PropertyAnimation属性动画

1、基本介绍

前面介绍了补间动画，我们说过，他不会改变View原本的属性状态，举个普通例子，当一个Button发送移动以后，并且设置个fillAfter将Button固定在变化后的位置，此时点击Button，是没有反应的，但是点击Button移动前的那个位置，发现点击有左右，这就是因为Button的实际位置还是在原本的位置，位置属性没有发生变化。

属性动画是在补间动画以后的，可以认为是它的改良版，因此属性动画是可以真正改变View的属性的，并且补间动画也就是前面介绍的四种变化以及混合使用，局限很大。但是属性动画显得很灵活，毕竟一个View的属性怎么可能只限制在这四种类型

同时，属性动画注重点不在是动画的本身，而是View的属性变化上，通过数值发生器（后面介绍），根据一定的变化，生成一系列数值，然后作用在View的属性上，才形成动画的效果。

2、核心类

（1）ValueAnimator

就是上面介绍的数值发生器

（2）ObjectAnimator

是ValueAnimator的子类，对ValueAnimator操作进行了封装，操作起来更加方便

（3）AnimationSet

前面介绍动画集合的时候已经介绍过了，将几个动画放在一起执行

3、ObjectAnimator介绍

因为是ValueAnimator的子类，比较简单，因此先来介绍这个

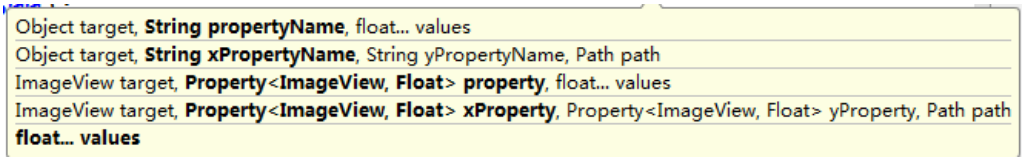
（1） 缩放（View.Scale_X，View.Scale_Y）

```
ObjectAnimator scale = ObjectAnimator.ofFloat(objectView, View.SCALE_X,1,2);
scale.setDuration(500);
scale.start();
```

注意ofFloat() 表示属性值是Float类型的，四个参数的含义。objectVIew

是要变化的View，View.SCALE_X 是指 x轴方向的缩放值，1是变化前是原本View的一倍，2是变化后时原本View的两倍。效果就是500毫秒内，View的宽从原来的一倍变为原来的2倍

他的构造函数重载很多



介绍多个Value值变化的含义

每一个Value值都表示变化的经过倍数（都是原本VIew的倍数）

如 1,2,3,2,1 就表示从1,2,3倍依次方法，然后又变成2,1倍，效果就是向一直扩大到3倍，接着又缩小到和与原本一样大小

（2）旋转(View.ROTATE, View.ROTATE_X, VIew.ROTATE_Y)

```
ObjectAnimator rotate = ObjectAnimator.ofFloat(objectView,View.ROTATION,0,360);
rotate.setDuration(1500);
rotate.start();
```

构造函数依旧是重载非常多

多个Value值的情况也是和Scale是一样的，按着每一个值依次变化

旋转点默认是View的中心点

（3）平移（View.translate_X,View.translate_Y,View.x View.y）

```
ObjectAnimator animator = ObjectAnimator.ofFloat(objectView,
        View.TRANSLATION_Y,0,100,200,100,0);
animator.setDuration(1500);
animator.start();
```

注意（translate_x 和 x 的区别）

translate_x: 表示View偏移父布局的距离(偏移量)

x:表示到View左上角父布局的距离(绝对值)

（4）背景颜色

```
ObjectAnimator animator = ObjectAnimator.ofInt(objectView,"backgroundColor",
        Color.BLUE,Color.RED);
animator.setDuration(1500);
animator.start();
```

只要是VIew的属性，基本都能进行改变

这是补间动画不能做到的

注意ofInt 和 ofFloat 的区别

这是根据我们需要改变的VIew的属性来选择的（宽度，alpha值等是float

的，但是颜色肯定是Int值的,类型不匹配会发生异常）

4、ValueAnimator介绍

前面介绍了，ValueAnimator就是数值发生器，用来产生一系列的值。起点值和结束值是确认的，也就是我们想要View从什么状态变成什么状态。然后ValueAnimator会在这两个值之间产生一系列的值来充当过度的作用，而过度的速率是由interpolator决定的，整个过程的时间是有duration决定的。

因此：

valueAnimator 是根据duration时间，interpolator类型，和起点终点值的不用，从而产生出一系列不用的过度数值。

实例：

```
ValueAnimator valueAnimator = ValueAnimator.ofInt(0,10000);
valueAnimator.setDuration(1000);
valueAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener()
{
    @Override
    public void onAnimationUpdate(ValueAnimator animation)
    {
        Log.d("value","当前值："+animation.getAnimatedValue());
        Log.d("value","当前动画进度："+animation.getAnimatedFraction());
        valueText.setText("当前value值："+animation.getAnimatedValue());
    }
});
valueAnimator.start();
```

获取ValueAnimator实例

添加监听

每16ms就调用一次该更新函数

可以看到，那个监听事件就是使用ValueAnimator最大的作用，在里面的方法调用中，我们可以非常灵活的进行我们的逻辑处理。

各种相关设置

```
setDuration(long duration)  ValueAnimator
setInterpolator(TimeInterpolator v…  void
setTarget(Object target)  void
setCurrentFraction(float fraction)  void
setCurrentPlayTime(long playTime)  void
setEvaluator(TypeEvaluator value)  void
setFloatValues(float... values)  void
setIntValues(int... values)  void
setObjectValues(Object... values)  void
setRepeatCount(int value)  void
setRepeatMode(int value)  void
```

5、AnimatorSet

简单使用

```
ObjectAnimator a1 = ObjectAnimator.ofFloat(imageView,View.SCALE_X,1,2);
ObjectAnimator a2 = ObjectAnimator.ofFloat(imageView,View.SCALE_Y,1,2);

AnimatorSet set = new AnimatorSet();
set.setDuration(1500);
set.playTogether(a1,a2);   ← 表示两个动画同时执行
set.start();
```

这个比较简单，就是将几个动画集合在一起播放

但是也有几个比较重要的方法需要理解清楚

```
set.playTogether(a1,a2);   ← 表示a1和a2动画同时执行
set.play(a1).with(a2);

set.playSequentially(a1,a2);   ← 表示按顺序执行，即a1先执行，执行完a2再执行

set.play(a1).after(a2);   ← 表示a1在a2之后执行，即a2执行完a1再执行
set.play(a1).before(a2);  ← 表示a1在a2之前执行，即a1执行完a2再执行

set.play(a1).after(1000);   ← 表示延迟多久执行，即开始1秒后，a1才开始执行
```

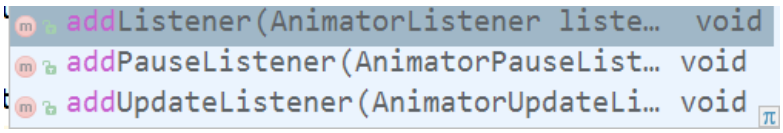
6、path动画

一种非常强大的动画类型，可以自定义一个path，然后让view按着path的路径运动

```
float x = imageView.getX();
float y = imageView.getY();
Path path = new Path();
path.lineTo(x+200,y+200);
ObjectAnimator animator = ObjectAnimator
        .ofFloat(imageView, View.X,View.Y,path);
animator.setDuration(1500);
animator.start();
```

api21以上才能使用，并且不知道为什么path.addCycle()。想让view围绕着圆来变化不行

7、动画监听器



有三种类型的监听器

第一种就是和前面介绍Animation 的是一样的，动画的整生命周期过程的监听

第二种监听动画pause 和 resume 的过程

第三种就是介绍ValueAnimator时的监听动画

对于第一种，提供了比较简洁的适配器类，可以利用AnimationListenerAdapter，可以自己决定需要重写的方法

```
animator.addListener(new AnimatorListenerAdapter()
{
    @Override
    public void onAnimationCancel(Animator animation)
    {
        super.onAnimationCancel(animation);
    }

    @Override
    public void onAnimationStart(Animator animation)
    {
        super.onAnimationStart(animation);
    }
});
```