Android程序设计

动画

2019.6.11

isszym　sysu.edu.cn

官方文档（中文） 官方文档（英文） runoob cnblogs

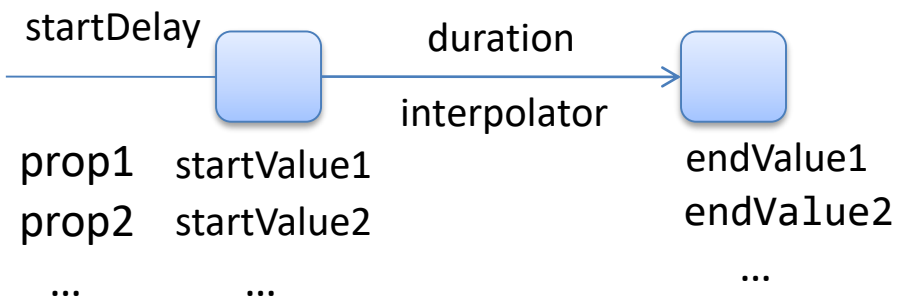
主目录

动画概述

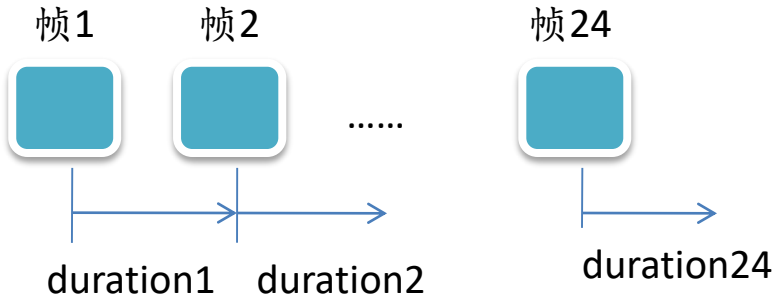
参考 参考

- 安卓提供了三种动画设计：
  - **补间(Tween Animation)动画**
    在给出控件的起止位置、大小、透明度和旋转角度后通过添加中间值的变化过程而产生动画效果。也称为视图动画(View Animation)。
  - **逐帧动画(Frame Animation)**
    通过用多幅图片替换显示而产生动画效果。也称为Drawable Animation。
  - **属性动画(Property Animation)**
    通过连续改变对象属性的属性值而产生动画效果。与补间动画只改变控件的四个属性的属性值不同的是属性动画可以改变对象的任何数值类型的属性的属性值。

补间动画、属性动画



逐帧动画



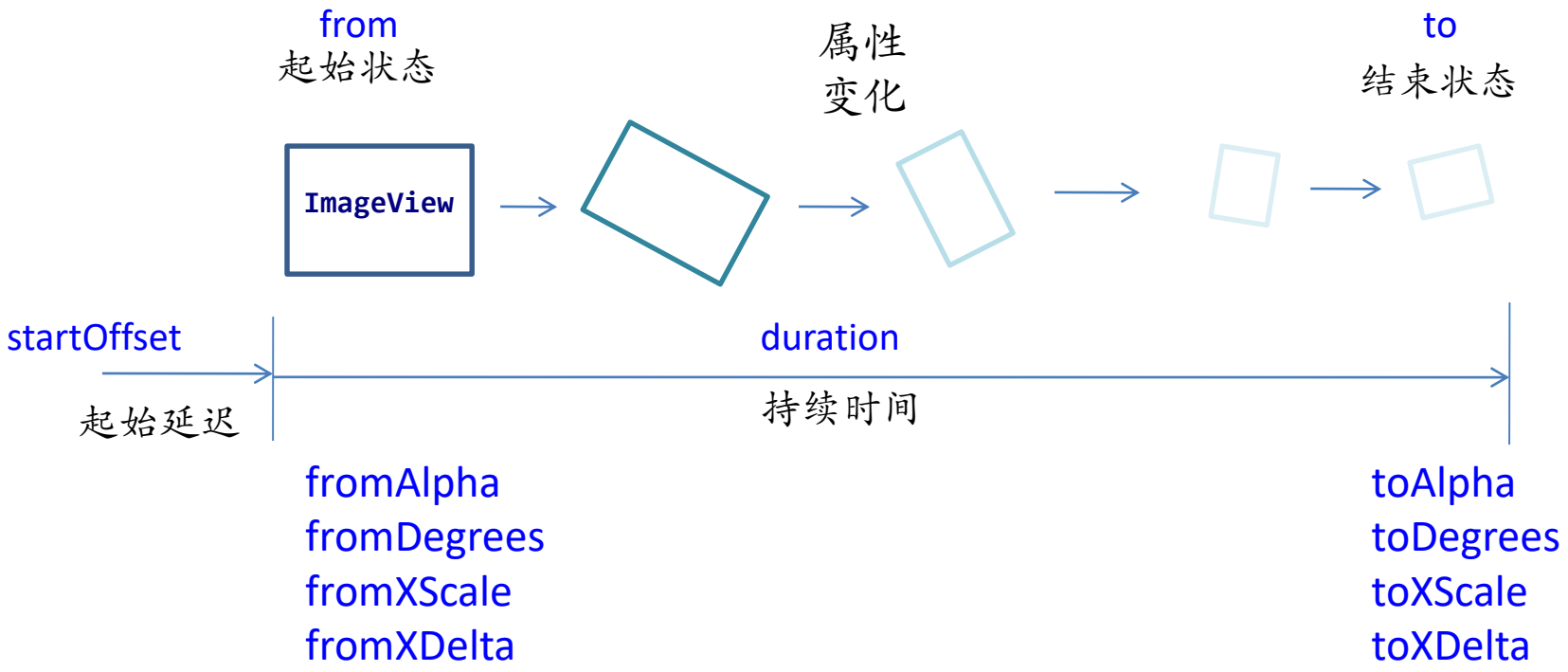
补间动画

参考 参考1 参考2
参考3 参考4

- **概述**

补间动画(Tween Animation)可以为控件做透明度动画(Alpha Animation)、旋转动画(Rotate Animation)、缩放动画(Scale Animation)、平移动画(Translate Animation)，也可以把这些动画组合起来(AnimationSet)。

每种动画都要定义一个起始状态、一个结束状态、持续时间，系统会补齐中间过程。下图展示了旋转动画、透明度动画、平移动画和缩放动画的组合。



参考

- **补间动画的基本属性**

AlphaAnimation

fromAlpha 开始透明度 toAlpha 结束透明度 取值：0.0 全透明 ~ 1.0 不透明

RotateAnimation

fromDegrees 起始角度 toDegrees 结束角度 取值：+n顺时针，-n逆时针， 0~n/360圈
pivotX 动画在X方向相对于物件的位置 ： 80%，80%p，300 。

ScaleAnimation

fromXScale 起始X坐标伸缩倍数 toXScale 结束X坐标伸缩倍数
pivotX 动画在X方向相对于物件的位置 ： 80%，80%p，300。

TranslateAnimation

fromXDelta 起始X坐标的相对偏移量 toXDelta 结束X坐标的相对偏移量。
取值: 80%,80%p,300。

- 相对于物件的取值类型：相对于自己Animation.RELATIVE_TO_SELF（80%），相对于父容器Animation.RELATIVE_TO_PARENT（80%p），相对于屏幕Animation.ABSOLUTE（300）。
- 对于pilotX和fromXDelta作为参数都分为取值类型和取值， pivotXType ， pivotXValue， fromXType，fromXValue。
- Y方向与X方向类似。

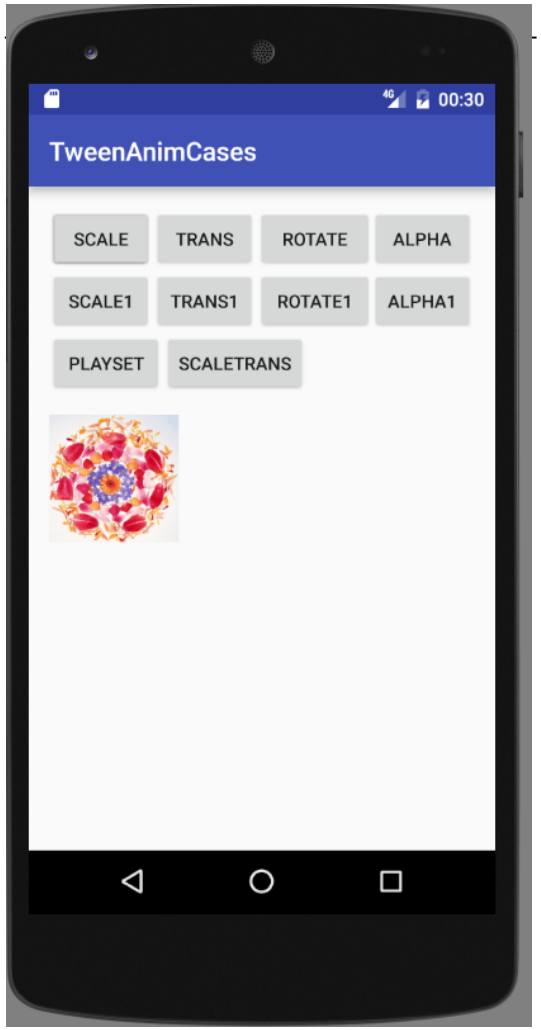
动画事件

- 实现补间动画的例子

*下面的例子省略了定位信息

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:id="@+id/activity_main">
    <Button
        android:text="Scale"
        android:id="@+id/btnScale"
        android:onClick="Scale" />
    <Button
        android:text="Scale1"
        android:id="@+id/btnScale1"
        android:onClick="Scale1"/>
    <Button
        android:text="Scale"
        android:id="@+id/button2"
        android:onClick="Scale" />
    <Button
        android:text="Scale"
        android:id="@+id/button"
        android:onClick="Scale" />
    <Button
        android:text="Rotate1"
        android:id="@+id/btnRotate1"
        android:onClick="Rotate1" />
```



```
    <Button  android:text="Alpha"
        android:id="@+id/btnAlpha"
        android:onClick="Alpha" />
    <Button
        android:text="Trans"
        android:id="@+id/btnTranslate"
        android:onClick="Translate" />
    <Button     android:text="Alpha1"
        android:id="@+id/btnAlpha1"
        android:onClick="Alpha1" />
    <Button    android:text="TRANS1"
        android:onClick="Translate1"
        android:id="@+id/btnTranslate1" />
    <Button    android:text="Rotate"
        android:id="@+id/btnRotate"
        android:onClick="Rotate" />
    <Button    android:text="ScaleTrans"
        android:id="@+id/btnReverseSet"
        android:onClick="reverse" />
    <Button    android:text="PlaySet"
        android:id="@+id/btnPlaySet"
        android:onClick="play" />

    <ImageView
        android:layout_width="100dp"
        android:layout_height="100dp"
        app:srcCompat="@drawable/flower"
        android:id="@+id/imgFlower" />
</RelativeLayout>
```

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    ImageView imgFlower;
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);  setContentView(R.layout.activity_main);
        imgFlower = (ImageView) findViewById(R.id.imgFlower);
    }
    public void Alpha(View vw){
        final Animation anim = AnimationUtils.loadAnimation(this, R.anim.alpha);
        imgFlower.startAnimation(anim);
    }
    public void Rotate(View vw){
        final Animation anim = AnimationUtils.loadAnimation(this, R.anim.rotate);
        imgFlower.startAnimation(anim);
    }
    public void Scale(View vw){
        final Animation anim = AnimationUtils.loadAnimation(this, R.anim.scale);
        imgFlower.startAnimation(anim);
    }
    public void Translate(View vw){
        final Animation anim = AnimationUtils.loadAnimation(this, R.anim.translate);
        imgFlower.startAnimation(anim);
    }
    public void Scale1(View vw){ …}
    public void Rotate1(View vw) {…}
    public void Alpha1(View vw){…}
    public void Translate1(View vw){…}
    public void play(View vw){…}
    public void reverse(View vw){…}
}
```

- **通过XML配置实现动画**

在XML设置好动画属性，然后通过*loadAnimation()装载动画，用*ImageView对象的*方法*startAnimation(anim)启动该动画。

```
ImageView imgFlower  = (ImageView) findViewById(R.id.imgFlower);
final Animation anim = AnimationUtils.loadAnimation(this, R.anim.rotate);
imgFlower.startAnimation(anim);   // 其它装载动画的Java程序见上一页
```

anim/rotate.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="3000"
    android:fromDegrees="0"
    android:toDegrees="1800"
    android:interpolator="@android:interpolator/accelerate_cubic"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="2000" />
```

xmlns：XML命名空间(NAMESPACE)

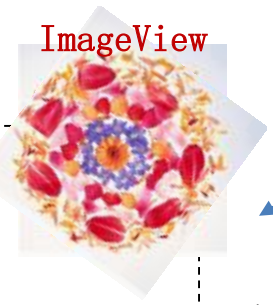
动画持续3000毫秒

从0旋转到1800度（5圈）

插值器

绕自身水平50%和垂直50%的位置（中心位置）

延迟2000毫秒开始动画



rotate.xml

- android:pivotX="50%p"和android:pivotY="50%p"表示绕父元素中心位置。
- 插值器指出的是旋转角度（从0到1800度）如何随时间变化：

（1） accelerate_cubic：立方加速

（2） *accelerate_interpolator：平方加速*

（3） linear_interpolator：线性变化

anim/alpha.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 定义不透明度变换-->
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="3000"
    android:fromAlpha="1"          取值0~1
    android:toAlpha="0.05"
    android:fillAfter="true"       结束时停留在最后画面
    android:interpolator="@android:interpolator/accelerate_cubic"/>
```

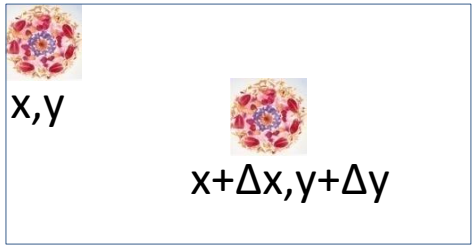
anim/scale.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromXScale="1.0"        缩放倍数
    android:fromYScale="1.0"
    android:toXScale="0.05"
    android:toYScale="0.05"
    android:duration="3000"
    android:pivotX="50%"
    android:pivotY="50%"
    android:interpolator="@android:anim/linear_interpolator" />
```

anim/translate.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<translate xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="3000"
    android:fromXDelta="0"
    android:toXDelta="50%p"
    android:fromYDelta="0"
    android:toYDelta="50%p"
    android:interpolator="@android:interpolator/accelerate_cubic"/>
```

x增加值（父元素宽度的一半）



- 通过Java程序实现动画

```
public void Alpha1(View vw){
    AlphaAnimation anim = new AlphaAnimation(1f,0.1f);
    anim.setDuration(3000);
    anim.setFillBefore(false);
    anim.setFillAfter(true);
    anim.setInterpolator(AnimationUtils.loadInterpolator(this,
                        android.R.anim.accelerate_decelerate_interpolator));
    imgFlower.startAnimation(anim);
}
```

```
public void Rotate1(View vw) {
    int method=0;
    //  fromDegrees,toDegrees,PivotXType,pivotX,PivotYType,pivotY
    RotateAnimation anim= new RotateAnimation(0,1845,
            Animation.RELATIVE_TO_SELF,0.5f,Animation.RELATIVE_TO_SELF,0.5f);
    if(method==1)
        anim = new RotateAnimation(0,1845,0,0);// RELATIVE_TO_SELF
    else if(method==2)
        anim = new RotateAnimation(0,1845); //pivotX=0,pivotY=0 (RELATIVE_TO_SELF)
    else if(method==3)
        anim = new RotateAnimation(0,1845,
           Animation.RELATIVE_TO_PARENT,0.5f,Animation.RELATIVE_TO_PARENT,0.5f);
    anim.setDuration(3000);
    anim.setFillAfter(true);
    anim.setInterpolator(AnimationUtils.loadInterpolator(this,
                            android.R.anim.accelerate_interpolator));
    imgFlower.startAnimation(anim);
}
```

method=1: fromDegrees,toDegrees,pivotX,pivotY (RELATIVE_TO_SELF)
method=2: fromDegrees,toDegrees   default：pivotX=0,pivotY=0 (RELATIVE_TO_SELF)

```
public void Scale1(View vw){
    //fromScaleX, toScaleX, fromScaleY, toScaleY, pivotX, piVotY
    ScaleAnimation anim = new ScaleAnimation(0, 2, 0, 3,
                    Animation.RELATIVE_TO_SELF, 4f, Animation.RELATIVE_TO_SELF, 4f);
    anim.setDuration(3000);
    anim.setFillAfter(true);
    anim.setInterpolator(AnimationUtils.loadInterpolator(this,
                                        android.R.anim.accelerate_interpolator));
    imgFlower.startAnimation(anim);
}
```

```
public void Translate1(View vw){
    TranslateAnimation  anim = new TranslateAnimation(Animation.RELATIVE_TO_SELF,
                    0f, Animation.RELATIVE_TO_SELF, 2f, Animation.RELATIVE_TO_SELF,
                    0f,Animation.RELATIVE_TO_SELF,3f);
    anim.setStartOffset(2000);
    anim.setDuration(3000);
    anim.setRepeatMode(Animation.REVERSE);
    anim.setRepeatCount(Animation.INFINITE);
    anim.setInterpolator(AnimationUtils.loadInterpolator(this,
                                        android.R.anim.anticipate_interpolator));
    imgFlower.startAnimation(anim);
}
```

- 使用动画集

如果想同时进行几个动画，则需要动画集（set）元素或动画集对象。

anim/anim.xml

```
<?xml version="1.0" encoding="UTF-8"?><!-- 指定动画匀速改变 -->
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"
    android:startOffset="3000">
    <scale android:duration="3000"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:interpolator="@android:anim/linear_interpolator"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toXScale="0.01"
        android:toYScale="0.01" />
    <alpha android:duration="3000"
        android:fromAlpha="1"
        android:toAlpha="0.05" />
    <rotate android:duration="3000"
        android:fromDegrees="0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toDegrees="1800" />
    <translate android:duration="3000"
        android:fromYDelta="0%"
        android:interpolator="@android:interpolator/accelerate_cubic"
        android:toYDelta="80%" />
</set>
```

动画集元素的播放方法与单个元素相同：

```
public void playset(View vw){
    // 加载第一份动画资源
    final Animation anims = AnimationUtils.loadAnimation(this, R.anim.anim);
    // 设置动画结束后保留结束状态
    animSet.setFillAfter(true);   //默认为false
    imgFlower.startAnimation(anims);
}
```

动画集对象可以加入多个动画对象，进行播放：

```
public void reverse(View vw){
    final Animation scale = AnimationUtils.loadAnimation(this, R.anim.scale);
    scale.setRepeatCount(Animation.INFINITE);
    scale.setRepeatMode(Animation.REVERSE);
    scale.setFillAfter(true);  // 设置动画结束后保留结束状态

    final Animation trans = AnimationUtils.loadAnimation(this, R.anim.translate);
    trans.setRepeatCount(Animation.INFINITE);
    trans.setRepeatMode(Animation.REVERSE);
    trans.setFillAfter(true);  // 设置动画结束后保留结束状态

    AnimationSet animationSet = new AnimationSet(true);
    animationSet.addAnimation(scale);
    animationSet.addAnimation(trans);
    imgFlower.startAnimation(animationSet);
}
```

逐帧动画

- **概述**

逐帧动画(Frame Animation)可以得到播放幻灯片一样的效果。这种动画用元素animation-list定义每一帧的图像和播放时间，并把该元素作为一个ImageView对象的背景：

```
<animation-list
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:oneshot="false">
   <item android:drawable="@drawable/zombie1" android:duration="120" />
    ……
   <item android:drawable="@drawable/zombie20" android:duration="120" />
</animation-list>
```

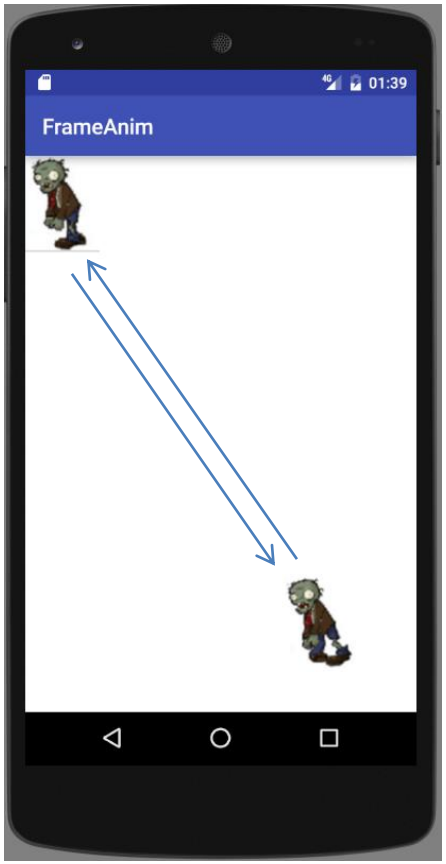
帧图像　　帧持续时间

oneshot：是否只播放一次

- 项目**FrameAnim**

drawable/zombie.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义动画循环播放 -->
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
   android:oneshot="false">
   <item android:drawable="@drawable/zombie1" android:duration="120" />
   <item android:drawable="@drawable/zombie2" android:duration="120" />
   <item android:drawable="@drawable/zombie3" android:duration="120" />
   <item android:drawable="@drawable/zombie4" android:duration="120" />
   <item android:drawable="@drawable/zombie5" android:duration="120" />
   <item android:drawable="@drawable/zombie6" android:duration="120" />
   <item android:drawable="@drawable/zombie7" android:duration="120" />
   <item android:drawable="@drawable/zombie8" android:duration="120" />
   <item android:drawable="@drawable/zombie9" android:duration="120" />
   <item android:drawable="@drawable/zombie10" android:duration="120" />
   <item android:drawable="@drawable/zombie11" android:duration="120" />
   <item android:drawable="@drawable/zombie12" android:duration="120" />
   <item android:drawable="@drawable/zombie13" android:duration="120" />
   <item android:drawable="@drawable/zombie14" android:duration="120" />
   <item android:drawable="@drawable/zombie15" android:duration="120" />
   <item android:drawable="@drawable/zombie16" android:duration="120" />
   <item android:drawable="@drawable/zombie17" android:duration="120" />
   <item android:drawable="@drawable/zombie18" android:duration="120" />
   <item android:drawable="@drawable/zombie19" android:duration="120" />
   <item android:drawable="@drawable/zombie20" android:duration="120" />
</animation-list>
```



activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#fff">
    <ImageView
        android:id="@+id/zombie"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/zombie"
        />
</LinearLayout>
```

MainActivity.java

```java
public class MainActivity extends AppCompatActivity {
    @Override protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);   setContentView(R.layout.activity_main);
    final ImageView imageView = (ImageView) findViewById(R.id.zombie);
    final AnimationDrawable zombie =
              (AnimationDrawable) imageView.getBackground();
    zombie.start();        // 开始逐帧动画
    // 平移动画：x从0平移到600，y从0平移到900
    TranslateAnimation anim = new TranslateAnimation(0, 600, 0, 900);
    anim.setDuration(20000);                          // 动画次序时间为20秒
    anim.setRepeatMode(Animation.REVERSE);            // 到了动画末尾，再逆向变换
    anim.setRepeatCount(-1);                          // 无穷循环
    imageView.startAnimation(anim);                   // 开始位移动画
  }
}
```

属性动画

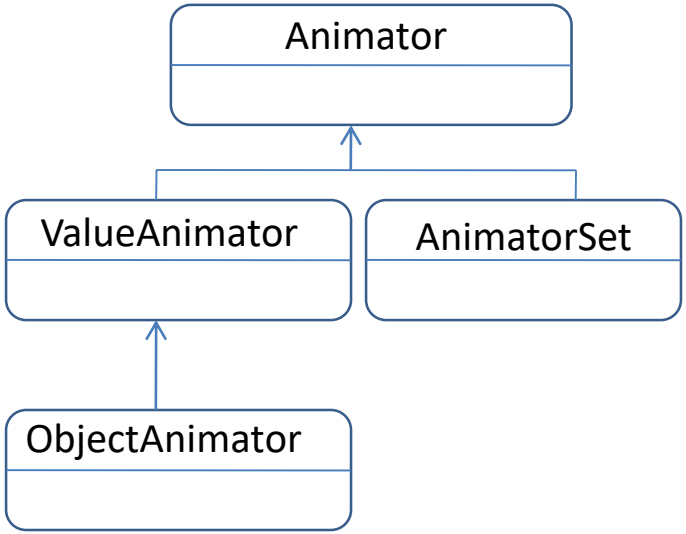
- **概述**

参考 参考

属性动画可以看成是增强版的补间动画。补间动画只定义两个关键帧，只能对UI控件的透明度、旋转、缩放、位移执行动画，属性动画可以对自定义对象的任何数值属性执行动画。

属性动画涉及四个类：

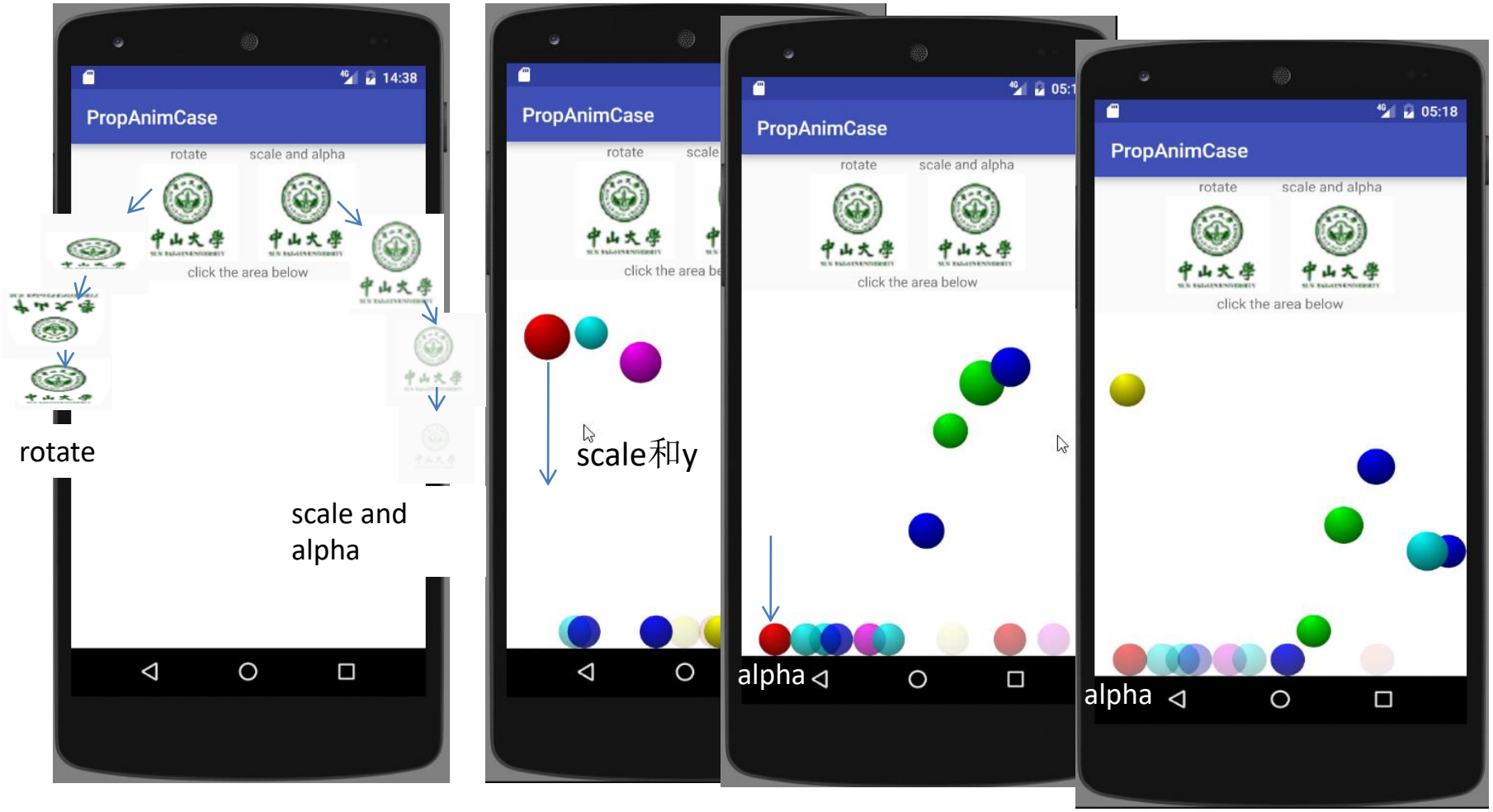
- **Animator**
  属性动画的基类。可以设置动画的开始延迟、持续时间、插值方式、动画重复次数、动画重复行为，并保持者当前状态。
- **ValueAnimator**
  Animator的子类，属性动画的时间引擎，负责计算各个帧的属性值和产生属性更新事件。
- **ObjectAnimator**
  ValueAnimator的子类，对指定对象及其属性执行动画。
- **AnimatorSet**
  Animator 的子类， 用于组合多个Animator，并指定多个Animator是按次序播放，还是同时播放。



属性动画
XML定义

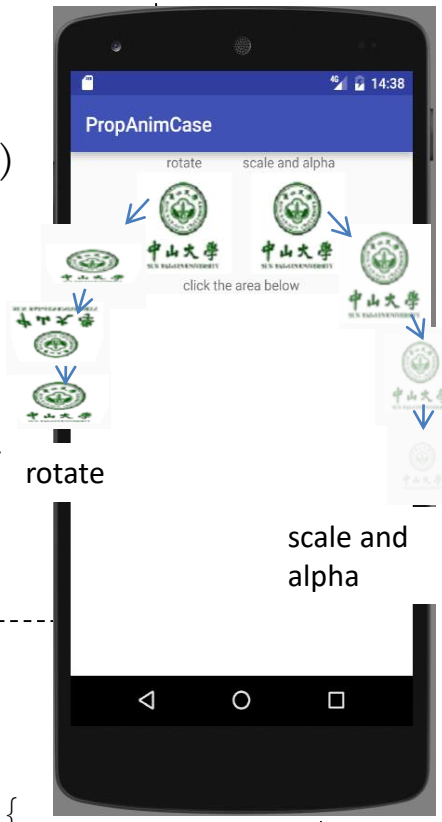
- 属性动画的例子

这个例子包含三种属性动画方法。一是直接改变控件原有属性；二是为控件定义新属性并用属性动画改变它；三是定义一个新类，通过改变该类的某些属性值实现属性动画。



方法1、直接改变一个控件的原有属性，下例中点击图片会产生旋转效果。

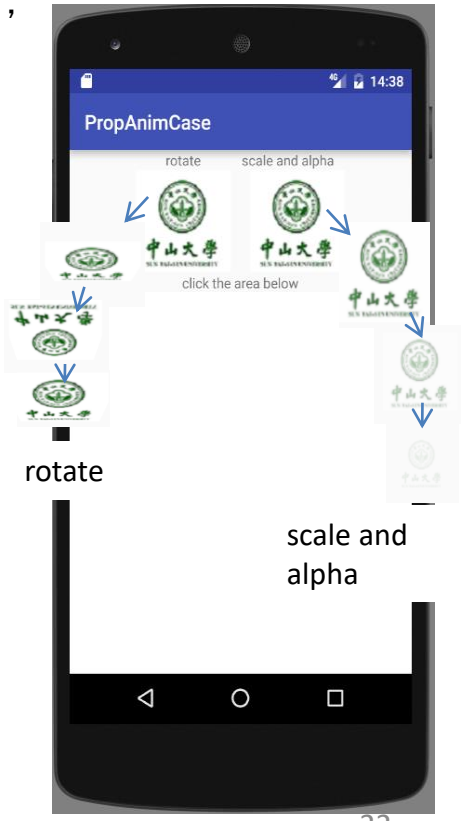
```
final ImageView imgView1 = (ImageView) findViewById(R.id.sysu1);
imgView1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        ObjectAnimator.ofFloat(v, "rotationX", 0.0F, 360.0F)
                .setDuration(2000)
                .start();

    }
});
```



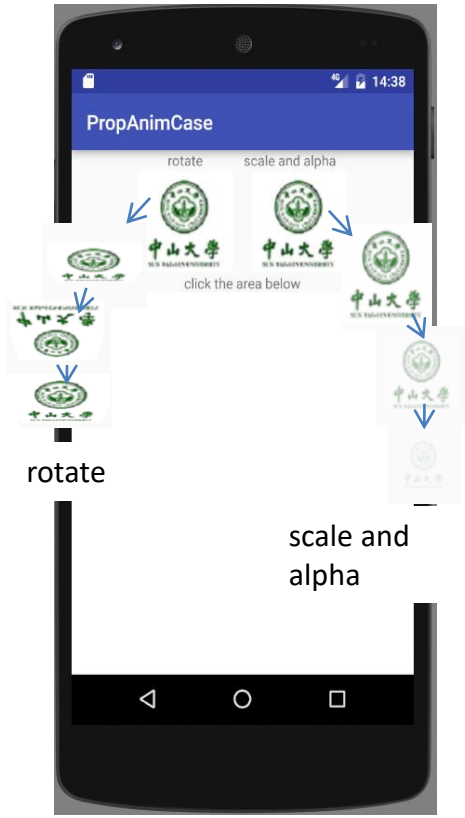
方法2、改变一个控件的用户自定义属性。

```
ImageView imgView2 = (ImageView) findViewById(R.id.sysu2);
imgView2.setOnClickListener(new View.OnClickListener() {
                                @Override
                                public void onClick(View v){
                                    startAnim();
                                }
                            }

);
```

```
void startAnim() {
    final ImageView imgView2 = (ImageView) findViewById(R.id.sysu2);
    ObjectAnimator anim1 = new ObjectAnimator()
            .ofFloat(imgView2, "scale", 1.0f, 0.5f)
            .setDuration(8000);
    anim1.start();
    anim1.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
        @Override                //10ms一次
        public void onAnimationUpdate(ValueAnimator animation) {
            float scale = (float) animation.getAnimatedValue();
            imgView2.setScaleX(scale);
            imgView2.setScaleY(scale);
        }
    });
    ObjectAnimator anim2 = new ObjectAnimator()
            .ofInt(imgView2, "alpha", 255, 0)
            .setDuration(8000);
    anim2.start();
```

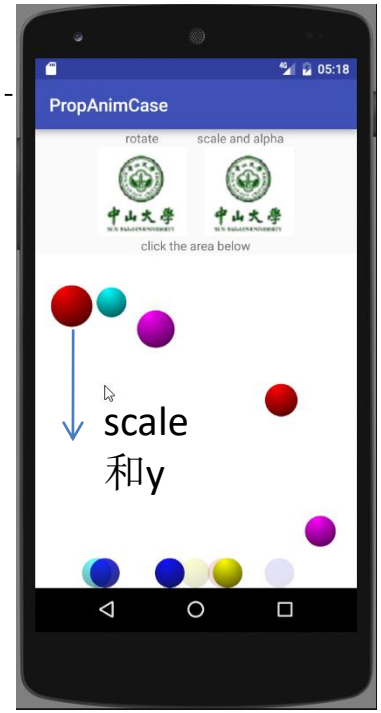


```
    anim1.addListener(new Animator.AnimatorListener() {
        @Override
        public void onAnimationStart(Animator animation) { }
        @Override
        public void onAnimationEnd(Animator animation) {
            imgView2.setAlpha(255);
            imgView2.setScaleX(1);
            imgView2.setScaleY(1);
        }

        @Override
        public void onAnimationCancel(Animator animation) {
        }

        @Override
        public void onAnimationRepeat(Animator animation) {
        }
    });
}
```



*也可以用anim1.setAnimationListener(…)

方法3、改变自定义对象的属性。




```
public class ShapeHolder {
    private float x = 0, y = 0, radius = 0;
    private int color = Color.BLUE;
    private int alpha = 255;

    public ShapeHolder(float x, float y,
                       float radius, int color) {
        this.x = x;  this.y = y;
        this.radius = radius;  this.color = color;
    }
    public float getX() { return x; }
    public void setX(float x) { this.x = x; }
    public float getY() { return y; }
    public void setY(float y) { this.y = y; }
    public float getRadius() { return radius; }
    public void setRadius(float radius) { this.radius = radius; }
    public int getColor() { return color; }
    public int getAlpha() { return alpha; }
    public void setAlpha(int alpha) { this.alpha = alpha;}
}
```

```
class MyAnimationView extends View implements ValueAnimator.AnimatorUpdateListener {
    static final float DROP_TIME = 2000;  // 小球从触摸位置下落到屏幕底端的总时间
    static final int ALHPA_TIME = 2000;  // 定义小球透明度变化的时间
    static final int END_RADIUS = 100;  // 定义小球大小变化
    final int[] colors = {Color.RED, Color.BLUE, Color.CYAN,
                          Color.GREEN, Color.MAGENTA, Color.YELLOW};
    public final ArrayList<ShapeHolder> balls = new ArrayList<ShapeHolder>();
    public MyAnimationView(Context context) {
        super(context);
        setBackgroundColor(Color.WHITE);
    }
    // 通过ObjectAnimator在一段时间内逐步改变对象的某些属性值
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        if (event.getAction() != MotionEvent.ACTION_DOWN) {//MotionEvent.ACTION_MOVE
            return false;
        }
        //获得随机的半径（不小于END_RADIUS）和随机的颜色（六种颜色选一种）
        Random rnd = new Random();
        float radius = rnd.nextFloat() * END_RADIUS + END_RADIUS;
        int colorIndex = rnd.nextInt(colors.length);
```

```
float viewHeight = (float) getHeight();      // 获取控件的高度
float touchX = event.getX();
float touchY = event.getY();
int duration = (int) (DROP_TIME * ((viewHeight - touchY) / viewHeight));

ShapeHolder ball=new ShapeHolder(touchX, touchY, radius, colors[colorIndex]);
balls.add(ball);

// ball的属性y由startY变到endY
ObjectAnimator fallAnim = ObjectAnimator.ofFloat(ball, "y",
                                touchY, viewHeight - END_RADIUS);
fallAnim.setDuration(duration);        // 设置下落动画持续时间
fallAnim.setInterpolator(new AccelerateInterpolator());   // 采用加速插值
fallAnim.addUpdateListener(this);    // 监听属性值改变事件(每10秒一次)

// ball的属性radius由radius变到END_RADIUS
ObjectAnimator radiusAnim = ObjectAnimator.ofFloat(ball, "radius",
                                                radius, END_RADIUS);
radiusAnim.setDuration(duration);              // 设置渐隐动画持续时间
radiusAnim.setInterpolator(new LinearInterpolator()); // 动画采用线性插值
//radiusAnim.addUpdateListener(this);                  // 监听属性值改变事件
```

```
    // ball的属性alpha由255变到0
    ObjectAnimator fadeAnim = ObjectAnimator.ofInt(ball, "alpha", 255, 0);
    fadeAnim.setDuration(ALHPA_TIME);                    // 设置渐隐动画持续时间
    fadeAnim.setInterpolator(new LinearInterpolator());// 动画采用线性插值
    fadeAnim.addUpdateListener(this);                    // 监听属性值改变事件
    fadeAnim.addListener(new AnimatorListenerAdapter() {
        @Override   // 动画结束时删除ball
        public void onAnimationEnd(Animator animation) {
            balls.remove(((ObjectAnimator) animation).getTarget());
        }
    });
    AnimatorSet animatorSet = new AnimatorSet();    // 用AnimatorSet来组合
    animatorSet.play(fallAnim)
            .with(radiusAnim)
            .before(fadeAnim); // 先播完fallAnim和radiusAnim再进行fadeAnim
    animatorSet.start();        // 播放动画
    return true;
}
```

```
    @Override
    protected void onDraw(Canvas canvas) {
        for (ShapeHolder ball : balls) {
            canvas.save();      // 保存canvas的当前坐标系统
            OvalShape circle = new OvalShape();  // 创建一个椭圆
            circle.resize(ball.getRadius(), ball.getRadius()); //设置宽和高
            ShapeDrawable drawable = new ShapeDrawable(circle);
            RadialGradient gradient = new RadialGradient(37.5f, 12.5f,
                    ball.getRadius(), ball.getColor(),
                    0xff4E4E4E & ball.getColor(), Shader.TileMode.CLAMP);
            drawable.getPaint().setShader(gradient);
            drawable.getPaint().setAlpha(ball.getAlpha());
            canvas.translate(ball.getX(), ball.getY());
            drawable.draw(canvas); // 在Canvas上绘制drawable对象
            canvas.restore();      // 恢复Canvas坐标系统
        }
    }

    @Override  //属性值改变事件（默认地10ms发生一次）
    public void onAnimationUpdate(ValueAnimator animation) {
        this.invalidate();  // 产生重绘消息
    }
}
```

自定义动画

项目CustomTween

自定义补间动画。对一个TextView对象做3D旋转动画

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);

     WindowManager windowManager =
               (WindowManager)getSystemService(WINDOW_SERVICE);

     Display display = windowManager.getDefaultDisplay();
     DisplayMetrics metrice = new DisplayMetrics();

     //获取屏幕的宽(metrice.xdpi/2)和高(metrice.ydpi/2)
     display.getMetrics(metrice);

     TextView tv = (TextView) findViewById(R.id.tv);
     tv.setTextSize(60);
     tv.setAnimation(new MyAnimation(metrice.xdpi/2, metrice.ydpi/2, 20500));
  }
}
```

Activity_Main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"   >
    <TextView
        android:id="@+id/tv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, world!"/>
</LinearLayout>
```

MyAnimation.java

```
public class MyAnimation extends Animation {
   private float centerX,centerY;
   private int duration; // 定义动画的持续事件
   private Camera camera = new Camera();
   public MyAnimation(float x, float y, int duration){
      this.centerX = x;
      this.centerY = y;
      this.duration = duration;
   }
   @Override
   public void initialize(int width, int height, int parentWidth, int parentHeight){
      super.initialize(width, height, parentWidth, parentHeight);
      setDuration(duration);    setFillAfter(true);   // 动画结束后保留最后的图像
      setInterpolator(new LinearInterpolator());
   }
```

```
@Override
  protected void applyTransformation(float interpolatedTime, Transformation t){
     /* interpolatedTime代表了抽象的动画持续时间，取值0（动画开始时）～1（动画结束时）
     *  Transformation表示对目标组件所做的变化. */
     camera.save();
     camera.translate(100.0f - 100.0f * interpolatedTime,  // 控制X、Y、Z上的偏移
           150.0f * interpolatedTime - 150,  80.0f - 80.0f * interpolatedTime);
     // 设置根据interpolatedTime时间在Y轴上旋转不同角度。
     camera.rotateY(360 * interpolatedTime);
     // 设置根据interpolatedTime时间在X轴上旋转不同角度
     camera.rotateX(360 * interpolatedTime);
     // 获取Transformation参数的Matrix对象
     Matrix matrix = t.getMatrix(); camera.getMatrix(matrix);
     matrix.preTranslate(-centerX, -centerY);
     matrix.postTranslate(centerX, centerY);
     camera.restore();
  }
}
```

动画总结

方法

补间动画

(1)

```
final ImageView flower = (ImageView) findViewById(R.id.flower);
final Animation anim = AnimationUtils.loadAnimation(this, R.anim.anim);
flower.startAnimation(anim);
```

(2)

```
TranslateAnimation transanim = new TranslateAnimation(0,300,0,300);
Transanim.strat();
```

(3)

```
AnimationSet animationSet = new AnimationSet(true);
animationSet.addAnimation(transanim);
flower.startAnimation(animationSet);
```

逐帧动画

```
final ImageView imageView = (ImageView) findViewById(R.id.zombie);
final AnimationDrawable zombie =
              (AnimationDrawable) imageView.getBackground();
zombie.start();       // 开始逐帧动画
```

属性动画

```
ValueAnimator fadeAnim =
        ObjectAnimator.ofInt(ball,"alpha",255,0);
fadeAnim.setDuration(2000);                    // 设置动画持续时间
fadeAnim.setInterpolator(new LinearInterpolator());
fadeAnim.addUpdateListener(this);
fadeAnim.addListener(new AnimatorListenerAdapter(){
            @Override
            public void onAnimationEnd(Animator animation) {
                balls.remove(((ObjectAnimator)animation).getTarget());
            }
        });

AnimatorSet animatorSet = new AnimatorSet();
//指定在播放fadeAnim之前先播放的fallAnim动画
animatorSet.play(fallAnim)
            .before(fadeAnim);
animatorSet.start();
```

附录1、Animation属性

xml属性	java方法	解释
android:detachWallpaper	setDetachWallpaper(boolean)	是否在壁纸上运行
android:duration	setDuration(long)	动画持续时间，毫秒为单位
android:fillAfter	setFillAfter(boolean)	控件动画结束时是否保持动画最后的状态
android:fillBefore	setFillBefore(boolean)	控件动画结束时是否还原到开始动画前的状态
android:fillEnabled	setFillEnabled(boolean)	与android:fillBefore效果相同
android:interpolator	setInterpolator(Interpolator)	设定插值器（指定的动画效果，譬如回弹等）
android:repeatCount	setRepeatCount(int)	重复次数（<0表示无穷循环）
android:repeatMode	setRepeatMode(int)	重复类型有两个值，reverse表示倒序回放，restart表示从头播放
android:startOffset	setStartOffset(long)	调用start函数之后等待开始运行的时间，单位为毫秒
android:zAdjustment	setZAdjustment(int)	表示被设置动画的内容运行时在Z轴上的位置（top/bottom/normal），默认为normal

Alpha属性

xml属性	java方法	解释
android:fromAlpha	AlphaAnimation(float fromAlpha, float toAlpha)	动画开始的透明度（0.0是全透明，1.0是不透明）
android:toAlpha		动画结束的透明度，同上

Rotate属性

xml属性	java方法	解释
android:fromDegrees	RotateAnimation(float fromDegrees, float toDegrees, float pivotX, float pivotY)	旋转开始角度，正代表顺时针度数，负代表逆时针度数
android:toDegrees		旋转结束角度
android:pivotX		中心X坐标.
android:pivotY		中心Y坐标.

* 中心点(android:pivotX, android:pivotY)取值: 数值、百分数、百分数p，譬如pivotX为50表示以当前View左上角坐标加50px为中心点、50%表示以当前View的左上角加上当前View宽的50%做为中心点。

Scale属性

xml属性	java方法	解释
android:fromXScale	ScaleAnimation(float fromX, float toY, float fromY, float toY, float pivotX, float pivotY)	初始X轴缩放比例，1.0表示无变化
android:toXScale		结束X轴缩放比例
android:fromYScale		初始Y轴缩放比例
android:toYScale		结束Y轴缩放比例
android:pivotX		缩放起点X轴坐标
android:pivotY		缩放起点Y轴坐标

Translate属性

xml属性	java方法	解释
android:fromXDelta	TranslateAnimation(float fromXDelta, toXDelta, float fromYDelta, toYDelta)	起始点X轴坐标。50%p表示以当前View的左上角加上父控件宽高的50%做为初始点
android:fromYDelta		起始点Y轴从标，同上规律
android:toXDelta		结束点X轴坐标，同上规律
android:toYDelta		结束点Y轴坐标，同上规律

Animation类的方法

reset()	重置Animation的初始化
cancel()	取消Animation动画
start()	开始Animation动画
setAnimationListener(AnimationListener listener)	给当前Animation设置动画监听
hasStarted()	判断当前Animation是否开始
hasEnded()	判断当前Animation是否结束

View类的常用动画操作方法

startAnimation(Animation animation)	对当前View开始设置的Animation动画
clearAnimation()	取消当View在执行的Animation动画

补间动画执行之后并未改变View的真实布局属性值。切记这一点，譬如我们在Activity中有一个Button在屏幕上方，我们设置了平移动画移动到屏幕下方然后保持动画最后执行状态呆在屏幕下方，这时如果点击屏幕下方动画执行之后的Button是没有任何反应的，而点击原来屏幕上方没有Button的地方却响应的是点击Button的事件。

Interpolator接口取值：

java类	xml id值	描述
AccelerateDecelerateInterpolator	@android:anim/accelerate_decelerate_interpolator	动画始末速率较慢，中间加速
AccelerateInterpolator	@android:anim/accelerate_interpolator	动画开始速率较慢，之后慢慢加速
AnticipateInterpolator	@android:anim/anticipate_interpolator	开始的时候从后向前甩
AnticipateOvershootInterpolator	@android:anim/anticipate_overshoot_interpolator	类似上面AnticipateInterpolator
BounceInterpolator	@android:anim/bounce_interpolator	动画结束时弹起
CycleInterpolator	@android:anim/cycle_interpolator	循环播放速率改变为正弦曲线
DecelerateInterpolator	@android:anim/decelerate_interpolator	动画开始快然后慢
LinearInterpolator	@android:anim/linear_interpolator	动画匀速改变
OvershootInterpolator	@android:anim/overshoot_interpolator	向前弹出一定值之后回到原来位置
PathInterpolator		新增，定义路径坐标后按照路径坐标来跑。

可以自定义插值器:

```
<set android:interpolator="@android:anim/accelerate_interpolator">
    ...
 </set>
```

AnimationSet

- startOffset　　动画开始前的等待时间(ms)
- duration　　动画持续时间(ms)，默认值是300ms 。
- repeatCount　　动画重复次数。 取值：infinite(-1)或重复次数
- Interpolator　　动画插值方式：*accelerate_interpolator， linear_interpolator*
- repeatMode　　重复下次动画时是从头开始(restart)还是反向播放(reverse)。
- fillAfter　　是否保持动画结束时的状态
- fillBefore　　动画结束时是否还原起始的状态
- fillEnabled　　与android:fillBefore效果相同
- zAdjustment　　动画内容在Z轴上的位置top/bottom/normal（默认）
- detachWallpaper 是否在壁纸上运行

- addAnimation(Animation anim) 增加一个动画
- setAnimationListener(AnimationListener listener)　　动画监听
- reset()　　重启动画。cancel() 取消动画，start() 开始动画
- hasStarted()　　动画是否开始　　hasEnded()动画是否结束
- startAnimation(Animation anim)　　设置并开始动画
- clearAnimation() 清除动画