第 18 章

18 上下文菜单与上下文操作模式

本章，我们将为应用实现长按列表项删除crime记录的功能。删除一条crime记录是一种上下文操作（contextual action），即它是与某个特定屏幕视图（单个列表项）而非整个屏幕相关联的。

在Honeycomb以前版本的设备上，上下文操作是在浮动上下文菜单中呈现的。而在之后版本的设备上，上下文操作主要是通过上下文操作栏呈现的。位于activity的操作栏之上，上下文操作栏为用户提供了各种操作，如图18-1所示。

图18-1　长按列表项删除一条crime记录

第16章我们已看到，对于选项菜单而言，处理不同API级别的兼容性问题很简单：只需定义一种菜单资源并实现一组菜单相关的回调方法，不同设备上的操作系统会自行决定菜单项的显示方式。

而对于上下文操作，事情就没那么简单了。虽然还是定义一种菜单资源，但我们必须实现两组不同的回调方法，一组用于上下文操作栏，一组用于浮动上下文菜单。

本章，我们将在运行API 11级及以上系统版本的设备上实施一个上下文操作，然后，再在Froyo及Gingerbread设备上实施一个浮动上下文菜单。

（我们也许见过旧设备上运行的带有上下文操作栏的应用。通常来说，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系统设备实现了上下文操作栏的功能。本章末尾我们将详细讨论ActionBarSherlock库的使用。）

18.1 定义上下文菜单资源

在res/menu/目录中，以menu为根元素，新建名为crime_list_item_context.xml的菜单资源文件。然后参照代码清单18-1添加需要的菜单项。

代码清单18-1 用于crime列表的上文菜单（crime_list_item_context.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_delete_crime"
      android:icon="@android:drawable/ic_menu_delete"
      android:title="@string/delete_crime" />
</menu>
```

以上定义的菜单资源将用于上下文操作栏和浮动上下文菜单的实施。

18.2 实施浮动上下文菜单

首先，我们来创建浮动上下文菜单。Fragment的回调方法类似于第16章中用于选项菜单的回调方法。要实例化生成一个上下文菜单，可使用以下方法：

```
public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenu.ContextMenuInfo menuInfo)
```

要响应用户的上下文菜单选择，可实现以下Fragment方法：

```
public boolean onContextItemSelected(MenuItem item)
```

18.2.1 创建上下文菜单

在CrimeListFragment.java中，实现onCreateContextMenu(...)方法，实例化菜单资源，并用它填充上下文菜单，如代码清单18-2所示。

代码清单18-2 创建上下文菜单（CrimeListFragment.java）

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

```
            ...

            default:
                return super.onOptionsItemSelected(item);
        }
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
        getActivity().getMenuInflater().inflate(R.menu.crime_list_item_context, menu);
    }
```

不像onCreateOptionsMenu(...)方法，以上菜单回调方法不接受MenuInflater实例参数，因此，我们应首先获得与CrimeListActivity关联的MenuInflater。然后调用MenuInflater.inflate(...)方法，传入菜单资源ID和ContextMenu实例，用菜单资源文件中定义的菜单项填充菜单实例，从而完成上下文菜单的创建。

当前我们只定义了一个上下文菜单资源，因此，无论用户长按的是哪个视图，菜单都是以该资源实例化生成的。假如定义了多个上下文菜单资源，通过检查传入onCreateContextMenu(...)方法的View视图ID，我们可以自由决定使用哪个资源来生成上下文菜单。

18.2.2　为上下文菜单登记视图

默认情况下，长按视图不会触发上下文菜单的创建。要触发菜单的创建，必须调用以下Fragment方法为浮动上下文菜单登记一个视图：

```
public void registerForContextMenu(View view)
```

该方法需传入触发上下文菜单的视图。

在CriminalIntent应用里，我们希望点击任意列表项，都能弹出上下文菜单。这岂不是意味着需要逐个登记列表项视图吗？不用那么麻烦，直接登记ListView视图即可，然后它会自动登记各个列表项视图。

在CrimeListFragment.onCreateView(...)方法中，引用并登记ListView，如代码清单18-3所示。

代码清单18-3　为上下文菜单登记ListView（CrimeListFragment.java）

```
@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);
         }
    }

    ListView listView = (ListView)v.findViewById(android.R.id.list);
    registerForContextMenu(listView);

    return v;
}
```

在onCreateView(...)方法中，使用android.R.id.list资源ID获取ListFragment管理着的ListView。ListFragment也有一个getListView()方法，但在onCreateView(...)方法中却无法使用。这是因为，在onCreateView(...)方法完成调用并返回视图之前，getListView()方法返回的永远是null值。

运行CriminalIntent应用，长按任意列表项，确认可弹出具有Delete菜单项的浮动上下文菜单，如图18-2所示。

图18-2 长按列表项弹出上下文菜单项

18.2.3 响应菜单项选择

Delete菜单项要可用，需要一个能够从模型层删除crime数据的方法。在CrimeLab.java中，新增deleteCrime(Crime)方法，如代码清单18-4所示。

代码清单18-4 新增删除crime的方法（CrimeLab.java）

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

public void deleteCrime(Crime c) {
    mCrimes.remove(c);
}
```

然后，在onContextItemSelected(MenuItem)方法中处理菜单项选择事件。MenuItem有一个资源ID可用于识别选中的菜单项。除此之外，还需明确具体要删除的crime对象，才能确定用户想要删除crime数据的意图。

可调用MenuItem的getMenuInfo()方法，获取要删除的crime对象的信息。该方法返回一个实现了ContextMenu.ContextMenuInfo接口的类实例。

在CrimeListFragment中，新增onContextItemSelected(MenuItem)实现方法，使用menu信息和adapter，确定被长按的Crime对象，然后从模型层数据中删除它，如代码清单18-5所示。

代码清单18-5　监听上下文菜单项选择事件（CrimeListFragment.java）

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
    getActivity().getMenuInflater().inflate(R.menu.crime_list_item_context, menu);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo)item.getMenuInfo();
    int position = info.position;
    CrimeAdapter adapter = (CrimeAdapter)getListAdapter();
    Crime crime = adapter.getItem(position);

    switch (item.getItemId()) {
        case R.id.menu_item_delete_crime:
            CrimeLab.get(getActivity()).deleteCrime(crime);
            adapter.notifyDataSetChanged();
            return true;
    }
    return super.onContextItemSelected(item);
}
```

以上代码中，因为ListView是AdapterView的子类，所以getMenuInfo()方法返回了一个AdapterView.AdapterContextMenuInfo实例。然后，将getMenuInfo()方法的返回结果进行类型转换，获取选中列表项在数据集中的位置信息。最后，使用列表项的位置，获取要删除的Crime对象。

运行CriminalIntent应用，新增一条crime记录，然后长按删除它。（要在模拟器上模拟长按动作，可按下鼠标左键不放直到菜单弹出。）

18.3　实施上下文操作模式

通过浮动上下文菜单删除crime记录的实现代码，可在任何Android设备上运行。例如，图18-2为Jelly Bean系统设备上弹出的浮动菜单。

然而，在新系统设备上，长按视图进入上下文操作模式是提供上下文操作的主流方式。屏幕进入上下文操作模式时，上下文菜单中定义的菜单项会出现在覆盖着操作栏的上下文操作栏上，如图18-3所示。相比浮动菜单，上下文操作栏不会遮挡屏幕，因此是更好的菜单展现方式。

上下文操作栏的实现代码不同于浮动上下文菜单。此外，上下文操作栏实现代码所使用的类

和方法不支持Froyo或Gingerbread等老系统，因此必须保证仅支持新系统的代码在老系统上不会被调用。

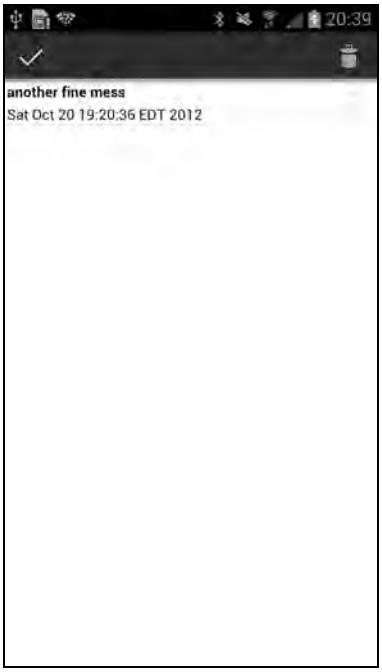

图18-3 长按列表项出现上下文操作栏

18.3.1 实现列表视图的多选操作

列表视图进入上下文操作模式时，可开启它的多选模式。多选模式下，上下文操作栏上的任何操作都将同时应用于所有已选视图。

在CrimeListFragment.onCreateView(...)方法中，设置列表视图的选择模式为CHOICE_MODE_MULTIPLE_MODAL，如代码清单18-6所示。最后，为处理兼容性问题，记得使用编译版本常量，将登记ListView的代码与设置选择模式的代码区分开来。

代码清单18-6 设置列表视图的选择模式（CrimeListFragment.java）

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

    ...

    ListView listView = (ListView)v.findViewById(android.R.id.list);
```

```
        if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
            // Use floating context menus on Froyo and Gingerbread
            registerForContextMenu(listView);
        } else {
            // Use contextual action bar on Honeycomb and higher
            listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
        }

        return v;
    }
```

18.3.2　列表视图中的操作模式回调方法

接下来，为ListView设置一个实现AbsListView.MultiChoiceModeListener接口的监听器。该接口包含以下回调方法，视图在选中或撤销选中时会触发它：

```
public abstract void onItemCheckedStateChanged(ActionMode mode, int position,
    long id, boolean checked)
```

MultiChoiceModeListener实现了另一个接口，即ActionMode.Callback。用户屏幕进入上下文操作模式时，会创建一个ActionMode类实例。随后在其生命周期内，ActionMode.Callback接口的回调方法会在不同时点被调用。以下为ActionMode.Callback接口中必须实现的四个方法：

```
public abstract boolean onCreateActionMode(ActionMode mode, Menu menu)
```

在ActionMode对象创建后调用。也是实例化上下文菜单资源，并显示在上下文操作栏上的任务完成的地方。

```
public abstract boolean onPrepareActionMode(ActionMode mode, Menu menu)
```

在onCreateActionMode(...)方法之后，以及当前上下文操作栏需要刷新显示新数据时调用。

```
public abstract boolean onActionItemClicked(ActionMode mode, MenuItem item)
```

在用户选中某个菜单项操作时调用。是响应上下文菜单项操作的地方。

```
public abstract void onDestroyActionMode(ActionMode mode)
```

在用户退出上下文操作模式或所选菜单项操作已被响应，从而导致ActionMode对象将要销毁时调用。默认的实现会导致已选视图被反选。这里，也可完成在上下文操作模式下，响应菜单项操作而引发的相应fragment更新。

在CrimeListFragment.onCreateView(...)方法中，为列表视图设置实现MultiChoiceModeListener接口的监听器。这里，只需实现onCreateActionMode(...)和onActionItemClicked(ActionMode, MenuItem)方法即可，如代码清单18-7所示。

代码清单18-7　设置MultiChoiceModeListener监听器（CrimeListFragment.java）

```
...
} else {
    listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
    listView.setMultiChoiceModeListener(new MultiChoiceModeListener() {
```

```
            public void onItemCheckedStateChanged(ActionMode mode, int position,
                    long id, boolean checked) {
                // Required, but not used in this implementation
            }

            // ActionMode.Callback methods
            public boolean onCreateActionMode(ActionMode mode, Menu menu) {
                MenuInflater inflater = mode.getMenuInflater();
                inflater.inflate(R.menu.crime_list_item_context, menu);
                return true;
            }

            public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
                return false;
                // Required, but not used in this implementation
            }

            public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
                switch (item.getItemId()) {
                    case R.id.menu_item_delete_crime:
                        CrimeAdapter adapter = (CrimeAdapter)getListAdapter();
                        CrimeLab crimeLab = CrimeLab.get(getActivity());
                        for (int i = adapter.getCount() - 1; i >= 0; i--) {
                            if (getListView().isItemChecked(i)) {
                                crimeLab.deleteCrime(adapter.getItem(i));
                            }
                        }
                        mode.finish();
                        adapter.notifyDataSetChanged();
                        return true;
                    default:
                        return false;
                }
            }

            public void onDestroyActionMode(ActionMode mode) {
                // Required, but not used in this implementation
            }
        });
        }
        return v;
    }
```

注意，如使用Eclipse的代码自动补全来创建`MultiChoiceModeListener`接口，系统自动产生的`onCreateActionMode(...)`存根方法会返回false值。记得将其改为返回true值，因为返回false值会导致操作模式创建失败。

另外要注意的是，在`onCreateActionMode(...)`方法中，我们是从操作模式，而非activity中获取`MenuInflater`的。操作模式负责对上下文操作栏进行配置。例如，可调用`ActionMode.setTitle(...)`方法为上下文操作栏设置标题，而activity的`MenuInflater`则做不到这一点。

接下来，在`onActionItemClicked(...)`方法中，响应菜单项删除操作，从`CrimeLab`中删除一个或多个`Crime`对象，然后重新加载显示列表。最后，调用`ActionMode.finish()`方法准备销毁操作模式。

运行CriminalIntent应用。长按选择任意列表项，进入上下文操作模式。此时，还要选择其他

列表项的话，直接点击即可。而再次点击已选中的列表项则撤销选择。点击删除图标将结束操作模式并返回到刷新后的列表项界面。也可以点击上下文操作栏最左边的取消图标，这将取消操作模式并返回到没有任何变化的列表项界面。

图18-4　第二、三项crime记录已被选中

如图18-4所示，尽管功能使用上没有什么问题，但用户的使用体验很糟糕，因为很难看出哪些列表项被选中了。不过，该问题可通过改变已选中列表项背景的方式解决。

18.3.3　改变已激活视图的显示背景

依据自身的不同状态，有时需要差别化地显示某个视图。CriminalIntent应用中，在列表项处于激活状态时，我们希望能够改变其显示背景。视图处于激活状态，是指该视图已被用户标记为关注处理对象。

基于视图的状态，可使用state list drawable资源来改变其显示背景。state list drawable是一种以XML定义的资源。该资源定义中，我们指定一个drawable（位图或彩图），并列出该drawable对应的状态。（可查阅`StateListDrawable`参考手册页，了解更多视图相关状态。）

不像其他drawable资源，state list drawable资源通常和屏幕显示像素密度无关，因而只存放在不带修饰符的drawable目录中。创建一个res/drawable目录，然后以selector为根元素在该目录下新建一个名为background_activated.xml的文件。参照代码清单18-8完成内容的添加。

代码清单18-8　简单的state list drawable资源（res/drawable/background_activated.xml）

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

```
    <item
      android:state_activated="true"
      android:drawable="@android:color/darker_gray"
      />
</selector>
```

以上XML文件告诉我们：当引用该drawable资源的视图处于激活状态时，则使用android:drawable属性值指定的资源；反之，则不采取任何操作。如android:state_activated的属性值设置为false，则只要视图未处于激活状态，android:drawable指定的资源都会被使用。

修改res/layout/list_item_crime.xml文件，引用drawable目录下background_activated.xml定义的资源，如代码清单18-9所示。

代码清单18-9 改变列表项的显示背景（res/layout/list_item_crime.xml）

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:background="@drawable/background_activated">
  ...
</RelativeLayout>
```

重新运行CriminalIntent应用。这次，已选列表项一眼便能看出了，如图18-5所示。

图18-5 醒目的第二、三列表项

第25章，我们会学习到更多有关state list drawable的内容。

18.3.4 实现其他视图的上下文操作模式

本章实现的上下文菜单栏可以完美地应用于ListView和GridView。（GridView是AdapterView的子类，第26章中会用到它。）但如果视图既非ListView，也非GridView，要使用上下文操作栏

又该如何处理呢?

首先，设置一个实现View.OnLongClickListener接口的监听器。然后在监听器实现体内，调用Activity.startActionMode(...)方法创建一个ActionMode实例。(前面已经看到，如果使用的是MultiChoiceModeListener接口，ActionMode实例是自动创建的。)

startActionMode(...)方法需要一个实现ActionMode.Callback接口的对象作为参数。因此，创建一个ActionMode.Callback接口的实现，该接口实现自然也包括前面使用过的四个ActionMode生命周期方法：

- public abstract boolean onCreateActionMode(ActionMode mode, Menu menu)
- public abstract boolean onPrepareActionMode(ActionMode mode, Menu menu)
- public abstract boolean onActionItemClicked(ActionMode mode, MenuItem item)
- public abstract void onDestroyActionMode(ActionMode mode)

具体实现时,可先创建一个实现ActionMode.Callback接口的对象，然后在调用startActionMode(...)方法时传入，或直接调用startActionMode(...)方法并传入一个匿名实现参数。

18.4　兼容性问题：回退还是复制

本章，我们已经用过一种叫做“优雅的回退”的兼容性策略。优雅的回退是指，应用在新系统平台上运行时，可自动使用新平台的特色功能及代码，而在旧系统平台上运行时，则回退使用早期的特色功能。从技术手段角度来说，这是通过在运行时检查SDK版本来实现的。

优雅的回退并非是唯一可用的策略。通过模仿，旧平台可提供类似于新平台的特色功能，而不用再回退使用旧平台的功能。这种模仿策略也称为复制。复制策略又细分为两种：

- 按需复制。仅在老平台上使用复制功能。
- 替换复制。无论新老平台都使用复制功能，即便是在有原生支持的新平台上。

借助支持库使用fragment就是替换复制策略的运用。即使应用运行的设备支持使用android.app.Fragment原生类,应用也总是使用android.support.v4.app.Fragment替换类。

Android支持库并不包含复制版本的操作栏，不过我们可使用一些第三方的复制版本。如果看到运行在Gingerbread设备上的应用支持操作栏，则该应用肯定使用了某个第三方支持库。目前最优秀、应用最广的第三方支持库是Jake Wharton开发的ActionBarSherlock，可登陆网站http://www.actionbarsherlock.com找到它。基于最新的Android源码开发，ActionBarSherlock以按需复制的方式提供了操作栏特色功能，且支持Android 3.0以前的所有系统版本。本章随后的深入学习以及第二个挑战练习部分，我们首先会学习如何使用ActionBarSherlock，然后再通过挑战练习进行实践。

(据Google资深人士透露，官方支持库也将推出复制版本的操作栏功能，希望这激动人心的一天早点到来。)

回退与复制，哪一种策略更好呢？显然复制策略更胜一筹。它的主要优势在于，无论是在哪

个系统版本上，复制功能都能保持统一的风格。这一点尤其适用于替换复制策略，使用该策略可以保证任何系统版本上都运行着同样的代码。另外，应用的设计和测试人员也会因此受益，因为他们只需设计一种用户界面风格以及测试验证一套应用交互。最后，使用复制策略，用户无需升级设备，就可使应用看上去像全新的ICS或Jelly Bean应用。复制是Android世界最流行的使用策略。这也很好地解释了，为什么样式以及新应用几乎总是复制最新库的特色功能以保持最新。

总结完了复制策略的优势，再来看看它的两个主要缺点。

- 为保持最新，必须依赖于第三方库。这也是本书采用优雅的回退策略处理操作栏的原因所在。
- 与设备上的其他应用相比，使用复制策略的应用看上去有点另类。当然，如果应用采用了定制设计，这就不是什么问题，因为无论怎么看，该应用总是会与众不同。而如果我们希望与手机上的标准应用保持一致的风格，那么由于使用了复制策略，应用看上去会非常突兀。

18.5 挑战练习：在 CrimeFragment 视图中删除 crime 记录

如果在列表项界面以及记录明细界面都能删除crime记录，应用的用户体验应该会更好。记录明细界面的删除操作是作用于整个屏幕的。因此，删除操作应该置于选项菜单中或操作栏上。在CrimeFragment视图中，实现一个删除crime记录的选项菜单。

18.6 深入学习：ActionBarSherlock

与理解Android标准库的基本工作原理及其如何在不同设备和操作系统上运作一样关键，开发者们早已掌握了设备兼容性问题的处理。当前，最大的兼容性问题是本章及第16章中学习到的操作栏。

ActionBarSherlock（简称为ABS）旨在解决这种兼容性问题。ABS提供了一个向后兼容的操作栏版本。此外，它还提供了一些支持类，可根据不同的版本系统，确定是使用原生还是向后兼容版本的操作栏。可访问网站http://www.actionbarsherlock.com找到它。值得一提的是，它还提供了Android最新主题的向后兼容版本（包含操作栏）。

是不是觉得ABS更像一个功能丰富的支持库？没错，事实的确如此。不过，与支持库不同，ABS提供有主题和资源ID。这意味着ABS是作为Android库项目而不是jar文件来分发的。库项目看起来就像一个创建独立应用的常规项目，但实际上它只是一个供其他应用使用的类库。这允许Android编译引入类库提供的任何Android附加资源。任何提供附加资源的类库都必须以库项目的形式存在。

既然ABS是一个库项目，要将它整合到项目中，需完成以下步骤：

- 下载并解压ABS源文件；
- 将源文件导入名为ActionBarSherlock的Eclipse项目；
- 在CriminalIntent应用中引用ActionBarSherlock库项目；
- 升级CriminalIntent应用，使用ActionBarSherlock支持类。

（如打算跟随向导学习如何整合ActionBarSherlock，应首先为后续章节的学习备份一份没有

引入ABS的CriminalIntent应用。）

要下载ABS，请登陆网站http://www.actionbarsherlock.com/download.html，点击下载链接，下载它的zip或tgz压缩包（二者皆可），下载完成后将其解压到本地。

接下来，我们来创建ABS库项目。在Eclipse中，右键单击选择包浏览器的New → Project....菜单项。

我们现在需创建的不是一个全新Android项目，而是一个包含了已下载源码的项目。因此在弹出的新建项目对话框中，选择Android Project From Existing Code，如图18-6所示。

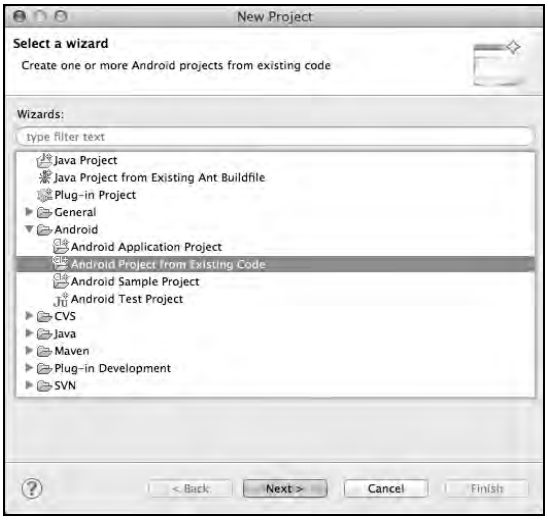

图18-6　创建包含现有源码的Android项目

单击Next按钮，在随后的对话框中，Eclispe要求我们选择源码存在的根目录。单击Browse...，浏览至已解压的ABS目录。解压文件中有三个子目录：library、samples和website。选择library目录，单击Open按钮，然后单击Finish按钮完成，如图18-7所示。

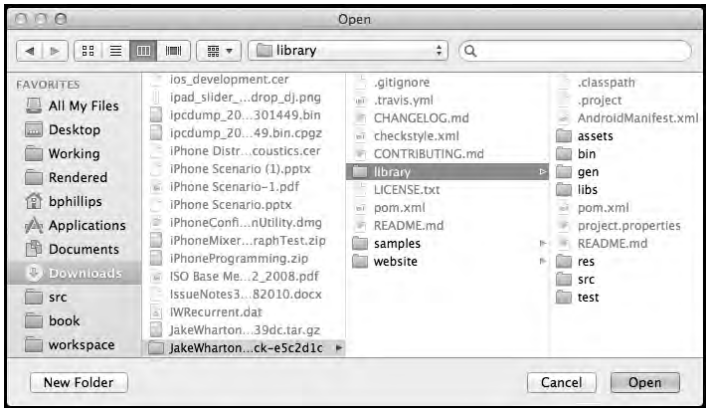

图18-7　选择ABS库文件目录

Eclipse完成新项目的创建后，项目名称显示为library。显然这不是一个有意义的项目名称，因此右键单击项目，选择Refactor → Rename...菜单项，重命名项目为ActionBarSherlock。

现在，我们来完成添加ABS引用的最后一步：添加CriminalIntent应用库项目的引用。在包浏览器中，右键单击CriminalIntent应用，选择Properties...菜单。在弹出的属性对话框中，单击左边的Android选项，然后查看屏幕底部，如图18-8所示。

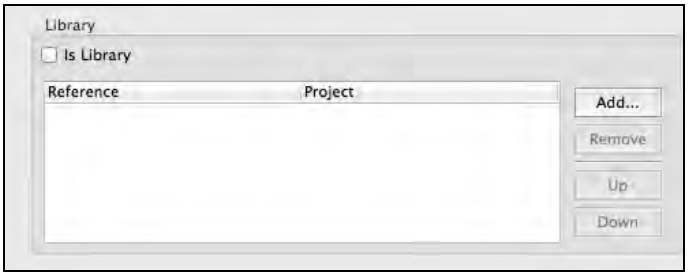

图18-8 Android库项目的引用

这里会列出Android所有库项目的引用。目前为止，我们还没有引用过任何库项目，因此此处显示为空。单击Add…按钮，如图18-9所示。

图18-9 添加ActionBarSherlock库项目的引用

最后，在库项目选择界面，双击ActionBarSherlock添加库项目引用。

18.7 挑战练习：使用 ActionBarSherlock

既然已添加完ActionBarSherlock库项目的引用，是时候将其整合到CriminalIntent应用中了。ABS的使用与Android支持库类似，提供了`Activity`、`Fragment`以及`ActionBar`等Android核心类的替换版本。为方便与系统支持库区分，大多数的ABS类名都是以`Sherlock-`为前缀的。

现在，如何使用ActionBarSherlock应该有眉目了。注意，虽然fragment和activity类都是以

Sherlock-为前缀的，但菜单相关类的命名并不遵循此规则。

18.7.1　CriminalIntent应用中ABS的基本整合

下面是最基本的ABS整合步骤。

- 将SingleFragmentActivity及CrimePagerActivity的父类从FragmentActivity调整为SherlockFragmentActivity。
- 将各fragment的父类从支持库版本的相关类调整为SherlockFragment、SherlockDialogFragment或者SherlockListFragment类。
- 将Menu、MenuItem及MenuInflater类引用调整为它们在com.actionbarsherlock.view中的对应实现类。

前两步相对简单，只需修改超类名即可。这之后，CrimeFragment及CrimeListFragment类中会出现很多错误。为消除这些错误信息，需完成相对复杂的第三个步骤。

CrimeFragment类的处理比较容易：删除菜单相关类的导入语句，然后使用组合键Command+Shift+O/Ctrl+Shift+O，组织导入com.actionbarsherlock.view中的对应版本类。

假设以同样的方式处理CrimeListFragment类则行不通。这是因为CrimeListFragment类使用了上下文菜单以及MultiChoiceModeListener类，而它们依然需要使用Android原生库版本的菜单类。

那么如何解决这个问题呢？与组织导入类包的处理方式不同，我们需在onCreateOptionsMenu(...)和onOptionsItemSelected(...)方法中使用全路径包名引用Menu、MenuItem及MenuInflater类。也就是说，不应使用以下类引用形式：

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    ...
}
```

而应使用以下类引用形式：

```
@Override
public void onCreateOptionsMenu(com.actionbarsherlock.view.Menu menu,
        com.actionbarsherlock.view.MenuInflater inflater) {
    ...
}
```

18.7.2　ABS的深度整合

如已完成第一个挑战练习，则ABS应该已经基本整合到应用中了。然而，要在真正意义上使用它，还需通过更进一步整合来删除兼容代码。这可以通过使用getSherlockActivity().getSupportActionBar()方法替代getActivity().getActionBar()原生方法来完成。SherlockActivity的操作栏适用于所有的系统版本，因此改调该方法后，可直接删除部分兼容性相关的代码。完成后，也可将res/menu-v11/目录下的fragment_crime_list.xml文件移至res/menu目录中，以避免配置级别的操作系统版本切换。

18.7.3 ABS的完全整合

ABS整合的逐步深入是不是很刺激？接下来，为保证CriminalIntent应用在不同系统版本上都能有一致的表现，还需弃用原生版本的MultiChoiceModeListener和上下文菜单类。删除上下文菜单的相关代码很简单，但要替换MultiChoiceModeListener类，首先必须能复制其功能。

如何做到这一点呢？首先是使用旧式的`ListView.CHOICE_MODE_MULTIPLE`选择模式。本章使用的是仅适用于新系统的`ListView.CHOICE_MODE_MULTIPLE_MODAL`。虽然`ListView.CHOICE_MODE_MULTIPLE`不支持视图长按行为，但至少可兼容所有版本的Android系统。要支持多选操作，可设置ListView的选择模式为`ListView.CHOICE_MODE_MULTIPLE`。要禁止选择操作，则将ListView的选择模式重新设置为`ListView.CHOICE_MODE_NONE`。

然后，我们需要复制`CHOICE_MODE_MULTIPLE_MODAL`提供的操作栏行为。要实现兼容各系统版本的复制版本操作栏行为，可调用`getSherlockActivity().startActionMode()`方法。需要提醒的是，请确保使用的是`com.actionbarsherlock.view.ActionMode.Callback`中的方法，而不是通常的Android版本方法。

最后，需和以前一样监听长按动作。要监听长按动作，可在`ListView.setOnItemLongClickListener(...)`监听方法中实现`OnItemLongClickListener`接口。