片段

本文内容

关键类

- `Fragment`
- `FragmentManager`
- `FragmentTransaction`

另请参阅

- 利用片段构建动态 UI
- 支持平板电脑和手机

`Fragment` 表示 `Activity` 中的行为或用户界面部分。您可以将多个片段组合在一个 Activity 中来构建多窗格 UI，以及在多个 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（有点像您可以在不同 Activity 中重复使用的“子 Activity”）。

片段必须始终嵌入在 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，其中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）时，您可以独立操纵每个片段，如添加或移除它们。 当您执行此类片段事务时，您也可以将其添加到由 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。 返回栈让用户可以通过按*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，它存在于 Activity 视图层次结构的某个 `ViewGroup` 内部，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 布局中，或者通过将其添加到某个现有 `ViewGroup`，利用应用代码进行插入。不过，片段并非必须成为 Activity 布局的一部分；您还可以将没有自己 UI 的片段用作 Activity 的不可见工作线程。

本文描述如何在开发您的应用时使用片段，包括将片段添加到 Activity 返回栈时如何保持其状态、如何与 Activity 及 Activity 中的其他片段共享事件、如何为 Activity 的操作栏发挥作用等等。

设计原理

Android 在 Android 3.0（API 级别 11）中引入了片段，主要是为了给大屏幕（如平板电脑）上更加动态和灵活的 UI 设计提供支持。由于平板电脑的屏幕比手机屏幕大得多，因此可用于组合和交换 UI 组件的空间更大。利用片段实现此类设计时，您无需管理对视图层次结构的复杂更改。 通过将 Activity 布局分成片段，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。

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，每个片段都具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，用户不需要使用一个 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。

您应该将每个片段都设计为可重复使用的模块化 Activity 组件。也就是说，由于每个片段都会通过各自的生命周期回调来定义其自己的布局和行为，您可以将一个片段加入多个 Activity，因此，您应该采用可复用式设计，避免直接从某个片段直接操纵另一个片段。 这特别重要，因为模块化片段让您可以通过更改片段的组合方式来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，可能必须利用单独片段来实现单窗格 UI。

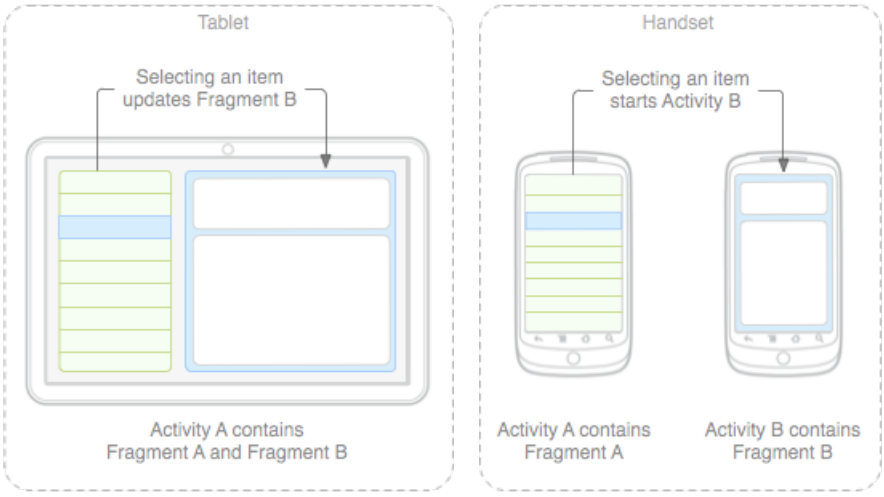

图 1. 有关由片段定义的两个 UI 模块如何适应不同设计的示例：通过组合成一个 Activity 来适应平板电脑设计，通过单独片段来适应手机设计。

例如 — 仍然以新闻应用为例 — 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，在手机尺寸的屏幕上，没有足以储存两个片段的空间，因此*Activity A* 只包括用于显示文章列表的片段，当用户选择文章时，它会启动*Activity B*，其中包括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。

如需了解有关通过利用不同片段组合来适应不同屏幕配置这种方法设计应用的详细信息，请参阅支持平板电脑和手机指南。

创建片段

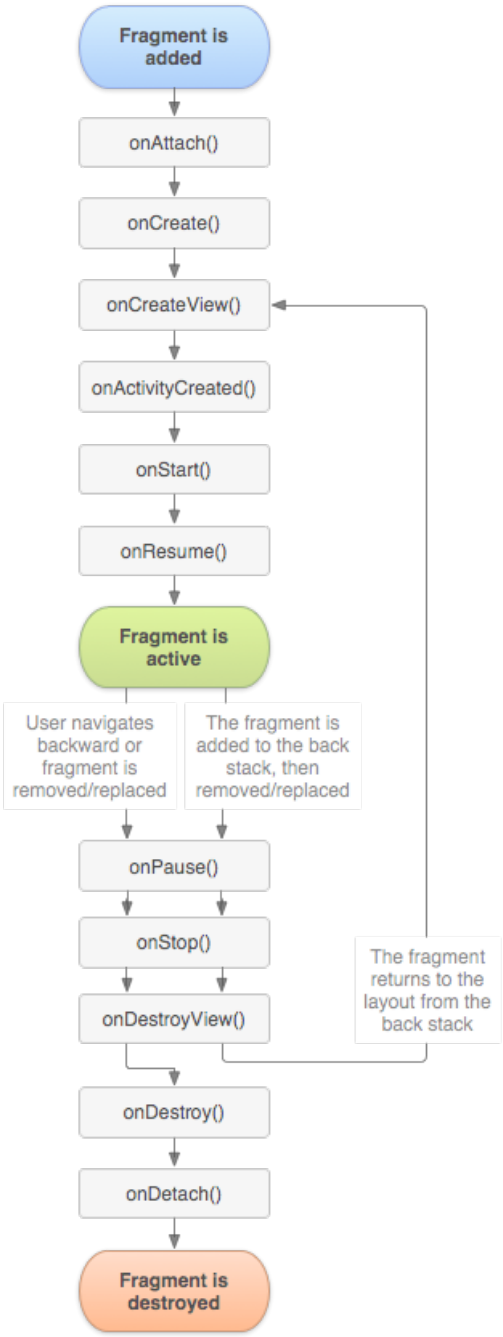

图 2. 片段的生命周期（其 Activity 运行时）。

要想创建片段，您必须创建 `Fragment` 的子类（或已有其子类）。`Fragment` 类的代码与 `Activity` 非常相似。它包含与 Activity 类似的回调方法，如 `onCreate()`、`onStart()`、`onPause()` 和 `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()`

系统会在创建片段时调用此方法。您应该在实现内初始化您想在片段暂停或停止后恢复时保留的必需片段组件。

`onCreateView()`

系统会在片段首次绘制其用户界面时调用此方法。 要想为您的片段绘制 UI，您从此方法中返回的 `View` 必须是片段布局的根视图。如果片段未提供 UI，您可以返回 null。

`onPause()`

系统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。 您通常应该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）。

大多数应用都应该至少为每个片段实现这三个方法，但您还应该使用几种其他回调方法来处理片段生命周期的各个阶段。 处理片段生命周期部分对所有生命周期回调方法做了更详尽的阐述。

您可能还想扩展几个子类，而不是 `Fragment` 基类：

`DialogFragment`

显示浮动对话框。使用此类创建对话框可有效地替代使用 `Activity` 类中的对话框帮助程序方法，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`

显示由适配器（如 `SimpleCursorAdapter`）管理的一系列项目，类似于 `ListActivity`。它提供了几种管理列表视图的方法，如用于处理点击事件的 `onListItemClick()` 回调。

`PreferenceFragment`

以列表形式显示 `Preference` 对象的层次结构，类似于 `PreferenceActivity`。这在为您的应用创建“设置” Activity 时很有用处。

添加用户界面

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

要想为片段提供布局，您必须实现 `onCreateView()` 回调方法，Android 系统会在片段需要绘制其布局时调用该方法。您对此方法的实现返回的 `View` 必须是片段布局的根视图。

> **注**：如果您的片段是 `ListFragment` 的子类，则默认实现会从 `onCreateView()` 返回一个 `ListView`，因此您无需实现它。

要想从 `onCreateView()` 返回布局，您可以通过 XML 中定义的布局资源来扩展布局。为帮助您执行此操作，`onCreateView()` 提供了一个 `LayoutInflater` 对象。

例如，以下这个 `Fragment` 子类从 `example_fragment.xml` 文件加载布局：

```
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);
    }
}
```

传递至 `onCreateView()` 的 `container` 参数是您的片段布局将插入到的父 `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`（处理片段生命周期部分对恢复状态做了详细阐述）。

`inflate()` 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`。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；
- 指示是否应该在扩展期间将扩展布局附加至 `ViewGroup`（第二个参数）的布尔值。（在本例中，其值为 false，因为系统已经将扩展布局插入 `container` — 传递 true 值会在最终布局中创建一个多余的视图组。）

> **创建布局**
>
> 在上例中，`R.layout.example_fragment` 是对应用资源中保存的名为 `example_fragment.xml` 的布局资源的引用。如需了解有关如何在 XML 中创建布局的信息，请参阅用户界面文档。

现在，您已经了解了如何创建提供布局的片段。接下来，您需要将该片段添加到您的 Activity 中。

向 Activity 添加片段

通常，片段向宿主 Activity 贡献一部分 UI，作为 Activity 总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，以下是一个具有两个片段的 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实例化的 `Fragment` 类。

  当系统创建此 Activity 布局时，会实例化在布局中指定的每个片段，并为每个片段调用 `onCreateView()` 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` 来替代 `<fragment>` 元素。

  > **注**：每个片段都需要一个唯一的标识符，重启 Activity 时，系统可以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以执行某些事务，如将其移除）。 可以通过三种方式为片段提供 ID：
  >
  > - 为 `android:id` 属性提供唯一 ID。
  > - 为 `android:tag` 属性提供唯一字符串。
  > - 如果您未给以上两个属性提供值，系统会使用容器视图的 ID。

- **或者通过编程方式将片段添加到某个现有 `ViewGroup`**

  您可以在 Activity 运行期间随时将片段添加到 Activity 布局中。您只需指定要将片段放入哪个 `ViewGroup`。

  要想在您的 Activity 中执行片段事务（如添加、移除或替换片段），您必须使用 `FragmentTransaction` 中的 API。您可以像下面这样从 `Activity` 获取一个 `FragmentTransaction` 实例：

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

  然后，您可以使用 `add()` 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：

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

  传递到 `add()` 的第一个参数是 `ViewGroup`，即应该放置片段的位置，由资源 ID 指定，第二个参数是要添加的片段。

  一旦您通过 `FragmentTransaction` 做出了更改，就必须调用 `commit()` 以使更改生效。

添加没有 UI 的片段

上例展示了如何向您的 Activity 添加片段以提供 UI。不过，您还可以使用片段为 Activity 提供后台行为，而不显示额外 UI。

要想添加没有 UI 的片段，请使用 `add(Fragment, String)` 从 Activity 添加片段（为片段提供一个唯一的字符串“标记”，而不是视图 ID）。这会添加片段，但由于它并不与 Activity 布局中的视图关联，因此不会收到对 `onCreateView()` 的调用。因此，您不需要实现该方法。

并非只能为非 UI 片段提供字符串标记 — 您也可以为具有 UI 的片段提供字符串标记 — 但如果片段没有 UI，则字符串标记将是标识它的唯一方式。如果您想稍后从 Activity 中获取片段，则需要使用 `findFragmentByTag()`。

如需查看将没有 UI 的片段用作后台工作线程的示例 Activity，请参阅 `FragmentRetainInstance.java` 示例，该示例包括在 SDK 示例（通过 Android SDK 管理器提供）中，以 `<sdk_root>/APIDemos/app/src/main/java/com/example/android/apis/app/FragmentRetainInstance.java` 形式位于您的系统中。

管理片段

要想管理您的 Activity 中的片段，您需要使用 `FragmentManager`。要想获取它，请从您的 Activity 调用 `getFragmentManager()`。

您可以使用 `FragmentManager` 执行的操作包括：

- 通过 `findFragmentById()`（对于在 Activity 布局中提供 UI 的片段）或 `findFragmentByTag()`（对于提供或不提供 UI 的片段）获取 Activity 中存在的片段。
- 通过 `popBackStack()`（模拟用户发出的*返回*命令）将片段从返回栈中弹出。
- 通过 `addOnBackStackChangedListener()` 注册一个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` 类文档。

如上文所示，您也可以使用 `FragmentManager` 打开一个 `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，可以根据用户行为通过它们执行添加、移除、替换以及其他操作。 您提交给 Activity 的每组更改都称为事务，您可以使用 `FragmentTransaction` 中的 API 来执行一项事务。您也可以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）。

您可以像下面这样从 `FragmentManager` 获取一个 `FragmentTransaction` 实例：

```
FragmentManager fragmentManager = getFragmentManager();
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()`、`remove()` 和 `replace()` 等方法为给定事务设置您想要执行的所有更改。然后，要想将事务应用到 Activity，您必须调用 `commit()`。

不过，在您调用 `commit()` 之前，您可能想调用 `addToBackStack()`，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，以下示例说明了如何将一个片段替换成另一个片段，以及如何在返回栈中保留先前状态：

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

在上例中，`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()` 可将替换事务保存到返回栈，以便用户能够通过按*返回*按钮撤消事务并回退到上一片段。

如果您向事务添加了多个更改（如又一个 `add()` 或 `remove()`），并且调用了 `addToBackStack()`，则在调用 `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 `FragmentTransaction` 添加更改的顺序无关紧要，不过：

- 您必须最后调用 `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的出现顺序

如果您没有在执行移除片段的事务时调用 `addToBackStack()`，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了 `addToBackStack()`，则系统会*停止*该片段，并在用户回退时将其恢复。

> **提示**：对于每个片段事务，您都可以通过在提交前调用 `setTransition()` 来应用过渡动画。

调用 `commit()` 不会立即执行事务，而是在 Activity 的 UI 线程（“主”线程）可以执行该操作时再安排其在线程上运行。不过，如有必要，您也可以从 UI 线程调用 `executePendingTransactions()` 以立即执行 `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（用户离开 Activity）之前使用 `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()`。

与 Activity 通信

尽管 `Fragment` 是作为独立于 `Activity` 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包含它的 Activity。

具体地说，片段可以通过 `getActivity()` 访问 `Activity` 实例，并轻松地执行在 Activity 布局中查找视图等任务。

```
View listView = getActivity().findViewById(R.id.list);
```

同样地，您的 Activity 也可以使用 `findFragmentById()` 或 `findFragmentByTag()`，通过从 `FragmentManager` 获取对 `Fragment` 的引用来调用片段中的方法。例如：

```
ExampleFragment fragment = (ExampleFragment) getFragmentManager().findFragmentById(R.id.example_fragment);
```

创建对 Activity 的事件回调

在某些情况下，您可能需要通过片段与 Activity 共享事件。执行此操作的一个好方法是，在片段内定义一个回调接口，并要求宿主 Activity 实现它。 当 Activity 通过该接口收到回调时，可以根据需要与布局中的其他片段共享这些信息。

例如，如果一个新闻应用的 Activity 有两个片段 — 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）— 那么片段 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 内声明：

```
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` 接口并替代 `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()` 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()` 中的 `Activity` 来实例化 `OnArticleSelectedListener` 的实例：

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnArticleSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement OnArticleSelectedListener");
        }
    }
    ...
}
```

如果 Activity 未实现接口，则片段会引发 `ClassCastException`。实现时，`mListener` 成员会保留对 Activity 的 `OnArticleSelectedListener` 实现的引用，以便片段 A 可以通过调用 `OnArticleSelectedListener` 接口定义的方法与 Activity 共享事

件。例如，如果片段 A 是 `ListFragment` 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()`，然后该方法会调用 `onArticleSelected()` 以与 Activity 共享事件：

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @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(ArticleColumns.CONTENT_URI, id);
        // Send the event and Uri to the host activity
        mListener.onArticleSelected(noteUri);
    }
    ...
}
```

传递到 `onListItemClick()` 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` 获取文章的 ID。

内容提供程序文档中提供了有关内容提供程序用法的更多详情。

向应用栏添加项目

您的片段可以通过实现 `onCreateOptionsMenu()` 向 Activity 的选项菜单（并因此向应用栏）贡献菜单项。不过，为了使此方法能够收到调用，您必须在 `onCreate()` 期间调用 `setHasOptionsMenu()`，以指示片段想要向选项菜单添加菜单项（否则，片段将不会收到对 `onCreateOptionsMenu()` 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()` 的回调。

您还可以通过调用 `registerForContextMenu()`，在片段布局中注册一个视图来提供上下文菜单。用户打开上下文菜单时，片段会收到对 `onCreateContextMenu()` 的调用。当用户选择某个菜单项时，片段会收到对 `onContextItemSelected()` 的调用。

> **注**：尽管您的片段会收到与其添加的每个菜单项对应的菜单项选定回调，但当用户选择菜单项时，Activity 会首先收到相应的回调。 如果 Activity 对菜单项选定回调的实现不会处理选定的菜单项，则系统会将事件传递到片段的回调。 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，请参阅菜单开发者指南和应用栏培训课程。

处理片段生命周期

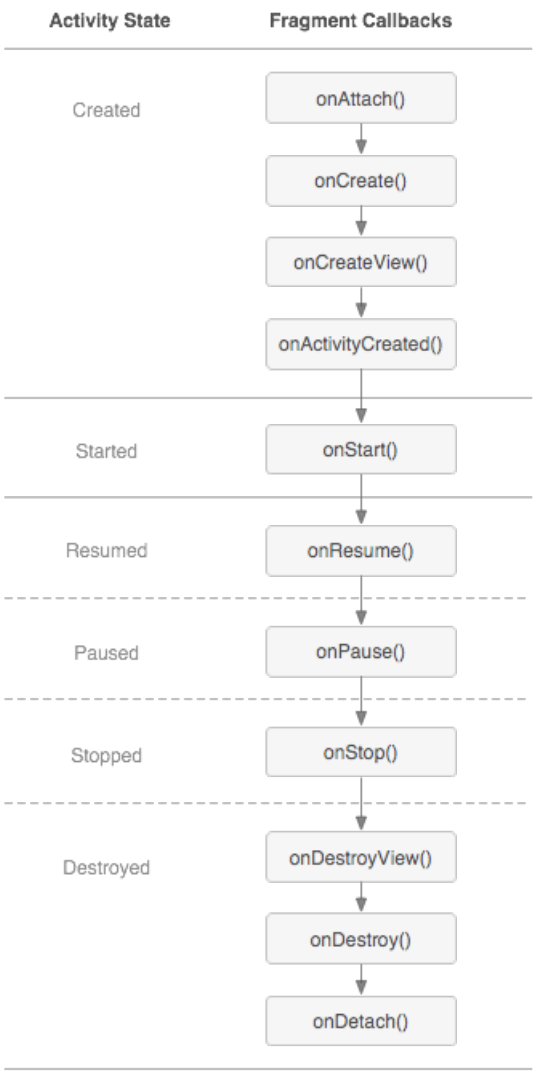

图 3. Activity 生命周期对片段生命周期的影响。

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：

继续

片段在运行中的 Activity 中可见。

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）。

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。 停止片段仍然处于活动状态（系统会保留所有状态和成员信息）。 不过，它对用户不再可见，如果 Activity 被终止，它也会被终止。

同样与 Activity 一样，假使 Activity 的进程被终止，而您需要在重建 Activity 时恢复片段状态，您也可以使用 `Bundle` 保留片段的状态。您可以在片段的 `onSaveInstanceState()` 回调期间保存状态，并可在 `onCreate()`、`onCreateView()` 或 `onActivityCreated()` 期间恢复状态。如需了解有关保存状态的详细信息，请参阅 Activity 文档。

Activity 生命周期与片段生命周期之间的最显著差异在于它们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（以便用户通过*返回*按钮回退到 Activity，任务和返回栈对此做了阐述）。不过，仅当您在移除片段的事务执行期间通过调用 `addToBackStack()` 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。 因此，管理 Activity 生命周期的做法同样适用于片段。 但您还需要了解 Activity 的生命周期对片段生命周期的影响。

> **注意**：如需 `Fragment` 内的某个 `Context` 对象，可以调用 `getActivity()`。但要注意，请仅在片段附加到 Activity 时调用

> `getActivity()`。如果片段尚未附加，或在其生命周期结束期间分离，则 `getActivity()` 将返回 null。

与 Activity 生命周期协调一致

片段所在的 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()` 时，Activity 中的每个片段也会收到 `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，以执行构建和销毁片段 UI 等操作。 这些额外的回调方法是：

`onAttach()`

在片段已与 Activity 关联时调用（`Activity` 传递到此方法内）。

`onCreateView()`

调用它可创建与片段关联的视图层次结构。

`onActivityCreated()`

在 Activity 的 `onCreate()` 方法已返回时调用。

`onDestroyView()`

在移除与片段关联的视图层次结构时调用。

`onDetach()`

在取消片段与 Activity 的关联时调用。

图 3 图示说明了受其宿主 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决定片段可以收到的回调方法。例如，当 Activity 收到其 `onCreate()` 回调时，Activity 中的片段只会收到 `onActivityCreated()` 回调。

一旦 Activity 达到恢复状态，您就可以随意向 Activity 添加片段和移除其中的片段。 因此，只有当 Activity 处于恢复状态时，片段的生命周期才能独立变化。

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

示例

为了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：**`FragmentLayout.java` 中提供了此 Activity 的完整源代码。

主 Activity 会在 `onCreate()` 期间以常规方式应用布局：

```
@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.FragmentLayout$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，系统可在 Activity 加载布局时立即实例化 `TitlesFragment`（列出戏剧标题），而 `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。 正如您将在下文所见的那样，用户从列表中选择某个项目后，系统才会将片段放入 `FrameLayout`。

不过，并非所有屏幕配置都具有足够的宽度，可以并排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`）。

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`）：

```
<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_parent" />
</FrameLayout>
```

此布局仅包括 `TitlesFragment`。这意味着，当设备纵向显示时，只有戏剧标题列表可见。 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，而不是加载另一个片段。

接下来，您可以看到如何在片段类中实现此目的。第一个片段是 `TitlesFragment`，它显示莎士比亚戏剧标题列表。该片段扩展了 `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`），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。

```
public static class TitlesFragment extends ListFragment {
    boolean mDualPane;
    int mCurCheckPosition = 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.TITLES));

        // 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);
        mDualPane = detailsFrame != null && detailsFrame.getVisibility() == View.VISIBLE;

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

        if (mDualPane) {
            // 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(mCurCheckPosition);
```

```
        }
    }

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

    @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) {
        mCurCheckPosition = index;

        if (mDualPane) {
            // 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)
                    getFragmentManager().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 = getFragmentManager().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：

```
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
            // won't be displayed.  Note this is not 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_DIP,
                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“根本不”包括 `R.id.details` 视图（即 `DetailsFragment` 所属视图），则应用会启动 `DetailsActivity` Activity 以显示该项目的内容。

以下是 `DetailsActivity`，它简单地嵌入了 `DetailsFragment`，以在屏幕为纵向时显示所选的戏剧摘要：

```
public static class DetailsActivity extends Activity {

    @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());
            getFragmentManager().beginTransaction().add(android.R.id.content, details).commit();
        }
    }
}
```

请注意，如果配置为横向，则此 Activity 会自行完成，以便主 Activity 可以接管并沿 `TitlesFragment` 显示 `DetailsFragment`。如果用户在纵向显示时启动 `DetailsActivity`，但随后旋转为横向（这会重启当前 Activity），就可能出现这种情况。

如需查看使用片段的更多示例（以及本示例的完整源文件），请参阅 ApiDemos（可从示例 SDK 组件下载）中提供的 API Demos 示例应用。