Fragment

1. 简介

Fragment相当于Activity的一个界面的控件，一个Activity的界面可以完全有不同的Fragment组成，使界面过渡更加美观，界面操作更加流畅, Fragment拥有自己的生命周期和接收、处理用户的事件，你可以动态的添加、替换和移除某个Fragment.

1. 生命周期

必须依赖于activity，与activity生命周期关系如下



onCreate过程

01-22 15:30:28.091: E/HJJ(10315): Activity &&&& onCreate...

01-22 15:30:28.091: E/HJJ(10315): ArrayListFragment \*\*\*\* onAttach...

01-22 15:30:28.091: E/HJJ(10315): ArrayListFragment \*\*\*\* onCreate...

01-22 15:30:28.115: E/HJJ(10315): ArrayListFragment \*\*\*\* onCreateView...

01-22 15:30:28.123: E/HJJ(10315): ArrayListFragment \*\*\*\* onActivityCreated...

onStart过程

01-22 15:30:28.123: E/HJJ(10315): Activity &&&& onStart...

01-22 15:30:28.123: E/HJJ(10315): ArrayListFragment \*\*\*\* onStart...

onResume过程

01-22 15:30:28.123: E/HJJ(10315): Activity &&&& onResume...

01-22 15:30:28.123: E/HJJ(10315): ArrayListFragment \*\*\*\* onResume...

onPause过程

01-22 15:31:26.748: E/HJJ(10315): ArrayListFragment \*\*\*\* onPause...

01-22 15:31:26.748: E/HJJ(10315): Activity &&&& onPause...

onStop过程

01-22 15:31:27.638: E/HJJ(10315): ArrayListFragment \*\*\*\* onStop...

01-22 15:31:27.638: E/HJJ(10315): Activity &&&& onStop...

onStart过程

01-22 15:31:57.537: E/HJJ(10315): Activity &&&& onStart...

01-22 15:31:57.537: E/HJJ(10315): ArrayListFragment \*\*\*\* onStart...

onResume过程

01-22 15:31:57.537: E/HJJ(10315): Activity &&&& onResume...

01-22 15:31:57.537: E/HJJ(10315): ArrayListFragment \*\*\*\* onResume...

onPause过程

01-22 15:32:47.412: E/HJJ(10315): ArrayListFragment \*\*\*\* onPause...

01-22 15:32:47.412: E/HJJ(10315): Activity &&&& onPause...

onStop过程

01-22 15:32:47.865: E/HJJ(10315): ArrayListFragment \*\*\*\* onStop...

01-22 15:32:47.865: E/HJJ(10315): Activity &&&& onStop...

onDestroy过程

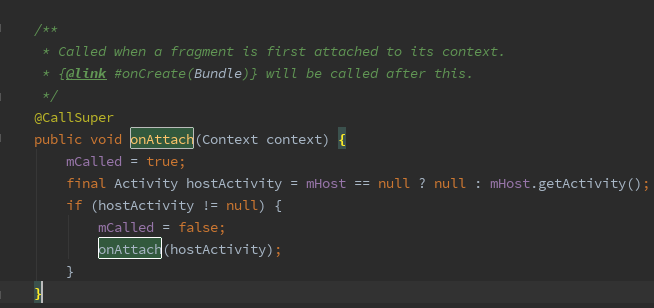
01-22 15:32:47.865: E/HJJ(10315): ArrayListFragment \*\*\*\* onDestroyView...

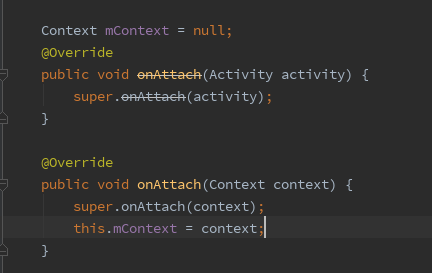
01-22 15:32:47.865: E/HJJ(10315): ArrayListFragment \*\*\*\* onDestroy...

01-22 15:32:47.865: E/HJJ(10315): ArrayListFragment \*\*\*\* onDetach...

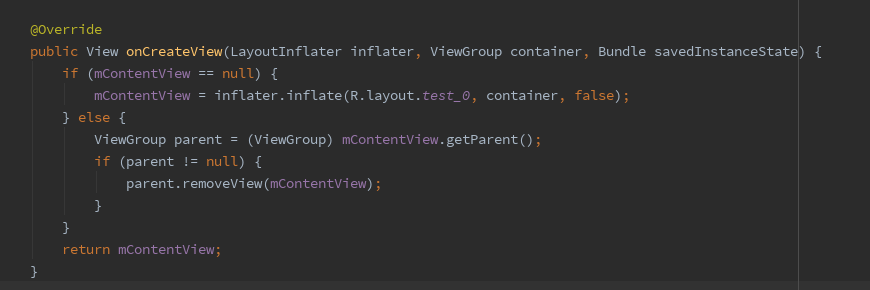
01-22 15:32:47.865: E/HJJ(10315): Activity &&&& onDestroy...

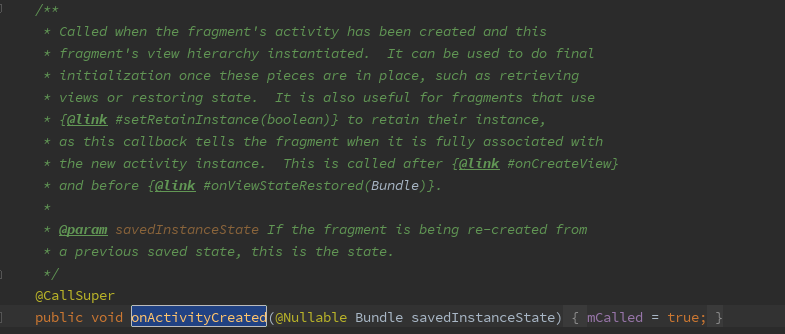
**onAttach(Activity)**  
当Fragment与Activity(FragmentActivity)发生关联时调用。用这个方法传入context



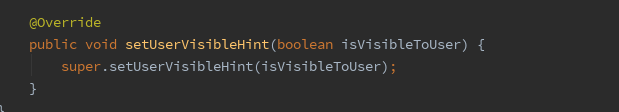


**onCreateView(LayoutInflater, ViewGroup,Bundle)**  
创建该Fragment的视图

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



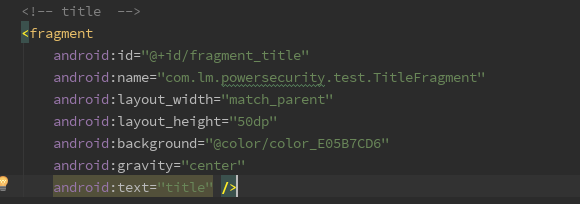
**onDestoryView()**  
与onCreateView想对应，当该Fragment的视图被移除时调用  
**onDetach()**  
与onAttach相对应，当Fragment与Activity关联被取消时调用  
注意：除了onCreateView，其他的所有方法如果你重写了，必须调用父类对于该方法的实现，



类似懒加载

1. 静态模式

类如：



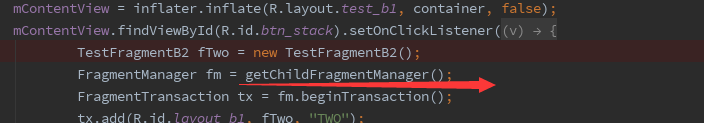
1. 动态模式



1. FragmentManager

主要用于在Activity中操作Fragment

分别有 getFragmentManager() 和getSupportFragmentManager（v4中）



Fragment 的嵌套： fragment 可以在里面 再管理fragment组，getchildFragmentmanager() 就是 父类的 fragment manager

1. FragmentTransaction 使用Fragment时，可以通过用户交互来执行一些动作，比如增加、移除、替换等

FragmentTransaction transaction = fragmentManager.benginTransatcion();//开启一个事务

**transaction.add()**

**transaction.remove()**

从Activity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。

**transaction.replace() ，**使用另一个Fragment替换当前的

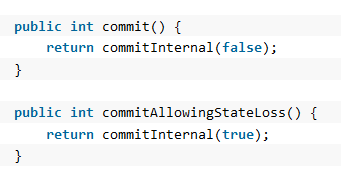
**transaction.hide()** 隐藏当前的Fragment，仅仅是设为不可见，并不会销毁

**transaction.show()**

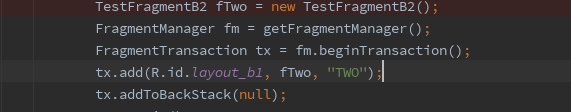
transatcion.commit()//提交一个事务

commitAllowingStateLoss()

你只能在activity存储它的状态（当用户要离开activity时）之前调用commit()，如果在存储状态之后调用commit()，将会抛出一个异常。  
这是因为当activity再次被恢复时commit之后的状态将丢失。如果丢失也没关系，那么使用commitAllowingStateLoss()方法。这两种方法区别就是 一个状态的判断，commitAllowingStateLoss（）不会走状态判断 所以 不抛出异常



AddToBackStack（）



AddToBackStack 加入栈 ，如果开了多个的话 ，回退会逐步移除站内元素，知道退出activity。

1. 与activity 通信





如果activity 包含fragment 的引用，可以直接强转，也可以在add的时候 加入id 和tag，然后通过findFragmentByTag findFragmentById 取出来。

Fragment中获取activity 直接 getActivity()



也可以通过接口回调来触发，这样会把activity 和 fragment 解耦

像日历 需要复用fragment 那些的 需要activity 与 fragment 绑定联系不那么深，可以使用setArguments 来在oncreate的时候传入简单参数

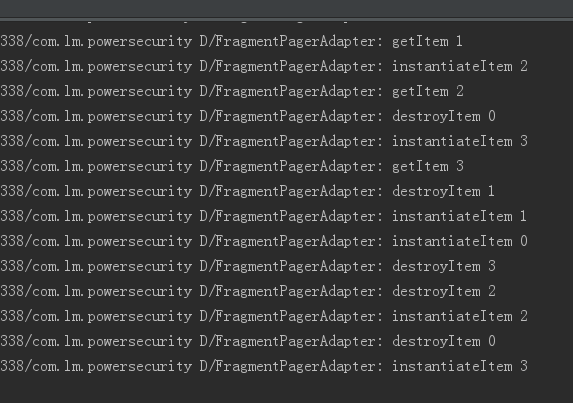


1. FragmentPagerAdapter与FragmentStatePagerAdapter

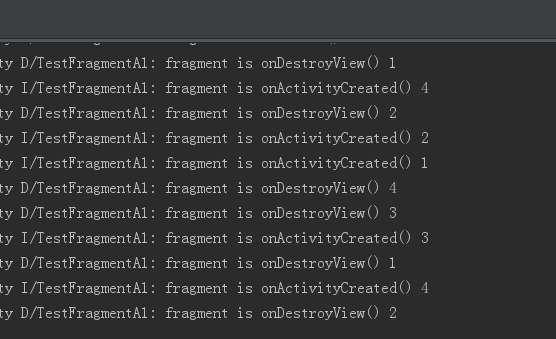
FragmentPagerAdapter：对于不再需要的fragment，选择调用detach方法，仅销毁视图，并不会销毁fragment实例。

FragmentStatePagerAdapter：会销毁不再需要的fragment，当当前事务提交以后，会彻底的将fragmeng从当前Activity的FragmentManager中移除，state标明，销毁时，会将其onSaveInstanceState(Bundle outState)中的bundle信息保存下来，当用户切换回来，可以通过该bundle恢复生成新的fragment，也就是说，你可以在onSaveInstanceState(Bundle outState)方法中保存一些数据，在onCreate中进行恢复创建。

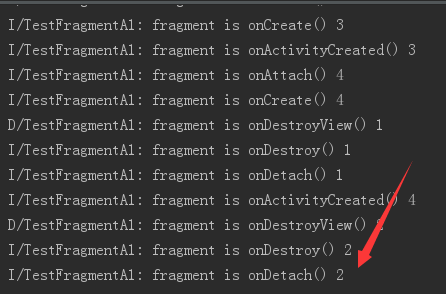
如上所说，使用FragmentStatePagerAdapter当然更省内存，但是销毁新建也是需要时间的。一般情况下，如果你是制作主页面，就3、4个Tab，那么可以选择使用FragmentPagerAdapter，如果你是用于ViewPager展示数量特别多的条目时，那么建议使用FragmentStatePagerAdapter。



Getinem 只会在创建时去调用 以后每次 都调用instantiateintem（）



上面是FragmentPagerAdapter 中fragment的生命周期



上面是FragmentStatePagerAdapter中fragment的生命周期

1. 一些注意事项

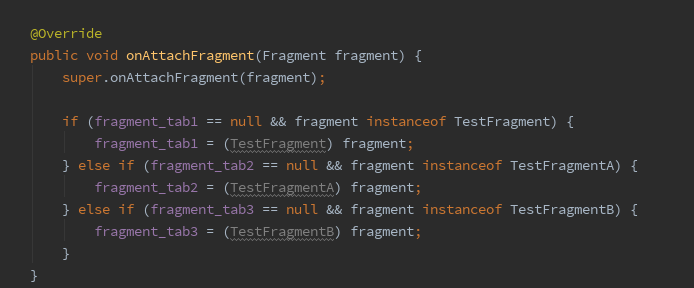
* Activity继承FragmentActivity
* 要注意使用 getSupportFragmentManager().beginTransaction().add（int containerViewId, Fragment fragment）这个fragment 用的是v4的



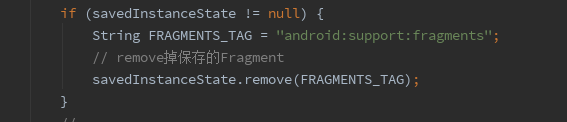
* getActivity() 为null 用onAttach（）传入context
* 如果使用hide()这些方法，比如微信、QQ的底部tab等情景，如果你什么都不做的话，在“内存重启”后回到前台，app的这几个Fragment界面会重叠。原因是FragmentManager帮我们管理Fragment,每当我们离开该Activity，FragmentManager都会保存它的Fragments，当发生“内存重启”，他会从栈底向栈顶的顺序一次性恢复Fragments，并且全都都是以show()的方式，所以我们看到了界面重叠。（如果是replace，恢复形式和Activity一致，只有当你pop之后上一个Fragment才开始重新恢复，所有使用replace不会造成重叠现象）

解决：

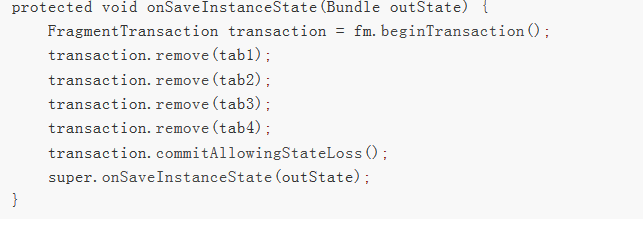
onAttachFragment，重新让新的Fragment指向了原本未被销毁的fragment，它就是onAttach方法对应的Fragment对象



或者



还有一种方法



* 当使用Fragment去嵌套另外一些子Fragment的时候，我们需要去管理子Fragment，这时候需要调用ChildFragmentManager去管理这些子Fragment，由此可能产生的Exception主要是：   
  java.lang.IllegalStateException: No activity

这个Exception是由什么原因造成的呢？如果想知道造成异常的原因，那就必须去看Fragment的相关代码，发现Fragment在detached之后都会被reset掉，但是它并没有对ChildFragmentManager做reset，所以会造成ChildFragmentManager的状态错误。

找到异常出现的原因后就可以很容易的去解决问题了，我们需要在Fragment被detached的时候去重置ChildFragmentManager

