Android动画学习笔记-Android Animation

3.0以前，android支持两种动画模式，tween animation,frame animation，在android3.0中又引入了一个新的动画系统：property animation，这三种动画模式在SDK中被称为property animation,view animation,drawable animation。 可通过 NineOldAndroids 项目在3.0之前的系统中使用Property Animation

1. View Animation (Tween Animation)

View Animation (Tween Animation) ：补间动画，给出两个关键帧，通过一些算法将给定属性值在给定的时间内在两个关键帧间渐变。

View animation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改变。

而且对于View animation，它只是改变了View对象绘制的位置，而没有改变View对象本身，比如，你有一个Button，坐标 (100,100) ，Width:200,Height:50，而你有一个动画使其变为Width：100，Height：100，你会发现动画过程中触发按钮点击的区域仍是(100,100)-(300,150)。

View Animation就是一系列View形状的变换，如大小的缩放，透明度的改变，位置的改变，动画的定义既可以用代码定义也可以用XML定义，当然，建议用XML定义。

可以给一个View同时设置多个动画，比如从透明至不透明的淡入效果，与从小到大的放大效果，这些动画可以同时进行，也可以在一个完成之后开始另一个。

用XML定义的动画放在/res/anim/文件夹内，XML文件的根元素可以为<alpha>,<scale>,<translate>,<rotate>,interpolator元素或<set>(表示以上几个动画的集合，set可以嵌套)。默认情况下，所有动画是同时进行的，可以通过startOffset属性设置各个动画的开始偏移 (开始时间) 来达到动画顺序播放的效果。

可以通过设置interpolator属性改变动画渐变的方式，如AccelerateInterpolator，开始时慢，然后逐渐加快。默认为AccelerateDecelerateInterpolator。

定义好动画的XML文件后，可以通过类似下面的代码对指定View应用动画。

```
ImageView spaceshipImage = (ImageView)findViewById(R.id.spaceshipImage);
Animation hyperspaceJumpAnimation=AnimationUtils.loadAnimation(this, R.anim.hyperspace_jump);
spaceshipImage.startAnimation(hyperspaceJumpAnimation);
```

2. Drawable Animation (Frame Animation)

Drawable Animation (Frame Animation) ：帧动画，就像GIF图片，通过一系列Drawable依次显示来模拟动画的效果。在XML中的定义方式如下：

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">
    <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3" android:duration="200" />
</animation-list>
```

必须以<animation-list>为根元素，以<item>表示要轮换显示的图片，duration属性表示各项显示的时间。XML文件要放在/res/drawable/目录下。示例：

```
protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        imageView = (ImageView) findViewById(R.id.imageView1);
        imageView.setBackgroundResource(R.drawable.drawable_anim);
        anim = (AnimationDrawable) imageView.getBackground();
    }

    public boolean onTouchEvent(MotionEvent event) {
```


个人信息
angeldeviljy@gmail.com
我的个人博客
我的CSDN
昵称：AngelDevil
园龄：5年2个月
粉丝：301
关注：3
+加关注

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

我的标签
android (30) 快速androi
binder (3) Java (2) Java
Java虚拟机 (1) JVM (1)
layout_weight (1) loope
message (1) 更多

随笔分类(50)
android(30)
Android Frameworks(8)
C++(1)
Java(1)
JavaScript(2)
程序设计(1)
读书笔记(2)
技术相关(1)
快速Android开发系列(4)

积分与排名
积分 - 99351
排名 - 1887

最新评论
1. Re:快速Android开
Volley
(请

```
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            anim.stop();
            anim.start();
            return true;
        }
        return super.onTouchEvent(event);
    }
```

我在实验中遇到两点问题：

1. 要在代码中调用Imageview的setBackgroundResource方法，如果直接在XML布局文件中设置其src属性当触发动画时会FC。
2. 在动画start()之前要先stop()，不然在第一次动画之后会停在最后一帧，这样动画就只会触发一次。
3. 最后一点是SDK中提到的，不要在onCreate中调用start，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动画的话，可以在onWindowFoucsChanged()中调用start()。

3. Property Animation

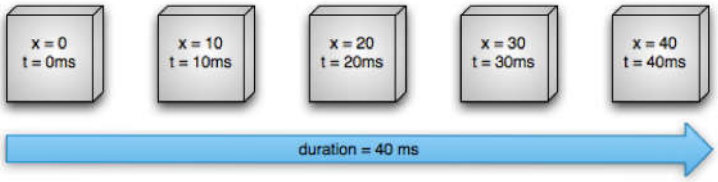
属性动画，这个是在Android 3.0中才引进的，以前学WPF时里面的动画机制好像就是这个，它更改的是对象的实际属性，在View Animation（Tween Animation）中，其改变的是View的绘制效果，真正的View的属性保持不变，比如无论你在对话中如何缩放Button的大小，Button的有效点击区域还是没有应用动画时的区域，其位置与大小都不变。而在Property Animation中，改变的是对象的实际属性，如Button的缩放，Button的位置与大小属性值都改变了。而且Property Animation不止可以应用于View，还可以应用于任何对象。Property Animation只是表示一个值在一段时间内的改变，当值改变时要做什么事情完全是你自己决定的。

在Property Animation中，可以对动画应用以下属性：

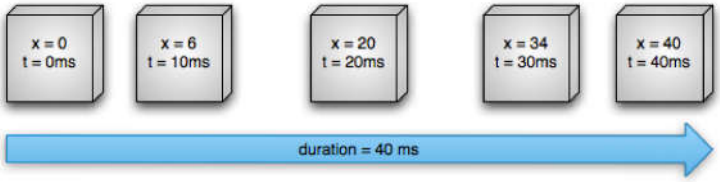
- Duration：动画的持续时间
- TimeInterpolation：属性值的计算方式，如先快后慢
- TypeEvaluator：根据属性的开始、结束值与TimeInterpolation计算出的因子计算出当前时间的属性值
- Repeat Count and behavoir：重复次数与方式，如播放3次、5次、无限循环，可以此动画一直重复，或播放完时再反向播放
- Animation sets：动画集合，即可以同时对一个对象应用几个动画，这些动画可以同时播放也可以对不同动画设置不同开始偏移
- Frame refreash delay：多少时间刷新一次，即每隔多少时间计算一次属性值，默认为10ms，最终刷新时间还受系统进程调度与硬件的影响

3.1 Property Animation的工作方式

对于下图的动画，这个对象的X坐标在40ms内从0移动到40 pixel.按默认的10ms刷新一次，这个对象会移动4次，每次移动40/4=10pixel。



也可以改变属性值的改变方法，即设置不同的interpolation，在下图中运动速度先逐渐增大再逐渐减小



下图显示了与上述动画相关的关键对象

不错

2. Re:快速Android开发系
EventBus

@sclgxt不需要注册，
定义一个不会用到的onEve
果onEvent(Object event)
用register如果在当前类及
一个onEvent开头的方法都

--

3. Re:快速Android开发系
EventBus

Fragment之中可以使
个父类的Fragment之中注
还需注册吗

4. Re:快速Android开发系
EventBus

介绍的很详细，非常感

--Alex_

5. Re:快速Android开发系
Android-Async-Http

好！！！

--民

阅读排行榜

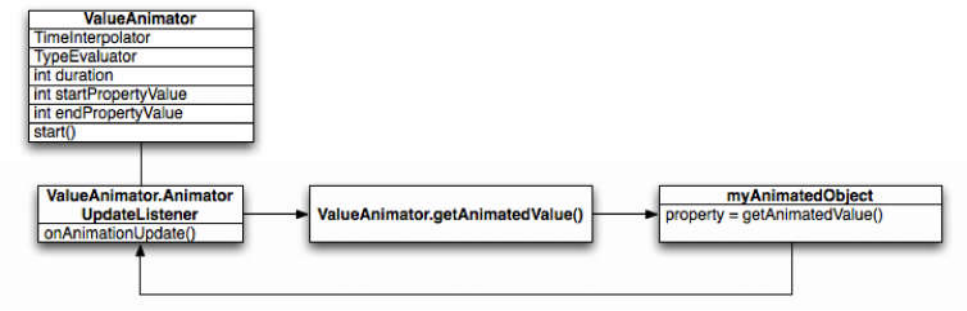
1. Android动画学习笔记-A
Animation(123949)
2. Android自定义对话框(D
置,大小(67899)
3. 快速Android开发系列网
Android-Async-Http(47(
4. 自定义SimpleAdapter(
5. 快速Android开发系列通
EventBus(40016)

评论排行榜

1. Git与Repo入门(50)
2. 快速Android开发系列通
EventBus(33)
3. Android动画学习笔记-A
Animation(17)
4. Android中自定义样式与
造函数中的第三个参数defS
(15)
5. android中layout_weig
(11)

推荐排行榜

1. Git与Repo入门(132)
2. Android动画学习笔记-A
Animation(62)
3. 快速Android开发系列通
EventBus(28)
4. Android中自定义样式与
造函数中的第三个参数defS
(20)
5. 快速Android开发系列网
Android-Async-Http(15)



ValueAnimator 表示一个动画，包含动画的开始值，结束值，持续时间等属性。

ValueAnimator封装了一个TimeInterpolator，TimeInterpolator定义了属性值在开始值与结束值之间的插值方法。

ValueAnimator还封装了一个TypeAnimator，根据开始、结束值与TimeIniterpolator计算得到的值计算出属性值。

ValueAnimator根据动画已进行的时间跟动画总时间（duration）的比计算出一个时间因子（0~1），然后根据TimeInterpolator计算出另一个因子，最后TypeAnimator通过这个因子计算出属性值，如上例中10ms时：

首先计算出时间因子，即经过的时间百分比：t=10ms/40ms=0.25

经插值计算(inteplator)后的插值因子:大约为0.15，上述例子中用了AccelerateDecelerateInterpolator，计算公式为（input即为时间因子）：

```
(Math.cos((input + 1) * Math.PI) / 2.0f) + 0.5f;
```

最后根据TypeEvaluator计算出在10ms时的属性值：0.15*（40-0）=6pixel。上例中TypeEvaluator为FloatEvaluator，计算方法为 ：

```
public Float evaluate(float fraction, Number startValue, Number endValue) {
    float startFloat = startValue.floatValue();
    return startFloat + fraction * (endValue.floatValue() - startFloat);
}
```

参数分别为上一步的插值因子，开始值与结束值。

3.2 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：

1. 计算属性值
2. 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如：

```
ValueAnimator animation = ValueAnimator.ofFloat(0f, 1f);
animation.setDuration(1000);
animation.addUpdateListener(new AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator animation) {
        Log.i("update", ((Float) animation.getAnimatedValue()).toString());
    }
});
animation.setInterpolator(new CycleInterpolator(3));
animation.start();
```

此示例中只是向Logcat输出了一些信息，可以改为想做的工作。

Animator.AnimatorListener

```
onAnimationStart()

onAnimationEnd()

onAnimationRepeat()
```

```
//当动画被取消时调用，同时会调用onAnimationEnd().
onAnimationCancel()
```

ValueAnimator.AnimatorUpdateListener

```
onAnimationUpdate()    //通过监听这个事件在属性的值更新时执行相应的操作，对于ValueAnimator一般要监听此事件执行相应的动作，不然Animation没意义，在ObjectAnimator（继承自ValueAnimator）中会自动更新属性，如无必要不必监听。在函数中会传递一个ValueAnimator参数，通过此参数的getAnimatedValue()取得当前动画属性值。
```

可以继承AnimatorListenerAdapter而不是实现AnimatorListener接口来简化操作，这个类对AnimatorListener中的函数都定义了一个空函数体，这样我们就只用定义想监听的事件而不用实现每个函数却只定义一空函数体。

```
ObjectAnimator oa=ObjectAnimator.ofFloat(tv, "alpha", 0f, 1f);
oa.setDuration(3000);
oa.addListener(new AnimatorListenerAdapter(){
    public void on AnimationEnd(Animator animation){
        Log.i("Animation","end");
    }
});
oa.start();
```

3.3 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。

如果上述条件不满足，则不能用ObjectAnimator，应用ValueAnimator代替。

```
tv=(TextView)findViewById(R.id.textview1);
btn=(Button)findViewById(R.id.button1);
btn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        ObjectAnimator oa=ObjectAnimator.ofFloat(tv, "alpha", 0f, 1f);
        oa.setDuration(3000);
        oa.start();
    }
});
```

把一个TextView的透明度在3秒内从0变至1。

根据应用动画的对象或属性的不同，可能需要在onAnimationUpdate函数中调用invalidate()函数刷新视图。

3.4 通过AnimationSet应用多个动画

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：

1. 播放anim1；
2. 同时播放anim2,anim3,anim4；
3. 播放anim5。

```
AnimatorSet bouncer = new AnimatorSet();
bouncer.play(anim1).before(anim2);
bouncer.play(anim2).with(anim3);
bouncer.play(anim2).with(anim4)
bouncer.play(anim5).after(amin2);
animatorSet.start();
```

3.5 TypeEvalutors

根据属性的开始、结束值与TimeInterpolation计算出的因子计算出当前时间的属性值，android提供了以下几个evalutor：

- IntEvaluator：属性的值类型为int；
- FloatEvaluator：属性的值类型为float；
- 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TypeEvalutor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);
    }
}
```

根据动画执行的时间跟应用的Interplator，会计算出一个0~1之间的因子，即evalute函数中的fraction参数，通过上述FloatEvaluator应该很好看出其意思。

3.6 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
- CycleIinterpolator 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

3.7 当Layout改变时应用动画

ViewGroup中的子元素可以通过setVisibility使其Visible、Invisible或Gone，当有子元素可见性改变时(VISIBLE、GONE)，可以向其应用动画，通过LayoutTransition类应用此类动画：

```
transition.setAnimator(LayoutTransition.DISAPPEARING, customDisappearingAnim);
```

通过setAnimator应用动画，第一个参数表示应用的情境，可以以下4种类型：

- APPEARING 当一个元素在其父元素中变为Visible时对这个元素应用动画
- CHANGE_APPEARING 当一个元素在其父元素中变为Visible时，因系统要重新布局有一些元素需要移动，对这些要移动的元素应用动画
- DISAPPEARING 当一个元素在其父元素中变为GONE时对其应用动画
- CHANGE_DISAPPEARING 当一个元素在其父元素中变为GONE时，因系统要重新布局有一些元素需要移动，这些要移动的元素应用动画.

第二个参数为一Animator。

```
mTransitioner.setStagger(LayoutTransition.CHANGE_APPEARING, 30);
```

此函数设置动画延迟时间，参数分别为类型与时间。

3.8 Keyframes

keyFrame是一个 时间/值 对，通过它可以定义一个在特定时间的特定状态，即关键帧，而且在两个keyFrame之

间可以定义不同的Interpolator，就好像多个动画的拼接，第一个动画的结束点是第二个动画的开始点。KeyFrame是抽象类，要通过ofInt(),ofFloat(),ofObject()获得适当的KeyFrame，然后通过PropertyValuesHolder.ofKeyframe获得PropertyValuesHolder对象，如以下例子：

```
Keyframe kf0 = Keyframe.ofInt(0, 400);
Keyframe kf1 = Keyframe.ofInt(0.25f, 200);
Keyframe kf2 = Keyframe.ofInt(0.5f, 400);
Keyframe kf4 = Keyframe.ofInt(0.75f, 100);
Keyframe kf3 = Keyframe.ofInt(1f, 500);
PropertyValuesHolder pvhRotation = PropertyValuesHolder.ofKeyframe("width", kf0, kf1, kf2, kf4, kf3);
ObjectAnimator rotationAnim = ObjectAnimator.ofPropertyValuesHolder(btn2, pvhRotation);
rotationAnim.setDuration(2000);
```

上述代码的意思为：设置btn对象的width属性值使其：

- 开始时 Width=400
- 动画开始1/4时 Width=200
- 动画开始1/2时 Width=400
- 动画开始3/4时 Width=100
- 动画结束时 Width=500

第一个参数为时间百分比，第二个参数是在第一个参数的时间时的属性值。
定义了一些Keyframe后，通过PropertyValuesHolder类的方法ofKeyframe一个PropertyValuesHolder对象，然后通过ObjectAnimator.ofPropertyValuesHolder获得一个Animator对象。
用下面的代码可以实现同样的效果（上述代码时间值是线性，变化均匀）：

```
ObjectAnimator oa=ObjectAnimator.ofInt(btn2, "width", 400,200,400,100,500);
oa.setDuration(2000);
oa.start();
```

3.9 Animating Views

在View Animation中，对View应用Animation并没有改变View的属性，动画的实现是通过其Parent View实现的，在View被drawn时Parents View改变它的绘制参数，draw后再改变参数invalidate，这样虽然View的大小或旋转角度等改变了，但View的实际属性没变，所以有效区域还是应用动画之前的区域，比如你把一按钮放大两倍，但还是放大这前的区域可以触发点击事件。为了改变这一点，在Android 3.0中给View增加了一些参数并对这些参数增加了相应的getter/setter函数（ObjectAnimator要用这些函数改变这些属性）：

- translationX,translationY: View相对于原始位置的偏移量
- rotation,rotationX,rotationY: 旋转，rotation用于2D旋转角度，3D中用到后两个
- scaleX,scaleY: 缩放比
- x,y: View的最终坐标，是View的left，top位置加上translationX，translationY
- alpha: 透明度

跟位置有关的参数有3个，以X坐标为例，可以通过getLeft(),getX(),getTranslateX()获得，若有一Button btn2，布局时其坐标为（40,0）：

```
//应用动画之前
btn2.getLeft();    //40
btn2.getX();    //40
btn2.getTranslationX();    //0
//应用translationX动画
ObjectAnimator oa=ObjectAnimator.ofFloat(btn2,"translationX", 200);
oa.setDuration(2000);
oa.start();
/*应用translationX动画后
btn2.getLeft();    //40
btn2.getX();    //240
btn2.getTranslationX();    //200
*/
//应用X动画，假设没有应用之前的translationX动画
ObjectAnimator oa=ObjectAnimator.ofFloat(btn2, "x", 200);
oa.setDuration(2000);
oa.start();
/*应用x动画后
btn2.getLeft();    //40
btn2.getX();    //200
btn2.getTranslationX();    //160
*/
```

无论怎样应用动画，原来的布局时的位置通过getLeft()获得，保持不变；
X是View最终的位置；
translationX为最终位置与布局时初始位置这差。
所以若就用translationX即为在原来基础上移动多少，X为最终多少
getX()的值为getLeft()与getTranslationX()的和
对于X动画，源代码是这样的：

```
case X:
       info.mTranslationX = value - mView.mLeft;
       break;
```

Property Animation也可以在XML中定义

- <set> - AnimatorSet
- <animator> - ValueAnimator
- <objectAnimator> - ObjectAnimator

XML文件应放大/res/animator/中，通过以下方式应用动画：

```
AnimatorSet set = (AnimatorSet) AnimatorInflater.loadAnimator(myContext, R.anim.property_animator);
set.setTarget(myObject);
set.start();
```

3.10 ViewPropertyAnimator

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。

以下两段代码实现同样的效果：

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

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


分类：android

标签：android , 动画 , android动画 , animation , Android animation , 属性动画

« 上一篇：Android 2.0以后的Contacts API--ContactsContract
» 下一篇：Android Touch事件

posted @ 2011-12-02 17:16 AngelDevil 阅读(123952) 评论(17) 编辑 收藏

评论列表

#1楼 2011-12-05 11:16 qianqianlianmeng
写的很不错哈！！！不知道如何才能联系上 作者呢？？
支持(0) 反对(0)

#2楼 [楼主] 2011-12-05 11:52 AngelDevil
@qianqianlianmeng
都是SDK上的嘛，简易翻译吧
支持(1) 反对(1)

#3楼 2012-01-18 09:25 vanezkw