Android 动画和图形

让你的应用视觉上和表现上达到最优通过使用Android强大的图形特性例如OpenGL，硬件加速和内置UI动画。

概览

Android提供一系列强大的API来将动画应用到 UI 组件上以及绘制2D和3D图形。下面的部分提供了API和系统可用功能的概览来帮助你决定最满足你需求的方法。

动画

Android 框架提供了两种动画系统：属性动画和视图动画。两种动画都可用，但是通常使用属性动画，因为其使用更灵活而且提供更多特性。除了这两种动画，你还可以使用 Drawable 动画，Drawable 动画允许你载入 Drawable 资源来一帧一帧的播放。

- 属性动画
  Android 3.0（API level 11）引入，属性动画系统允许你为任何对象的属性设置动画，包括没有渲染到屏幕上的对象。这个系统可以扩展而且可以为自定义类型的属性添加动画。
- 视图动画
  视图动画是较老的系统并且只能在视图（View）上使用。它使用起来相对容易些而且提供足够的功能来满足很多应用的需求。
- Drawable动画
  Drawable动画就是一帧一帧的播放Drawable资源，就像放映电影一样。当用Drawable资源更容易animate一些东西时这种动画很有用，比如一系列位图。

属性动画

属性动画系统一个健壮的框架允许你animate几乎所有事情。你可以定义一个动画来改变任何对象属性在一段时间里，无论这个对象是否绘制出来。一个属性动画改变属性值在指定时间内。要animate某东西，你需要指定你想animate的对象属性，比如一个对象在屏幕上的位置，以及动画时长和值的区间。

属性动画系统让你定义如下的动画特征：

- Duration：你可以定义动画时长，默认300ms。
- Time Interpolation:定义属性值随时间是如何计算的。
- Repeat count and behavior：你可以定义是否重复播放动画或者播放动画的次数。你也可以定义是否让动画反着播放回去。设置为 reverse 可以反向播放动画。
- Animator sets：你可以将动画设置成一组来一起播放或者依次序播放或者延迟播放。
- Frame Refresh Delay：你可以设置更新动画帧的频率，默认每10ms更新一次，但是你的程序更新帧的频率最终取决于系统是否繁忙以及系统可以多快的服务潜在的计时器。

属性动画是怎么工作的：

首先，举个简单例子说明属性动画怎么工作。图 1 描述了一个假设的物体动画它的x属性，x表示它在屏幕上的水平位置。动画时长设置为40ms，移动距离40个像素。每隔 10 ms，默认的帧刷新率，物体水平移动 10 个像素。40 ms 之后，动画结束，物体停在40 像素的位置。这个例子中动画使用线性插值，物体匀速移动。

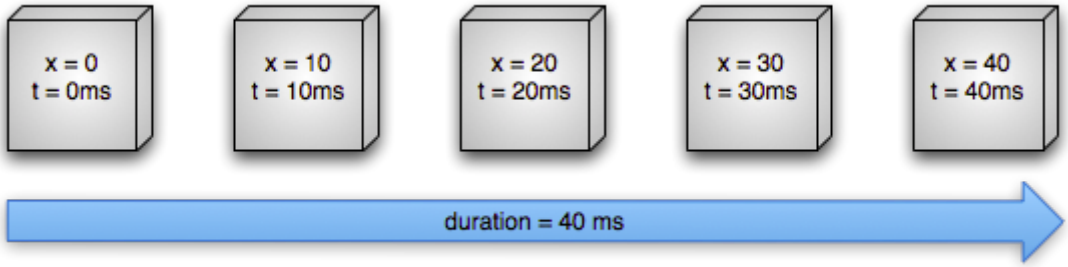

图 1 线性插值动画

你也可以定义非线性的插值动画。图 2 举例了一个假设的物体在动画开始时加速，动画要结束时减速。这个物体依然在40 ms 移动40个像素，但不是线性的。

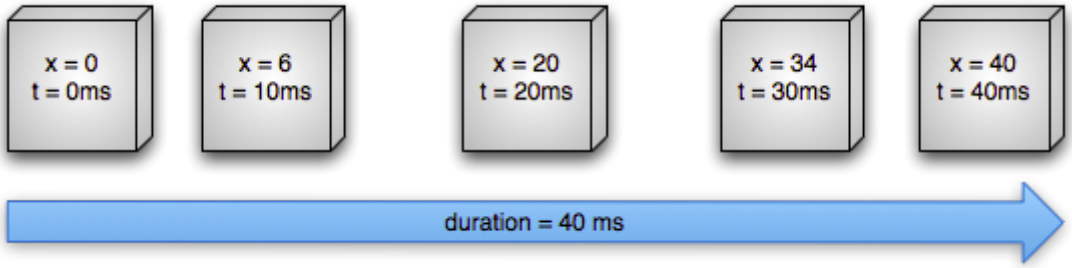

图 2 先加速后减速

下面仔细介绍属性动画中的组件是怎样计算上面例子中的动画的，图3描述了主要的类是如何配合工作的。

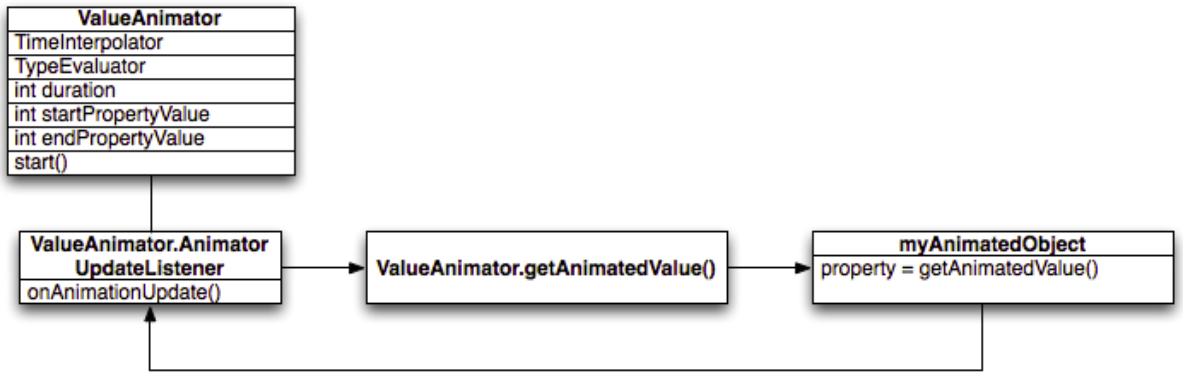

图3 动画计算过程

`ValueAnimator`对象保持追踪你的动画的时间，例如动画运行了多久，被设置动画的属性当前的值。

`ValueAnimator`封装一个`TimeInterpolator`，用来定义动画插值，还有一个`TypeEvaluator`，定义了怎样计算被设置动画的属性值。例如，图2中的`TimeInterpolator`使用`AccelerateDecelerateInterpolator`，`TypeEvaluator`使用`IntEvaluator`。

要开始一个动画，创建一个`ValueAnimator`对象并给要设置动画的属性制定起始值和最终值，还有动画时长。当调用`start ()`方法动画开始执行。在动画执行期间，`ValueAnimator`根据已经经过的时间计算一个介于0到1的经过的时间小数，代表动画完成时间的百分比。例如，图1中，在10ms时，经过的时间小数可以使0.25因为整个时间是40ms。

当`ValueAnimator`结束计算经过的时间小数，它调用`TimeInterpolator`，计算插值小数。例如，图2中，因为动画是慢慢加速的，在10ms时，插值小数大概是0.15，小于经过的时间小数0.25。图1中，插值小数和经过的时间小数一样。

当插值小数计算完了，`ValueAnimator`调用合适的`TypeEvaluator`来计算设置动画的属性值。例如，图2中，10ms时插值小数是0.15，所以这时的属性值是0.15×（40-0）或者 6.

属性动画和视图动画的区别

视图动画系统只能给视图（view）对象设置动画，如果你想给非视图对象设置动画，你需要自己实现代码。另外，视图动画只能给视图对象的一部分属性设置动画，比如缩放旋转，但不能设置比如背景，颜色。

另外，视图动画只改变view被绘制的地方，并不是实际的view。例如，给一个按钮设置动画在屏幕上移动，在移动后的按钮上并不能点击，只能点击原来的位置。

API预览

你可以在`android.animation`里找到大部分属性动画系统的API。由于视图动画系统在`android.view.animation`中已经预设了很多插值器，你可以在属性动画中使用这些插值器。下面的表格描述了属性动画系统的主要部分。

`Animator`类提供创建动画最基本的结构。通常不直接使用这个类因为它只提供最少的功能而需要被继承来实现所有的功能。下面的类继承`Animator`。

Class	Description
`ValueAnimator`	The main timing engine for property animation that also computes the values for the property to be animated. It has all of the core functionality that calculates animation values and contains the timing details of each animation, information about whether an animation repeats, listeners that receive update events, and the ability to set custom types to evaluate. There are two pieces to animating properties: calculating the animated values and setting those values on the object and property that is being animated. `ValueAnimator` does not carry out the second piece, so you must listen for updates to values calculated by the `ValueAnimator` and modify the objects that you want to animate with your own logic. See the section about Animating with ValueAnimator for more information.
`ObjectAnimator`	A subclass of `ValueAnimator` that allows you to set a target object and object property to animate. This class updates the property accordingly when it computes a new value for the animation. You want to use `ObjectAnimator` most of the time, because it makes the process of animating values on target objects much easier. However, you sometimes want to use `ValueAnimator` directly because `ObjectAnimator` has a few more restrictions, such as requiring specific acessor methods to be present on the target object.
`AnimatorSet`	Provides a mechanism to group animations together so that they run in relation to one another. You can set animations to play together, sequentially, or after a specified delay. See the section about Choreographing multiple animations with Animator Sets for more information.

表1 Animators

- `ValueAnimator`:属性动画主要的计时器，同时计算设置动画的属性值。它具有计算动画值的所有核心功能而且包含每个动画的时间细节，动画是否重复的信息，接受更新事件的监听器，以及设置自定义的`TypeEvaluator`。给属性设置动画有两部分：计算动画值以及将计算得到的值赋给对象属性。`ValueAnimator`并不执行第二部分，所以你需要监听更新事件来将计算得到的值赋给对象。参看`使用ValueAnimator设置动画`来获取更多信息。

- `ObjectAnimator`：`ValueAnimator`的一个子类，允许你设置目标对象和对象属性。这个类根据动画计算出的值更新属性。通常情况下使用`ObjectAnimator`即可，因为它使给目标对象赋值的过程更简单。但是，有时候你可能需要直接使用`ValueAnimator`，因为`ObjectAnimator`有一些限制，比如目标类中需要有`get`方法。
- `AnimatorSet`：一种设置一组动画的机制，以便于可以依照某种关系运行动画。你可以设置一起播放动画，或者按照顺序，或者一段延迟之后播放。参照`使用AnimatorSet编排多个动画`。

`Evaluator`告诉属性动画系统怎样计算给定的属性的值。通过`Animator`类提供的时间数据，起始值，来计算设置动画的属性值。属性动画系统提供如下的`Evaluator`。

Class/Interface	Description
`IntEvaluator`	The default evaluator to calculate values for `int` properties.
`FloatEvaluator`	The default evaluator to calculate values for `float` properties.
`ArgbEvaluator`	The default evaluator to calculate values for color properties that are represented as hexidecimal values.
`TypeEvaluator`	An interface that allows you to create your own evaluator. If you are animating an object property that is *not* an `int`, `float`, or color, you must implement the `TypeEvaluator` interface to specify how to compute the object property's animated values. You can also specify a custom `TypeEvaluator` for `int`, `float`, and color values as well, if you want to process those types differently than the default behavior. See the section about Using a TypeEvaluator for more information on how to write a custom evaluator.

表2 Evaluators

- `IntEvalator`：默认的计算整型属性的Evaluator。
- `FloatEvaluator`：默认的计算浮点型属性的Evaluator。
- `ArgbEvaluator`：默认的计算十六进制颜色属性的Evaluator。
- `TypeEvaluator`：一个允许你创建自己Evaluator的接口。参看`使用TypeEvaluator`获取更多信息。

时间插值器定义了属性值是如何根据时间计算的。例如，你可以指定动画线性变化，也就是动画在整个时间里均匀变化，或者你可以指定动画使用非线性时间。表3描述了在android.view.animation中的插值器。如果没有适合你的插值器，实现TimeInterpolator接口创建自定义的插值器。

Class/Interface	Description
`AccelerateDecelerateInterpolator`	An interpolator whose rate of change starts and ends slowly but accelerates through the middle.
`AccelerateInterpolator`	An interpolator whose rate of change starts out slowly and then accelerates.
`AnticipateInterpolator`	An interpolator whose change starts backward then flings forward.
`AnticipateOvershootInterpolator`	An interpolator whose change starts backward, flings forward and overshoots the target value, then finally goes back to the final value.
`BounceInterpolator`	An interpolator whose change bounces at the end.
`CycleInterpolator`	An interpolator whose animation repeats for a specified number of cycles.
`DecelerateInterpolator`	An interpolator whose rate of change starts out quickly and and then decelerates.
`LinearInterpolator`	An interpolator whose rate of change is constant.
`OvershootInterpolator`	An interpolator whose change flings forward and overshoots the last value then comes back.
`TimeInterpolator`	An interface that allows you to implement your own interpolator.

表3 插值器

使用**ValueAnimator**设置动画

ValueAnimator类为某些类型的的值设置动画通过指定一个整型、浮点型、颜色集合。获取一个ValueAnimator通过工厂方法ofInt(), ofFloat(), 或者 ofObject(). 例如：

```
ValueAnimator animation = ValueAnimator.ofFloat( 0f, 100f);
animation.setDuration( 1000);
animation.start();
```

在这段代码中，ValueAnimator在1000ms内开始计算动画的值，介于0到100之间。

你也可以指定自定义的类型来计算动画：

```
ValueAnimator animation = ValueAnimator.ofObject( new MyTypeEvaluator(),
 startPropertyValue, endPropertyValue);
animation.setDuration( 1000);
animation.start();
```

你可以通过添加`AnimatorUpdateListener`使用动画的值，如下

```
animation.addUpdateListener( new ValueAnimator.AnimatorUpdateListener() {
    @Override
    public void onAnimationUpdate(ValueAnimator updatedAnimation)  {
        // You can use the animated value in a property that uses the
        // same type as the animation. In this case, you can use the
        // float value in the translationX property.
        float animatedValue = (float)updatedAnimation.getAnimatedValue
();
        textView.setTranslationX(animatedValue);
    }
});
```

在`onAnimationUpdate`方法中，你可以获取更新的动画的值并将它使用到你的view的属性中。参看`Animation Listeners`获取更多监听器。

使用ObjectAnimator

该类是`ValueAnimator`的子类。不需要实现`ValueAnimator.AnimatorUpdateListener`接口，因为动画计算过的值自动更新。

示例与ValueAnimator类似，不同的是你指定对象和对象的属性（字符串形式）以及起始值。

```
ObjectAnimator animation = ObjectAnimator.ofFloat(textView,  "translation
X", 100f);
animation.setDuration( 1000);
animation.start();
```

要使ObjectAnimator正确的更新属性，你需要做如下的事情：

- 你要设置动画的属性必须有setter方法。如果setter方法不存在，你有三个选项：
    - 添加setter方法

    - 使用包装类包含setter方法
    - 使用`ValueAnimator`

- 如果你在`ObjectAnimator`的工厂方法中为`values...`形参只设置一个参数，默认是结束值。因此，你要设置动画的对象属性需要有`getter`方法来获取初始值。
- 属性的`getter`和`setter`方法使用的参数类型必须和你在`ObjectAnimator`中指定的起始值类型相同。比如你必须要有`targetObject.setPropName(float)`和`targetObject.getPropName(float)`如果你构造如下的`ObjectAnimator`：

```
ObjectAnimator.ofFloat(targetObject, "propName", 1f)
```

- 根据你要设置对象的属性，你可能需要调用`invalidate()`方法来使用更新的值让屏幕重绘。在`onAnimationUpdate()`回调方法中实现上面的过程。例如，动画一个`Drawable`对象的颜色属性只有在该对象重绘自己时产生更新。`View`中的所有属性的`setter`方法，例如`setAlpha()`，`setTranslationX()`，会自动调用重绘，所以在使用新值调用这些方法时不用调用`invalidate()`。更多信息参考`Animation Listeners`。

使用AnimatorSet编排多个动画

很多情况下，你想播放一个动画取决于另一个动画开始或者结束。`Android`系统允许你将多个动画放入一个`AnimatorSet`中，以便于你决定动画播放的方式。`AnimatorSet`中也可以设置`AnimatorSet`。

下面的示例代码来自`Bouncing Balls`例子播放如下的`Animator`对象以如下的方法：

1. 播放`bounceAnim`
2. 同时播放`squashAnim1, squashAnim2, stretchAnim1, and stretchAnim2`
3. 播放`bounceBackAnim`
4. 播放`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();
```

动画监听器

你可以在动画播放期间监听重要时间通过使用下面的监听器：

- `Animator.AnimatorListener`
    - `onAnimationStart()` 动画开始时调用
    - `onAnimationEnd()` 动画结束时调用
    - `onAnimationRepeat()` 动画重复时调用
    - `onAnimationCancel()` 动画取消时调用，同时调用`onAnimationEnd ()`
- `ValueAnimator.AnimatorUpdateListener`
    - `onAnimationUpdate()` 播放每一帧动画时调用，监听此事件来使用`ValueAnimator`计算的值。要得到计算过的值，使用传入此事件中的ValueAnimator对象，调用其`getAnimatedValue()` 方法。如果你使用`ValueAnimator`，需要实现这个监听器。
      取决于你设置动画的哪种属性或对象，你可能对一个view组件需要调用`invalidate()`来重绘屏幕。

你可以继承`AnimatorListenerAdapter` 类来代替实现`Animator.AnimatorListener` 接口。可是选择要实现的方法。

例如，`Bouncing Balls`示例代码中，创建一个`AnimatorListenerAdapter` 只使用`onAnimationEnd()` 回调方法。

```
ValueAnimator fadeAnim = ObjectAnimator.ofFloat(newBall,  "alpha", 1f,
0f);
fadeAnim.setDuration(250);
fadeAnim.addListener(new AnimatorListenerAdapter() {
public void onAnimationEnd(Animator animation) {
    balls.remove(((ObjectAnimator)animation).getTarget());
}
```

给ViewGroup布局变化设置动画

属性动画系统提供给`ViewGroup`对象变化设置动画的功能，同时提供一种简单的方法来给`View`对象设置动画。

你可以使用`LayoutTransition`类来给`ViewGroup`中的布局变化设置动画。当你向`ViewGroup`中添加或者删除`view`，或者当你对`ViewGroup`中`view`调用`setVisibility()` 方法时，可以设置出现和消失动画。`ViewGroup`中其他的`view`在移动到新的位置时也可以设置动画当你添加或者删除其它`view`时。你可以通过调用`setAnimator()` 方法给一个`LayoutTransition` 对象定义如下的动画或者传入一个`Animator`对象使用下面的`LayoutTransition`常量。

- `APPEARING` 当一个新的`view`出现时的动画的标记。
- `CHANGE_APPEARING` 当一个新的`view`出现时其它`view`的动画的指示。
- `DISAPPEARING` `view`消失的动画的标记
- `CHANGE_DISAPPEARING` `view`消失时其它`view`的动画的标记

你可以定义你自己的动画来实现上面的四种效果，或者就使用默认的。

`LayoutAnimations` 示例代码演示了如何为布局变化设置动画以及为你想要的`view`设置动画。

`LayoutAnimationsByDefault` 和对应的 `layout_animations_by_default.xml` 布局文件演示了怎样为 `ViewGroup` 在XML中设置默认的布局动画。你只需要将 `ViewGroup的android:animateLayoutchanges` 属性设置为 `true` 即可。例如：

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:id="@+id/verticalContainer"
    android:animateLayoutChanges="true" />
```

使用TypeEvaluator

如果你想给一个对于 `Android` 系统位置的类型设置动画，你可以通过实现 `TypeEvaluator` 来创建你自己的 `Evaluator` 。

`TypeEvaluator` 只需要实现 `evaluate()` 方法，这个方法让你使用的 `Animator` 返回一个合适的值。 `FloatEvaluator` 的示例如下：

```
public class FloatEvaluator implements TypeEvaluator {

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

其中的 `fraction` 是经过你指定的 `interpolator` 插值计算的值。

使用Interpolator

`interpolator` 定义了动画中值是怎样根据时间计算的，例如，你可以定义线性的 `interpolator` 或者非线性的 `interpolator` 。

`interpolator` 从 `Animator` 中接收一个时间因数，然后把它修改为符合要求的因数。 `Android` 系统提供了一些常用的 `interpolator` 在 `android.view.animation` 包中，或者你可以实现 `TimeInterpolator` 来实现你自己的插值器。

下面是 AccelerateDecelerateInterpolator和 LinearInterpolator 的实现。

```
public float getInterpolation(float input) {
    return (float)(Math.cos((input + 1) * Math.PI) / 2.0f) + 0.5f;
}
```

```
public float getInterpolation(float input) {
    return input;
}
```

指定关键帧

一个`Keyframe`对象包含一个time/value键值对来让你定义特定时间下的动画的特定状态。每隔Keyframe可以有自己的`interpolator`来控制上一个`Keyframe`和这个`Keyframe`之间的行为。

要实例化一个`Keyframe`对象,你需要使用`ofInt(), ofFloat(), or ofObject()`这些工厂方法中的一种来获取适当类型的`Keyframe`。然后调用`ofKeyframe ()`方法来获取一个`PropertyValuesHolder`对象。然后传入`PropertyValuesHolder`对象来获取`Animator`对象。参看下面的示例。

```
Keyframe kf0 = Keyframe.ofFloat( 0f, 0f);
Keyframe kf1 = Keyframe.ofFloat(. 5f, 360f);
Keyframe kf2 = Keyframe.ofFloat( 1f, 0f);
PropertyValuesHolder pvhRotation = PropertyValuesHolder.ofKeyframe( "rota
tion", kf0, kf1, kf2);
ObjectAnimator rotationAnim = ObjectAnimator.ofPropertyValuesHolder(targ
et, pvhRotation)
rotationAnim.setDuration( 5000ms);
```

参看`MultiPropertyAnimation`示例代码获取`KeyFrame`的更多用法。

使用ViewPropertyAnimator添加动画

ViewPropertyAnimator提供了一种简单的并行的给view属性们添加动画的方法。跟ObjectAnimator类似，不同的就是可以同时改变属性，并且代码更简明。如下
Multiple ObjectAnimator objects

```
ObjectAnimator animX = ObjectAnimator.ofFloat(myView,  "x", 50f);
ObjectAnimator animY = ObjectAnimator.ofFloat(myView,  "y", 100f);
AnimatorSet animSetXY =  new AnimatorSet();
animSetXY.playTogether(animX, animY);
animSetXY.start();
```

One ObjectAnimator

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

ViewPropertyAnimator

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

更过信息参看博客blog post

在xml中声明动画

为了区分使用属性动画API的动画和使用视图动画框架的动画，将属性动画的XML放到`res/animator`中。

动画类和队形的XML标签：

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

参看Animation Resources寻找你可以使用的属性。下面的示例演示了先后播放两组属性动画，其中第一组属性动画同时播放。

```
<set android:ordering="sequentially">
    <set>
        <objectAnimator
            android:propertyName="x"
            android:duration="500"
            android:valueTo="400"
            android:valueType="intType"/>
        <objectAnimator
            android:propertyName="y"
            android:duration="500"
            android:valueTo="300"
            android:valueType="intType"/>
    </set>
    <objectAnimator
        android:propertyName="alpha"
        android:duration="500"
        android:valueTo="1f"/>
</set>
```

如下代码演示怎样在代码中使用XML文件。

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

你也可以在`XML`中定义`ValueAnimator`：

```
<animator xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:valueType="floatType"
    android:valueFrom="0f"
    android:valueTo="-100f" />
```

下面代码演示在代码中使用上述XML中的`ValueAnimator`：

```
ValueAnimator xmlAnimator = (ValueAnimator) AnimatorInflater.loadAnimato
r(this,
        R.animator.animator);
xmlAnimator.addUpdateListener( new ValueAnimator.AnimatorUpdateListener()
 {
    @Override
    public void onAnimationUpdate(ValueAnimator updatedAnimation)  {
        float animatedValue = (float)updatedAnimation.getAnimatedValue
();
        textView.setTranslationX(animatedValue);
    }
});

xmlAnimator.start();
```