

片段

Part of **Android Jetpack** (/jetpack).

[Fragment](/reference/androidx/fragment/app/Fragment) (/reference/androidx/fragment/app/Fragment) 表示 [FragmentActivity](/reference/androidx/fragment/app/FragmentActivity) (/reference/androidx/fragment/app/FragmentActivity) 中的行为或界面的一部分。您可以在一个 Activity 中组合多个片段，从而构建多窗格界面，并在多个 Activity 中重复使用某个片段。您可以将片段视为 Activity 的模块化组成部分，它具有自己的生命周期，能接收自己的输入事件，并且您可以在 Activity 运行时添加或移除片段（这有点像可以在不同 Activity 中重复使用的“子 Activity”）。

片段必须始终托管在 Activity 中，其生命周期直接受宿主 Activity 生命周期的影响。例如，当 Activity 暂停时，Activity 的所有片段也会暂停；当 Activity 被销毁时，所有片段也会被销毁。不过，当 Activity 正在运行（处于已恢复生命周期状态 (/guide/components/activities#Lifecycle)）时，您可以独立操纵每个片段，如添加或移除片段。当执行此类片段事务时，您也可将其添加到由 Activity 管理的返回栈 — Activity 中的每个返回栈条目都是一条已发生片段事务的记录。借助返回栈，用户可以通过按返回按钮撤消片段事务（后退）。

当您将片段作为 Activity 布局的一部分添加时，其位于 Activity 视图层次结构的某个 [ViewGroup](/reference/android/view/ViewGroup) (/reference/android/view/ViewGroup) 中，并且片段会定义其自己的视图布局。您可以通过在 Activity 的布局文件中声明片段，将其作为 <fragment> 元素插入您的 Activity 布局，或者通过将其添加到某个现有的 [ViewGroup](/reference/android/view/ViewGroup) (/reference/android/view/ViewGroup)，利用应用代码将其插入布局。

本文介绍如何在开发应用时使用片段，包括如何在将片段添加到 Activity 返回栈时保持其状态、如何与 Activity 及 Activity 中的其他片段共享事件、如何为 Activity 的应用栏发挥作用等等。

如需了解有关处理生命周期的信息（包括最佳实践的相关指导），请参阅以下资源：

- [使用具有生命周期感知能力的组件处理生命周期](/topic/libraries/architecture/lifecycle) (/topic/libraries/architecture/lifecycle)
- [应用架构指南](/topic/libraries/architecture/guide) (/topic/libraries/architecture/guide)
- [支持平板电脑和手机](/guide/practices/tablets-and-handsets) (/guide/practices/tablets-and-handsets)

设计原理

Android 在 Android 3.0（API 级别 11）中引入了片段，主要目的是为大屏幕（如平板电脑）上更加动态和灵活的界面设计提供支持。由于平板电脑的屏幕尺寸远胜于手机屏幕尺寸，因而有更多空间可供您组合和交换界面组件。利用片段实现此类设计时，您无需管理对视图层次结构做出的复杂更改。通过将 Activity 布局分成各个片段，您可以在运行时修改 Activity 的

外观，并在由 Activity 管理的返回栈中保留这些更改。现在，您可以通过[片段支持库](/topic/libraries/support-library/packages#v4-fragment) (/topic/libraries/support-library/packages#v4-fragment) 获取大量片段。

例如，新闻应用可以使用一个片段在左侧显示文章列表，使用另一个片段在右侧显示文章。两个片段并排显示在一个 Activity 中，**每个片段都拥有自己的一套生命周期回调方法，并各自处理自己的用户输入事件**。因此，用户无需使用一个 Activity 来选择文章，然后使用另一个 Activity 来阅读文章，而是可以在同一个 Activity 内选择文章并进行阅读，如图 1 中的平板电脑布局所示。

您应将每个片段都设计为可重复使用的模块化 Activity 组件。换言之，由于每个片段都会通过各自的生命周期回调来定义自己的布局和行为，您可以将一个片段加入多个 Activity，因此，您应采用可复用式设计，**避免直接通过某个片段操纵另一个片段**。这一点颇为重要，因为模块化片段允许您**更改片段的组合方式，从而适应不同的屏幕尺寸**。在设计可同时支持平板电脑和手机的应用时，您可以在不同的布局配置中重复使用您的片段，以根据可用的屏幕空间优化用户体验。例如，在手机上，如果不能在同一 Activity 内储存多个片段，则可能必须利用单独的片段来实现单窗格界面。

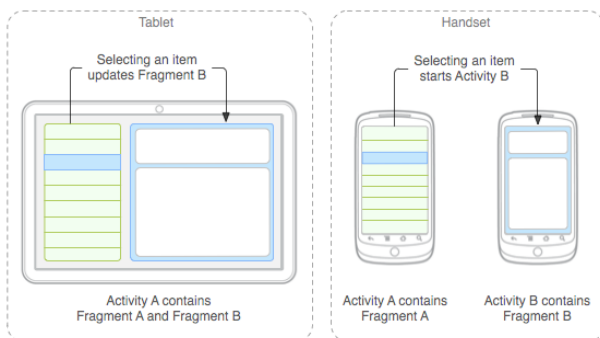


图 1. 由片段定义的两个界面模块如何适应不同设计的示例：通过组合成一个 Activity 来适应平板电脑设计，通过单独片段来适应手机设计。

例如（仍以新闻应用为例），在平板电脑尺寸的设备上运行时，该应用可以在 Activity A 中嵌入两个片段。不过，手机尺寸的屏幕没有足够的空间来存储两个片段，因此 Activity A 只包含用于显示文章列表的片段，并且当用户选择文章时，它会启动 Activity B，其包含用于阅读文章的第二个片段。因此，应用可通过重复使用不同组合的片段来同时支持平板电脑和手机（如图 1 所示）。

如需详细了解在设计应用时利用不同片段组合来适应不同屏幕配置，请参阅[屏幕兼容性概览](/guide/practices/screens_support) (/guide/practices/screens_support)。

创建片段

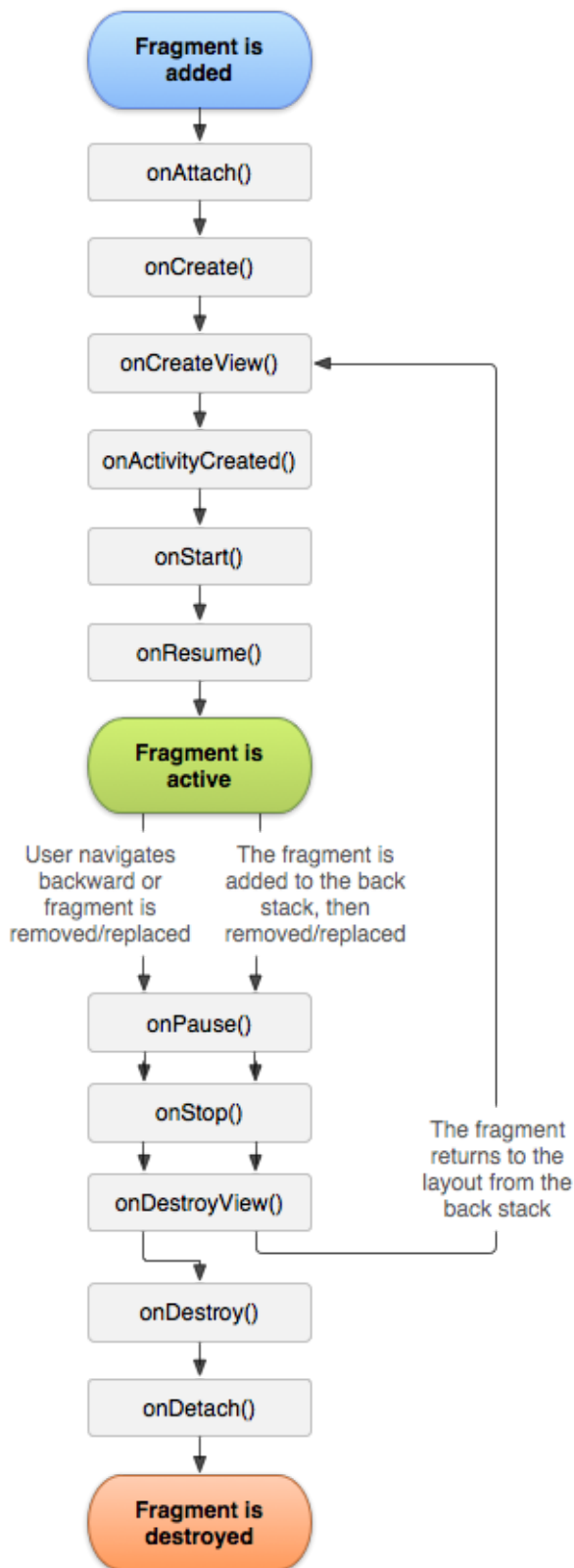


图 2. 片段的生命周期（当其 Activity 运行时）。

如要创建片段，您必须创建 `Fragment` (/reference/androidx/fragment/app/Fragment) 的子类（或已有其子类）。`Fragment` (/reference/androidx/fragment/app/Fragment) 类的代码与 `Activity` (/reference/android/app/Activity) 非常相似。它包含与 `Activity` 类似的回调方法，如 `onCreate()` (/reference/androidx/fragment/app/Fragment#onCreate(android.os.Bundle))、`onStart()` (/reference/androidx/fragment/app/Fragment#onStart())、`onPause()`。

([/reference/androidx/fragment/app/Fragment#onPause\(\)](#)) 和 [onStop\(\)](#)。

([/reference/androidx/fragment/app/Fragment#onStop\(\)](#))。实际上，如果您要将现有 Android 应用转换为使用片段，可能只需将代码从 Activity 的回调方法移入片段相应的回调方法中。

通常，您至少应实现以下生命周期方法：

[onCreate\(\)](#) ([/reference/androidx/fragment/app/Fragment#onCreate\(android.os.Bundle\)](#))

系统会在创建片段时调用此方法。当片段经历暂停或停止状态继而恢复后，如果您希望保留此片段的基本组件，则应在您的实现中将其初始化。

[onCreateView\(\)](#)

([/reference/androidx/fragment/app/Fragment#onCreateView\(android.view.LayoutInflater, android.view.ViewGroup, android.os.Bundle\)](#))

系统会在片段首次绘制其界面时调用此方法。如要为您的片段绘制界面，您从此方法中返回的 [View](#) ([/reference/android/view/View](#)) 必须是片段布局的根视图。如果片段未提供界面，您可以返回 null。

[onPause\(\)](#) ([/reference/androidx/fragment/app/Fragment#onPause\(\)](#))

系统会将此方法作为用户离开片段的第一个信号（但并不总是意味着此片段会被销毁）进行调用。通常，您应在此方法内确认在当前用户会话结束后仍然有效的任何更改（因为用户可能不会返回）。

大多数应用至少应为每个片段实现这三个方法，但您还应使用几种其他回调方法来处理片段生命周期的各个阶段。[处理片段生命周期](#) (#Lifecycle)部分对所有生命周期回调方法做了更详尽的阐述。

请注意，用于实现依赖组件生命周期的代码应放在组件本身内，而非直接放在片段回调实现中。请参阅[使用具有生命周期感知能力的组件处理生命周期](#)

([/topic/libraries/architecture/lifecycle](#))，了解如何让您的依赖组件获得生命周期感知能力。

您可能还想扩展几个子类，而非 [Fragment](#) ([/reference/androidx/fragment/app/Fragment](#)) 基类：

[DialogFragment](#) ([/reference/androidx/fragment/app/DialogFragment](#))

显示浮动对话框。使用此类创建对话框可有效代替使用 [Activity](#)

([/reference/android/app/Activity](#)) 类中的对话框辅助方法，因为您可以将片段对话框纳入由 Activity 管理的片段返回栈，从而使用户能够返回清除的片段。

[ListFragment](#) ([/reference/androidx/fragment/app/ListFragment](#))

显示由适配器（如 [SimpleCursorAdapter](#)

([/reference/android/widget/SimpleCursorAdapter](#))) 管理的一系列项目，类似于

[ListActivity](#) (/reference/android/app/ListActivity)。该类提供几种管理列表视图的方法，如用于处理点击事件的 [onListItemClick\(\)](#)

(/reference/android/app/ListFragment#onListItemClick(android.widget.ListView, android.view.View, int, long))

回调。（请注意，显示列表的首选方法是使用 RecyclerView，而非 ListView。在此情况下，您需在列表布局中创建包含 [RecyclerView](#)

(/reference/androidx/recyclerview/widget/RecyclerView) 的片段。如需了解具体操作方法，请参阅[使用 RecyclerView 创建列表](#) (/guide/topics/ui/layout/recyclerview))

[PreferenceFragmentCompat](#) (/reference/androidx/preference/PreferenceFragmentCompat)

以列表形式显示 [Preference](#) (/reference/androidx/preference/Preference) 对象的层次结构。此类用于为您的应用创建设置屏幕 (/guide/topics/ui/settings)。

添加界面

片段通常用作 Activity 界面的一部分，并且会将其自己的布局融入 Activity。

如要为片段提供布局，您必须实现 [onCreateView\(\)](#)。

(/reference/androidx/fragment/app/Fragment#onCreateView(android.view.LayoutInflater, android.view.ViewGroup, android.os.Bundle))

回调方法，**Android 系统会在片段需要绘制其布局时调用该方法**。此方法的实现所返回的 [View](#) (/reference/android/view/View) 必须是片段布局的根视图。

注意：如果您的片段是 [ListFragment](#) (/reference/androidx/fragment/app/ListFragment) 的子类，则默认实现会从 [onCreateView\(\)](#)。

(/reference/androidx/fragment/app/Fragment#onCreateView(android.view.LayoutInflater, android.view.ViewGroup, android.os.Bundle))

返回一个 [ListView](#) (/reference/android/widget/ListView)，因此您无需实现它。

如要从 [onCreateView\(\)](#)。

(/reference/androidx/fragment/app/Fragment#onCreateView(android.view.LayoutInflater, android.view.ViewGroup, android.os.Bundle))

返回布局，您可以通过 XML 中定义的[布局资源](#) (/guide/topics/resources/layout-resource)来扩展布局。为帮助您执行此操作，[onCreateView\(\)](#)。

(/reference/androidx/fragment/app/Fragment#onCreateView(android.view.LayoutInflater, android.view.ViewGroup, android.os.Bundle))

提供了一个 [LayoutInflater](#) (/reference/android/view/LayoutInflater) 对象。

例如，以下 [Fragment](#) (/reference/androidx/fragment/app/Fragment) 子类从 `example_fragment.xml` 文件加载布局：

KOTLIN (#KOTLIN-KOTLIN).JAVA

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

注意：在上例中，`R.layout.example_fragment` 是对应用资源中保存的布局资源 `example_fragment.xml` 的引用。如需了解有关如何在 XML 中创建布局的信息，请参阅[界面 \(/guide/topics/ui\)](#)文档。

传递至 [onCreateView\(.\)](#)。

([/reference/androidx/fragment/app/Fragment#onCreateView\(android.view.LayoutInflater, android.view.ViewGroup, android.os.Bundle\)](#))

的 `container` 参数是您的片段布局将插入到的父级 [ViewGroup](#)

([/reference/android/view/ViewGroup](#)) (来自 Activity 的布局)。 `savedInstanceState` 参数是在恢复片段时，提供上一片段实例相关数据的 [Bundle](#) ([/reference/android/os/Bundle](#)) ([处理片段生命周期 \(#Lifecycle\)](#)部分对恢复状态做了详细阐述)。

[inflate\(.\)](#) ([/reference/android/view/LayoutInflater#inflate\(int, android.view.ViewGroup, boolean\)](#)) 方法带有三个参数：

- 您想要扩展的布局的资源 ID。
- 将作为扩展布局父项的 [ViewGroup](#) ([/reference/android/view/ViewGroup](#))。传递 `container` 对系统向扩展布局的根视图（由其所属的父视图指定）应用布局参数具有重要意义。
- 指示是否应在扩展期间将扩展布局附加至 [ViewGroup](#) ([/reference/android/view/ViewGroup](#))（第二个参数）的布尔值。（在本例中，此值为 `false`，因为系统已将扩展布局插入 `container`，而传递 `true` 值会在最终布局中创建一个多余的视图组。）

现在，您已了解如何创建提供布局的片段。接下来，您需将该片段添加到您的 Activity 中。

向 Activity 添加片段

通常，片段会向宿主 Activity 贡献一部分界面，作为 Activity 整体视图层次结构的一部分嵌入到 Activity 中。可以通过两种方式向 Activity 布局添加片段：

- **在 Activity 的布局文件内声明片段。**

在本例中，您可以将片段当作视图来为其指定布局属性。例如，以下是拥有两个片段的 Activity 的布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
        android:id="@+id/viewer"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```

<fragment> 中的 android:name 属性指定要在布局中进行实例化的 [Fragment](#) (/reference/androidx/fragment/app/Fragment) 类。

创建此 Activity 布局时，系统会将布局中指定的每个片段实例化，并为每个片段调用 [onCreateView\(\)](#)。

(/reference/androidx/fragment/app/Fragment#onCreateView(android.view.LayoutInflater, android.view.ViewGroup, android.os.Bundle))

方法，以检索每个片段的布局。系统会直接插入 **片段返回的 View**

(/reference/android/view/View)，从而 **代替 <fragment> 元素**。

★ **注意：**每个片段都需要唯一标识符，重启 Activity 时，系统可使用该标识符来恢复片段（您也可以使用该标识符来捕获片段，从而执行某些事务，如将其移除）。可以通过两种方式为片段提供 ID：

- 为 **android:id** 属性提供 **唯一 ID**。
- 为 **android:tag** 属性提供 **唯一字符串**。

- **或者，通过编程方式将片段添加到某个现有 [ViewGroup](#)** (/reference/android/view/ViewGroup)。

在 Activity 运行期间，您可以随时将片段添加到 Activity 布局中。您只需指定要将片段放入哪个 [ViewGroup](/reference/android/view/ViewGroup) (/reference/android/view/ViewGroup)。

如要在您的 Activity 中执行片段事务（如添加、移除或替换片段），则必须使用 [FragmentManager](/reference/androidx/fragment/app/FragmentManager) (/reference/androidx/fragment/app/FragmentManager) 中的 API。如下所示，您可以从 [FragmentManager](/reference/androidx/fragment/app/FragmentManager) (/reference/androidx/fragment/app/FragmentManager) 获取一个 [FragmentManager](/reference/androidx/fragment/app/FragmentManager) (/reference/androidx/fragment/app/FragmentManager) 实例：

KOTLIN (#KOTLIN-KOTLIN)**JAVA**

```
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentManager fragmentManager = fragmentManager.beginTransaction();
```

然后，您可以使用 [add\(\)](/reference/androidx/fragment/app/FragmentManager) (/reference/androidx/fragment/app/FragmentManager#add(int, android.support.v4.app.Fragment))

方法添加一个片段，指定要添加的片段以及将其插入哪个视图。例如：

KOTLIN (#KOTLIN-KOTLIN)**JAVA**

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

传递到 [add\(\)](/reference/androidx/fragment/app/FragmentManager) (/reference/androidx/fragment/app/FragmentManager#add(int, android.support.v4.app.Fragment))

的第一个参数是 [ViewGroup](/reference/android/view/ViewGroup) (/reference/android/view/ViewGroup)，即应放置片段的位置，由资源 ID 指定，第二个参数是要添加的片段。

一旦您通过 [FragmentManager](/reference/androidx/fragment/app/FragmentManager) (/reference/androidx/fragment/app/FragmentManager) 做出了更改，就必须调用 [commit\(\)](/reference/androidx/fragment/app/FragmentManager) (/reference/androidx/fragment/app/FragmentManager#commit()) 以使更改生效。

管理片段

如要管理 Activity 中的片段，您需使用 [FragmentManager](#) ([/reference/androidx/fragment/app/FragmentManager](#))。如要获取它，请从您的 Activity 调用 [getSupportFragmentManager\(\)](#) ([/reference/androidx/fragment/app/FragmentManager#getSupportFragmentManager\(\)](#))。

可使用 [FragmentManager](#) ([/reference/androidx/fragment/app/FragmentManager](#)) 执行的操作包括：

- 通过 [findFragmentById\(\)](#) ([/reference/androidx/fragment/app/FragmentManager#findFragmentById\(int\)](#))（针对在 Activity 布局中提供界面的片段）或 [findFragmentByTag\(\)](#) ([/reference/androidx/fragment/app/FragmentManager#findFragmentByTag\(java.lang.String\)](#))（针对提供或不提供界面的片段）获取 Activity 中存在的片段。
- 通过 [popBackStack\(\)](#) ([/reference/androidx/fragment/app/FragmentManager#popBackStack\(\)](#))（模拟用户发出的返回命令）使片段从返回栈中弹出。
- 通过 [addOnBackStackChangeListener\(\)](#) ([/reference/androidx/fragment/app/FragmentManager#addOnBackStackChangeListener\(android.support.v4.app.FragmentManager.OnBackStackChangeListener\)](#)) 注册侦听返回栈变化的侦听器。

如需了解有关这些方法以及其他方法的详细信息，请参阅 [FragmentManager](#) ([/reference/androidx/fragment/app/FragmentManager](#)) 类文档。

如上文所述，您也可使用 [FragmentManager](#) ([/reference/androidx/fragment/app/FragmentManager](#)) 打开一个 [FragmentTransaction](#) ([/reference/androidx/fragment/app/FragmentTransaction](#))，通过它来执行某些事务，如添加和移除片段。

执行片段事务

在 Activity 中使用片段的一大优点是，您可以通过片段执行添加、移除、替换以及其他操作，从而响应用户交互。提交给 Activity 的每组更改均称为事务，并且您可使用 [FragmentTransaction](#) ([/reference/androidx/fragment/app/FragmentTransaction](#)) 中的 API 来执行一项事务。您也可将每个事务保存到由 Activity 管理的返回栈内，从而让用户能够回退片段更改（类似于回退 Activity）。

如下所示，您可以从 [FragmentManager](#) ([/reference/androidx/fragment/app/FragmentManager](#)) 获取一个 [FragmentTransaction](#) ([/reference/androidx/fragment/app/FragmentTransaction](#)) 实例：

KOTLIN (#KOTLIN-KOTLIN) JAVA

```
FragmentManager fragmentManager = getSupportFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

每个事务都是您想要同时执行的一组更改。您可以使用 [add\(\)](#)

([/reference/androidx/fragment/app/FragmentTransaction#add\(android.support.v4.app.Fragment, java.lang.String\)](#))

、[remove\(\)](#)

([/reference/androidx/fragment/app/FragmentTransaction#remove\(android.support.v4.app.Fragment\)](#))

和 [replace\(\)](#)

([/reference/androidx/fragment/app/FragmentTransaction#replace\(int, android.support.v4.app.Fragment\)](#))

等方法，为给定事务设置您想要执行的所有更改。然后，如要**将事务应用到 Activity**，您**必须调用 [commit\(\)](#)**([/reference/androidx/fragment/app/FragmentTransaction#commit\(\)](#))。

不过，在调用 [commit\(\)](#)([/reference/androidx/fragment/app/FragmentTransaction#commit\(\)](#)) 之前，您可能希望调用 [addToBackStack\(\)](#)

([/reference/androidx/fragment/app/FragmentTransaction#addToBackStack\(java.lang.String\)](#))，以将事务添加到片段事务返回栈。该返回栈由 Activity 管理，允许用户通过按**返回**按钮返回上一片段状态。

例如，以下示例说明如何将一个片段替换为另一个片段，以及如何在返回栈中保留先前的状态：

KOTLIN (#KOTLIN-KOTLIN) JAVA

```
// Create new fragment and transaction
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getSupportFragmentManager().beginTransaction()

// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

在本例中，newFragment 会替换目前在 R.id.fragment_container ID 所标识的布局容器中的任何片段（如有）。通过调用 [addToBackStack\(\)](#)

(/reference/androidx/fragment/app/FragmentManager#addToBackStack(java.lang.String)), 您可以将替换事务保存到返回栈, 以使用户能够通过 **按返回按钮撤消事务** 并回退到上一片段。

然后, FragmentManager (/reference/androidx/fragment/app/FragmentManager) 会自动通过 onBackPressed() (/reference/android/app/Activity#onBackPressed()) 从返回栈检索片段。

如果您向事务添加多个更改 (如又一个 add()。

(/reference/androidx/fragment/app/FragmentManager#add(android.support.v4.app.Fragment, java.lang.String))

或 remove()。

(/reference/androidx/fragment/app/FragmentManager#remove(android.support.v4.app.Fragment)) , 并调用 addToBackStack()。

(/reference/androidx/fragment/app/FragmentManager#addToBackStack(java.lang.String)), 则调用 commit() (/reference/androidx/fragment/app/FragmentManager#commit()) 前应用的所有更改都将作为单一事务添加到返回栈, 并且 **返回按钮** 会将它们一并撤消。

向 FragmentManager (/reference/androidx/fragment/app/FragmentManager) 添加更改的顺序无关紧要, 不过:

- 您必须最后调用 commit()。
(/reference/androidx/fragment/app/FragmentManager#commit())。
- 如果您要向同一容器添加多个片段, 则您添加片段的顺序将决定它们在视图层次结构中出现的顺序。

如果您**没有在执行删除片段的事务时调用 addToBackStack()**

(/reference/androidx/fragment/app/FragmentManager#addToBackStack(java.lang.String)), 则事务提交时**该片段会被销毁**, 用户将无法回退到该片段。不过, 如果您在删除片段时调用 addToBackStack()。



(/reference/androidx/fragment/app/FragmentManager#addToBackStack(java.lang.String)), 则**系统会停止该片段**, 并随后在用户回退时将其恢复。

提示: 对于每个片段事务, 您都可通过在提交前调用 setTransition()。

(/reference/androidx/fragment/app/FragmentManager#setTransition(int)) 来应用过渡动画。

调用 commit() (/reference/androidx/fragment/app/FragmentManager#commit()) 不会立即执行事务, 而是在 Activity 的界面线程 (“主”线程) 可执行该操作时, 再安排该事务在线程上运行。不过, 如有必要, 您也可以从界面线程调用 executePendingTransactions()。

(/reference/androidx/fragment/app/FragmentManager#executePendingTransactions()), 以立即执行 commit() (/reference/androidx/fragment/app/FragmentManager#commit()) 提交的事务。通常不必这样做, 除非其他线程中的作业依赖该事务。

注意：您只能在 Activity 保存其状态 (/guide/components/activities#SavingActivityState) (当用户离开 Activity) 之前使用 commit() (/reference/androidx/fragment/app/FragmentManager#commit()) 提交事务。如果您试图在该时间点后提交，则会引发异常。这是因为如需恢复 Activity，则提交后的状态可能会丢失。对于丢失提交无关紧要的情况，请使用 commitAllowingStateLoss() (/reference/androidx/fragment/app/FragmentManager#commitAllowingStateLoss())。

与 Activity 通信

尽管 Fragment (/reference/androidx/fragment/app/Fragment) 作为独立于 FragmentActivity (/reference/androidx/fragment/app/FragmentActivity) 的对象实现，并且可在多个 Activity 内使用，但片段的给定实例会直接绑定到托管该片段的 Activity。

具体而言，**片段可通过 `getActivity()`**

(/reference/androidx/fragment/app/Fragment#getActivity()) **访问 `FragmentActivity`**

(/reference/androidx/fragment/app/FragmentActivity) 实例，并轻松执行在 Activity 布局中查找视图等任务：

KOTLIN (#KOTLIN-KOTLIN)**JAVA**

```
View listView = getActivity().findViewById (/reference/android/app/Activity#findViewById
```

同样，您的 Activity 也可使用 `findFragmentById()`

(/reference/androidx/fragment/app/FragmentManager#findFragmentById(int)) 或

`findFragmentByTag()`

(/reference/androidx/fragment/app/FragmentManager#findFragmentByTag(java.lang.String))，通过从

FragmentManager (/reference/androidx/fragment/app/FragmentManager) 获取对 Fragment

(/reference/androidx/fragment/app/Fragment) 的引用来调用片段中的方法。例如：

KOTLIN (#KOTLIN-KOTLIN)**JAVA**

```
ExampleFragment fragment = (ExampleFragment) getSupportFragmentManager().find
```

创建 Activity 的事件回调

在某些情况下，您可能需使用片段来与 Activity 和/或 Activity 托管的其他片段共享事件或数据。**如要共享数据**，请依照 [ViewModel 指南](/topic/libraries/architecture/viewmodel) (/topic/libraries/architecture/viewmodel) 中“在片段之间共享数据”部分所述，**创建共享的 ViewModel**。如需传播无法使用 ViewModel 处理的事件，则可改为**在片段内定义回调接口**，并要求**宿主 Activity 实现此接口**。当 **Activity 通过该接口收到回调**时，可根据需要**与布局中的其他片段共享这些信息**。

例如，如果某个新闻应用的 Activity 有两个片段，其中一个用于显示文章列表（片段 A），另一个用于显示文章（片段 B），则片段 A 必须在列表项被选定后告知 Activity，以便它告知片段 B 显示该文章。在本例中，OnArticleSelectedListener 接口在片段 A 内进行声明：

KOTLIN (#KOTLIN-KOTLIN)**JAVA**

```
public static class FragmentA extends ListFragment {
    ...
    // Container Activity must implement this interface
    public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
    }
    ...
}
```

然后，该片段的宿主 Activity 会实现 OnArticleSelectedListener 接口并重写 onArticleSelected()，将来自片段 A 的事件通知片段 B。为确保宿主 Activity 实现此接口，片段 A 的 [onAttach\(\)](#)。

([/reference/androidx/fragment/app/Fragment#onAttach\(android.content.Context\)](/reference/androidx/fragment/app/Fragment#onAttach(android.content.Context))) 回调方法（系统在向 Activity 添加片段时调用的方法）会通过转换传递到 [onAttach\(\)](#)。

([/reference/androidx/fragment/app/Fragment#onAttach\(android.content.Context\)](/reference/androidx/fragment/app/Fragment#onAttach(android.content.Context))) 中的 [Activity](#) (</reference/android/app/Activity>) 来实例化 OnArticleSelectedListener 的实例：

KOTLIN (#KOTLIN-KOTLIN)**JAVA**

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener listener;
    ...
    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        try {
            listener = (OnArticleSelectedListener) context;
        } catch (ClassCastException e) {
            throw new ClassCastException(context.toString() + " must implemen
        }
    }
}
```

```

    }
    ...
}

```

如果 Activity 未实现接口，则片段会抛出 [ClassCastException](#)

([/reference/java/lang/ClassCastException](#))。若实现成功，mListener 成员会保留对 Activity 的 [OnArticleSelectedListener](#) 实现的引用，以便片段 A 可通过调用 [OnArticleSelectedListener](#) 接口定义的方法与 Activity 共享事件。例如，如果片段 A 是 [ListFragment](#) ([/reference/androidx/fragment/app/ListFragment](#)) 的一个扩展，则用户每次点击列表项时，系统都会调用片段中的 [onListItemClick\(\)](#)

([/reference/androidx/fragment/app/ListFragment#onListItemClick\(android.widget.ListView, android.view.View, int, long\)](#))

，然后该方法会通过调用 [onArticleSelected\(\)](#) 与 Activity 共享事件：

KOTLIN (#KOTLIN-KOTLIN)**JAVA**

```

public static class FragmentA extends ListFragment {
    OnArticleSelectedListener listener;
    ...
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Append the clicked item's row ID with the content provider Uri
        Uri noteUri = ContentUris.withAppendedId (/reference/android/content/ContentU
        // Send the event and Uri to the host activity
        listener.onArticleSelected(noteUri);
    }
    ...
}

```

传递到 [onListItemClick\(\)](#)

([/reference/androidx/fragment/app/ListFragment#onListItemClick\(android.widget.ListView, android.view.View, int, long\)](#))

的 id 参数是被点击项的行 ID，即 Activity（或其他片段）用来从应用的 [ContentProvider](#) ([/reference/android/content/ContentProvider](#)) 获取文章的 ID。

如需了解关于使用内容提供程序的详细信息，请参阅[内容提供程序](#) ([/guide/topics/providers/content-providers](#))文档。

向应用栏添加项目

您的片段可通过实现 `onCreateOptionsMenu()`。

([/reference/androidx/fragment/app/Fragment#onCreateOptionsMenu\(android.view.Menu, android.view.MenuInflater\)](/reference/androidx/fragment/app/Fragment#onCreateOptionsMenu(android.view.Menu, android.view.MenuInflater)))

向 Activity 的[选项菜单](/guide/topics/ui/menus#options-menu) (并因此向[应用栏](/training/appbar) (</training/appbar>)) 贡献菜单项。不过，为使此方法能够收到调用，您必须在 `onCreate()` ([/reference/androidx/fragment/app/Fragment#onCreate\(android.os.Bundle\)](/reference/androidx/fragment/app/Fragment#onCreate(android.os.Bundle))) 期间调用 `setHasOptionsMenu()`。

([/reference/androidx/fragment/app/Fragment#setHasOptionsMenu\(boolean\)](/reference/androidx/fragment/app/Fragment#setHasOptionsMenu(boolean)))，以指示片段想要向选项菜单添加菜单项。否则，片段不会收到对 `onCreateOptionsMenu()`。

([/reference/androidx/fragment/app/Fragment#onCreateOptionsMenu\(android.view.Menu, android.view.MenuInflater\)](/reference/androidx/fragment/app/Fragment#onCreateOptionsMenu(android.view.Menu, android.view.MenuInflater)))

的调用。

您之后从片段添加到选项菜单的任何菜单项都将追加到现有菜单项之后。选定菜单项时，片段还会收到对 `onOptionsItemSelected()`。

([/reference/androidx/fragment/app/Fragment#onOptionsItemSelected\(android.view.MenuItem\)](/reference/androidx/fragment/app/Fragment#onOptionsItemSelected(android.view.MenuItem))) 的回调。

您还可通过调用 `registerForContextMenu()`。

([/reference/androidx/fragment/app/Fragment#registerForContextMenu\(android.view.View\)](/reference/androidx/fragment/app/Fragment#registerForContextMenu(android.view.View)))，在片段布局中注册一个视图来提供上下文菜单。当用户打开上下文菜单时，片段会收到对 `onCreateContextMenu()`。

([/reference/androidx/fragment/app/Fragment#onCreateContextMenu\(android.view.ContextMenu, android.view.View, android.view.ContextMenu.ContextMenuInfo\)](/reference/androidx/fragment/app/Fragment#onCreateContextMenu(android.view.ContextMenu, android.view.View, android.view.ContextMenu.ContextMenuInfo)))

的调用。当用户选择某个菜单项时，片段会收到对 `onContextItemSelected()`。

([/reference/androidx/fragment/app/Fragment#onContextItemSelected\(android.view.MenuItem\)](/reference/androidx/fragment/app/Fragment#onContextItemSelected(android.view.MenuItem))) 的调用。

注意：尽管您的片段会收到与其添加的每个菜单项对应的 on-item-selected 回调，但当用户选择菜单项时，Activity 会首先收到相应的回调。如果 Activity 对 on-item-selected 回调的实现不处理选定的菜单项，则系统会将事件传递至片段的回调。这适用于选项菜单和上下文菜单。

如需了解有关菜单的详细信息，请参阅[菜单](/guide/topics/ui/menus) (</guide/topics/ui/menus>) 开发者指南和[应用栏](/training/appbar) (</training/appbar>) 培训课程。

处理片段生命周期

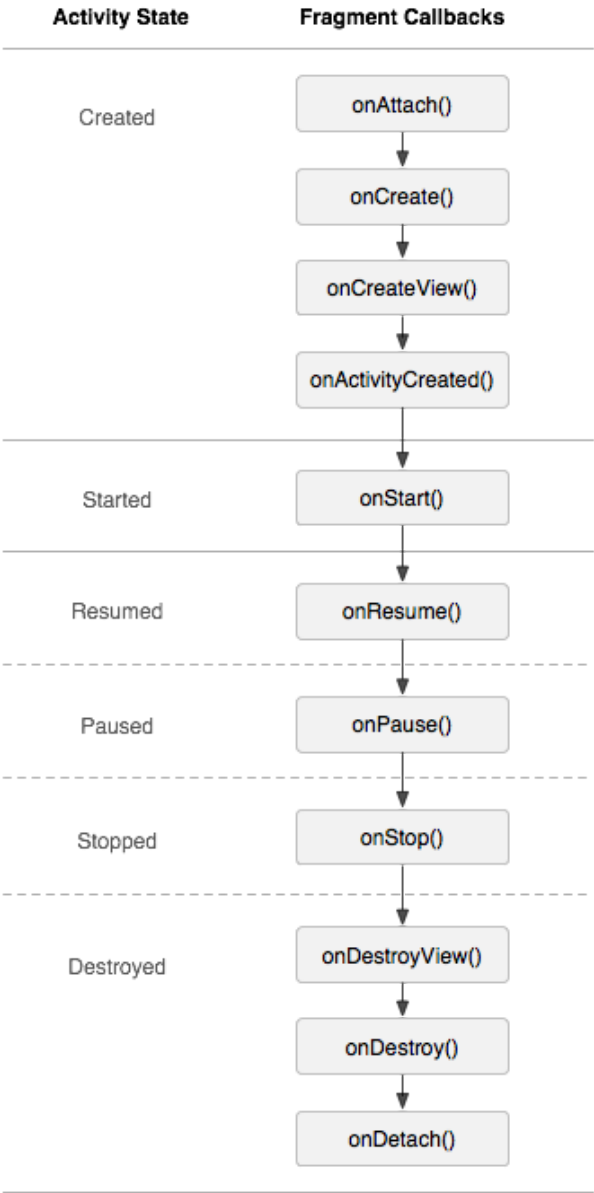


图 3. Activity 生命周期对片段生命周期的影响。

管理片段生命周期与管理 Activity 生命周期很相似。和 Activity 一样，片段也以三种状态存在：

已恢复

片段在运行中的 Activity 中可见。

已暂停

另一个 Activity 位于前台并具有焦点，但此片段所在的 Activity 仍然可见（前台 Activity 部分透明，或未覆盖整个屏幕）。

已停止

片段不可见。宿主 Activity 已停止，或片段已从 Activity 中移除，但已添加到返回栈。已停止的片段仍处于活动状态（系统会保留所有状态和成员信息）。不过，它对用户不

再可见，并随 Activity 的终止而终止。

与 Activity 一样，您也可使用 `onSaveInstanceState(Bundle)`。

([/reference/androidx/fragment/app/Fragment#onSaveInstanceState\(android.os.Bundle\)\)](/reference/androidx/fragment/app/Fragment#onSaveInstanceState(android.os.Bundle)))、[ViewModel](/reference/androidx/lifecycle/ViewModel) (</reference/androidx/lifecycle/ViewModel>) 和持久化本地存储的组合，在配置变更和进程终止后保留片段的界面状态。如要了解保留界面状态的更多信息，请参阅[保存界面状态](/topic/libraries/architecture/saving-states) (</topic/libraries/architecture/saving-states>)。

对于 Activity 生命周期与片段生命周期而言，二者最显著的差异是在其各自返回栈中的存储方式。默认情况下，Activity 停止时会被放入由系统管理的 Activity 返回栈中（以使用户通过返回按钮回退到 Activity，详细介绍请参阅[任务和返回栈](/guide/components/tasks-and-back-stack) (</guide/components/tasks-and-back-stack>)）。不过，只有当您


在移除片段的事务执行期间通过调用 `addToBackStack()`。

([/reference/androidx/fragment/app/FragmentTransaction#addToBackStack\(java.lang.String\)\)](/reference/androidx/fragment/app/FragmentTransaction#addToBackStack(java.lang.String))) 显式请求保存实例时，系统才会将片段放入由宿主 Activity 管理的返回栈。

在其他方面，管理片段生命周期与管理 Activity 生命周期非常相似；对此，您可采取相同的做法。请参阅 [Activity 生命周期](/guide/components/activities/activity-lifecycle) (</guide/components/activities/activity-lifecycle>) 指南和[使用具有生命周期感知能力的组件处理生命周期](/topic/libraries/architecture/lifecycle) (</topic/libraries/architecture/lifecycle>)，了解有关 Activity 生命周期及其管理措施的详情。

注意：如果您的 [Fragment](/reference/androidx/fragment/app/Fragment) (</reference/androidx/fragment/app/Fragment>) 中需要 [Context](/reference/android/content/Context)

(</reference/android/content/Context>) 对象，则可以调用 `getContext()`。

([/reference/androidx/fragment/app/Fragment#getContext\(\)](/reference/androidx/fragment/app/Fragment#getContext()))。但请注意， 只有在片段附加到 Activity 时才需调用 `getContext()` ([/reference/androidx/fragment/app/Fragment#getContext\(\)](/reference/androidx/fragment/app/Fragment#getContext()))。如果尚未附加该片段，或者其在生命周期结束期间已分离，则 `getContext()`。

([/reference/androidx/fragment/app/Fragment#getContext\(\)](/reference/androidx/fragment/app/Fragment#getContext())) 返回 null。

与 Activity 生命周期协调一致

片段所在 Activity 的生命周期会直接影响片段的生命周期，其表现为，Activity 的每次生命周期回调都会引发每个片段的类似回调。例如，当 Activity 收到 `onPause()`。

([/reference/android/app/Activity#onPause\(\)](/reference/android/app/Activity#onPause())) 时，Activity 中的每个片段也会收到 `onPause()`。

([/reference/androidx/fragment/app/Fragment#onPause\(\)](/reference/androidx/fragment/app/Fragment#onPause()))。

不过，片段还有几个额外的生命周期回调，用于处理与 Activity 的唯一交互，从而执行构建和销毁片段界面等操作。这些额外的回调方法是：

`onAttach()` ([/reference/androidx/fragment/app/Fragment#onAttach\(android.content.Context\)](/reference/androidx/fragment/app/Fragment#onAttach(android.content.Context))))

在片段已与 Activity 关联时进行调用 ([Activity](#) (/reference/android/app/Activity) 传递到此方法内)。

[onCreateView\(\)](#)

(/reference/androidx/fragment/app/Fragment#onCreateView(android.view.LayoutInflater, android.view.ViewGroup, android.os.Bundle))

调用它可创建与片段关联的视图层次结构。

[onActivityCreated\(\)](#)

(/reference/androidx/fragment/app/Fragment#onActivityCreated(android.os.Bundle))

当 Activity 的 [onCreate\(\)](#) (/reference/android/app/Activity#onCreate(android.os.Bundle)) 方法已返回时进行调用。

[onDestroyView\(\)](#) (/reference/androidx/fragment/app/Fragment#onDestroyView())

在移除与片段关联的视图层次结构时进行调用。

[onDetach\(\)](#) (/reference/androidx/fragment/app/Fragment#onDetach())

在取消片段与 Activity 的关联时进行调用。

图 3 所示为受宿主 Activity 影响的片段生命周期流。在该图中，您可以看到 Activity 的每个连续状态如何确定片段可收到的回调方法。例如，当 Activity 收到其 [onCreate\(\)](#)

(/reference/android/app/Activity#onCreate(android.os.Bundle)) 回调时，Activity 中的片段只会收到 [onActivityCreated\(\)](#)

(/reference/androidx/fragment/app/Fragment#onActivityCreated(android.os.Bundle)) 回调。

一旦 Activity 达到已恢复状态，您便可随意向 Activity 添加片段和移除其中的片段。因此，**只有当 Activity 处于已恢复状态时，片段的生命周期才能独立变化。**

不过，当 Activity 离开已恢复状态时，片段会在 Activity 的推动下再次经历其生命周期。

示例

为将本文阐述的所有内容融会贯通，以下提供了一个示例，其中的 Activity 使用两个片段来创建一个双窗格布局。下面的 Activity 包括两个片段：一个用于显示莎士比亚戏剧标题列表，另一个用于在从列表中选定戏剧时显示其摘要。此外，它还展示了如何根据屏幕配置提供不同的片段配置。

注意：有关此 Activity 的完整源代码，请参见[示例应用](#)

(https://github.com/aosp-mirror/platform_development/blob/master/samples/ApiDemos/src/com/example/android/apis/app/FragmentManagerLayout.java)

，该应用展示了示例 `FragmentManager` 类的用法。

主 Activity 在 `onCreate()` ([/reference/android/app/Activity#onCreate\(android.os.Bundle\)](#)) 期间以常规方式应用布局：

KOTLIN (#KOTLIN-KOTLIN)**JAVA**

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.fragment_layout);
}
```

应用的布局为 `fragment_layout.xml`：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent" android:layout_height="match_parent">

    <fragment class="com.example.android.apis.app.FragmentManager$TitlesFragment"
        android:id="@+id/titles" android:layout_weight="1"
        android:layout_width="0px" android:layout_height="match_parent" />

    <FrameLayout android:id="@+id/details" android:layout_weight="1"
        android:layout_width="0px" android:layout_height="match_parent"
        android:background="?android:attr/detailsElementBackground" />

</LinearLayout>
```

通过使用此布局，系统会在 Activity 加载布局时立即实例化 `TitlesFragment`（其列出了戏剧标题），而 [FrameLayout](#) ([/reference/android/widget/FrameLayout](#))（用于显示戏剧摘要的片段所在位置）则会占用屏幕右侧的空间，但最初处于空白状态。如下文所示，只有当用户从列表中选择某个项目后，系统才会将片段放入 [FrameLayout](#) ([/reference/android/widget/FrameLayout](#))。

不过，并非所有屏幕配置都具有足够的宽度，可以一同显示戏剧列表和摘要。因此，以上布局仅用于横向屏幕配置（布局保存在 `res/layout-land/fragment_layout.xml` 中）。

因此，当屏幕纵向显示时，系统会应用以下布局（保存在 `res/layout/fragment_layout.xml` 中）：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <fragment class="com.example.android.apis.app.FragmentLayout$TitlesFragment"
        android:id="@+id/titles"
        android:layout_width="match_parent" android:layout_height="match_
</FrameLayout>
```

此布局仅包括 `TitlesFragment`。这意味着，当设备纵向显示时，用户只能看到戏剧标题列表。因此，当用户在此配置中点击某个列表项时，应用会启动一个新 `Activity` 来显示摘要，而非加载另一个片段。

接下来，您将了解如何在片段类中实现此目的。第一个片段是 `TitlesFragment`，用于显示莎士比亚戏剧标题列表。该片段扩展了 `ListFragment` (</reference/android/app/ListFragment>)，并依靠它来处理大多数列表视图工作。

当您检查此代码时，请注意，用户点击列表项时可能会出现两种行为：系统可能会创建并显示一个新片段，从而在同一 `Activity` 中显示详细信息（将片段添加到 `FrameLayout` (</reference/android/widget/FrameLayout>)），也可能会启动一个新 `Activity`（在该 `Activity` 中可显示片段），具体取决于这两个布局中哪一个处于活动状态。

KOTLIN (#KOTLIN-KOTLIN)**JAVA**

```
public static class TitlesFragment extends ListFragment {
    boolean dualPane;
    int curCheckPosition = 0;

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        // Populate list with our static array of titles.
        setListAdapter(new ArrayAdapter<String>(getActivity(),
            android.R.layout.simple_list_item_activated_1, Shakespeare.TI

        // Check to see if we have a frame in which to embed the details
        // fragment directly in the containing UI.
```



```
View detailsFrame = getActivity().findViewById(R.id.details);
dualPane = detailsFrame != null && detailsFrame.getVisibility() == Vi

if (savedInstanceState != null) {
    // Restore last state for checked position.
    curCheckPosition = savedInstanceState.getInt("curChoice", 0);
}

if (dualPane) {
    // In dual-pane mode, the list view highlights the selected item.
    getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
    // Make sure our UI is in the correct state.
    showDetails(curCheckPosition);
}
}

@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putInt("curChoice", curCheckPosition);
}

@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    showDetails(position);
}

/**
 * Helper function to show the details of a selected item, either by
 * displaying a fragment in-place in the current UI, or starting a
 * whole new activity in which it is displayed.
 */
void showDetails(int index) {
    curCheckPosition = index;

    if (dualPane) {
        // We can display everything in-place with fragments, so update
        // the list to highlight the selected item and show the data.
        getListView().setItemChecked(index, true);

        // Check what fragment is currently shown, replace if needed.
        DetailsFragment details = (DetailsFragment)
            getSupportFragmentManager().findFragmentById(R.id.details);
        if (details == null || details.getShownIndex() != index) {
            // Make new fragment to show this selection.
            details = DetailsFragment.newInstance(index);

            // Execute a transaction, replacing any existing fragment
```

```

        // with this one inside the frame.
        FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
        if (index == 0) {
            ft.replace(R.id.details, details);
        } else {
            ft.replace(R.id.a_item, details);
        }
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
    }

} else {
    // Otherwise we need to launch a new activity to display
    // the dialog fragment with selected text.
    Intent intent = new Intent();
    intent.setClass(getActivity(), DetailsActivity.class);
    intent.putExtra("index", index);
    startActivity(intent);
}
}
}

```

第二个片段 DetailsFragment 显示从 TitlesFragment 的列表中选择的项目的戏剧摘要：

KOTLIN (#KOTLIN-KOTLIN)**JAVA**

```

public static class DetailsFragment extends Fragment {
    /**
     * Create a new instance of DetailsFragment, initialized to
     * show the text at 'index'.
     */
    public static DetailsFragment newInstance(int index) {
        DetailsFragment f = new DetailsFragment();

        // Supply index input as an argument.
        Bundle args = new Bundle();
        args.putInt("index", index);
        f.setArguments(args);

        return f;
    }

    public int getShownIndex() {
        return getArguments().getInt("index", 0);
    }
}

```

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    if (container == null) {
        // We have different layouts, and in one of them this
        // fragment's containing frame doesn't exist. The fragment
        // may still be created from its saved state, but there is
        // no reason to try to create its view hierarchy because it
        // isn't displayed. Note this isn't needed -- we could just
        // run the code below, where we would create and return the
        // view hierarchy; it would just never be used.
        return null;
    }

    ScrollView scroller = new ScrollView(getActivity());
    TextView text = new TextView(getActivity());
    int padding = (int)TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_
        4, getActivity().getResources().getDisplayMetrics());
    text.setPadding(padding, padding, padding, padding);
    scroller.addView(text);
    text.setText(Shakespeare.DIALOGUE[getShownIndex()]);
    return scroller;
}
}

```

从 `TitlesFragment` 类中重新调用，如果用户点击某个列表项，且当前布局不包括 `R.id.details` 视图（即 `DetailsFragment` 所属视图），则应用会启动 `DetailsActivity` 以显示该项目的内容。

以下是 `DetailsActivity`，它通过简单嵌入 `DetailsFragment`，在屏幕为纵向时显示所选的戏剧摘要：

KOTLIN (#KOTLIN-KOTLIN)JAVA

```

public static class DetailsActivity extends FragmentActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (getResources().getConfiguration().orientation
            == Configuration.ORIENTATION_LANDSCAPE) {
            // If the screen is now in landscape mode, we can show the
            // dialog in-line with the list so we don't need this activity.

```

```
        finish();
        return;
    }

    if (savedInstanceState == null) {
        // During initial setup, plug in the details fragment.
        DetailsFragment details = new DetailsFragment();
        details.setArguments(getIntent().getExtras());
        getSupportFragmentManager().beginTransaction().add(android.R.id.c
    }
}
}
```

请注意，如果配置为横向，则此 Activity 会自行完成，这样主 Activity 便可接受并显示 TitlesFragment 和 DetailsFragment。如果用户在纵向模式下启动 DetailsActivity，但随后将设备旋转为横向（这会重启当前 Activity），则可能会出现这种情况。

其他资源

[Sunflower](https://github.com/googlesamples/android-sunflower) (<https://github.com/googlesamples/android-sunflower>) 演示应用中使用了 Fragment。

Content and code samples on this page are subject to the licenses described in the [Content License](#) (/license).
Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2019-12-27.