Animating Views Using Scenes and Transitions

`activity`的用户界面经常因为用户输入和其它事件而改变。比如，一个`activity`中的表单当用户输入提交后可以隐藏而在其位置上显示搜索结果。

为了提供视觉上的延续性，你可以在你的用户界面使用动画来表现切换view。这些动画给用户反馈他们所做的工作并且帮助他们学习你的应用是怎么工作的。

`Android`包含了`transition`框架，它可以让你用动画轻松表现在两个`view`层级之间的变化。这个框架在在运行时通过改变`view`的属性值来实现动画。这个框架包括一些一般效果的内置动画你也可以在`transition`的生命周期回调函数中创建你自己的动画。

这个课程教你如何使用`transition`框架中的内置动画来表现view层级之间的变化，课程也包括如何创建自定义动画。

> 注意：Android 4.4.2（API 19）之前但大于或等于Android 4.0（API 14），使用Android support library的 android.support.transition 包。

Transitions框架

给你的用户界面设置动画不仅在视觉上有吸引力。突出改变的地方并且提供视觉的提示可以帮助用户学习你的应用是怎么工作的。

`Android`提供了`transitions`框架来帮助你在不同`view`层级间添加动画。框架将一个或多个动画应用到`view`层级中的所有`view`上，当他们中发生变化时。

框架有如下特性：

- 组级别的动画
  将一个或多个动画应用到`view`层级的所有`view`上
- 基于转变的动画
  根据`view`属性始和终的值来运行动画
- 内置动画
  包含一些一般效果的预先定义的动画，比如淡出或者移动
- 资源文件支持
  从布局文件中加载`view`层级和内置动画
- 生命周期回调函数
  定义的回调函数对动画和层级变化的过程提供优秀的控制

预览

图1的例子演示了动画如何为用户提供视觉山的提示。

transition_sample_video.mp4

804.6 KB

这个动画是使用`transition`框架的例子。框架给两个`view`层级的所有`view`的改变添加动画。一个`view`层级可以是一个单独的`view`，也可以是像一个包含`view`树的`ViewGroup`一样复杂。框架根据`view`的一些属性的始和终的值来实现动画。

`transitions`框架和`view`层级和动画平行工作。这个框架的目的是存储`view`层级的状态，改变层级来修改设备屏幕的外观，给改变添加动画通过存储和应用动画定义。

图2中的图表说明了视图层级、框架对象、和动画之间的关系。

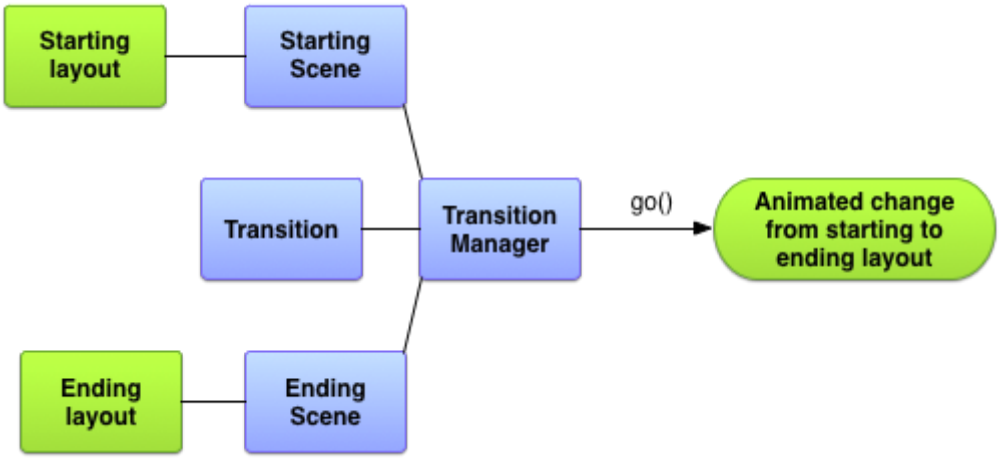

图2 transitions框架中的关系

`transition`框架的`scenes`，`transition`，`transition manager`的抽象定义，会在下面的部分详细说明。要使用`transition`框架，你需要给要在之中改变的`view`层级们创建`scenes`。接下来，为每个你要使用的动画创建`transition`。要开始两个`view`层级之间的动画，你使用`transition manager`来指定`transition`和结束`scene`。

Scenes

一个`scene`存储`view`层级的状态，包括其中的所有`view`以及他们的属性。存储`view`层级的状态到`scene`中使你可以变换到另一个`scene`中的状态中。框架提供`Scene`类来表现一个场景。

`transition`框架允许你从布局文件或者代码中的`ViewGroup`对象来创建`scene`。在代码中创建`scene`对你动态创建`view`层级或者在运行时改变`view`层级有帮助。

大部分情况下，你不用明确的创建开始的`scene`。如果你应用了`transition`，框架使用上一个结束的`scene`作为接下来所有的`scene`的开始`scene`。如果你没有应用，框架从当前屏幕的`view`中收集信息。

当你改变`scene`时，`scene`也可以定义它自己的动作来执行。比如，当你转换到一个`scene`之后要清除`view`的设置时，这个特性很有用。

除了`view`层级和它的属性值之外，`scene`还存储了`view`层级的父级的引用。根视图称为`scene root`。`scene`的改变和影响`scene`的动画都在`scene root`中发生。

Transition

`transitions`框架中，动画创建了一系列的帧来描述在开始`scene`和结束`scene`之间的`view`层级。动画的信息存储在一个`Transition`对象中。要开始这个动画，你要使用`Transition manager`实例来应用一个`transition`。框架可以在两个不同的`scene`之间变化，也可以从当前`scene`变换到一个不同的状态。

框架为常用的动画效果包括了一组内置动画，比如淡入淡出和调整大小。你也可以定义你自己的`transition`使用动画框架的API来创建动画效果。`transition`框架允许你将不同的动画包含在一个`transition`集合中。

`transition`的生命周期跟`activity`的生命周期类似，它代表了在开始和结束动画间的系统监测的变换状态。在重要的生命周期阶段，框架调用回调方法，你可以实现这些回调方法来在`transition`的不同阶段对你的用户界面做出调整。

一些限制

这部分列出了`transition`框架的一些已知的限制：

- 应用到`SurfaceView`的动画可能显示不正常。`SurfaceView`不从UI线程更新，所以它的更新可能跟其它`view`的动画不同步。
- 当一些特定的`transition`应用到`TextureView`时，可能不会产生预期的动画。
- 继承`AdapterView`的类，比如`ListView`，管理它们的子`view`的方法和`transition manager`不兼容。
- 如果你想给`TextView`调整大小添加动画，在完成大小调整之前，文字会弹到一个新的位置。为了避免这种情况，不要给包含文字的`view`大小改变添加动画。

创建Scene

开始的`scene`通常从用户界面的当前状态自动生成。对于终止`scene`，框架允许你从布局文件或者代码中的一组`view`创建。

这个课程向你展示怎样创建一个`scene`以及如何定义`scene`的动作。下一课想你展示如何两个`scene`中转换。

> 注意：框架可以给一个单独的view设置动画而不必创建scene，参见Apply a Transition Without Scenes。

从布局文件中创建scene

当文件中的`view`层级大部分时候是静止的时候使用这个方法。产生的`scene`代表你创建这个`scene`时的`view`层级的状态。如果你改变了视图层级，你需要重新创建`scene`。框架根据文件中整个层级创建`scene`，你不能从一部分布局文件中创建`scene`。

为scene定义布局文件

下面的代码演示了如何用同一个`scene root`元素创建两个不同的`scene`。代码也演示了你可以加载多个不相关的`scene`对象而不用表明他们是如何相互联系的。

例子包含了如下的布局定义：

- 包含一个文本标签的主布局和一个子布局
- 第一个场景的有两个文本标签的`relativelayout`
- 第二个场景的同样有两个文本标签但顺序不一样的`relativelayout`

所有的动画都发生在主布局的子布局中。主布局中的文本标签是静态的。

主布局如下：
`res/layout/activity_main.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/master_layout" >
    <TextView
        android:id="@+id/title"
        ...
        android:text="Title"/>
    <FrameLayout
        android:id="@+id/scene_root" >
        <include layout="@layout/a_scene" />
    </FrameLayout>
</LinearLayout>
```

第一个场景的布局文件包含在主布局中，这可以让应用把它作为初始化用户界面的一部分并且把它载入一个`scene`中，应为框架只能将一个完整的布局文件载入一个`scene`中。

第一个场景的布局文件如下：
`res/layout/a_scene.xml`

```
<RelativeLayout xmlns:android= "http://schemas.android.com/apk/res/androi
d"
    android: id="@+id/scene_container"
    android:layout_width= "match_parent"
    android:layout_height= "match_parent" >
    <TextView
        android: id="@+id/text_view1
        android:text=" Text Line 1" />
    <TextView
        android:id=" @+id/text_view2
        android: text="Text Line 2" />
</RelativeLayout>
```

第二个场景包含两个想同的文本标签（ID也相同），但顺序不一样。
`res/layout/another_scene.xml`

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/androi
d"
    android:id="@+id/scene_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/text_view2
        android:text="Text Line 2" />
    <TextView
        android:id="@+id/text_view1
        android:text="Text Line 1" />
</RelativeLayout>
```

从布局生成scene

在你定义两个`relative`布局之后，你可以为它们每一个获取一个`scene`。这使得你稍后可以在这两个`UI`配置之间转换。要获得一个`scene`，你需要一个`scene root`的引用以及布局资源`ID`。

如下代码演示了如何获取`scene root`的引用以及从布局文件创建两个`Scene`对象：

```
Scene mAScene;
Scene mAnotherScene;

// Create the scene root for the scenes in this app
mSceneRoot = (ViewGroup) findViewById(R.id.scene_root);

// Create the scenes
mAScene = Scene.getSceneForLayout(mSceneRoot, R.layout.a_scene,  this);
mAnotherScene =
    Scene.getSceneForLayout(mSceneRoot, R.layout.another_scene,  this);
```

这个应用中现在有两个`Scene`对象。二者使用的`scene root`都是在`res/layout/activity_main.xml.`中定义的`FrameLayout`。

在代码中创建scene

你也可以根据一个`ViewGroup`对象在你的代码中创建`Scene`实例。当你在代码中直接修改`view`层级或者你要动态生成它们时使用这个技术。

要在代码中根据`view`层级生成`scene`，使用`Scene(sceneRoot, viewHierarchy)`这个构造方法。这个方法与`Scene.getSceneForLayout()`方法等价。

下面的代码片段演示了如何在你的代码中创建scene：

```
Scene mScene;

// Obtain the scene root element
mSceneRoot = (ViewGroup) mSomeLayoutElement;

// Obtain the view hierarchy to add as a child of
// the scene root when this scene is entered
mViewHierarchy = (ViewGroup) someOtherLayoutElement;

// Create a scene
mScene = new Scene(mSceneRoot, mViewHierarchy);
```

创建Scene action

框架允许你自定义`scene actions`，在进入或者退出一个`scene`时运行。大部分情况下，自定义`scene action`不是必须的，因为框架自动给`scenes`之间的变化添加动画。

`scene actions`在处理这些情况时有用：

- 给不在同一个视图层级的`view`添加动画。你可以使用退出和进入的`scene action`来给开始和结束`scene`添加动画。
- 框架不能自动添加动画的`view`，例如`ListView`对象。更多信息参见一些限制

要提供自定义的`scene action`，把你的`actions`定义为`Runnable`对象然后传递给`Scene.setExitAction()`或`Scene.setEnterAction()`。框架在转换动画开始之前在开始`scene`上调用`setExitAction ()`，在转换动画之后在结束`scene`上调用`setEnterAction()`。

> 注意：不要使用scene action在始和终的scene的view中传递数据。更多信息参见 Defining Transition Lifecycle Callbacks

应用一个Transition

在`transition`框架中，动画创建一系列的帧来描述在始和终的`scene`中视图层级之间的变化。框架把这些动画作为`transition`对象，包含了一个动画的信息。要运行动画，你需要提供要使用的`transition`和结束`scene`给`transition manager`。

这个课程教你使用内置的`transition`来在两个`scene`之间运行移动、改变大小、淡入淡出动画。下节课教你怎么自定义`transition`。

创建一个transition

上节课中，你学习了怎么创建代表视图层级状态的`scene`。当你定义了始和终的`scene`，你需要创建一个定义了动画的`transition`对象。框架允许你在布局文件中指定一个内置的`transition`并且将它填充到你的代码中，或者在你的代码中直接创建一个内置`transition`的实例。

表1 内置`transition`类型

Class	Tag	Attributes	Effect
AutoTransition	<autoTransition/>	-	Default transition. Fade out, move and resize, and fade in views, in that order.
Fade	<fade/>	android:fadingMode=" [fade_in | fade_out | fade_in_out]"	fade_in fades in views fade_out fades out views fade_in_out (default) does a fade_out followed by a fade_in.
ChangeBounds	<changeBounds/>	-	Moves and resizes views.

从资源文件创建transition实例

这个方法允许你不修改代码而改变`transition`定义。在从你的代码中分离复杂`transition`定义时此方法也很有用，参见Specify Multiple Transitions.

在资源文件中定义一个内置的`transition`，按照以下步骤：

1. 给你的工程添加`res/transition/`目录
2. 在这个目录中新建一个`XML`文件
3. 添加一个内置`transition`的`XML`节点

例如，下面的资源文件制定了Fade transition:
`res/transition/fade_transition.xml`

```
<fade xmlns:android="http://schemas.android.com/apk/res/android"  />
```

下面的代码片段演示了怎么从资源文件填充一个`Transition`实例：

```
Transition mFadeTransition =
        TransitionInflater.from(this).
        inflateTransition(R.transition.fade_transition);
```

在你的代码中创建一个transition实例

这个方法对动态创建`transition`对象很有用，如果你在代码中修改用户界面和使用或者不使用参数创建简单的内置`transition`实例。

要创建一个内置`transition`的实例，调用`Transition`子类的某个构造方法。例如，下面的代码片段创建了一个`Fade Transition`的实例：

```
Transition mFadeTransition = new Fade();
```

应用一个Transition

通常根据一个事件将`Transition`应用到不同的`view`层级变化上，比如用户动作。例如，一个搜索应用：当用户输入一个字，点击搜索按钮，应用变化到代表结果的布局中同时应用一个`Transition`淡出搜索按钮淡入搜索结果。

要实现一个场景变换同时应用一个`Transition`来响应你的`activity`的某些事件，使用结束`scene`和`transition`实例调用`TransitionManager.go()`静态方法实现动画，如下所示：

```
TransitionManager.go(mEndingScene, mFadeTransition);
```

框架使用结束`scene`中的视图层级改变在`scene root`中的视图层级同时运行`transition`实例指定的动画。开始`transition`是上一个`transition`的结束`transition`。如果没有上一个`transition`，开始`transition`由当前用户界面状态决定。

如果你不指定一个`transition`实例，`transition manager`会应用一个自动的`transition`符合大部分情况。更多信息参见TransitionManager类。

选择指定的目标view

框架默认将`transition`应用到始和终的场景中的所有`view`上。某些情况下，你可能只想将`transition`应用到`scene`的一部分`view`上。例如，框架不支持对`ListView`的变化设置动画，所以你不应在`transition`里给他们设置动画。

`transition`设置动画的每个`view`称为`target`。你只能选择与一个`scene`相关的视图层级的一部分。

要从`target`列表删除一个或多个`view`，在开始`transition`之前调用`removeTarget()`。要想`target`列表添加`view`，调用`addTarget()`。更多信息参看Transition类。

指定多个Transition

为了获得一个动画的大部分效果，你应该将它与`scene`之间的变化类型匹配。例如，如果你在`scene`间删除和添加一些`view`，淡出/淡入动画会给出显而易见的提示一些`view`不可用了。如果你将view移到屏幕不同的地方，最好用一个动画这样用户可以注意到`view`的新位置。

你不用只选择一个动画，因为框架允许在包含一组独立的内置或者自定义的`transition`的`transition`集中设置多个动画效果。

根据`XML`中的一些`transition`来定义一个`transition`集，在`/res/transition/`文件夹中新建一个文件然后在`transitionset`元素下列出`transition`。例如，下面的代码片段展示了如何指定一个`transitionSet`跟`AutoTransition`有同样的效果：

```
<transitionSet xmlns:android="http://schemas.android.com/apk/res/androi
d"
    android:transitionOrdering="sequential">
    <fade android:fadingMode="fade_out" />
    <changeBounds />
    <fade android:fadingMode="fade_in" />
</transitionSet>
```

在代码中给`TransitionSet`对象填充`transition`集，在`activity`中调用`TransitionInflater.from()`。`TransitionSet`类继承于`Transition`类，所以你可以通过`TransitionManager`来使用它。

应用Transition不使用Scene

改变`view`层级不是修改用户界面的唯一方法。你也可以在当前层级中添加、修改、移除子视图。例如，你可以用一个布局实现搜索交互。以一个搜索框和一个搜索图标开始。要改变界面来显示结果，移除搜索按钮当用户点击时通过调用`ViewGroup.removeView()`方法，然后添加搜索结果通过调用`ViewGroup.addView()`方法。

如果可选择的有两个差不多的`view`层级，你可能想使用这个方法。当用户界面中的差别较小时，比起要创建和维持两个独立的布局文件，你可以使用一个布局文件包含一个视图层级然后在代码中修改。

如果在当前视图层级使用这种方法改变用户界面，你就不用创建一个`scene`。代替的是，你可以在一个视图层级的两种状态之间使用延迟的`transition`。`transition`框架的这个特性以当前视图层级状态开始，记录你对它其中`view`的改变，然后当系统重绘用户界面时应用一个`transition`来添加动画。

在单一视图层级中创建一个延迟`transition`，按照下面的步骤：

1. 当触发`transition`的事件发生时，调用`TransitionManager.beginDelayedTransition()`方法来提供一个所有你要改变的`view`的父`view`以及`transition`。框架存储当前子`view`的状态以及它们的属性值。
2. 按照需要改变子`view`。框架记录你对子`view`所做的改变以及它们的属性。
3. 当系统根据你的改变重绘界面时，框架根据初始状态和新状态来添加动画。

下面的例子展示了使用延迟`transition`给添加一个`text view`到一个`view`层级添加动画。第一个代码片段是布局定义文件：

`res/layout/activity_main.xml`

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/androi
d"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <EditText
        android:id="@+id/inputText"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    ...
</RelativeLayout>
```

下一个代码片段展示了添加一个`text view`的动画代码：

`MainActivity.java`

```
private TextView mLabelText;
private Fade mFade;
private ViewGroup mRootView;
...

// Load the layout
this.setContentView(R.layout.activity_main);
...

// Create a new TextView and set some View properties
mLabelText = new TextView();
mLabelText.setText("Label").setId("1");

// Get the root view and create a transition
mRootView = (ViewGroup) findViewById(R.id.mainLayout);
mFade = new Fade(IN);

// Start recording changes to the view hierarchy
TransitionManager.beginDelayedTransition(mRootView, mFade);

// Add the new TextView to the view hierarchy
mRootView.addView(mLabelText);

// When the system redraws the screen to show this update,
// the framework will animate the addition as a fade in
```

定义Transition生命周期回调函数

`transition`的生命周期跟`activity`的类似。它代表了框架监测的在调用`TransitionManager.go()`和动画结束之间的`transition`的状态。在重要的生命周期状态下，框架调用`TransitionListener`接口定义的回调方法。

`transition`生命周期回调方法很有用，例如，在`scene`改变时从开始的`view`层级复制一个`view`的属性值到结束的`view`层级。你可以简单的复制值从它的开始`view`到结束层级的`view`中，因为结束`view`层级不填充直到`transition`完成。代替的，你需要存储值到一个变量中然后将它复制到结束`view`中当框架完成`transition`时。要获得`transition`完成的通知，你可以在你的`activity`中实现`TransitionListener.onTransitionEnd()`。

更过信息，参见TransitionListener类。

创建自定义的transition

一个自定义的`transition`允许你创建内置`transition`中没有的动画。例如，你可以定义一个`transition`将文本和输入域的前景色设置为灰色来指示该输入域已禁用。

自定义`transition`，为始和终的`scene`应用动画到子`view`。和内置`transition`不同的是，你需要提供获取属性值和产生动画的代码。你可能也想为你的动画定义目标view的子集。

这个课程教你获取属性值和生成动画来创建自定义`transition`。

继承Transition类

要创建自定义`transition`，创建一个类继承`Transition`类并重写如下方法：

```
public class CustomTransition extends Transition {

    @Override
    public void captureStartValues(TransitionValues values)  {}

    @Override
    public void captureEndValues(TransitionValues values)  {}

    @Override
    public Animator createAnimator(ViewGroup sceneRoot,
                                   TransitionValues startValues,
                                   TransitionValues endValues)  {}
}
```

下面的部分介绍如何重写这些方法。

获取view属性值

`transition`动画使用属性动画系统。

但是，属性动画通常只需要`view`属性的一部分。例如，一个颜色动画需要颜色属性，一个移动动画需要位置属性。因为需要的属性指定给了`transition`，`transition`框架不会提供所有属性值给一个`transition`。而是，框架调用回调方法允许一个`transition`获取它需要的属性值然后存储它们。

获取开始值

要传递开始值给框架，实现`captureStartValues(transitionValues)`方法。框架为开始`scene`的每个`view`调用这个方法。这个方法参数是一个包含view引用以及一个你可以存储`view`值的`Map`实例的`TransitionValues`对象。在你的实现里，获取这些属性值通过存储它们到`map`中来传递回框架。

为了确保属性值的键不与其它`TransitionValue`的键冲突，使用下面的命名规则：
`package_name:transition_name:property_name`
下面的代码片段展示了一个`captureStartValues()`方法的实现：

```
public class CustomTransition extends Transition {

    // Define a key for storing a property value in
    // TransitionValues.values with the syntax
    // package_name:transition_class:property_name to avoid collisions
    private static final String PROPNAME_BACKGROUND =
            "com.example.android.customtransition:CustomTransition:backg
round";

    @Override
    public void captureStartValues(TransitionValues transitionValues) {
        // Call the convenience method captureValues
        captureValues(transitionValues);
    }


    // For the view in transitionValues.view, get the values you
    // want and put them in transitionValues.values
    private void captureValues(TransitionValues transitionValues) {
        // Get a reference to the view
        View view = transitionValues.view;
        // Store its background property in the values map
        transitionValues.values.put(PROPNAME_BACKGROUND, view.getBackgro
und());
    }
    ...
}
```

获取结束值

框架为结束`scene`每个目标`view`调用`captureEndValues(TransitionValues)`方法一次。`captureEndValues()`工作方式与`captureStartValues()`相同。

下面的代码片段展示了一个`captureEndValues()`方法的实现：

```
@Override
public void captureEndValues(TransitionValues transitionValues) {
    captureValues(transitionValues);
}
```

在这个例子中，`captureStartValues()`和`captureEndValues()`都调用了`captureValues()`来获取`view`属性，但是获取的值不同。

创建自定义Animator

给一个view在始和终的`scene`的状态变化添加动画，你需要提供一个`Animator`通过重写`createAnimator()`方法。当框架调用这个方法，它传入`scene root`视图和`TransitionValues`对象。

框架调用`createAnimator()`方法的次数取决于始和终的`scene`之间发生的变化。例如，考虑一个自定义的淡入淡出动画。如果开始`scene`有五个目标，其中两个在结束`scene`时移除了，结束`scene`有三个目标还有一个新目标，那么框架调用`createAnimator()`6次：其中3次发生在始和终的`scene`都存在的3个目标的淡入淡出动画上，2次发生在移除的目标的淡出动画上，1次发生在淡入新目标的淡入动画上。

对于始和终的`scene`中都存在的目标`view`，框架为`startValue`和`endValue`参数提供`TransitionValues`对象。对于只存在与开始或者结束`scene`的目标`view`，框架为对应的`scene`提供`TransitionValues`对象，另一个为`null`。

要实现`createAnimator(ViewGroup, TransitionValues, TransitionValues)`方法当你自定义`transition`时，使用你获取的`view`属性值来创建一个`Animator`对象然后将它返回给框架。实现的例子，参见`CustomeTransition`例子中的`ChangeColor`类。