第 3 章

外观与感觉

3.1 事 件 处 理

由于安卓应用使用 Java 程序编写，在应用程序运行中，用户对移动终端键盘、屏幕和位置的操作都转化成事件对象，安卓系统通过对这些事件的捕获，执行相应的处理代码，实现与用户的交互，完成预定的功能。这个过程就是安卓的事件处理。安卓的事件处理机制有两种：基于监听接口和基于回调机制。这两种机制的原理和实现方法都有所不同。安卓的基于监听接口的事件处理机制，完全采用了 Java 的事件处理机制。

3.1.1 基于监听接口

安卓的事件处理机制是一种委派式事件处理方式，如图 3-1 所示。普通组件作为事件源将整个事件处理委托给特定的对象，即事件监听器：当该事件源发生指定的事件时，就通知所委托的事件监听器，由事件监听器来处理这个事件。每个组件均可以针对特定的事件指定一个事件监听器，每个事件监听器也可监听一个或多个事件源，因为同一个事件源上可能发生多种事件，委派式事件处理方式可以把事件源上所有可能发生的事件分别授权给不同的事件监听器来处理，同时也可以让一类事件都使用同一个事件监听器来处理。在事件处理的过程中主要涉及三个主要部分：事件源、事件和事件监听器。

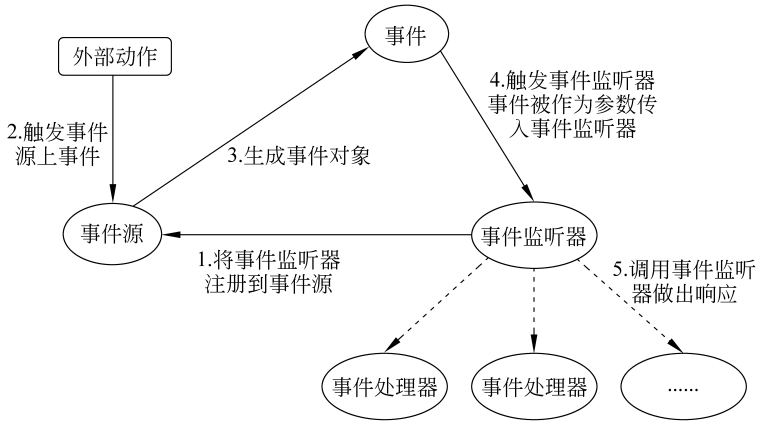

图 3-1 事件流程处理

(1) 事件源(Source)：事件源，是指触摸屏、键盘或位置传感器操作针对的控件或容器。事件发生时，也就是出现某个控件被触摸操作或移动终端位置移动，这个控件也就是事件源类

负责发送事件发生的通知，并通过事件源查找自己的事件监听者队列，并将事件信息通知队列中的监听者来完成，同时事件源还在得到有关监听者信息时负责维护自己的监听者队列。

（2）事件（Event）：事件是指对组件或容器的触摸屏、键盘或位置的一个操作用类描述，例如，键盘事件类描述键盘事件的所有信息：键按下、释放、双击、组合键以及键码等相关键的信息。

（3）事件监听器（Listener）：安卓的事件处理由事件监听器类和事件监听器接口来实现。事件发生后，事件源将相关的信息通知对应的监听器，事件源和监听器之间通过监听器接口完成这类信息交换。事件监听器类就是事件监听器接口的具体实现，事件发生后该主体负责进行相关的事件处理，同时还负责通知相关的事件源，自己关注它的特定事件，以便事件源在事件发生时能够通知该主体。

外部的操作，例如，按下按键、触摸屏幕、单击按钮或转动移动终端等动作，会触发事件源上的事件，对于单击按钮的操作来说事件源就是按钮，会根据这个操作生成一个按钮按下的事件对象，这对于系统来说就产生了一个事件。事件的产生会触发事件监听器，事件本身作为参数传入到事件处理器中。事件监听器是通过代码在程序初始化时注册到事件源的，也就是说，在按钮上设置一个可以监听按钮操作的监听器，并且通过这个监听器调用事件处理器，事件处理器针对这个事件所编写的代码，例如弹出一条信息，下面的示例讲解简单的事件处理模型。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center_horizontal"
android:orientation="vertical">

<EditText
android:id="@+id/txt"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:cursorVisible="false"
android:textSize="12pt" />

<Button
android:id="@+id/bn"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="单击" />
</LinearLayout>
```

代码 3-1　按钮将作为事件源

上面的程序定义的按钮将作为事件源（如代码 3-1 所示），接下来程序将会为该按钮绑定一个事件监听器，监听器类必须由开发者来实现。

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
```

代码 3-2　为该按钮绑定一个事件监听器

```
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button bn = (Button) findViewById(R.id.bn);
        bn.setOnClickListener(new MyClickListener());
    }
    class MyClickListener implements View.OnClickListener {
        @Override
        public void onClick(View v) {
            EditText txt = (EditText) findViewById(R.id.txt);
            txt.setText("按钮被单击");
        }
    }
}
```

代码 3-2　（续）

上面程序中的代码定义了一个 View.OnClickListener 实现类，这个实现类将会作为事件监听器使用。程序为按钮注册事件监听器，当按钮被单击时该处理器被触发。从上面的程序可以看出，基于监听器的事件处理模型的编程步骤如下。

（1）获取普通界面组件（事件源），也就是被监听的对象。

（2）实现事件监听器类，该监听器类是一个特殊的类，必须实现 XxxListener 接口。

（3）调用事件源的 setXxxListener（）方法将事件监听器对象注册给普通组件（事件源）。

当事件源上发生指定事件时，安卓会触发事件监听器，由事件监听器调用相应的方法（事件处理器）来处理事件。将代码 3-2 与图 3-1 结合起来看，事件源就是程序中的按钮，其实开发者不需要太多的额外处理，应用程序中任何组件都可作为事件源；事件监听器是程序中的 MyClickListener 类，监听器类必须由程序员负责实现，实现监听器类的关键就是实现处理器方法；注册监听器就是调用事件源的 setXxxListener（XxxListener）。所谓事件监听器，其实就是实现了特定接口类的实例，在程序中实现事件监听器，通常有以下几种形式。

（1）内部类形式：将事件监听器类定义成当前类的内部类。

（2）外部类形式：将事件监听器类定义成一个外部类。

（3）活动本身作为事件监听器类：让活动本身实现监听器接口，并实现事件处理方法。

（4）匿名内部类形式：使用匿名内部类创建事件监听器对象，是目前使用最广泛的事件监听器形式。

（5）直接绑定到布局标签：直接在界面布局文件中为指定标签绑定事件处理方法。

对于很多安卓界面组件标签而言都支持 onClick 属性，该属性的属性值就是一个方法名，定义了单击控件要执行的操作，示例如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center_horizontal"
android:orientation="vertical">

<EditText
android:id="@+id/txt"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:cursorVisible="false"
android:textSize="12pt"/>

<Button
android:id="@+id/bn"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="单击"
android:onClick="clickHandler"/>
</LinearLayout>
```

代码 3-3　onClick 属性

代码 3-3 在界面布局文件中为按钮绑定一个事件处理方法 clickHanlder()，这就意味着开发者需要在该界面布局对应的活动中定义一个 clickHandler()方法，该方法将会负责处理该按钮上的单击事件，对应的 Java 代码如下。

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void clickHandler(View source) {
        EditText show = (EditText) findViewById(R.id.txt);
        show.setText("按钮被单击");
    }
}
```

代码 3-4　clickHandler()方法

在安卓的开发中，对于单击事件的 OnClickListener 有下面三种实现方式，可以根据实际场景的需要选择合适的用法，下面以按钮来举例说明。

方法一：使用匿名类定义，代码如下。

```
Button bt_Demo = (Button)findViewById(R.id.bt_Demo);
bt_Demo.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //具体单击操作的逻辑
    }
});
```

代码 3-5　使用匿名类定义

方法二：使用外部类定义，代码如下。

```
Button bt_Demo = (Button)findViewById(R.id.bt_Demo);
bt_Demo.setOnClickListener(new ButtonListener());
private class ButtonListener implements OnClickListener{
    @Override
    public void onClick(View arg0) {
        //TODO Auto-generated method stub
        switch(arg0.getId()){
            case R.id.btn_Demo:
                //具体单击操作的逻辑
                break;
            default:
                break;
        }
    }
}
```

代码 3-6　使用外部类定义

方法三：在活动中实现 OnClickListener 接口，代码如下。

```
public class MyActivity extends Activity implements OnClickListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //按钮
        Button btn_Demo = (Button)findViewById(R.id.bt_Demo);
        bt_Demo.setOnClickListener(this);
    }
    @Override
    public void onClick(View v) {
        switch(v.getId()){
            case R.id.btn_Demo:
                //具体单击操作的逻辑
                break;
            default:
                break;
        }
    }
}
```

代码 3-7　在活动中实现 OnClickListener 接口

3.1.2 基于回调机制

安卓的另一种事件处理机制是回调机制。通常情况下，程序员写程序时需要使用系统工具类提供的方法来完成某种功能，但是某种情况下系统会反过来调用一些类的方法。例如，对于用作组件或插件的类则需要编写一些供系统调用的方法，这些专门用于被系统调用的方法被称为回调方法，也就是回过来系统调用的方法。

安卓平台中，每个视图都有自己的处理事件的回调方法，开发人员可以通过重写视图中的这些回调方法来实现需要的响应事件。当某个事件没有被任何一个视图处理时，便会调用活动中相应的回调方法。例如，有一个按钮按下的事件发生了，但编码过程中这个按钮并没有对这个事件做任何处理，所在的活动中的任何组件也并没有对这个事件做任何处理，这时系统会调用活动相应的回调方法 onKeyDown()。回调机制实质上就是将事件的处理绑定在组件上，由界面组件自己处理事件，回调机制需要自定义视图来实现，自定义视图重写事件处理方法就可以了，例如，活动和片段的生命周期中的各种状态发生变化时，调用的 onResume()等方法也是回调方法。

3.2 按 钮 控 件

安卓提供的按钮控件有很多种，包括基本的 Button、ImageButton、ToggleButton、CheckBox 和 RadioButton 都是按钮的类型。Button 类控件继承自 TextView，因此也具有 TextView 的宽和高设置、文字显示等一些基本属性。Button 类控件在应用程序中的定义，与其他图形控件一样，一般都在布局文件中进行定义、设置和布局设计。setText()和 getText()是 Button 类控件最常用的方法，用于设置和获取 Button 显示的文本。Button 类控件一般会与单击事件联系在一起。对于基本的 Button，可以采用两种方式处理单击事件。一种使用 Button 的 setOnClickListener()方法为其设置 OnClickListener，把具体的事件处理代码写在 onClick(View v)方法中；另一种在 XML 布局文件中，使用 android:OnClick 属性为 Button 指定单击事件发生时执行的方法。如果在 XML 布局文件中，使用 android:OnClick 属性指定了单击事件的回调方法，这个方法在 Java 应用程序中必须是 public 的，而且只有一个 View 类型的参数。在按钮类控件的使用过程中，属性设置和事件处理稍有不同，下面具体说明各按钮类控件如何对事件进行处理。在具体调试运行过程中，创建资源文件和活动的具体步骤与前面例子相同，请参考其编写完整的代码，运行查看效果。

3.2.1 按钮

按钮控件可以有文本或者图标，也可以文本和图标同时存在(如图 3-2 所示)，当用户触摸时就会触发事件。

图 3-2 各种按钮

根据按钮控件的组成方式，创建按钮控件有如下三种方式。

(1) 如果由文本组成，使用 Button 类创建，如下。

```
<Button
android:id="@+id/ccbtn1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Basic Button" />
```

代码 3-8　由文本组成，使用 Button 类创建

(2) 如果由图标组成，使用 ImageButton 类创建，显示一个带有可以单击图像的按钮。默认情况下，ImageButton 看起来像一个常规的 Button，具有在不同按钮状态期间更改颜色的标准按钮背景。按钮表面上的图像由 XML 元素＜ImageButton＞中的 android:src 属性或 ImageView.setImageResource(int)方法定义。要删除标准按钮背景图像，可定义自己的背景图像或将背景颜色设置为透明，要指示不同的按钮状态，如聚焦、选中等，可以为每个状态定义不同的图像。例如，默认为红色图像，聚焦时为橙色，按下时为蓝色。一个简单的方法是使用 XML 绘制选择器，代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
<item android:drawable="@drawable/button_pressed" android:state_pressed=
"true" /><!-- pressed -->
<item android:drawable="@drawable/button_normal" /><!-- default -->
</selector>
```

代码 3-9　由图标组成，使用 ImageButton 类创建

StateListDrawable 是一个 Drawable 对象，使用不同的图像来表示同一对象，具体取决于对象所处的状态。例如，按钮可以以多种状态之一存在(按下、聚焦、悬停或不存在)，可以为每个状态提供不同的背景图像。在 XML 文件中描述状态列表。每个图形都由＜selector＞元素中的一个＜item＞元素表示，每个＜item＞都使用一个“state_属性”来指示使用图形的情况。在每次状态变化期间，都会从上到下遍历状态列表，使用与当前状态匹配的第一项，这意味着选择不是基于最佳匹配，而仅仅是满足状态最低标准的第一项。代码 3-9 中的状态列表定义了按钮当处于不同状态时显示的图像，当按钮按下时即当 state_pressed＝"true"时，显示一个名为 button_pressed 的图像；当按钮处于初始状态时显示一个名为 button_normal 的图像。会根据按钮的状态和 XML 中定义的相应图像自动更改图像。元素的＜item＞顺序很重要，因为是按顺序计算的，这就是为什么 button_normal 按钮图像最后出现的原因，因为只会在 android:state_pressed 并且 android:state_focused 都被评估为 false 之后应用。

```
<ImageButton
android:id="@+id/ccbtn2"
android:layout_width="80dp"
android:layout_height="45dp"
android:layout_centerInParent="true"
android:background="@drawable/imagebuttonselector" />
```

代码 3-10　在 android:background 属性中引用

将 XML 文件保存在项目 res\drawable\文件夹中，然后将其作为 ImageButton 源的可绘制对象引用，放在 android:background 属性中（如代码 3-10 所示）。

（3）如果文本和图标都有，使用 Button 类的 android:drawableLeft 属性，代码如下。

```
<Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/button_text"
   android:drawableLeft="@drawable/button_icon"
    ... />
```

代码 3-11　使用 Button 类的 android:drawableLeft 属性

除了按钮上的文本和图标，按钮的外观（如背景图像和字体）可能会因为设备或者安卓版本的不同而有所不同，随着安卓版本的升级，其界面的样式也发生变化，而厂家也会定制输入控件的默认样式。如果要控制控件使用适用于整个应用程序的样式，例如，要确保所有运行安卓 4.0 甚至更高版本的设备在应用程序中使用 Holo 主题，需要在 AndroidManifest.xml 文件中声明 android:theme="@android:style/Theme.Holo"。在 XML 布局文件中，可以使用 Button 的一些属性来定义按钮的外观。定制不同的背景，可以指定＜android:background＞属性为绘图或颜色，也可以是自定义的背景。其他的属性，例如字体、大小、边框等，可以参照 TextView 和 View 的 XML 属性。下面是一个简单的例子，使用了一种无边框按钮。无边框按钮与基本按钮相似，但是无边框按钮没有边框或背景，但在不同状态如单击时会改变外观。如需创建无边框按钮，对按钮应用 borderlessButtonStyle 样式，如代码 3-12 所示。

```
<Button
   android:id="@+id/button_send"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/button_send"
   android:onClick="sendMessage"
        style="?android:attr/borderlessButtonStyle" />
```

代码 3-12　按钮外观设置

3.2.2　单选按钮

单选按钮就是 RadioButton（如图 3-3 所示），在安卓开发中应用非常广泛。RadioButton 的外形是单个圆形的单选框，具有选择或不选择两种状态。在 RadioButton 没有被选中时，用户能够单击选中它。与复选框不同的是，用户一旦选中它就不能够取消。

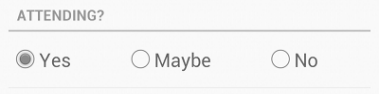

图 3-3　单选按钮

一般来说，实现单选按钮需要由 RadioButton 和 RadioGroup 配合使用。RadioGroup 是单选组合框，可以容纳多个 RadioButton 的容器。在没有 RadioGroup 的情况下，RadioButton 可以全部都选中；当多个 RadioButton 被 RadioGroup 包含的情况下，RadioButton 只可以选择一

个。RadioButton 的事件处理，可以使用 setOnCheckedChangeListener()方法注册单选按钮的监听器，也可以采用在 XML 布局文件中指定处理方法的方式。下面这个例子，在 XML 布局文件中定义了一个具有四个 RadioButton 的 RadioGroup，一个文本显示框 TextView 控件和一个按钮 Button 控件，见代码 3-13。当一个 RadioButton 被选中时，在 TextView 控件中显示选择项的文本，如果单击按钮，将清除选中的项目。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >

<RadioGroup
android:id="@+id/menu"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:checkedButton="@+id/lunch"
android:orientation="vertical" >

<RadioButton
android:id="@+id/breakfast"
android:text="@string/radio_group_1_breakfast" />

<RadioButton
android:id="@id/lunch"
android:text="@string/radio_group_1_lunch" />

<RadioButton
android:id="@+id/dinner"
android:text="@string/radio_group_1_dinner" />

<RadioButton
android:id="@+id/all"
android:text="@string/radio_group_1_all" />

<TextView
android:id="@+id/choice"
android:text="@string/radio_group_1_selection" />
</RadioGroup>

<Button
android:id="@+id/clear"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/radio_group_1_clear" />

</LinearLayout>
```

代码 3-13　radiobutton_layout.xml

在这个例子中没有指定事件处理的方法，因此在 Java 应用程序中，采用控件相对应的两个事件监听器 RadioGroup.OnCheckedChangeListener 和 View.OnClickListener 来处理

对 RadioGroup 和 RadioButton 的事件，具体的事件处理代码写在 onCheckedChanged()和 onClick()接口方法中，分别实现根据选项更新 TextView 的显示和清除 RadioButton 选中的功能，见代码 3-14。

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

import com.example.ch03.materialdesign.R;

public class RadioGroupDemoActivity extends AppCompatActivity implements
RadioGroup.OnCheckedChangeListener, View.OnClickListener {
    private TextViewmChoice;
    private RadioGroupmRadioGroup;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.radio_group);
        mRadioGroup = (RadioGroup) findViewById(R.id.menu);
        //test adding a radio button programmatically
        RadioButtonnewRadioButton = new RadioButton(this);
        newRadioButton.setText(R.string.radio_group_snack);
        newRadioButton.setId(R.id.snack);
        LinearLayout.LayoutParamslayoutParams = new RadioGroup.LayoutParams(
        RadioGroup.LayoutParams.WRAP_CONTENT,
        RadioGroup.LayoutParams.WRAP_CONTENT);
        mRadioGroup.addView(newRadioButton, 0, layoutParams);
        //test listening to checked change events
        String selection = getString(R.string.radio_group_selection);
        mRadioGroup.setOnCheckedChangeListener(this);
        RadioButtondefauld = (RadioButton) findViewById(mRadioGroup
        .getCheckedRadioButtonId());
        mChoice = (TextView) findViewById(R.id.choice);
        mChoice.setText(selection + defauld.getText());
        //test clearing the selection
        Button clearButton = (Button) findViewById(R.id.clear);
        clearButton.setOnClickListener(this);
    }
    public void onCheckedChanged(RadioGroup group, int checkedId) {
        String selection = getString(R.string.radio_group_selection);
        RadioButton checked = (RadioButton) findViewById(checkedId);
        String none = getString(R.string.radio_group_none);
        mChoice.setText(selection
                + (checkedId == View.NO_ID ? none : checked.getText()));
    }
```

代码 3-14　RadioGroupDemoActivity.java

```
    public void onClick(View v) {
        mRadioGroup.clearCheck();
    }
}
```

代码 3-14　（续）

完成应用程序编码后，同样不要忘记了要到 AndroidManifest.xml 中注册才能运行，效果如图 3-4 所示。从上面的例子可以看出，安卓控件的事件处理方法与一般的 Java 图形界面处理类似，只是控件和监听器有所不同，所采用的事件处理机制和原理以及实现步骤都基本相同。

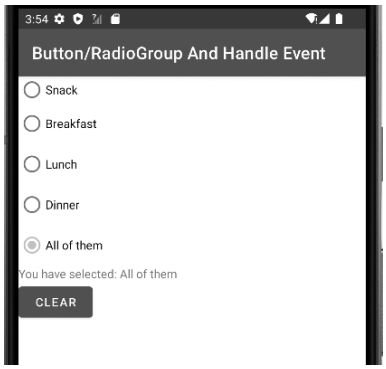

图 3-4　RadioGroupDemoActivity

3.2.3　复选框

复选框就是 CheckBox，具备选中和未选中两种状态。复选框的外形是矩形框，可以通过单击选中或取消选中。在进行事件处理时，应用程序可以根据是否被选中来进行相应的操作，并且对复选框加载事件监听器，来对控件状态的改变做出响应。下面这个例子，通过 XML 布局文件在用户界面使用 CheckBox 控件来创建一个复选框，实现当复选框被单击时，弹出一个文本消息显示复选框的当前状态。

（1）创建 XML 布局文件 checkbox_layout.xml，定义三个 CheckBox 控件，并在其中使用 android:onClick 属性指定事件处理的方法名为 doClick，见代码 3-15。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical" >

<CheckBox
android:id="@+id/chickenCB"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:checked="true"
android:text="Chicken" />

<CheckBox
android:id="@+id/fishCB"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Fish" />

<CheckBox
android:id="@+id/steakCB"
```

代码 3-15　checkbox_01.xml

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:onClick="doClick"
    android:text="Steak" />

    </LinearLayout>
```

代码 3-15 （续）

（2）创建新类 CheckBoxDemoActivity，实现 XML 文件中指定的，单击 CheckBox 控件后的事件处理方法 doClick（View v），见代码 3-16。

```
import android.os.Bundle;
import android.view.View;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import com.example.ch03.materialdesign.R;
public class CheckBoxDemoActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.checkbox_01);
        //No handling in here for the Chicken checkbox
        CheckBoxfishCB = (CheckBox) findViewById(R.id.fishCB);
        if (fishCB.isChecked())
            fishCB.toggle(); //flips the checkbox to unchecked if it was
            //checked
            fishCB.setOnCheckedChangeListener(
                    new CompoundButton.OnCheckedChangeListener() {
            public void onCheckedChanged(CompoundButton arg0, booleanisChecked) {
                Toast.makeText(
                        CheckBoxDemoActivity.this,
                        "The fish checkbox is now "
                        + (isChecked ?"checked" : "not checked"),
                        Toast.LENGTH_SHORT).show();
            }
        });
    }
    public void doClick(View view) {
        Toast.makeText(
                this,
                "The steak checkbox is now "
                        + (((CheckBox) view).isChecked() ?"checked"
                        : "not checked"), Toast.LENGTH_SHORT).show();
    }
}
```

代码 3-16 CheckBoxDemoActivity.java

完成应用程序编码后，不要忘记需要到 AndroidManifest.xml 中注册才能运行，效果如

图 3-5 所示。从上面的例子可以看出，事件处理方法可以采用与 Button 相同的模式，只是在处理过程中，可以针对 CheckBox 不同的状态进行不同的编码，实现不同的功能。也会触发 OnCheckedChange 事件，可以对应地使用 OnCheckedChangeListener 监听器来监听这个事件，重写其中的 onCheckedChanged()方法，使用 setOnCheckedChangeListener()方法设置监听器。

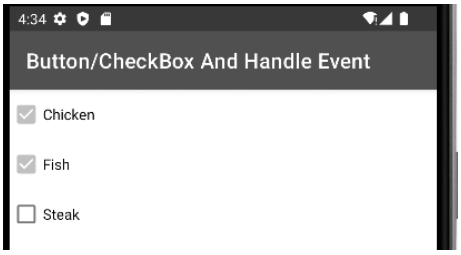

图 3-5　CheckBox

3.2.4　切换按钮

如果设置选项只有两种状态，可以使用开关按钮 ToggleButton（见图 3-6(a)）。安卓 4.0（API 级别 14）提供了另外一种叫作 Switch 的开关按钮，这个按钮提供一个滑动控件，可以通过添加 Switch 对象来实现（见图 3-6(b)）。

(a)　　(b)

图 3-6　ToggleButton 和 Switch

ToggleButton 和 Switch 控件都是 CompoundButton 组合按钮的子类并且有着相同的功能，所以可以用同样的方法来实现它们的功能。当用户选择 ToggleButtons 和 Switch 时，对象就会接收到相应的单击事件。要定义这个单击事件的响应操作，添加 android:onClick 属性到 XML 布局文件的开关按钮控件中去。例如，代码 3-17 定义了一个 ToggleButton 开关按钮并且设置了 android:onClick 事件单击响应属性。

```
<ToggleButton
   android:id="@+id/togglebutton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textOn="Vibrate on"
   android:textOff="Vibrate off"
android:onClick="onToggleClicked"/>
```

代码 3-17　在布局文件中定义 ToggleButton

在这个布局对应的活动里，在 android:onClick 指定的 onToggleClicked()方法中，定义事件处理代码（见代码 3-18）。

```
public void onToggleClicked(View view) {
    //Is the toggle on?
    boolean on = ((ToggleButton) view).isChecked();
    if (on) {
        //Enable vibrate
    } else {
        //Disable vibrate
    }
}
```

代码 3-18　onToggleClicked 事件处理

与其他图形控件一样，除了在布局文件中定义之外，也可以通过代码的方式，为控件注册一个事件监听器。代码 3-19 中说明了为 ToggleButton 注册监听器的具体实现代码：首先创建一个 CompoundButton.OnCheckedChangeListener 对象，覆盖 OnCheckedChangeListener 接口的抽象方法 onCheckedChanged()，在其中具体实现单击 ToggleButton 对象后的事件处理，然后通过调用此 ToggleButton 对象的 setOnCheckedChangeListener()方法，将监听器绑定到按钮上，见代码 3-19。

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButtonbuttonView, booleanisChecked) {
        if (isChecked) {
            //The toggle is enabled
        } else {
            //The toggle is disabled
        }
    }
});
```

代码 3-19　为 ToggleButton 注册事件监听器

完整的应用程序可以参考 CheckBox 和 RadioButton 编写，Switch 按钮代码如下。

```
<LinearLayout
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical" >

<Switch
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginBottom="32dip"
android:text="Standard switch" />

<Switch
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

代码 3-20　Switch 按钮

```
    android:layout_marginBottom="32dip"
    android:checked="true"
    android:text="Default is on" />

    <Switch
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="32dip"
    android:text="Customized text"
    android:textOff="Stop"
    android:textOn="Start" />
    </LinearLayout>
```

代码 3-20　（续）

3.3　提 示 控 件

提示控件是指 Toast，是在窗口表面弹出的一个简短的小消息，只填充消息所需要的空间，并且用户当前的活动依然保持可见性和交互性。这种通知可自动地淡入淡出，且不接受用户的交互事件。例如，如果用户正在编写一封邮件，需要接通一个电话，这时界面会弹出一个提示，将邮件保存为草稿，如图 3-7 所示。

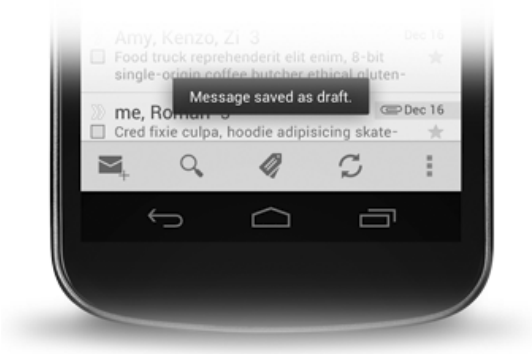

图 3-7　Toast 显示

Toast 是一个在屏幕上显示片刻的提示消息，但是 Toast 不能获得焦点，不能够与用户进行交互。可以自定义包括图像的 Toast 布局文件。Toast 通知能够被活动或 Service 创建并显示。如果创建了一个源自 Service 的 Toast 通知，它会显示在当前活动的最上层。如果用户需要对通知做出响应，可以考虑使用安卓的另一种视图对象状态栏通知（Status Bar Notification），这会在后面的章节中介绍。

如果要使用 Toast，可以直接用 Toast 类的方法 Toast.makeText()实例化一个 Toast 对象。这个方法有三个参数，分别为 Context、要显示的文本消息和 Toast 通知持续显示的时间。Toast.makeText()方法会返回一个按参数设置且被初始化的 Toast 对象，Toast 对象的内容用 show()方法显示。代码 3-21 示例了在活动的 onCreate()方法中如何创建和显示 Toast 信息，在其他视图中实现的代码类似。

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

代码 3-21　显示 Toast 通知

代码 3-21 中最后两行代码也可以用链式组合方法写且避免创建 Toast 对象，代码如下。

```
Toast.makeText(context, text, duration).show();
```

代码 3-22　链式组合方法

标准的 Toast 通知水平居中显示在屏幕底部附近。如果要把 Toast 通知放到不同的位置显示，可以使用布局文件来设置 Toast 对象的具体布局，然后在活动加载布局文件后，通过 ID 获取 Toast 对象，使用 show()方法显示其文本消息。下面使用一个简单的例子，来说明如何定义和使用 Toast 自定义的布局文件。要创建一个自定义的布局文件，可以在 XML 布局文件或程序代码中定义一个 View 布局，然后把 View 对象传递给 setView()方法。代码 3-23 中 layout.custom_toast.xml 布局文件是专门为 Toast 对象的布局所做的定义，其中，android:id="@+id/toast_layout_root"定义了这个 Toast 布局的 id。这个是一个包含一个图形和一个文本框的布局，其背景、对齐方式和文本颜色也进行了设置。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/toast_layout_id"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="#000"
android:orientation="horizontal"
android:padding="5dp" >

<ImageView
android:id="@+id/image"
android:src="@drawable/ic_launcher_foreground"
android:layout_width="wrap_content"
android:layout_height="fill_parent"
android:contentDescription="@string/image_content"
android:layout_marginRight="5dp" />

<TextView
android:id="@+id/text"
android:layout_width="wrap_content"
android:layout_height="fill_parent"
android:textColor="#FFF"/>

</LinearLayout>
```

代码 3-23　Toast 自定义布局

Toast 的布局文件创建完成后，需要把这个布局应用到用户界面的 Toast 对象。在应用程序活动的 onCreate()中，首先导入活动的布局资源文件，然后需要使用 LayoutInflater 的对象，通过其 inflate()方法，利用布局文件名和布局的 ID 来获取布局文件中定义的布局，下一步使用 Toast 对象的 setView()方法使用这个布局，如代码 3-24 所示。

```
import android.os.Bundle;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import com.example.ch03.materialdesign.R;
public class ToastCustomActivity extends AppCompatActivity {
    private Button button;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.toast);
        button = (Button) findViewById(R.id.mainbutton);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                //get your toast.xml layout
                LayoutInflater inflater = getLayoutInflater();
                View layout = inflater.inflate(R.layout.toast_custom,
                        (ViewGroup) findViewById(R.id.toast_layout_id));
                //set a message
                TextView text = (TextView) layout.findViewById(R.id.text);
                text.setText("This is a Custom Toast Message");
                //Toast configuration
                Toast toast = new Toast(getApplicationContext());
                toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
                toast.setDuration(Toast.LENGTH_LONG);
                toast.setView(layout);
                toast.show();
            }
        });
    }
}
```

代码 3-24　使用自定义 Toast 布局

除非使用 setView()方法设置自定义布局，否则不要使用公共的 Toast 类构造器。如果不使用自定义的布局，必须使用 makeText(Context, int, int)方法来创建 Toast 对象。代码 3-24 中的 setGravity(int, int, int)方法，可以重新设置 Toast 对象的显示位置。这个方法有三个参数，分别为 Gravity 常量、X 轴偏移量、Y 轴偏移量。运行效果如图 3-8 所示。

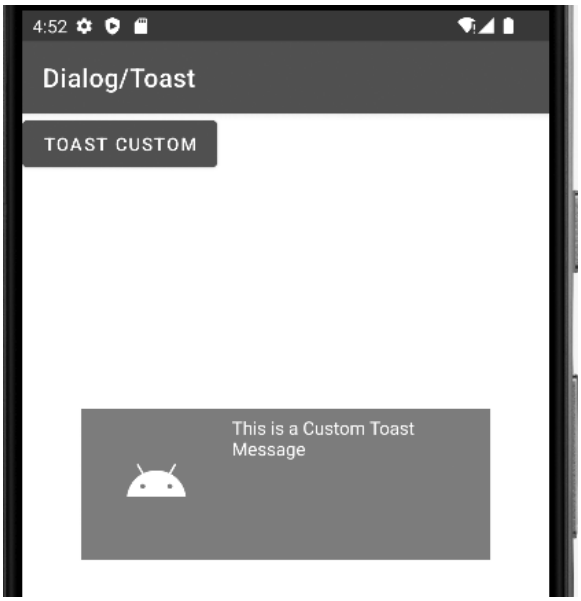

图 3-8　自定义提示控件

3.4　文 本 控 件

安卓用于文本显示和编辑的控件主要包括 TextView 和 EditText 两种，实际上，安卓的很多控件都继承自 TextView 类，包括 Button、CheckTextView、EditText 等，但用于文本显示时常用的还是 TextView 和 EditText，因此这里主要介绍这两个控件如何定义和使用。

3.4.1　TextView

TextView 是安卓中常用的组件之一，用于显示文字，类似 Java 图形界面里的 Label 标签。TextView 中提供了大量的属性用于设置 TextView 的字体大小、字体颜色、字体样式等。由于很多控件都是 TextView 的子类，它们也继承 TextView 的属性，这给应用程序的界面提供了多种显示组合和样式。TextView 的属性可以直接在 XML 布局文件中设置，也可以在 Java 应用程序中设置和修改。例如，用户界面的布局文件 textview_layout.xml 中定义了一个 TextView，在 TextView 几个基本属性基础上增加如下几个属性设置。

(1) android:textColor="#ff0000"设置文字颜色为红色。

(2) android:textSize="24sp"设置文字字号为 24sp。

(3) android:textStyle="bold"设置文字字形加粗。

如果要在 Java 代码中对 TextView 控件属性进行修改，在其布局文件中必须要给这个 TextView 的 ID 属性赋值。TextView 的 ID 属性是这个 TextView 部件的唯一标识，用于 Java 程序对其进行引用。设定 TextView 的 ID 属性的具体语法如下。

```
android:id="@+id/textview_name"
```

代码 3-25　TextView 的 ID 属性具体语法

假设在 textview_layout.xml 文件中设定为 android:id ="@+id/textvw",没有增加属性的设置,在 Java 应用程序 TextViewModifyByCodeActivity 中可以通过 findViewById() 获取 TextView 控件,然后通过对象修改其属性也可以达到同样的效果,如代码 3-26 所示。

```
import android.graphics.Color;
import android.graphics.Typeface;
import android.os.Bundle;
import android.util.TypedValue;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

import com.example.ch03.materialdesign.R;

public class TextViewModifyByCodeActivity extends AppCompatActivity {
    /**
     * Called when the activity is first created.
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.textview);          //设置内容显示的 XML 布局文件
        //取得我们的 TextView 组件
        TextViewtextView = (TextView) findViewById(R.id.textView);
        textView.setTextColor(Color.RED);           //设置成红色
        textView.setTextSize(TypedValue.COMPLEX_UNIT_SP, 24f);  //设置成 24sp
        textView.setTypeface(Typeface.defaultFromStyle(Typeface.BOLD));
                                                    //加粗
    }
}
```

代码 3-26　TextViewModifyByCodeActivity.java

通过上面的尝试,说明通过 Java 代码程序和 XML 布局文件都可以实现 TextView 属性的设置。不过在安卓应用系统开发过程中,还是推荐使用 XML 进行布局和界面外观的设计,使用 Java 程序代码实现程序逻辑。

3.4.2　EditText

EditText 是安卓的文本编辑框,是用户和安卓应用进行数据交互的窗口,可以接受用户的文本数据输入,并将其传送到应用程序中。EditText 是 TextView 的子类,所以 EditText 继承了 TextView 的所有方法和所有属性。EditText 类似于 Java 图形界面的文本编辑框,但与后者相比,增加从 TextView 继承的属性之后,设置 EditText 的显示和输入时,就可以根据不同的需求设计出更加有个性和特点的交互界面。例如,可以通过 EditText 的属性设置文本编辑框的最大长度、空白提示文字等,或者限制输入的字符类型只能为电话号码。表 3-1 中列出了 EditText 常用的一些属性和说明,这些属性也同样适用于 TextView。

表 3-1 EditText 属性

属　性	说　明
android:editable	是否可编辑
android:gravity	设置控件显示的位置，默认为 top
android:height	设置高度
android:hint	设置 EditText 为空时，文本提示信息内容
android:imeOptions	设置附加功能，设置右下角 IME 动作与编辑框相关动作
android:inputType	设置文本的类型，用于帮助输入法显示合适的键盘类型
android:lines	设置 EditText 显示的行数
android:maxLength	设置最大长度
android:maxLines	设置文本的最大显示行数，与 width 或者 layout_width 结合使用，超出部分自动换行，超出行数将不显示
android:numeric	设置为数字输入方式
android:password	以小点“.”显示文本
android:phoneNumber	设置为电话号码的输入方式
android:scrollHorizontally	设置文本超出 TextView 的宽度时，是否出现横拉条
android:textColor	设置文本颜色
android:textColorHighlight	设置被选中后文本颜色，默认为蓝色
android:textColorHint	设置提示文本颜色，默认为灰色
android:textSize	设置文字大小，推荐度量单位“sp”
android:textStyle	设置字形（bold，italic，bolditalic 其一）
android:typeface	设置文本字体（normal，sans，serif，monospace 其一）
android:width	设置宽度

布局设计时，可以根据需要，在 XML 文件中使用上面某些 EditText 的属性，来进行特殊的设置。例如，要求 EditText 中输入特定个数的字符，如身份证号、手机号码等，可以使用 android:maxLength＝"18"设定。下面给出一个例子，说明如何使用 EditText 的常用属性，见代码 3-27。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical" >

<EditText
android:id="@+id/edit_text1"
android:layout_width="fill_parent"
```

代码 3-27 edittext_layout.xml