Android Animation Basics

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Animation

Android offers three kinds of animation:

- **Tweened Animations (View Animation):** are applied to Views and transform them in size, position or opacity. For example you can fade in a view or rotate it
- **Frame-by-frame Animations (Drawable Animation):** shows different drawables in a View. Both animations are limited to the original size of the view
- **Property Animations :** The property animation system enables you to animate almost anything within your application. It's a framework designed to affect any object property over a period of time using the specified interpolation technique

Tweened Animations

Tweened animations are commonly used to:

- Transition between Activities
- Transition between layouts within an Activity
- Transition between different content displayed within the same View
- Provide user feedback, such as indicating progress or “shaking” an input box to indicate an incorrect or invalid data entry

Tweened Animations

Tweened View Animations:-

- Alpha Animation — Lets you animate a change in the View's transparency (opacity or alpha blending)
- Rotate Animation — Lets you spin the selected View canvas in the XY plane
- Scale Animation — Lets you to zoom in to or out from the selected View
- Translate Animation — Lets you move the selected View around the screen (although it will only be drawn within its original bounds)
- The animation XML file belongs in the res/anim/ directory of your Android project
- The file must have a single root element: this will be either a single <alpha>, <scale>, <translate>, <rotate>, interpolator element, or <set> element that holds groups of these elements (which may include another <set>)`.

Tweened Animations

Animations can be defined

Via XML Resource

- **xml resource files in the directory "res/anim"**
- Load the resource file via AnimationUtils.loadAnimation(this,R.anim.Animation).

Via Code

- Using AnimationSet (android.view.animation.AnimationSet)

Alpha Animation

- Alpha animations allows us to create fade-in/fade-out effects
- All animations are stored in *res/anim* so we create a new xml file named *alpha.xml* there

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
   <alpha
      android:fromAlpha="0"
      android:toAlpha="1"
      android:duration="2000" >
   </alpha>
</set>
```

Alpha Animation

- **fromAlphaStarting -**

  alpha value for the animation, where 1.0 means fully opaque and 0.0 means fully transparent.

- **toAlpha-**

  Ending alpha value for the animation.

Rotate Animation

- An animation that controls the rotation of an object. This rotation takes place int the X-Y plane. You can specify the point to use for the center of the rotation, where (0,0) is the top left point. If not specified, (0,0) is the default rotation point.

```
<set
xmlns:android="http://schemas.android.com/apk/res/android">
   <rotate
      android:fromDegrees="0"
      android:toDegrees="360"
      android:pivotX="50%"
      android:pivotY="50%"
      android:duration="2000" >
   </rotate>
</set>
```

Rotate Animation

- **fromDegreesRotation**

  offset to apply at the start of the animation.
- **toDegreesRotation**

  offset to apply at the end of the animation.
- **pivotX**
- The X coordinate of the point about which the object is being rotated, specified as an absolute number where 0 is the left edge.
- **pivotY**

  The Y coordinate of the point about which the object is being rotated, specified as an absolute number where 0 is the top edge.

Translate Animation

- TranslateAnimation Lets you move the selected View around the screen (although it will only be drawn within its original bounds).

```
<set
xmlns:android="http://schemas.android.com/apk/res/android">
   <translate
      android:fromXDelta="200%"
      android:toXDelta="0%"
      android:fromYDelta="200%"
      android:toYDelta="0%"
      android:duration="2000" >
   </translate>
</set>
```

Translate Animation

- **fromXDelta**

  Change in X coordinate to apply at the start of the animation
- **toXDelta**

  Change in X coordinate to apply at the end of the animation
- **fromYDelta**

  Change in Y coordinate to apply at the start of the animation to
- **Ydelta**

  Change in Y coordinate to apply at the end of the animation

Scale Animation

- ScaleAnimation Allows you to zoom in to or out from the selected View.

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
   <scale
      android:fromXScale="4"
      android:toXScale="1"
      android:fromYScale="3"
      android:toYScale="1"
      android:duration="2000" >
   </scale>
</set>
```

Animation Utils

- AnimationUtils is the utility class for interface between xml and Animation Class
- Xml files are converted in Animation object using loadAnimation static method of this class

Animation anim01 = AnimationUtils.*loadAnimation*(getApplicationContext(), R.anim.***alfa***);

- Now Animation object may be applied on any UI component using startAnimation. Eg applying animation on a TextView (tv):

tv.startAnimation(anim01);

Animation Demo

- Demo of Scale, Rotate, Alpha & Translate

Animation Set

- Android offers the AnimationSet class, to group and configure animations to be run as a set
- You can define the start time and duration of each animation used within a set to control the timing and order of the animation sequence

Animation Set

```
AnimationSet animationSet = new AnimationSet(true);

    RotateAnimation rotateAnimation = new RotateAnimation(0,360,Animation.RELATIVE_TO_SELF, 0.5f,
            Animation.RELATIVE_TO_SELF, 0.5f);
            rotateAnimation.setDuration(500);
           rotateAnimation.setRepeatCount(2);
           animationSet.addAnimation(rotateAnimation);

     TranslateAnimation translateAnimation = new TranslateAnimation(0, 400, 0, 0);
           translateAnimation.setStartOffset(500);
           translateAnimation.setDuration(1000);
            animationSet.addAnimation(translateAnimation);

     AlphaAnimation alphaAnimation = new AlphaAnimation(1, 0);
           alphaAnimation.setStartOffset(500);
           alphaAnimation.setDuration(1000);
            animationSet.addAnimation(alphaAnimation);

final Button  button2 = (Button) findViewById(R.id.button2);
button2.startAnimation(animationSet);
```

Using Interpolator

- Animation will have the same speed in getting applied, However this can be changed by applying an "Interpolator" class
- Android includes several Interpolator subclasses that specify various speed curves
- AccelerateInterpolator: rate of change starts slowly and then accelerates
- DecelerateInterpolator: rate of change starts out quickly and and then decelerates
- BounceInterpolator: change bounces at the end
- CycleInterpolator : Repeats the animation for a specified number of cycles
- LinearInterpolator: The rate of change is constant
- OvershootInterpolator : change flings forward and overshoots the last value then comes back
- AnticipateOvershootInterpolator: change starts backward then flings forward and overshoots the target value and finally goes back to the final value
- AccelerateDecelerateInterpolator : Rate of change starts and ends slowly but accelerates through the middle

Using Interpolator

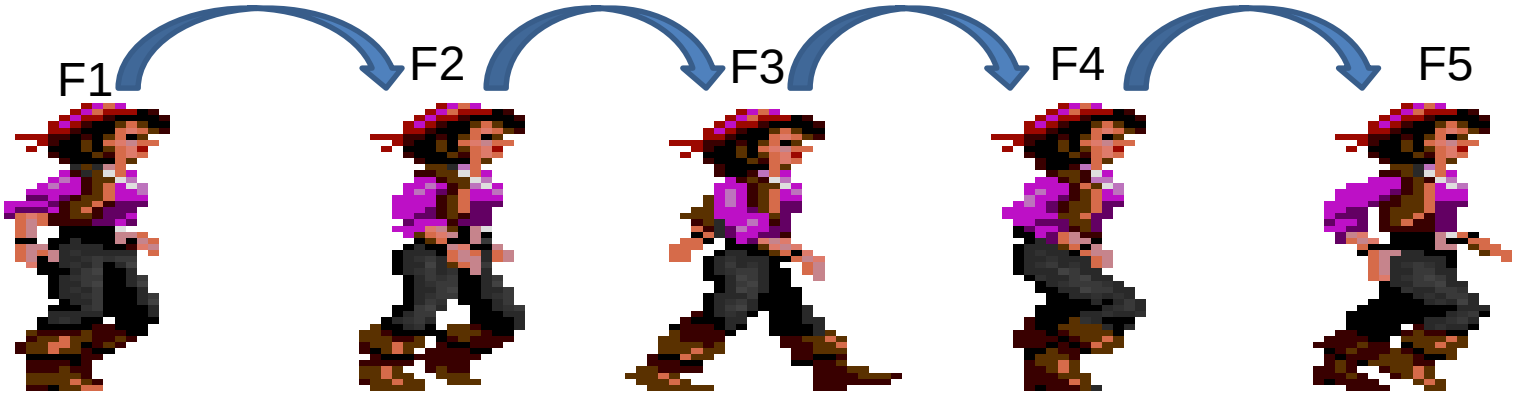
An interpolator is applied to an animation element with the **android:interpolator** attribute, the value of which is a reference to an interpolator resource.

Interpolator class	Resource ID
AccelerateDecelerateInterpolator	@android:anim/accelerate_decelerate_interpolator
AccelerateInterpolator	@android:anim/accelerate_interpolator
AnticipateInterpolator	@android:anim/anticipate_interpolator
AnticipateOvershootInterpolator	@android:anim/anticipate_overshoot_interpolator
BounceInterpolator	@android:anim/bounce_interpolator
CycleInterpolator	@android:anim/cycle_interpolator
DecelerateInterpolator	@android:anim/decelerate_interpolator
LinearInterpolator	@android:anim/linear_interpolator
OvershootInterpolator	@android:anim/overshoot_interpolator

Frame-by-Frame (Drawable) Animations

frame-by-frame animation

- Frame-by-frame animations are akin to traditional cell-based cartoons in which an image is chosen for each frame



- The XML file for this kind of animation belongs in the **res/drawable/** directory of your Android project.
- The XML file consists of an **<animation-list>** element as the root node and a series of child <item> nodes that each define a frame: a drawable resource for the frame and the frame duration
- The AnimationDrawable class is the basis for Drawable animations

XML Drawable for Animation

```
<animation-list
        xmlns:android=”http://schemas.android.com/apk/res/android”
        android:oneshot=”false”>
        <item android:drawable=”@drawable/rocket1” android:duration=”500” />
        <item android:drawable=”@drawable/rocket2” android:duration=”500” />
        <item android:drawable=”@drawable/rocket3” android:duration=”500” />
</animation-list>
```

By setting the **android:oneshot** attribute of the list to *true*, it will cycle just once then stop and hold on the last frame. If it is set *false* then the animation will loop for ever

Frame-by-Frame animation

- To display your animation, set it as the background to a View using the setBackgroundResource method or setBackgroundDrawable
- Get the reference to Animation Drawable Object
- Run the animation calling its start method
- It's important to note that the start() method called on the AnimationDrawable cannot be called during the onCreate() method of your Activity, because the AnimationDrawable is not yet fully attached to the window

```
ImageView  image = (ImageView)findViewById(R.id.my_animation_frame);
image.setBackgroundResource(R.drawable.animated_rocket);
AnimationDrawable animation =  (AnimationDrawable)image.getBackground();
animation.start();
```

Property Animations

Property Animations

- Android 3.0 (API level 11) introduced a new animation technique that animates object properties.
- A property animation changes a property's (a field in an object) value over a specified length of time.

- Any property of any object may be modified - visual or otherwise - using a property animator to transition it from one value to another, over a given period of time, using the interpolation algorithm of your choice, and setting the repeat behavior as required.

- you can use property animators to create a smooth transition for anything within your code; the target property doesn't even need to represent something visual.

- Property animations are effectively iterators implemented using a background timer to increment or decrement a value according to a given interpolation path over a given period of time.

Property Animation : API overview

- The Animator class provides the basic structure for creating animations. The following subclasses extend Animator:
- ValueAnimator: The main timing engine for property animation that also computes the values for the property to be animated.
- ObjectAnimator: A subclass of ValueAnimator that allows you to set a target object and object property to animate.
- AnimatorSet: Provides a mechanism to group animations together so that they run in relation to one another

Creating Property Animations

- The simplest technique for creating property animations is using an ObjectAnimator
- The Object Animator class includes the ofFloat, ofInt, and ofObject static methods to create an animation that transitions the specified property of the target object between the values provided

```
ObjectAnimator anim = ObjectAnimator.ofFloat(targetObject, propertyName, from, to);
```

- Alternatively, you can provide a single value to animate the property from its current value to its final value:

```
ObjectAnimator anim = ObjectAnimator.ofFloat(targetObject, propertyName, to);
```

- To use ofObject method, an implementation of the TypeEvalutor class must be provided.
- The Animator.AnimationListener class lets you create event handlers that are fired when an animation begins, ends, repeats, or is canceled.
- To apply an Animation Listener to your property animation, use the addListener method:

Property Animation : In XML

- The XML files for property animations are stored in the res/animator/ directory (instead of res/anim/)
- Following XML tags are supported
  - ValueAnimator - <animator>
  - ObjectAnimator - <objectAnimator>
  - AnimatorSet - <set>
- The **AnimatorInflator.loadAnimator** method is used to get Object Animator from XML animator resource
- Use the **setTarget** method to apply it to an object:

```
Animator anim = AnimatorInflater.loadAnimator(context, R.animator.file1);
anim.setTarget(targetObject);
```

Fall leaf Animation Demo

Activity Transition Animations

- Activity transition can be achieved by **ActivityOptions** helper class
- makeScaleUpAnimation(View source, int startX, int startY, int startWidth, int startHeight) Create an ActivityOptions specifying an animation where the new activity is scaled from a small originating area of the screen to its final full representation.
- makeThumbnailScaleUpAnimation(View source, Bitmap thumbnail, int startX, int startY) Create an ActivityOptions specifying an animation where a thumbnail is scaled from a given position to the new activity window that is being started.
- makeCustomAnimation(Context context, int enterResId, int exitResId) Create an ActivityOptions specifying a custom animation to run when the activity is displayed.

Activity Transition Animations

- Activity transition can also be achieved by **Activity** class method
- overridePendingTransition(int enterAnim, int exitAnim) Call immediately after one of the flavors of startActivity(Intent) or finish() to specify an explicit transition animation to perform next

- Sample Code:

```
Intent intent = new Intent(this,SecondActivity.class);
ActivityOptions options = ActivityOptions.makeScaleUpAnimation(view, 50, 50, 100, 100);
startActivity(intent, options.toBundle());
```

References

- http://developer.android.com/guide/topics/resources/animation-resource.html
- http://developer.android.com/reference/android/animation/package-summary.html
- http://developer.android.com/guide/topics/graphics/prop-animation.html
- http://developer.android.com/guide/topics/graphics/drawable-animation.html