使用过渡为布局变化添加动画效果

借助 Android 的过渡框架，您只需提供起始布局和结束布局，即可为界面中的各种运动添加动画效果。您可以选择所需的动画类型（例如，淡入/淡出视图或更改视图尺寸），而过渡框架会确定如何为从起始布局到结束布局的运动添加动画效果。

过渡框架包含以下功能：

- **群组级动画**：将一个或多个动画效果应用于视图层次结构中的所有视图。
- **内置动画**：对淡出或移动等常见效果使用预定义动画。
- **资源文件支持**：从布局资源文件加载视图层次结构和内置动画。
- **生命周期回调**：接收可控制动画和层次结构更改流程的回调。

注意：本页介绍了如何在同一 Activity 的各个布局之间打造过渡效果。如果用户在多个 Activity 之间移动，您应改为参阅启动使用动画的 Activity (/training/transitions/start-activity)。

如需查看为布局变化添加动画效果的示例代码，请参阅 BasicTransition (https://github.com/android/animation-samples/tree/master/BasicTransition)。

在两种布局之间添加动画效果的基本流程如下所示：

1. 为起始布局和结束布局创建一个 `Scene` (/reference/android/transition/Scene) 对象。然而，起始布局的场景通常是根据当前布局自动确定的。
2. 创建一个 `Transition` (/reference/android/transition/Transition) 对象以定义所需的动画类型。
3. 调用 `TransitionManager.go()` (/reference/android/transition/TransitionManager#go(android.transition.Scene))，然后系统会运行动画以交换布局。

图 1 中的示意图说明了布局、场景、过渡和最终动画之间的关系。

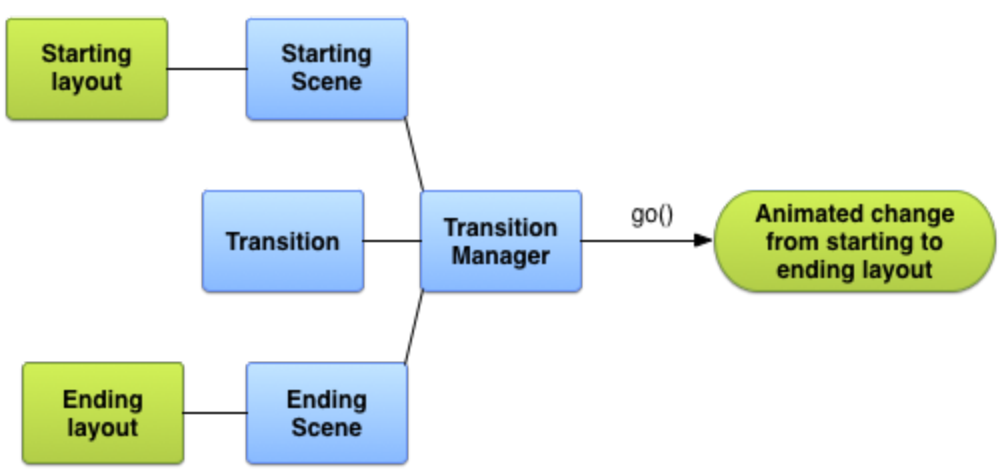

图 1. 过渡框架如何创建动画的基本图示

创建场景

场景可存储视图层次结构的状态，包括其中的所有视图及其属性值。过渡框架可以在起始场景和结束场景之间运行动画。

您可以从布局资源文件或代码中的一组视图创建场景。然而，过渡的起始场景通常是根据当前界面自动确定的。

场景还可以定义其自己的操作（这些操作在您进行场景更改时运行）。例如，此功能对于在过渡到场景后清理视图设置非常有用。

注意：框架可以为未使用场景的单个视图层次结构中的变化添加动画效果，如应用没有场景的过渡 (#NoScenes)中所述。不过，了解场景对于使用过渡而言至关重要。

从布局资源创建场景

您可以直接从布局资源文件创建 **Scene** (/reference/android/transition/Scene) 实例。如果文件中的视图层次结构大部分是静态的，请使用此方法。生成的场景表示您在创建 **Scene** (/reference/android/transition/Scene) 实例时视图层次结构的状态。如果您更改视图层次结构，则必须重新创建场景。该框架从文件内的整个视图层次结构创建场景；您无法从部分布局文件创建场景。

如需从布局资源文件创建 **Scene** (/reference/android/transition/Scene) 实例，请从布局中检索场景根作为 **ViewGroup** (/reference/android/view/ViewGroup) 实例，然后使用场景根和包含场景视图层次结构的布局文件的资源 ID 调用 **Scene.getSceneForLayout()** (/reference/android/transition/Scene#getSceneForLayout(android.view.ViewGroup, int, android.content.Context)) 函数。

定义场景布局

本节其余部分的代码段展示了如何使用相同的场景根元素创建两个不同的场景。这些代码段还表明您可以加载多个不相关的 **Scene** (/reference/android/transition/Scene) 对象，而不暗示它们彼此相关。

该示例包含以下布局定义：

- 具有文本标签和子布局的 Activity 的主布局。
- 具有两个文本字段的第一个场景的相对布局。
- 具有两个以不同顺序放置的相同文本字段的第二个场景的相对布局。

该示例经过特别设计，使所有的动画都发生在 Activity 的主布局的子布局中。主布局中的文本标签保持静态。

Activity 的主布局定义如下：

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>
```

此布局定义包含一个文本字段和一个场景根的子布局。第一个场景的布局包含在主布局文件中。这样，应用便可以将其作为初始界面的一部分显示，还可以将其加载到场景中，因为该框架只能将整个布局文件加载到场景中。

第一个场景的布局定义如下：

res/layout/a_scene.xml

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

第二个场景的布局包含两个以不同顺序放置的相同文本字段（具有相同的 ID），其定义如下：

res/layout/another_scene.xml

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

从布局生成场景

为两个相对布局创建定义后，您可以针对每个布局获取一个场景。通过这种方式，您之后可以在两种界面配置之间进行过渡。如需获取场景，您需要引用场景根和布局资源 ID。

以下代码段展示了如何获取对场景根的引用，以及如何从布局文件创建两个 **Scene** (/reference/android/transition/Scene) 对象：

KOTLIN (#KOTLIN)**JAVA**

```
Scene aScene;
Scene anotherScene;

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

// Create the scenes
aScene = Scene.getSceneForLayout(sceneRoot, R.layout.a_scene, this);
anotherScene =
    Scene.getSceneForLayout(sceneRoot, R.layout.another_scene, this);
```

在应用中，现在有两个基于视图层次结构的 **Scene** (/reference/android/transition/Scene) 对象。这两个场景均使用 **res/layout/activity_main.xml** 中的 **FrameLayout** (/reference/android/widget/FrameLayout) 元素定义的场景根。

在代码中创建场景

您还可以在代码中通过 **ViewGroup** (/reference/android/view/ViewGroup) 对象创建 **Scene** (/reference/android/transition/Scene) 实例。如果您直接在代码中修改视图层次结构或动态生成视图层次结构，请使用此方法。

如需在代码中通过视图层次结构创建场景，请使用 **Scene(sceneRoot, viewHierarchy)** (/reference/android/transition/Scene#Scene(android.view.ViewGroup, android.view.View)) 构造函数。调用此构造函数等同于在扩充布局文件后调用 **Scene.getSceneForLayout()** (/reference/android/transition/Scene#getSceneForLayout(android.view.ViewGroup, int, android.content.Context)) 函数。

以下代码段展示了如何在代码中通过场景根元素和场景的视图层次结构创建 **Scene** (/reference/android/transition/Scene) 实例：

KOTLIN (#KOTLIN)**JAVA**

```
Scene mScene;

// Obtain the scene root element
sceneRoot = (ViewGroup) someLayoutElement;

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

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

创建场景操作

借助该框架，您可以定义在进入或退出场景时系统运行的自定义场景操作。在很多情况下，无需定义自定义场景操作，因为框架会自动为场景更改添加动画效果。

场景操作有助于处理以下情况：

- 为不在同一层次结构中的视图添加动画效果。您可以使用退出和进入场景操作为起始场景和结束场景的视图添加动画效果。
- 为过渡框架无法自动添加动画效果的视图（例如，`ListView` (/reference/android/widget/ListView) 对象）添加动画效果。如需了解详情，请参阅限制 (#Limitations)。

如需提供自定义场景操作，请将您的操作定义为 `Runnable` (/reference/java/lang/Runnable) 对象，并将它们传递给 `Scene.setExitAction()` (/reference/android/transition/Scene#setExitAction(java.lang.Runnable)) 或 `Scene.setEnterAction()` (/reference/android/transition/Scene#setEnterAction(java.lang.Runnable)) 函数。该框架会在运行过渡动画之前，在起始场景中调用 `setExitAction()` (/reference/android/transition/Scene#setExitAction(java.lang.Runnable)) 函数；并在运行过渡动画之后，在结束场景中调用 `setEnterAction()` (/reference/android/transition/Scene#setEnterAction(java.lang.Runnable)) 函数。

注意：请勿使用场景操作在起始场景和结束场景的视图之间传递数据。如需了解详情，请参阅定义过渡生命周期回调 (#Callbacks)。

应用过渡

过渡框架表示具有 `Transition` (/reference/android/transition/Transition) 对象的场景之间的动画样式。您可以使用多个内置子类（例如，`AutoTransition` (/reference/android/transition/AutoTransition) 和 `Fade` (/reference/android/transition/Fade)）实例化 `Transition` (/reference/android/transition/Transition)，也可以定义自己的过渡 (/training/transitions/custom-transitions)。然后，您可以将结束 `Scene` (/reference/android/transition/Scene) 和 `Transition` (/reference/android/transition/Transition) 传递到 `TransitionManager.go()` (/reference/android/transition/TransitionManager#go(android.transition.Scene, android.transition.Transition))，以在场景之间运行动画。

过渡生命周期与 Activity 生命周期类似，它表示框架在动画开始和完成期间监控到的过渡状态。在重要的生命周期状态下，框架会调用一些回调函数（您可以实现这些回调函数，以在不同的过渡阶段对界面进行调整）。

创建过渡

在上一部分中，您学习了如何创建代表不同视图层次结构状态的场景。定义要在这些场景之间更改的起始场景和结束场景之后，您需要创建一个定义动画的 `Transition` (/reference/android/transition/Transition) 对象。通过该框架，您可以在资源文件中指定内置过渡并在代码中对其进行扩充，或者直接在代码中创建内置过渡实例。

表 1. 内置过渡类型。

类	标记	属性	影响
`AutoTransition` (/reference/android/transition/AutoTransition)	<autoTransition/>	-	默认过渡。按该顺序淡出视图、移动视图并调整其大小，以及淡入视图。
`Fade` (/reference/android/transition/Fade)	<fade/>	`android:fadingMode="[fade_in | fade_out | fade_in_out]"`	`fade_in` 淡入视图 `fade_out` 淡出视图 `fade_in_out`（默认）先执行 `fade_out` 后执行 `fade_in`。
`ChangeBounds` (/reference/android/transition/ChangeBounds)	<changeBounds/>	-	移动视图并调整其大小。

从资源文件创建过渡实例

通过此方法，您无需更改 Activity 的代码即可修改过渡定义。它还有助于将复杂过渡定义与应用代码分离，如指定多个过渡 (#Multiple)中所述。

如需在资源文件中指定内置过渡，请按以下步骤操作：

1. 将 **res/transition/** 目录添加到项目中。
2. 在此目录中新建一个 XML 资源文件。
3. 为其中一个内置过渡添加 XML 节点。

例如，以下资源文件指定了 **Fade** (/reference/android/transition/Fade) 过渡：

res/transition/fade_transition.xml

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

以下代码段展示了如何从资源文件扩充 Activity 内的 **Transition** (/reference/android/transition/Transition) 实例：

KOTLIN (#KOTLIN)**JAVA**

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

在代码中创建过渡实例

如果您在代码中修改界面，或者使用较少或不使用参数创建简单的内置过渡实例，则该方法有助于动态创建过渡对象。

如需创建内置过渡的实例，请在 **Transition** (/reference/android/transition/Transition) 类的子类中调用其中一个公开构造函数。例如，以下代码段可创建 **Fade** (/reference/android/transition/Fade) 过渡的实例：

KOTLIN (#KOTLIN)**JAVA**

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

应用过渡

您通常会应用过渡以在不同的视图层次结构之间更改，从而响应用户操作等事件。例如，假设有这样一个搜索应用：当用户输入搜索字词并点击搜索按钮时，该应用会切换到表示结果布局的场景，同时应用淡出搜索按钮并淡入搜索结果的过渡。

如需在应用过渡以响应 Activity 中的某个事件时更改场景，请使用用于动画的结束场景和过渡实例调用 **TransitionManager.go()** (/reference/android/transition/TransitionManager#go(android.transition.Scene)) 类函数，如以下代码段所示：

KOTLIN (#KOTLIN)**JAVA**

```
TransitionManager.go(endingScene, fadeTransition);
```

该框架会在运行由过渡实例指定的动画时，使用结束场景中的视图层次结构更改场景根内的视图层次结构。起始场景是上次过渡的结束场景。如果之前没有过渡，则起始场景根据界面的当前状态自动确定。

如果您未指定过渡实例，过渡管理程序可以应用自动过渡，此类过渡可执行适合大多数情况的合理操作。如需了解详情，请参阅 **TransitionManager** (/reference/android/transition/TransitionManager) 类的 API 参考文档。

选择特定的目标视图

默认情况下，该框架会将过渡应用于起始场景和结束场景中的所有视图。在某些情况下，您可能只希望将动画应用于某个场景中的部分视图。例如，该框架不支持为 **`ListView`** (/reference/android/widget/ListView) 对象的变化添加动画效果，因此您不应尝试在过渡期间为它们添加动画效果。借助该框架，您可以选择想要添加动画效果的特定视图。

过渡添加动画效果的各个视图称为目标。您只能选择与某个场景关联的视图层次结构中的目标。

如需从目标列表中移除一个或多个视图，请先调用 **`removeTarget()`** (/reference/android/transition/Transition#removeTarget(android.view.View)) 方法，然后再开始过渡。如果仅将您指定的视图添加到目标列表，请调用 **`addTarget()`** (/reference/android/transition/Transition#addTarget(android.view.View)) 函数。如需了解详情，请参阅 **`Transition`** (/reference/android/transition/Transition) 类的 API 参考文档。

指定多个过渡

要使动画发挥最大效果，您应该使其与场景之间发生的更改类型相匹配。例如，如果您要移除某些视图并在场景之间添加其他视图，则动画中的淡出/淡出效果会提供明显的指示，以表明某些视图不再可用。如果您要将视图移动到屏幕上的不同位置，则最好选择为这种移动添加动画效果，以便用户能够注意到新的视图位置。

您不必只选择一个动画，因为借助该过渡框架，您可以组合一个过渡集（其中包含一组单独的内置过渡或自定义过渡）的动画效果。

如需从 XML 中的一系列过渡定义一个过渡集，请在 **`res/transitions/`** 目录中创建一个资源文件，并在 **`transitionSet`** 元素下列出这些过渡。例如，以下代码段展示了如何指定与 **`AutoTransition`** (/reference/android/transition/AutoTransition) 类具有相同行为的过渡集：

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

如需在代码中将过渡集扩充到 **`TransitionSet`** (/reference/android/transition/TransitionSet) 对象，请在您的 Activity 中调用 **`TransitionInflater.from()`** (/reference/android/transition/TransitionInflater#from(android.content.Context))。**`TransitionSet`** (/reference/android/transition/TransitionSet) 类扩展自 **`Transition`** (/reference/android/transition/Transition) 类，因此您可以像使用任何其他 **`Transition`** (/reference/android/transition/Transition) 实例一样，将其与过渡管理程序结合使用。

应用没有场景的过渡

更改视图层次结构并非修改界面的唯一方式，您还可以通过添加、修改和移除当前层次结构中的子视图来进行更改。例如，您可以实现仅与一个布局的搜索互动。首先使用显示一个搜索输入字段和一个搜索图标的布局。如需更改界面以显示结果，请通过调用 **`ViewGroup.removeView()`** (/reference/android/view/ViewGroup#removeView(android.view.View)) 函数在用户点击“搜索”按钮时将其移除，并通过调用 **`ViewGroup.addView()`** (/reference/android/view/ViewGroup#addView(android.view.View)) 函数添加搜索结果。

如果备选方案是具有两个几乎完全相同的层次结构，则您可能希望使用此方法。您可以具有一个包含您在代码中修改的视图层次结构的布局文件，而无需创建和维护界面只有细微差别的两个单独的布局文件。

如果您以这种方式在当前视图层次结构中进行更改，则无需创建场景。您可以改为使用延迟过渡，在两种视图层次结构状态之间创建和应用过渡。过渡框架的这一功能先显示当前视图层次结构的状态，记录您对其视图所做的更改，并在系统重新绘制界面时应用过渡（为变化添加动画效果）。

如需在单个视图层次结构中创建延迟过渡，请按以下步骤操作：

1. 发生触发过渡的事件时，请调用 **`TransitionManager.beginDelayedTransition()`** (/reference/android/transition/TransitionManager#beginDelayedTransition(android.view.ViewGroup)) 函数，以提供您想要更改的全部视图的父级视图以及要使用的过渡。该框架会存储子视图的当前状态及其属性值。
2. 根据用例的要求更改子视图。该框架会记录您对子视图及其属性所做的更改。
3. 当系统根据您的更改重新绘制界面时，框架会为原始状态和新状态之间的变化添加动画效果。

以下示例展示了如何使用延迟过渡为向视图层次结构添加文本视图添加动画效果。第一个代码段展示了布局定义文件：

res/layout/activity_main.xml

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

下一个代码段展示了为添加文本视图添加动画效果的代码：

MainActivity

KOTLIN (#KOTLIN)JAVA

```
private TextView labelText;
private Fade mFade;
private ViewGroup rootView;
...

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

// Create a new TextView and set some View properties
labelText = new TextView(this);
labelText.setText("Label");
labelText.setId(R.id.text);

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

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

// Add the new TextView to the view hierarchy
rootView.addView(labelText);

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

定义过渡生命周期回调

过渡生命周期与 Activity 生命周期类似。它表示框架在调用 `TransitionManager.go()` (/reference/android/transition/TransitionManager#go(android.transition.Scene)) 函数和完成动画期间监控到的过渡状态。在重要的生命周期状态下，框架会调用由 `TransitionListener` (/reference/android/transition/Transition.TransitionListener) 接口定义的回调。

在场景更改期间将视图属性值从起始视图层次结构复制到结束视图层次结构等情况下，过渡生命周期回调非常有用。您不能简单地将该值从其起始视图复制到结束视图层次结构中的视图，因为在过渡完成之前，系统不会扩充结束视图层次结构。相反，您需要将该值存储在某个变量中，然后在框架完成过渡之后，将其复制到结束视图层次结构中。如需在过渡完成时收到通知，您可以在 Activity 中实现 `TransitionListener.onTransitionEnd()` (/reference/android/transition/Transition.TransitionListener#onTransitionEnd(android.transition.Transition)) 函数。

如需了解详情，请参阅 `TransitionListener` (/reference/android/transition/Transition.TransitionListener) 类的 API 参考文档。

限制

本部分列出了过渡框架的一些已知限制：

- 应用于 **`SurfaceView`** (/reference/android/view/SurfaceView) 的动画可能无法正确显示。**`SurfaceView`** (/reference/android/view/SurfaceView) 实例是从非界面线程更新的，因此这些更新可能与其他视图的动画不同步。
- 当应用于 **`TextureView`** (/reference/android/view/TextureView) 时，某些特定过渡类型可能无法产生所需的动画效果。
- 扩展 **`AdapterView`** (/reference/android/widget/AdapterView) 的类（例如 **`ListView`** (/reference/android/widget/ListView)）会以与过渡框架不兼容的方式管理它们的子视图。如果您尝试为基于 **`AdapterView`** (/reference/android/widget/AdapterView) 的视图添加动画效果，则设备显示屏可能会挂起。
- 如果您尝试使用动画调整 **`TextView`** (/reference/android/widget/TextView) 的大小，则文本会在该对象完全调整大小之前弹出到新位置。为避免出现此问题，请勿为调整包含文本的视图的大小添加动画效果。