

android:valueTo="1"

android:valueType="floatType" />

</set>

</set>

<set android:ordering="together">

<objectAnimator

android:duration="1500"

android:propertyName="scaleX"

android:valueFrom="1"

android:valueTo="2"

android:valueType="floatType" />

<objectAnimator

android:duration="1500"

android:propertyName="scaleX"

android:valueFrom="2"

android:valueTo="1"

android:valueType="floatType" />

</set>

</set>

我们实现了View先平移，然后同时进行旋转和透明度变化，最后进行缩放的动画效果。其中ordering属相我们使用了sequentially、together两种，分别代表顺序播放和同时播放。还

有以下两个我们没用到的属性：startOffset：表示动画的延时启动时间，以及repeatCount、repeatMode。

使用xml动画文件也是非常简单的：

Animator animator = AnimatorInflater.loadAnimator(context, R.animator.anim\_file);

animator.setTarget(view);

animator.start();

4. Animator类当中提供了一个addListener()方法，可以用来监听动画的执行情况，无论ObjectAnimator、ValueAnimator还是AnimatorSet都是Animator的子类，所以它们都可以使用addListener()：

anim.addListener(new Animator.AnimatorListener() {

@Override

public void onAnimationStart(Animator animation) { }

@Override

public void onAnimationEnd(Animator animation) { }

@Override

public void onAnimationCancel(Animator animation) { }

@Override

public void onAnimationRepeat(Animator animation) { }

});

如果想监听其中的某些事件则可以通过AnimatorListenerAdapter来实现：

animator.addListener(new AnimatorListenerAdapter() {

@Override

public void onAnimationStart(Animator animation) {

super.onAnimationStart(animation);

}

@Override

public void onAnimationEnd(Animator animation) {

super.onAnimationEnd(animation);

}

});

这样我们只监听了动画的开始和结束事件。

5. 属性动画的插值器是兼容补间动画的插值器的，所以补间动画中的插值器完全可以在属性动画中使用。另外属性动画提供了一个TimeInterpolator 接口，它的作用是根据时间流逝的百分比计算出当前属性值改变的百分比，通过这个接口我们来自定义属性动画插值器：

public interface TimeInterpolator {

/\*\*

\* Maps a value representing the elapsed fraction of an animation to a value that represents

\* the interpolated fraction. This interpolated value is then multiplied by the change in

\* value of an animation to derive the animated value at the current elapsed animation time.

\*

\* @param input A value between 0 and 1.0 indicating our current point

\* in the animation where 0 represents the start and 1.0 represents

\* the end

\* @return The interpolation value. This value can be more than 1.0 for

\* interpolators which overshoot their targets, or less than 0 for

\* interpolators that undershoot their targets.

\*/

float getInterpolation(float input);

}

接口中只有一个getInterpolation()方法，其中input参数会根据动画设置的时长在0到1之间匀速增长的变化。如果要自定义插值器可以这样写：

public class MyInterpolator implements TimeInterpolator {

@Override

public float getInterpolation(float input) {

float result = 0;

// TODO

return result;

}

}

具体的实现细节可参考系统插值器。有一点需要注意，我们计算出来的result的值必须在0到1之间。

6. 回顾一下，我们在java代码中可以通过ObjectAnimator.ofFloat()或ObjectAnimator.ofInt()来实现属性动画其中第二个参数可以是alpha、rotation、scaleX、translateX等等。为什么第二个参数可以是这些呢？这是因为ObjectAnimator内部的工作机制并不是对传入的属性名进行操作的，而是根据属性名在当前子View类以及父类中去找对应的get和set方法，然后通过方法不断地对值进行改变，从而实现动画效果的，例如我们可以在View类中找到了参数rotation对应的get和set方法：

public float getRotation() {

return mRenderNode.getRotation();

}

public void setRotation(float rotation) {

if (rotation != getRotation()) {

// Double-invalidation is necessary to capture view's old and new areas

invalidateViewProperty(true, false);

mRenderNode.setRotation(rotation);

invalidateViewProperty(false, true);

invalidateParentIfNeededAndWasQuickRejected();

notifySubtreeAccessibilityStateChangedIfNeeded();

}

}

既然如此，除了系统提供的属性动画外，如果要给一个自定义Button添加一个widths属相动画，实现其宽度的变化，如果使用translateX属性会导致Button内容的拉伸，这并不是我们愿意看到的，所以我们自定义的widths属相动画并没有这种问题。首先看我们自定义的Button类：

public class MyButton extends Button{

public MyButton(Context context) { super(context); }

public MyButton(Context context, AttributeSet attrs) { super(context, attrs); }

public int getWidths() { return getLayoutParams().width; }

public void setWidths(int width){

getLayoutParams().width = width;

requestLayout();

}

}

因为我们规定属性名为widths，所以我们提供了getWidths()、setWidths()两个方法，当然这两个方法也是必须的。接下来还需要编写一个TypeEvaluator类来告诉系统宽度如何过渡：

public class WidthsEvaluator implements TypeEvaluator {

@Override

public Object evaluate(float fraction, Object startValue, Object endValue) {

int startWidth = (int) startValue;

int endWidth = (int) endValue;

return (int)(startWidth + fraction \* (endWidth - startWidth));

}

}

有了WidthsEvaluator 类，我们需要的东西也就够了，看下布局文件、以及使用方法：

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

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<com.example.othershe.mybutton.MyButton

android:id="@+id/my\_btn"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_centerInParent="true"

android:text="哎呦，不错哦！" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:onClick="start"

android:text="开始" />

</RelativeLayout>

public class MainActivity extends AppCompatActivity {

private MyButton button;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

button = (MyButton) findViewById(R.id.my\_btn);

}

public void start(View view) {

int width = button.getWidth();

ObjectAnimator animator = ObjectAnimator.ofObject(button, "widths", new WidthsEvaluator(), width, 600);

animator.setDuration(3000);

animator.start();

}

}

通过ObjectAnimator.ofObject()来调用的，很简单。

7. 通过java代码实现属性动画除了通过ObjectAnimator类，还有另外一种方式，就是使用ViewPropertyAnimator类。例如我们要实现一个球形View自由落体的效果，可以这样写：

view.animate().x(0).y(500)

.setDuration(5000)

.setInterpolator(new BounceInterpolator());

通过view.animate()方法得到一个ViewPropertyAnimator对象，之后的操作都是基于该对象的方法，而且是链式调用的，同时在链尾系统会默认的添加start()方法，所以动画会自动执行。仅仅是写法的不同，根据喜好选择吧，其它的方法有兴趣的话可以自行测试。

三、帧动画（**Drawable Animation**）

帧动画是顺序的播放一系列图片，从而产生动画的效果，其实也就是图片的切换。但是如果图片过多、过大的话是很容易产生OOM的，所以使用时需要注意。

首先在res目录下的drawable文件夹编写一个xml文件：

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

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

android:oneshot="true">//

<item android:drawable="@mipmap/icon1" android:duration="300"/>

<item android:drawable="@mipmap/icon2" android:duration="300"/>

<item android:drawable="@mipmap/icon3" android:duration="300"/>

<item android:drawable="@mipmap/icon4" android:duration="300"/>

<item android:drawable="@mipmap/icon5" android:duration="300"/>

<item android:drawable="@mipmap/icon6" android:duration="300"/>

<item android:drawable="@mipmap/icon7" android:duration="300"/>

<item android:drawable="@mipmap/icon8" android:duration="300"/>

<item android:drawable="@mipmap/icon9" android:duration="300"/>

</animation-list>

oneshot属性表示是否循环播放，值为true则只播放一次。

通如下方法调用，其中view是一个ImageView对象：

view.setImageResource(R.drawable.icons);

AnimationDrawable animationDrawable = (AnimationDrawable) view.getDrawable();

animationDrawable.start();

如果要停止播放可通过如下方法：

AnimationDrawable animationDrawable = (AnimationDrawable) view.getDrawable();

animationDrawable.stop();

到这里Android动画相关的内容就结束了，足以应对开发中的使用场景了。