第 5 章

Fragment 基础

学习目标

- 掌握 Fragment 的生命周期。
- 掌握 Fragment 的应用。
- 掌握 Fragment 与 Activity 之间的通信。

随着移动设备的快速发展，平板电脑越来越普及，而平板电脑与手机的最大差别就在于屏幕的大小。为了同时兼顾手机和平板电脑的开发，自 Android 3.0（API 11）开始引入了 Fragment。接下来对 Fragment 进行详细的介绍。

5.1 Fragment 概述

Fragment 翻译为中文就是“碎片”的意思，它是一种嵌入到 Activity 中使用的 UI 片段。一个 Activity 中可以包含一个或多个 Fragment，而且一个 Activity 可以同时展示多个 Fragment。使用它能够让程序更加合理地利用拥有大屏幕空间的移动设备，因此 Fragment 在平板电脑上应用非常广泛。

Fragment 与 Activity 类似，也拥有自己的布局与生命周期，但是它的生命周期会受到它所在的 Activity 的生命周期的控制。例如，当 Activity 暂停时，它所包含的 Fragment 也会暂停；当 Activity 被销毁时，该 Activity 内的 Fragment 也会被销毁；当该 Activity 处于活动状态时，开发者才可独立地操作 Fragment。

为了更加清楚地讲解 Fragment 的功能，接下来通过一个图例来说明，如图 5-1 所示。

从图 5-1 可以看出，在一般的手机或者平板电脑竖屏情况下，Fragment1 需要嵌入到 Activity1 中，Fragment2 需要嵌入到 Activity2 中；如果在平板电脑横屏的情况下，则可以把两个 Fragment 同时嵌入到 Activity1 中，这样的布局既节约了空间，也会更美观。

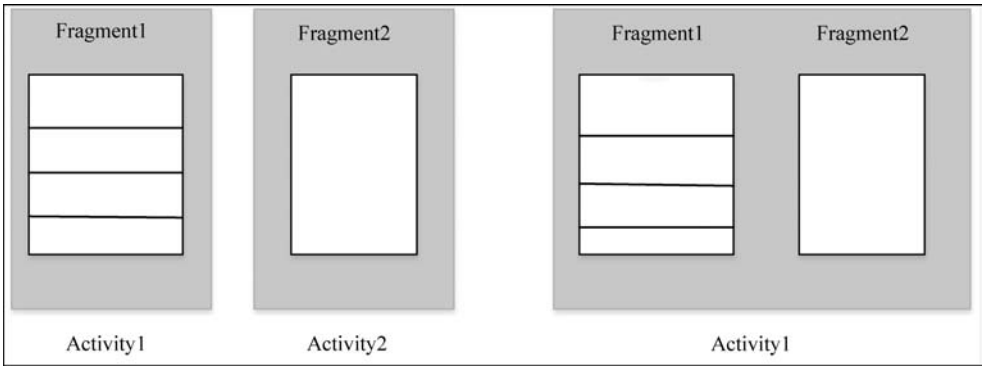

图 5-1　Fragment 的功能

5.2　Fragment 生命周期

通过第 3 章的学习,我们知道 Activity 生命周期有 3 种状态,分别是运行状态、暂停状态和停止状态。Fragment 与 Activity 非常相似,其生命周期也会经历这几种状态。接下来详细介绍这几种状态。

运行状态:当嵌入该 Fragment 的 Activity 处于运行状态时,并且该 Fragment 是可见的,那么该 Fragment 是处于运行状态的。

暂停状态:当嵌入该 Fragment 的 Activity 处于暂停状态时,那么该 Fragment 也是处于暂停状态的。

停止状态:当嵌入该 Fragment 的 Activity 处于停止状态时,那么该 Fragment 也会进入停止状态。或者通过调用 FragmentTranslation 的 remove()、replace()方法将 Fragment 从 Activity 中移除。

Fragment 必须是依存于 Activity 而存在的,因此 Activity 的生命周期会直接影响到 Fragment 的生命周期。图 5-2 很好地说明了两者生命周期的关系。

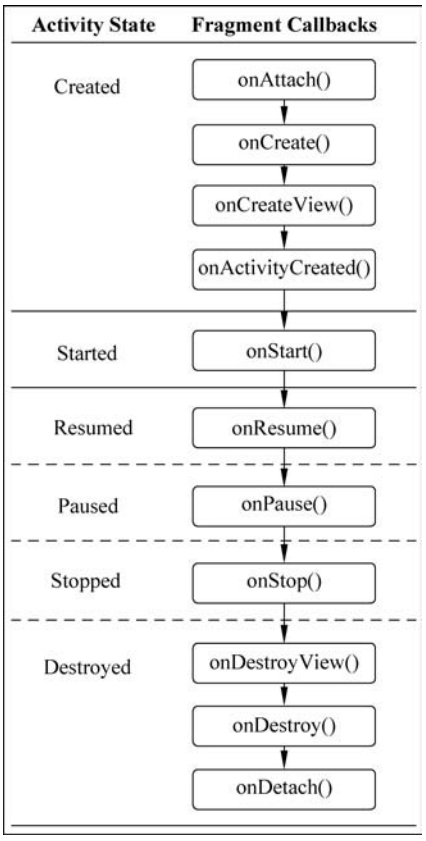

图 5-2　Fragment 和 Activity 生命周期对比图

可以看到,Fragment 比 Activity 多了几个额外的生命周期回调方法。

- onAttach(Activity):当 Fragment 与 Activity 发生关联时调用。
- onCreateView(LayoutInflater,ViewGroup,Bun-

dle)：创建该 Fragment 的视图(加载布局)时调用。

- onActivityCreated(Bundle)：当 Activity(与 Fragment 相关联)的 onCreate 方法返回时调用。
- onDestroyView()：与 onCreateView 相对应，当与该 Fragment 关联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。

以上就是 Fragment 的生命周期与 Activity 的生命周期之间的关系，接下来将讲解如何创建 Fragment 以及 Fragment 之间的通信。

5.3 Fragment 的创建


视频讲解

Fragment 的创建与 Activity 的创建类似，要创建一个 Fragment 必须要创建一个类继承自 Fragment。Android 系统提供了两个 Fragment 类，分别是 android.app.Fragment 和 android.support.v4.app.Fragment。继承前者只能兼容 Android 4.0 以上的系统，继承后者可以兼容更低的版本。接下来将具体讲解 Fragment 的创建过程。

(1) 创建新项目 Chapter5_Fragment，在项目 app 的 java 下面的包中分别创建 LeftFragment、RightFragment 和 SecondFragment，创建 Fragment 的方法如图 5-3 和图 5-4 所示。

图 5-3 选择 Android 提供的 Fragment(Blank)

(2) Fragment 创建时可以自动生成对应的布局文件。修改左侧碎片 LeftFragment 布局文件 fragment_left.xml，代码如下：

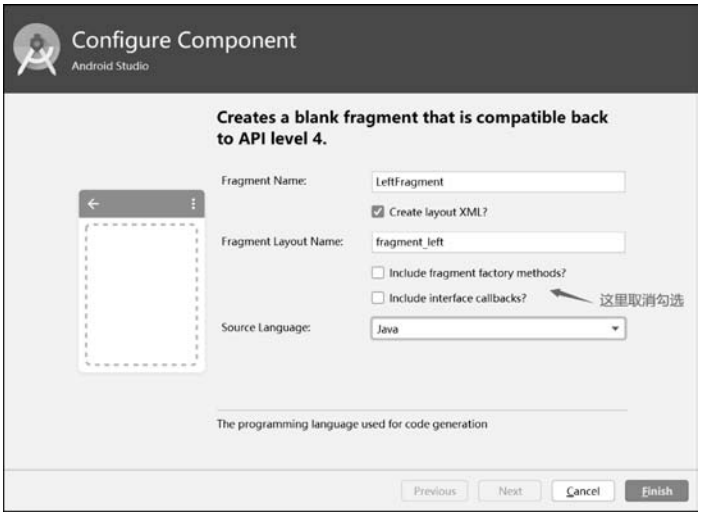

图 5-4　创建自定义的 Fragment

```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:orientation = "vertical" >
    <Button
        android:id = "@ + id/button"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_gravity = "center_horizontal"
        android:text = "点击我"/>
</LinearLayout >
```

(3) 修改右侧碎片 RightFragment 布局文件 fragment_right.xml,代码如下:

```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:background = "@android:color/darker_gray"
    android:orientation = "vertical" >
<ImageView
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:layout_marginLeft = "60dp"
    android:src = "@mipmap/ic_launcher"/>
</LinearLayout >
```

(4) 在 LeftFragment 类中重写 onCreateView()方法,onCreateView()方法通过 LayoutInflater 的 inflate()方法将 fragment_left 布局动态加载进来,代码如下:

```
public class LeftFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.left_fragment, container, false);
        return view;
     }
}
```

(5) 接着修改 RightFragment,代码如下:

```
public class RightFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.right_fragment, container, false);
        return view;
    }
}
```

(6) 修改 fragment_second.xml 文件,用来显示单击按钮时更换的界面,代码如下:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/holo_blue_dark"
    android:orientation="vertical" >
    <Button
        android:id="@ + id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="我是左边单击出来的哦" />
</LinearLayout>
```

(7) 修改 SecondFragment 作为另一个右侧碎片,代码如下:

```
public class SecondFragment extends Fragment {
    @Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.right_fragment, container, false);
        return view;
    }
}
```

(8) 修改 activity_main.xml,代码如下:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
    android:layout_height = "match_parent">
    < fragment
        android:id = "@ + id/left_fragment"
        android:name = "com.example.chapter5_fragment.LeftFragment"
        android:layout_width = "0dp"
        android:layout_height = "match_parent"
        android:layout_weight = "1" />
    < FrameLayout
        android:id = "@ + id/right_layout"
        android:layout_width = "0dp"
        android:layout_height = "match_parent"
        android:layout_weight = "1" >
         <! -- 可以在这个容器中动态加载 Fragment -- >
        < fragment
            android:id = "@ + id/right_fragment"
            android:name = "com.example.chapter5_fragment.RightFragment"
            android:layout_width = "match_parent"
            android:layout_height = "match_parent" />
    </FrameLayout >
</LinearLayout >
```

(9) 可以看到,现在将右侧碎片放在了一个 FrameLayout 中,这是 Android 中最简单的一种布局,它没有任何的定位方式,所有的控件都会摆放在布局的左上角。由于这里仅需要在布局中放入一个碎片,因此非常适合使用 FrameLayout。之后将在代码中替换 FrameLayout 里的内容,从而实现动态添加碎片的功能。修改 MainActivity 中的代码如下:

```
public class MainActivity extends FragmentActivity implements View.OnClickListener {
    Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = (Button)findViewById(R.id.button);
        button.setOnClickListener(this);
    }
    @Override
    public void onClick(View view) {
        switch (view.getId()){
           case R.id.button:
           SecondFragment secFragment = new SecondFragment();
           FragmentManager fragmentManager = getSupportFragmentManager();
           FragmentTransaction transaction =
               fragmentManager.beginTransaction();
               transaction.replace(R.id.right_layout, secFragment);
               transaction.commit();
               break;
           default:
```

```
                break;
            }
        }
    }
```

可以看到，首先给左侧碎片中的按钮注册了一个单击事件，然后将动态添加碎片的逻辑都放在了单击事件中进行。结合代码可以看出，动态添加碎片主要分为如下 5 步：

① 创建待添加的碎片实例。

② 获取到 FragmentManager，在活动中可以直接调用 getFragmentManager()方法得到。

③ 开启一个事务，通过调用 beginTransaction()方法开启。

④ 向容器内加入碎片，一般使用 replace()方法实现，需要传入容器的 id 和待添加的碎片实例。

⑤ 提交事务，调用 commit()方法来完成。

这样就完成了在活动中动态添加碎片的功能，运行程序，可以看到启动界面如图 5-5 所示，然后单击按钮，效果如图 5-6 所示。

图 5-5 启动界面

图 5-6 单击按钮结果

上述代码成功实现了向活动中动态添加碎片的功能，不过这时按下键盘上的返回键程序就会直接退出。如果这里想模仿类似返回栈的效果，可以通过 FragmentTransaction 中提供的一个 addToBackStack()方法将一个事务添加到返回栈中，修改 MainActivity 中的代码如下：

```
@Override
    public void onClick(View view) {
        switch (view.getId()){
            case R.id.button:
                SecondFragment secFragment = new SecondFragment();
                FragmentManager fragmentManager = getSupportFragmentManager();
                FragmentTransaction transaction = fragmentManager.beginTransaction();
                transaction.replace(R.id.right_layout,secFragment);
```

```
                transaction.addToBackStack(null);
                transaction.commit();
                break;
            default:
                break;
        }
    }
```

这里在事务提交之前调用了 FragmentTransaction 的 addToBackStack()方法,它可以接收一个名字用于描述返回栈的状态,一般传入 null 即可。现在重新运行程序,并单击按钮将 SecondFragment 添加到活动中,然后按下返回键,会发现程序并没有退出,而是回到了 RightFragment 界面,再次按下返回键程序才会退出。


视频讲解

5.4 Fragment 与 Activity 之间的通信

由于 Fragment 与 Activity 各自存在于一个独立的类中,它们之间并没有明显的方式进行直接通信。在实际开发过程中,经常需要在 Activity 中获取 Fragment 实例或者在 Fragment 中获取 Activity 实例。接下来详细讲解 Fragment 和 Activity 之间的通信。

(1) 在 Activity 中获取 Fragment 实例。

为了实现 Fragment 和 Activity 之间的通信,FragmentManager 提供了一个 findFragmentById()的方法,专门用于从布局文件中获取 Fragment 实例。该方法有一个参数,它代表 Fragment 在 Activity 布局中的 id。例如,在布局文件中指定 SecondFragment 的 id 为 R.id.second_fragment,这时就可以使用 getFragmentManager().findFragmentById(R.id.second_fragment)方法得到 SecondFragment 的实例。

为了更好理解,下面通过一段代码讲解,具体的代码如下:

```
SecondFragment second_frag = (SecondFragment) getFragmentManager()
.findfragmentById(R.id.second_fragmnet);
```

以上就是在 Activity 中获取 Fragment 实例的代码。

(2) 在 Fragment 中获取 Activity 实例。

在 Fragment 中获取 Activity 实例对象,可以通过在 Fragment 中调用 getActivity()方法来获取与当前 Fragment 相关联的 Activity 实例对象。例如在 MainActivity 中添加了 SecondFragment,那么就可以通过在 Fragment 中调用 getActivity()来获取 MainActivity 实例对象。具体的代码如下:

```
MainActivity main = (MainActivity)getActivity();
```

获取到 Activity 中的实例以后,就可以调用该 Activity 中的方法了。当 Fragment 需要使用 Context 对象时,也可以使用该方法。

以上就是在 Activity 中获取 Fragment 实例和在 Fragment 中获取 Activity 实例对象的具体代码。接下来通过具体的例子讲解两者之间的通信方式。

为了更好地掌握 Fragment 与 Activity 之间的通信，接下来介绍一个左边显示新闻标题，右边展示单击新闻标题以后出现新闻的具体内容的例子，具体的操作步骤如下。

(1) 创建新闻展示项目。

首先创建新闻展示项目，然后修改 activity_main.xml 中的布局代码，因为需要展示标题和对应的内容，所以需要添加两个 FrameLayout，后边将会被 Fragment 所代替。具体的代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    xmlns:app = "http://schemas.android.com/apk/res - auto"
    xmlns:tools = "http://schemas.android.com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    tools:context = ".MainActivity"
    android:orientation = "horizontal">
    <! -- 标题 -- >
    < FrameLayout
        android:id = "@ + id/settitle"
        android:layout_width = "0dp"
        android:layout_weight = "1"
        android:layout_height = "match_parent">
    </FrameLayout >
    <! -- 内容 -- >
    < FrameLayout
        android:id = "@ + id/setcontent"
        android:layout_width = "0dp"
        android:layout_weight = "2"
        android:layout_height = "match_parent">
    </FrameLayout >
</LinearLayout >
```

(2) 创建两个 Fragment 布局文件。

由于需要实现在一个 Activity 中展示两个 Fragment，因此需要创建相应的 Fragment 的布局。用来展示新闻标题的布局文件 title_layout.xml 代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent">
    <! -- 用来展示新闻标题列表 -- >
    < ListView
        android:id = "@ + id/titlelist"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content">
    </ListView >
</LinearLayout >
```

用来展示右边标题和内容的布局文件 content_layout.xml 代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:orientation = "vertical">
    < TextView
        android:id = "@ + id/show_title"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:textSize = "20sp"
        android:text = "显示新闻标题" />
    < TextView
        android:id = "@ + id/show_content"
        android:layout_width = "match_parent"
        android:layout_marginTop = "20dp"
        android:layout_height = "wrap_content"
        android:textSize = "16sp"
        android:text = "显示新闻内容" />
</LinearLayout >
```

(3) 创建 ListView 中每一项的内容布局。

由于左边的新闻标题采用了 ListView，因此需要创建一个显示 ListView 中每一项的布局文件，title_item_layout.xml 文件的代码如下：

```
< TextView
    android:id = "@ + id/titles"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:textSize = "16sp"/>
```

(4) 创建显示标题的 Fragment 类文件。

创建一个 TitleFragment 类文件(继承自 Fragment 类)，用来显示左边的新闻标题，具体代码如下：

```
import androidx.fragment.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.BaseAdapter;
import android.widget.ListView;
import android.widget.TextView;
public class TitleFragment extends Fragment {
```

```
    private View view;
    private String[] title;
    private String[][] contents;
    private ListView listView;
    public View onCreateView(LayoutInflater inflater, final ViewGroup container, Bundle
savedInstanceState){
        view = inflater.inflate(R.layout.title_layout,container,false);
        //获取 Activity 实例对象
        MainActivity activity = (MainActivity)getActivity();
        //获取 Activity 中的标题
        title = activity.getTitle();
        //获取 Activity 中的标题和内容
        contents = activity.getSettingText();
        if (view!= null){
            init();
        }
        //为 listview 添加监听
        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
           public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
                //通过 activity 实例获取另一个 Fragment 对象
                ContentFragment content = (ContentFragment)((MainActivity)getActivity()).
getSupportFragmentManager().findFragmentById(R.id.setcontent);
                content.setText(contents[i]);
            }
        });
        return view;
    }
    private void init() {
        listView = (ListView)view.findViewById(R.id.titlelist);
        if (title!= null){
            listView.setAdapter(new MyAdapter());
        }
    }
    //适配器
    class MyAdapter extends BaseAdapter{
        @Override
        public int getCount() {
            return title.length;
        }
        @Override
        public Object getItem(int i) {
            return title[i];
        }
        @Override
        public long getItemId(int i) {
            return i;
        }
        @Override
        public View getView(int i, View view, ViewGroup viewGroup) {
```

```
            view = View.inflate(getActivity(),R.layout.title_item_layout,null);
            TextView titletext = (TextView)view.findViewById(R.id.titles);
            titletext.setText(title[i]);
            return view;
        }
    }
}
```

(5) 创建显示标题和内容的 Fragment 类文件。

创建一个类 ContentFragment(继承自 Fragment 类),然后编写相应的逻辑代码,用来显示左边单击以后出现的内容,具体代码如下:

```
package com.jxust.cn.chapter5_news;
import androidx.fragment.app.Activity;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
public class ContentFragment extends Fragment {
    private View view;
    private TextView text1,text2;
    public void onAttach(Activity activity){
        super.onAttach(activity);
    }
      public View onCreateView (LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState){
        //获取布局文件
        view = inflater.inflate(R.layout.content_layout,container,false);
        if (view!= null){
            init();
        }
        //获取 activity 中设置的文字
        setText(((MainActivity)getActivity()).getSettingText()[0]);
        return view;
    }
    private void init() {
        text1 = (TextView)view.findViewById(R.id.show_title);
        text2 = (TextView)view.findViewById(R.id.show_content);
    }
    public void setText(String[] text) {
        text1.setText(text[0]);
        text2.setText(text[1]);
    }
}
```

(6) 编写 MainActivity 中的代码。

编写好两个 Fragment 类的代码以后,就需要在 MainActivity 中添加,具体的代码如下:

```
import androidx.fragment.app.FragmentActivity;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentTransaction;
import android.os.Bundle;
public class MainActivity extends FragmentActivity {
    //设置标题
    private String title[] = {"标题一","标题二","标题三"};
    private String settingText[][] = {{"标题一","标题一的内容"},{"标题二","标题二的内
容"},{"标题三","标题三的内容"}};
    //获取标题数组的方法
    public String[] getTitle(){
        return title;
    }
    //获取标题和内容
    public String[][] getSettingText(){
        return settingText;
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //创建 Fragment
        TitleFragment titleFragment = new TitleFragment();
        ContentFragment contentFragment = new ContentFragment();
        //获取事务
        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction transaction = fragmentManager.beginTransaction();
        //添加 Fragment
        transaction.replace(R.id.settitle,titleFragment);
        transaction.replace(R.id.setcontent,contentFragment);
        //提交事务
        transaction.commit();
    }
}
```

（7）测试运行。

以上就是 Activity 与 Fragment 之间的通信过程。上述代码实现了如图 5-7 所示的界面。

图 5-7 Fragment 与 Activity 通信案例图

从图 5-7 可以看出,当单击屏幕左侧的标题以后,右侧的界面也会跟着显示对应的标题和内容,这就说明了本实例实现了 Activity 与 Fragment 之间的通信以及 Fragment 与 Fragment 之间的通信。需要开发者熟练掌握。

本章小结

本章主要讲解了 Fragment 的概念、生命周期、Fragment 与 Activity 之间的通信方式以及 Fragment 和 Fragment 之间的通信方式,这些知识在平板电脑开发或者考虑到屏幕兼容性开发中经常使用,需要开发者熟练掌握并应用到实际的项目中。

习题

1. 说明 Fragment 的生命周期。
2. 对于 Android 的两种事件处理机制,分别写一个案例测试,了解其执行过程。
3. 实现一个类似于 5.4 节 Fragment 与 Activity 之间通信的例子。