第 7 章

UI fragment与fragment管理器

7

本章，我们将学习开发一个名为CriminalIntent的应用。CriminalIntent应用可详细记录种种办公室陋习，如随手将脏盘子丢在休息室水槽里，以及打印完自己的文件便径直走开，全然不顾打印机里已经缺纸等。

通过CriminalIntent应用，陋习记录可包含标题、日期以及照片。也可在联系人中查找当事人，然后通过Email、Twitter、Facebook或其他应用提出不满意见。记录并报告陋习后，有了好心情，就可以继续完成工作或处理手头上的事情。真是个不错的应用。

CriminalIntent应用比较复杂，我们需要13章的篇幅来完成它。应用的用户界面主要由列表以及记录明细组成。主屏幕会显示已记录陋习的列表清单。用户可新增记录或选中现有记录进行查看和细节编辑，如图7-1所示。

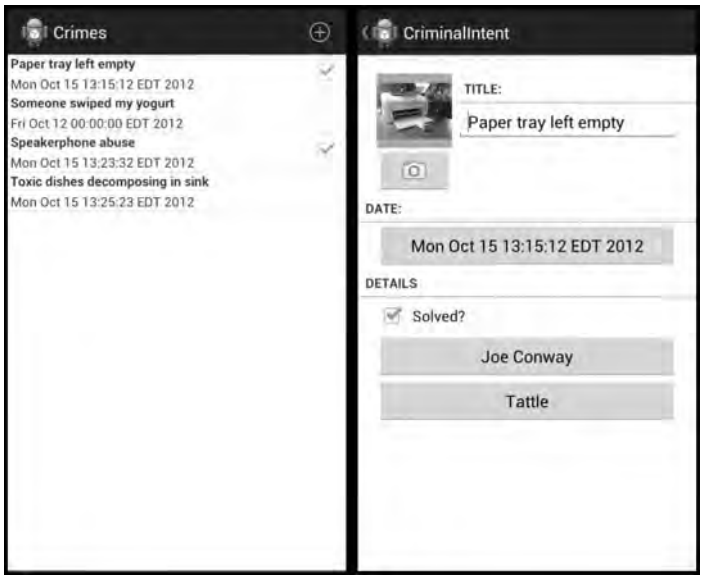

图7-1　CriminalIntent，一个列表明细应用

7.1 UI 设计的灵活性需求

想象开发一个由两个activity组成的列表明细类应用，其中一个activity管理着记录列表界面，另一个activity管理着记录明细界面。单击列表中一条记录启动一个记录明细activity实例。单击后退键销毁明细activity并返回到记录列表activity界面，接下来可以再选择一条记录。

理论上这行的通。但如果需要更复杂的用户界面呈现及跳转呢?

- 假设用户正在平板设备上运行CriminalIntent应用。平板以及大尺寸手机通常拥有比较大的屏幕，能够同时显示列表以及明细，至少在水平方位模式下。显示模式如图7-2所示。

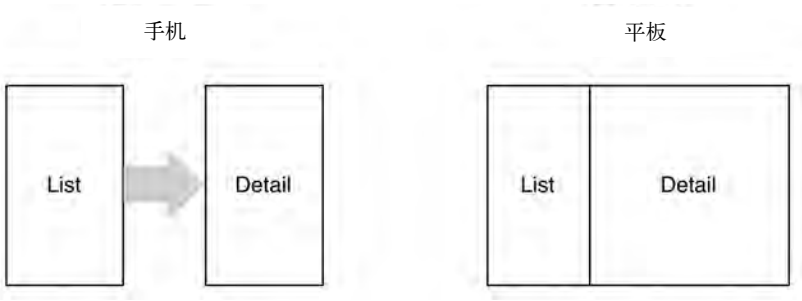

图7-2 手机和平板上理想的列表明细界面

- 假设用户正在手机上查看记录信息，并想查看列表中的下一条记录信息。如无需返回列表界面，通过滑动屏幕即可查看下一条记录信息就好了。每滑动一次屏幕，应用便自动切换到下一条记录明细。

可以看出，UI设计具有灵活性是以上假设情景的共同点。即根据用户或设备的需要，activity界面可以在运行时组装，甚至重新组装。

activity自身并不具有这样的灵活性。activity视图可以在运行时切换，但控制视图的代码必须在activity中实现。因而，各个activity还是得和特定的用户屏幕紧紧绑定在一起。

7.2 fragment 的引入

采用fragment而不是activity进行应用的UI管理，可绕开Android系统activity规则的限制。

fragment是一种控制器对象，activity可委派它完成一些任务。通常这些任务就是管理用户界面。受管的用户界面可以是一整屏或是整屏的一部分。

管理用户界面的fragment又称为UI fragment。它也有自己产生于布局文件的视图。fragment视图包含了用户可以交互的可视化UI元素。

activity视图含有可供fragment视图插入的位置。如果有多个fragment要插入，activity视图也可提供多个位置。

根据应用和用户的需求，可联合使用fragment及activity来组装或重新组装用户界面。在整个生命周期过程中，技术上来说activity的视图可保持不变。因此不用担心会违反Android系统activity规则。

一个列表明细应用准备同时显示列表与明细内容，下面我们就来看看该应用是怎么做到这一点的。应用里的activity视图是通过列表fragment和明细fragment组装而成的。明细视图显示所选列表项的明细内容。

选择不同的列表项会显示不同的明细视图，fragment很容易做到这一点。activity将以一个明细fragment替换另一个明细fragment，如图7-3所示。视图切换的过程中，任何activity都无需被销毁。

图7-3　明细fragment的切换

用UI fragment将应用的UI分解成构建块，除列表明细应用外，也适用于其他类型的应用。利用一个个构建块，很容易做到构建分页界面、动画侧边栏界面等更多其他定制界面。

不过，达成这种UI设计的灵活性也是有代价的，即更加复杂的应用、更多的部件管理以及更多的实现代码。我们会在第11章和第22章中体会到使用fragment的好处。现在先来感受下它的复杂性。

7.3　着手开发 CriminalIntent

本章，我们将着手开发CriminalIntent应用的记录明细部分。完成后的画面如图7-4所示。

看上去不是一个激动人心的开发目标？记住，本章是在为应用的后续开发夯实基础。

图7-4所示的用户界面将由一个名为`CrimeFragment`的UI fragment进行管理。`CrimeFragment`的实例将通过一个名为`CrimeActivity`的activity来托管。

我们可以暂时把托管理解成activity在其视图层级里提供一处位置用来放置fragment的视图，如图7-5所示。Fragment本身不具有在屏幕上显示视图的能力。因此，只有将它的视图放置在activity的视图层级结构中，fragment视图才能显示在屏幕上。

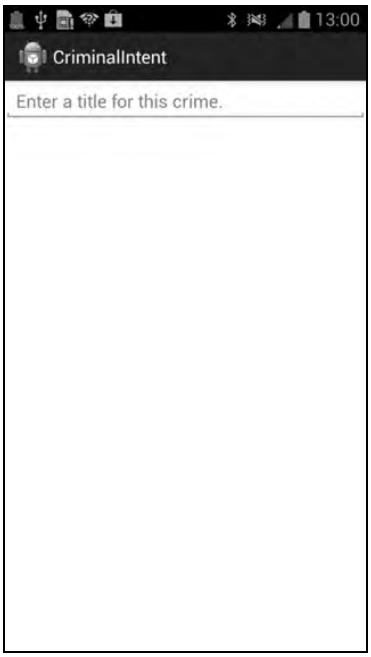

图7- 4 本章结束时，CriminalIntent应用的界面

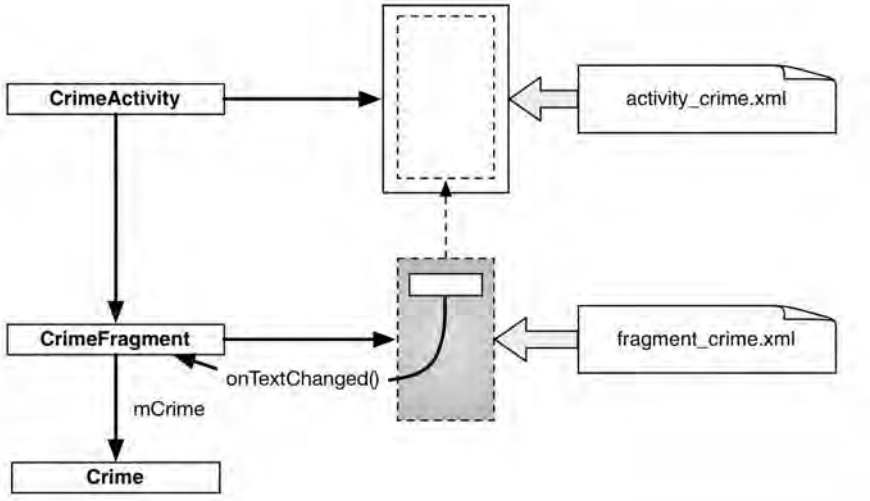

图7-5 CrimeActivity托管着CrimeFragment

CriminalIntent是一个大型项目，可通过对象图解来更好地理解它。图7-6展示了CriminalIntent项目对象的整体图解。无需记住这些对象及其间的关系。但预先对开发目标有一个清楚地认识将有助于我们的开发。

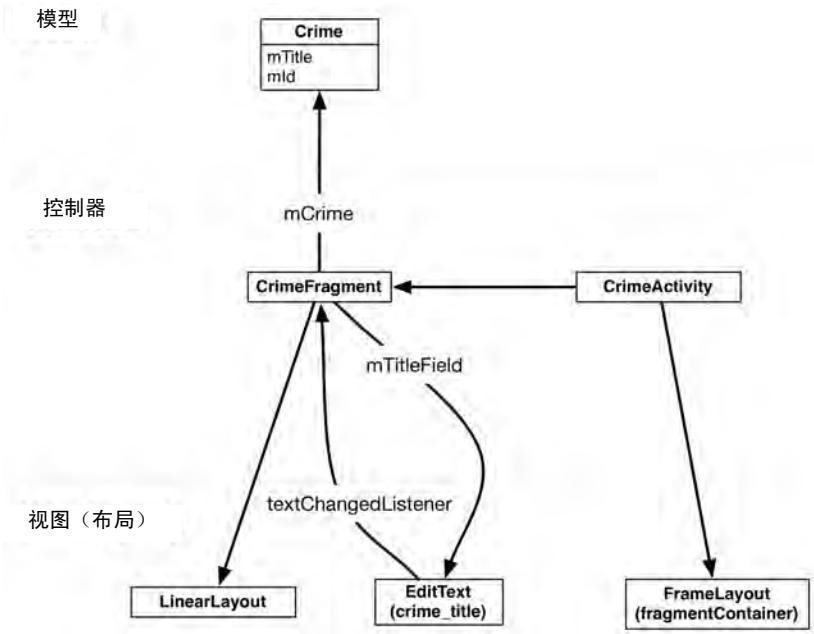

图7-6　CriminalIntent应用的对象图解（本章应完成部分）

可以看到，CrimeFragment的作用与activity在GeoQuiz应用中的作用差不多，都负责创建并管理用户界面，与模型对象进行交互。

其中Crime、CrimeFragment以及CrimeActivity是我们要开发的类。

Crime实例代表了某种办公室陋习。在本章中，一个crime只有一个标题和一个标识ID。标题是一段描述性名称，如“有毒的水池垃圾堆”或“某人偷了我的酸奶!”等。标识ID是识别Crime实例的唯一元素。

简单起见，本章我们只使用一个Crime实例，并将其存放在CrimeFragment类的成员变量（mCrime）中。

CrimeActivity视图由FrameLayout组件组成，FrameLayout组件为CrimeFragment要显示的视图安排了存放位置。

CrimeFragment的视图由一个LinearLayout组件及一个EditText组件组成。CrimeFragment类中有一个存储EditText的成员变量（mTitleField）。mTitleField上设有监听器，当EditText上的文字发生改变时，用来更新模型层的数据。

7.3.1　创建新项目

介绍了这么多，是时候创建新应用了。选择New → Android Application Project菜单项创建新的Android应用。如图7-7所示，将应用命名为CriminalIntent，包名命名为com.bignerdranch.android.criminalintent。编译及目标版本设置为最新的API级别并保证应用兼容运行Froyo系统的设备。

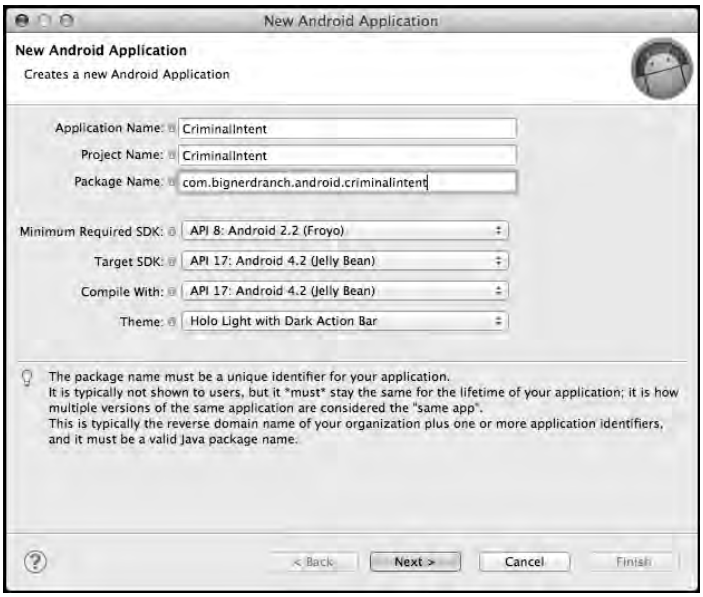

图7-7 创建CriminalIntent应用

在接下来的对话框中，不勾选创建定制启动器图标，单击Next继续。然后利用空白activity模板创建activity，单击Next继续。

最后，命名activity为`CrimeActivity`，单击Finish按钮完成，如图7-8所示。

图7-8 配置`CrimeActivity`

7.3.2　fragment与支持库

随着Android平板设备的首发，为满足平板设备的UI灵活性设计要求，Fragment被引入到API 11级中。CriminalIntent应用支持的SDK最低版本为API 8级，因此必须设法保证应用兼容旧版本设备。

幸运的是，对于fragment来说，保证向后兼容相对比较容易，仅需使用Android支持库中的fragment相关类即可。

支持库位于libs/android-support-v4.jar内，并通过创建项目模板已被自动添加到项目中。支持库包含了`Fragment`类（`android.support.v4.app.Fragment`），该类可以使用在任何API 4级及更高版本的设备上。

支持库中的类不仅可以在无原生类的旧版本设备上使用，而且可以代替原生类在新版本设备上使用。

另一个重要的支持库类是`FragmentActivity`（`android.support.v4.app.FragmentActivity`）。activity知道如何管理fragment，因此fragment的使用需要activity的支持。在Honeycomb及后续的Android版本中，`Activity`的所有子类都知道如何管理fragment。而这之前版本的`Activity`则完全不了解fragment，`Activity`的子类自然也就无从知晓。为兼容较低版本的设备，可继承`FragmentActivity`类。`FragmentActivity`是`Activity`的子类，具有新系统版本`Activity`类管理fragment的能力，即便是在较早版本的Android设备上也可对fragment进行管理。新旧版本设备上的fragment支持类如图7-9所示。

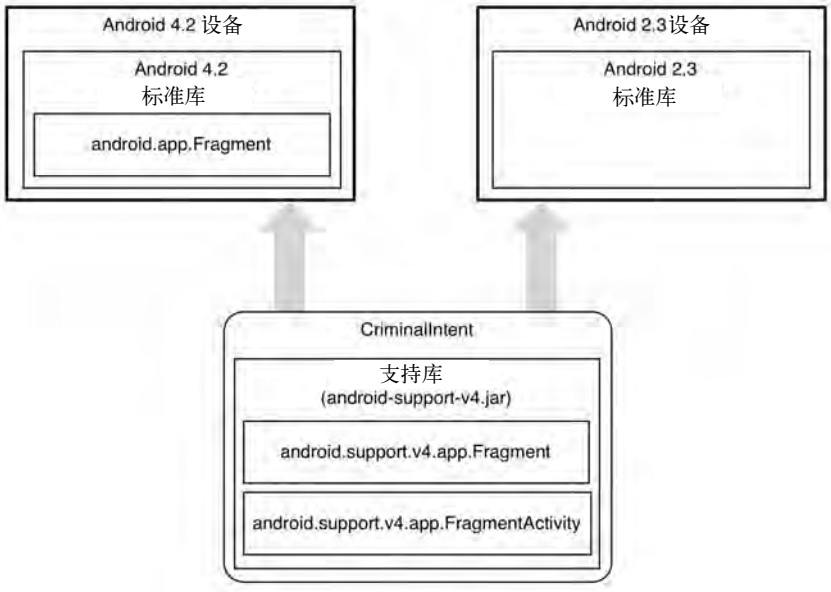

图7-9　新旧版本设备上的fragment支持类

图7-9显示了这些类的名称及位置。既然支持库（以及android.support.v4.app.Fragment）支持我们的应用，我们就可以放心使用它。

在包浏览器中，找到并打开CrimeActivity.java文件。将CrimeActivity的超类更改为FragmentActivity，同时删除由模板生成的onCreateOptionsMenu(Menu)方法实现代码，如代码清单7-1所示。（第16章将学习如何从头创建CriminalIntent应用的选项菜单。）

代码清单7-1 修改模板自动产生的代码（CrimeActivity.java）

```
public class CrimeActivity extends Activity FragmentActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_crime, menu);
        return true;
    }
}
```

在进一步完善CrimeAcitivty类之前，我们先来为CriminalIntent应用创建模型层的Crime类。

7.3.3 创建Crime类

在包浏览器中，右键单击com.bignerdranch.android.criminalintent包，选择New → Class菜单项。在新建类对话框中，命名类为Crime，保持java.lang.Object超类不变，单击Finish按钮完成。

在随后打开的Crime.java中，增加代码清单7-2所示的代码。

代码清单7-2 Crime类的新增代码（Crime.java）

```
public class Crime {

    private UUID mId;
    private String mTitle;

    public Crime() {
        // 生成唯一标识符
        mId = UUID.randomUUID();
    }
}
```

接下来，需为只读成员变量mId生成一个getter方法，为成员变量mTitle生成getter和setter方法。右键单击构造方法下面的空白处，选择Source → Generate Getters and Setters菜单项。为只生成变量mId的getter方法，单击该变量名称左边的箭头符号，展示所有可能的方法，只勾选getId()方法，如图7-10所示。

图7-10　生成两个getter方法和一个setter方法

代码清单7-3　已生成的getter与setter方法（Crime.java）

```
public class Crime {
    private UUID mId;

    private String mTitle;

    public Crime() {
        mId = UUID.randomUUID();
    }

    public UUID getId() {
        return mId;
    }

    public String getTitle() {
        return mTitle;
    }

    public void setTitle(String title) {
        mTitle = title;
    }
}
```

以上是本章CriminalIntent模型层及组成它的Crime类所需的全部实现代码工作。

至此，除了模型层，我们还创建了能够托管fragment且兼容Froyo及GingerBread的activity。接下来，我们将继续学习activity托管fragment的实现细节部分。

7.4 托管 UI fragment

为托管UI fragment，activity必须做到：

- 在布局中为fragment的视图安排位置；
- 管理fragment实例的生命周期。

7.4.1 fragment的生命周期

图7-11展示了fragment的生命周期。类似于activity的生命周期，它既具有停止、暂停以及运行状态，也拥有可以覆盖的方法，用来在关键节点完成一些任务。可以看到，许多方法对应着activity的生命周期方法。

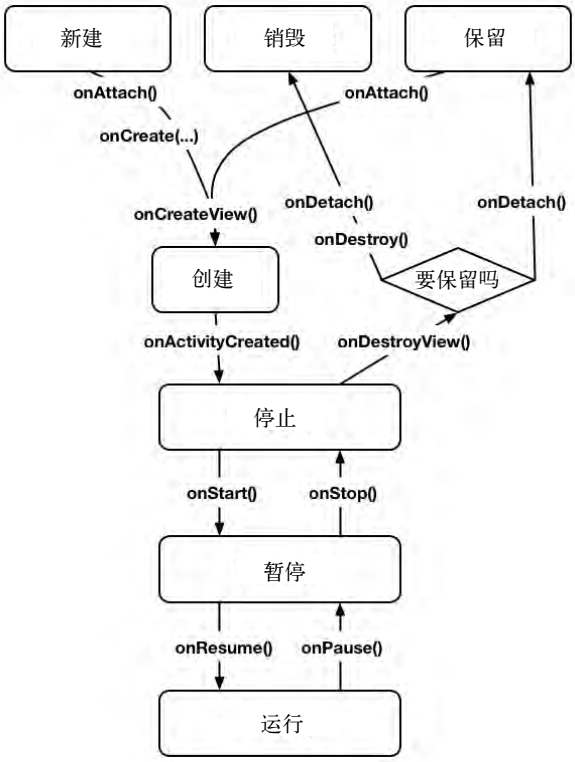

图7-11 fragment的生命周期图解

生命周期方法的对应非常重要。因为fragment代表activity在工作，它的状态应该也反映了activity的状态。因而，fragment需要对应的生命周期方法来处理activity的工作。

fragment生命周期与activity生命周期的一个关键区别就在于，fragment的生命周期方法是由

托管activity而不是操作系统调用的。操作系统无从知晓activity用来管理视图的fragment。fragment的使用是activity自己内部的事情。

随着CriminalIntent应用开发的深入，我们会看到更多的fragment生命周期方法。

7.4.2　托管的两种方式

在activity中托管一个UI fragment，有如下两种方式：

- 添加fragment到activity*布局*中；
- 在activity*代码*中添加fragment。

第一种方式即使用*布局*fragment。这种方式虽然简单但灵活性不够。 添加fragment到activity布局中，就等同于将fragment及其视图与activity的视图绑定在一起，且在activity的生命周期过程中，无法切换fragment视图。

尽管布局fragment使用起来不够灵活，但它也不是一无用处。第13章，我们将接触到更多有关这一点的内容。

第二种方式是一种比较复杂的托管方式，但也是唯一一种可以在运行时控制fragment的方式。我们可以决定何时将fragment添加到activity中以及随后可以完成何种具体任务；也可以移除fragment，用其他fragment代替当前fragment，然后再重新添加已移除的fragment。

因而，为获得真正的UI设计灵活性，我们必须通过代码的方式添加fragment。这也是我们使用`CrimeActivity`托管`CrimeFragment`的方式。本章后续内容会介绍代码的实现细节。现在，我们先来学习定义`CrimeActivity`的布局。

7.4.3　定义容器视图

虽然我们要在托管activity代码中添加UI fragment，但还是需要在activity视图层级结构中为fragment视图安排位置。在`CrimeActivity`的布局中，该位置如图7-12中的`FrameLayout`所示。

图7-12　`CrimeActivity`类的fragment托管布局

`FrameLayout`是服务于`CrimeFragment`的*容器视图*。注意容器视图是通用性视图，不局限于`CrimeFragment`类，我们可以并且也将使用同一个布局来托管其他的fragment。

定位到`CrimeActivity`的布局文件res/layout/activity_crime.xml。打开该文件，使用图7-12所示的`FrameLayout`替换默认布局。完成后的XML文件应如代码清单7-4所示。

代码清单7-4 创建fragment容器布局（activity_crime.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/fragmentContainer"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  />
```

注意，虽然当前的activity_crime.xml布局文件仅由一个服务于单个fragment的容器视图组成，但托管activity布局本身也可以非常复杂。除自身组件外，托管activity布局还可定义多个容器视图。

现在预览一下布局文件，或者运行CriminalIntent应用检查下实现代码。不过，由于`CrimeActivity`还没有托管任何fragment，因此我们只能看到一个空的`FrameLayout`，如图7-13所示。

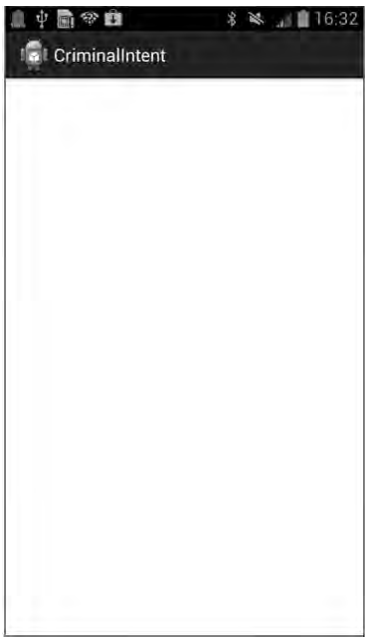

图7-13 一个空的`FrameLayout`

稍后，我们会编写代码，将fragment的视图放置到`FrameLayout`中。不过，首先我们需要先来创建一个fragment。

7.5 创建 UI fragment

创建一个UI fragment的步骤与创建一个activity的步骤相同，具体步骤如下所示：

- 通过定义布局文件中的组件，组装界面；
- 创建fragment类并设置其视图为定义的布局；
- 通过代码的方式，连接布局文件中生成的组件。

7.5.1　定义CrimeFragment的布局

CrimeFragment视图将显示包含在Crime类实例中的信息。应用最终完成时，Crime类及CrimeFragment视图将包含很多有趣的东西。但本章，我们只需完成一个文本栏位来存放crime的标题。

图7-14显示了CrimeFragment视图的布局。该布局包括一个垂直LinearLayout布局（含有一个EditText组件）。EditText组件提供一块区域供用户添加或编辑文字信息。

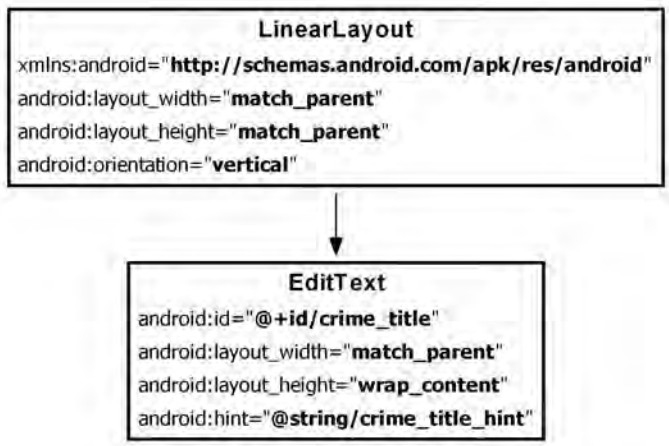

图7-14　CrimeFragment的初始布局

要创建布局文件，可在包浏览器中，右键单击res/layout文件夹，选择New → Android XML File菜单项。在弹出的对话框中，确认资源类型选择了Layout，命名布局文件为fragment_crime.xml。选择LinearLayout作为根元素节点后，单击Finish按钮完成。

新建文件打开后，查看XML，会发现向导已经添加了LinearLayout。如图7-14所示，对fragment_crime.xml做必要的调整。可使用代码清单7-5检查有无差错。

代码清单7-5　fragment视图的布局文件（fragment_crime.xml）

```
<?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"
  >
  <EditText android:id="@+id/crime_title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/crime_title_hint"
    />
</LinearLayout>
```

打开res/values/strings.xml，添加crime_title_hint字符串资源，删除模板产生的不需要的

hello_world以及menu_settings字符串资源，增删字符串资源的做法如代码清单7-6所示。

代码清单7-6 增删字符串资源（res/values/strings.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">CriminalIntent</string>
  <string name="hello_world">Hello world!</string>
  <string name="menu_settings">Settings</string>
  <string name="title_activity_crime">CrimeActivity</string>
  <string name="crime_title_hint">Enter a title for the crime.</string>
</resources>
```

保存所有文件。删除menu_settings字符串资源会导致项目发生错误。要想修正错误，可通过包浏览器找到res/menu/activity_crime.xml文件。该文件定义了模板创建的menu，引用了menu_settings字符串。CriminalIntent应用不需要该menu文件，因此可直接将其从包浏览器中删除。

删除menu资源会促使Eclipse进行重新编译。现在项目应该不会出现错误提示了。切换到图形布局工具，预览已完成的fragment_crime.xml布局。

7.5.2 创建CrimeFragment类

右键单击com.bignerdranch.android.criminalintent包，选择New → Class菜单项。在弹出的新建类对话框中，命名类为`CrimeFragment`，然后单击旁边的Browse按钮设置超类。在弹出的超类选择对话框中，输入`Fragment`。向导自动显示了一些匹配的类。选择支持库中`Fragment`类下的的`android.support.v4.app.Fragment`类，如图7-15所示。确认无误后，单击OK按钮完成。

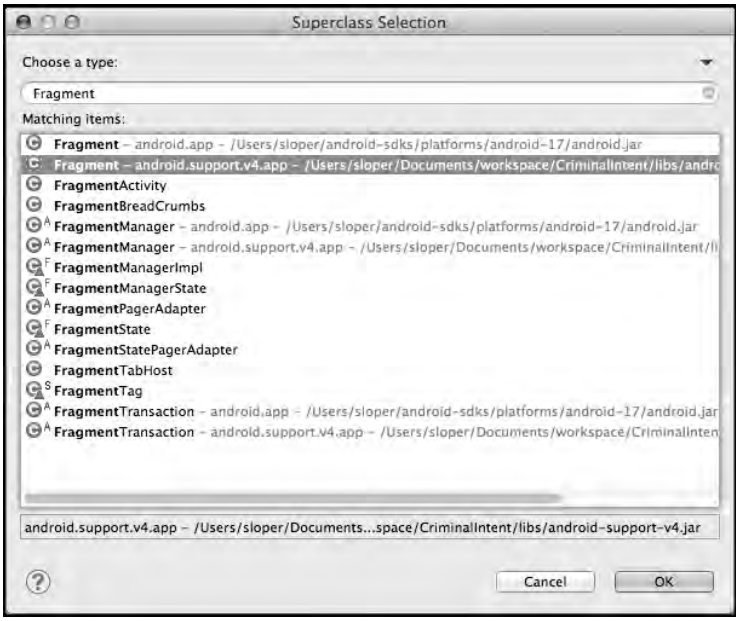

图7-15 选择支持库中的`Fragment`类

（如有多个版本的android.support.v4.app.Fragment类可供选择，请确认选择了CriminalIntent/libs/目录下CriminalIntent项目中的Fragment类。）

1. 实现fragment生命周期方法

CrimeFragment类是与模型及视图对象交互的控制器，用于显示特定crime的明细信息，并在用户修改这些信息后立即进行内容更新。

在GeoQuiz应用中，activity通过其生命周期方法完成了大部分逻辑控制工作。而在CriminalIntent应用中，这些工作是由fragment通过其生命周期方法完成的。fragment的许多方法对应着我们熟知的Activity方法，如onCreate(Bundle)方法。

在CrimeFragment.java中，新增一个Crime实例成员变量，实现Fragment.onCreate(Bundle)方法，如代码清单7-7所示。

代码清单7-7　覆盖Fragment.onCreate(Bundle)方法（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }

}
```

以上实现代码中需注意以下几点：

首先，Fragment.onCreate(Bundle)是公共方法，而Activity.onCreate(Bundle)是保护方法。因为需要被托管fragment的任何activity调用，因此Fragment.onCreate(...)方法及其他Fragment生命周期方法必须设计为公共方法。

其次，类似于activity，fragment同样具有保存及获取状态的bundle。如同使用Activity.onSaveInstanceState(Bundle)方法那样，我们也可以根据需要覆盖Fragment.onSaveInstanceState(Bundle)方法。

最后注意，在Fragment.onCreate(...)方法中，并没有生成fragment的视图。虽然在Fragment.onCreate(...)方法中配置了fragment实例，但创建和配置fragment视图是通过另一个fragment生命周期方法来完成的（如下所示）：

```
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState)
```

通过该方法生成fragment视图的布局，然后将生成的View返回给托管activity。LayoutInflater及ViewGroup是用来生成布局的必要参数。Bundle包含了供该方法在保存状态下重建视图所使用的数据。

在CrimeFragment.java中，添加onCreateView(...)方法的实现代码，从fragment_crime.xml布局中产生并返回视图，如代码清单7-8所示。

代码清单7-8 覆盖onCreateView(...)方法（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);
        return v;
    }

}
```

在onCreateView(...)方法中，fragment的视图是直接通过调用LayoutInflater.inflate(...)方法并传入布局的资源ID生成的。第二个参数是视图的父视图，通常我们需要父视图来正确配置组件。第三个参数告知布局生成器是否将生成的视图添加给父视图。这里，我们传入了false参数，因为我们将通过activity代码的方式添加视图。

2. 在fragment中关联组件

onCreateView(...)方法也是生成EditText组件并响应用户输入的地方。视图生成后，引用EditText组件并添加对应的监听器方法。生成并使用EditText组件的具体代码如代码清单7-9所示。

代码清单7-9 生成并使用EditText组件（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);

        mTitleField = (EditText)v.findViewById(R.id.crime_title);
        mTitleField.addTextChangedListener(new TextWatcher() {
            public void onTextChanged(
                    CharSequence c, int start, int before, int count) {
                mCrime.setTitle(c.toString());
            }

            public void beforeTextChanged(
                    CharSequence c, int start, int count, int after) {
                // This space intentionally left blank
            }

            public void afterTextChanged(Editable c) {
                // This one too
```

```
            }
        });

        return v;
    }
}
```

Fragment.onCreateView(...)方法中的组件引用几乎等同于Activity.onCreate(...)方法的处理。唯一的区别是我们调用了fragment视图的View.findViewById(int)方法。以前使用的Activity.findViewById(int)方法十分便利，能够在后台自动调用View.findViewById(int)方法。而Fragment类没有对应的便利方法，因此我们必须自己完成调用。

fragment中监听器方法的设置和activity中的处理完全一样。如代码清单7-9所示，创建实现TextWatcher监听器接口的匿名内部类。TextWatcher有三种方法，不过我们现在只需关注其中的onTextChanged(...)方法。

在onTextChanged(...)方法中，调用CharSequence（代表用户输入）的toString()方法。该方法最后返回用来设置Crime标题的字符串。

CrimeFragment类的代码实现部分完成了。但现在还不能运行应用查看用户界面和检验代码。因为fragment无法将自己的视图显示在屏幕上。接下来我们首先要把CrimeFragment添加给CrimeActivity。

7.6　添加 UI fragment 到 FragmentManager

Fragment类引入到Honeycomb时，为协同工作，Activity类被更改为含有FragmentManager类。FragmentManager类负责管理fragment并将它们的视图添加到activity的视图层级结构中。

FragmentManager类具体管理的是：

- fragment队列；
- fragment事务的回退栈（这一点稍后将会学习到）。

FragmentManager的关系图如图7-16所示。

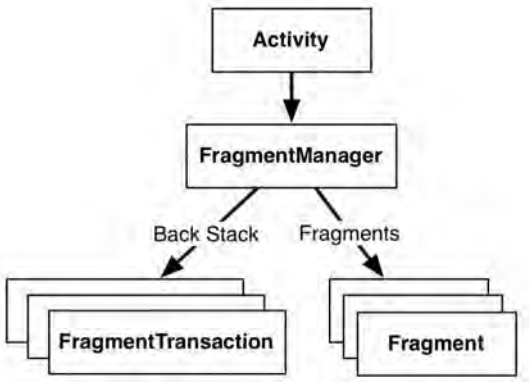

图7-16　FragmentManager关系图

在CriminalIntent应用中，我们只需关心FragmentManager管理的fragment队列即可。

要通过代码的方式将fragment添加到activity中，可直接调用activity的FragmentManager。首先，我们需要获取FragmentManager本身。在CrimeActivity.java中，添加代码清单7-10所示代码到onCreate(...)方法中。

代码清单7-10 获取FragmentManager（CrimeActivity.java）

```
public class CrimeActivity extends FragmentActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);

        FragmentManager fm = getSupportFragmentManager();
    }
}
```

因为使用了支持库及FragmentActivity类，因此这里调用的方法是getSupportFragmentManager()。如果不考虑Honeycomb以前版本的兼容性问题，可直接继承Activity类并调用getFragmentManager()方法。

7.6.1 fragment事务

获取到FragmentManager后，添加代码清单7-11所示代码，获取一个fragment交由FragmentManager管理。（稍后，我们会逐行解读代码，现在只管对照添加即可。）

代码清单7-11 添加一个CrimeFragment（CrimeActivity.java）

```
public class CrimeActivity extends FragmentActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}
```

在代码清单7-11中，理解新增代码的最佳位置并非开头。相反，应查看add(...)方法及其周围的代码。这段代码创建并提交了一个fragment事务，如代码清单7-12所示。

代码清单7-12　一个fragment事务（CrimeActivity.java）

```
if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragmentContainer, fragment)
        .commit();
}
```

fragment事务被用来添加、移除、附加、分离或替换fragment队列中的fragment。这是使用fragment在运行时组装和重新组装用户界面的核心方式。FragmentManager管理着fragment事务的回退栈。

FragmentManager.beginTransaction()方法创建并返回FragmentTransaction实例。FragmentTransaction类使用了一个fluent interface接口方法，通过该方法配置FragmentTransaction返回FragmentTransaction类对象，而不是void，由此可得到一个FragmentTransaction队列。因此，代码清单7-12加亮部分代码可解读为："创建一个新的fragment事务，加入一个添加操作，然后提交该事物。"

add(...)方法是整个事务的核心部分，并含有两个参数，即容器视图资源ID和新创建的CrimeFragment。容器视图资源ID我们应该很熟悉了，它是定义在activity_crime.xml中的FrameLayout组件的资源ID。容器视图资源ID主要有两点作用：

- 告知FragmentManagerfragment视图应该出现在activity视图的什么地方；
- 是FragmentManager队列中fragment的唯一标识符。

如需从FragmentManager中获取CrimeFragment，即可使用容器视图资源ID，如代码清单7-13所示：

代码清单7-13　使用容器视图资源ID获取fragment（CrimeActivity.java）

```
FragmentManager fm = getSupportFragmentManager();
Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragmentContainer, fragment)
        .commit();
}
```

FragmentManager使用FrameLayout组件的资源ID去识别CrimeFragment，这看上去可能有点怪。但实际上，使用容器视图资源ID去识别UI fragment已被内置在FragmentManager的使用机制中。

现在我们对代码清单7-11中的新增代码从头到尾地做一下总结。

首先，使用R.id.fragmentContainer的容器视图资源ID，向FragmentManager请求获取fragment。如要获取的fragment在队列中已经存在，FragmentManager随即会将之返还。

为什么队列中已经有fragment存在了呢？CrimeActivity因设备旋转或回收内存被销毁后重建时，CrimeActivity.onCreate(...)方法会响应activity的重建而被调用。activity被销毁时，它的FragmentManager会将fragment队列保存下来。这样，activity重建时，新的FragmentManager会首先获取保存的队列，然后重建fragment队列，从而恢复到原来的状态。

另一方面，如指定容器视图资源ID的fragment不存在，则fragment变量为空值。这时应创建一个新的CrimeFragment，并开启一个新的fragment事务，然后在事务里将新建的fragment添加到队列中。

CrimeActivity目前托管着CrimeFragment。运行CriminalIntent应用验证这一点。如图7-17所示，应该可以看到定义在fragment_crime.xml中的视图。

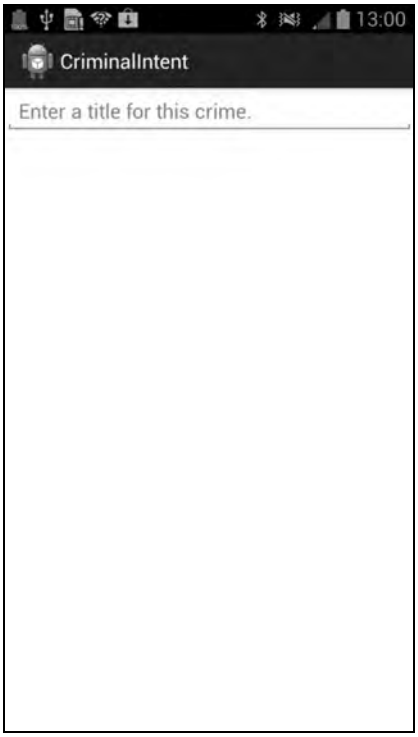

图7-17 CrimeActivity托管的CrimeFragment视图

我们在本章中付出了这么多努力，似乎不应该只得到屏幕上的这么一个组件。不要沮丧，事实上，本章所做的一切已经为后续章节有关CriminalIntent应用的深入开发打下了坚实的基础。

7.6.2 FragmentManager与fragment生命周期

掌握了FragmentManager的基本使用后，我们来重新审视一下fragment的生命周期，如图7-18所示。

activity的FragmentManager负责调用队列中fragment的生命周期方法。添加fragment供FragmentManager管理时，onAttach(Activity)、onCreate(Bundle)以及onCreateView(...)方法会被调用。

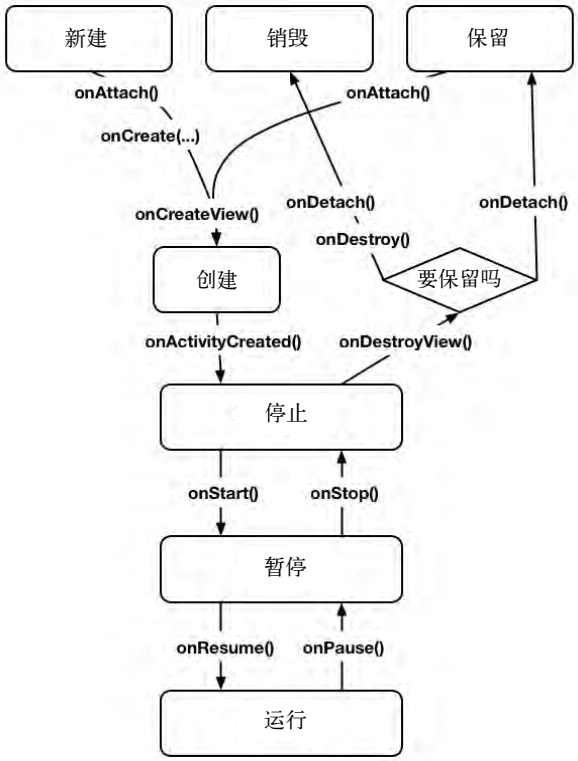

图7-18　再探fragment生命周期

托管activity的onCreate(...)方法执行后，onActivityCreated(...)方法也会被调用。因为我们正在向CrimeActivity.onCreate(...)方法中添加CrimeFragment，所以fragment被添加后，该方法会被调用。

在activity处于停止、暂停或运行状态下时，添加fragment会发生什么呢？此种情况下，FragmentManager立即驱使fragment快速跟上activity的步伐，直到与activity的最新状态保持同步。例如，向处于运行状态的activity中添加fragment时，以下fragment生命周期方法会被依次调用：onAttach(Activity)、onCreate(Bundle)、onCreateView(...)、onActivityCreated(Bundle)、onStart()，以及onResume()方法。

只要fragment的状态与activity的状态保持了同步，托管activity的FragmentManager便会继续调用其他生命周期方法以继续保持fragment与activity的状态一致，而几乎就在同时，它接收到了从操作系统发出的相应调用。但fragment方法究竟是在activity方法之前还是之后调用的这一点是无法保证的。

如使用了支持库，fragment生命周期的某些方法被调用的顺序则略有不同。如在Activity.onCreate(...)方法中添加fragment,那么onActivityCreated(...)方法是在Activity.onStart()方法执行后被调用的，而不是紧随Activity.onCreate(...)方法之后被调用。为什么会这样？

在Honeycomb以前的版本中，立即从FragmentActivity中调用onActivityCreated(...)方法是不可能的。因此，下一个生命周期方法一执行，它才会被调用。在实际开发中，二者通常没什么区别；onStart()方法会紧随Activity.onCreate(...)方法之后被调用。

7.7 activity 使用 fragment 的理由

在本书的后续章节中，无论应用多么简单，我们都将使用fragment进行开发。这貌似有点激进，要知道，后续章节很多应用案例的开发都不必使用fragment，用户界面也可以只使用activity来创建和管理，这样做的实现代码甚至会更少。

然而，我们认为，这是实际开发中最可能使用的模式，因此大家最好尽快适应它。

在应用开发初期暂不使用fragment，等到需要时再添加它，有人可能觉得这样做会好一些。极限编程理论中有个YAGNI原则。YAGNI（You Aren't Gonna Need It）的意思是“你不会需要它”，该原则鼓励大家不要去编码实现那些可能需要的东西。为什么呢？因为你不会需要它。因此，按照YAGNI原则，我们更倾向于不使用fragment。

不幸的是，后期添加fragment如同脚踏雷区。从activity管理用户界面的调整到由activity托管UI fragment并不困难，但会有一大堆恼人的问题需要我们去处理。保持部分用户界面由activity管理不变，另一部分用户界面使用fragment来管理，这只会使情况变得更加糟糕，因为我们必须维护这种毫无意义的内部差异。显然，从一开始就使用fragment要容易得多，既不用担心返工带来的麻烦和痛苦，也无需麻烦地记住应用每个部分使用了哪种的风格的界面控制。

因而，对于fragment，我们坚持AUF（Always Use Fragments）原则，即“总是使用fragment”原则。不值得为使用fragment还是activity而伤脑筋，相信我们，总是使用fragment！

7.8 深入学习：Honeycomb、ICS、Jelly Bean 以及更高版本系统上的应用开发

本章，对于SDK最低版本低于API 11级以下的项目，我们已经学过了如何利用支持库来使用fragment。然而，如只打算为最新版本设备开发应用，那么就没必要使用支持库了，我们可以直接使用标准库中的原生fragment类。

要想使用标准库里的fragment，需对项目做以下四处调整：

- 设置应用的编译目标及SDK最低版本为API 11级或更高；
- 放弃使用FragmentActivity类，转而使用标准库中的Activity类（android.app.Activity）。activity默认支持API 11级或更高版本中的fragment；
- 放弃使用android.support.v4.app.Fragment类，转而使用android.app.Fragment类；
- 放弃使用getSupportFragmentManager()方法获取FramentManager，转而使用getFragmentManager()方法。