第 16 章 操 作 栏 16

在Honeycomb版本系统中，Android引入了全新的操作栏。操作栏不仅取代了用来显示标题和应用图标的传统标题栏（title bar），还带来了更多其他功能，例如，安置菜单选项、配置应用图标作为导航按钮，等等。

本章，我们将为CriminalIntent应用创建一个菜单，并在其中提供可供用户新增crime记录的菜单项，然后让应用的图标支持向上的导航操作，如图16-1所示。

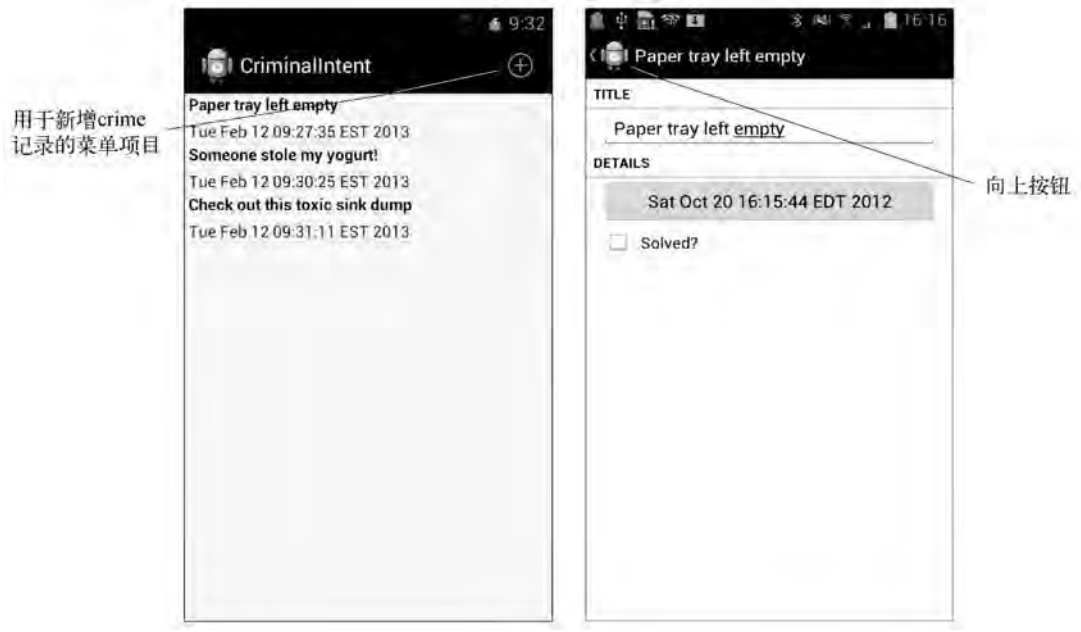

图16-1　创建选项菜单文件

16.1　选项菜单

可显示在操作栏上的菜单被称作选项菜单。选项菜单提供了一些选项，用户选择后可以弹出一个全屏activity界面，也可以退出当前应用。新增一条crime记录就是一个很好的例子。而从列表

中删除crime记录的操作，使用上下文菜单（context menu）来处理则更合适。因为删除记录的操作需要知道上下文信息，即应该删除哪一条crime记录。第18章，我们将学习如何使用上下文菜单。

本章的选项菜单以及第18章的上下文菜单均需要一些字符串资源。参照代码清单16-1，将这两章所需的字符串资源添加到string.xml文件中。虽然现在可能还不太明白这些新增的字符串资源，但有必要现在就完成添加。这样，在需要它们的时候，就可以直接使用，而无需停下手头的工作。

代码清单16-1　为菜单添加字符串资源（res/values/strings.xml）

```
    ...
    <string name="crimes_title">Crimes</string>
    <string name="crime_date_label">Date:</string>
    <string name="date_picker_title">Date of crime:</string>
    <string name="new_crime">New Crime</string>
    <string name="show_subtitle">Show Subtitle</string>
    <string name="hide_subtitle">Hide Subtitle</string>
    <string name="subtitle">If you see something, say something.</string>
    <string name="delete_crime">Delete</string>
</resources>
```

在操作栏上放置选项菜单虽然比较新颖，但选项菜单本身早在Android问世的时候就已经存在了，如图16-2所示。

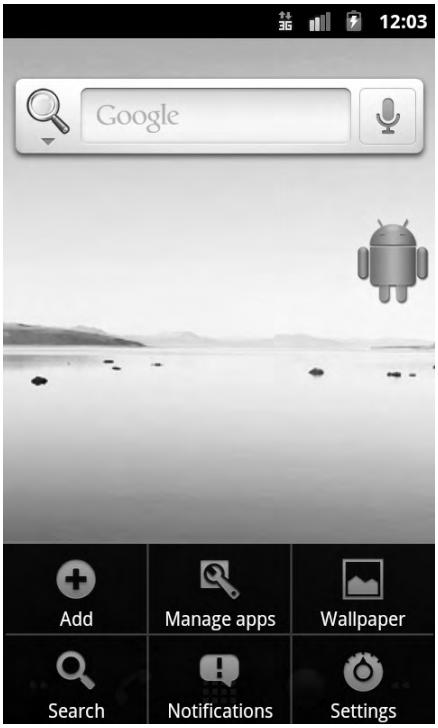

图16-2　Honeycomb以前的选项菜单

还不错，选项菜单基本没什么兼容性问题。不过，代码虽然都是一样的，根据不同的API级别，各设备呈现选项菜单的方式会稍有不同。本章后续学习过程中，我们会再来看看可能会涉及到的兼容性、选项菜单以及操作栏问题。

16.1.1 在XML文件中定义选项菜单

菜单是一种类似于布局的资源。创建一个定义菜单的XML文件，然后将其放置在项目的res/menu目录下。Android会自动生成该XML文件的对应资源ID，以供在代码中生成菜单之用。

在包浏览中，首先在res/目录下创建menu子目录。然后右键单击该新建目录，选择New → Android XML File菜单项。在弹出的窗口界面，确保选择了Menu文件资源类型，并命名新建文件为fragment_crime_list.xml，如图16-3所示。

图16-3 创建选项菜单文件

打开新建的fragment_crime_list.xml并切换到XML代码模式。参照代码清单16-2，添加新的item元素。

代码清单16-2 创建菜单资源（fragment_crime_list.xm）

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

```
    <item android:id="@+id/menu_item_new_crime"
      android:icon="@android:drawable/ic_menu_add"
      android:title="@string/new_crime"
      android:showAsAction="ifRoom|withText"/>
</menu>
```

showAsAction属性用于指定菜单选项是显示在操作栏上，还是隐藏到溢出菜单（overflow menu）中。该属性当前设置为ifRoom和withText的一个组合值。因此，只要空间足够，菜单项图标及其文字描述都会显示在操作栏上。如空间仅够显示菜单项图标，则不会显示文字描述。如空间大小不够任何一项显示，则菜单项会被转移隐藏到溢出菜单中。

如何访问溢出菜单取决于具体设备。如设备具有物理菜单键，则必须单击该键查看溢出菜单。目前，大多数较新设备都已取消物理菜单键，因此可通过操作栏最右端带有三个点的图标来访问溢出菜单，如图16-4所示。

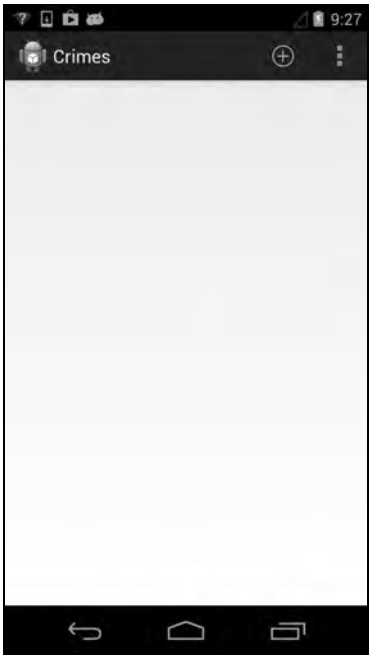

图16-4　操作栏中的溢出菜单

属性showAsAction还有另外两个可选值：always和never。不推荐使用always，应尽量使用更为方便的ifRoom属性值，让操作系统决定如何显示菜单项。对于那些很少用到的菜单项，使用never是个不错的选择。总的来说，为避免看到混乱的用户界面，只应将用户经常使用的菜单项放置在操作栏上。

注意，虽然android:showAsAction属性是在API 11级引入的，但Android Lint并没有报出兼容性问题。不同于Java代码，XML属性是不需要注解保护的。在早期API级别的设备上，后期新版本引入的XML属性会被自动忽略。

在属性android:icon中，@android:drawable/ic_menu_add值引用了一个系统图标资源。不要试图在项目资源里寻找系统图标资源，它只存在于设备上。

使用系统自带图标

在应用原型设计阶段，使用系统自带图标没有什么问题。然而，应用开发完成准备发布时，最好能保持统一的用户界面风格，而不是交由不同设备自行决定。要知道，不同设备或操作系统版本间，系统自带图标的显示风格往往具有很大差异。甚至有些设备自带的系统图标与应用的整体设计风格完全不搭。

一种解决方案是创建自己的定制图标。但需针对不同的屏幕显示密度或一些可能的设备配置，准备不同版本的图标。可访问http://developer.android.com/guide/practices/ui_guidelines/icon_design.html，查看Android的图标设计指南，了解更多相关信息。

还有一种解决方案，即找到满足应用要求的系统图标，将其直接复制到项目的drawable资源目录中。这种方式简单易行，可轻松获得风格一致、可打包到应用中的图标。

在Android SDK的安装目录下，可在类似your-android-SDK-home/platforms/android-API level/data/res的路径下找到系统图标。例如，在Mac电脑上，Android 4.2版本的图标资源路径为/Developer/android-sdk-mac_86/platforms/android-17/data/res。

根据当前工作的SDK版本，浏览或搜索找到ic_menu_add系统图标。也可将ic_menu_add系统图标复制到对应的项目资源目录中，然后修改布局文件的icon属性为android:icon="@drawable/ic_menu_add，从而实现从项目资源直接引用图标。

16.1.2 创建选项菜单

在代码中，Activity类提供了管理选项菜单的回调函数。在需要选项菜单时，Android会调用Activity的onCreateOptionsMenu(Menu)方法。

然而，按照CriminalIntent应用的设计，选项菜单相关的回调函数需在fragment而非activity里实现。不用担心，Fragment也有自己的一套选项菜单回调函数。稍后，我们会在CrimeListFragment中实现这些方法。以下为创建选项菜单和响应菜单项选择事件的两个回调方法：

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
public boolean onOptionsItemSelected(MenuItem item)
```

在CrimeListFragment.java中，覆盖onCreateOptionsMenu(Menu, MenuInflater)方法，实例化生成fragment_crime_list.xml中定义的菜单，如代码清单16-3所示。

代码清单16-3 实例化生成选项菜单（CrimeListFragment.java）

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    ((CrimeAdapter)getListAdapter()).notifyDataSetChanged();
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}
```

在以上方法中，调用MenuInflater.inflate(int, Menu)方法并传入菜单文件的资源ID，我们将文件中定义的菜单项目填充到Menu实例中。

注意，我们调用了超类的onCreateOptionsMenu(...)方法。虽然不是必须的，但作为一种约定的开发规范，我们推荐这么做。通过该超类方法的调用，任何超类定义的选项菜单功能在子类方法中也能获得应用。不过，这里的超类方法调用仅仅是遵循约定而已，因为Fragment超类的onCreateOptionsMenu(...)方法什么也没做。

Fragment的onCreateOptionsMenu(Menu, MenuInflater)方法是由FragmentManager负责调用的。因此，当activity接收到来自操作系统的onCreateOptionsMenu(...)方法回调请求时，我们必须明确告诉FragmentManager：其管理的fragment应接收onCreateOptionsMenu(...)方法的调用指令。要通知FragmentManager，需调用以下方法：

```
public void setHasOptionsMenu(boolean hasMenu)
```

在CrimeListFragment.onCreate(...)方法中，通知FragmentManager：CrimeListFragment需接收选项菜单方法回调。如代码清单16-4所示。

代码清单16-4　调用SethasOptionsMenu方法（CrimeListFragment.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);

    getActivity().setTitle(R.string.crimes_title);

    ...
```

运行CriminalIntent应用，查看新创建的选项菜单，如图16-5所示。

图16-5　显示在操作栏上的菜单项图标

菜单项标题怎么没有显示？大多数设备在竖直模式下屏幕空间都有限，因此，应用的操作栏上只够显示菜单项图标。长按操作栏上的菜单图标，可弹出菜单标题，如图16-6所示。

图16-6 长按操作栏上的图标，显示菜单项标题

水平模式下，操作栏上会有足够的空间同时显示菜单图标和菜单项标题，如图16-7所示。

图16-7 同时显示在操作栏上的菜单图标和菜单标题

在Honeycomb以前的系统版本设备上运行CriminalIntent应用，要查看会出现在屏幕底部的选项菜单，必须点按设备内置的菜单键。图16-8为CriminalIntent应用在Gingerbread设备上运行的效果。

图16-8　Gingerbread设备上的选项菜单

虽然没有使用类似`@TargetApi(11)`的注解保护，选项菜单实现代码依然能够很好的兼容新旧设备。不过，新老SDK版本在选项菜单的处理上仍存在一些细微差别。在运行Honeycomb及后续系统版本的设备上，应在activity启动后，调用`onCreateOptionsMenu(...)`方法并创建选项菜单。为保证菜单项能够显示在操作栏上，选项菜单的创建在activity生命周期的一开始就完成显然是必须的。而在较旧的系统版本设备上，应在用户首次点按菜单键时，调用`onCreateOptionsMenu(...)`方法并创建选项菜单。

（也许我们看到过在旧版本设备上运行的带有操作栏的应用。通常，这些应用都是基于一个名为ActionBarSherlock的第三方库开发的。该库通过模仿复制为旧版本设备实现了操作栏的功能。在第18章末尾我们将详细介绍有关ActionBarSherlock库的使用。）

16.1.3　响应菜单项选择

为响应用户点击New Crime菜单项，需实现新方法以添加新的`Crime`到crime数组列表中。在CrimeLab.java中，新增以下方法，实现添加`Crime`到数组列表中，如代码清单16-5所示。

代码清单16-5　添加新的crime（CrimeLab.java）

```
...

public void addCrime(Crime c) {
    mCrimes.add(c);
}
```

```
public ArrayList<Crime> getCrimes() {
    return mCrimes;
}

...
```

既然可以手动添加crime记录，也就没必要再让程序自动生成100条crime记录了。在CrimeLab.java中，删除生成随机crime记录的代码，如代码清单16-6所示。

代码清单16-6 再见，随机crime记录！（CrimeLab.java）

```
public CrimeLab(Context appContext) {
    mAppContext = appContext;
    mCrimes = new ArrayList<Crime>();
    for (int i = 0; i < 100; i++) {
        Crime c = new Crime();
        c.setTitle("Crime #" + i);
        c.setDate(new Date());
        c.setSolved(i % 2 == 0); // Every other one
        mCrimes.add(c);
    }
}
```

用户点击选项菜单中的菜单项时，fragment会收到`onOptionsItemSelected(MenuItem)`方法的回调请求。该方法接受的传入参数是一个描述用户选择的`MenuItem`实例。

尽管当前的选项菜单只包含一个菜单项，但通常菜单可包含多个菜单项。通过检查菜单项ID，可确定被选中的是哪一个菜单项，然后做出相应的响应。代码中使用的菜单项ID实际就是在菜单XML定义文件中赋予菜单项的资源ID。

在CrimeListFragment.java中，实现`onOptionsItemSelected(MenuItem)`方法响应菜单项的选择事件。在该方法中，创建一个新的`Crime`实例，并将其添加到`CrimeLab`中，然后启动一个`CrimePagerActivity`实例，让用户可以编辑新创建的`Crime`记录，如代码清单16-7所示。

代码清单16-7 响应菜单项选择事件（CrimeListFragment.java）

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            Crime crime = new Crime();
            CrimeLab.get(getActivity()).addCrime(crime);
            Intent i = new Intent(getActivity(), CrimePagerActivity.class);
            i.putExtra(CrimeFragment.EXTRA_CRIME_ID, crime.getId());
            startActivityForResult(i, 0);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

注意，onOptionsItemSelected(MenuItem)方法返回的是布尔值。一旦完成菜单项事件处理，应返回true值以表明已完成菜单项选择需要处理的全部任务。另外，case表达式中，如果菜单项ID不存在，默认的超类版本方法会被调用。

运行CriminalIntent应用，尝试使用选项菜单，添加一些crime记录并对它们进行编辑。

16.2　实现层级式导航

目前为止，CriminalIntent应用主要依靠*后退*键在应用内导航。使用*后退*键的导航又称为*临时性导航*，只能返回到上一次的用户界面。而Ancestral navigation，有时也称为*层级式导航*（hierarchical navigation），可逐级向上在应用内导航。

Android可轻松利用操作栏上的应用图标实现层级式导航。也可利用应用图标实现直接回退至主屏，即逐级向上直至应用的初始界面。实际上，操作栏上的应用图标最初是用作Home键的。不过，Android现在只推荐利用应用图标，实现向上回退一级至当前activity的父界面。这样一来，应用图标实际上就起到了向上按钮的作用。

本节中，针对显示在CrimePagerActivity操作栏上的应用图标，我们将编码使其具有向上按钮的功能。点击该图标，可回退至crime列表界面。

16.2.1　启用应用图标的导航功能

通常，应用图标一旦启用了向上导航按钮的功能，在应用图标的左边会显示一个如图16-9所示的向左指向图标。

为启用应用图标向上导航按钮的功能，并在fragment视图上显示向左的图标，须调用以下方法设置fragment的DisplayHomeAsUpEnabled属性：

```
public abstract void setDisplayHomeAsUpEnabled(boolean showHomeAsUp)
```

该方法来自于API 11级，因此需进行系统版本判断保证应用向下兼容，并使用@TargetApi(11)注解阻止Android Lint报告兼容性问题。

在CrimeFragment.onCreateView(...)中，调用setDisplayHomeAsUpEnabled(true)方法，如代码清单16-8所示。

代码清单16-8　启用向上导航按钮（CrimeFragment.java）

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        getActivity().getActionBar().setDisplayHomeAsUpEnabled(true);
    }

    ...
}
```

图16-9 带有向上导航按钮的操作栏

注意，调用setDisplayHomeAsUpEnabled(...)方法只是让应用图标转变为按钮，并显示一个向左的图标而已。因此我们必须进行编码，实现点击按钮可向上逐级回退的功能。对于那些以API 11-13级别为目标版本开发的应用，应用图标已默认启用为向上按钮的功能，但仍需调用setDisplayHomeAsUpEnabled(true)方法，以在应用图标左边显示向左的指向图标。

在代码清单16-8中，我们对onCreateView(...)方法使用了@TargetApi注解。实际上只注解setDisplayHomeAsUpEnabled(true)方法的调用即可。不过，onCreateView(...)方法很快就会有一些特定API级别的代码加入，因此，这里我们选择直接注解整个onCreateView(...)实施方法。

16.2.2 响应向上按钮

如同响应选项菜单项那样，可通过覆盖onOptionsItemSelected(MenuItem)方法的方式，响应已启用向上按钮功能的应用图标。因此，首先应通知FragmentManager：CrimeFragment将代表其托管activity实现选项菜单相关的回调方法。如同前面对CrimeListFragment的处理一样，在CrimeFragment.onCreate(...)方法中，调用setHasOptionsMenu(true)方法，如代码清单16-9所示。

代码清单16-9 开启选项菜单处理（CrimeFragment.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {

    ...
```

```
        setHasOptionsMenu(true);
    }
```

无需在XML文件中定义或生成应用图标菜单项。它已具有现成的资源ID：android.R.id.home。在CrimeFragment.java中，覆盖onOptionsItemSelected(MenuItem)方法，响应用户对该菜单项的点击事件，如代码清单16-10所示。

代码清单16-10　响应应用图标（Home键）菜单项（CrimeFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            // To be implemented next
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

为实现用户点击向上按钮返回至crime列表界面，我们可能会想到去创建一个intent，然后启动CrimePagerActivity实例，如以下实现代码：

```
Intent intent = new Intent(getActivity(), CrimeListActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
finish();
```

FLAG_ACTIVITY_CLEAR_TOP指示Android在回退栈中寻找指定activity的存在实例，如图16-10所示。如存在，则弹出栈中的所有其他activity，让启动的activity出现在栈顶，从而显示在屏幕上。

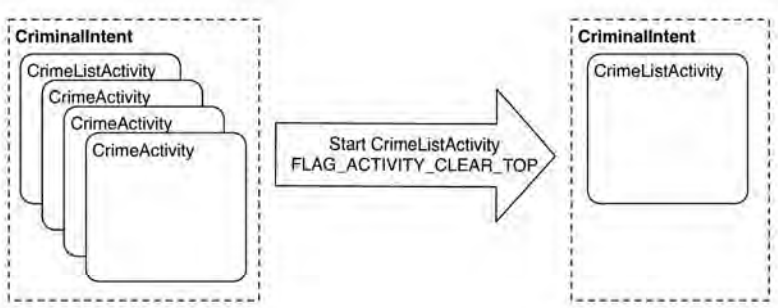

图16-10　工作中的FLAG_ACTIVITY_CLEAR_TOP

然而，Android有更好的办法实现层级式导航：配合使用NavUtils便利类与manifest配置文件中的元数据。

先来处理元数据。打开AndroidManifest.xml文件，在CrimePagerActivity声明中添加新的meta-data属性，指定CrimePagerActivity的父类为CrimeListActivity，如代码清单16-11所示。

代码清单16-11 添加父activity元数据属性（AndroidManifest.xml）

```
<activity android:name=".CrimePagerActivity"
  android:label="@string/app_name">
  <meta-data android:name="android.support.PARENT_ACTIVITY"
    android:value=".CrimeListActivity"/>
</activity>
...
```

把元数据标签想象为张贴在activity上的一个便利贴。类似这样的便利贴信息都保存在系统的PackageManager中。只要知道便利贴的名字，任何人都可以获取它的内容。也可创建自己的名–值（name-value）对以便在需要的时候获取它们。这种特别的名-值对由NavUtils类定义，这样它就能知道谁是指定activity的父类，配合以下NavUtils类方法一起使用尤其有用：

```
public static void navigateUpFromSameTask(Activity sourceActivity)
```

在CrimeFragment.onOptionsItemSelected(...)方法中，首先通过调用NavUtils.getParentActivityName(Activity)方法，检查元数据中是否指定了父activity。如指定有父activity，则调用navigateUpFromSameTask(Activity)方法，导航至父activity界面。如代码清单16-12所示。

代码清单16-12 使用NavUtils类（CrimeFragment.java）

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            if (NavUtils.getParentActivityName(getActivity()) != null) {
                NavUtils.navigateUpFromSameTask(getActivity());
            }
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

如元数据中未指定父activity，则为避免误导用户，无需再显示向左的箭头图标。回到onCreateView(...)方法中，在调用setDisplayHomeAsUpEnabled(true)方法前，先检查父activity是否存在，如代码清单16-13所示。

代码清单16-13 控制导航图标的显示（CrimeFragment.java）

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        if (NavUtils.getParentActivityName(getActivity()) != null) {
            getActivity().getActionBar().setDisplayHomeAsUpEnabled(true);
        }
    }

    ...
}
```

为什么使用NavUtils类要好于手动启动activity？首先，NavUtils类的实现代码既简洁又优雅。其次，使用NavUtils类也可实现在manifest配置文件中统一管理activity间的关系。如果activity间的关系发生改变，无需费力地去修改Java代码，我们只要简单修改配置文件中的一行代码即可。

除此之外，使用NavUtils类还可保持层级关系处理与fragment的代码相分离。这样，即使在各个具有不同父类的activity中使用同一CrimeFragment， CrimeFragment依然能正常工作。

运行CriminalIntent应用。创建新的crime记录，然后点击应用图标，返回至crime列表界面。实际上，CriminalIntent应用两个层级的关系并不是太容易区分。但navigateUpFromSameTask(Activity)方法实现了向上导航的功能，使得用户可以轻松地向上导航一级至CrimePagerActivity的父类界面。

16.3　可选菜单项

本小节，利用前面学过的有关菜单、应用兼容性以及可选资源的知识，我们添加一个菜单项实现显示或隐藏CrimeListActivity操作栏的子标题。

16.3.1　创建可选菜单XML文件

对于使用Honeycomb之前系统版本的用户，只适用于操作栏的菜单项对他们来说应该是不可见的。因此，首先应创建可供API 11级以后的系统版本使用的可选菜单资源。在项目的res目录下创建一个menu-v11目录，然后将fragment_crime_list.xml文件复制到该目录中。

打开res/menu-v11/fragment_crime_list.xml文件，参照代码清单16-14，新增一个显示为Show Subtitle的菜单项。如空间足够，它将显示在操作栏上。

代码清单16-14　添加Show Subtitle菜单项（res/menu-v11/fragment_crime_list.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        android:showAsAction="ifRoom|withText"/>
    <item android:id="@+id/menu_item_show_subtitle"
        android:title="@string/show_subtitle"
        android:showAsAction="ifRoom"/>
</menu>
```

在onOptionsItemSelected(...)方法中，设置操作栏的子标题以响应菜单项的单击事件，如代码清单16-15所示。

代码清单16-15　响应Show Subtitle菜单项单击事件（CrimeListFragment.java）

```
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
```

```
            case R.id.menu_item_new_crime:
                ...
                return true;
            case R.id.menu_item_show_subtitle:
                getActivity().getActionBar().setSubtitle(R.string.subtitle);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
```

注意，这里只是在代码中使用@TargetApi(11)注解阻止Android Lint报告兼容性问题。而操作栏的相关代码并没有置于版本条件判断之中。在早期版本的设备上，R.id.menu_item_show_subtitle不会出现，自然也就不会调用操作栏相关代码，所以这里没必要处理设备兼容性问题。

在新设备上运行CriminalIntent应用，使用新增菜单显示子标题。然后在Froyo或Gingerbread设备上（虚拟设备或实体设备皆可）运行应用。点按菜单键，确认Show Subtitle没有显示。最后，添加新的crime记录，确认应用运行如前。

16.3.2 切换菜单项标题

操作栏上的子标题显示后，菜单项标题依然显示为Show Subtitle。如果菜单项标题的切换与子标题的显示或隐藏能够联动，用户体验会更好。

在onOptionsItemSelected(...)方法中，选中菜单项后，检查子标题的显示状态并采取相应操作，如代码清单16-16所示。

代码清单16-16 实现菜单项标题与子标题的联动显示（CrimeListFragment.java）

```
    @TargetApi(11)
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_item_new_crime:
                ...
                return true;
            case R.id.menu_item_show_subtitle:
                if (getActivity().getActionBar().getSubtitle() == null) {
                    getActivity().getActionBar().setSubtitle(R.string.subtitle);
                    item.setTitle(R.string.hide_subtitle);
                } else {
                    getActivity().getActionBar().setSubtitle(null);
                    item.setTitle(R.string.show_subtitle);
                }
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
```

如果操作栏上没有显示子标题，则应设置显示子标题，同时切换菜单项标题，使其显示为Hide Subtitle。如果子标题已经显示，则应设置其为null值，同时将菜单项标题切换回Show Subtitle。

运行CriminalIntent应用，确认菜单项标题与子标题的显示能够联动。

16.3.3 “还有个问题”

Android编程如同回答神探科伦坡的盘问。你以为你的回答丝丝入扣、天衣无缝，可以高枕无忧了。但每次都会被Android堵在门口提醒道：“还有一个问题。”

这个问题就是经典的设备旋转问题。子标题显示后，旋转设备，这时因为用户界面的重新生成，显示的子标题会消失。为解决此问题，需要一个实例变量记录子标题的显示状态，并且设置保留`CrimeListFragment`，使得变量值在设备旋转后依然可用。

在CrimeListFragment.java中，添加一个布尔类型的成员变量，在`onCreate(...)`方法中保留`CrimeListFragment`并对变量进行初始化，如代码清单16-17所示。

代码清单16-17　保留`CrimeListFragment`并初始化变量（CrimeListFragment.java）

```
public class CrimeListFragment extends ListFragment {
    private ArrayList<Crime> mCrimes;
    private boolean mSubtitleVisible;
    private final String TAG = "CrimeListFragment";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        setRetainInstance(true);
        mSubtitleVisible = false;
    }
```

然后在`onOptionsItemSelected(...)`方法中，根据菜单项的选择设置对应的变量值，如代码清单16-18所示。

代码清单16-18　根据菜单项的选择设置`subtitleVisible` (CrimeListFragment.java)

```
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
            return true;
        case R.id.menu_item_show_subtitle:
            if (getActivity().getActionBar().getSubtitle() == null) {
                getActivity().getActionBar().setSubtitle(R.string.subtitle);
                mSubtitleVisible = true;
                item.setTitle(R.string.hide_subtitle);
            }
            else {
                getActivity().getActionBar().setSubtitle(null);
                mSubtitleVisible = false;
                item.setTitle(R.string.show_subtitle);
            }
            return true;
        default:
```

```
            return super.onOptionsItemSelected(item);
        }
    }
```

现在需要查看设备旋转后是否应该显示子标题。在CrimeListFragment.java中，覆盖onCreateView(...)方法，根据变量mSubtitleVisible的值确定是否要设置子标题，如代码清单16-19所示。

代码清单16-19 根据变量mSubtitleVisible的值设置子标题（CrimeListFragment.java）

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
    View v = super.onCreateView(inflater, parent, savedInstanceState);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        if (mSubtitleVisible) {
            getActivity().getActionBar().setSubtitle(R.string.subtitle);
        }
    }

    return v;
}
```

同时需要在onCreateOptionsMenu(...)方法中查看子标题的状态，以保证菜单项标题与之匹配显示，如代码清单16-20所示。

代码清单16-20 基于mSubtitleVisible变量的值，正确显示菜单项标题（CrimeListFragment.java）

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
    MenuItem showSubtitle = menu.findItem(R.id.menu_item_show_subtitle);
    if (mSubtitleVisible && showSubtitle != null) {
        showSubtitle.setTitle(R.string.hide_subtitle);
    }
}
```

运行CriminalIntent应用。显示子标题并旋转设备，可以看到子标题在重新创建的视图中依然能正确显示。

16.4 挑战练习：用于列表的空视图

当前，CriminalIntent应用启动后，会显示一个空白列表。从用户界面友好的角度来讲，即使列表中没有任何crime记录可以显示，也应展示一些用户友好信息。

作为AdapterView的子类，ListView支持显示一种被称为“空视图”的特殊View。该空视图适用于CriminalIntent应用刚才所述的场景。如为空视图指定一个具体视图，ListView可自动切换于两种视图模式之间。也就是说，没有crime记录可以显示时，就显示空视图；有crime记录可以显示时，就显示列表视图。

使用下列AdapterView方法，在代码中指定空视图：

```
public void setEmptyView(View emptyView)
```

也可创建XML布局文件，同时定义ListView和空视图。然后将@android:id/list和@android:id/empty资源ID分别赋予给它们，以实现ListView在两种视图模式之间的自动切换。

CrimeListFragment类当前没有在onCreateView(...)方法中实例化自己的布局，但为了利用布局实施空视图，它必须要做布局实例化。因此，为CrimeListFragment类创建一个XML布局资源。该布局使用FrameLayout作为根容器，并同时包含一个ListView视图和一个空视图的View。

设置空视图显示类似“没有crime记录可以显示”的信息。再添加一个按钮到该视图，方便用户点击时直接创建新的crime记录，这样，用户就不必再去选项菜单或操作栏上操作了。