Fragment与ToolBar

1. 本章概述

通过本章的学习，掌握 Fragment 的创建和应用，Activity 与 Fragment 之间的交互，最后掌握 ToolBar 的应用。

2. 本章重点与难点

1）重点

（1）Fragment 的创建和应用。

（2）ToolBar 的应用。

2）难点

（1）Activity 与 Fragment 的综合应用。

（2）ToolBar 的应用。

3. 重难点学习建议

本章主要帮助读者学习并掌握 Android 中非常重要的 Fragment 和 ToolBar 的应用。学习本章时，知识的广度逐渐扩大，建议读者勤于练习，勤于思考，对本教材提供的项目更是要多体会，多实践。想深入了解本章知识的读者可进行课后延伸阅读，加深理解。

5.1 Fragment简介与应用场合

Fragment（碎片）是 Android Honeycomb 3.0 新增的概念，Fragment 代表用户界面中一个活动的部分或者是一种行为能力，它可以有自己单独的用户界面，也可以单独接收用户输入和处理事件的消息。

在原来的设计理念中，手机的屏幕比较小，整个屏幕作为一个整体出现，应用程序运行过程就是在多个屏幕之间切换的过程。但是随着平板电脑的出现，这一切都随之改变。单个应用界面不一定占据整个屏幕。而且单个界面包含的内容更加丰富，许多时候仅需要更新界面的局部，而不是切换整个界面。因此，代表应用界面局部区域，并能单独与用户进行交互的组件 Fragment 应运而生。下面举个例子，以便更好地理解 Fragment 出现的意义。

例如，设计一个读新闻的程序，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身感兴趣的事件。于是，不需要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。现在有了 Fragment 可以把两者放在一个 Activity 上同时显示出来，如图 5.1 所示。

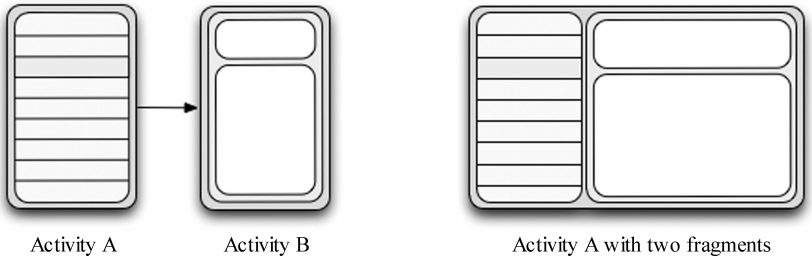

图 5.1 手机显示的 Fragment

由此可以看出，使用 Fragment 可以让用户更加充分地利用平板的屏幕空间。

5.2 创建 Fragment

Fragment 是 Android 3.0 以上版本引入的新事物，默认的包为 android.app.FragmentManager，如果要兼容 3.0 以下版本就需要引入 android.support.v4.app.Fragment 包。这两个包对 Fragment 相同方法的定义有所不同，因此在 import 时，经常容易混淆导致程序报错，如下事项在使用前需要了解。

最低支持版本不同：

(1) android.app.Fragment 兼容的最低版本是 android:minSdkVersion="11"，即 3.0 版。

(2) android.support.v4.app.Fragment 兼容的最低版本是 android:minSdkVersion="4"，即 1.6 版。

(3) 需要导入 jar 包：android.support.v4.app.Fragment，需要引入包：android-support-v4.jar。

如果在 Activity 中要使用 android.app.Fragment，需要满足以下三个条件。

(1) 当前的 Activity 直接继承 Activity。

(2) 在方法中使用 getFragmentManager().findFragmentById(XXX)获取 Fragment 对象。

(3) support.v4 包中的方法一般都会加上 support.，例如获取 fragmentManager，就需要 getSupportFragmentManager，而 app 包中的直接 getFragmentManager 即可。

创建一个 MyFirstFragemt 事例，应该继承 Fragment 或者其子类，覆写相应的方法。Fragment 有两类，一类是具有 UI 界面的，Fragment 作为 ViewGroup 组件的节点增加到当前的 Activity 界面；另一类是没有 UI 界面的，它可能只是作为后台的一个工作线程。

这里创建的 Fragment 有用户界面，因此首先要声明一个用户界面。

【例 5.1】 Fragment 应用案例。

本案例共有三个 Java 文件，其中两个是 Fragment 类的子类，一个是 Activity 的子类，程序能够显示图 5.2 右面的界面，接下来就对案例进行详尽的分析。

图 5.2 Fragment 案例

activity_main.xml 文件是 Activity 的布局文件，两个 Fragment 在界面上的位置关系就是在这个文件中进行的定义。activity_main.xml 代码如下。

```
<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="edu.hrbeu.FragmentDemo.AFragment"
          android:id="@+id/fragment_a"
          android:layout_weight="1"
          android:layout_width="0px"
          android:layout_height="match_parent" />
<fragment android:name="edu.hrbeu.FragmentDemo.BFragment"
          android:id="@+id/fragment_b"
          android:layout_weight="1"
          android:layout_width="0px"
          android:layout_height="match_parent" />
</LinearLayout>
```

FragmentDemoActivity 是该示例主界面的 Activity，加载了 activity_main.xml 文件声明的界面布局。FragmentDemoActivity.java 文件的完整代码如下。

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
public class FragmentDemoActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Android 系统会根据代码“setContentView(R. layout. main);”的内容加载界面布局文件 activity_main. xml，然后通过 activity_main. xml 文件中对 Fragment 所在的“包＋类”的描述，找到 Fragment 的实现类，并调用类中的 onCreateView()函数绘制界面元素。

AFragment. java 文件的核心代码如下。

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

AFragment 中只实现了 onCreateView()函数，返回值是 AFragment 的视图，代码“return inflater. inflate(R. layout. frag_a，container，false);”使用 inflate()函数，通过指定资源文件 R. layout. frag_a 获取 AFragment 的视图。

BFragment. java 文件的核心代码如下。

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

最后，给出 fragment_a. xml 文件的全部代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:orientation="vertical" >
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="AFragment" />
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="这是 AFragment 的显示区域,通过这行文字可以看到与
        BFragment 的边界" />
<CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="AF 选项" />
<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="AF 按钮" />
</LinearLayout>
```

fragment_b. xml 文件的全部代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BAFragment" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="这是 BFragment 的显示区域,通过这行文字可以看到与
        AFragment 的边界" />
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BF 选项" />
    <Button
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="BF按钮" />
</LinearLayout>
```

Fragment也可以通过代码动态添加,下面通过一个例子了解Fragment的动态添加。

【例5.2】 Fragment的动态添加应用案例。

本案例程序能够显示图5.3的界面。

图5.3 Fragment动态添加的案例图

activity_main.xml文件是Activity的布局文件,Fragment将显示在container组件中。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity5_2">
    <LinearLayout
        android:id="@+id/container"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >
</LinearLayout>
</LinearLayout>
```

Fragment的布局文件的代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ContentFragment">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:textSize="32dp"
        android:text="这是ContentFragment界面" />
</FrameLayout>
```

MainActivity 是该示例主界面的 Activity，加载了 activity_main. xml 文件声明的界面布局。MainActivity. java 文件的完整代码如下。

```
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        FragmentManager fragmentManager=this.getFragmentManager();
        FragmentTransaction tx=fragmentManager.beginTransaction();
        //第一个参数是 activity_main 布局文件中的 linearlayout 的 id,第二个参
        //数是 Framgment 的类名
        tx.add(R.id.container1, new ContentFragment());
        tx.commit();
    }
}
```

Fragment 的界面代码如下。

```
import android.app.Fragment;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class ContentFragment extends Fragment {
    public ContentFragment() {
        //Required empty public constructor
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        //Inflate the layout for this fragment
        Log.v("ContentFragment", "onCreateView");
        return inflater.inflate(R.layout.fragment_content, container, false);
    }
}
```

5.3 Fragment生命周期

为了更好地使用Fragment，有必要深入了解Fragment的生命周期。因为Fragment必须嵌入在Acitivity中使用，所以Fragment的生命周期和它所在的Activity是密切相关的。Activity直接影响它所包含的Fragment的生命周期，所以对Activity的某个生命周期方法的调用也会产生对Fragment相同方法的调用。例如，当Activity的onPause()方法被调用时，它所包含的所有Fragment的onPause()方法都会被调用。当然，Fragment比Activity还要多出几个生命周期回调方法，这些额外的方法是为了与Activity的交互而设立的。下面是具体方法的说明，如表5.1所示。

表5.1 Fragment的生命周期方法

方　法	描　述
onAttach()	当Fragment被加入到Activity时调用(在这个方法中可以获得所在的Activity)
onCreate()	完成Fragment的初始化创建
onCreateView()	当Activity要得到Fragment的layout(界面)时，调用此方法，Fragment在其中创建自己的layout
onActivityCreated()	当Activity要得到Fragment的layout时，调用此方法，Fragment在其中创建自己的layout
onStart()	Fragment可见，当主Activity处于started状态后执行
onResume()	Fragment能与用户交互，当主Activity处于resumed状态后执行
onPause()	Fragment不再与用户交互，可能在主Activity将要处于paused前执行，可能该Fragment被修改
onStop()	Fragment不可见，可能在主Activity将要处于stopped前执行，可能该Fragment被修改
onDestroyView()	当Fragment的layout被销毁时被调用，允许该Fragment清理视图相关资源
onDestroy()	清理视图state信息
onDetach()	当Fragment被从Activity中删除时被调用。该Fragment不再与Activity关联

一旦Activity进入resumed状态(也就是running状态)，就可以自由地添加和删除Fragment了。因此，只有当Activity在resumed状态时，Fragment的生命周期才能独立地运转，其他时候是依赖于Activity的生命周期变化的。

【例5.3】 Fragment的生命周期以及与Activity的关系。

```
import android.support.v4.app.FragmentTransaction;
import android.os.Bundle;
import android.support.v4.app.Fragment;
```

```
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
public class MainActivity5_3 extends AppCompatActivity {
    static String Tag="Fragment_Activity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.i(Tag, "in onCreate()......");
        setContentView(R.layout.activity_main5_3);
        addFragment();
        Log.i(Tag, "out onCreate()......");
    }

    @Override
    protected void onDestroy() {
        //TODO Auto-generated method stub
        Log.i(Tag, "in onDestroy()......");
        super.onDestroy();
        Log.i(Tag, "out onDestroy()......");
    }
    void addFragment() {
        Fragment newFragment=new Fragmentlife();
        FragmentTransaction ft=this.getSupportFragmentManager().
        beginTransaction();
       ft.add(android.R.id.content,newFragment,"first");
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN);
        ft.commit();
    }
}
```

Fragmentlife. java 的代码如下。

```
import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class Fragmentlife extends Fragment {
    static String Tag="Fragmentlife";
    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        Log.i(Tag, "in onCreate().....");
        super.onCreate(savedInstanceState);
        Log.i(Tag, "in outCreate().....");
    }
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        Log.i(Tag, "in onActivityCreated().....");
        super.onActivityCreated(savedInstanceState);
        Log.i(Tag, "out onActivityCreated().....");
    }
    @Override
    public void onDestroyView() {
        //TODO Auto-generated method stub
        Log.i(Tag, "in onDestroyView().....");
        super.onDestroyView();
        Log.i(Tag, "out onDestroyView().....");
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
        //TODO Auto-generated method stub
        Log.i(Tag, "in onCreateView().....");
        View v=inflater.inflate(R.layout.fragmentlife, container,false);
        Log.i(Tag, "out onCreateView().....");
        return v;
    }
    @Override
    public void onAttach(Activity activity) {
        //TODO Auto-generated method stub
        Log.i(Tag, "in onAttach().....");
        super.onAttach(activity);
        Log.i(Tag, "out onAttach().....");
    }
}
```

运行程序,切换到 LogCat 子窗口,可以看到图 5.4 所示的结果。