Android动画

1. AlphaAnimation

```
RelativeLayout rl_splash = (RelativeLayout) findViewById(R.id.rl_splash);
//播放动画效果
AlphaAnimation animation = new AlphaAnimation(1.0f, 0.0f);
//设置Alpha动画的持续时间
animation.setDuration(2000);
//播放Alpha动画
rl_splash.setAnimation(animation);
```

2. RotateAnimation

```
//相对于自身的哪个位置旋转，这里是相对于自身的右下角
RotateAnimation ra = new RotateAnimation(0, 360,  //从哪旋转，旋转多少度
        Animation.RELATIVE_TO_SELF, 1.0f, Animation.RELATIVE_TO_SELF,
        1.0f);
ra.setDuration(800);
ra.setRepeatCount(Animation.INFINITE);
ra.setRepeatMode(Animation.RESTART);
iv_scan.startAnimation(ra);
```

3. ScaleAnimation(缩放动画)

```
ScaleAnimation(float fromX, float toX, float fromY, float toY)
Constructor to use when building a ScaleAnimation from code
```

4. TranslateAnimation(位移动画)

```
TranslateAnimation(int fromXType, float fromXValue, int toXType, float
toXValue, int fromYType, float fromYValue, int toYType, float toYValue)
Constructor to use when building a TranslateAnimation from code
```

5. AnimationSet (多组动画)

```
ScaleAnimation sa = new ScaleAnimation(0.2f, 1.0f, 0.4f,1.0f);//缩放的动画效
果,1.0f就代表窗体的总宽或者高
sa.setDuration(400);
TranslateAnimation ta = new TranslateAnimation(//位移动的动画效果
          Animation.RELATIVE_TO_SELF, 0,//指定这个位置是相对于谁
          Animation.RELATIVE_TO_SELF, 0.1f,
          Animation.RELATIVE_TO_SELF, 0,
          Animation.RELATIVE_TO_SELF, 0);
ta.setDuration(300);
AnimationSet set = new AnimationSet(false);//如果想播放多种动画的组合，这里就要用到
了AnimationSet
set.addAnimation(sa);
set.addAnimation(ta);
contentView.startAnimation(set); // 播放一组动画.
```

6. Frame动画
   在SDK中提到，不要在onCreate中调用start方法开始播放Frame动画，因为AnimationDrawable还没有完全跟Window相关联，如果想要界面显示时就开始播放帧动画的话，可以在onWindowFocusChanged()中调用start()。
   - 在drawable目录下新建一个xml文件，内容如下:

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list
xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true" > //onshot是指定是否循环播放
    <item
        android:drawable="@drawable/desktop_rocket_launch_1"  //Frame动画的图片
        android:duration="50"/> //播放这个图片持续的时间
    <item
        android:drawable="@drawable/desktop_rocket_launch_2"
        android:duration="100"/>
</animation-list>
```

   - 播放Frame动画

```
AnimationDrawable rocketAnimation;
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      ImageView rocketImage = (ImageView) findViewById(R.id.iv);
      rocketImage.setBackgroundResource(R.drawable.animlist); //将上边建的Frame动画的xml文件通过背景资源设置给图片
      rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
 //获取到图片的背景资源
}
public void start(View view) {
      if (!rocketAnimation.isRunning()) {
           rocketAnimation.start();  //播放
      }
}
```

7. 保持动画播放完成后的状态animation.setFillAfter(true);

```
Interpolator //定义了动画的变化速度，可以实现匀速、正加速、负加速、无规则变加速度
AccelerateDecelerateInterpolator//先加速后减速。
AccelerateInterpolator//逐渐加速。
LinearInterpolator//平稳不变的
DecelerateInterpolator//逐渐减速
CycleInterpolator//曲线运动特效，要传递float型的参数。
animation.setInterpolator(new LinearInterpolator());//指定动画的运行效果
```

上面所讲的动画都是Android 3.0之前的动画，也就是我们最先熟悉的Frame动画和Tween动画。

从3.0开始又引入了一个新的动画叫做Property动画。并且针对这三种动画的动画模式分为:

- Property Animation：属性动画,从Android 3.0开始引进，更改的是对象的实际属性，而且属性动画不止可以应用于View还可以应用于任何对象。

- `View Animation`：指之前的`Tween`动画，`alpha scale translate rotate`等。为什么叫做`View Animation`呢？因为它只能应用于`View`对象，而且只支持一部分属性，如支持缩放旋转而不支持背景颜色的变化。对于`View Animation`而言，它只改变了`View`对象绘制的位置，而没有改变`View`对象本身的属性，比如，有一个200*200大小的`Button`，你用`Tween`动画给放大到500*500但是它的有效点击区域还是200*200。
- `Drawable Animation`：指之前的`Frame`动画。因为它是通过一帧帧的图片来播放的。

下面我们开始仔细讲解一下`Property Animation`

官方文档中是这样这样介绍属性动画的:

```
The property animation system is a robust framework that allows you to animate
almost anything. You can define an animation to change any object property over
time, regardless of whether it draws to the screen or not. A property animation
changes a property's (a field in an object) value over a specified length of
time. To animate something, you specify the object property that you want to
animate, such as an object's position on the screen, how long you want to animate
it for, and what values you want to animate between.
```

一个强大的框架。

在`Property Animation`中，可以对动画应用以下属性:

- `Duration`: 指定动画持续时间，默认时间是`300ms`
- `TimeInterpolation`: 一些效果，如加速、加速等。
- `Repeat count and behavior`: 重复次数已经
- `Animation Set`: 动画合集。用来同时或者顺序播放多个动画。
- `Frame Refresh Delay`: 多长时间刷新一次，默认是`10ms`。

`ValueAnimator`

`ValueAnimator`包含`Property Animation`动画的所有核心功能，如动画时间，开始、结束属性值，相应时间属性值计算方法等。应用`Property Animation`有两个步聚:

- 计算属性值
- 根据属性值执行相应的动作，如改变对象的某一属性。

`ValuAnimiator`只完成了第一步工作，如果要完成第二步，需要实现`ValueAnimator.onUpdateListener`接口，这个接口只有一个函数`onAnimationUpdate()`，在这个函数中会传入`ValueAnimator`对象做为参数，通过这个`ValueAnimator`对象的`getAnimatedValue()`函数可以得到当前的属性值。

`ObjectAnimator`

继承自`ValueAnimator`，要指定一个对象及该对象的一个属性，当属性值计算完成时自动设置为该对象的相应属性，即完成了`Property Animation`的全部两步操作。实际应用中一般都会用`ObjectAnimator`来改变某一对象的某一属性，但用`ObjectAnimator`有一定的限制，要想使用`ObjectAnimator`，应该满足以下条件:

- 对象应该有一个`setter`函数: `set<PropertyName>`（驼峰命名法）
- 如上面的例子中，像`ofFloat`之类的工场方法，第一个参数为对象名，第二个为属性名，后面的参数为可变参数，如果`values...`参数只设置了一个值的话，那么会假定为目的值，属性值的变化范围为当前值到目的值，为了获得当前值，该对象要有相应属性的`getter`方法: `get<PropertyName>`
- 如果有`getter`方法，其应返回值类型应与相应的`setter`方法的参数类型一致。

- `object`的`setXxx`对属性`xxx`所做的改变必须能够通过某种方法反映出来，比如会带来`ui`的改变啥的（如果这条不满足，动画无效果）,例如我对`TextView`或者`Button`使用`width`的`ObjectAnimator`动画，就会发现无效，虽然他们都有`setWidth`和`getWidth`方法，但是`setWidth`方法的内部实现是改变`TextView`的最大宽度和最小宽度的，和`TextView`的宽度不是一个东西。所以动画就会无效。确切的说`TextView`的宽度对应的是`xml`中`android:layout_width`属性，而`TextView`还有另外一个属性:`android:width`，而`android:width`属性对应的就是`TextView`中的`setWidth`方法。

如果上述条件不满足，则不能用`ObjectAnimator`，应用`ValueAnimator`代替。
也就是说`ObjectAnimator`内部的工作机制是通过寻找特定属性的`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。

`AnimationSet`

`AnimationSet`提供了一个把多个动画组合成一个组合的机制，并可设置组中动画的时序关系，如同时播放，顺序播放等。

以下例子同时应用5个动画:

- Plays bounceAnim.
- Plays squashAnim1, squashAnim2, stretchAnim1, and stretchAnim2 at the same time.
- Plays bounceBackAnim.
- Plays fadeAnim.

```
AnimatorSet bouncer = new AnimatorSet();
bouncer.play(bounceAnim).before(squashAnim1);
bouncer.play(squashAnim1).with(squashAnim2);
bouncer.play(squashAnim1).with(stretchAnim1);
bouncer.play(squashAnim1).with(stretchAnim2);
bouncer.play(bounceBackAnim).after(stretchAnim2);
ValueAnimator fadeAnim = ObjectAnimator.ofFloat(newBall, "alpha", 1f, 0f);
fadeAnim.setDuration(250);
AnimatorSet animatorSet = new AnimatorSet();
animatorSet.play(bouncer).before(fadeAnim);
animatorSet.start();
```

TypeEvalutors

根据属性的开始、结束值与`TimeInterpolation`计算出的比例值来计算当前时间对应的属性值，`Android`提供了一下几种`evalutor`:

- `IntEvalutor`:`int`类型的属性值
- `FloatEvaluator`:
- `ArgbEvaluator`: 属性的值类型为十六进制颜色值;
- `TypeEvaluator`: 一个接口，可以通过实现该接口自定义Evaluator。
  自定义`TypeEvaluator`也很简单，只需要实现一个方法，如`FloatEvalutor`的定义:

```
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);
    }
}
```

`evaluate()` 方法当中传入了三个参数，第一个参数 `fraction` 非常重要，这个参数用于表示动画的完成度的，我们应该根据它来计算当前动画的值应该是多少，第二第三个参数分别表示动画的初始值和结束值。那么上述代码的逻辑就比较清晰了，用结束值减去初始值，算出它们之间的差值，然后乘以 `fraction` 这个系数，再加上初始值，那么就得到当前动画的值了。

`TimeInterplator`

Time interplator定义了属性值变化的方式，如线性均匀改变，开始慢然后逐渐快等。在Property Animation中是TimeInterplator，在View Animation中是Interplator，这两个是一样的，在3.0之前只有Interplator，3.0之后实现代码转移至了TimeInterplator。Interplator继承自TimeInterplator，内部没有任何其他代码。

- AccelerateInterpolator 加速，开始时慢中间加速
- DecelerateInterpolator 减速，开始时快然后减速
- AccelerateDecelerateInterolator 先加速后减速，开始结束时慢，中间加速
- AnticipateInterpolator 反向 ，先向相反方向改变一段再加速播放
- AnticipateOvershootInterpolator 反向加回弹，先向相反方向改变，再加速播放，会超出目的值然后缓慢移动至目的值
- BounceInterpolator 跳跃，快到目的值时值会跳跃，如目的值100，后面的值可能依次为85，77，70，80，90，100
- CycleInterpolator 循环，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，线性均匀改变
- OvershottInterpolator 回弹，最后超出目的值然后缓慢改变到目的值
- TimeInterpolator 一个接口，允许你自定义interpolator，以上几个都是实现了这个接口

`PropertyValuesHolder`

如果要实现一个对象不同属性的动画效果，除了 `Set` ，我们还可以利用 `PropertyValuesHolder` 和 `ViewPropertyAnimator` 对象来实现，具体做法如下：

```
PropertyValuesHolder pvhX = PropertyValuesHolder.ofFloat("x", 50f);
PropertyValuesHolder pvhY = PropertyValuesHolder.ofFloat("y", 100f);
ObjectAnimator.ofPropertyValuesHolder(myView, pvhX, pvyY).start();
```

`ViewPropertyAnimator`

- This class is not constructed by the caller, but rather by the View whose properties
- it will animate. Calls to {@link android.view.View#animate()} will return a reference
- to the appropriate ViewPropertyAnimator object for that View.

  `

如果需要对一个View的多个属性进行动画可以用ViewPropertyAnimator类，该类对多属性动画进行了优化，会合并一些invalidate()来减少刷新视图，该类在3.1中引入。

`view.animate()` 方法会返回 `ViewPropertyAnimator` 类。

```
myView.animate().x(50f).y(100f);
```

的效果与上面的`PropertyValuesHolder`例子中的效果完全一致。
但是你有没有发现我们自始至终没有调用过`start()`方法，这是因为新的接口中使用了隐式启动动画的功能，只要我们将动画定义完成之后，动画就会自动启动。并且这个机制对于组合动画也同样有效，只要我们不断地添加新的方法，那么动画就不会立刻执行，等到所有在`ViewPropertyAnimator`上设置的方法都执行完毕后，动画就会自动启动。当然如果不想使用这一默认机制的话，我们也可以显式地调用`start()`方法来启动动画。

XML中定义

在`res/animator`中定义对应的动画`xml`

- 对应代码中的ValueAnimator
- 对应代码中的ObjectAnimator
- 对应代码中的AnimatorSet
  例如:

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:propertyName="scaleX"
    android:duration="2000"
    android:valueFrom="1.0"
    android:valueTo="2.0"
    android:valueType="floatType"
    android:repeatCount="1"
    android:repeatMode="reverse">
</objectAnimator>
```

这里说明一下`valueFrom`和`valueTo`:是动画开始和结束值，如果我们缩放，则它们对应的是倍数，如果我们平移则对应的就是距离了。

接下来就是调用了

```
scaleXAnimator = (ObjectAnimator)AnimatorInflater.loadAnimator(this,
R.animator.scalex);
scaleXAnimator.setTarget(btnScaleX);
scaleXAnimator.start();
```

那如果我们要播放多个动画怎么办?

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:ordering="together">
    <objectAnimator
        android:duration="2000"
        android:propertyName="scaleX"
        android:repeatCount="1"
        android:repeatMode="reverse"
        android:valueFrom="1.0"
        android:valueTo="2.0" >
    </objectAnimator>
    <objectAnimator
        android:duration="2000"
        android:propertyName="scaleY"
        android:repeatCount="1"
```

```
        android:repeatMode="reverse"
        android:valueFrom="1.0"
        android:valueTo="2.0" >
    </objectAnimator>
</set>
```

```
animatorScaleSet = (AnimatorSet)AnimatorInflater.loadAnimator(this,
R.animator.scale);
animatorScaleSet.setTarget(btnScale);
animatorScaleSet.start();
```

`Android L` 又增加了一种动画样式，叫做 `Reveal Animation`。

可以直接通过 `ViewAnimationUtils.createCircularReveal()` 方法来创建。

```
/**
 * Returns an Animator which can animate a clipping circle.
 * <p>
 * Any shadow cast by the View will respect the circular clip from this
animator.
 * <p>
 * Only a single non-rectangular clip can be applied on a View at any time.
 * Views clipped by a circular reveal animation take priority over
 * {@link View#setClipToOutline(boolean) View Outline clipping}.
 * <p>
 * Note that the animation returned here is a one-shot animation. It cannot
 * be re-used, and once started it cannot be paused or resumed. It is also
 * an asynchronous animation that automatically runs off of the UI thread.
 * As a result {@link AnimatorListener#onAnimationEnd(Animator)}
 * will occur after the animation has ended, but it may be delayed depending
 * on thread responsiveness.
 *
 * @param view The View will be clipped to the animating circle.
 * @param centerX The x coordinate of the center of the animating circle,
relative to
 *                <code>view</code>.
 * @param centerY The y coordinate of the center of the animating circle,
relative to
 *                <code>view</code>.
 * @param startRadius The starting radius of the animating circle.
 * @param endRadius The ending radius of the animating circle.
 */
public static Animator createCircularReveal(View view,
        int centerX,  int centerY, float startRadius, float endRadius) {
    return new RevealAnimator(view, centerX, centerY, startRadius, endRadius);
}
```

代码如下:

```
void enterReveal() {
    final View myView = findViewById(R.id.my_view);

    int cx = myView.getMeasuredWidth() / 2;
    int cy = myView.getMeasuredHeight() / 2;
```

```
    int finalRadius = Math.max(myView.getWidth(), myView.getHeight()) / 2;

    Animator anim =
        ViewAnimationUtils.createCircularReveal(myView, cx, cy, 0, finalRadius);

    anim.start();
}

void exitReveal() {
    final View myView = findViewById(R.id.my_view);

    int cx = myView.getMeasuredWidth() / 2;
    int cy = myView.getMeasuredHeight() / 2;

    int initialRadius = myView.getWidth() / 2;

    Animator anim =
        ViewAnimationUtils.createCircularReveal(myView, cx, cy, initialRadius,
0);

    anim.addListener(new AnimatorListenerAdapter() {
        @Override
        public void onAnimationEnd(Animator animation) {
            super.onAnimationEnd(animation);
            myView.setVisibility(View.INVISIBLE);
        }
    });

    anim.start();
}
```