第 3 章 用 户 界 面

3.1 用户界面基础知识

对于一个应用程序来说，首先呈献给用户的肯定是用户界面（User Interface，UI），所以用户界面对一个应用程序来说是非常重要的。

程序的用户界面是指用户看到的并与之交互的一切，Android 提供了一个强大的模式来定义用户界面，这个模式基于基础的布局类：视图（View）和视图组（ViewGroup）。Android 提供了多种预先生成的视图和视图组的子类，用于构建你自己的用户界面。其实 ViewGroup 也是继承了 View 的，所以 Android 预先生成的这些组件都是 View 类的子类。

用户界面的设计可分为两种方式：一种设计方式是单一的采用代码的方式进行用户界面的设计，例如编写一个 Java 的 Swing 应用，你需要用 Java 代码去创建和操纵界面 JFrame 和 JButton 等对象；另一种设计方式是像设计网页时采用类似 XML 的 HTML 标记语言去描述你想看到的页面效果。Android 采取这两种方法均可，让你可以选择任意一种方式进行界面设计，既可以使用 Java 代码也可以采用 XML 声明你想要的界面效果。如果查看 Android 的用户界面 API 文档，你会同时看到 Java 的方法和对应的 XML 属性（如图 3-1 所示）。

XML Attributes

Attribute Name	*Related Method*	*Description*
android:cacheColorHint		Indicates that this list will always be drawn on top of solid, single-color opaque background.
android:choiceMode		Defines the choice behavior for the view.
android:drawSelectorOnTop	setDrawSelectorOnTop (boolean)	When set to true, the selector will be drawn over the selected item.
android:fastScrollEnabled		Enables the fast scroll thumb that can be dragged to quickly scroll through the list.
android:listSelector	setSelector(int)	Drawable used to indicate the currently selected item in the list.
android:scrollingCache		When set to true, the list uses a drawing cache during scrolling.
android:smoothScrollbar	setSmoothScrollbarEnabled (boolean)	When set to true, the list will use a more refined calculation method based on the pixels height of the items visible on screen.
android:stackFromBottom		Used by ListView and GridView to stack their content from the bottom.
android:textFilterEnabled		When set to true, the list will filter results as the user types.
android:transcriptMode		Sets the transcript mode for the list.

图 3-1　Java 方法与对应的 XML 属性表

那到底使用哪一种好点呢？其实两者都可以，但 Google 建议尽量使用 XML，因为相比 Java 代码而言，XML 语句更简短易懂，在以后的版本中不易被改变。本书的所有程序也都基本采用 XML 进行界面设计。

3.2 界面基本组件

在本章开头提到 Android 提供了很多预先生成的组件，下面大致介绍一下这些组件。我们知道，其实每个组件都是视图类（View）的子类，所以每个组件都是一个视图，继承了多种 View 的属性。

3.2.1 界面基本属性

在界面设计时，我们需要对不同的组件设置不同的属性，但上面讲到其实每个组件都是 View 的子类，那么它们肯定会有一些相同的基本属性。下面就列举一些常用的属性。

- android:layout_width：设置组件的宽度。
- android:layout_height：设置组件高度。
- android:background：设置组件的背景，在代码中可以使用 setBackgroundResource() 来设置该属性。
- android:onClick：为组件添加点击事件响应函数，有关事件响应的知识将在 3.5 节中具体讲解。
- android:id：设置组件的 id，同样在代码中使用 setId()方法可达到同样的效果。

界面的基本属性不止这些，而且不同组件拥有自己的特殊属性，比如 LinearLayout 的 android:orientation 属性设置布局的方位是水平还是垂直，这些都将在接下来的组件和布局的讲解中涉及。

3.2.2 TextView

TextView 是标准的只读标签。它支持多行显示，支持字符串格式化和自动换行。对于 TextView 我们最关心的就是怎样设置显示的文本，怎样设置字体的大小、颜色和样式，TextView 提供的大量属性可帮我们轻松地完成这些，图 3-2 就是利用这些属性完成的一个 TextView。

图 3-2 的 XML 布局如下：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:orientation="vertical"
4.      android:layout_width="fill_parent"
5.      android:layout_height="fill_parent"
6.      >
7.  <TextView
```

```
8.      android:layout_width="fill_parent"
9.      android:layout_height="wrap_content"
10.     android:textColor="#fff000"
11.     android:textSize="20dp"
12.     android:textStyle="bold"
13.     android:text="我是文本框"
14.     />
15. </LinearLayout>
```

Activity 的代码如下：

```
1.  public class TextView extends Activity{
2.          public void onCreate(Bundle savedInstanceState) {
3.           super.onCreate(savedInstanceState);
4.          setContentView(R.layout.textview);
5.       }}
```

这里增加了 3 个属性的设置，分别是 android:textColor="#fff000"设置字体为黄色，android:textSize="20dp"设置字体为 20dp，android:textStyle="bold"设置字体加粗。

本部分代码见本书配套资源中的工程 Chapter3.2.1。

我们都见过 HTMl 中只要加个<a/>标记就可以将一段文字变成超链接的形式，可以单击访问链接地址。TextView 中也有超链接形式。TextView 提供了 android:autoLink 属性，只要把它设置成 web，该 TextView 中的网址形式的文件就会自动变成超链接的形式，如图 3-3 所示。

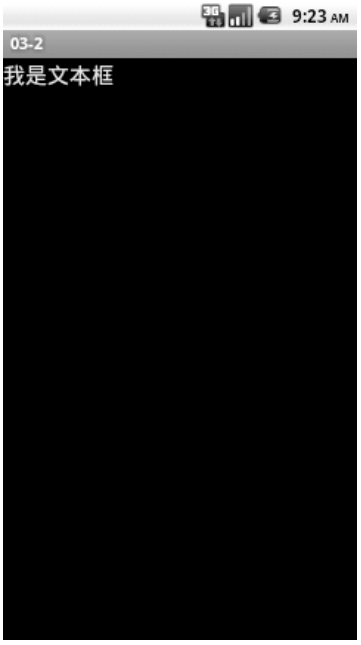

图 3-2 TextView

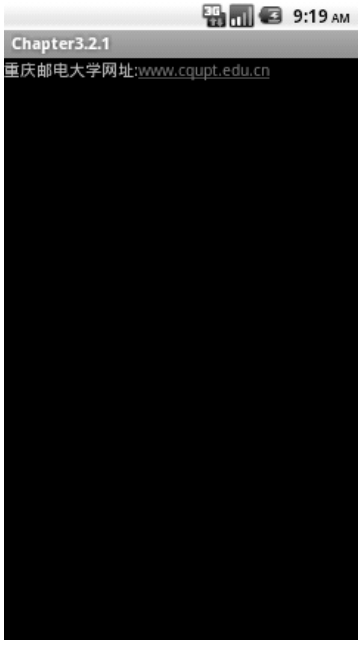

图 3-3 网址超链接

图 3-3 的 XML 布局如下：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="fill_parent"
4.      android:layout_height="fill_parent"
5.      android:orientation="vertical" >
6.      <TextView
7.          android:id="@+id/text_view"
8.          android:layout_width="fill_parent"
9.          android:layout_height="wrap_content"
10.         android:autoLink="web"
11.         android:text="重庆邮电大学网址:www.cqupt.edu.cn " />
12. </LinearLayout>
```

Activity 的代码如下：

```
1.  public class TextView extends Activity{
2.           public void onCreate(Bundle savedInstanceState) {
3.            super.onCreate(savedInstanceState);
4.           setContentView(R.layout.textview);
5.       }}
```

这里是将网址以超链接显示，如果要将电话号码显示为超链接，那么只需将 android:autoLink 属性设置为 phone，E-mail 也是同理。那能不能将网址、电话、E-mail 都设为超链接形式呢？当然可以，只需将 android:autoLink 属性设为 all，这样里面的网址、电话和 E-mail 就都显示为超链接。

当然我们经常会在代码中修改这些属性，那时只需要调用这些属性对应的方法即可，如 android:textColor 对应的 setTextColor(int)方法、android:autoLink 对应的 setAutoLinkMask(int)方法等，这里就不一一列举了，读者可查看 TextView 的 API 了解其他属性设置。

本部分代码见本书配套资源中的工程 Chapter3.2.2。

3.2.3 EditText

EditText 是 TextView 的子类，它与 TextView 一样具有支持多行显示、字符串格式化和自动换行的功能，它的使用和 TextView 并无太大区别。但在实际编程中经常要求编辑框输入一些特定的内容，例如 0～9 的数字、E-mail 等，如图 3-4 所示。接下来要实现的就是这个实例。

图 3-4　不同输入类型的 EditText

图 3-4 的 XML 布局如下：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="fill_parent"
4.      android:layout_height="fill_parent"
5.      android:orientation="vertical" />
6.      <TextView
7.          android:layout_width="fill_parent"
8.          android:layout_height="wrap_content"
9.          android:text="使用 android:inputType 属性，输入 E-mail" />
10.     <EditText
11.         android:layout_width="fill_parent"
12.         android:layout_height="wrap_content"
13.         android:inputType="textE-mailAddress" />
14.     <TextView
15.         android:layout_width="fill_parent"
16.         android:layout_height="wrap_content"
17.         android:text="使用 android:digits 属性，输入 26 个小写字母" />
18.     <EditText
19.         android:layout_width="fill_parent"
20.         android:layout_height="wrap_content"
21.         android:digits="abcdzfghijklmnopqrstuvwxyz" />
22.     <TextView
23.         android:layout_width="fill_parent"
```

```
24.        android:layout_height="wrap_content"
25.        android:text="使用 android:numeric 属性，输入 0~9 数字" />
26.    <EditText
27.        android:layout_width="fill_parent"
28.        android:layout_height="wrap_content"
29.        android:numeric="integer" />
30. </LinearLayout>
```

Activity 的代码如下：

```
1.  public class TextView extends Activity{
2.          public void onCreate(Bundle savedInstanceState) {
3.           super.onCreate(savedInstanceState);
4.          setContentView(R.layout.textview);
5.      }}
```

上面这段代码中使用了 EditText 的 3 个属性指定了 3 种不同的输入字符，分别是：

- 将 android:inputType 的属性值设置为 textEmailAddress，指定输入为 E-mail。但要注意的是用于输入 E-mail 的 EditText 逐渐不会限制输入非 E-mail 字符，只是在虚拟键盘上多了一个“@”键。
- 将 android:digits 属性值设为 26 个小写英文字母，将输入内容限制在 26 个小写字母内。
- 将 android:numeric 属性值设为 integer，设置其内容为整数。

关于 3 个标签的其他属性可以查阅官方的参考文档。本节代码见本书配套资源中的工程 Chapter3.2.3。

3.2.4 Button

Button 是最常见的界面组件，在 Android 中也不例外。Android 除了提供一些典型的按钮外，还提供了一些额外的按钮。本节将讲解 3 种不同的按钮：普通按钮（Button）、图片按钮（ImageButton）、开关按钮（ToggleButton）。图 3-5 展示了这 3 种按钮的效果图。最上面的是普通按钮，中间的是图片按钮，最下面的是开关按钮，当前处于关闭状态。

1．普通按钮（Button）

关于普通按钮，除了掌握单击事件（click event）外没有太多的知识点。

首先使用下面的代码定义一个普通的 Button。

```
1.  <Button
2.     android:id="@+id/button"
3.     android:layout_width="wrap_content"
4.     android:layout_height="wrap_content"
5.     android:text="普通按钮"/>
```

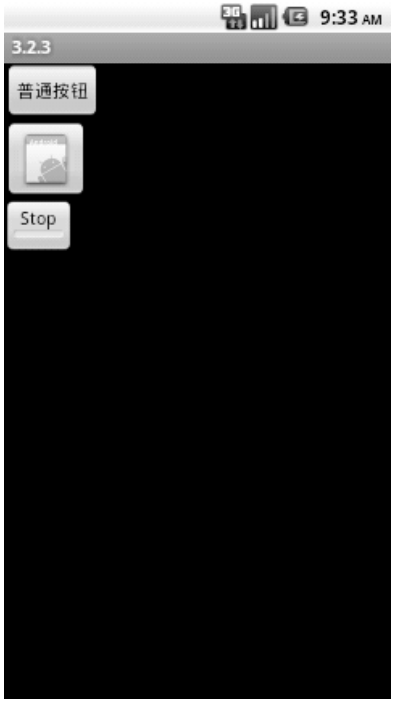

图 3-5　3 种按钮

接下来就是为该 Button 添加按钮单击事件，只需要调用 Button 的 setOnClickListener 函数，OnClickListener 接口的子类为参数，在 onClick 中做出事件的响应。有关监听事件的讲解将在 3.5 节进行。事件监听代码如下：

```
1.  Button btn = (Button) this.findViewById(R.id.button);
2.          btn.setOnClickListener(new OnClickListener() {
3.              public void onClick(View v) {
4.                  Toast.makeText(MainActivity.this, "button",
                                    Toast.LENGTH_SHORT).show();
5.              }
6.          });
```

2．图片按钮（ImageButton）

Android 在 android.widget 包定义了一个 ImageButton。ImageButton 的用法除了要为按钮添加图片资源外其他用法与普通按钮的用法没有太大差别，ImageButton 的定义如下：

```
1.  <ImageButton
2.      android:id="@+id/imageButton1"
3.      android:layout_width="wrap_content"
4.      android:layout_height="wrap_content"
5.      android:src="@drawable/ic_launcher" />
```

与 Button 的定义比较，ImageButton 的定义只是多了个 android:src 属性为 ImageButton 添加图片资源，此外还可以在 Java 代码中调用 setImageURI (Uri uri)方法添加图片资源。

关于 ImageButton 的其他属性读者可以参考官方文档。

3．开关按钮（ToggleButton）

ToggleButton 是 Android 定义的一种特殊按钮，它有两种状态：开（On）、关（Off）。ToggleButton 的默认状态是在开启状态时显示一条绿色的进度条，关闭时显示一条灰色的滚动条。下面的代码展示了一个 ToggleButton 的定义。

```
1.  <ToggleButton
2.          android:id="@+id/toggleButton"
3.          android:layout_width="wrap_content"
4.          android:layout_height="wrap_content"
5.          android:text="开关按钮"
6.          android:textOff="Stop"
7.          android:textOn="Run" />
```

这里的 android:textOff 和 android:textOn 属性设置了默认状态下开启和关闭状态时显示的文字。ToggleButton 在其他方面的应用与 Button 也没有太大的区别，其双状态功能的特性在开关等功能时较有用处。

本节代码见本书配套资源中的工程 Chapter3.2.4。

3.2.5 复选框（CheckBox）

复选框（CheckBox）在所有小部件工具包都有涉及。HTML、图形用户界面和 JSF 都支持 CheckBox 这个概念。同开关按钮一样，CheckBox 是一种拥有两种状态的按钮，它允许用户在两种状态间自由切换。下面就来在 Android 中创建一个 CheckBox，如图 3-6 所示。

图 3-6 CheckBox

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="语文" />
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="数学" />
    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="外语" />
</LinearLayout>
```

在编程中可以通过 setChecked()或 toggle()来管理一个 CheckBox，通过 isChecked()来获得一个复选框的状态。

如果当一个复选框被选中或取消选中时需要实现特定的逻辑，可以通过 setOnCheckedChangeListener()为复选框注册一个实现了 OnCheckedChangeListener 的类，并实现 OnCheckedChangeListener 的 onCheckedChanged()方法，这个方法会在复选框的状态改变时调用。有关 CheckBox 的知识读者可参考官方文档 android.widget.CheckBox。

本节代码见本书配套资源中的工程 Chapter3.2.5。

3.2.6 单选按钮

单选按钮（RadioButton）控件是任何 UI 工具包的组成部分，它为用户提供了单击选中并要求必须选中一个项目的功能。为了完成这个单选功能，每个单选按钮须归属于一个组，每个组在每段时间只能有一个选项被选中，如图 3-7 所示。

RadioButton 的定义如下：

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

    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <RadioGroup
        android:id="@+id/rBtnGrp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical" >
        <RadioButton
```

```
12.            android:id="@+id/RBtn1"
13.            android:layout_width="wrap_content"
14.            android:layout_height="wrap_content"
15.            android:checked="true"
16.            android:text="语文" />
17.        <RadioButton
18.            android:id="@+id/RBtn2"
19.            android:layout_width="wrap_content"
20.            android:layout_height="wrap_content"
21.            android:text="数学" />
22.        <RadioButton
23.            android:id="@+id/RBtn3"
24.            android:layout_width="wrap_content"
25.            android:layout_height="wrap_content"
26.            android:text="英语" />
27.    </RadioGroup>
28. </LinearLayout>
```

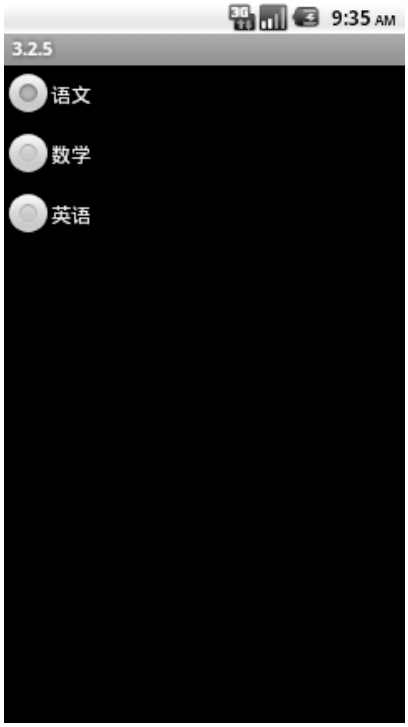

图 3-7　RadioButton

3 个单选按钮都放在一个单选组里面，并将第一个单选按钮的 android:checked 属性设为 true，这样第一个按钮就默认被选中，android:checked 在默认情况下是 false。RadioButton 的管理和事件监听处理和与 CheckBox 是一致的，此处不再赘述。

本节代码见本书配套资源中的工程 Chapter3.2.6。

3.2.7 Listview

最后介绍一个在 Android 开发中非常重要的一个组件——Listview。ListView 是一个 ViewGroup，用于创建一个滚动的项目清单，通过使用 ListAdapter，列表中的项目会自动插入到列表中。本节将创建一个简单的滚动图书列表，当单击了一项后就会弹出一个 Toast（即一闪而过的简单提示框。如图 3-8 屏幕下方“美术”处的提示框）提示单击的图书。具体效果如图 3-8 所示。

图 3-8 图书列表

具体创建步骤如下。

（1）创建工程 BookListView。

（2）创建一个 XML 布局文件 list_item.xml，该文件定义了每个将被放置在 ListView 中的项目的布局。list_item.xml 的内容如下：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <TextView xmlns:android=http://schemas.android.com/apk/res/android
3.      android:layout_width="fill_parent"
4.      android:layout_height="fill_parent"
5.      android:padding="15dp"
6.      android:textSize="20px" >
7.  </TextView>
```

（3）打开 BookListViewActivity.Java，让 BookListViewActivity 类继承 ListActivity。BookListViewActivity 的代码如下：

```
1.  public class BookListViewActivity extends ListActivity {
2.      public void onCreate(Bundle savedInstanceState) {
3.          super.onCreate(savedInstanceState);
4.          setListAdapter(new ArrayAdapter<String>(this, R.layout.list_item,
5.                  BOOKS));
6.          ListView lv = getListView();
7.          lv.setTextFilterEnabled(true);
8.          lv.setOnItemClickListener(new OnItemClickListener() {
9.              public void onItemClick(AdapterView<?> parent, View view,
10.                     int position, long id) {
11.                 Toast.makeText(getApplicationContext(),
12.                         ((TextView)  view).getText(),Toast.LENGTH_
                                SHORT).show();
13.             }
14.         });
15.     }
16.     private static final String[] BOOKS = new String[] {"数学",
        "英语","物理","化学","生物","体育","健康教育","美术","程序设计",
        "Android 程序教程"};
17. }
```

这样简单的图书列表就完成了，但只是简单地使用 Listview 显示了一行文字，在实际开发中每个项目的布局要比这复杂得多，这就需要修改 list_item.xml 的布局并使用合适的 ListAdapter。

本节代码见本书配套资源中的工程 Chapter3.2.7。

3.3 布　　局

Android 拥有许多的组件，而 TextView、EditText 和 Button 就是其中比较常用的几个组件。那么 Android 又是怎样将组件简洁而又美观地分布在界面上的呢？这里就用到了 Android 的布局管理器。这些独立的组件通过 Android 布局组合到一起，就可以给用户提供复杂而有序的界面。

3.3.1 FrameLayout（帧布局）

FrameLayout 是从屏幕的左上角的（0,0）坐标开始布局，多个组件层叠排列，第一个添加的组件放到最底层，最后添加到框架中的视图显示在最上面。上一层的会覆盖下一层的控件。

FrameLayout 是最简单的一个布局对象。它被定制为屏幕上的一个空白备用区域，

之后你可以在其中填充一个单一对象。例如，一张要发布的图片。所有的子元素将会固定在屏幕的左上角；你不能为 FrameLayout 中的一个子元素指定一个位置。后一个子元素将会直接在前一个子元素之上进行覆盖填充，把它们部分或全部挡住（除非后一个子元素是透明的）。图 3-9 就是一个 FrameLayout 的布局效果。

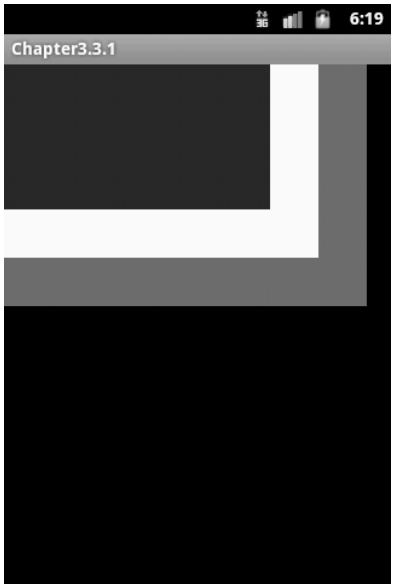

图 3-9　FrameLayout（帧布局）

图 3-9 对应的代码如下：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.          android:layout_width="fill_parent"
4.          android:layout_height="fill_parent" >
5.          <TextView
6.              android:layout_width="300dp"
7.              android:layout_height="200dp"
8.              android:background="#F11EEE" />
9.          <TextView
10.             android:layout_width="260dp"
11.             android:layout_height="160dp"
12.             android:background="#FFFFFF" />
13.         <TextView
14.             android:layout_width="220dp"
15.             android:layout_height="120dp"
16.             android:background="#000FFF" />
17.     </FrameLayout>
```

完整代码见本书配套资源中的工程 Chapter3.3.1。

3.3.2 LinearLayout（线性布局）

LinearLayout 以为它设置的垂直或水平的属性值来排列所有的子元素。所有的子元素都被堆放在其他元素之后，因此一个垂直列表的每一行只会有一个元素，而不管它们有多宽，一个水平列表将会只有一个行高（高度为最高子元素的高度加上边框高度）。LinearLayout 保持子元素之间的间隔并互相对齐（相对一个元素的右对齐、中间对齐或者左对齐）。

LinearLayout 还支持为单独的子元素指定 weight。好处就是允许子元素可以填充屏幕上的剩余空间。这也避免了在一个大屏幕中，一串小对象挤成一堆的情况，而是允许它们放大填充空白。子元素指定一个 weight 值，剩余的空间就会按这些子元素指定的 weight 比例分配。默认的 weight 值为 0。例如，如果有 3 个文本框，其中两个指定了 weight 值为 1，那么，这两个文本框将等比例地放大，并填满剩余的空间，而第三个文本框不会放大。

线性布局是 Android 开发中最常见的一种布局方式，它是按照垂直或者水平方向来布局的，通过 android:orientation 属性可以设置线性布局的方向。属性值有垂直（vertical）和水平（horizontal）两种。

常用的属性如下：

- android:orientation——可以设置布局的方向。
- android:gravity——用来控制组件的对齐方式。
- layout_weight——控制各个组件在布局中的相对大小。

如图 3-10 就是一个 LinearLayout 的布局效果。

图 3-10　LinearLayout（线性布局）

图 3-10 对应的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        >
   <TextView
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:text="请输入用户名："
       android:textSize="10pt" />
   <EditText
       android:layout_width="fill_parent"
       android:layout_height="wrap_content" />
   <TextView
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:text="请输入密码："
       android:textSize="10pt" />
   <EditText
       android:layout_width="fill_parent"
       android:layout_height="wrap_content" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="right"
        >
    <!-- android:gravity="right"表示 Button 组件向右对齐 -->
        <Button
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:text="确定"
            />
        <Button
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
```

```
43.            android:text="取消"
44.            />
45.      </LinearLayout>
46. </LinearLayout>
```

完整代码见本书配套资源中的工程 Chapter3.3.2。

3.3.3 RelativeLayout（相对布局）

RelativeLayout 允许子元素指定它们相对于其他元素或父元素的位置（通过 ID 指定）。因此，可以以右对齐、或上下、或置于屏幕中央的形式来排列两个元素。元素按顺序排列，因此如果第一个元素在屏幕的中央，那么相对于这个元素的其他元素将以屏幕中央的相对位置来排列。如果使用 XML 来指定这个 layout，那么在定义它之前，必须先定义被关联的元素。

图 3-11 就是一个 RelativeLayout 的效果。

图 3-11　RelativeLayout（相对布局）

图 3-11 对应的代码如下：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      android:layout_width="fill_parent"
4.      android:layout_height="fill_parent"
5.      android:orientation="vertical" >
6.      <EditText
7.          android:id="@+id/user"
8.          android:layout_width="220dip"
```

```
9.          android:layout_height="40dip"
10.         android:layout_centerHorizontal="true"
11.         android:layout_centerVertical="true"
12.         android:inputType="text"
13.         android:textColor="#000000" />
14.     <EditText
15.         android:id="@+id/password"
16.         android:layout_width="220dip"
17.         android:layout_height="40dip"
18.         android:layout_alignLeft="@+id/user"
19.         android:layout_below="@+id/user"
20.         android:layout_marginTop="16dp"
21.         android:inputType="textPassword"
22.         android:textColor="#000000" />
23.     <Button
24.         android:id="@+id/button_login"
25.         android:layout_width="220dip"
26.         android:layout_height="40dip"
27.         android:layout_alignLeft="@+id/password"
28.         android:layout_below="@+id/password"
29.         android:layout_marginTop="14dp"
30.         android:text="登录"
31.         android:textColor="#000000" />
32.     <TextView
33.         android:id="@+id/textView_user"
34.         android:layout_width="fill_parent"
35.         android:layout_height="wrap_content"
36.         android:layout_alignBaseline="@+id/password"
37.         android:layout_alignBottom="@+id/password"
38.         android:layout_alignParentLeft="true"
39.         android:text="账号"
40.         android:textColor="#FFFFFF"
41.         android:textSize="20dp" />
42.     <TextView
43.         android:id="@+id/textView_password"
44.         android:layout_width="fill_parent"
45.         android:layout_height="wrap_content"
46.         android:layout_alignBaseline="@+id/user"
47.         android:layout_alignBottom="@+id/user"
48.         android:layout_alignParentLeft="true"
49.         android:text="密码"
50.         android:textColor="#FFFFFF"
51.         android:textSize="20dp" />
52.     <ImageView
53.         android:id="@+id/imageView_login"
```

```
54.        android:layout_width="wrap_content"
55.        android:layout_height="wrap_content"
56.        android:layout_above="@+id/user"
57.        android:layout_centerHorizontal="true"
58.        android:layout_marginBottom="22dp"
59.        android:src="@drawable/yu" />
60. </RelativeLayout>
```

完整代码见本书配套资源中的工程 Chapter3.3.3。

3.3.4 TableLayout（表格布局）

TableLayout 将子元素的位置分配到行或列中。一个 TableLayout 由许多的 TableRow 组成，每个 TableRow 都会定义一个 row（事实上，你可以定义其他的子对象）。TableLayout 容器不会显示 rowcloumns 或 cell 的边框线。每个 row 拥有 0 个或多个的 cell；每个 cell 拥有一个 View 对象。表格由列和行组成许多单元格。表格允许单元格为空。单元格不能跨列，这与 HTML 中的不一样。

如图 3-12 所示为 TableLayout 布局效果。

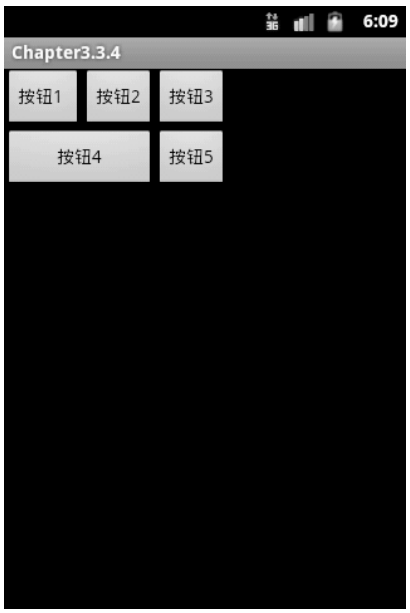

图 3-12　TableLayout（表格布局）

图 3-12 对应的代码如下：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.   <TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.       android:layout_width="fill_parent"
4.       android:layout_height="fill_parent" >
5.       <TableRow>
6.           <Button
```

```
7.              android:text="按钮1 " />
8.          <Button
9.              android:text="按钮2" />
10.         <Button
11.             android:text="按钮3" />
12.     </TableRow>
13.     <TableRow>
14.         <Button
15.             android:layout_span="2"
16.             android:text="按钮4"/>
17.          <Button
18.             android:text="按钮5" />
19.     </TableRow>
20. </TableLayout>
```

完整代码见本书配套资源中的工程 Chapter3.3.4。

3.3.5 AbsoluteLayout（绝对布局）

AbsoluteLayout 可以让子元素指定准确的 x/y 坐标值，并显示在屏幕上。(0, 0)为左上角，当向下或向右移动时，坐标值将变大。AbsoluteLayout 没有页边框，允许元素之间互相重叠（尽管不推荐）。手机应用需要适应不同的屏幕大小，而这种布局模型不能自适应屏幕尺寸大小，所以通常不推荐使用 AbsoluteLayout，除非你有正当理由要使用它，因为它使界面代码太过刚性，以至于在不同的设备上可能不能很好地工作。

图 3-13 为 AbsoluteLayout 布局效果。

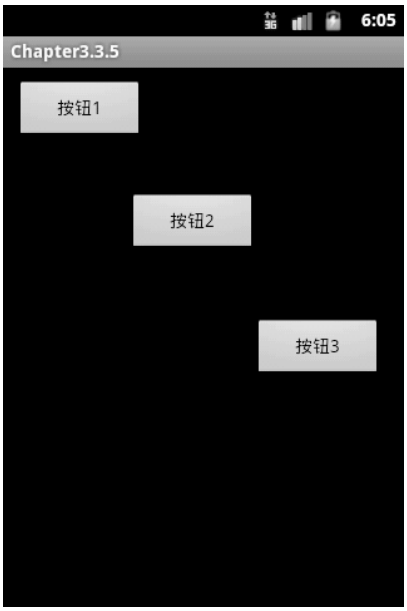

图 3-13 AbsoluteLayout（绝对布局）

图 3-13 对应的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
  <AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <Button
      android:layout_width="102dp"
      android:layout_height="wrap_content"
      android:layout_x="10dp"
      android:layout_y="10dp"
      android:text="按钮 1" />
   <Button
      android:layout_width="102dp"
      android:layout_height="wrap_content"
      android:layout_x="100dp"
      android:layout_y="100dp"
      android:text="按钮 2" />
    <Button
      android:layout_width="102dp"
      android:layout_height="wrap_content"
      android:layout_x="200dp"
      android:layout_y="200dp"
      android:text="按钮 3" />
</AbsoluteLayout>
```

完整代码见本书配套资源中的工程 Chapter3.3.5。

3.3.6 多种布局混合使用

在 Android 开发中，如果发现一个界面布局文件只使用一种布局不能达到理想的效果，则可混合使用两种或两种以上的布局。图 3-14 就是 Android 的混合布局。

图 3-14 对应的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
   <Button
      android:layout_width="102dp"
      android:layout_height="wrap_content"
      android:text="按钮 1" />
   <Button
      android:layout_width="102dp"
      android:layout_height="wrap_content"
```

```
13.        android:text="按钮 2" />
14.    <Button
15.        android:layout_width="102dp"
16.        android:layout_height="wrap_content"
17.        android:text="按钮 3" />
18.    <RelativeLayout
19.        android:layout_width="fill_parent"
20.        android:layout_height="fill_parent"
21.        android:background="#FFFFFF"
22.        android:orientation="horizontal" >
23.        <ImageView
24.            android:id="@+id/imageView_item_book"
25.            android:layout_width="200dip"
26.            android:layout_height="160dip"
27.            android:src="@drawable/ic_book" />
28.        <TextView
29.            android:id="@+id/information_item"
30.            android:layout_width="220dip"
31.            android:layout_height="wrap_content"
32.            android:layout_alignParentRight="true"
33.            android:layout_below="@+id/imageView_item_book"
34.            android:text="文本框"
35.            android:textColor="#000000"
36.            android:textSize="50dp" />
37.    </RelativeLayout>
38. </LinearLayout>
```

完整代码见本书配套资源中的工程 Chapter3.3.6。

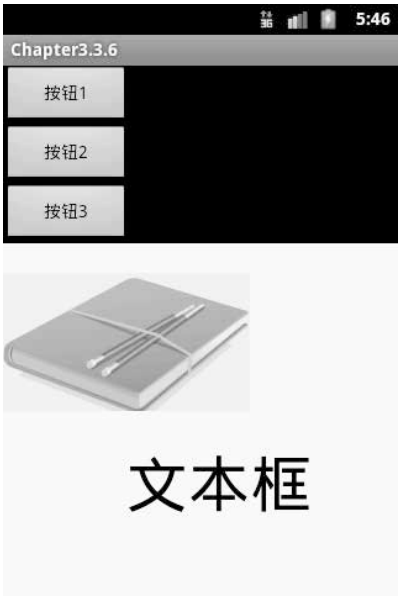

图 3-14　混合布局

3.4 菜　　单

Android 提供了一些简单的方法来为应用添加 Menu 菜单。大致分为 3 种类型：选项菜单、上下文菜单和子菜单。

3.4.1 选项菜单

选项菜单（Options Menu）是最常规的菜单，它是通过 Menu 按钮调用菜单。

如图 3-15 所示，按下手机上的 Menu 按钮（ 

）就会出现图中的菜单。

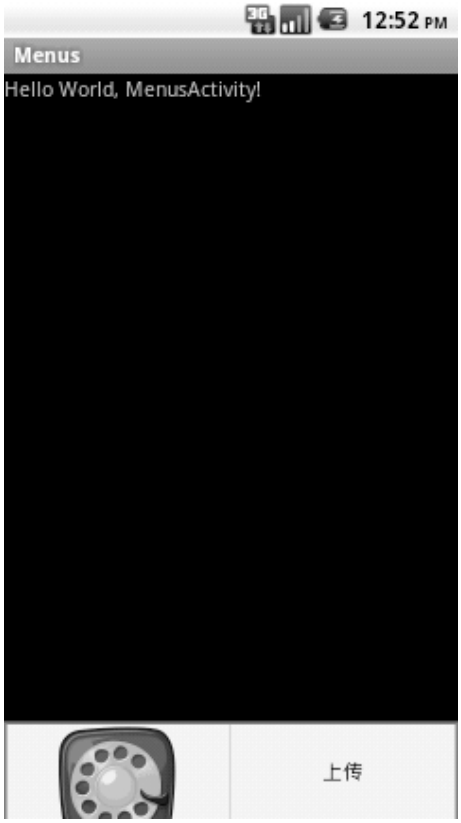

图 3-15　选项菜单

选项菜单最多只有 6 个，超过 6 个时第六个就会自动显示“更多”选项来展开显示。

具体用法如下：

- 通过 onCreateOptionsMenu()方法只会调用一次，即第一次显示的时候会调用。
- 如果需要更新菜单项，可以在 onPrepareOptionsMenu()方法中操作。
- 当菜单被选择的时候，在 OnOptionsItemSelected()方法中实现方法响应事件。

- 还可以调用 Menu 的 add()方法添加菜单项（MenuItem），可以调用 MenuItem 的 setIcon()方法为菜单项设置图标。

图 3-15 对应的代码实现如下：

```
1.  public class MenusActivity extends Activity {
2.     private static final int item1 = Menu.FIRST;
3.     private static final int item2 = Menu.FIRST + 1;
4.     public void onCreate(Bundle savedInstanceState) {
5.        super.onCreate(savedInstanceState);
6.        setContentView(R.layout.main);
7.     }
8.      public boolean onCreateOptionsMenu(Menu menu) {
9.         menu.add(0, item1, 0, "").setIcon(R.drawable.pic);
10.        menu.add(0, item2, 0, "上传");
11.       return true;
12.    }
13.    public boolean onOptionsItemSelected(MenuItem item) {
14.       switch (item.getItemId()) {
15.       case item1:
16.         setTitle("单击了菜单子项 1");
17.         break;
18.       case item2:
19.         setTitle("单击了菜单子项 2");
20.         break;
21.        }
22.     return true;
23.    }
24. }
```

详细代码见本书配套资源中的工程 Chapter3.4.1。

3.4.2 上下文菜单

上下文菜单（Context Menu）和 Windows 中的右键快捷菜单差不多。它一般是通过长按屏幕调用注册的上下文菜单。

如图 3-16 所示，程序运行后长按屏幕就会出现图中菜单。

实现上下文菜单要做的有以下几点：

- 覆盖 Activity 的 onCreateContextMenu()方法，调用 Menu 的 add()方法可以添加菜单项 MenuItem。
- 覆盖 onContextItemSelected()方法，响应菜单单击事件。
- 调用 registerForContextMenu()方法，为视图注册上下文菜单。

图 3-16　上下文菜单

图 3-16 对应的代码实现如下：

```
1.  public class MenusActivity extends Activity {
2.     private LinearLayout bc;
3.     public void onCreate(Bundle savedInstanceState) {
4.        super.onCreate(savedInstanceState);
5.        setContentView(R.layout.main);
6.        bc=(LinearLayout)findViewById(R.id.lay);
7.        registerForContextMenu(bc);
8.     }
9.     public void onCreateContextMenu(ContextMenu menu,View v,ContextMenuInfo
                                          menuInfo)
10.    {
11.        setTitle("菜单");
12.        menu.add(0,2,0,"菜单 1");
13.        menu.add(0,3,0,"菜单 2");
14.        super.onCreateContextMenu(menu, v, menuInfo);
15.    }
16.    public boolean onContextItemSelected(MenuItem item)
17.    {
18.        switch(item.getItemId()){
```

```
19.        case 2:
20.           Toast.makeText(this, "1", Toast.LENGTH_LONG).show();
21.           break;
22.        case 3:
23.            Toast.makeText(this, "2", Toast.LENGTH_LONG).show();
24.               break;
25.        }
26.       return true;
27.    }
28. }
```

详细代码见本书配套资源中的工程 Chapter3.4.2。

3.4.3 子菜单

子菜单（Submenu）就是将相同功能的分组进行多级显示的一种菜单，比如 Windows 的“文件”菜单中就有“新建”“打开”“关闭”等子菜单。

如图 3-17 所示，单击图中的任意一个菜单按钮就会出现图 3-18 中的子菜单。

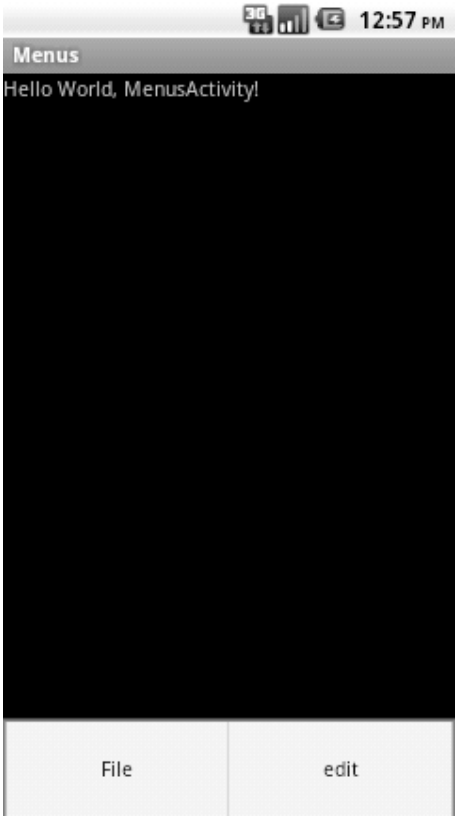

图 3-17 主菜单

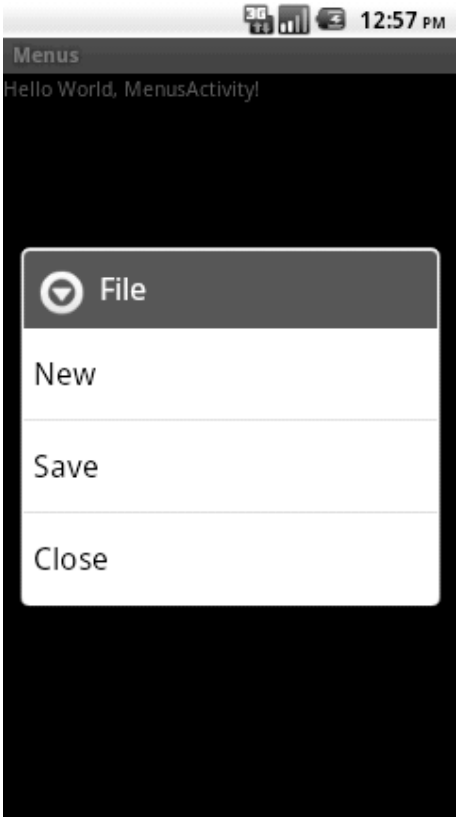

图 3-18 子菜单

有关子菜单需要注意以下几个方面：

- 通过触摸 Menu Item，调用子菜单选项。子菜单不支持嵌套，即子菜单中不能再包括其他子菜单。
- 覆盖 Activity 的 onCreateOptionsMenu()方法，调用 Menu 的 addSubMenu()方法添加子菜单项。
- 调用 SubMenu 的 add()方法，添加子菜单项。
- 覆盖 onCreateItemSelected()方法，响应菜单单击事件。

代码如下：

```
1.  public class MenusActivity extends Activity {
2.      public void onCreate(Bundle savedInstanceState) {
3.          super.onCreate(savedInstanceState);
4.         setContentView(R.layout.main);
5.      }
6.      public boolean onCreateOptionsMenu(Menu menu) {
7.          super.onCreateOptionsMenu(menu);
8.         SubMenu fileMenu=menu.addSubMenu(1, 1, 1, "File");
9.         SubMenu editMenu=menu.addSubMenu(1, 2, 2, "edit");
10.         fileMenu.add(2, 11, 11, "New");
11.         fileMenu.add(2, 12, 12, "Save");
12.         fileMenu.add(2, 13, 13, "Close");
13.         editMenu.add(2, 21, 21, "first");
14.         editMenu.add(2, 22, 22, "second");
15.         return true;
16.     }
17.     public boolean onOptionsItemSelected(MenuItem item) {
18.        super.onOptionsItemSelected(item);
19.         switch(item.getItemId()){
20.            case 1:{
21.               Toast.makeText(MenusActivity.this,"单击了"+item.getTitle(),
                                        Toast.LENGTH_SHORT).show();
22.                break;
23.            }
24.            case 2:{
25.               Toast.makeText(MenusActivity.this, "单击了"+item.getTitle(),
                                         Toast.LENGTH_SHORT).show();
26.                break;
27.            }
28.            case 11:{
29.               Toast.makeText(MenusActivity.this, "单击了"+item.getTitle(),
                                         Toast.LENGTH_SHORT).show();
30.                break;
```

```
31.            }
32.            case 12:{
33.               Toast.makeText(MenusActivity.this, "单击了"+item.getTitle(),
                                          Toast.LENGTH_SHORT).show();
34.                break;
35.            }
36.            case 13:{
37.               Toast.makeText(MenusActivity.this, "单击了"+item.getTitle(),
                                          Toast.LENGTH_SHORT).show();
38.                break;
39.            }
40.            case 21:{
41.               Toast.makeText(MenusActivity.this, "单击了"+item.getTitle(),
                                          Toast.LENGTH_SHORT).show();
42.                break;
43.            }
44.            case 22:{
45.               Toast.makeText(MenusActivity.this, "单击了"+item.getTitle(),
                                          Toast.LENGTH_SHORT).show();
46.                break;
47.            }
48.        }
49.        return true;
50.    }
51. }
```

详细代码见本书配套资源中的工程 Chapter3.4.3。

3.4.4 定义 XML 菜单文件

在此之前我们都是直接在代码中添加菜单项、给菜单分组等。在代码中添加菜单项是存在很多不足的，比如为了响应每个菜单项，就要用常量来保存每个菜单的 ID 等。那么有什么更好的方法来添加菜单呢？其实在 Android 中，可以把 menu 也定义为应用程序资源（也就是定义 XML 菜单文件），通过 Android 对资源的本地支持，可以更方便地实现菜单的创建与响应。

如图 3-19 所示，单击 first1 菜单按钮，出现图 3-20 中的子菜单。

创建步骤如下：

（1）在/res 文件夹下创建 menu 文件夹。

（2）在 menu 文件夹下使用与 menu 相关的元素定义 XML 文件。

（3）使用 XML 文件的资源 ID，将 XML 文件中定义的菜单项添加到 menu 对象中。

（4）响应菜单，使用每个菜单项所对应的资源 ID。

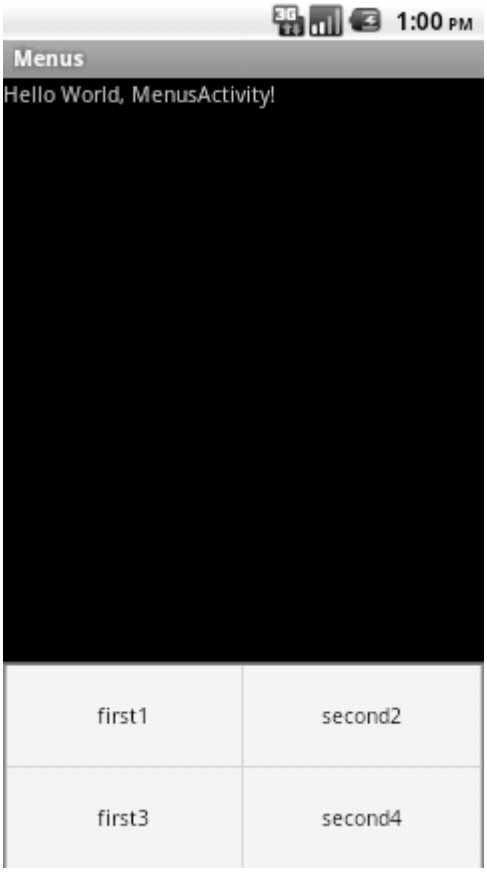

图 3-19　主菜单

图 3-20　子菜单

在菜单的 XML 文件中，如果要创建子菜单，在 item 元素下嵌套一个 menu 就可以实现。

代码如下：

（1）XML 代码。

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.    <menu xmlns:android="http://schemas.android.com/apk/res/android" >
3.      <item android:id="@+id/item1"
4.          android:title="first1">
5.          <menu >
6.               <item android:id="@+id/item5"
7.                    android:title="one"/>
8.                <item android:id="@+id/item6"
9.                    android:title="two"/>
10.         </menu>
11.      </item>
12.      <item android:id="@+id/item2"
```

```
13.          android:title="second2"/>>
14.      <item android:id="@+id/item3"
15.          android:title="first3"/>
16.      <item android:id="@+id/item4"
17.          android:title="second4"/>
18.      </menu>
```

（2）Java 代码。

```
1.  public class MenusActivity extends Activity {
2.              public void onCreate(Bundle savedInstanceState){
3.              super.onCreate(savedInstanceState);
4.              setContentView(R.layout.main);
5.      }
6.  public boolean onCreateOptionsMenu(Menu menu){
7.          MenuInflater inflater=getMenuInflater();
8.           inflater.inflate(R.menu.menu1, menu);
9.           return true;
10.     }
11. }
```

详细代码见本书配套资源中的工程 Chapter3.4.4。

3.5 事 件 响 应

事件处理在 Android 开发中是一个非常重要的课题。事件是用户与界面交互时所触发的操作，比如单击一个按钮就会触发一个按钮的单击事件。事件处理是应用程序与用户交互的必要环节。在 Android 框架的设计中，以事件监听器（event listener）的方式来处理 UI 的使用者事件。

3.5.1 基本事件

在之前 Button 的介绍中已经讲到了如何为 Button 设置监听事件。我们已经知道，View 是绘制 UI 的基础类别，每个 View 组件都可以向 Android 框架注册一个事件监听器。每个事件监听器都包含一个回调函数（callback method），这个回调函数的作用就是处理用户的操作。Android 中常见的事件有以下几种：

- onClick(View v) 一个普通的单击按钮事件。
- boolean onKeyMultiple(int keyCode,int repeatCount,KeyEvent event)用于在多个事件连续出现时发生，重复按键，必须通过重载实现。
- boolean onKeyDown（int keyCode,KeyEvent event）在按键按下时发生。
- boolean onKeyUp（int keyCode,KeyEvent event）在按键释放时发生。
- onTouchEvent（MotionEvent event）触摸屏事件，当在触摸屏上有动作时发生。

- boolean onKeyLongPress（int keyCode, KeyEvent event）当长时间按下时发生。

上面这些事件都是在事件监听器中进行的，当需要响应某个组件的某个事件动作时，先要为该组件注册一个该事件的监听器。

3.5.2 事件的响应

下面就来完成图 3-21 的单击效果。

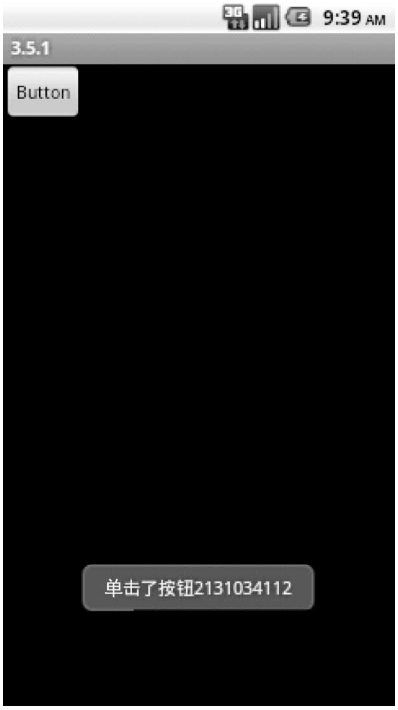

图 3-21 按钮单击

一个事件响应可通过 3 种方式实现。

首先是 XML 属性的方式，XML 布局代码如下：

```
1.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.      android:layout_width="fill_parent"
3.      android:layout_height="fill_parent"
4.      android:orientation="vertical" >
5.       <Button
6.          android:id="@+id/button1"
7.          android:layout_width="wrap_content"
8.          android:layout_height="wrap_content"
9.          android:onClick="click"
10.         android:text="Button"/>
11. </LinearLayout>
```

然后需要在 Activity 中写一个 click 函数作为 Button 的响应函数。代码如下：

```
1.  public class MainActivity extends Activity {
2.      public void onCreate(Bundle savedInstanceState) {
3.          super.onCreate(savedInstanceState);
4.          setContentView(R.layout.main);
5.
6.      }
7.
8.      public void click(View v)
9.      {
10. Toast.makeText(MainActivity.this, "单击了按钮" + v.getId()
    Toast.LENGTH_SHORT).show();
11.     }
12. }
```

代码第 8、9 行就是按钮的响应函数具体实现，需要注意的是，这个响应函数应写在调用这个布局文件的 Activity 类中，而且函数的参数必须按照标准的响应函数的参数格式。运行后的效果如图 3-14 所示。我们刚才实现的代码见本书配套资源中的工程 Chapter3.5.1。

再来看一下让 Activity 实现监听器完成事件响应，代码如下：

```
1.  public class MainActivity extends Activity implements OnClickListener {
2.      public void onCreate(Bundle savedInstanceState) {
3.          super.onCreate(savedInstanceState);
4.          setContentView(R.layout.main);
5.      }
6.      public void onClick(View v) {
7.
8.          if (v.getId() == R.id.button1)
9.              Toast.makeText(MainActivity.this, "单击了按钮"+v.getId(),
10.                     Toast.LENGTH_SHORT).show();
11.     }
12. }
```

注意看第 1 行，MainActivity 除了继承了 Activity 还继承了 OnClickListener 接口并重载了 onClick()方法，在 onClick()方法中可以根据 View 的 Id 做出不同的响应，其运行效果也如图 3-14 所示。刚才实现的代码见本书配套资源中的工程 Chapter3.5.2。

最后以内部类的方式为按钮添加一个监听器完成事件响应。代码如下：

```
1.  public class MainActivity extends Activity {
2.      public void onCreate(Bundle savedInstanceState) {
3.          super.onCreate(savedInstanceState);
4.          setContentView(R.layout.main);
```

```
5.          Button btn = (Button)findViewById(R.id.button1);
6.          btn.setOnClickListener(new OnBtnClick());
7.      }
8.      private class OnBtnClick implements OnClickListener {
9.          public void onClick(View v) {
10.             if (v.getId() == R.id.button1)
11.                 Toast.makeText(MainActivity.this, "单击了按钮"+v.getId(),
12.                         Toast.LENGTH_SHORT).show();
13.         }
14.     }
15. }
```

刚才实现的代码见本书配套资源中的工程 Chapter3.5.3。

至此，Android 的 3 种事件响应方式就都介绍完了，虽然只以 Button 的 OnClick()事件为例，但无论什么事件其原理都是一样的。读者可以自己研究一下其他事件的处理。

3.6 界面切换与数据传递

3.6.1 Intent 与 Bundle

Intent 与 Bundle 在界面切换中用得比较多。因为两个界面的切换肯定需要交代一些信息，这时就有 Intent 和 Bundle 的用武之地了，当然 Intent 的作用在于实现界面的切换。下面先来初步认识一下 Intent 和 Bundle，它们在数据传递中的用法会在后面中讲到。

1．Intent

我们已经知道 Android 有四大组件，这四大组件是独立的，它们之间相互协调，最终构成一个完整的 Android 应用。这些组件间的通信都是通过 Intent 的协助来完成的，Intent 负责对一次动作、动作涉及的数据进行描述，Android 根据这个 Intent 的描述找到相应组件，将 Intent 传给该组件并完成组件的调用。作为初学者需要理解的是 Intent 的两种启动模式。

- 显式的 Intent：即在构造 Intent 对象时就指定接收者，这种方式与普通的函数调用类似。
- 隐式的 Intent：即 Intent 的发送者在构造 Intent 对象时，并不知道也不关心接收者是谁。

1）显式的 Intent

在同一个程序中需要从当前 Activity 跳转到另一个指定的 Activity 时，就常用到显式的 Intent。

要创建一个显式的 Intent 可以使用构造函数 Intent(Context packageContext, Class<?> cls)。两个参数分别指定 Context 和 Class，Context 设置为当前的 Activity 对象，Class 设置为需要跳转到的 Activity 的类对象，比如要从当前界面跳到 TestActivity，就可以这样构造一个 Intent 对象：

```
1.  Intent intent = new Intent(this, TestActivity.class);
2.  startActivity(intent);
```

最后调用当前 Activity 的 startActivity()方法启动这个 Intent。除此之外，还可以采用下面的方法：

```
1.  Intent intent=new Intent();
2.  intent.setClass(this, TestActivity.class);
3.  startActivity(intent);
```

这段代码采用了 setClass()方法将第一段代码中的第 1 行写成了两行，但目的是一样的。但这里使用的 Activity 都必须在 AndroidManifest.xml 文件中配置。

2）隐式的 Intent

其实，Intent 机制更重要的作用在于隐式的 Intent，即 Intent 的发送者不指定接收者，很可能不知道也不关心接收者是谁，而由 Android 框架去寻找最匹配的接收者。比如，程序要调用 Android 的电话功能，就要采用下面这种方式创建一个 Intent。代码如下：

```
1. Intent intent = new Intent(Intent.ACTION_DIAL);
2. startActivity(intent);
```

代码第 1 行创建 Intent 采用 Intent(String action)的构造函数，Action 可以理解为描述这个 Intent 的一种方式，Intent 的发送者只要指定了 Action 为 Intent.ACTION_DIAL，系统就能找到对应的 Activity 作为接收者。Android 系统提供了很多的 Action，读者可以查看官方的 API 文档（reference\android\content\Intent）。同样也可以使用 setAction(String action)的方法来达到同样的效果。代码如下：

```
1. Intent intent = new Intent();
2. intent. setAction (Intent.ACTION_DIAL);
3. startActivity(intent);
```

不管是显式还是隐式的 Intent，在完成 Activity 的切换时都可能涉及数据的传递，Intent 提供了一系列的方法，如 putExtra(String name, String value)：采用 Key-Value 的形式，Key 是数据的键，Value 是数据的值。该方法在 Intent 中有多种重载形式，读者可以查看 Intent 的 API。有关数据传递部分将在后面详细讲解。

2．Bundle

Bundle 其实就是一个 Key-Value 的映射，前面介绍 Intent 时说到，Intent 描述数据的 putExtra 方法采用的是 Ket-Value 形式，其实 Intent 在内部定义时就有个 Bundle 类型的成员变量，putExtra 方法就是将传进来的参数放到这个 Bundle 变量中。所以 Bundle 有许多类似于 putExtra 的方法来存放数据，这里就不多讲解了。这里介绍 Bundle 用于传递对象时的几个方法。

首先要传递对象不能是一般的对象，这个对象必须是可序列化的，这就要求它们继承 Parcelable 或 Serializable 接口。

- putParcelable (String key, Parcelable value)：顾名思义，该方法用于将 Parcelable 接口的对象存入 Bundle 中，对应的 getParcelable (String key)用于接收键值为 key 的 Parcelable 对象。
- putSerializable(String key, Serializablevalue)：与 putParcelable 的用法一样，只不过该方法用于存放实现了 Serializable 接口的对象。

Intent 和 Bundle 就先了解到这里，有关它们的其他知识会在后面涉及，也可查看官方 API 进行更深入的了解。

3.6.2 界面切换

在一个程序中，经常需要将当前的 Activity 转跳到另一个 Activity 进行操作，这时就需要运用到 Activity 切换。Activity 切换有两种方式，下面就用这两种方式来实现从图 3-22 的界面切换到图 3-23 的界面。

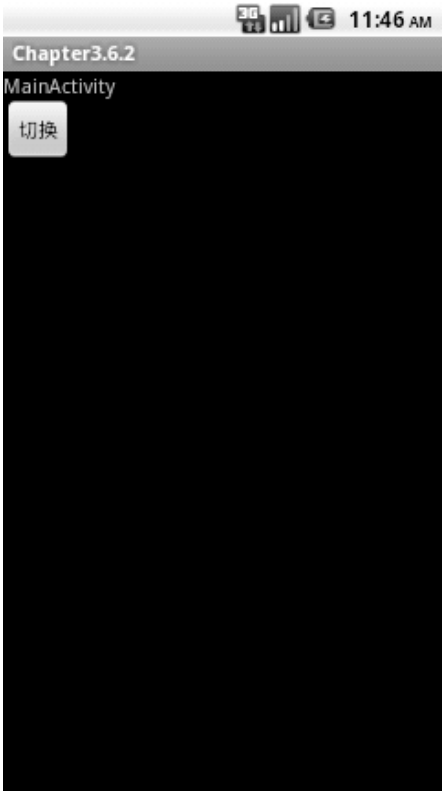

图 3-22　MainActivity 界面

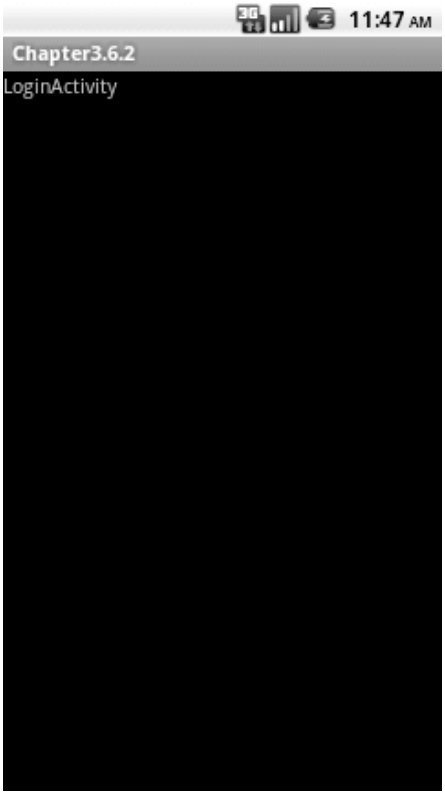

图 3-23　LoginActivity 界面

方法一，可以通过 setContentView 切换布局来实现界面的切换。步骤如下：

（1）新建一个想要切换的界面的 XML 文件。

（2）通过触发一个加载了监听器的控件，在监听器中使用 setContentView 函数切换界面。这样的实现过程都是在一个 Activity 上面实现的，所有变量都在同一状态，因此

所有变量都可以在这个 Activity 状态中获得。这里可以设置一个加载了监听器的 Button 按钮，实现代码如下：

```
1.  public void onCreate(Bundle savedInstanceState) {
2.     super.onCreate(savedInstanceState);
3.     setContentView(R.layout.main);
4.     Button button = (Button) this.findViewById(R.id.button1);
5.        button.setOnClickListener(new OnClickListener() {
6.     public void onClick(View v) {
7.       setContentView(R.layout.login);
8.                  }
9.          });
10.    }
```

这样就从 main 这个 XML 布局切换到了 login 这个 XML 布局。但严格意义上讲，这并不是一种界面的切换而是对界面的重画，就像是老师上课的时候将黑板上以前的板书擦掉重新写一块板书。

> ☞**注意**：在有些教程中会看到方法一，但我们不提倡使用这种方法，应使用方法二。

刚才实现的代码见本书配套资源中的工程 Chapter3.6.1。

方法二，在一个程序中往往会使用 Intent 对象来指定一个 Activity，并通过 startActivity 方法启动这个 Activity。当需要在不同的 Activity 之间进行切换的时候，可以在响应事件中实例化一个 Intent 对象作为 startActivity 方法的参数，从而实现不同 Activity 的切换。

下面完成相同的界面切换，我们需要做的是将方法一代码中的第 9 行改成下面这段代码：

```
1.  Intent intent=new Intent();
2.      intent.setClass(MainActivity.this, LoginActivity.class);
3.      MainActivity.this.startActivity(intent);
```

上述代码实现了从当前 MainActivity 切换到 LoginActivity。

刚才实现的代码见本书配套资源中的工程 Chapter3.6.2。

3.6.3 传递数据

1．Intent 数据传递

之前已经介绍了 Android 利用 Intent 完成 Activity 间的切换。在界面切换的时候经常需要数据的传递，同样需要用到 Intent。我们先来完成两个简单的 Activity 间的数据传递。首先需要两个 Activity：DataTransferActivity 和 ShowActivity，界面效果如图 3-24 所示。

图 3-24　Activity 间的数据传递

当单击了 DataTransferActivity 中的“确认”按钮时就会切换到 ShowActivity 界面，同时将编辑框中的书名和 ID 传递到 ShowActivity 中显示出来。下面主要研究数据传递的代码。下面是事件响应代码。

```
1.  Button btn = (Button) findViewById(R.id.button1);
2.          btn.setOnClickListener(new OnClickListener() {
3.              public void onClick(View v) {
4.                  EditText bookNameTV = (EditText) findViewById(R.id.
                                        editText1);
5.                  EditText bookIdTV = (EditText) findViewById(R.id.
                                        editText2);
6.                  Intent intent = new Intent(DataTransferActivity.this,
                                        ShowActivity.class);
7.                  intent.putExtra("bookName",bookNameTV.getText().
                                            toString());
8.                  intent.putExtra("bookId", bookIdTV.getText().toString());
9.                  DataTransferActivity.this.startActivity(intent);
10.             }
11.         });
```

注意第 7、8 行代码，这里调用了 Intent 的 putExtra 方法传递 bookName 和 bookId，

而且它们每一个都有一个键值，这是为了方便获取时使用。putExtra 方法在 Intent 中有多种重载形式，可以存放多种类型的数据。那么接下来再看一下 ShowActivity 是怎样接收数据的。代码如下：

```
1.  public void onCreate(Bundle savedInstanceState) {
2.      super.onCreate(savedInstanceState);
3.      Intent intent = getIntent();
4.      String bookName = intent.getStringExtra("bookName");
5.      String bookId =  intent.getStringExtra("bookId");
6.      TextView textview = new TextView(this);
7.      textview.setText("bookName：+bookName+"\nbookId：+bookId);
8.      setContentView(textview);}
```

上面的代码中第 4、5 行的作用就是接收数据。它调用了 Intent 的 getStringExtra 方法，用于获取 String 类型的数据，其他数据类型获取方法也与此类似，就是通过键值来获取对应的数据。但是有时候数据太多，类型也不一样，使用这样的方法就有些复杂了，下面介绍另一种数据传递的方式——利用 Bundle 传递数据。

2．Bundle 传递对象

查看 Intent 的源码可以发现，其实 Intent 的数据传递同样是将数据绑定在 Bundle 中传递的。

Bundle 还有一个功能就是传递对象，但前提是这个对象需要序列化。还是如图 2-9 所示的效果，但这次将 bookName 和 bookId 封装在一个 Book 对象中，然后利用 Bundle 和 IntentBook 对象传递到 ShowActivity 中。首先看一下 Book 类的代码。代码如下：

```
1.  public class Book implements Serializable{
2.      public String getBookName() {
3.          return bookName;
4.      }
5.      public String getBookId() {
6.          return bookId;
7.      }
8.      private String bookName = null;
9.      private String bookId = null;
10.
11.     public Book(String bookName,String bookId)
12.     {
13.         this.bookId = bookId;
14.         this.bookName = bookName;
15.     }
16.     }
```

因为需要 Book 的对象是可序列化的，所以第 1 行代码让 Book 继承了 Serializable 接口。再看一下 DataTransferActivity 中事件监听改变的代码：

```
1.  btn.setOnClickListener(new OnClickListener() {
2.
3.              public void onClick(View v) {
4.                  EditText bookNameTV = (EditText) findViewById(R.id.
                                                editText1);
5.                  EditText bookIdTV = (EditText) findViewById(R.id.
                                                editText2);
6.                 Intent intent = new Intent(DataTransferActivity.this,
                                                ShowActivity.class);
7.                  Bundle mExtra = new Bundle();
8.                  String bookName = bookNameTV.getText().toString();
9.                  String bookId = bookIdTV.getText().toString();
10.                 Book book= new Book(bookName,bookId);
11.                 mExtra.putSerializable("book", book);
12.                 intent.putExtras(mExtra);
13.                 DataTransferActivity.this.startActivity(intent);
14.
15.             }
16.         });
```

上面代码的第 7 行定义了一个 Bundle 对象 mExtra；第 10 行代码将 bookName 和 bookId 封装到旧 book 对象中，因为 Book 实现的是 Serializable 接口所以调用 Bundle 中对应的 putSerializable 方法，参数同样采用了 Key-Value 的形式；最后调用 intent 的 putExtras 方法将 mExtra 存放到 intent 中。这样剩下的就是如何接收了。下面就来看下 ShowActivity 是怎样接收的。代码如下：

```
1.  public void onCreate(Bundle savedInstanceState) {
2.      super.onCreate(savedInstanceState);
3.      Intent intent = getIntent();
4.      Book book = (Book) intent.getSerializableExtra("book");
5.      String bookName = book.getBookName();
6.      String bookId =  book.getBookId();
7.      TextView textview = new TextView(this);
8.      textview.setText("bookName：+bookName+"\nbookId：+bookId);
9.      setContentView(textview);
10. }
```

既然传递时存放对象采用的是 putSerializable 方法，接收时肯定采用对应的 getSerializableExtra 方法，参数就是这个对象的键值，然后再将之强制转换为 Book 对象，并调用 Book 的相应方法获得想要的数据。

这里使用的继承 Serializable 接口序列化对象，还可以实现 Parcelable 接口，但对应的存放和读取对象方式就改为 putParcelable 和 getParcelable 方法。

本部分代码见本书配套资源中的工程 Chapter3.6.3。

3.7 Activity 界面刷新

在进行 Activity 刷新时不能在子线程中进行，只能在该程序的主线程中刷新 Activity。当需要在子线程中进行刷新时，主线程运行子线程，然后从子线程中返回一个刷新的操作对主线程进行刷新，如图 3-25 所示。

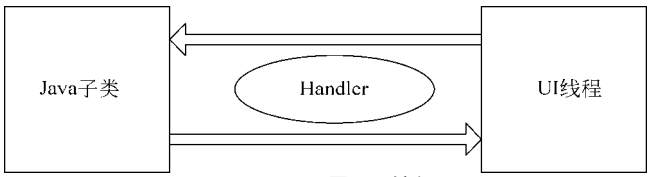

图 3-25 界面刷新

3.8 Activity 栈及 4 种启动模式

3.8.1 Activity 栈概述

1．Activity 栈

开发者是无法控制 Activity 的状态的，那么 Activity 的状态又是按照何种逻辑来运作的呢？这就要知道 Activity 栈。

每个 Activity 的状态是由它在 Activity 栈（包含所有正在运行 Activity 的队列）中的位置决定的。

当一个新的 Activity 启动时，当前活动的 Activity 将会移到 Activity 栈的顶部。

如果用户使用后退按钮返回，或者前台的 Activity 结束，在栈上的 Activity 将会移上来并变为活动状态。

一个应用程序的优先级是受最高优先级的 Activity 影响的。当决定某个应用程序是否要终结去释放资源，Android 内存管理使用栈来决定基于 Activity 的应用程序的优先级。

2．Activity 的 4 种状态

1）活动的

当一个 Activity 在栈顶时，它是可视、有焦点、可接收用户输入的。Android 试图尽最大可能保持其活动状态，杀死其他 Activity 来确保当前活动 Activity 有足够的资源可以使用。当另一个 Activity 被激活时，这个将会被暂停。

2）暂停

在很多情况下，某个 Activity 可视但是它没有焦点，换句话说它被暂停了。可能的原因是一个透明或者非全屏的 Activity 被激活。

当被暂停时，一个 Activity 仍会处于活动状态，只不过是不可以接收用户输入。在极特殊的情况下，Android 将会杀死一个暂停的 Activity 来为活动的 Activity 提供充足的资源。当一个 Activity 变为完全隐藏的时候，它将会变成停止。

3）停止

当一个 Activity 不是可视的，它就是“停止”的。这个 Activity 将仍然在内存中保存其所有的状态和会员信息。尽管如此，当其他地方需要内存时，它将是最有可能被释放资源的。当一个 Activity 停止后，一个很重要的步骤是要保存数据和当前 UI 状态。一旦一个 Activity 退出或关闭了，它将变为待用状态。

4）待用

在一个 Activity 被杀死后和被装载前，它是待用状态的。待用 Acitivity 被移出 Activity 栈，并且需要在显示和可用之前重新启动它。

3.8.2 Activity 启动模式定义方法

在 Android 的多 Activity 开发中，Activity 之间的跳转可能需要有多种方式，有时是普通地生成一个新实例，有时希望跳转到原来某个 Activity 实例，而不是生成大量重复的 Activity。加载模式便是决定以哪种方式启动一个或跳转到原来某个 Activity 实例。

启动模式有两种不同的定义方法。

1）使用清单文件

当在清单文件（AndroidManifest.xml）中声明一个 Activity 时，你能够指定这个 Activity 在启动时应该如何与任务进行关联。这些启动模式可以在功能清单文件 AndroidManifest.xml 中设置 launchMode 属性。

2）使用 Intent 标识

在调用 startActivity()方法时，你能够在 Intent 中包含一个标识，用来声明这个新的 Activity 应该如何与当前的任务进行关联。

Intent 标识设置实例如下：

```
1.  Intent intent = new Intent(MainActivity.this, SecActivity.class);
2.  intent.addFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
3.  startActivity(intent);
```

下面列举一些常用的 Intent 标识：

- FLAG_ACTIVITY_BROUGHT_TO_FRONT——这个标识一般不是由程序代码设置的，如在 launchMode 中设置 singleTask 模式时系统帮你设定。
- FLAG_ACTIVITY_CLEAR_TOP——当这个 Activity 已经在当前的 Task 中运行时，Android 不再重新启动一个这个 Activity 的实例，而是在这个 Activity 上方的所有 Activity 都将关闭，然后这个 Intent 会作为一个新的 Intent 投递到旧的 Activity（现在位于顶端）中。
- FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET——若设置了这个标识，则将在 Task 的 Activity 栈中设置一个还原点，当 Task 恢复时，需要清理 Activity。也就是说，下一次 Task 带着 FLAG_ACTIVITY_RESET_TASK_IF_NEEDE 标识进入前台时（典型的操作是用户在主画面重启它），这个 Activity 和它之上的 Activity 都将关闭，以至于用户不能再返回到它们，但是可以回到之前的 Activity。

- FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS——若设置了这个标识，则新的 Activity 不会在最近启动的 Activity 的列表中保存。
- FLAG_ACTIVITY_FORWARD_RESULT——如果这个 Intent 用于从一个存在的 Activity 启动一个新的 Activity，那么作为答复目标的 Activity 将会传到新的 Activity 中。在这种方式下，新的 Activity 可以调用 setResult(int)，并且这个结果值将发送给那个作为答复目标的 Activity。

除此以外，Intent 还有很多的标识，无法全部列举，读者可以查看 Intent 的 API 了解更多的标识。

3.8.3 standard 启动模式

这是默认模式，每次激活 Activity 时都会调用 startActivity()方法创建一个新的 Activity 实例，并放入任务栈中。

假如此时有 A、B 两个 Activity，如果 Activity A 采用 standard 模式启动 Activity B，则不管 Activity B 是否已经有一个实例位于 Activity 栈中，都会产生一个 Activity B 的新实例并放入任务栈中。

清单文件配置方式：

```
1.  <activity
2.       android:name=".StandardActivity"
3.       android:label="@string/standard"
4.       android:launchMode="standard">
5.       </activity>
```

standard 的运行机制，部分代码如下：

```
1.  private TextView text;
2.        private Button button;
3.           public void onCreate(Bundle savedInstanceState) {
4.           super.onCreate(savedInstanceState);
5.           setContentView(R.layout. mainactivity);
6.           text = (TextView) this.findViewById(R.id.text);
7.           text.setText(this.toString());
8.           button = (Button) this.findViewById(R.id.button_stand);
9.        }
10.       //按钮单击事件
11.       public void LaunchStandard(){
12.          startActivity(new Intent(this,StandardActivity.class));
13.          text.setText(this.toString());
14.       }
```

初始化界面如图 3-26 所示。

当单击按钮时，会创建新的 Activity，通过 TextView@后的十六进制数的显示可看

出界面如图 3-27 所示。

图 3-26　standard 模式的初始界面

图 3-27　单击按钮后的界面

再次单击按钮时，还会创建新的 Activity，通过 TextView@后的十六进制数的显示可看出界面如图 3-28 所示。

分析其运行机制可知，当程序运行到此时，栈中的数据形式如图 3-29 所示。

图 3-28　再次单击按钮后的界面

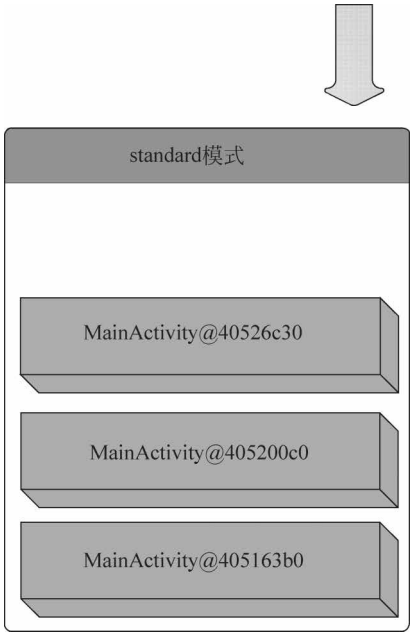

图 3-29　栈中的数据形式

因此，这种 standard 模式是每次都会创建新的 Activity 对象，当单击返回按钮时，它会将栈顶（当前 Activity）消灭，然后跳到下一层，例如，如果现在 Activity 是 405263c0，那么当单击返回时 Activity 会变为 405200c0，不过此时在这个 Activity 中再次单击按钮创建对象时，它会另外创建新的 Activity 对象，这种模式可能在大多数情况下不是我们需要的，因为对系统性能的消耗过大。

完整代码见工程 Chapter3.8.3。

3.8.4 singleTop 启动模式

如果已经有一个实例位于 Activity 栈的顶部，就不产生新的实例，而只是调用 Activity 中的 newInstance()方法。如果不位于栈顶，则会产生一个新的实例。

清单文件配置方式：

```
1.  <activity
2.      android:name=".SingleTopActivity"
3.      android:label="@string/singleTop"
4.      android:launchMode="singleTop" >
5.      </activity>
```

界面初始化如图 3-30 所示。

单击“启动 singleTop 模式”按钮，结果如图 3-31 所示。

图 3-30 singleTop 模式的初始界面

图 3-31 单击“启动 singleTop 模式”按钮后的界面

分析其运行机制可知，当程序运行到此时，栈中的数据形式如图 3-32 所示。

再次单击“启动 singleTop 模式”按钮，结果如图 3-33 所示，由于此 Activity 设置的启动模式为 singleTop，因此它首先会检测当前栈顶是否为我们要请求的 Activity 对象，经验证成立，因此它不会创建新的 Activity，而是引用当前栈顶的 Activity。

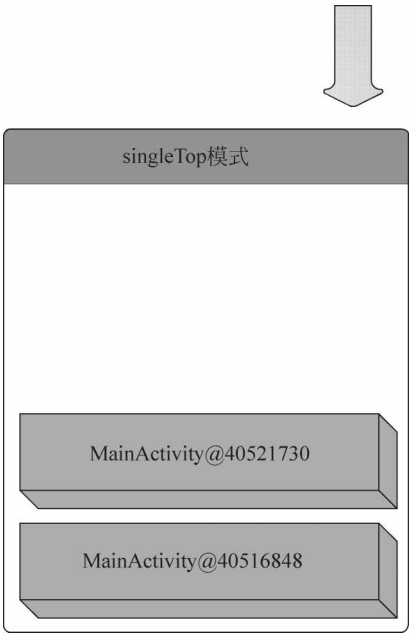

图 3-32　栈中的数据形式

图 3-33　再次单击“启动 singleTop 模式”按钮

此时栈中的数据形式依然为图 3-32。

完整代码见工程 Chapter3.8.4。

3.8.5　singleTask 启动模式

如果在栈中已经有该 Activity 的实例，则重用该实例（会调用实例的 onNewIntent()）。重用时，会让该实例回到栈顶，因此在它上面的实例将会被移出栈。如果栈中不存在该实例，将会创建新的实例放入栈中。

清单文件配置方式：

```
1.  <activity
2.       android:name=".SingleTaskActivity"
3.       android:label="@string/singleTask"
4.       android:launchMode="singleTask" >
5.       </activity>
6.
```

界面初始化为如图 3-34 所示的形式。

单击“启动 singleTask 模式”按钮，界面如图 3-35 所示。

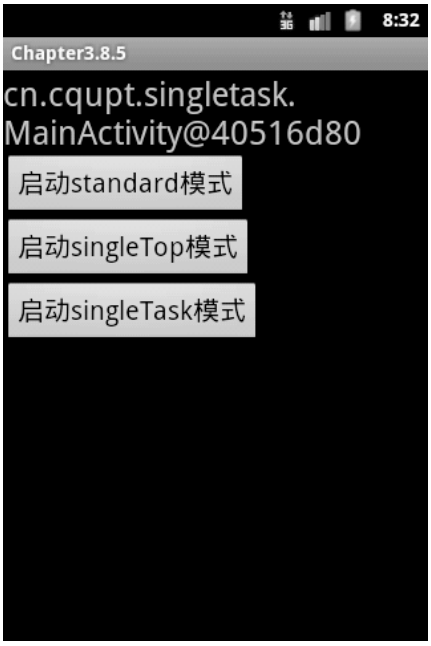

图 3-34 singleTask 模式初始化界面

图 3-35 单击“启动 singleTask 模式”按钮

在此界面中单击“启动 singleTask 模式”按钮，根据定义会检测当前栈中是否有此 Activity 对象，因此显示的还是当前的 Activity，不会重新创建，如图 3-36 所示。

再次单击“启动 standard 模式”按钮，由于 MainActivity 的启动模式为 standard，所以在此会重新创建一个 MainActivity 对象，如图 3-37 所示。

图 3-36 再次单击“启动 singleTask 模式”按钮

图 3-37 再次单击“启动 standard 模式”按钮

此时栈中数据形式如图 3-38 所示。

当在如图 3-39 所示界面中单击“启动 singleTask 模式”按钮时，由于检测到当前栈中第二个为我们要创建的 Activity，所以会将最上面的 MainActivity 消灭，然后将 SingleTaskActivity 设置为栈顶。

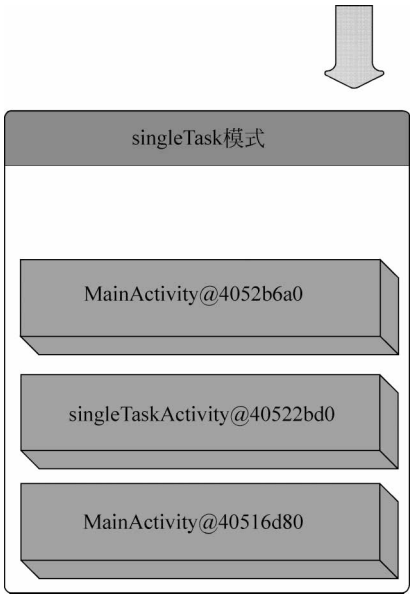

图 3-38　栈中数据形式

图 3-39　再次单击“启动 singleTask 模式”按钮

此时栈中数据形式变为如图 3-40 所示。

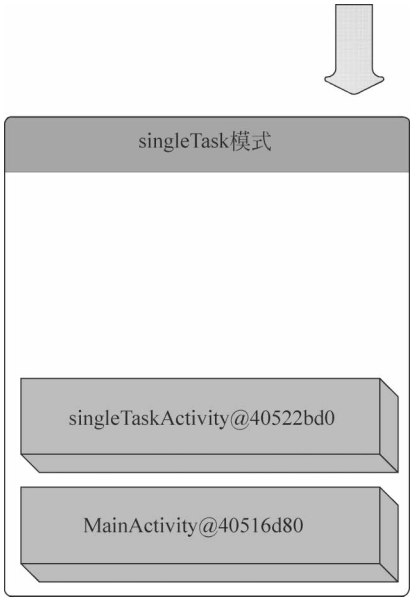

图 3-40　栈中数据形式

完整代码见工程 Chapter3.8.5。

3.9 有多个界面的单机版图书管理系统

本节将在 Chapter02 上进行改进，实现一个有多个界面的单机版图书管理系统。本节灵活运用了本章所学的 3 个知识点：界面组件、布局、事件监听和 Activity 之间的切换。首先在虚拟机上运行 Chapter03，效果如图 3-41 所示。

图 3-41 运行效果图

图 3-41（续）

在 Chapter03 中有 3 个包，分别是 ui.cqupt、control.cqupt 和 model.cqupt，如图 3-42 所示。

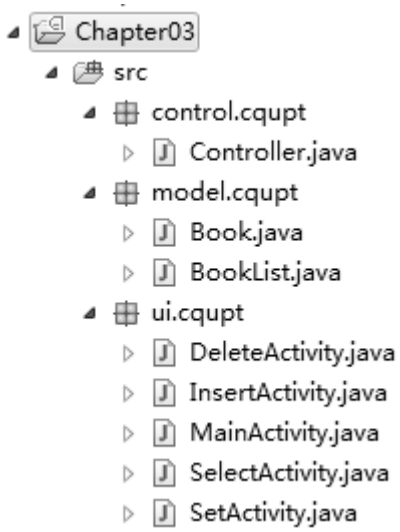

图 3-42　包结构

ui.cqupt 包的功能是对界面操作，包括了图 3-42 所示的所有界面的类。control.cqupt 包起到控制的作用，ui.cqupt 包通过 control.cqupt 包对 model.cqupt 包进行操控。model.cqupt 包用于存放模型，里面的 BookList 存储了图书的信息。这种界面、控制、模型的设计思路就是典型的 MVC 设计模式，MVC 设计模式的框架如图 3-43 所示。

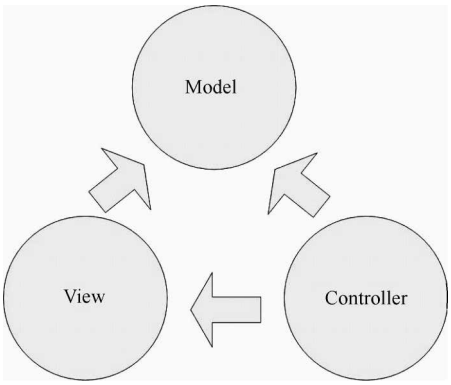

图 3-43　MVC 三层架构

1）什么是 MVC 设计模式

MVC 模式（三层架构模式）（Model-View-Controller）是软件工程中的一种软件架构模式，把软件系统分为 3 个基本部分：模型（Model）、视图（View）和控制器（Controller）。

控制器（Controller）——负责转发请求，对请求进行处理。

视图（View）——界面设计人员在此进行图形界面设计。

模型（Model）——程序员利用它编写程序应有的功能（实现算法等等）、数据库专家进行数据管理和数据库设计（可以实现具体的功能）。

2）为什么要使用 MVC 设计模式

MVC 实现了视图层和业务层分离，这样就允许更改视图层代码而不用重新编写模型和控制器代码；同样，一个应用的业务流程或者业务规则的改变只需要改动 MVC 的模型层即可，因为模型与控制器和视图相分离。这种分离有许多好处：

（1）清晰地将应用程序分隔为独立的部分；

（2）业务逻辑代码能够很方便地在多处重复使用；

（3）方便开发人员分工协作；

（4）如果需要，可以方便开发人员对应用程序各个部分的代码进行测试。

这里已经对 Chapter03 的整体结构和设计思路有了一定的了解。Chapter03 中各个类之间的关系如图 3-44 所示。

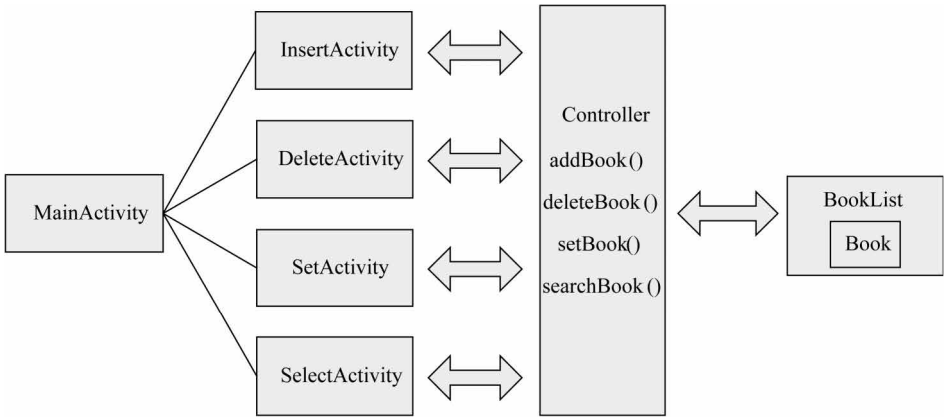

图 3-44　各个类之间的关系

下面主要对增加图书的流程做详细讲解，删除、修改和查询图书将在增加图书流程讲解之后介绍。

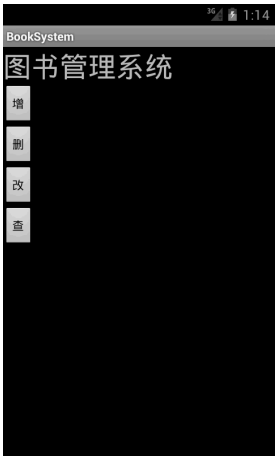

图 3-45　Chpater03 主界面

Chapter03 主界面如图 3-45 所示是由 ui.cqupt 包中的 MainActivity.java 实现的，运用

了图形界面和事件监听的知识点。下面将对主界面的实现进行详细的分析，MainActivity.java 代码如下所示：

```
1.  package ui.cqupt;
2.
3.  import ui.cqupt.R;
4.  import android.app.Activity;
5.  import android.content.Intent;
6.  import android.os.Bundle;
7.  import android.view.View;
8.  import android.view.View.OnClickListener;
9.  import android.widget.Button;
10.
11. public class MainActivity extends Activity {
12.
13.     public void onCreate(Bundle savedInstanceState) {
14.         super.onCreate(savedInstanceState);
15.         setContentView(R.layout.main);
16.         Button insert = (Button) findViewById(R.id.m_insert);
17.         Button delete = (Button) findViewById(R.id.m_delete);
18.         Button set = (Button) findViewById(R.id.m_set);
19.         Button select = (Button) findViewById(R.id.m_select);
20.         ButtonListener buttonListener = new ButtonListener();
21.         insert.setOnClickListener(buttonListener);
22.         delete.setOnClickListener(buttonListener);
23.         set.setOnClickListener(buttonListener);
24.         select.setOnClickListener(buttonListener);
25.     }
26.
27.     class ButtonListener implements OnClickListener {
28.
29.         public void onClick(View v) {
30.             int id = v.getId();
31.             Intent intent = new Intent();
32.             switch (id) {
33.             case R.id.m_insert:
34.                 intent.setClass(MainActivity.this, InsertActivity.class);
35.                 MainActivity.this.startActivity(intent);
36.                 break;
37.             case R.id.m_delete:
38.                 intent.setClass(MainActivity.this, DeleteActivity.class);
39.                 MainActivity.this.startActivity(intent);
40.                 break;
41.             case R.id.m_set:
42.                 intent.setClass(MainActivity.this, SetActivity.class);
43.                 MainActivity.this.startActivity(intent);
44.                 break;
45.             case R.id.m_select:
46.                 intent.setClass(MainActivity.this, SelectActivity.class);
```

```
47.                 MainActivity.this.startActivity(intent);
48.                 break;
49.             }
50.         }
51.
52.     }
53. }
```

MainActivity 首先覆写了 onCreate(Bundle savedInstanceState)方法，在方法体中第 15 行设置了 MainActivity 的界面布局——main.xml，布局文件 main.xml 代码如下所示：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:tools="http://schemas.android.com/tools"
4.      android:layout_width="fill_parent"
5.      android:layout_height="fill_parent"
6.      android:orientation="vertical"
7.      tools:ignore="HardcodedText" >
8.      <TextView
9.          android:textSize="15pt"
10.         android:layout_width="fill_parent"
11.         android:layout_height="wrap_content"
12.         android:text="图书管理系统" />
13.     <Button
14.         android:id="@+id/m_insert"
15.         android:layout_width="wrap_content"
16.         android:layout_height="wrap_content"
17.         android:text="增" />
18.     <Button
19.         android:id="@+id/m_delete"
20.         android:layout_width="wrap_content"
21.         android:layout_height="wrap_content"
22.         android:text="删" />
23.     <Button
24.         android:id="@+id/m_set"
25.         android:layout_width="wrap_content"
26.         android:layout_height="wrap_content"
27.         android:text="改" />
28.     <Button
29.         android:id="@+id/m_select"
30.         android:layout_width="wrap_content"
31.         android:layout_height="wrap_content"
32.         android:text="查" />
33. </LinearLayout>
```

主界面的布局采用了线性布局，所有组件都放在<LinearLayout></LinearLayout>标签中，第 13～17 行的<Button/>标签为声明一个按钮组件，每个 Button 标签代表一个按钮

组件，标签中的代码为定义这个组件的属性，第 14 行为此 Button 设定一个名为 m_insert 的 id，程序会在 R.java 中生成一个 id 名为 m_insert 的代码，此 id 唯一标识这个 Button 组件。第 17 行设定了 Button 组件上的显示，其余 3 个组件也以相同的方式设定。R.java 中的代码如下所示：

```
1.  public static final int m_delete=0x7f050007;
2.  public static final int m_insert=0x7f050006;
3.  public static final int m_select=0x7f050009;
4.  public static final int m_set=0x7f050008;
```

MainActivity.java 中第 16～19 行通过 findViewById(R.id.XX)方法查找 R.java 文件中生成名为 xx 的 id 号来实例化在布局文件中声明的按钮主件。第 20 行定义一个 Button 监听器对象；第 21～24 行，通过 Button 类的 setOnClickListener()方法设置处理单击事件的对象实例；第 27～52 行定义了一个 ButtonListener 的内部类来实现监听器，ButtonLiistener 实现了 android.view.View.OnClickListener 接口的 public void onClick(View v) 方法处理单击事件。

第 29～50 行实现了单击事件的处理，完成了 Activity 之间的切换功能。第 30 行得到按钮的唯一标识 id，并通过 switch 语句判断单击事件的执行代码。第 31 行定义了一个 intent 对象，Activity 通过 intend 对象完成界面的切换， 第 34 行通过 Intend 的 setClass(MainActivity.this, InsertActivity.class)设置了从主界面跳转到 InsertActivity 插入界面，在第 35 行通过访问外部类的 startActivity(intent)方法启动 intent 实现界面切换，其余的 3 个组件也通过同样的方式实现了 Activity 的切换。4 个按钮组件执行后的结果如图 3-46～图 3-49 所示。

图 3-46 插入界面（InsertActivity）

图 3-47 删除界面（DeleteActivity）

图 3-48　修改界面（SetActivity）

图 3-49　查询界面（SelectActivity）

首先分析插入界面（insertActivity）。InsertActivity 位于 ui.cqupt 包中，代码如下所示：

```
1.  package ui.cqupt;
2.
3.  import control.cqupt.Controller;
4.  import ui.cqupt.R;
5.  import android.app.Activity;
6.  import android.app.AlertDialog.Builder;
7.  import android.content.DialogInterface;
8.  import android.os.Bundle;
9.  import android.view.View;
10. import android.view.View.OnClickListener;
11. import android.widget.Button;
12. import android.widget.EditText;
13.
14. public class InsertActivity extends Activity {
15.     private EditText name;
16.     private EditText id;
17.     private EditText price;
18.
19.     public void onCreate(Bundle savedInstanceState) {
20.         super.onCreate(savedInstanceState);
21.         setContentView(R.layout.insert);
22.         name = (EditText) findViewById(R.id.name);
```

```
23.         id = (EditText) findViewById(R.id.id);
24.         price = (EditText) findViewById(R.id.price);
25.         Button insert = (Button) findViewById(R.id.i_insert);
26.         insert.setOnClickListener(new ButtonListener());
27.     }
28.
29.     class ButtonListener implements OnClickListener {
30.
31.         public void onClick(View v) {
32.             String bookname = name.getText().toString();
33.             String bookid = id.getText().toString();
34.             String bookprice = price.getText().toString();
35.
36.             Controller control = new Controller();
37.             if (bookname.equals("") || bookid.equals("")
38.                     || bookprice.equals("")) {
39.             new Builder(InsertActivity.this).setMessage("图书信息不
                能为空").show();
40.             } else {
41.                 if (control.addBook(bookid, bookname, bookprice)) {
42.                     id.setText("");
43.                     name.setText("");
44.                     price.setText("");
45.                     buildDialog();
46.                 } else {
47.                     new Builder(InsertActivity.this).setMessage("已
                        有此图书").show();
48.                 }
49.             }
50.         }
51.
52.         private void buildDialog() {
53.             Builder builder = new Builder(InsertActivity.this);
54.             builder.setTitle("插入成功,是否继续插入图书");
55.             builder.setNegativeButton("返回首页",
56.                     new DialogInterface.OnClickListener() {
57.                         public void onClick(DialogInterface dialog,
58.                                 int whichButton) {
59.                             finish();
60.                         }
61.
62.                     });
63.             builder.setPositiveButton("继续插入", null);
64.             builder.show();
```

```
65.         }
66.
67.     }
68. }
```

InsertActivity 继承了 Activity 并且覆写了 onCreate()方法初始化界面。本界面的布局采用线性布局，使用了 Button、TextView 和 EditText 组件，布局文件 insert.xml 代码如下所示：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:tools="http://schemas.android.com/tools"
4.      android:layout_width="fill_parent"
5.      android:layout_height="fill_parent"
6.      android:gravity="center|top"
7.      android:orientation="vertical"
8.      tools:ignore="HardcodedText" >
9.
10.     <TextView
11.         android:layout_width="fill_parent"
12.         android:layout_height="wrap_content"
13.         android:gravity="center_horizontal"
14.         android:text="请输入图书信息"
15.         android:textSize="15pt" />
16.
17.     <TextView
18.         android:layout_width="wrap_content"
19.         android:layout_height="wrap_content"
20.         android:text="图书编号" />
21.
22.     <EditText
23.         android:id="@+id/id"
24.         android:layout_width="180dp"
25.         android:layout_height="wrap_content"
26.         android:background="#FFFFFF"
27.         android:ems="10"
28.         android:inputType="text"
29.         android:textColor="#000000" />
30.
31.     <TextView
32.         android:layout_width="wrap_content"
33.         android:layout_height="wrap_content"
34.         android:text="图书名称" />
35.
```

```
36.     <EditText
37.         android:id="@+id/name"
38.         android:layout_width="180dp"
39.         android:layout_height="wrap_content"
40.         android:background="#FFFFFF"
41.         android:ems="10"
42.         android:inputType="text"
43.         android:textColor="#000000" />
44.
45.     <TextView
46.         android:layout_width="wrap_content"
47.         android:layout_height="wrap_content"
48.         android:text="图书价格" />
49.
50.     <EditText
51.         android:id="@+id/price"
52.         android:layout_width="180dp"
53.         android:layout_height="wrap_content"
54.         android:background="#FFFFFF"
55.         android:ems="10"
56.         android:inputType="text"
57.         android:textColor="#000000" />
58.
59.     <Button
60.         android:id="@+id/i_insert"
61.         android:layout_width="102dp"
62.         android:layout_height="wrap_content"
63.         android:text="插入" />
64.
65. </LinearLayout>
```

在 insert.xml 文件中，第 10～15 行通过 TextView 组件在界面上显示 “请输入图书信息”的信息，第 22～29 行通过<EditText/>标签声明一个文本框，在文本框中可输入图书的信息。Insert.xml 文件中所有<EditText/>标签都是声明的一个文本框，通过 android:id="@+id/ 名称 " 在 R.java 文件中生成唯一标识 id。第 26 行 android:background="#FFFFFF"设置文本框背景颜色，第 27 行设置字体大小，第 28 行设置输入类型，第 29 行设置输入的字体颜色。第 59～63 行声明了一个 Button 组件，命名为“插入”。

现在回到 InsertActivity.java 的代码中，第 15～17 行声明了 EditText 组件，分别命名为 id、name、price。在第 22～24 行通过 findViewById(R.id.xx)实例化 EditText 组件，第 25 行实例化 Button 组件，并且在第 26 行通过 Button 类的 setOnClickListener()方法设置处理单击事件的对象实例。第 29～67 行定义了一个 ButtonListener 的内部类来实现监听

器，并且实现了 public void onClick(View v)方法进行事件处理。第 32～34 行通过 getText()方法得到 EditText 文本框中的内容，并且使用 toString()方法转换为字符串。第 36 行创建一个 control 对象，通过 control 对象对存储数据的 BookList 操作。第 37～50 行是对事件处理的核心，首先在第 37、38 行判断输入框是否为空，如果为空，就生成一个消息对话框，显示“图书信息不能为空”。第 41～48 行通过调用 control 的 addBook 方法对 BookList 进行增加操作，如果返回真，则执行第 42～45 行，把输入框设置为空，并且在界面上弹出一个对话框，第 45 行通过自定义的 builderDialogue 建立对话框。如果返回假，就生成一个消息对话框显示“已有此图书”。第 52～65 行是建立对话框的函数实现，首先通过 Builder 类构成一个对话框，第 54 行设置对话框的标题，第 56～62 行设定对话框上按钮的事件处理，setNegativeButton 方法的第一个参数设定按钮的名称，第二个参数实现按钮的监听，当单击“返回首页”按钮时执行第 59 行的 finish()方法将当前 Activity 移出 Activity 栈。第 63 行定义取消按钮的名称，第 64 行调用 bulider 的 show()方法显示对话框。这里已经对 InsertActivity 界面做了详细的讲解。

下面讲解一下用 BookList 来模拟的存储数据的类，代码如下所示：

```
1.  package model.cqupt;
2.
3.  import java.util.ArrayList;
4.
5.  public class BookList extends ArrayList<Book> {
6.
7.      private static BookList booklist = null;
8.
9.      private BookList() {
10.         Book b1 = new Book("001", "c++", "24");
11.         Book b2 = new Book("002", "java", "45");
12.         Book b3 = new Book("003", "计算机", "21");
13.         add(b1);
14.         add(b2);
15.         add(b3);
16.     }
17.
18.     public static BookList getBookList() {
19.         if (booklist == null)
20.             booklist = new BookList();
21.         return booklist;
22.
23.     }
24. }
```

BookList 类采用了单例模式。单例模式也叫单子模式，是一种常用的软件设计模式。在应用这个模式时，单例对象的类必须保证只有一个实例存在。许多时候整个系统只需

要拥有一个全局对象，这样有利于协调系统整体的行为。Chapter03 中只需要一个 booklist 对象用于存取数据，所以采用了单例模式。BookList 类继承了 ArrayList，并且声明一个私有的静态对象作为整个程序的全局对象。在第 18～23 行是一个静态的成员方法，外部可以直接通过类名调用，这个函数的功能是得到 booklist 对象，如果对象没有被创建过，那么就调用构造函数初始化，然后再返回 booklist 对象，如果已经创建了，就会直接返回 booklist 对象。booklist 对象在第 9～16 行的私有构造函数中初始化，私有的目的是防止外部调用构造函数，由于 BookList 继承了 ArrayList，所以可以直接调用 ArrayList 的 add 方法存入 3 本图书，这里将图书的信息封装到了 model.cqupt 包下的 Book 类中，Book 类代码如下所示：

```
1.  package model.cqupt;
2.
3.  public class Book {
4.
5.      private String name;
6.      private String id;
7.      private String price;
8.
9.      public Book(String id, String name, String price) {
10.         this.id = id;
11.         this.name = name;
12.         this.price = price;
13.     }
14.
15.     public String getName() {
16.         return name;
17.     }
18.
19.     public void setName(String name) {
20.         this.name = name;
21.     }
22.
23.     public String getId() {
24.         return id;
25.     }
26.
27.     public void setId(String id) {
28.         this.id = id;
29.     }
30.
31.     public String getPrice() {
32.         return price;
33.     }
```

```
34.
35.     public void setPrice(String price) {
36.         this.price = price;
37.     }
38.
39.     public String toString() {
40.         return id + "  " + name + "  " + price;
41.     };
42.
43. }
```

在 Book 类中有 set 和 get 函数分别设置和得到图书的信息。

Activity 界面类通过对 Controller 类的操控来对 BookList 进行操作。下面详细讲解 Controller 类。Controller 类的代码如下所示：

```
1.  package control.cqupt;
2.
3.  import model.cqupt.Book;
4.  import model.cqupt.BookList;
5.
6.  public class Controller {
7.      public boolean addBook(String id,String name,String price) {
8.          BookList bookList = BookList.getBookList();
9.          int i = 0;
10.         for (; i < bookList.size(); ++i) {
11.             Book book2 = bookList.get(i);
12.             String bid = book2.getId();
13.             if (bid.equals(id)) {
14.                 break;
15.             }
16.         }
17.         if (i == bookList.size()) {
18.             Book book = new Book(id, name, price);
19.             bookList.add(book);
20.             return true;
21.
22.         }
23.         return false;
24.     }
25.
26.     public BookList searchBook() {
27.         BookList bookList = BookList.getBookList();
28.         return bookList;
29.     }
```

```
30.
31.     public boolean deleteBook(String name)
32.     {
33.         BookList bookList = BookList.getBookList();
34.         for (int i=0; i<bookList.size(); ++i)
35.         {
36.             Book book2 = bookList.get(i);
37.             if(book2.getName().equals(name))
38.             {
39.                 bookList.remove(i);
40.                 return true;
41.             }
42.         }
43.         return false;
44.     }
45.     public boolean setBook(String id,String name,String price)
46.     {
47.         BookList bookList = BookList.getBookList();
48.         for(int i=0;i<bookList.size();++i)
49.         {
50.             Book book2 = bookList.get(i);
51.             if(book2.getId().equals(id))
52.             {
53.                 Book book = new Book(id, name, price);
54.                 bookList.set(i,book);
55.                 return true;
56.             }
57.         }
58.         return false;
59.
60.     }
61. }
```

界面类通过 Controller 类操作 BookList。前面已经讲到了视图类中的 InsertActivity 和模型类中的 Booklist，这里将详细讲述控制类 Controller。Controller 在 Controller 中有 4 个方法，分别对 BookList 进行增、删、改、查操作。

增加图书（addBook），代码中第 7～24 行自定义了增加图书的方法，首先获得 booklist 对象，在第 9～23 行对 booklist 对象中图书进行循环遍历，如果在 booklist 对象中有与新增图书编号一样的图书，则跳出循环，并且返回 false，代表新增图书失败；如果 booklist 没有相同编号的图书，则执行第 17～22 行，对 booklist 操作增加图书，并且返回 true。

删除图书（deleteBook），代码第 31～34 行自定义了删除图书方法，与增加图书相同，首先获得 booklist 对象，在第 33～42 行循环遍历 booklist 对象中的图书名称，查看是否有需要删除的图书名称，如果有就返回 true，否则返回 false。

修改图书（setBook），代码中第 45～60 行自定义了修改图书的方法，首先获得 booklist 对象，然后在第 48～57 行循环遍历 booklist 对象中图书的编号，如果有此编号，那么就执行第 53～55 行对 booklist 对象进行修改操作，然后返回 true。如果没有这个编号的图书，则返回 false。

查询图书（searchBook），代码第 26～29 行自定义了查询图书的方法，与以上方法不同的是，查询图书只需要获得 booklist 对象，然后将此对象返回给 SearchActivity 界面，在 SearchActivity 界面中显示 booklist 对象中的图书，关于 SearchActivity 将在后面进行详细讲述。

下面将对剩下的 3 个视图类 DeleteActivity、SetActivity、SelectActivity 进行讲解。

首先讲解 DeleteActivity，界面如图 3-50 所示。

图 3-50　DeleteActivity

代码如下所示：

```
1.  package ui.cqupt;
2.
3.  import control.cqupt.Controller;
4.  import android.app.Activity;
5.  import android.app.AlertDialog.Builder;
6.  import android.content.DialogInterface;
7.  import android.os.Bundle;
8.  import android.view.View;
9.  import android.view.View.OnClickListener;
10. import android.widget.Button;
11. import android.widget.EditText;
12.
13. public class DeleteActivity extends Activity {
```

```
14.     private EditText name;
15.
16.     public void onCreate(Bundle savedInstanceState) {
17.         super.onCreate(savedInstanceState);
18.         setContentView(R.layout.delete);
19.         name = (EditText) findViewById(R.id.dname);
20.         Button delete = (Button) findViewById(R.id.d_delete);
21.         delete.setOnClickListener(new ButtonListener());
22.     }
23.
24.     class ButtonListener implements OnClickListener {
25.
26.         public void onClick(View v) {
27.             String bookname = name.getText().toString();
28.             Controller control = new Controller();
29.             if(bookname.equals(""))
30.             {
31.                 new Builder(DeleteActivity.this).setMessage("图书名
                    不能为空").show();
32.             }
33.             else
34.             {
35.             if (control.deleteBook(bookname)) {
36.                 name.setText("");
37.                 buildDialog();
38.             } else {
39.                 new Builder(DeleteActivity.this).setMessage("没有此
                    图书").show();
40.             }
41.             }
42.         }
43.
44.         private void buildDialog() {
45.             Builder builder = new Builder(DeleteActivity.this);
46.             builder.setTitle("删除成功,是否继续删除图书");
47.             builder.setNegativeButton("返回首页",
48.                     new DialogInterface.OnClickListener() {
49.                         public void onClick(DialogInterface dialog,
50.                                 int whichButton) {
51.                             finish();
52.                         }
53.
54.                     });
55.             builder.setPositiveButton("继续删除", null);
```

```
56.             builder.show();
57.         }
58.     }
59. }
```

DeleteActivity 继承了 Activity 类，并且覆写了 onCreate 方法初始化界面，通过 setContentView 方法设置了界面的布局，界面布局文件是 delete.xml 文件，代码如下所示：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:tools="http://schemas.android.com/tools"
4.      android:layout_width="fill_parent"
5.      android:layout_height="fill_parent"
6.      android:gravity="center|top"
7.      android:orientation="vertical"
8.      tools:ignore="HardcodedText" >
9.
10.     <TextView
11.         android:layout_width="fill_parent"
12.         android:layout_height="wrap_content"
13.         android:text="输入删除的书名"
14.         android:gravity="center_horizontal"
15.         android:textSize="15pt" />
16.
17.     <TextView
18.         android:layout_width="wrap_content"
19.         android:layout_height="wrap_content"
20.         android:text="图书名称"/>
21.
22.     <EditText
23.         android:id="@+id/dname"
24.         android:layout_width="180dp"
25.         android:layout_height="wrap_content"
26.         android:background="#FFFFFF"
27.         android:ems="10"
28.         android:inputType="text"
29.         android:textColor="#000000" />
30.
31.     <Button
32.         android:id="@+id/d_delete"
33.         android:layout_width="102dp"
34.         android:layout_height="wrap_content"
35.         android:text="删除" />
```

```
36.
37. </LinearLayout>
```

在 delete.xml 文件中声明了 TextView、EditText 和 Button 组件，整个代码的风格与前面所讲的 insert.xml 文件相同，此处不再赘述。

现在回到之前的 DeleteActivity 代码上，onCreate 方法对在 delete.xml 文件中声明的组件实例化，并且在第 21 行为 Button 组件加上事件监听。第 24～58 行定义了一个 ButtonListener 的内部类来实现监听器，并且实现了 public void onClick(View v)方法进行事件处理。onClick 方法的实现和 InsertActivity 中的实现基本相同，此处不再赘述。

接下来是 SetActivity，界面如图 3-51 所示。

图 3-51 SetActivity

可以看到 SetActivity 的界面和 InsertActivity 的界面基本上一样（除了显示的文字不同），由此可知，SetActivity 的实现和 InsertActivity 的实现基本上相同。SetActivity 代码如下所示：

```
1.  package ui.cqupt;
2.
3.  import control.cqupt.Controller;
4.  import android.app.Activity;
5.  import android.app.AlertDialog.Builder;
6.  import android.content.DialogInterface;
7.  import android.os.Bundle;
8.  import android.view.View;
9.  import android.view.View.OnClickListener;
10. import android.widget.Button;
```

```
11. import android.widget.EditText;
12.
13. public class SetActivity extends Activity {
14.     private EditText name;
15.     private EditText id;
16.     private EditText price;
17.
18.     public void onCreate(Bundle savedInstanceState) {
19.         super.onCreate(savedInstanceState);
20.         setContentView(R.layout.set);
21.         name = (EditText) findViewById(R.id.sname);
22.         id = (EditText) findViewById(R.id.sid);
23.         price = (EditText) findViewById(R.id.sprice);
24.         Button set = (Button) findViewById(R.id.s_set);
25.         set.setOnClickListener(new ButtonListener());
26.     }
27.
28.     class ButtonListener implements OnClickListener {
29.
30.         public void onClick(View v) {
31.             String bookname = name.getText().toString();
32.             String bookid = id.getText().toString();
33.             String bookprice = price.getText().toString();
34.             Controller control = new Controller();
35.             if (bookname.equals("") || bookid.equals("")
36.                     || bookprice.equals("")) {
37.                 new Builder(SetActivity.this).setMessage("图书信息不
                    能为空").show();
38.             } else {
39.                 if (control.setBook(bookid, bookname, bookprice)) {
40.                     name.setText("");
41.                     id.setText("");
42.                     price.setText("");
43.                     buildDialog();
44.                 } else {
45.             new Builder(SetActivity.this).setMessage("没有此编号的图
                书,请重新输入")
46.                             .show();
47.
48.                 }
49.             }
50.         }
51.
52.         private void buildDialog() {
```

```
53.             Builder builder = new Builder(SetActivity.this);
54.             builder.setTitle("修改成功,是否继续修改图书");
55.             builder.setNegativeButton("返回首页",
56.                     new DialogInterface.OnClickListener() {
57.                         public void onClick(DialogInterface dialog,
58.                                 int whichButton) {
59.                             finish();
60.                         }
61.
62.                     });
63.             builder.setPositiveButton("继续修改", null);
64.             builder.show();
65.         }
66.     }
67.
68. }
```

可以对比一下 InsertActivity 的代码，两者不同的是各个组件的名称和通过 control 对象对 booklist 进行的操作不同，其他功能完全一样。SetActivity 的布局文件是 set.xml 文件，代码如下所示：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:tools="http://schemas.android.com/tools"
4.      android:layout_width="fill_parent"
5.      android:layout_height="fill_parent"
6.      android:gravity="center|top"
7.      android:orientation="vertical"
8.      tools:ignore="HardcodedText" >
9.
10.     <TextView
11.         android:layout_width="fill_parent"
12.         android:layout_height="wrap_content"
13.         android:gravity="center_horizontal"
14.         android:text="请输入图书信息"
15.         android:textSize="15pt" />
16.
17.     <TextView
18.         android:layout_width="wrap_content"
19.         android:layout_height="wrap_content"
20.         android:text="图书编号" />
21.
22.     <EditText
23.         android:id="@+id/sid"
```

```
24.         android:layout_width="180dp"
25.         android:layout_height="wrap_content"
26.         android:background="#FFFFFF"
27.         android:ems="10"
28.         android:inputType="text"
29.         android:textColor="#000000" />
30.
31.     <TextView
32.         android:layout_width="wrap_content"
33.         android:layout_height="wrap_content"
34.         android:text="修改名称" />
35.
36.     <EditText
37.         android:id="@+id/sname"
38.         android:layout_width="180dp"
39.         android:layout_height="wrap_content"
40.         android:background="#FFFFFF"
41.         android:ems="10"
42.         android:inputType="text"
43.         android:textColor="#000000" />
44.
45.     <TextView
46.         android:layout_width="wrap_content"
47.         android:layout_height="wrap_content"
48.         android:text="修改价格" />
49.
50.     <EditText
51.         android:id="@+id/sprice"
52.         android:layout_width="180dp"
53.         android:layout_height="wrap_content"
54.         android:background="#FFFFFF"
55.         android:ems="10"
56.         android:inputType="text"
57.         android:textColor="#000000" />
58.
59.     <Button
60.         android:id="@+id/s_set"
61.         android:layout_width="102dp"
62.         android:layout_height="wrap_content"
63.         android:text="修改" />
64.
65. </LinearLayout>
```

set.xml 和 insert.xml 基本相同，前面已经对 insert.xml 进行了详细讲解，这里就不再

重复了，如果对 set.xml 代码还不是很清楚，那么请回顾一下对 insert.xml 的讲解。

接下来要讲解最后一个界面 SelectActivity。这个界面和 InsertActivity、DeleteActivity 有点不同，我们将会做详细的讲述。下面先看一下 SelectActivity 的界面，如图 3-52 所示。

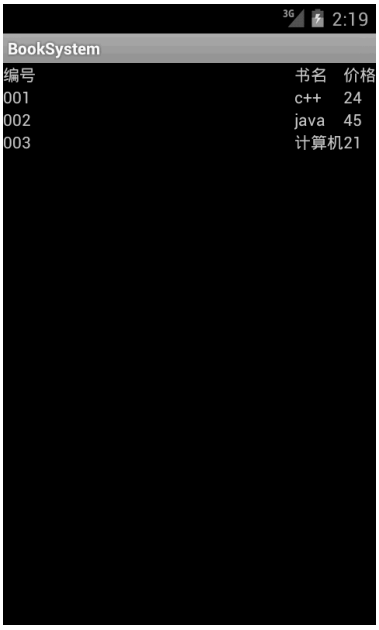

图 3-52 SelectActivity

SelectActivity 的代码如下所示：

```
1.  package ui.cqupt;
2.
3.  import control.cqupt.Controller;
4.
5.  import model.cqupt.Book;
6.  import model.cqupt.BookList;
7.
8.  import ui.cqupt.R;
9.
10. import android.app.Activity;
11. import android.os.Bundle;
12. import android.widget.TableLayout;
13. import android.widget.TableRow;
14. import android.widget.TextView;
15.
16. public class SelectActivity extends Activity {
17.
```

```
18.     public void onCreate(Bundle savedInstanceState) {
19.         super.onCreate(savedInstanceState);
20.         setContentView(R.layout.select);
21.         Controller control = new Controller();
22.         BookList booklist = control.searchBook();
23.         CreateTable(booklist);
24.     }
25.
26.     private void CreateTable(BookList booklist) {
27.      TableLayout table = (TableLayout) findViewById(R.id.SELECT_
         ACTIVITY_TableLayout);
28.         for (int i = 0; i < booklist.size(); ++i) {
29.             Book book = booklist.get(i);
30.             String id = book.getId();
31.             String name = book.getName();
32.             String price = book.getPrice();
33.             TableRow row = new TableRow(this);
34.             TextView tid = new TextView(this);
35.             TextView tname = new TextView(this);
36.             TextView tprice = new TextView(this);
37.             tid.setText(id);
38.             tname.setText(name);
39.             tprice.setText(price);
40.             row.addView(tid);
41.             row.addView(tname);
42.             row.addView(tprice);
43.             table.addView(row);
44.         }
45.     }
46. }
```

SelectActivity 和之前的界面不同点是在布局上，SelectActivity 采用了线性布局和表格布局，并且通过表格的形式显示图书。下面是 select.xml 文件代码：

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.      xmlns:tools="http://schemas.android.com/tools"
4.      android:layout_width="fill_parent"
5.      android:layout_height="fill_parent"
6.      android:orientation="vertical"
7.      tools:ignore="HardcodedText" >
8.
9.      <TableLayout
10.         android:id="@+id/SELECT_ACTIVITY_TableLayout"
11.         android:layout_width="fill_parent"
12.         android:layout_height="wrap_content"
```

```
13.        android:stretchColumns="0" >
14.
15.        <TableRow>
16.
17.            <TextView
18.                android:id="@+id/TextView01"
19.                android:layout_width="fill_parent"
20.                android:layout_height="wrap_content"
21.                android:text="编号" />
22.
23.            <TextView
24.                android:id="@+id/TextView02"
25.                android:layout_width="fill_parent"
26.                android:layout_height="wrap_content"
27.                android:text="书名" />
28.
29.            <TextView
30.                android:id="@+id/TextView03"
31.                android:layout_width="fill_parent"
32.                android:layout_height="wrap_content"
33.                android:text="价格" />
34.        </TableRow>
35.    </TableLayout>
36. </LinearLayout>
```

可以看到，上面代码的第 9～35 行是将所有的组件都放入<TableLayout></TableLayout>中，这是一个表格布局。表格布局中<TableRow></TableRow>标签，这个标签代表了表格的一行，每一行的内容都要在 TableRow 中显示，SelectActivity 的显示功能采用了动态增加组件的方式。SelectActivity 继承了 Activity 类并且覆写了 onCreate 方法，在第 22 行通过 control 的 searchBook 方法得到 booklist 对象的引用，将 booklist 对象作为参数传递给自定义创建表格的方法 CreateTable。第 26～46 行是 CreateTable 方法的实现，创建表格显示 booklist 对象中的图书。首先实例化 TableLayout，然后在第 28～44 行循环遍历 booklist，为每个 booklist 中的 book 创建一个 TableRow，代表表格的一行内容，第 30～32 行提取 booklist 中图书信息，在第 34～36 行为图书的编号、名称和价格创建显示的组件 TextView，然后把图书信息显示到 TextView 组件中，在第 40～42 行把显示组件 TextView 加入到代表表格中每行的 TableRow 中，最后将这一行加入到表格布局中，完成对图书信息的显示功能。

这里完成了对 Chapter03 的讲解，在以后的学习中，我们会在 Chapter03 的基础上进行修改，加入学习的新知识。