**Fragment**

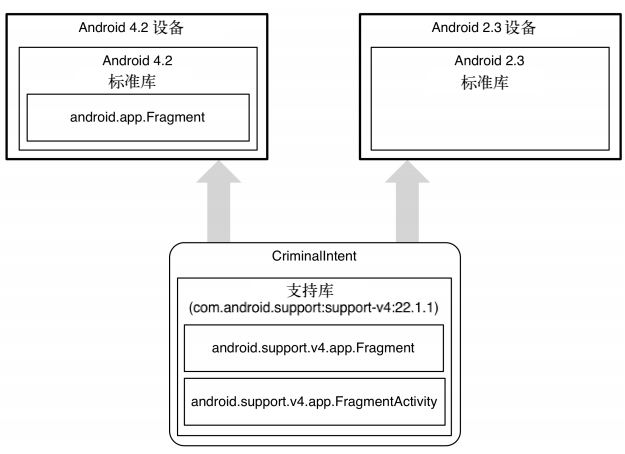
资料：<https://www.jianshu.com/p/fd71d65f0ec6>

<https://www.cnblogs.com/purediy/p/3276545.html>

Fragment是用来管理Activity部分视图的任务的。Fragment是不能单独的展示视图的，它必须添加到Activity的视图的层级中，Fragment才能显示它管理的视图。

**版本**

Fragment是Android系统3.0的标准库中开始提供的android.app.Fragment。为了兼容android系统3.0以前的版本也可以使用Fragment，android提供了support-v4:22.1.1支持库来兼容以前的系统，主要提供了android.support.v4.app.Fragment和android.support.v4.app.FragmentActivity两个类。使用Fragment的前提是Activity知道如何管理它视图层级中的Fragment，所以只有v4包中的FragmentActivity才能管理v4包中Fragment。



**生命周期**

Fragment是必须依赖Activity而存在的，所以Fragment的生命周期是由Activity所管理，Fragment的生命周期大部分与Activity的生命周期相对应，如下图：

****

Fragment的生命周期比Activity要多几个状态，除了多出来的几个方法，其他方法都是跟随Activity生命周期的调用而回调，其中onAttach、onCreateView、onCreate三个方法是在Activity的setContentView方法中调用的，下面一次解释下多出来的几个方法：

**onAttach():**当Fragment与Activity关联时调用（当fragment被加入到activity时）。

**onCreateView()：**activity获取Fragment视图时调用，创建Fragment视图。

**onActivityCreated()：**当Activity的onCreate方法返回时调用。

**onDestoryView()：**当该Fragment的视图被移除时调用。

**onDetach()：**当Fragment与Activity关联被取消时调用。

* **使用**

Fragment的使用有两种方式：

1. 静态管理Fragment：在Activity的布局文件中添加<fragment/>标签，并设置name属性为自己的

Fragment的路径即可。

1. 动态管理Fragment：在Activity的布局文件中添加<FrameLayout/>标签指定Fragment的位置，

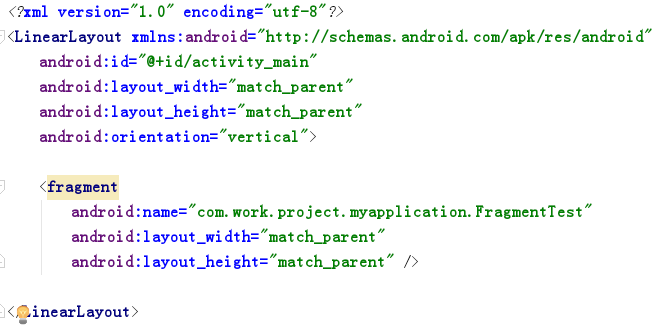
然后在Activity的代码中动态的添加Fragment。

不管上面哪种方式使用，Fragment是需要添加到Activity的视图层级结构中才能显示的，所以Activity需要在自己的的布局文件中使用<fragment/>或者<FrameLayout/>标签给自己管理的Fragment指定一个位置。指定好位置后将定义好的Fragment添加到这个位置上就可以显示Fragment了。指定好位置后通过Activity来管理Fragment。所以使用Fragment需要必要的两个步骤：

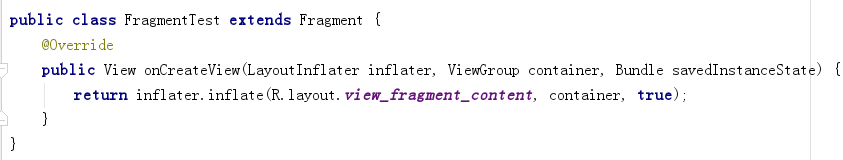
1. 为Fragment在Activity的视图结构中添加位置。
2. 管理Fragment的生命周期。

**静态管理Fragment**

静态管理Fragment不需要在Activity中添加任何代码，只需要在Activity的布局文件中添加一个<fragment/>标签，来指定Fragment的位置，然后把定义好的Fragment路径告诉<fragment/>标签的name属性：



下面是自己定义的Fragment并重写生命周期方法，最基本的必须重写onCreateView方法来指定视图：



**动态管理Fragment**

* **添加位置**

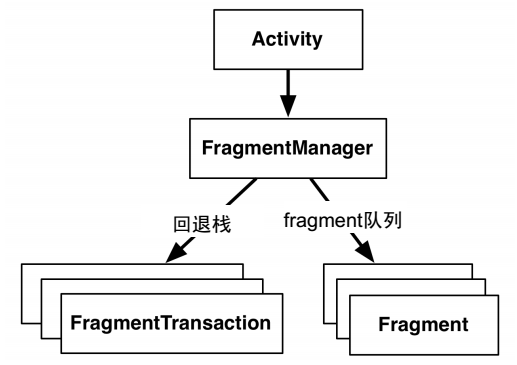
动态管理Fragment需要在Activity的布局文件中添加一个<FrameLayout/>标签，来指定Fragment的位置:



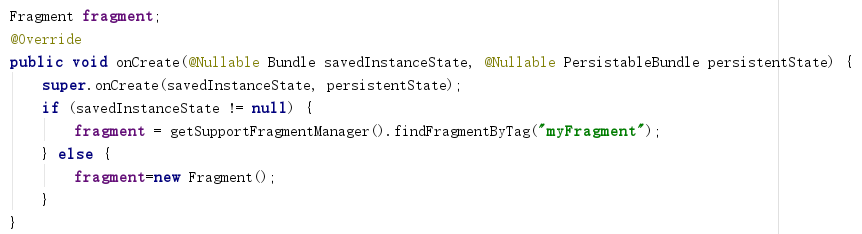
* **FragmentManager**

Activity通过FragmentManager来管理Fragment，FragmentManager负责将Fragment添加到Activity的视图层级中去。FragmentManager具体管理的是：

1. fragment队列
2. fragment事务回退栈

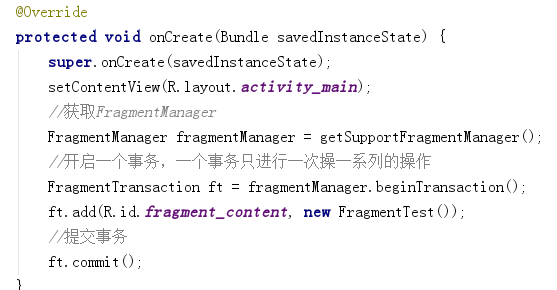


其中每个Activity都会有一个属于自己的FragmentManager。并保存一个fragment队列，当Activity由于异常销毁时（例如旋转屏幕），会临时保存这个fragment队列。然后可以在Activity的onCreate()方法中首先判断，如下：



* **fragment事务**

fragment事务被用来添加、移除、附加、分离或替换FragmentManager的**fragment队列中**的fragment等操作：



当事务被提交后就可以正常显示fragment了。但是我们在使用的时候，一定要知道哪个会销毁视图，哪个会销毁实例，哪个仅仅只是隐藏，下面是事务的其它方法：

****remove()**：**从队列中移除一个Fragment，如果被移除的Fragment没有添加到回退栈，这个Fragment实例将会被销毁并且会调用到Fragment的onPause、onStop、 onDestroyView、onDestroy方法。

****replace()**：**使用另一个Fragment替换当前的，实际上就是remove()然后add()的合体。

****hide(Fragment fragment)：****隐藏当前的Fragment，仅仅是设为不可见，并不会销毁

****show(Fragment fragment)：****显示之前隐藏的Fragment

****detach(Fragment fragment)：****会销毁fragment的视图，实例并不会被销毁,此时会调onPause、onStop、onDestroyView方法。和remove()不同,此时fragment的状态依然由FragmentManager维护。

****attach()：****重建view视图附加到UI上并显示并调用 onViewCreated、onActivityCreated、onStart、onResume方法。

****add(Fragment fragment, String tag)：**** 调用add(int, Fragment, String),填入为0的containerViewId.

****add(int containerViewId, Fragment fragment)：****调用add(int, Fragment, String),填入为null的tag.

****add(int containerViewId, Fragment fragment, String tag)：****向Activity中添加一个Fragment.

****addSharedElement(View sharedElement, String name)：****添加共享元素

****addToBackStack(String name)：****将事务添加到回退栈

****commit()：**** 提交事务

****commitAllowingStateLoss()：****类似commit()，但允许在Activity状态保存之后提交（即允许状态丢失）。

****commitNow()：****同步提交事务

****commitNowAllowingStateLoss()：****类似commitNow()，但允许在Activity状态保存之后提交（即允许状态丢失）。

****disallowAddToBackStack()：****不允许调用addToBackStack(String)操作

****isAddToBackStackAllowed()：****是否允许添加到回退栈

****isEmpty()：****事务是否未包含的任何操作

****setBreadCrumbShortTitle(int res)：****为事务设置一个BreadCrumb短标题

****setBreadCrumbShortTitle(CharSequence text)：****为事务设置一个BreadCrumb短标题，将会被FragmentBreadCrumbs使用

****setBreadCrumbTitle(int res)：****为事务设置一个BreadCrumb全标题，将会被FragmentBreadCrumbs使用

****setBreadCrumbTitle(CharSequence text)：****为事务设置一个BreadCrumb全标题

****setCustomAnimations(int enter, int exit, int popEnter, int popExit)****：自定义事务进入/退出以及入栈/出栈的动画效果

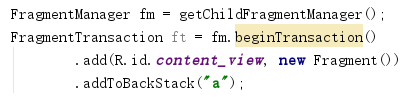
****setCustomAnimations(int enter, int exit)：****自定义事务进入/退出的动画效果

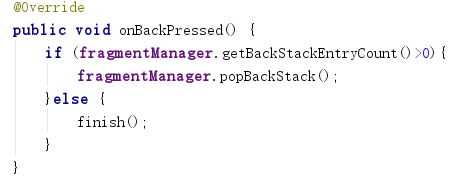
****setTransition(int transit)：****为事务设置一个标准动画

****setTransitionStyle(int styleRes)：****为事务标准动画设置自定义样式

* **fragment回退栈**

回退栈是针对事务来说的，不是针对fragment来说的。Activity维护一个回退栈来保存每次Fragment事务的变化，如果你将Fragment添加到回退栈当用户点击返回按钮时，你将看到上次保存的Fragment。一旦Fragment完全从后退站中弹出，用户再次点击后退键，则退出当前Activity。将Fragment添加到回退栈：

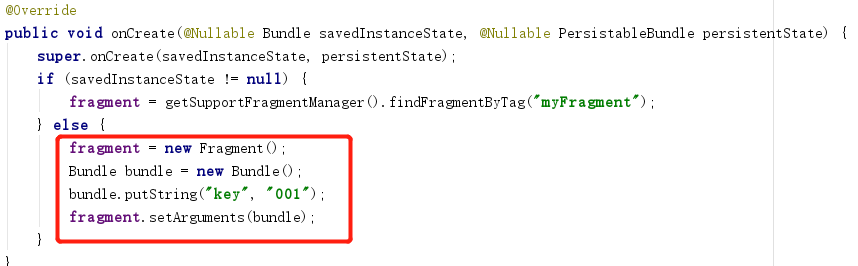




* **fragment与Activity之间通信**

首先可以在Fragment实例中通过getActivity（）回去Activity的实例，然后就可以调用Activity中的方法进行通信。

在创建Fragment的实例时可以通过fragment.setArguments(bundle);向Fragment传递一些数据，来初始化Fragment页面:



* **fragment的onHinddenChanged()与setUserVisibleHint()方法**

**onHinddenChange():**在切换Fragment的时候使用hint或者show的方式（通过显示和隐藏来切换），此时Fragment的生命周期不会被执行，但是会调用onHinddenChange()，如果想要刷新页面内容则可以在这个方法中进行相应操作。此方法在ViewPage+Fragment结构的页面中，通过ViewPage切换Fragment时，不会被调用。因为ViewPage切换Fragment是通过事务的replace()方法来切换的。

**setOnUserVisibleHint():**它在ViewPage+Fragment结构的页面中，当viewpager切换tab时，tab切换一次便会执行一次setUserVisibleHint()方法。它并不存在于Fragment的生命周期中，也不会因为Fragmentd的状态改变而自动被调用，除非人为的调用它。并且在viewpage调用setAdapter方法时也会调用setUserVisibleHint()方法，所以它会在Fragment生命周期之前被调用。另外，getUserVisibleHint的初始值为true。