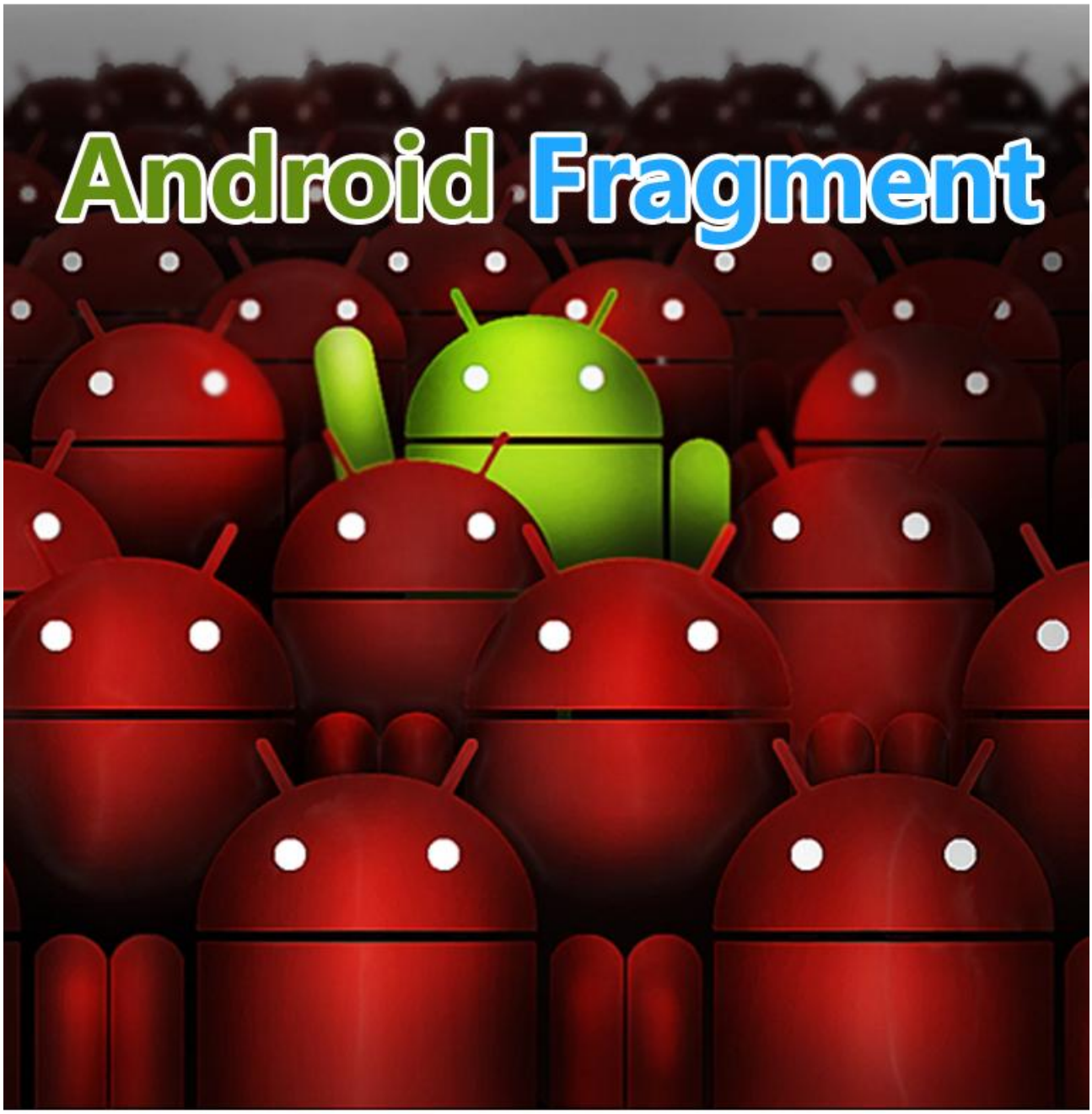
第28期

eoe 特刊



eoe 开发者社区

让开发更简单，做最棒的移动开发者社区

目录

前言

继上次特刊以后，我们讨论出来的新一期特刊主题为 Fragment。经过 40 多天的选材、制作、编辑，龙年的最后一期特刊终于出炉了。

这次特刊还是秉着在一个主题的前提下，eoer 们进行自由发挥，做自己想要做的内容，并在很短的时间做完提交，并对反馈的意见进行细心的修改。

内容上比上期特刊更加的简洁明了，对知识的更加统一，项目的分析，以及对特别的类进行了说明，所包含的项目是由浅到深的过程，值得我们去进行学习。

在这里要对参与本次特刊制作的网友（维王，1026438521，liangpingyy，hebang32624）表示衷心感谢。别外对一些由于时间问题不能参与到特刊制作的网友，表示感谢。

特刊内容中如有不当之处，欢迎各位开发者纠正。如有疑问请发帖至特刊专区进行反馈：
http://www.eoeAndroid.com/forum-39-1.html

eoe，让开发更简单，做最棒的移动开发者社区
http://www.eoe.cn

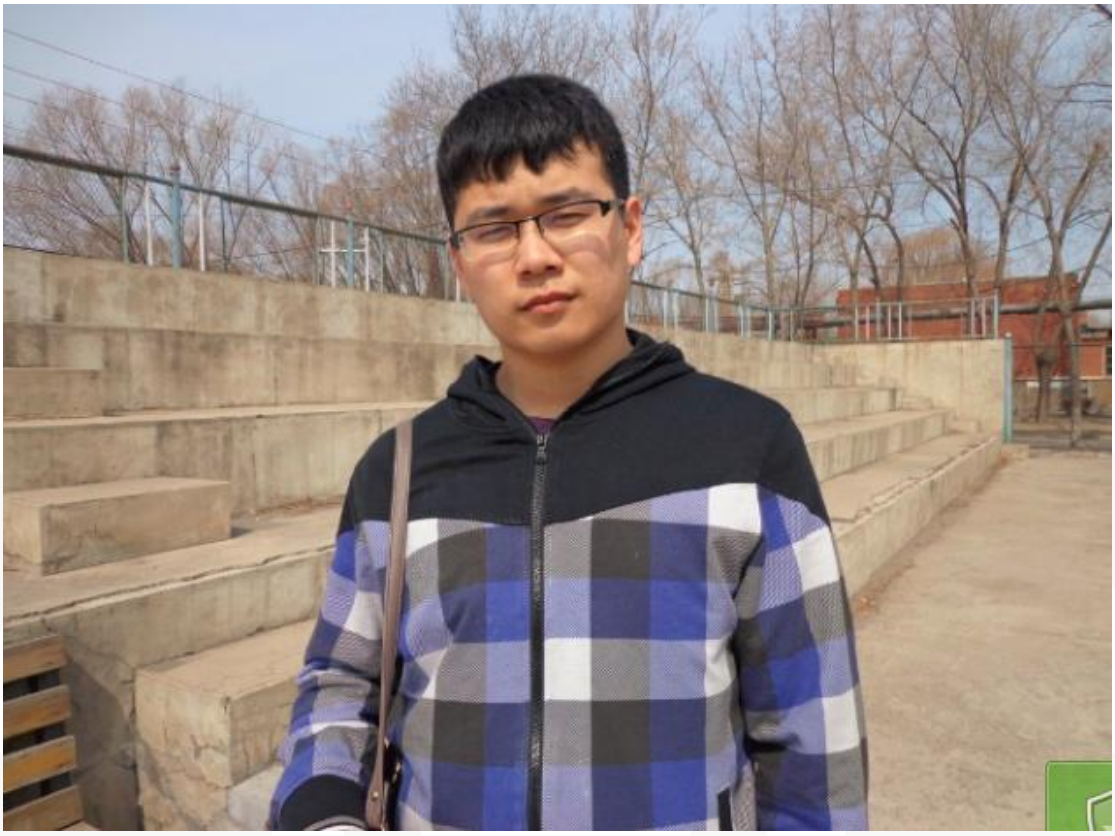
jiayouhe123
2013.1.28

作者简介

维王

论坛 ID：维王，程序媛一枚，11 年就已经是社区 framework 版块斑竹。在学校就开始跟着老师做 java 的项目，09 年末 10 年初接触 Android，7 月份毕业之后，直接从事 Android 开发，一直到现在。性格安静。喜欢旅游，古筝。上面就是西藏旅行的照片。版主大人自己坚持每天都学习一点点，同时相信有付出就会有收获。让我们祝愿美女版主在新的一年里工作顺利，事业有成。

1026438521

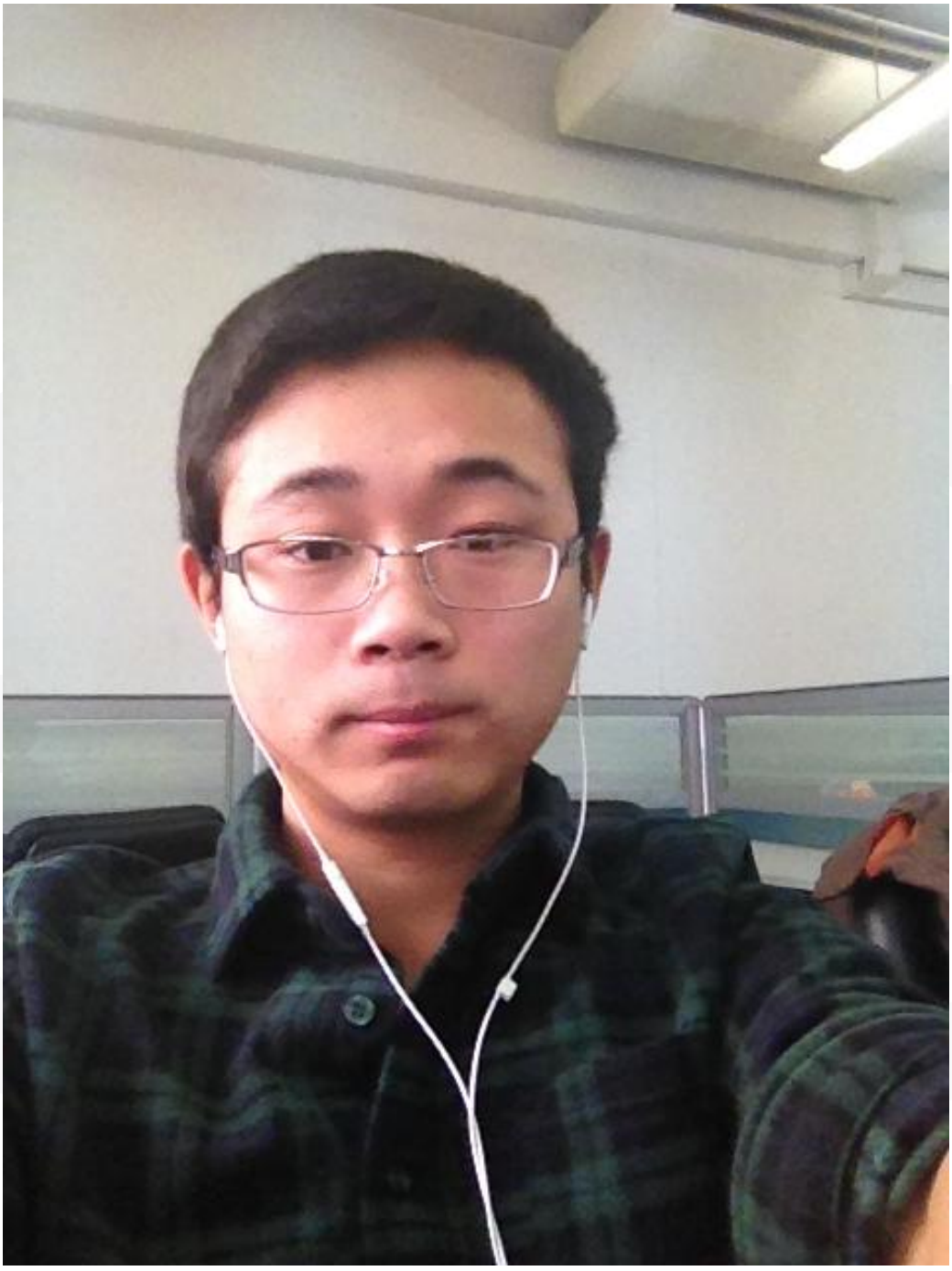


论坛 id：1026438521，在一次偶然的机会发现了这位亲。并找到他和我一起制作特刊，欣然的同意了。他说：第一次看到这个论坛的时候，感觉非常不错，有很多人分享了他的代码，这让很多初学者收益匪浅，也让正在工作的人不用自己敲代码或者看到新思路，也让大神们有了相互交流的平台，然而自己却谦虚的说自己不属于其中任何一种，总之就是这么稀里糊涂的进来了，希望自己发的帖子对别人有帮助。非常感谢 1026438521 的奉献。

liangpingyy

社区 ID：liangpingyy，第一批为社区做出贡献的 eoer，liangpingyy 在报考大学的时候糊里糊涂的选择了 IT，于是便果断的从了 IT。现为西安一所高校研究僧，踢足球、打篮球，最爱乒乓球。作为一个准社会人，发觉对校园依稀还有留恋。专业电子，误入软件公司，被骗自学 Android 一年有余。总结这些年，有些奇遇，有些感慨，有些感悟，还想感谢大家，一起走过的路，无论是坛里坛外，无论是认识不认识，感谢 eoeAndroid 所提供的这个平台。最后，一起祝愿 liangpingyy 在明年的毕业答辩顺利完成，新的工作会更好。

hebang32624



社区 ID：hebang32624，hebang32624 是一位在读研究生。喜欢编程，Android 是个人爱好，把自己归纳为入门级大牛，大家要多包涵一下 hebang32624。现在学习的主要方向是集群航天器系统和卫星编队飞行星间通讯。听着好吓人的。不过 hebang32624 说自己现在就是从硬件到软件，到建模.反正老师需要啥就去做啥，整个一个全能儿，最近，hebang32624 还要出国去做演讲， 让我们祝愿他演讲成功。是英文的哦。

一、Fragment 的基础知识介绍

By hebang32624

1.1 概述

1.1.1 特性

Fragment 是 activity 的界面中的一部分或一种行为。可以把多个 Fragment 组合到一个 activity 中来创建一个多面界面并且可以在多个 activity 中重用一个 Fragment。可以把 Fragment 认为模块化的一段 activity，它具有自己的生命周期，接收它自己的事件，并可以在 activity 运行时被添加或删除。

Fragment 不能独立存在，它必须嵌入到 activity 中，而且 Fragment 的生命周期直接受所在的 activity 的影响。例如：当 activity 暂停时，它拥有的所有的 Fragment 都暂停了，当 activity 销毁时，它拥有的所有 Fragment 都被销毁。然而，当 activity 运行时（在 onResume()之后，onPause()之前），可以单独地操作每个 Fragment，比如添加或删除它们。当在执行上述针对 Fragment 的事务时，可以将事务添加到一个栈中，这个栈被 activity 管理，栈中的每一条都是一个 Fragment 的一次事务。有了这个栈，就可以反向执行 Fragment 的事务，这样就可以在 Fragment 级支持“返回”键（向后导航）。

当向 activity 中添加一个 Fragment 时，它须置于 ViewGroup 控件中，并且需定义 Fragment 自己的界面。可以在 layoutxml 文件中声明 Fragment，元素为：<fragment>；也可以在代码中创建 Fragment，然后把它加入到 ViewGroup 控件中。然而，Fragment 不一定非要放在 activity 的界面中，它可以隐藏在后台为 actvitiy 工作。

1.1.2 生命周期

onCreate()：
当创建 fragment 时系统调用此方法。在其中必须初始化 fragment 的基础组件们。可参考 activity 的说明。
onCreateView()：
系统在 fragment 要画自己的界面时调用（在真正显示之前）此方法。这个方法必须返回 frament 的 layout 的根控件。如果这个 fragment 不提供界面，那它应返回 null。
onPause()：
大多数程序应最少对 fragment 实现这三个方法。当然还有其它几个回调方法可应该按情况实现之。所有的生命周期回调函数在“操控 fragment 的生命周期”一节中有详细讨论。

下图为 fragment 的生命周期（它所在的 activity 处于运行状态）。

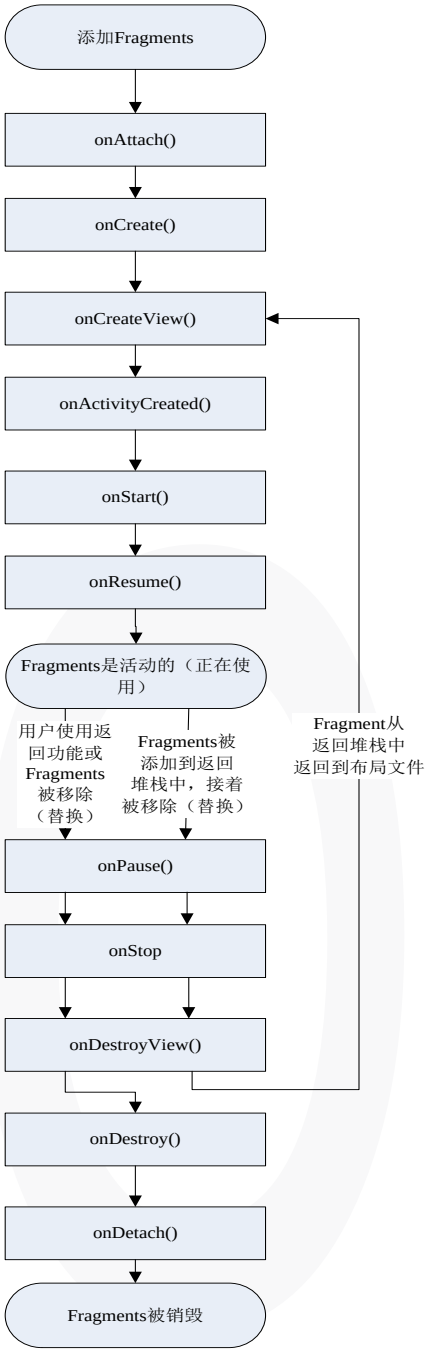


图 1 Fragment 生命周期

1.1.3 派生类

DialogFragment
显示一个浮动的对话框。使用这个类创建对话框是替代 activity 创建对话框的最佳选择．因为可以把 fragmentdialog 放入到 activity 的返回栈中，使用户能再返回到这个对话框。

ListFragment
显示一个列表控件，就像 ListActivity 类，它提供了很多管理列表的方法，比如 onListItemClick()方法响应 click 事件。

PreferenceFragment
显示一个由 Preference 对象组成的列表，与 PreferenceActivity 相同。它用于为程序创建“设置”activity。

1.2 范例

写一个读新闻的程序，可以用一个 fragment 显示标题列表，另一个 fragment 显示选中标题的内容，这两个 fragment 都在一个 activity 上，并排显示。那么这两个 fragment 都有自己的生命周期并响应自己感兴趣的事件。于是，不需再像手机上那样用一个 activity 显示标题列表，用另一个 activity 显示新闻内容；现在可以把两者放在一个 activity 上同时显示出来。如下图：

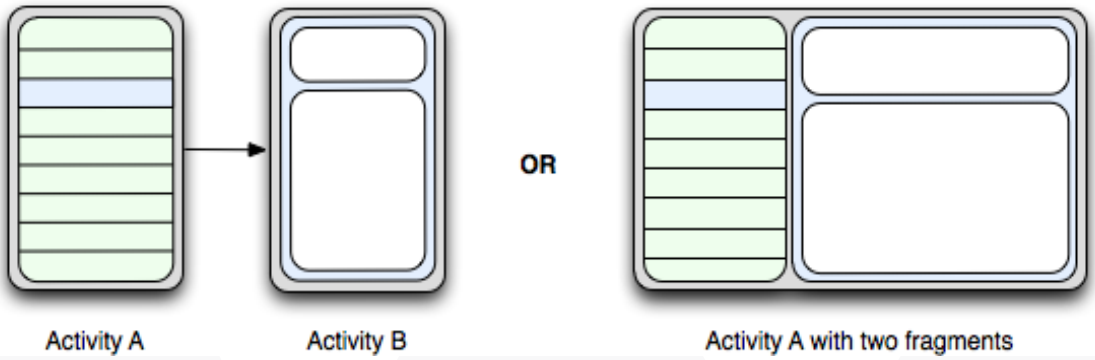


图 2 Fragment 说明性示例

Fragment 必须被写成可重用的模块。因为 fragment 有自己的 layout，自己进行事件响应，拥有自己的生命周期和行为，所以可以在多个 activity 中包含同一个 Fragment 的不同实例。这对于让界面在不同的屏幕尺寸下都能给用户完美的体验尤其重要。比如可以在程序运行于大屏幕中时启动包含很多 fragment 的 activity，而在运行于小屏幕时启动一个包含少量 fragment 的 activity。

刚才读新闻的程序，当检测到程序运行于大屏幕时，启动 activityA，将标题列表和新闻内容这两个 fragment 都放在 activityA 中；当检测到程序运行于小屏幕时，还是启动 activityA，但此时 A 中只有标题列表 fragment，当选中一个标题时，activityA 启动 activityB，B 中含有新闻内容 fragment。

1.3 创建 Fragmet

要创建 fragment，必须从 Fragment 或 Fragment 的派生类派生出一个类。Fragment 的代码写起来有些像 activity。它具有跟 activity 一样的回调方法，比如 onCreate(),onStart(),onPause()和 onStop()。实际上，如果想把老的程序改为使用 fragment，基本上只需要把 activity 的回调方法的代码移到 fragment 中对应的方法即可。

1.3.1 添加有界面的 Fragment

Fragment 一般作为 activity 的用户界面的一部分，把它自己的 layout 嵌入到 activity 的 layout 中。一个要为 fragment 提供 layout，必须实现 onCreateView()回调方法，然后在这个方法中返回一个 View 对象，这个对象是 fragment 的 layout 的根。

注意：如果的 fragment 是从 ListFragment 中派生的，就不需要实现 onCreateView()方法了，因为默认的实现已经为返回了 ListView 控件对象。

要从 onCreateView()方法中返回 layout 对象，可以从 layoutxml 中生成 layout 对象。为了帮助这样做，onCreateView()提供了一个 LayoutInflater 对象。举例：以下代码展示了一个 Fragment 的子类如何从 layoutxml 文件 example_fragment.xml 中生成对象。

```
PublicstaticclassExampleFragmentextendsFragment{
    @Override
   PublicViewonCreateView(LayoutInflaterinflater,ViewGroupcontainer,
BundlesavedInstanceState){
        //Inflate the layout for this fragment
```

```
        returninflater.inflate(R.layout.example_fragment,container,false);
    }
}
```

onCreateView()参数中的 container 是存放 fragment 的 layout 的 ViewGroup 对象。savedInstanceState 参数是一个 Bundle，跟 activity 的 onCreate()中 Bundle 差不多，用于状态恢复。但是 fragment 的 onCreate()中也有 Bundle 参数，所以此处的 Bundle 中存放的数据与 onCreate()中存放的数据还是不同的。

Inflate()方法有三个参数：

layout 的资源 ID。

存放 fragment 的 layout 的 ViewGroup。

布尔型数据表示是否在创建 fragment 的 layout 期间，把 layout 附加到 container 上（在这个例子中，因为系统已经把 layout 插入到 container 中了，所以值为 false，如果为 true 会导至在最终的 layout 中创建多余的 ViewGroup）。

下面讲述如何把它添加到 activity 中。把 fragment 添加到 activity 一般情况下，fragment 把它的 layout 作为 activitiy 的 loyout 的一部分合并到 activity 中，有两种方法将一个 fragment 添加到 activity 中：

方法一：在 activity 的 layoutxml 文件中声明 fragment

如下代码，一个 activity 中包含两个 fragment：

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

以上代码中，<fragment>中声明一个 fragment。当系统创建上例中的 layout 时，它实例化每一个 fragment，然后调用它们的 onCreateView()方法，以获取每个 fragment 的 layout。系统把 fragment 返回的 view 对象插入到<fragment>元素的位置，直接代替<fragment>元素。

注：每个 fragment 都需要提供一个 ID，系统在 activity 重新创建时用它来恢复 fragment，也可以用它来操作 fragment 进行其它的事物，比如删除它。有三种方法给 fragment 提供 ID：

为 Android:id 属性赋一个数字。

为 Android:tag 属性赋一个字符串。

如果没有使用上述任何一种方法，系统将使用 fragment 的容器的 ID。

方法二：在代码中添加 fragment 到一个 ViewGroup

这种方法可以在运行时，把 fragment 添加到 activity 的 layout 中。只需指定一个要包含 fragment 的 ViewGroup。为了完成 fragment 的事务（比如添加，删除，替换等），必须使用 FragmentTransaction 的方法。可以从 activity 获

取到 FragmentTransaction，如下：

```
FragmentManagerfragmentManager =getFragmentManager()
FragmentTransactionfragmentTransaction=fragmentManager.beginTransaction();
```

然后可以用 add()方法添加一个 fragment，它有参数用于指定容纳 fragment 的 ViewGroup。如，Add()的第一个参数是容器 ViewGroup，第二个是要添加的 fragment。一旦通过 FragmentTransaction 对 fragment 做出了改变，必须调用方法 commit()提交这些改变。不仅在无界面的 fragment 中，在有界面的 fragment 中也可以使用 tag 来作为为一标志，这样在需要获取 fragment 对象时，要调用 findFragmentTag()。

1.3.2 添加没有界面的 Fragment

上面演示了如何添加 fragment 来提供界面，然而，也可以使用 fragment 为 activity 提供后台的行为而不用显示 fragment 的界面。要添加一个没有界面的 fragment，需在 activity 中调用方法 add(Fragment,String)（它支持用一个唯一的字符串做为 fragment 的“tag”，而不是 viewID）。这样添加的 fragment 由于没有界面，所以在实现它时不需调用实现 onCreateView()方法。

使用 tag 字符串来标识一个 fragment 并不是只能用于没有界面的 fragment 上，也可以把它用于有界面的 fragment 上，但是，如果一个 fragment 没有界面，tag 字符串将成为它唯一的选择。获取以 tag 标识的 fragment，需使用方法 findFragmentByTab()。

1.4 Frament 管理

要管理 fragment，需使用 FragmentManager，要获取它，需在 activity 中调用方法 getFragmentManager()。

可以用 FragmentManager 来做以上事情：

使用方法 findFragmentById()或 findFragmentByTag()，获取 activity 中已存在的 fragment

使用方法 popBackStack()从 activity 的后退栈中弹出 fragment（这可以模拟后退键引发的动作）

用方法 addOnBackStackChangedListerner()注册一个侦听器以监视后退栈的变化

还可以使用 FragmentManager 打开一个 FragmentTransaction 来执行 fragment 的事务，比如添加或删除 fragment。

在 activity 中使用 fragment 的一个伟大的好处是能跟据用户的输入对 fragment 进行添加、删除、替换以及执行其它动作的能力。提交的一组 fragment 的变化叫做一个事务。事务通过 FragmentTransaction 来执行。还可以把每个事务保存在 activity 的后退栈中，这样就可以让用户在 fragment 变化之间导航（跟在 activity 之间导航一样）。

可以通过 FragmentManager 来取得 FragmentTransaction 的实例，如下:

```
FragmentManagerfragmentManager = getFragmentManager();
FragmentTransactionfragmentTransaction =fragmentManager.beginTransaction();
```

一个事务是在同一时刻执行的一组动作（很像数据库中的事务）。可以用 add(),remove(),replace()等方法构成事务，最后使用 commit()方法提交事务。在调用 commint()之前，可以用 addToBackStack()把事务添加到一个后退栈中，这个后退栈属于所在的 activity。有了它，就可以在用户按下返回键时，返回到 fragment 执行事务之前的状态。如下例：演示了如何用一个 fragment 代替另一个 fragment，同时在后退栈中保存被代替的 fragment 的状态。

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

    //Replace whatever is in the fragment_container view with thisfragment,
    //and add the transaction to the backstack
    transaction.replace(R.id.fragment_container,newFragment);
```

```
transaction.addToBackStack(null);

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

解释：newFragment 代替了控件 IDR.id.fragment_container 所指向的 ViewGroup 中所含的任何 fragment。然后调用 addToBackStack()，此时被代替的 fragment 就被放入后退栈中，于是当用户按下返回键时，事务发生回溯，原先的 fragment 又回来了。

如果向事务添加了多个动作，比如多次调用了 add(),remove()等之后又调用了 addToBackStack()方法，那么所有的在 commit()之前调用的方法都被作为一个事务。当用户按返回键时，所有的动作都被反向执行（事务回溯）。

事务中动作的执行顺序可随意，但要注意以下两点：

必须最后调用 commit()

如果添加了多个 fragment，那么它们的显示顺序跟添加顺序一至（后显示的覆盖前面的）

如果在执行的事务中有删除 fragment 的动作，而且没有调用 addToBackStack()，那么当事务提交时，那些被删除的 fragment 就被销毁了。反之，那些 fragment 就不会被销毁，而是处于停止状态。当用户返回时，它们会被恢复。

但是，调用 commit()后，事务并不会马上执行。它会在 activity 的 UI 线程（其实就是主线程）中等待直到线程能执行的时候才执行（废话）。如果必要，可以在 UI 线程中调用 executePendingTransactions()方法来立即执行事务。但一般不需这样做，除非有其它线程在等待事务的执行。

注意：只能在 activity 处于可保存状态的状态时，比如 running 中，onPause()方法和 onStop()方法中提交事务，否则会引发异常。这是因为 fragment 的状态会丢失。如果要在可能丢失状态的情况下提交事务，请使用 commitAllowingStateLoss()。

1.5 Fragment 与 Activity 通讯

尽管 fragment 的实现是独立于 activity 的，可以被用于多个 activity，但是每个 activity 所包含的是同一个 fragment 的不同的实例。Fragment 可以调用 getActivity()方法很容易的得到它所在的 activity 的对象，然后就可以查找 activity 中的控件们（findViewById()）。例如：

ViewlistView =getActivity().findViewById(R.id.list);同样的，activity 也可以通过 FragmentManager 的方法查找它所包含的 frament 们。

例如：

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

有时，可能需要 fragment 与 activity 共享事件。一个好办法是在 fragment 中定义一个回调接口，然后在 activity 中实现之。例如，还是那个新闻程序的例子，它有一个 activity，activity 中含有两个 fragment。fragmentA 显示新闻标题，fragmentB 显示标题对应的内容。fragmentA 必须在用户选择了某个标题时告诉 activity，然后 activity 再告诉 fragmentB，fragmentB 就显示出对应的内容。

如下例，OnArticleSelectedListener 接口在 fragmentA 中定义：

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

然后 activity 实现接口 OnArticleSelectedListener，在方法 onArticleSelected()中通知 fragmentB。当 fragment 添加到 activity 中时，会调用 fragment 的方法 onAttach()，这个方法中适合检查 activity 是否实现了

OnArticleSelectedListener 接口，检查方法就是对传入的 activity 的实例进行类型转换，如下所示：

```
public static class FragmentA extends ListFragment{
 OnArticleSelectedListenermListener;
 ...
 @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 没有实现那个接口，fragment 抛出 ClassCastException 异常。如果成功了，mListener 成员变量保存 OnArticleSelectedListener 的实例。于是 fragmentA 就可以调用 mListener 的方法来与 activity 共享事件。例如，如果 fragmentA 是一个 ListFragment，每次选中列表的一项时，就会调用 fragmentA 的 onListItemClick()方法，在这个方法中调用 onArticleSelected()来与 activity 共享事件，如下：

```
public static class FragmentA extends ListFragment{
 OnArticleSelectedListenermListener;
 ...
 @Override
 public void onListItemClick(ListViewl,Viewv,intposition,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);
 }
```

onListItemClick()传入的参数 id 是列表的被选中的行 ID，另一个 fragment 用这个 ID 来从程序的 ContentProvider 中取得标题的内容。

1.6 Fragment 示例

fragment 们可以向 activity 的菜单（按 Manu 键时出现的东西）添加项，同时也可向动作栏（界面中顶部的那个区域）添加条目，这都需通过实现方法 onCreateOptionManu()来完成。

从 fragment 添加到菜单的任何条目，都会出现在现有菜单项之后。Fragment 之后可以通过方法 onOptionsItemSelected()来响应自己的菜单项被选择的事件。也可以在 fragemnt 中注册一个 view 来提供快捷菜单（上下文菜单）。当用户要打开快捷菜单时，fragment 的 onCreateContextMenu()方法会被调用。当用户选择其中一项时，fragemnt 的 onContextItemSelected()方法会被调用。

注意：尽管的 fragment 可以分别收到它所添加的菜单项的选中事件，但是 activity 才是第一个接收这些事件的家伙，只有当 activity 对某个事件置之不理时，fragment 才能接收到这个事件，对于菜单和快捷菜单都是这样。

下例中实验了之前所讲的所有内容。此例有一个 activity，其含有两个 fragment。一个显示莎士比亚剧的播放曲目，另一个显示选中曲目的摘要。此例还演示了如何跟据屏幕大小配置 fragment。

MainActivity:

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

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

Layout.xml:

```
<LinearLayoutxmlns: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" />
  <FrameLayoutAndroid:id="@+id/details" Android:layout_weight="1"
        Android:layout_width="0px" Android:layout_height="match_parent"
        Android:background="?Android:attr/detailsElementBackground" />
  </LinearLayout>
```

系统在 activity 加载此 layout 时初始化 TitlesFragment（用于显示标题列表），TitlesFragment 的右边是一个 FrameLayout,用于存放显示摘要的 fragment,但是现在它还是空的,fragment 只有当用户选择了一项标题后，摘要 fragment 才会被放到 FrameLayout 中。

然而，并不是所有的屏幕都有足够的宽度来容纳标题列表和摘要。所以，上述 layout 只用于横屏，现把它存放于 res/layout-land/fragment_layout.xml。

之外，当用于竖屏时，系统使用下面的 layout，它存放于 res/layout/fragment_layout.xml：

```
<FrameLayoutxmlns: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>
```

这个 layout 只包含 TitlesFragment。这表示当使用竖屏时，只显示标题列表。当用户选中一项时，程序会启动一个新的 activity 去显示摘要，而不是加载第二个 fragment。下一步，会看到 Fragment 类的实现。第一个是 TitlesFragment，它从 ListFragment 派生，大部分列表的功能由 ListFragment 提供。

当用户选择一个 Title 时,代码需要做出两种行为,一种是在同一个 activity 中显示创建并显示摘要 fragment，另一种是启动一个新的 activity。

```
public static class TitlesFragment extends ListFragment {
  booleanmDualPane;
  intmCurCheckPosition = 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.
     */
    voidshowDetails(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.
            FragmentTransactionft = getFragmentManager().beginTransaction();
            ft.replace(R.id.details, 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 fragment，DetailsFragment 显示被选择的 Title 的摘要：

```
public static class DetailsFragment extends Fragment {
/**
 * Create a new instance of DetailsFragment, initialized to
 * show the text at 'index'.
 */
public static DetailsFragmentnewInstance(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;
}

publicintgetShownIndex() {
    returngetArguments().getInt("index", 0);
}

@Override
public View onCreateView(LayoutInflaterinflater, 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;
        }

        ScrollViewscroller = 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()]);
        returnscroller;
    }
}
```

如果当前的 layout 没有 R.id.detailsView（它被用于 DetailsFragment 的容器），那么程序就启动 DetailsActivity 来显示摘要。下面是 DetailsActivity，它只是简单地嵌入 DetailsFragment 来显示摘要。

```
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();
        }
    }
}
```

注意：这个 activity 在检测到是竖屏时会结束自己，于是主 activity 会接管它并显示出 TitlesFragment 和 DetailsFragment。这可以在用户在竖屏时显示在 TitleFragment，但用户旋转了屏幕，使显示变成了横屏。

二、Android Fragment 示例讲解一

By 1026438521

2.1 Fragment 的简介

Fragment 表现 Activity 中用 UI 的一个行为或者一部分. 可以组合多个 fragment 放在一个单独的 activity 中来创建一个多界面区域的 UI,并可以在多个 activity 里重用某一个 fragment.把 fragment 想象成一个 activity 的模块化区域, 有它自己的生命周期, 接收属于它的输入事件, 并且可以在 activity 运行期间添加和删除。

Fragment 必须总是被嵌入到一个 activity 中, 它们的生命周期直接被其所属的宿主 activity 的生命周期影响. 例如, 当 activity 被暂停,那么在其中的所有 fragment 也被暂停; 当 activity 被销毁, 所有隶属于它的 fragment 也被销毁。

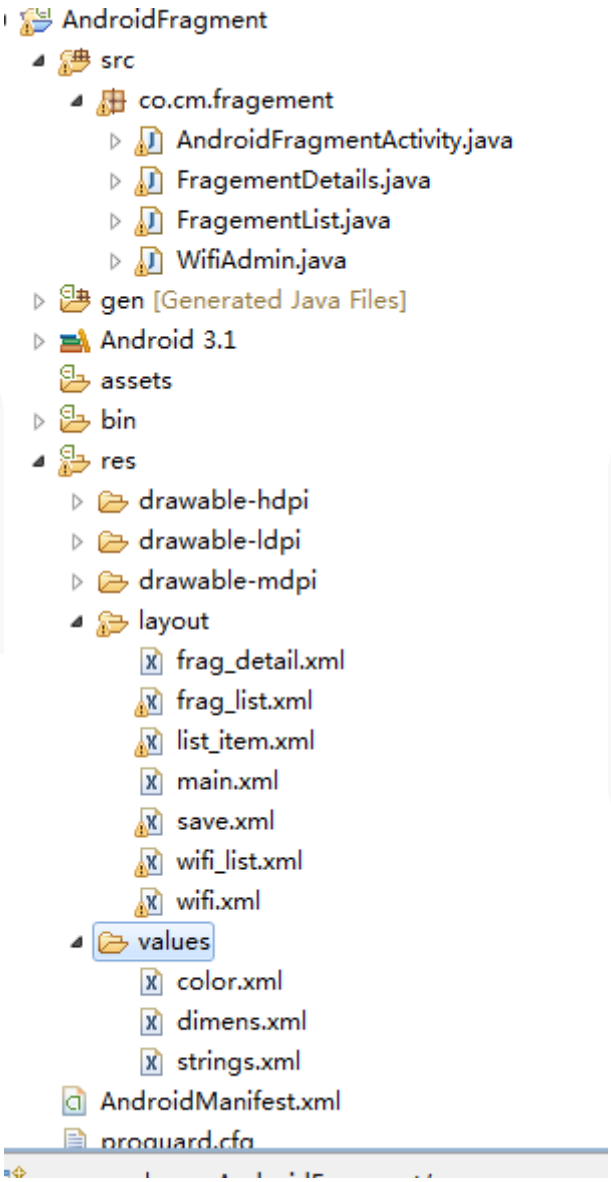
然而,当一个 activity 正在运行时(处于 resumed 状态), 我们可以独立地操作每一个 fragment, 比如添加或删除它们. 当处理这样一个 fragment 事务时, 也可以将它添加到 activity 所管理的 back stack -- 每一个 activity 中的 back stack 实体都是一个发生过的 fragment 事务的记录. back stack 允许用户通过按下 BACK 按键从一个 fragment 事务后退(往后导航)。

将一个 fragment 作为 activity 布局的一部分添加进来时, 它处在 activity 的 view hierarchy 中的 ViewGroup 中, 并且定义有它自己的 view 布局.通过在 activity 的布局文件中声明 fragment 来插入一个 fragment 到你的 activity 布局中, 或者可以写代码将它添加到一个已存在的 ViewGroup.然而, fragment 并不一定必须是 activity 布局的一部分; 也可以将一个 fragment 作为 activity 的隐藏的后台工作者。

Android 在 3.0 中引入了 fragments 的概念,主要目的是用在大屏幕设备上--例如平板电脑上,支持更加动态和灵活的 UI 设计. 平板电脑的屏幕要比手机的大得多,有更多的空间来放更多的 UI 组件,并且这些组件之间会产生更多的交互.Fragment 允许这样的一种设计,而不需要你亲自来管理 view hierarchy 的复杂变化. 通过将 activity 的布局分散到 fragment 中, 你可以在运行时修改 activity 的外观, 并在由 activity 管理的 back stack 中保存那些变化。

其实 fragment 跟 tab 差不多，但是它更灵活，因为程序员可以布局，我的这个 demo 灵感来自我的平板的”设置”程序。由于时间有限，我这里就举两个例子，大家如果有兴趣，可以往下扩展。

2.2 项目结构与内容分析




```
<!-- 主页面 -->
    <!-- 左边页面 -->

    <fragment
        Android:id="@+id/frag_list"
        Android:name="co.cm.fragement.FragementList"
        Android:layout_width="fill_parent"
        Android:layout_height="wrap_content"
        Android:layout_weight="2" />
    <!-- 右面页面 -->
```

```
<fragment
    Android:id="@+id/frag_detail"
    Android:name="co.cm.fragement.FragementDetails"
    Android:layout_width="fill_parent"
    Android:layout_height="wrap_content"
    Android:layout_weight="1" />
```

这个是 mainactivity

```
public class AndroidFragmentActivity extends Activity {
    // 主 activity
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    //初始化 wifi 的各种数据    WifiAdmin.getWifiAdmin().setmContext(AndroidFragmentActivity.this);
        WifiAdmin.getWifiAdmin().getWifiMeathod();
    }
}
```

左面 fragment 界面的代码。。。。

```
public class FragementList extends Fragment {

    TextView wifi; //点击切换到 wifi 存储界面
    ToggleButton toggleButton;//打开关闭 wifi 按钮
    TextView saveBut;//点击切换到 save 存储界面
    FragementDetails frag_detail;//右面 fragment 实例
    boolean isChecked = false; //toggleButton 是否被点击
    boolean butIsRunning = false;//监听 button 状态线程标志位

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        // 在这里初始化 fragment 的页面
        return inflater.inflate(R.layout.frag_list, container, false);
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        // 由于 fragment 不是 activity，不是 oncreated，而是 onActivityCreated
        setView();
        setListener();

        startThread();// 启动控制 button 的线程，当 wifi 状态不是在 1 或者 3 的时候，不可点击，
```

```
        // if (frag != null && frag.isInLayout()) {
        // switch (arg2) {
        // case 0:
        // frag.setText("0000");

    }

    public void setListener() {
        saveBut.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                frag_detail.setSaveShow();

            }
        });
        wifi.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                frag_detail.setWifiShow();
                Log.i("111", WifiAdmin.getWifiAdmin().checkState() + "===-=-");
                checktoggleButton();// 当点回到 wifi 界面时，刷新 button 的状态
            }
        });

        toggleButton.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                Log.i("111", isChecked + "/"
                        + WifiAdmin.getWifiAdmin().checkState());
                if (isChecked) {
                    WifiAdmin.getWifiAdmin().OpenWifi();
                    frag_detail.setWifiShow();
                    // toggleButton.setText("关闭");
                    toggleButton.setChecked(false);
                    isChecked = false;
                } else {
                    WifiAdmin.getWifiAdmin().CloseWife();
                    frag_detail.setWifiShow();
                    // toggleButton.setText("打开");
                    toggleButton.setChecked(true);
                    isChecked = true;
                }

                toggleButton.setClickable(false);
            }
```

```
        });

    }

    //
    public void checktoggleButton() {
        if (WifiAdmin.getWifiAdmin().checkState() == 1) {
            toggleButton.setChecked(true);
            isChecked = true;
        }
        if (WifiAdmin.getWifiAdmin().checkState() == 3) {
            toggleButton.setChecked(false);
            isChecked = false;
        }
    }

    public void setView() {
        wifi = (TextView) getView().findViewById(R.id.wifi);
        toggleButton = (ToggleButton) getView().findViewById(R.id.toggleButton);
        saveBut = (TextView) getView().findViewById(R.id.saveBut);
        // 实例化右面界面，以便操纵里面的方法 F
        frag_detail = (FragementDetails) getFragmentManager().findFragmentById(
                R.id.frag_detail);
        // 初始化 button 的装态
        if (WifiAdmin.getWifiAdmin().checkState() == 3) {
            toggleButton.setChecked(false);
            isChecked = false;
        }
        if (WifiAdmin.getWifiAdmin().checkState() == 1) {
            toggleButton.setChecked(true);
            isChecked = true;
        }
        toggleButton.setClickable(true);

    }

    @Override
    public void onDestroy() {
        frag_detail.stopWifiThread();
        butIsRunning = false;
        super.onDestroy();
    }

    private void startThread() {
        butIsRunning = true;
        new Thread(new Runnable() {
```

```
                @Override
                public void run() {
                    while (butIsRunning) {
//只有 wifi 状态变化完毕之后才能允许点击按钮
                        if (WifiAdmin.getWifiAdmin().checkState() == 3) {
                            if (!isChecked) {
                                toggleButton.setClickable(true);
                            }

                        } else if (WifiAdmin.getWifiAdmin().checkState() == 1) {
                            if (isChecked) {
                                toggleButton.setClickable(true);
                            }
                        }
                    }
                }
            }).start();
        }

}

//右面 fragment 界面类，，，，该类实现了右面显示的操作
public class FragementDetails extends Fragment {
    TextView mac_address, bssid, ip_address, id, info, wifiText;
    ListView listView;
    LinearLayout wifiLinear;
    RelativeLayout save, wifi;
    boolean ThreadFlag = false;
    WifiAdapter wifiAdapter;
    private List<ScanResult> mWifiList = new ArrayList<ScanResult>();// 扫描出的网络连接列表
    private List<WifiConfiguration> mWifiConfiguration = null;// 网络连接列表
    private int nowWifiState = 0;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        return inflater.inflate(R.layout.frag_detail, container, false);
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onActivityCreated(savedInstanceState);
        setView();
        // setListener();
        setWifiShow();
```

```
    }

    public void setWifiShow() {
//通过隐藏显示来达到不容页面内容的切换
        save.setVisibility(View.GONE);
        wifi.setVisibility(View.VISIBLE);
        stopWifiThread();
        refreshWifi();

    }

    public void setSaveShow() {
        stopWifiThread();
        save.setVisibility(View.VISIBLE);
        wifi.setVisibility(View.GONE);
    }

    public void setView() {
        // ----------wifi----------
        wifiText = (TextView) getView().findViewById(R.id.wifiText);
        mac_address = (TextView) getView().findViewById(R.id.mac_address);
        bssid = (TextView) getView().findViewById(R.id.bssid);
        ip_address = (TextView) getView().findViewById(R.id.ip_address);
        id = (TextView) getView().findViewById(R.id.id);
        info = (TextView) getView().findViewById(R.id.info);
        listView = (ListView) getView().findViewById(R.id.listview);
        wifiLinear = (LinearLayout) getView().findViewById(R.id.wifiLinear);
        save = (RelativeLayout) getView().findViewById(R.id.save);
        wifi = (RelativeLayout) getView().findViewById(R.id.wifi);
        wifiAdapter = new WifiAdapter();
        listView.setAdapter(wifiAdapter);
    }

    private Handler handler = new Handler() {

        @Override
        public void handleMessage(Message msg) {
            nowWifiState = WifiAdmin.getWifiAdmin().checkState();
            // 当 wifi 打开时，刷新 wifi 列表的内容
            if (nowWifiState == 3) {
                mWifiList = WifiAdmin.getWifiAdmin().GetWifiList();
                // 如果刚开始检测的 wifi 列表为空，则创建一个实例化的 wifi 而不是 null，负责会在 adpter 里面
报错
                if (mWifiList != null) {
                    // 如果 wifi 列表发生改变，则更新，else，不更新 s
                    if (!mWifiList.toString().equals(
```

```
                        WifiAdmin.getWifiAdmin().getLastWifiList()
                                .toString())) {
                    WifiAdmin.getWifiAdmin().setLastWifiList(mWifiList);
                    wifiAdapter.notifyDate();
                }
            } else {
                mWifiList = new ArrayList<ScanResult>();
            }

        }
        refreshMeathod();

        super.handleMessage(msg);
    }

};

// 刷新 wifi 的状态
public void refreshWifi() {

    new Thread(new Runnable() {

        @Override
        public void run() {
            ThreadFlag = true;
            while (ThreadFlag) {
                // Log.i("111", WifiAdmin.getWifiAdmin().checkState() +
                // "!!!");
                Message msg = handler.obtainMessage();
                handler.sendMessage(msg);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }

        }
    }).start();

}

public void refreshMeathod() {
    // Log.i("111", "refreshMeathod");
    // 此处可用 switch
    if (nowWifiState == 3) {
```

```
            // Log.i("111", "checkState==3");
            // stopWifiThread();
            wifiLinear.setVisibility(View.VISIBLE);
            wifiText.setVisibility(View.INVISIBLE);
            mac_address.setText(WifiAdmin.getWifiAdmin().GetMacAddress() + "");
            bssid.setText(WifiAdmin.getWifiAdmin().GetBSSID() + "");
            ip_address.setText(WifiAdmin.getWifiAdmin().GetIPAddress() + "");
            id.setText(WifiAdmin.getWifiAdmin().GetNetworkId() + "");
            info.setText(WifiAdmin.getWifiAdmin().GetWifiInfo() + "");
            // WifiAdmin.getWifiAdmin().StartScan();
            // if (WifiAdmin.getWifiAdmin().GetWifiList() != null) {
            // mWifiList = WifiAdmin.getWifiAdmin().GetWifiList();
            // }
            // mWifiConfiguration = WifiAdmin.getWifiAdmin()
            // .getmWifiConfiguration();

            // Log.i("111", mWifiList + "===");

        } else if (nowWifiState == 1) {
            // Log.i("111", "checkState==1");
            wifiText.setVisibility(View.VISIBLE);
            wifiLinear.setVisibility(View.INVISIBLE);
            wifiText.setText("要查看可用的网络，请打开 wifi");
        } else if (nowWifiState == 2) {
            // Log.i("111", "checkState==2");
            wifiText.setVisibility(View.VISIBLE);
            wifiLinear.setVisibility(View.INVISIBLE);
            wifiText.setText("wifi 正在打开");
        } else if (nowWifiState == 4) {
            // Log.i("111", "checkState==4");
            wifiText.setVisibility(View.VISIBLE);
            wifiLinear.setVisibility(View.INVISIBLE);
            wifiText.setText("wifi 正在关闭");
        } else {
            // Log.i("111", "checkState==0");
            wifiText.setVisibility(View.VISIBLE);
            wifiLinear.setVisibility(View.INVISIBLE);
            wifiText.setText("我不知道 wifi 正在做什么");
        }
    }

    public void stopWifiThread() {
        ThreadFlag = false;
    }

    public class WifiAdapter extends BaseAdapter {
```

```
@Override
public int getCount() {
    // TODO Auto-generated method stub
    // return mWifiList.size();
    return mWifiList.size();

}

@Override
public Object getItem(int position) {
    // TODO Auto-generated method stub
    return mWifiList.get(position);
    // return mWifiList.get(position);
}

@Override
public long getItemId(int position) {
    // TODO Auto-generated method stub
    return position;
}

@Override
public View getView(int position, View convertView, ViewGroup parent) {
    View view = convertView;

    final ChatViewHolder vh;

    if (convertView == null) {
        vh = new ChatViewHolder();
        view = View.inflate(WifiAdmin.getWifiAdmin().getmContext(),
                R.layout.wifi_list, null);
        vh.wifi_name = (TextView) view.findViewById(R.id.wifi_name);

        vh.wifi_name_state = (TextView) view
                .findViewById(R.id.wifi_name_state);

        view.setTag(vh);
    } else {
        vh = (ChatViewHolder) view.getTag();
    }
    vh.wifi_name.setText(mWifiList.get(position).SSID.toString());// 网络的名字，唯一区别 WIFI 网络的名
字

    vh.wifi_name_state.setText(mWifiList.get(position).level + "");
    return view;
}

public void notifyDate() {
```

```
                notifyDataSetChanged();
            }

        }

        public class ChatViewHolder {
            TextView wifi_name;// 网络的名字，唯一区别 WIFI 网络的名字
            TextView wifi_name_state;// 所发现的 WIFI 网络信号强度
        }

//wifiAdmin 提供了 wifi 操作的方法，

public class WifiAdmin {
    private static WifiAdmin wifiAdmin;
    private WifiManager mWifiManager = null;
    private WifiInfo mWifiInfo = null;
    private List<ScanResult> mWifiList = new ArrayList<ScanResult>();// 扫描出的网络连接列表
    private List<ScanResult> lastWifiList = new ArrayList<ScanResult>();// 扫描出的网络连接列表
    private List<WifiConfiguration> mWifiConfiguration = null;// 网络连接列表
    private WifiLock mWifiLock = null;
    Context mContext;
    private int lastWifiState = 0;// 上次网络状态


下面的 getter，setter 方法大家就自己实现吧
```

2.3 总结一下：

当我们需要在一个界面中处理很多事情的时候，可以推荐使用 fragment，因为他会把我们的 activity 分割成很多小块，每个小块都有他的生命周期，非常方便，而有时我会用单例模式来存储每个页面都有的东西。大家不如继续我的 demo 把“设置”完善下去

三、Android Fragment 示例讲解二

By 维王

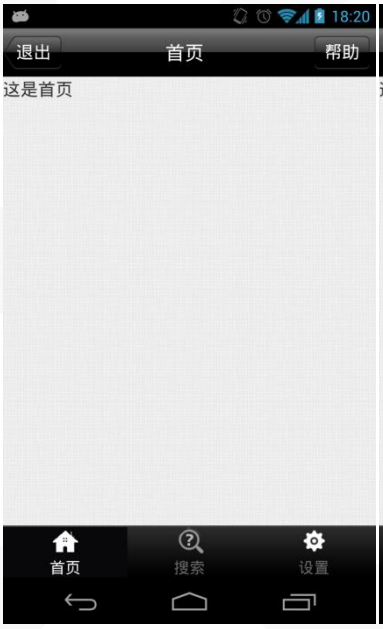
3.1 项目的效果图

下面是 fragment 的简单应用（fragment 的介绍官网上有很详细的说明，这里就不多说了，直接上效果图和代码说明）。

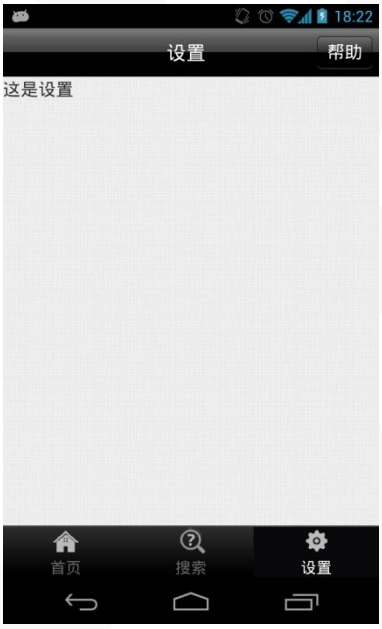
功能：点击底部导航栏，切换到相应的页面。

继承Fragment类时，引用Android-support-v4.jar(在sdk根目录下\extras\Android\support\v4文件夹下)。整体效果图如下：

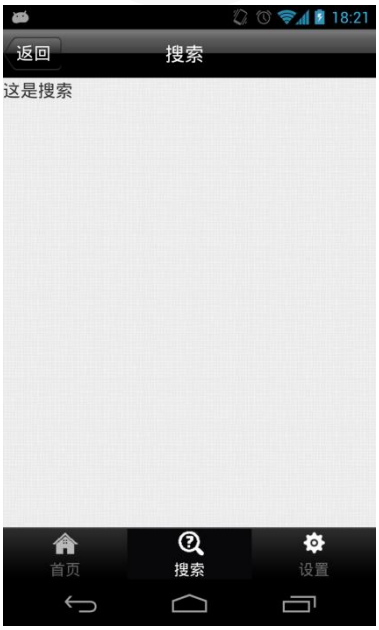
（图一）



（图二）

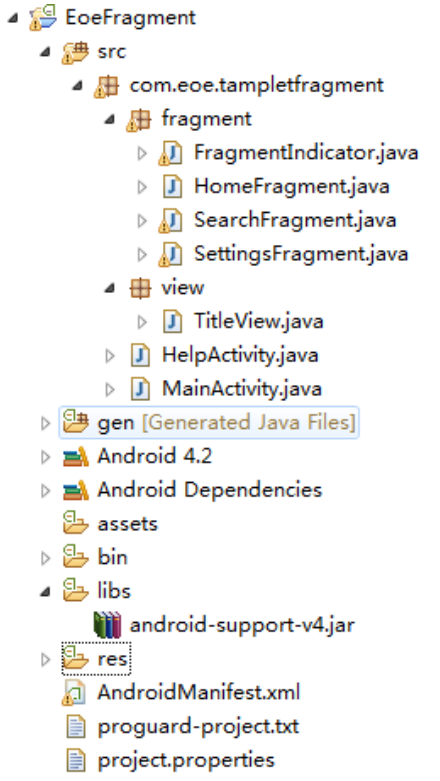


（图三）

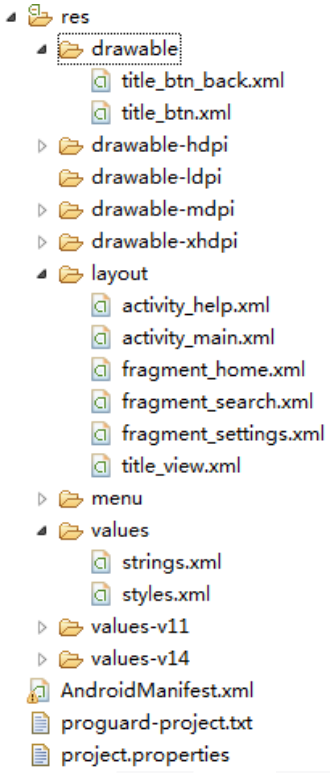


3.2 项目结构图与内容分析：

（图四）



（图五）



主页面的组成（图一）

MainActivity.java（com.eoe.tampletfragment 包下）

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:layout_width="fill_parent"
    Android:layout_height="fill_parent"
    Android:background="@drawable/activity_bg"
    Android:orientation="vertical" >

    <fragment
        Android:id="@+id/fragment_home"
        Android:layout_width="fill_parent"
        Android:layout_height="fill_parent"
        Android:layout_weight="1"
       class="com.eoe.tampletfragment.fragment.HomeFragment" />

    <fragment
        Android:id="@+id/fragment_search"
        Android:layout_width="fill_parent"
        Android:layout_height="fill_parent"
```

```
        Android:layout_weight="1"
        class="com.eoe.tampletfragment.fragment.SearchFragment" />

    <fragment
        Android:id="@+id/fragment_settings"
        Android:layout_width="fill_parent"
        Android:layout_height="fill_parent"
        Android:layout_weight="1"
        class="com.eoe.tampletfragment.fragment.SettingsFragment" />

    <com.eoe.tampletfragment.fragment.FragmentIndicator
        Android:id="@+id/indicator"
        Android:layout_width="fill_parent"
        Android:layout_height="wrap_content"
        Android:background="@drawable/tab_footer_bg" />
</LinearLayout>
```

主页面 MainActivity 继承 FragmentActivity 类，负责实现导航按钮所对应页面的显示和隐藏。（详细实现见源码）

主页面由底部导航栏和面板组成。

fragment 标签所对应 Fragment 的实现类。
com.eoe.tampletfragment.fragment.FragmentIndicator 标签所对应的是底部导航栏。

例如：

主页底部导航栏由 FragmentIndicator.java（com.eoe.tampletfragment.fragment 包下）实现。
对应布局：

```
<com.eoe.tampletfragment.fragment.FragmentIndicator
        Android:id="@+id/indicator"
        Android:layout_width="fill_parent"
        Android:layout_height="wrap_content"
Android:background="@drawable/tab_footer_bg" />
```

主页面中各个面板所对应的布局引用：

```
    <fragment
        Android:id="@+id/fragment_home"
        Android:layout_width="fill_parent"
        Android:layout_height="fill_parent"
        Android:layout_weight="1"
        class="com.eoe.tampletfragment.fragment.HomeFragment" />

    <fragment
        Android:id="@+id/fragment_search"
        Android:layout_width="fill_parent"
        Android:layout_height="fill_parent"
        Android:layout_weight="1"
```

```
        class="com.eoe.tampletfragment.fragment.SearchFragment" />

    <fragment
        Android:id="@+id/fragment_settings"
        Android:layout_width="fill_parent"
        Android:layout_height="fill_parent"
        Android:layout_weight="1"
        class="com.eoe.tampletfragment.fragment.SettingsFragment" />
```

底部导航栏的实现（首页、搜索、设置）

实现类：FragmentIndicator.java（com.eoe.tampletfragment.fragment 包下）
由代码实现底部导航栏的选中和不选中时的图标和文字的显示状态。
每一个按钮对应一个 Fragment 的实现类。

注：可根据自己的需要增减底部导航栏的数目

3.3 面板的实现（如图一）

以 HomeFragment 为例：HomeFragment.java（com.eoe.tampletfragment.fragment 包下）
fragment_home.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:Android="http://schemas.Android.com/apk/res/Android"
    Android:layout_width="fill_parent"
    Android:layout_height="fill_parent"
    Android:orientation="vertical" >

    <com.eoe.tampletfragment.view.TitleView
        Android:id="@+id/title"
        Android:layout_width="fill_parent"
        Android:layout_height="wrap_content" />

    <TextView
        Android:id="@+id/fragment_home_text"
        Android:layout_width="fill_parent"
        Android:layout_height="wrap_content"
        Android:text="@string/fragment_home_text"
        Android:textSize="18sp" />
</LinearLayout>
```

实现类 HomeFragment 继承了 *Fragment* 类，通过 onCreateView 方法加载 *fragment_home.xml* 布局。

```
@Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_home, container, false);
        return view;
```

```
}
```

通过在 onActivityCreated 方法中实例化控件。在加载控件时注意要引用 getView()的全局变量实例化控件。（首先在类中声明成员变量

```
private View mParent;
    private FragmentActivity mActivity;）
@Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        mActivity = getActivity();
    mParent = getView();

        mTitle = (TitleView) mParent.findViewById(R.id.title);
        mTitle.setTitle(R.string.title_home);
        mTitle.setLeftButton(R.string.exit, new OnLeftButtonClickListener(){

            @Override
            public void onClick(View button) {
                mActivity.finish();
            }

        });
        mTitle.setRightButton(R.string.help, new OnRightButtonClickListener() {

            @Override
            public void onClick(View button) {
                goHelpActivity();
            }
        });

        mText = (TextView) mParent.findViewById(R.id.fragment_home_text);

}
```

如上：
实例化一个 TextView 控件

```
TextView tv=(TextView)=mParent.findViewById(R.id.fragment_home_text);
```

同样，实例化自定义标题栏。

右按钮实现页面跳转功能：

```
private void goHelpActivity() {
        Intent intent = new Intent(mActivity, HelpActivity.class);
        startActivity(intent);
}
```

注：

在跳转页面时，Intent 实例化时，引用的是 FragmentActivity 的全局变量 mActivity，而不是当前 Fragment 实现类的 this。

面板之间的切换
以 SearchFragment 为例：
SearchFragment.java（com.eoe.tampletfragment.fragment 包下）

点击搜索页面标题栏的“返回”按钮，实现返回“首页”的功能（实现到某个特定页面的切换）。

```
/**
     * 返回到首页
     * 跳转到某个特定的 fragment 页面的实现类
     */
    private void backHomeFragment() {
        getFragmentManager().beginTransaction()
                .hide(MainActivity.mFragments[1])
                .show(MainActivity.mFragments[0]).commit();
        FragmentIndicator.setIndicator(0);
}
```

注：
页面切换完毕之后，一定要记得选中相应的底部导航栏按钮。

DialogFragment（父类是 fragment）
DialogFragment 类显示一个浮动的对话框。
用这个类来创建一个对话框，是使用在 Activity 类的对话框工具方法之外的一个好的选择，
因为可以将一个 fragment 对话框合并到 activity 管理的 fragment back stack 中，允许用户返回到一个之前曾被摒弃的 fragment。

可以用 DialogFragment 的实现类自定义对话框。

四、Android Fragment 示例讲解 三

By liangpingyy

4.1 开发概述

自从 Android 引入 Fragmnets 后，应用开发的布局将更加灵活，且不用只局限于一种分辨率条件下进行开发。本文就以 Android 短信收发为例进行讲解。

采用两个 Fragment 分别负责联系人界面和聊天界面，两个 Fragment 居于同一 Activity 中，可以进行相互通信，并可与主 Activity 进行通信。用户可以在选择联系人后，输入发送内容，点击发送按钮将短信内容发给目标用户，同时也可以自动接收其他用户所发出的短信。其界面如图 1 所示，左边为联系人列表，右边为收发短信界面。

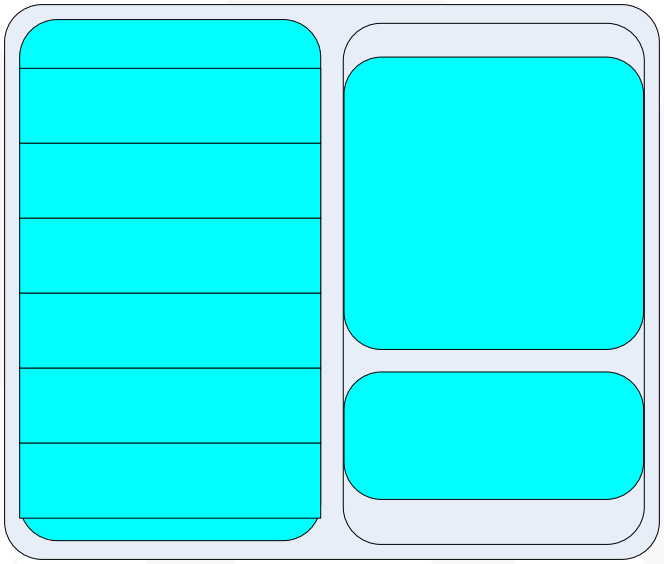


图 1 短信收发布局

4.2 技术要点

本次开发目的是给初学者一些指导，所以所涉及的要点比较少，主要体现在以下几个方面：

①Fragment 布局嵌入，在 Fragment 中可以动态嵌入布局也可以直接在 Fragment 的 onCreateView()中嵌入；

②静态上下文对象获取，在使用 Toast 时，可能会使用静态上下文对象，因此需要获取静态 Context；

③短信发送，通过 SmsManager 将短信发送到目标手机中；

④短信接收，若设备中有新的短信进入，可进行获取，并能将短信内容添加到聊天界面的聊天窗口中；

⑤数据共享，Fragment 中列表点击事件后，需要将所点击联系人的列表信息保存，以便短信发送使用。

4.3 开发实例

作为一个 Android 开发者，最感兴趣的还是在自己的 APP 中如何使用，下面将介绍一下 Fragments 的使用方法。

4.3.1Fragments 使用示例

本文以短信聊天界面作为示例讲解一下 Fragments 的用法，在这里需要注意几点：
第一，demo 所使用的版本为 Android3.0；
第二，由于本人现在还没有 3.0 以上的真机，调试是在模拟器中进行的；
第三，demo 中的联系人与电话号码都是事先设立好的，没有调用联系人列表，感兴趣的网友可以自己改一下。
进入正题，首先讲解一下布局

4.3.1.1 布局

Demo 中有三个布局文件：activity_main、chat_fragment 和 fragment1，其中 activity_main 为主 Activity，里面包含两个 fragment，左边的显示联系人列表，右边显示短信聊天内容。Fragment 布局如下：

```
<fragment
Android:id="@+id/fragment1"
Android:name="com.example.testfragments.Fragment1"
Android:layout_width="0px"
Android:layout_height="match_parent"
Android:layout_weight="1"
/>
```

注：name 里面的内容为开发者自己定义的 Fragment 类。
fragment1 中为联系人列表，布局文件中只有一个 ListView，嵌入到第一个 fragment 中。
Chat_fragment 中由两个 EditText 和一个 Button 组成，嵌入到第二个 fragment 中。

4.3.1.2 在 Fragment 中嵌入布局文件

demo 中所采用的嵌入布局文件方法主要是在 onCreateView()中将布局文件嵌入进去的，其实现代码如下：

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,  Bundle savedInstanceState) {
        // TODO Auto-generated method stub
View chatView=inflater.inflate(R.layout.chat_fragment, container,false);
    return chatView;
    }
```

4.3.1.3 Fragment 中控件与 Activity 通信

在主进程中需要对各个 Fragment 中控件进行监听，以达到各种功能目的。Demo 中在 MainActivity 中直接使用 findViewById()方法。监听按钮按下方法如下：

```
publicvoid sendListener(int buttonId,int etId1,int etId2){
        Button btn_send=(Button)findViewById(buttonId);//获取对应 id 的按钮
        final EditText et_text=(EditText)findViewById(etId2);
        final EditText et_chat=(EditText)findViewById(etId1);
        btn_send.setOnClickListener(new OnClickListener() {//设置监听器
```

```
            @Override
            publicvoid onClick(View v) {
                // TODO Auto-generated method stub
                if("".equals(et_text.getText().toString().trim())){//若用户没有输入短信，则提示
            Toast.makeText(MainActivity.context, "没有输入内容，请输入内容后发送",
Toast.LENGTH_SHORT).show();
                }else{
                    StringBuffer sb_chat=new StringBuffer();
                    String text=et_text.getText().toString();
            sb_chat.append(et_chat.getText()+"\r\n"+MY_NAME+":"+text);
            //利用 StringBuffer 将所发送的短信发送聊天界面中
                    et_chat.setText(sb_chat);
                    sendSMS(text);//发送短信
                }
            }
        });
    }
```

代码说明：本方法是对 fragment2 中按钮进行监听，当按钮按下时，将用户所输入短信内容发送到目标手机中，并在聊天窗口上添加上所发送内容。需要提示一下的是进行判定 EditText 中数据是否为空时，需要与 toString().trim()方法中的数据进行对比。

同样，在 fragment2 中聊天 EditText 内容实时更新页面也是用这种方法：

```
publicvoid getSMS(int etchatId,String text){
        EditText et_chat=(EditText)findViewById(etchatId);//获取对应 id 的 EditText
        StringBuffer sb_chat=new StringBuffer();
        sb_chat.append(et_chat.getText()+"\r\n"+text);
        et_chat.setText(sb_chat);//显示聊天内容
}
```

代码说明：本段代码将接收到的短信直接显示到给定 ID 所对应的 EditText 中。

4.3.1.4 短信发送

Demo 中使用短信发送功能，其代码如下：

```
publicstaticvoid sendSMS(String text){
        SmsManager smsManager=SmsManager.getDefault();
        Intent sIntent=new Intent(SMS_SEND);
        PendingIntent sentIntent=PendingIntent.getBroadcast(context, 0, sIntent, 0);
        smsManager.sendTextMessage(phnumber[conindex], null, text, sentIntent, null);
    }
```

代码说明：Android 中使用 SmsManager 进行短信发送，其中 SMS_SEND 中内容为“com.SMSDemo.SMS.send”，是系统预设。

4.3.1.5 短信及发送状态接收

当有新的短信或者发送状态到手机上时，需要将短信或状态内容读取出来，其读取方法如下：

```
publicclass SMSReceiver extends BroadcastReceiver{//继承广播机制
        @Override
        publicvoid onReceive(Context arg0, Intent arg1) {
            // TODO Auto-generated method stub
            String action=arg1.getAction();//获取广播状态
            int resultCode=getResultCode();
            if(action.equals(SMS_SEND)){//若广播状态为发送短信则进行一下操作
                switch(resultCode){
                case Activity.RESULT_OK://发送短信成功
                    Toast.makeText(context, "SMS is success", Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_GENERIC_FAILURE://发送短信失败
                    Toast.makeText(context, "SMS is failure", Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_NO_SERVICE://无服务
                    Toast.makeText(context, "No service!", Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_NULL_PDU://号码有误
                    Toast.makeText(context, "No such number!", Toast.LENGTH_SHORT).show();
                    break;
                case SmsManager.RESULT_ERROR_RADIO_OFF://手机关机
                    Toast.makeText(context, "The number is shut down!", Toast.LENGTH_SHORT).show();
                    break;
                }
            }elseif(action.equals(SMS_RECEIVE)){//若状态为有可获取短信则进行以下操作进行获取
                Bundle bundle=arg1.getExtras();
                if(bundle!=null){
                    Object[] object=(Object[])bundle.get("pdus");
                    SmsMessage[] messages=new SmsMessage[object.length];
                    for(int i=0;i<object.length;i++){
                messages[i]=SmsMessage.createFromPdu((byte[])object[i]);
                    }
                    SmsMessage message=messages[0];
                    Toast.makeText(context, message.getMessageBody().toString(), Toast.LENGTH_SHORT).show();
                    String text=message.getMessageBody().toString();
                    getSMS(R.id.et_context1,text);
                }
            }
        }
    }
```

代码说明：本段代码使用广播机制进行信息获取，所获取的信息可以是发送短信的状态，也可以是别人所发送到本机的短信，若是短信则将短信获取后通过 `getSMS()` 方法将短信显示在聊天界面上。

4.3.1.6 Error inflating class fragment 错误解决方法

调试过程中遇到以上错误，很崩溃。网上搜索一下，发觉类似错误很多，也有不少网友给出了解决方案。本人遇到的错误解决方法如下：

1. MainActivity 需要继续 FragmentActivity，而不是 Activity；

2. Fragment 类应导入的包为 Android.support.v4.app.Fragment，而不能导入 Android.app.Fragment；

4.3.1.7 小结

Fragment 的出现主要是为了解决屏幕分辨率问题，在大屏幕上可显示多个 Fragment，而小屏幕则显示少量 Fragment，以满足不同需求。

Fragment 的使用步骤主要是：一，布局；二，嵌入 fragment 中；三，监听，实现相应功能。

实现效果

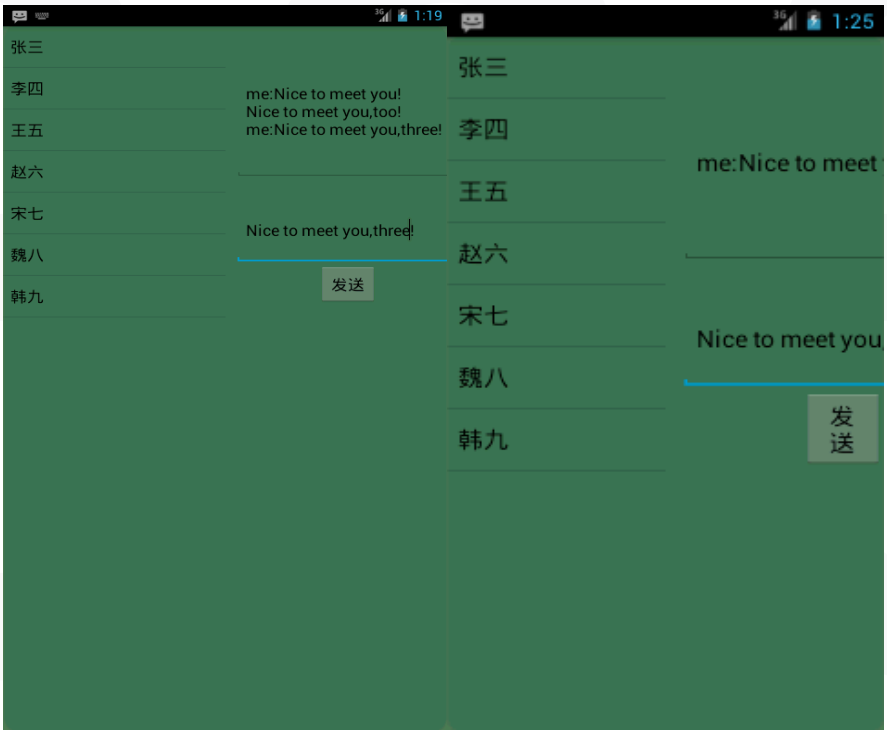


图 4 Fragments 实现效果图

在屏幕不同的模拟器上进行实验，效果如图 4 所示。点击发送按钮后，目标手机将会把短信读取到界面中，实现效果入左图所示。目前还是存在一个问题，就是小屏幕发送到大屏幕上可以正常显示，而大屏幕向小屏幕上发送信息则不能正常接收。

五、社区的最新动态：

【eoeAndroid 社区】抢注博客个性域名送新年勋章

eoe.cn 的博客系统已经上线啦，还没有个性域名的童鞋你就凹凸啦，个性域名是个性、自我的象征，想做一个与众不同的程序猿就要有个个性域名的说，有了个性域名不写点东西就走怎能对得起自己呢。
http://www.eoeandroid.com/thread-252111-1-1.html

【寻找文艺程序员】2012 在 eoe 的日子

将自己的 2012 的时光或者是 2013 的展望发表到 eoe 的海阔天空模块并且同步到 eoe 的博客。
标题以 2012，在 eoe 的日子 开头，可以加上自己的副标题。
http://www.eoeandroid.com/thread-253249-1-1.html

【eoeAndroid 社区】等级勋章新鲜出炉

新年新气象，在这辞旧迎新的时刻，eoeAndroid 社区隆重推出会员等级勋章。
不要再为自己木有勋章而苦苦发愁了，放下申请勋章迟迟不能佩带的怨念吧。
http://www.eoeandroid.com/thread-249165-1-1.html

eoe 移动开发者社区

责任编辑：jiayouhe123
美术支持：徐士勇 技术支持：郝留有
联系 QQ：1033843762 新浪微博@eoe 移动开发者社区