第5章 Widget控件的使用


上机练习

Widget控件是组成Android用户界面的重要内容。按钮、文本框、图片框等界面控件都属于Widget控件。这些控件利用事件处理机制与用户进行交互，因此，开发者需熟悉Android的事件处理机制。

5.1 事件处理机制

5.1.1 什么是事件处理

一般而言，一个Widget控件在手机屏幕上占据一块矩形区域。它不仅负责渲染这块区域，还处理控件区域内发生的各种事件，例如单击、滑动等操作。此外，还可以设置控件是否可见以及获取焦点等属性。

Android事件处理机制涉及以下几个重要的概念。

(1) 事件(Event)。当用户与界面控件进行交互(如单击、滑动等操作)时，这些控件(事件源)能够触发一系列相应的事件。例如用户单击按钮、滑动屏幕等，都将产生一系列相应的事件。这些事件被封装成各种对象，例如键盘事件KeyEvent对象，触摸屏移动事件MotionEvent对象。

(2) 事件源(Event Source)。事件源是指产生事件的Widget控件对象，例如Button对象、ImageView对象。一个事件源可能产生多个事件。

(3) 事件监听器(Event Listener)。事件监听器负责监听事件源，也就是Widget控件上发生的事件，一旦监听到特定事件，监听器就捕获该事件，并调用事件处理器指定的一个回调方法对事件进行处理。一个事件监听器可以监听一个或者多个事件源的事件。具体来说，事件监听器是View类中的一个接口，它用接口的方式定义了一个用于处理特定事件的回调方法。若要响应特定类型的事件，View类需注册相应的事件监听器并实现相应的回调方法。

(4) 回调(Callback)。它是一种基于接口编程的方法调用机制。在Android开发中，回调机制允许一个类(如A类)定义接口，该接口包含一些抽象方法。B类实现这个接口，并为这些抽象方法提供了具体的实现。A类可以通过注册接口的方法来接收B类的接口实现，并在需要的时候调用B类实现的方法，这种机制称为回调。

(5) 事件处理器(Event Handler)。事件监听器一旦捕获到事件，就会交由相应的事件处理器来识别这一特定事件，并执行相应的响应动作，即调用指定方法对事件进行处理。事件处理器通常是实现事件监听器接口的具体类或匿名内部类。在这个类中，开发者实现了

接口中定义的回调方法，定义了当事件发生时需要执行的代码。

(6) 事件注册(Event Registration)。指将事件处理器注册到事件监听器的过程。完成注册以后，当事件监听器捕获到相应的事件时，就会调用对应的事件处理器。

需要注意，用户的一个交互动作产生的事件可能只有一个，也可能不止一个，例如当用户单击按钮时，会产生一个单击事件，而当用户执行滑屏操作的时候，会产生一系列事件。具体而言，手指触及屏幕时会产生一个 MotionEvent. ACTION_DOWN 事件，手指滑动过程中会产生多个 MotionEvent. ACTION_MOVE 事件，滑屏结束松开手指后又会产生一个 MotionEvent. ACTION_UP 事件。

Android 中的事件处理可以分为两种类型：一种是基于监听的事件处理，另一种是基于回调的事件处理。我们将在后续章节逐一介绍这两种事件处理机制。

5.1.2 基于监听的事件处理

Android 提供了完善的事件处理机制来监听事件、识别事件源，并完成相应的事件处理。基于监听的事件处理流程如图 5.1 所示。

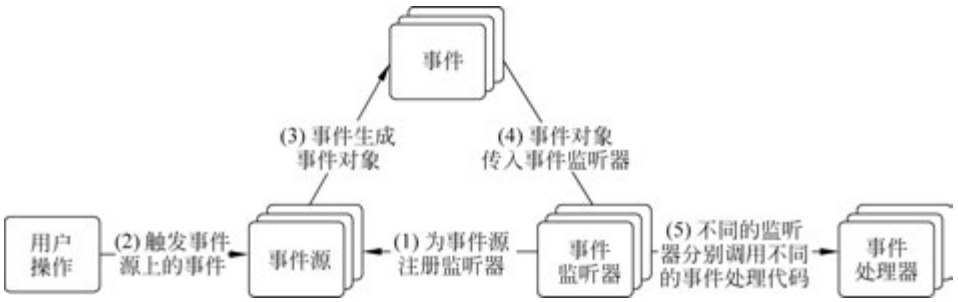

图 5.1 事件监听机制原理图

由图 5.1 可知，基于监听的事件处理机制的流程如下。

(1) 为事件源注册监听器，一个事件源(控件)既可以独享一个监听器，也可以由多个事件源共享一个监听器。

(2) 用户操作 App 时会触发事件源的事件，例如单击按钮(事件源)会产生单击事件。

(3) 事件生成相应的事件对象。

(4) 将事件对象传入事件监听器。

(5) 事件监听器调用相应的事件处理器。

Android 中常用的事件监听器以内部接口形式定义，具体包括以下内容。

(1) OnClickListener 接口。用于处理单击事件。当单击事件发生时，会回调 public void onClick(View v)方法对事件进行处理。其中，参数 v 代表发生单击事件的控件对象，即事件源。

(2) OnLongClickListener 接口。用于处理长按事件。当长按事件发生时，会回调 public boolean onLongClick (View v)方法对事件进行处理。其中，参数 v 代表发生长按事件的控件对象。注意：其返回值类型为布尔类型，当返回 true 时，表明此事件已经处理完毕；返回 false，则表示此事件尚未处理完，还可以继续被其他监听器捕获并处理。

(3) OnTouchListener 接口。用于处理触摸屏幕事件。当控件内产生触摸、按下、抬

起、滑动等动作时都会触发该事件，进而触发该接口中的回调方法 public boolean onTouch (View v, MotionEvent event)。其中，参数 v 代表发生触摸屏幕事件的控件对象。参数 event 为一个 MotionEvent 事件对象，包含触摸的类型、位置和发生时间等信息。

(4) OnKeyListener 接口。用于处理键盘事件。当控件获得焦点以及键盘被敲击时会触发该事件，进而触发该接口中的回调方法 public boolean onKey(View v, int keyCode, KeyEvent event)。其中，参数 v 代表发生键盘事件的控件对象。参数 keyCode 代表键盘码，每个按键都有独一无二的键盘码，用于区别其他按键。参数 event 为一个 KeyEvent 事件对象，包含类型、状态和发生时间等信息。

(5) OnFocusChangeListener 接口。用于处理控件焦点发生改变的事件。如果注册了该接口，当某个控件失去焦点或者获得焦点时，会触发该接口中的回调方法 public void onFocusChange(View v,boolean hasFocus)。其中，参数 v 代表发生控件焦点发生改变事件的控件对象。参数 hasFocus 表示 v 控件是否拥有焦点，true 表示拥有焦点，false 表示失去焦点。

(6) OnCreateContextMenuListener 接口。用于处理上下文菜单显示事件的监听接口，常被用于定义和注册上下文菜单。如果注册了该接口，在某个控件上显示上下文菜单时，会调用对应的回调方法 public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo info)，其中参数 menu 代表事件的上下文菜单，参数 v 代表 View 控件对象，参数 info 封装了有关上下文菜单的额外信息。

基于上述接口，开发者可以调用 setOnXXXListener()方法(XXX 为事件名)为某个控件设置一个监听器，用于监听用户操作引发的事件。当事件发生时，系统会将事件封装成对应的 MotionEvent 或者 KeyEvent 事件对象，然后将事件对象传递给事件源上相应的事件监听器对象，由该对象调用相应的事件处理方法来完成事件处理。

实现基于监听的事件处理方式大致有 6 种，开发者可以任意选择其中一种来实现控件的事件处理。下面以按钮控件为例，介绍如何为其增加事件处理代码。

(1) 基于内部类的事件处理。以第 2 章的示例代码为例，为 MainActivity 中的按钮增加事件处理代码，显示一个 Toast 提示信息。

首先，可以在 MainActivity 类的内部定义一个实现 View. OnClickListener 接口的类，类名由开发者自定义，例如 MyButtonListener。

```
public class MyButtonListener implements View.OnClickListener{
        @Override
        public void onClick(View view) {
           Toast.makeText(MainActivity.this,"基于内部类的事件处理",Toast.LENGTH_ SHORT).
show();
        }
}
```

然后，在 MainActivity 的 OnCreate 方法里注册按钮的事件监听器，代码示例如下：

```
Button myButton = findViewById(R.id.button);
MyButtonListener myButtonListener = new MyButtonListener();
myButton.setOnClickListener(myButtonListener);
```

（2）基于外部类的事件处理。在上面的例子中，实现事件处理时，MyOnClickListener类的定义置于MainActivity类内部，因此称之为基于内部类的事件处理。如果在另一个源文件中单独定义MyOnClickListener类，且注册按钮的事件监听器代码不变，则这种方法被称为基于外部类的事件处理。因为外部类不能直接访问Activity类中的控件，要通过构造方法将控件传入使用，代码相对烦琐，所以这种实现方式不太常用。

（3）基于匿名内部类的事件处理。对于基于内部类的事件处理代码，可以使用匿名内部类来简化代码。这种方法不需要额外定义一个类，直接在注册按钮事件监听器的位置定义相应的事件处理代码，代码示例如下：

```
            Button myButton = findViewById(R.id.button);
            myButton.setOnClickListener(new View.OnClickListener() {
             @Override
             public void onClick(View view) {
                  Toast.makeText(MainActivity.this,"基于匿名内部类的事件处理",Toast.
LENGTH_SHORT).show();
             }
         });
```

另外，也可以把匿名内部类定义为类的成员变量后再引用，代码示例如下：

```
private OnClickListener listener = new OnClickListener() {
@Override
public void onClick(View v) {
      Toast.makeText(MainActivity.this,"基于匿名内部类的事件处理",Toast.LENGTH_SHORT).
show();
}
};
Button myButton = findViewById(R.id.button);
myButton.setOnClickListener(listener);
```

（4）使用Activity作为事件监听器。这种实现方式通过让Activity类实现View.OnClickListener接口，进而实现该接口中唯一的onClick方法。

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener{
//代码略
Button myButton = findViewById(R.id.button);
         myButton.setOnClickListener(this);
//代码略
            @Override
            public void onClick(View view) {
                Toast.makeText(MainActivity.this,"Activity作为事件监听器", Toast.LENGTH_
SHORT).show();
            }
}
```

（5）绑定到标签处理。这种实现方式是在界面布局文件的控件中定义相应的事件处理方法名，以按钮控件为例，定义后的代码示例如下：

```
<Button
android:layout_width="wrap_content"
```

```
android:layout_height = "wrap_content"
android:text = "button"
android:onClick = "clickMe"
/>
```

上述代码指定了控件的 onClick 事件处理方法名为 clickMe，然后在相应的 MainActivity 类中增加该方法的定义如下：

```
    @Override
public void clickMe(View view) {
  Toast.makeText(MainActivity.this,"绑定到标签的事件处理", Toast.LENGTH_SHORT).show();
}
```

这种方式下，控件不需要调用 setOnClickListener()方法指定监听器。

(6) 基于 Lambda 表达式的事件处理。Lambda 表达式是 Java 8 引入的一个重要特性，它可以用更简洁的方式代替匿名内部类。因此，本质上来说，基于 Lambda 表达式的事件处理等价于基于匿名内部类的事件处理。使用 Lamda 表达式可以大大减少基于匿名内部类的事件处理方式的代码量。上面第(3)点的代码用基于 Lambda 表达式的写法改写后的代码如下：

```
    Button myButton = findViewById(R.id.button);
    myButton.setOnClickListener(view -> Toast.makeText(MainActivity.this,"基于匿名内部类
的事件处理",Toast.LENGTH_SHORT).show());
```

从上述代码中，我们惊喜地发现，原本需要 6 行代码才能实现的功能，现在仅需 2 行即可。我们之所以介绍基于 Lambda 表达式的事件处理方式，正是因为它的简洁性。

在上述 6 种实现事件处理方法中，基于匿名内部类的事件处理代码是专用的，只属于特定的控件对象。其余 5 种实现方式的事件处理代码是可以共用的，即多个控件可以共用一个事件处理方法。此时，开发者可以利用事件处理代码中的 View 对象参数结合 getId()方法来区别具体的事件源。以内部类为例，若两个按钮使用了同一个内部类作为监听器，代码示例如下：

```
public class MyOnClickListener implements View.OnClickListener{
        @Override
        public void onClick(View view) {
          int clickedButtonId = view.getId();
          if(clickedButtonId == R.id.button ){        //第一个按钮的 id 为 R.id.button
            Toast.makeText(MainActivity.this," 单击了按钮 1",Toast.LENGTH_ SHORT).show();
          }else if(clickedButtonId == R.id.button2 ){ //第一个按钮的 id 为 R.id.button2
            Toast.makeText(MainActivity.this," 单击了按钮 2",Toast.LENGTH_ SHORT).show();
          }
      }
}
```

综上，基于监听的事件处理方式多种多样。为了更直观地展示具体实现方法，下面我们来看一个完整的示例代码。

新建一个项目 EventDemo，修改布局文件 activity_main.xml，内容如下：

```
<LinearLayout
    xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:orientation = "vertical"
    android:gravity = "center">
    <Button
        android:id = "@ + id/btnInnerClass"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "内部类的示例代码"
        android:layout_margin = "10dp"/>
    <Button
        android:id = "@ + id/btnAnonymousClass"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "匿名内部类的示例代码"
        android:layout_margin = "10dp"/>
    <Button
        android:id = "@ + id/btnActivityListener"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "Activity 作为监听器"
        android:layout_margin = "10dp"/>
    <Button
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "标签处理的示例代码"
        android:onClick = "clickButton"
        android:layout_margin = "10dp"/>
    <Button
        android:id = "@ + id/btnLamda"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "Lamda 表达式的示例代码"
        android:layout_margin = "10dp"/>
    <Button
        android:id = "@ + id/btnOuterClass"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "外部类的示例代码"
        android:layout_margin = "10dp"/>
    </LinearLayout>
```

这是一个垂直线性布局，内部放置了 6 个按钮，分别对应不同的事件监听器实现方式。在布局文件中，除了基于标签的实现方式对应的按钮以外，其他 5 个按钮都需有 id，方便在 App 程序中用 findViewById()找到它们，并设置相应的事件监听器。

另外，需新建一个 Java 类 OuterButtonListener 用来实现基于外部类的按钮单击监听事件处理，由于这个类定义在 MainActivity 类外部，因此属于外部类实现监听器的实现方式。完整的代码如下：

```
public class OuterButtonListener implements View.OnClickListener {
    @Override
    public void onClick(View v) {
        Toast.makeText(v.getContext(), "基于外部类的事件处理",Toast.LENGTH_ SHORT).show();
    }
}
```

在 MainActivity 类添加相应的事件处理代码，完整的代码如下：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    Button btnInnerClass,btnAnonymousClass,btnActivityListener,btnLamda,btnOuterClass;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        //基于内部类的例子
        btnInnerClass = findViewById(R.id.btnInnerClass);
        MyButtonListener myButtonListener = new MyButtonListener();
        btnInnerClass.setOnClickListener(myButtonListener);
        //基于匿名内部类的例子
        btnAnonymousClass = findViewById(R.id.btnAnonymousClass);
        btnAnonymousClass.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(MainActivity.this,"基于匿名内部类的事件处理",Toast.LENGTH_
SHORT).show();
            }
        });
        btnActivityListener = findViewById(R.id.btnActivityListener);
        btnActivityListener.setOnClickListener(this); //Activity 作为监听器
        //Lamda 的例子
        btnLamda = findViewById(R.id.btnLamda);
        btnLamda.setOnClickListener(v -> Toast.makeText(MainActivity.this,"基于 Lamda 的
事件处理",Toast.LENGTH_SHORT).show());
    }
    @Override
    public void onClick(View v) {
        Toast.makeText(MainActivity.this,"Activity 作为监听器",Toast.LENGTH_SHORT).show();
    }
    public void clickButton(View v) {
        Toast.makeText(MainActivity.this,"绑定到标签",Toast.LENGTH_SHORT).show();
    }
    private class MyButtonListener implements View.OnClickListener{
        @Override
        public void onClick(View view) {
            Toast.makeText(MainActivity.this,"基于内部类的事件处理",Toast.LENGTH_ SHORT).
show();
        }
    }
        //外部类的例子
        btnOuterClass = findViewById(R.id.btnOuterClass);
        btnOuterClass.setOnClickListener(new OuterButtonListener());
}
```

MainActivity 的代码功能很简单，首先定义 6 个按钮对象变量对应布局文件中的 6 个按钮，然后在 onCreate()方法中调用 findViewById()方法找到布局文件中对应的按钮，继而调用 setOnClickListener()方法来设置相应的事件监听器。运行程序后，分别单击不同的按钮，就可以观察到这些事件处理代码的执行情况，如图 5.2 所示。

图 5.2　基于事件监听机制的示例代码运行截图

5.1.3　基于回调的事件处理

基于监听的事件处理模型中事件源和事件监听器是分离的，当事件源上产生特定事件之后，事件监听器捕获该事件，然后再调用事件监听器对应的事件处理器来进行处理。而对于基于回调的事件处理模型来说，事件源和事件监听器是统一的，当事件源发生特定事件之后，该事件是由事件源（即控件）通过调用自身重写的回调方法来处理事件。

Android 为所有的控件都提供了基于回调的事件处理方法，例如 View 中提供了 onKeyDown()、onKeyUp()、onTouchEvent()等事件回调方法。这些回调方法的命名很有特点，都是以 on 开头，后面接事件的名称。

常见的事件回调方法可参见表 5.1。

表 5.1　常见的事件回调方法

方　法	功 能 描 述
boolean onKeyDown(int keyCode, KeyEvent event);	在控件上按下键盘某个键时发生，第 1 个参数 keyCode 为被按下键的键码，第 2 个参数为按键事件对象 event，其中包含了触发事件的详细信息：标志、键码、事件发生的时间等
boolean onKeyUp(int keyCode, KeyEvent event);	在控件上松开键盘某个键时发生，第 1 个参数 keyCode 为被按下键的键码，第 2 个参数为按键事件对象 event，其中包含了触发事件的详细信息：标志、键码、事件发生的时间等

续表

方 法	功 能 描 述
boolean onKeyLongPress(int keyCode, KeyEvent event);	在控件上长按键盘某个键时发生，第1个参数keyCode为被按下键的键码，第2个参数为按键事件对象event，其中包含了触发事件的详细信息：标志、键码、事件发生的时间等
boolean onTouchEvent(MotionEvent event)	在控件上触摸屏幕时发生，参数event包含了触摸事件的详细信息：位置、动作类型(ACTION_DOWN：按下，ACTION_MOVE：移动，ACTION_UP：松开)以及触点个数

表5.1中，键码用来表示Android设备的屏幕键盘或物理键盘上的按键，每个按键被定义了一个静态整型常量与之对应，例如整型常量KEYCODE_0代表0键，整型常量KEYCODE_A代表A键，整型常量KEYCODE_HOME代表Home键。另外，每个回调方法的返回值都是boolean类型，当它返回true时，表示事件已经处理完毕，不需要传播出去；如果返回false，表示事件还没有处理完毕，该事件会继续传播出去，供后面的事件处理程序进行处理。

为了理解基于回调的事件处理方法，我们新建一个项目CallbackDemo，演示为Activity增加onKeyDown()和onTouchEvent()的事件回调方法。在默认生成的布局文件activity_main.xml中给TextView增加一个id：txtInfo，修改以后的布局文件如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< androidx.constraintlayout.widget.ConstraintLayout xmlns:android = "http://schemas.android.
com/apk/res/android"
    xmlns:app = "http://schemas.android.com/apk/res - auto"
    xmlns:tools = "http://schemas.android.com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    tools:context = ".MainActivity">
    < TextView
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "Hello World!"
        android:id = "@ + id/txtInfo"
        app:layout_constraintBottom_toBottomOf = "parent"
        app:layout_constraintEnd_toEndOf = "parent"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent" />
</androidx.constraintlayout.widget.ConstraintLayout >
```

若要增加onKeyDown()和onTouchEvent()的事件回调方法，可以单击Code菜单下的Override Methods菜单项，如图5.3所示。

此时，Android Studio会弹出选择重写方法的对话框，如图5.4所示。

双击onKeyDown()方法所在行，Android Studio就会自动在代码编辑区域中生成onKeyDown()的框架代码：

```
  @Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    return super.onKeyDown(keyCode, event);
  }
```

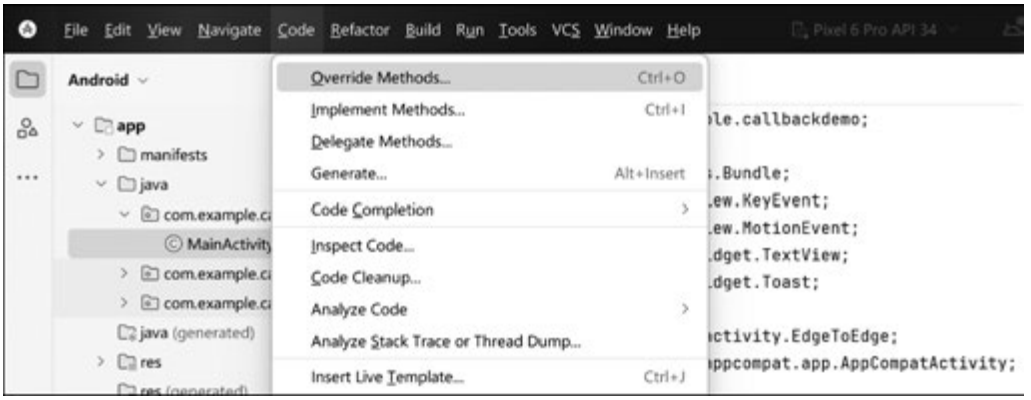

图 5.3　选择重写的菜单项

图 5.4　选择重写方法的对话框

还有一种办法是在代码窗口中直接输入方法名，Android Studio 的智能代码提示功能就会提示相应的方法，帮助用户快速建立重写方法的框架代码，例如输入 onto 这 4 个字母以后，Android Studio 就会把 onTouchEvent()方法推送到第 1 行，如图 5.5 所示。

此时，按 Enter 键或者 Tab 键就可以在代码编辑区域中看到 onTouchEvent()的框架代码：

```
    @Override
public boolean onTouchEvent(MotionEvent event) {
    return super.onTouchEvent(event);
}
```

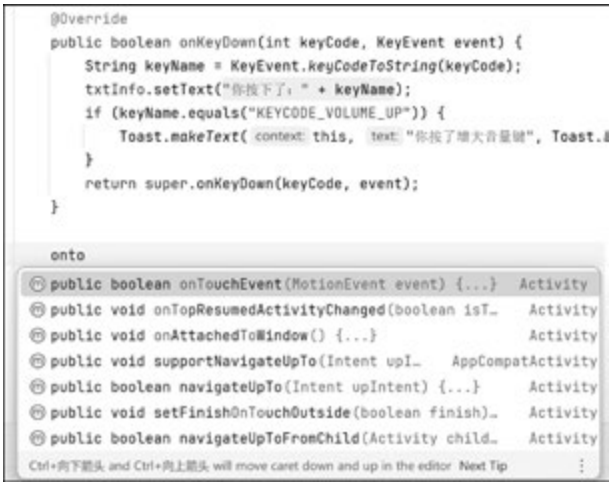

图 5.5 智能代码提示功能辅助完成重写方法

增加功能代码以后，完整的 MainActivity.java 源代码如下：

```
public class MainActivity extends AppCompatActivity {
     TextView txtInfo;
     @Override
     protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_main);
          txtInfo = findViewById(R.id.txtInfo);
     }
     @Override
     public boolean onKeyDown(int keyCode, KeyEvent event) {
          String keyName = KeyEvent.keyCodeToString(keyCode);
          txtInfo.setText("你按下了：" + keyName);
          if(keyName.equals("KEYCODE_VOLUME_UP")){
               Toast.makeText(this,"你按了增大音量键",Toast.LENGTH_LONG).show();
          }
          return super.onKeyDown(keyCode, event);
     }
     @Override
     public boolean onTouchEvent(MotionEvent event) {
          float posX = event.getX();
          float posY = event.getY();
          txtInfo.setText("当前的位置是：" + posX + "," + posY);
          return super.onTouchEvent(event);
     }
}
```

在 MainActivity 中，我们重写了 onKeyDown()和 onTouchEvent()两个方法。如果用户在 App 运行时，按了键盘上的某个键，相应的键码就会显示在屏幕上，如果按了音量增大键则还会弹出一个 Toast 加以提示，如图 5.6 所示。注意：如果是在虚拟设备中运行，直接按计算机键盘上的键即可，而音量增大键则在虚拟设备的工具栏上。

对于 onTouchEvent()，如果是在虚拟设备上运行，用户按下鼠标左键并在虚拟设备屏幕上滑动，可以观察到屏幕上显示的坐标信息在不断变化，显示情况如图 5.7 所示。

图 5.6　onKeyDown()运行截图

图 5.7　onTouchEvent()运行截图

这个例子是不是不难？回顾 5.1.2 节的内容，开发者可以调用 OnTouchListener()方法来为某个控件设置监听器，里面包含一个回调方法 onTouch()。也就是说，有些事件既有基于监听的实现方式，又有基于回调的实现方式。例如 Button 的触摸事件，既有基于监听的 setOnTouchListener()，又有基于回调的 onTouchEvent()。那么问题来了，如果一个控件对于同一个事件，既定义了基于监听的实现方式，又定义了基于回调的实现方式，App 会如何运行呢？

对此，Android 规定的执行顺序是：先执行基于监听的事件处理代码，如果监听器返回 true，则不执行基于回调的事件处理代码；反之，如果监听器返回 false，则继续执行基于回调的事件处理代码。例如如果 setOnTouchListener()中定义的 onTouch()方法返回 true 时，触摸事件已经处理完毕，则 onTouchEvent()方法将不会被执行；反之，如果 onTouch()方法返回 false，则触摸事件将继续传播，执行 onTouchEvent()方法。

学习完这两种事件处理机制后，下面让我们来总结一下这两种机制的区别。

(1) 在基于监听的事件处理机制中事件源和事件监听器是分离的，是一种委托式的事件处理机制——事件源将处理事件的方法委托给监听器对象，事件源不需要操心具体在哪里进行事件处理和怎样实现事件处理，事件源只负责绑定好一个监听器，由该监听器负责监听这个事件是否发生，以及发生之后该如何处理。

(2) 在基于回调的事件处理机制中事件源和事件监听器是统一的，当事件被触发的时候，事件源将执行自身实现的事件处理代码。

(3) 基于监听的事件处理机制由开发者动态添加方法实现，基于回调的事件处理机制需继承某个 UI 控件类，然后重写相应的回调方法，因此在灵活性方面不如基于监听的事件处理机制。

(4) 对于同一类事件,基于监听的事件处理代码优先级要高于基于回调的事件处理代码。

5.2 文本类控件

文本类控件是 Android 中常用的控件之一,主要包括文本框 TextView 和编辑框 EditText 两大类。其中,EditText 是 TextView 的子类,因此 TextView 的属性和方法同样适用于 EditText。另外,通过设置编辑框 EditText 的特定属性,还可以衍生出自动完成文本框和口令输入框等特殊的文本组件。

5.2.1 TextView(文本框)

TextView 文本框用来显示一段文本,其重要属性参见表 5.2。

表 5.2 TextView 的常用属性

属 性	功 能 描 述
android:id	控件的唯一 id 标识
android:text	显示的文本字符串
android:layout_width	设置控件的宽度,可以设为 wrap_content、match_parent 或者指定的宽度,例如 100dp
android:layout_height	设置控件的高度,可以设为 wrap_content、match_parent 或者指定的宽度,例如 30dp
android:textColor	设置文本颜色,可选属性,若没有设置,则默认为黑色
android:textSize	设置字体大小,单位为 sp,例如 12sp
android:gravity	设置文本对齐方式,用于设置控件内部的文本的对齐方式,可以设置为 left、center、right、top、bottom、start、end、fill_horizontal、fill_vertical,还可以用"|"将这些属性组合使用
android:layout_gravity	设置控件对齐方式,用于设置控件在父容器中的位置,可以设置为 left、center、right、top、bottom、start、end、fill_horizontal、fill_vertical,还可以用"|"将这些属性组合使用
android:visibility	设置控件的可见性,可以设为 visible(可见)、invisible(不可见,但占用布局空间)和 gone(不可见,并且不占用布局空间)

下面让我们通过代码来学习一下 TextView 的用法。新建一个项目 TextViewDemo 后,Android Studio 会自动打开 MainActivity.java 源代码,同时打开布局文件 activity_main.xml。单击 activity_main 可在可视化设计模式下展示出包含一个 TextView 控件的界面,该界面已经在前面的章节中展示过,如图 2.6 所示。

开发者可以通过滚动鼠标中键,在 Android Studio 界面右侧的属性视图中查看 TextView 的属性。你会发现它的属性非常多,但不用担心,大部分属性并不需要记住。在实际开发中,开发者通常只会用到其中很少一部分属性,布局文件中的控件也只会包含这些被用到的属性。

另外,属性视图提供了一种快速设置属性的方法,例如,在视图中找到 background 属性,单击这一行显示的吸管图标,Android Studio 会显示一个弹窗,如图 5.8 所示。默认显示 Custom 选项卡,开发者可以用鼠标在该页面的调色板里选择颜色。在鼠标滑动的过程

中，页面中间的 R、G、B 和 Hex 的值会随之而变，而 A%所代表的透明度则不会变化，始终保持为 100，需要手动进行修改。不过需注意，通常不会直接在调色板上选择颜色。因为界面上的控件颜色需要协调搭配，需要一定的美术功底才能设计出一个漂亮的界面。为了规范设计，Google 公司专门提出了 Material Design 设计语言，用于帮助界面设计人员快速实现颜色搭配的设计效果。开发者可以在弹窗下方的下拉列表中选择不同饱和度的 Material Design 颜色搭配。选择后，弹窗最下方的各种色块会随之发生变化，默认选项是 Material 500。

如果开发者记得属性的名称或者部分名称，可以单击属性视图上方的放大镜图标，利用查找功能快速定位属性。例如，需要设置文本大小，可以在放大镜后的输入框里输入 size 快速定位到 textSize 属性，如图 5.9 所示。

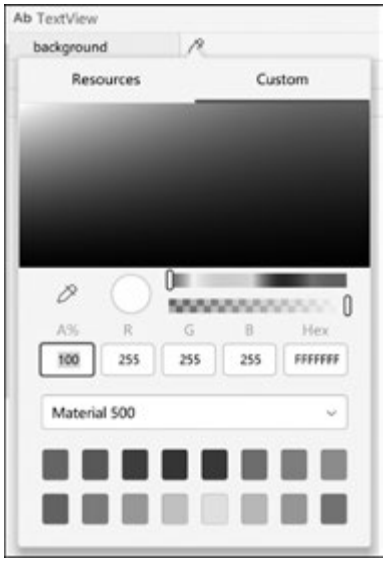

图 5.8　配置背景色的弹窗

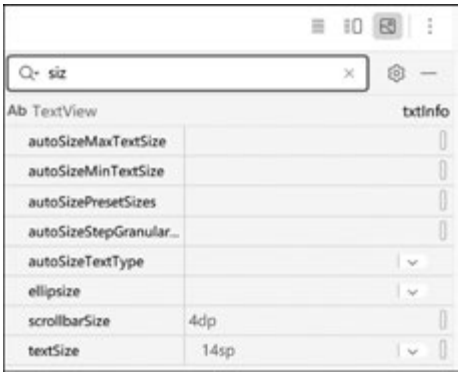

图 5.9　快速查找属性的示意图

除了通过上述的可视化方式设置属性外，开发者还可以单击属性视图上方的 Code 按钮或者 Split 按钮切换到代码模式或者混合模式，然后在布局文件的 XML 代码窗口直接输入新的属性、修改属性值。Android Studio 提供了快捷方便的属性代码提示功能，帮助开发者完成属性代码的输入，如图 5.10 所示。

图 5.10　属性代码提示功能

添加、修改控件属性时，开发者可以在 Android Studio 的界面视图上实时看到控件界面效果的变化，也可以运行程序后在虚拟设备上查看修改的效果。

5.2.2 EditText（输入框）

EditText 输入框用来录入文本信息。由于是 TextView 的子类，所以它不仅支持 TextView 的属性，还具有一些其他重要属性，具体可参见表 5.3。

表 5.3 EditText 的常用属性

属 性	功 能 描 述
android:autoText	设置为 true，EditText 会自动进行输入值的拼写校正
android:hintText	EditText 输入框为空时显示的提示文字信息，一旦开始输入，会自动消失
android:inputType	设置输入类型，如文本、数字、电子邮件、密码、电话等，相应的值可以为 text、number、textEmailAddress、textPassword、phone 等
android:lines	设置文本的行数。例如单行显示，则设置为 android:lines="1"
android:minLines	设置文本框的最小行数
android:maxLines	设置文本框的最大行数
android:digits	设置允许输入的字符列表，不在列表中的字符即使软键盘上有，单击后也无法输入 EditText 中

可视化属性设置的方式同样适用于 EditText 输入框及其他控件。但是，由于每种控件的属性繁多且篇幅有限，我们在后续内容中将不再演示这种方式，而是直接给出 XML 代码。

为了演示文本类控件的用法，我们新建一个项目 TextDemo，然后用可视化拖曳的方式在默认的 ConstraintLayout 布局上添加一个 EditText，操作方法如图 5.11 所示。

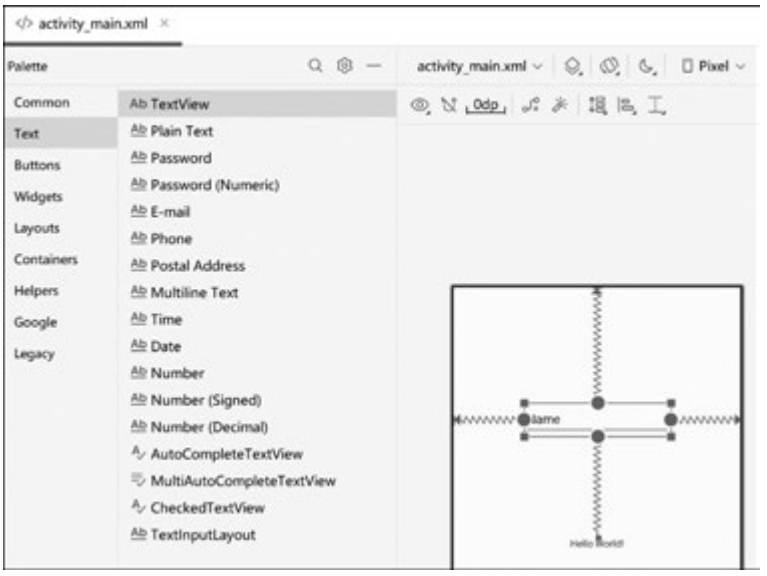

图 5.11 添加 EditText 控件示意图

在控件视图区域单击 Text 菜单项，然后从列出的控件清单中，选择 Plain Text 选项，用鼠标拖曳到界面视图以后，依次将周围的 4 个小圆圈固定到合适的位置，其中，下面的圆圈

固定到 EditText 的上方。

注意：从 Plain Text 到 Number(Decimal)的这些控件其实都是 EditText，它们的区别主要在于设置了不同的 inputType 属性。

单击属性视图上的 Code 按钮切换到代码视图，然后修改 EditText 控件的 id 属性为 editText，并将 inputType 属性修改为 text。修改后的布局文件代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< androidx. constraintlayout. widget. ConstraintLayout xmlns:android = "http://schemas. android.
com/apk/res/android"
    xmlns:app = "http://schemas. android. com/apk/res - auto"
    xmlns:tools = "http://schemas. android. com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:padding = "20dp"
    tools:context = ". MainActivity">
    < TextView
        android:id = "@ + id/textView"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "Hello World!"
        app:layout_constraintBottom_toBottomOf = "parent"
        app:layout_constraintEnd_toEndOf = "parent"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent" />
    < EditText
        android:id = "@ + id/editText"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:ems = "10"
        android:inputType = "text"
        android:text = "Name"
        android:minHeight = "48dp"
        app:layout_constraintBottom_toTopOf = "@ + id/textView"
        app:layout_constraintEnd_toEndOf = "parent"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent" />
</androidx. constraintlayout. widget. ConstraintLayout >
```

上述布局文件包含两个控件：一个是系统默认生成的 TextView，id 为 textView，另一个是手动添加的 EditText，id 为 editText。定义这两个 id 时，id 名称前面的符号“@＋id”表示如果此 id 不存在则新增一个 id。id 的名称位于“@＋id/”后面。

接下来在 MainActivity 的 onCreate()方法里添加相应的代码，内容如下：

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super. onCreate(savedInstanceState);
        EdgeToEdge. enable(this);
        setContentView(R. layout. activity_main);
        TextView txtView = findViewById(R. id. textView);
        EditText edtText = findViewById(R. id. editText);
```

```
            CharSequence csTemp = txtView.getText();
            edtText.setText(csTemp);
            edtText.setTextColor(Color.RED);
        }
    }
```

在 onCreate()方法中，首先调用父类的 onCreate()方法，接下来启用边缘到边缘显示模式，再使用 setContentView()显示界面。setContentView()方法的参数 R.layout.activity_main 可能让部分开发者感到有些困惑。它其实是一个资源 ID，用于关联 res/layout/activity_main.xml 布局文件。简单理解，可以把 R 视为 res 目录(resource 资源目录)，layout 是 res 目录下的子目录，用来存放布局文件，而 activity_main.xml 就是该目录下的一个布局文件，且扩展名".xml"不需要写在这个资源 ID 中。因此，setContentView(R.layout.activity_main)的功能就是显示资源目录的 layout 子目录下定义的 activity_main.xml 布局文件对应的界面。

res 目录下存放了 App 用到的所有资源，包括布局文件、位图、字符串等。在 Android Studio 左侧的项目视图中可以清晰地看到 res 目录的结构和包含的各种资源文件，如图 5.12 所示。

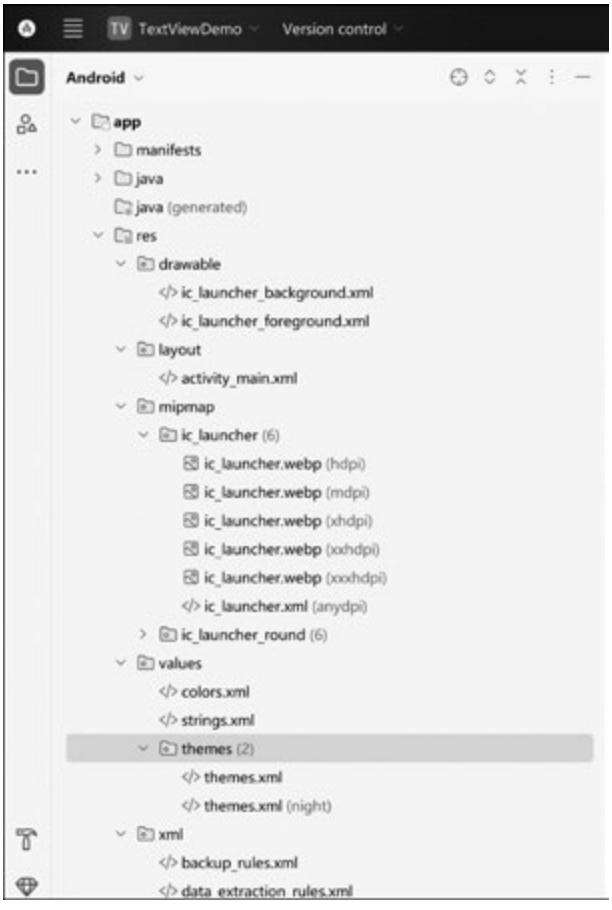

图 5.12 res 资源文件夹

从语法构成上来说，R 是一个类，这个类是 Android SDK 的自动打包工具 aapt 在编译时自动生成的，负责管理 App 开发时用到的各种资源。每个资源会对应一个唯一的整型常量。在 App 调试运行时，当开发者将鼠标移动到资源名上时，会显示一个小弹窗，显示该资源在 R 类中对应的整型常量值，如图 5.13 所示。

图 5.13　资源对应的整型常量

为了方便管理，R 类中包含很多静态内部类（如 layout 类、id 类）。需要说明的是，R 类由系统自动生成和管理，开发者不能对其进行修改。在早期的 Android Studio 版本中可以在项目文件夹下面找到 R.java 文件，而 Android Studio 4 以后的版本基于性能优化的考虑，跳过生成 java 文件，直接生成 R 文件的字节码，因此，R.java 文件已经找不到了，只有一个 R.txt 文件。并且绝大多数情况下，开发者都无须打开 R.java 文件和 R.txt 文件。

所有控件都支持 id 这个重要的属性。它是标识控件的唯一标识符，相当于控件的“身份证号”。在代码中，开发者经常利用 findViewById()方法和控件的 id 号来查找界面布局文件上的控件，获取控件实例对象，再通过对象调用具体的方法来访问控件的值（用控件类的 getXXX()方法），或者修改/设置控件的值（用控件类的 setXXX()方法）。

接下来用 TextView txtView 声明一个 TextView 控件对象变量，并用 findViewById()方法查找上述界面布局文件中 id 为 textView 的控件。然后用 EditText edtText 声明一个 EditText 对象变量，并用 findViewById()方法查找上述界面布局文件中 id 为 editText 的控件。需要说明的是，以前的许多 Android App 开发的图书和资料中，这部分的代码如下：

```
TextView txtView = (TextView)findViewById(R.id.textView);
EditText edtText = (EditText)findViewById(R.id.editText);
```

在 findViewById()方法名的前面需要根据控件的类型，编写强制类型转换的代码，如(TextView)或者(EditText)。而在 Android SDK 26（对应 Android 8.0）以后，使用 findViewById()时已经不需要做强制类型转换了。

接着，使用 txtView.getText()获取 TextView 控件上显示的文字，并将其赋值给 CharSequence 变量 csTemp，然后通过 edtText.setText(csTemp)将这些文字显示在 EditText 控件上。最后，调用 setTextColor(Color.RED)将 EditText 控件中的文本颜色设置为红色。

运行程序，就可以看到“Hello Word!”字样出现在了 EditText 控件上，同时颜色变为红色，如图 5.14 所示。

图 5.14 文本类控件示例的运行结果

在本节中，我们通过一个示例详细介绍了文本类控件的使用方法。我们演示了如何使用 id 属性和 findViewById()方法来获取控件实例，以及如何通过 getXXX()方法和 setXXX()方法对控件的属性进行简单的读取和设置。是不是觉得并不难？接下来，让我们继续学习其他控件的使用方法！

5.3 按钮类控件

按钮是 Android 系统中被广泛使用的一类控件，包括普通按钮(Button)、图片按钮(ImageButton)、单选按钮(RadioButton)等。

5.3.1 普通按钮(Button)

从类的继承关系看，Button 类的父类是 TextView，所以 TextView 控件的属性和方法，Button 控件均能使用，例如 id、text 等属性。除此之外，表 5.4 还列举了一些 Button 的常用属性。

表 5.4 Button 的常用属性

属 性	功 能 描 述
android：onClcik	设置单击按钮以后触发的事件处理方法名称
android：textAllCaps	设置控件中的文本字母是否为全大写。默认为 true，将这个属性设置为 false，则不做大写转换

为了演示 Button 控件的用法，我们新建一个项目 ButtonDemo，并用 3.2.7 节介绍的登录界面的 XML 代码替换布局文件 activity_main.xml 的内容。接下来，我们修改 XML 代

码，给 EditText 和 Button 控件添加 id 属性，并将密码输入框的 inputType 属性设置为 textPassword。修改后的布局文件如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< GridLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "wrap_content"
    android:columnCount = "3"
    android:layout_gravity = "center_vertical">
    < TextView
        android:layout_columnSpan = "3"
        android:layout_gravity = "center_horizontal"
        android:text = "用户登录"
        android:textSize = "24sp" />
    < TextView
        android:textAlignment = "textEnd"
        android:width = "80dp"
        android:text = "用户名：" />
    < EditText
        android:id = "@ + id/edtUserName"
        android:ems = "10" />
    < TextView
        android:layout_column = "0"
        android:layout_gravity = "end"
        android:text = "密码：" />
    < EditText
        android:id = "@ + id/edtPassword"
        android:inputType = "textPassword"
        android:ems = "10" />
    < Button
        android:id = "@ + id/btnLogin"
        android:layout_row = "3"
        android:layout_column = "1"
        android:text = "登录" />
    < Button
        android:id = "@ + id/btnExit"
        android:layout_row = "3"
        android:layout_column = "1"
        android:layout_gravity = "right"
        android:text = "退出" />
</GridLayout >
```

接下来，参照 4.2 节的内容新建一个 Activity，名字叫作 SecondActivity，添加一个 TextView 控件后修改其布局文件的内容为：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< androidx.constraintlayout.widget.ConstraintLayout xmlns:android = "http://schemas.android.
com/apk/res/android"
    xmlns:app = "http://schemas.android.com/apk/res - auto"
    xmlns:tools = "http://schemas.android.com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
```

```
    tools:context = ".SecondActivity">
    < TextView
        android:id = "@ + id/textView"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "登录成功,欢迎使用!"
        app:layout_constraintBottom_toBottomOf = "parent"
        app:layout_constraintEnd_toEndOf = "parent"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent" />
</androidx.constraintlayout.widget.ConstraintLayout >
```

然后,参照 5.1.2 节的内容修改 MainActivity 的代码,内容如下:

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener{
    Button btnLogin,btnExit;
    EditText edtUserName,edtPassword;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnLogin = findViewById(R.id.btnLogin);
        btnLogin.setOnClickListener(this);
        btnExit = findViewById(R.id.btnExit);
        btnExit.setOnClickListener(this);
        edtUserName = findViewById(R.id.edtUserName);
        edtPassword = findViewById(R.id.edtPassword);
    }
    @Override
    public void onClick(View v) {
        int viewId = v.getId();
        if (viewId == R.id.btnLogin) {
            String strUserName = edtUserName.getText().toString();
            String strPassword = edtPassword.getText().toString();
            if (strUserName.equals("admin") && strPassword.equals("123456")) {
                Intent intent = new Intent(this, SecondActivity.class);
                startActivity(intent);
            } else {
                Toast.makeText(this, "用户名或密码输入有误,请重新输入!", Toast.LENGTH_
SHORT).show();
            }
        } else if (viewId == R.id.btnExit) {
            finish();
        }
    }
}
```

在上述代码中,首先定义 MainActivity 实现 View.OnClickListener 接口。接着,针对程序中用到的 4 个控件,定义了相关的对象变量,两个 Buttton 对象变量 btnLogin 和 btnExit,以及两个 EditText 对象变量 edtUserName 和 edtPassword。随后,我们可以在 MainActivity 类的 onCreate()方法中,通过 findViewById()方法获取这 4 个控件,并对上述对象变量进行赋值。之后,针对 Buttton 控件,调用 setOnClickListener()方法设置

MainActivity 为监听器。

接下来，在 MainActivity 类的 onClick()方法中编写代码，以实现登录验证的功能。首先，通过 View 类的 getId()方法获取触发单击事件的控件 id，并将其赋值给 viewId，然后，根据这个值，使用 if 语句分别处理不同按钮的功能逻辑。

图 5.15　按钮示例程序的运行结果

（1）如果 id 是 R.id.btnLogin（即单击“登录”按钮），则用 EditText 类的 getText()方法获取输入的用户名和密码，保存在 String 变量 strUserName 和 strPassword 中。因为 getText()方法的返回值类型不是 String，所以要调用 toString()方法将返回值转换成 String 类型。后续的 if 语句判断用户名是否为 admin、密码是否为“123456”。如果输入正确，则用 4.3 节介绍的 Intent 和 startActivity()方法结合的方式跳转到 SecondActivity。如果输入有误，则用 Toast 类的方法提示用户“密码输入有误，请重新输入！”。

（2）如果 id 是 R.id.btnExit（即单击“退出”按钮），则用 finish()方法关闭 MainActivity。需注意：执行 finish()方法以后，并不意味着 App 已经完全退出，它只是使当前 Activity 不再显示。此时，单击虚拟设备的菜单项，还可以将这个 App 重新载入。

运行这个示例程序，运行结果如图 5.15 所示。

5.3.2　图片按钮（ImageButton）

ImageButton 和 Button 的用法类似，区别在于它显示的不是文字而是图片。另外，尽管 ImageButton 的名字里含有 Button，但它不是从 Button 类派生出来的，而是从用于显示图片的 ImageView 类（参见 5.4 节）派生出来的。ImageButton 的常用属性参见表 5.5。

表 5.5　ImageButton 的常用属性

属　　性	功 能 描 述
android:src	设置 ImageButton 显示的图片资源
app:srcCompat	设置 ImageButton 显示的图片资源。支持矢量图形格式
android:scaleType	设置图片资源在 ImageButton 类里面的缩放类型，例如 centerCrop、fitXY、centerInside 等。也可以通过功能代码使用 setScaleType()方法实现
android:enabled	设置 ImageButton 是否可单击。也可以通过功能代码使用 setEnabled()方法实现
android:visibility	设置 ImageButton 的可见性。也可以通过功能代码使用 setVisibility()方法实现
android:alpha	设置 ImageButton 的透明度，取值为 0 到 1 之间的数值。也可以通过功能代码使用 setAlpha()方法实现
android:rotation	设置 ImageButton 的旋转角度，以度为单位。也可以通过功能代码使用 setRotation()方法实现

为了演示 ImageButton 控件的用法，我们新建一个项目 ImageButtonDemo，然后用 3.2.7 节的登录界面的 XML 代码替换默认生成的布局文件 XML 代码。接着，将布局文件

XML 代码中的 Button 替换成 ImageButton，并为 EditText 和 ImageButton 控件增加 id 属性。最后，增加密码输入框的 inputType 属性，并设置为 textPassword。修改后的 XML 代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< GridLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "wrap_content"
    android:columnCount = "3"
    android:layout_gravity = "center_vertical">
    < TextView
        android:layout_columnSpan = "3"
        android:layout_gravity = "center_horizontal"
        android:text = "用户登录"
        android:textSize = "24sp" />
    < TextView
        android:textAlignment = "textEnd"
        android:width = "80dp"
        android:text = "用户名： " />
    < EditText
        android:id = "@ + id/edtUserName"
        android:ems = "10" />
    < TextView
        android:layout_column = "0"
        android:layout_gravity = "end"
        android:text = "密码： " />
    < EditText
        android:id = "@ + id/edtPassword"
        android:inputType = "textPassword"
        android:ems = "10" />
    < ImageButton
        android:id = "@ + id/btnLogin"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_row = "3"
        android:layout_column = "1"
        android:scaleX = "0.6"
        android:scaleY = "0.6"
        app:srcCompat = "@drawable/login" />
    < ImageButton
        android:id = "@ + id/btnExit"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_row = "3"
        android:layout_column = "1"
        android:layout_marginStart = "120dp"
        android:scaleX = "0.6"
        android:scaleY = "0.6"
        app:srcCompat = "@drawable/logout" />
</GridLayout >
```

接下来，参照 4.2 节的内容新建一个 Activity，名字叫作 SecondActivity，添加一个 TextView 控件后修改其布局文件的内容为：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< androidx. constraintlayout. widget. ConstraintLayout xmlns:android = "http://schemas. android.
com/apk/res/android"
    xmlns:app = "http://schemas. android. com/apk/res - auto"
    xmlns:tools = "http://schemas. android. com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    tools:context = ". SecondActivity">
    < TextView
        android:id = "@ + id/textView"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "登录成功,欢迎使用!"
        app:layout_constraintBottom_toBottomOf = "parent"
        app:layout_constraintEnd_toEndOf = "parent"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent" />
</androidx. constraintlayout. widget. ConstraintLayout >
```

然后,参照 5.1.2 节的内容修改 MainActivity 的代码,内容如下:

```
public class MainActivity extends AppCompatActivity implements View. OnClickListener{
    ImageButton btnLogin, btnExit;
    EditText edtUserName, edtPassword;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super. onCreate(savedInstanceState);
        setContentView(R. layout. activity_main);
        btnLogin = findViewById(R. id. btnLogin);
        btnLogin. setOnClickListener(this);
        btnExit = findViewById(R. id. btnExit);
        btnExit. setOnClickListener(this);
        edtUserName = findViewById(R. id. edtUserName);
        edtPassword = findViewById(R. id. edtPassword);
    }
    @Override
    public void onClick(View v) {
        int viewId = v. getId();
        if (viewId == R. id. btnLogin) {
            String strUserName = edtUserName. getText(). toString();
            String strPassword = edtPassword. getText(). toString();
            if (strUserName. equals("admin") && strPassword. equals("123456")) {
                Intent intent = new Intent(this, SecondActivity. class);
                startActivity(intent);
            } else {
                Toast. makeText(this, "密码输入有误,请重新输入!", Toast. LENGTH_SHORT). show();
            }
        } else if (viewId == R. id. btnExit) {
            finish();
        }
    }
}
```

对比 5.3.1 节的 Button 控件示例代码，本节只调整了 Button 控件的类型为 ImageButton，其余代码均未做改变。由此可见，许多 Android 控件的使用方法很相似，代码的通用性很高，这大大地降低了初学者学习的难度。

运行这个示例程序的结果如图 5.16 所示。

图 5.16 ImageButton 示例程序的运行结果

5.3.3 单选按钮(RadioButton)

RadioButton 按钮是一个常用的功能控件，用于实现从一组选项中选择一个选项的功能。关于 RadioButton 的常用属性，参见表 5.6。

表 5.6 RadioButton 的常用属性

属 性	功 能 描 述
android:button	设置 RadioButton 的按钮样式
android:checked	设置 RadioButton 的初始状态，true 表示选中，false 表示未选中。也可以通过功能代码使用 setChecked()方法实现类似的效果

为了演示 RadioButton 控件的用法，我们新建一个项目 RadioButtonDemo，界面的 XML 代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns: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:orientation = "vertical"
    android:padding = "16dp"
```

```
tools:context = ".MainActivity">
<TextView
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:layout_gravity = "center"
android:textSize = "20sp"
    android:text = "性别" />
<RadioGroup
    android:id = "@ + id/radioGroup"
    android:layout_width = "match_parent"
    android:layout_height = "wrap_content"
    android:orientation = "vertical">
    <RadioButton
        android:id = "@ + id/radioButtonMan"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "男" />
    <RadioButton
        android:id = "@ + id/radioButtonWoman"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "女" />
    <RadioButton
        android:id = "@ + id/radioButtonUnknown"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "性别不详" />
</RadioGroup>
<TextView
    android:layout_width = "match_parent"
    android:layout_height = "1dp"
    android:layout_marginStart = "20dp"
    android:layout_marginEnd = "10dp"
    android:background = "#D9D8D3" />
<TextView
   android:layout_width = "wrap_content"
   android:layout_height = "wrap_content"
   android:layout_gravity = "center"
   android:textSize = "20sp"
   android:text = "母语" />
<RadioGroup
    android:id = "@ + id/radioGroup2"
    android:layout_width = "match_parent"
    android:layout_height = "wrap_content"
    android:orientation = "vertical">
    <RadioButton
        android:id = "@ + id/radioButtonChineseLanguage"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "汉语" />
    <RadioButton
        android:id = "@ + id/radioButtonEnglishLanguage"
        android:layout_width = "wrap_content"
```

```
            android:layout_height = "wrap_content"
             android:text = "英语" />
    </RadioGroup >
    < Button
         android:id = "@ + id/btn_Submit"
         android:layout_width = "wrap_content"
         android:layout_height = "wrap_content"
         android:layout_gravity = "center"
         android:text = "提交" />
</LinearLayout >
```

修改 MainActivity 的代码,内容如下:

```
public class MainActivity extends AppCompatActivity {
    private RadioGroup radioGroup,radioGroup2;
    private Button btnSubmit;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         EdgeToEdge.enable(this);
         setContentView(R.layout.activity_main);
         radioGroup = findViewById(R.id.radioGroup);
         radioGroup.setOnCheckedChangeListener(new MyRadioGroupListener());
         radioGroup2 = findViewById(R.id.radioGroup2);
         radioGroup2.setOnCheckedChangeListener(new MyRadioGroupListener());
         btnSubmit = findViewById(R.id.btn_Submit);
         btnSubmit.setOnClickListener(new View.OnClickListener() {
             @Override
             public void onClick(View v) {
                 String str1 = getSelectedValueFromRadioGroup(radioGroup);
                 String str2 = getSelectedValueFromRadioGroup(radioGroup2);
                 Toast.makeText(MainActivity.this,"选择了:" + str1 + "," + str2, Toast.LENGTH_
LONG).show();
             }
         });
    }
    private String getSelectedValueFromRadioGroup(RadioGroup radioGroup){
         int selectedId = radioGroup.getCheckedRadioButtonId();
         if(selectedId == -1){
             return "";
         }
         RadioButton radioButton = findViewById(selectedId);
         String selectedValue = radioButton.getText().toString();
         return selectedValue;
    }
    class MyRadioGroupListener implements RadioGroup.OnCheckedChangeListener {
         @Override
         public void onCheckedChanged(RadioGroup group, int checkedId) {
             RadioButton radioButton = findViewById(checkedId);
             String text = radioButton.getText().toString();
             Toast.makeText(MainActivity.this, "你选择了:" + text, Toast.LENGTH_SHORT).show();
         }
    }

}
```

在上述代码中，首先针对程序中用到的控件定义了相关的对象变量，其中两个RadioGroup对象变量radioGroup1和radioGroup2，对应两组radioButton，一个Button对象变量为btnSubmit，对应“提交按钮”。于是，我们在MainActivity类的onCreate()方法中，通过findViewById()方法获取界面上相应的这3个控件，并对上述3个对象变量进行赋值。

针对两个radioGroup控件，程序定义了一个内部类MyRadioGroupListener，该类实现了RadioGroup.OnCheckedChangeListener接口，用于监听RadioGroup中RadioButton的选中状态的变化情况。当用户单击RadioGroup中的RadioButton时，就会触发OnCheckedChangeListener的onCheckedChanged()方法。此方法有以下两个参数。

(1) RadioGroup：触发事件的RadioGroup对象。

(2) checkedId：被选中RadioButton的ID值。

通过实现OnCheckedChangeListener接口，可以在onCheckedChanged()方法中编写相应的事件处理功能代码，根据RadioButton的选中状态实现相应功能。

图5.17 RadioButton示例程序的运行结果

本示例代码的功能逻辑较为简单，仅需获取单击的RadioButton的文字内容，因此两个RadioGroup内的控件功能代码逻辑没有区别。这两个RadioGroup都使用同一个类MyRadioGroupListener实现OnCheckedChangeListener接口，进而获取选中的RadioButton的文字内容。在onCheckedChanged()方法中，通过findViewById(checkedId)获取选中状态的RadioButton对象，然后调用getText()方法获取其文字内容，并赋值给text变量。

针对Buttton控件，用setOnClickListener()方法结合匿名类的方式设置监听器。在监听器的onClick()方法中，调用自定义的getSelectedValueFromRadioGroup()方法。该方法接收一个RadioGroup对象作为参数，然后通过getCheckedRadioButtonId()方法获取该RadioGroup中选中的RadioButton的ID，接着，使用findViewById()方法根据这个ID获取对应的RadioButton控件对象，并调用getText()方法获取该对象的文字内容。如果RadioGroup中没有处于选中状态的RadioButton对象时，则getCheckedRadioButtonId()方法的返回值为-1。

运行这个示例程序，选择RadioButton以后单击“提交”按钮，运行结果如图5.17所示。

5.3.4 复选框(CheckBox)

CheckBox控件是一个常用的功能组件，用于实现从一组选项中选择多个选项的功能。关于CheckBox的常用属性，参见表5.7。

表 5.7 CheckBox 的常用属性

属 性	功 能 描 述
android:clickable	设置 CheckBox 是否可以单击，true 表示可以单击，false 表示不可以单击。也可以在功能代码中通过 setClickable()方法实现类似的效果
android:checked	设置 CheckBox 的初始状态，true 表示选中，false 表示未选中。也可以在功能代码中通过 setChecked()方法实现类似的效果

为了演示 CheckBox 控件的用法，我们新建一个项目 CheckBoxDemo，在 5.3.3 节界面的基础上，增加两个 CheckBox 控件，具体方法是复制 5.3.3 节的主界面布局文件 XML 代码，粘贴覆盖这个项目的 activity_main.xml 文件，然后在“提交”按钮的前面增加如下 XML 代码：

```
<TextView
    android:layout_width = "match_parent"
    android:layout_height = "1dp"
    android:layout_marginStart = "20dp"
    android:layout_marginEnd = "10dp"
    android:background = "#D9D8D3" />
<TextView
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:layout_height = "wrap_content"
    android:layout_gravity = "center"
    android:text = "爱好" />
<CheckBox
    android:id = "@ + id/chkMusic"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "音乐"
    android:checked = "false" />
<CheckBox
    android:id = "@ + id/chkSport"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "体育"
    android:checked = "false" />
```

修改 MainActivity 的代码，在该类中增加两个 CheckBox 对象变量的定义，内容如下：

```
private CheckBox chkMusic,chkSport;
```

然后，再修改 MainActivity 的 onCreate()方法，内容如下：

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable(this);
    setContentView(R.layout.activity_main);
    radioGroup = findViewById(R.id.radioGroup);
    radioGroup.setOnCheckedChangeListener(new MyRadioGroupListener());
    radioGroup2 = findViewById(R.id.radioGroup2);
    radioGroup2.setOnCheckedChangeListener(new MyRadioGroupListener());
    chkMusic = findViewById(R.id.chkMusic);
```

```
            chkSport = findViewById(R.id.chkSport);
            btnSubmit = findViewById(R.id.btn_Submit);
            btnSubmit.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    String str1 = getSelectedValueFromRadioGroup(radioGroup);
                    String str2 = getSelectedValueFromRadioGroup(radioGroup2);
                    if(chkMusic.isChecked()){
                        str2 += ",音乐";
                    }
                    if(chkSport.isChecked()){
                        str2 += ",体育";
                    }
                    Toast.makeText(MainActivity.this,"选择了:" + str1 + "," + str2, Toast.LENGTH_
LONG).show();
                }
            });
        }
```

代码与 5.3.3 节的变动不大，在按钮的事件监听器中，用 if 语句分别检查两个 CheckBox 是否处于选中状态，如果是，则添加相应的信息用于显示。

运行这个示例程序，选择相应的选项以后，单击“提交”按钮，运行结果如图 5.18 所示。

图 5.18　CheckBox 示例程序的运行结果

5.4　图片显示控件

为了美化 Android App 的界面，一种常见的方法是在界面上使用美工设计的图片，这可以通过 ImageView 控件实现。这些图片通常保存于 Android 项目的 res/drawable 文件

夹，也可以来自互联网。除了图片以外，ImageView 还可以显示位图、颜色、矢量图形等资源。Android 系统中统称这些资源为 drawable 资源。drawable 是所有可绘制内容的抽象，并有相应的类及子类与之对应。因此，更准确地说，ImageView 控件是用来显示 drawable 资源的功能组件。表 5.8 列出了 ImageView 的常用属性。

表 5.8 ImageView 的常用属性

属 性	功 能 描 述
android:src	设置 App 运行时 ImageView 控件显示的 drawable 资源。若同时设置了 app:srcCompat 属性，则优先显示 android:src 属性指向的 drawable 资源
app:srcCompat	设置矢量 drawable 资源为 App 运行时 ImageView 显示的内容。ImageView 所属的 Activity 需继承自 AppcompatActivity，否则，不显示此矢量 drawable 资源
tools:srcCompat	设置矢量 drawable 资源为 ImageView 预览的内容，App 运行时不显示
android:background	设置 ImageView 的背景
android:alpha	设置图片的透明度，取值为 0 到 1 的数
android:scaleType	设置 drawable 资源的缩放方式

为了演示 ImageView 控件的用法，我们新建一个项目 ImageViewDemo，打开布局文件 activity_main.xml，单击属性视图上方的 Design 按钮，切换到可视化设计模式下。然后，选中默认生成的 TextView 控件，再按 Delete 键将其删除。

接着，将控件视图中的 ImageView 控件拖曳到界面上，松开鼠标时会弹出一个对话框，如图 5.19 所示。

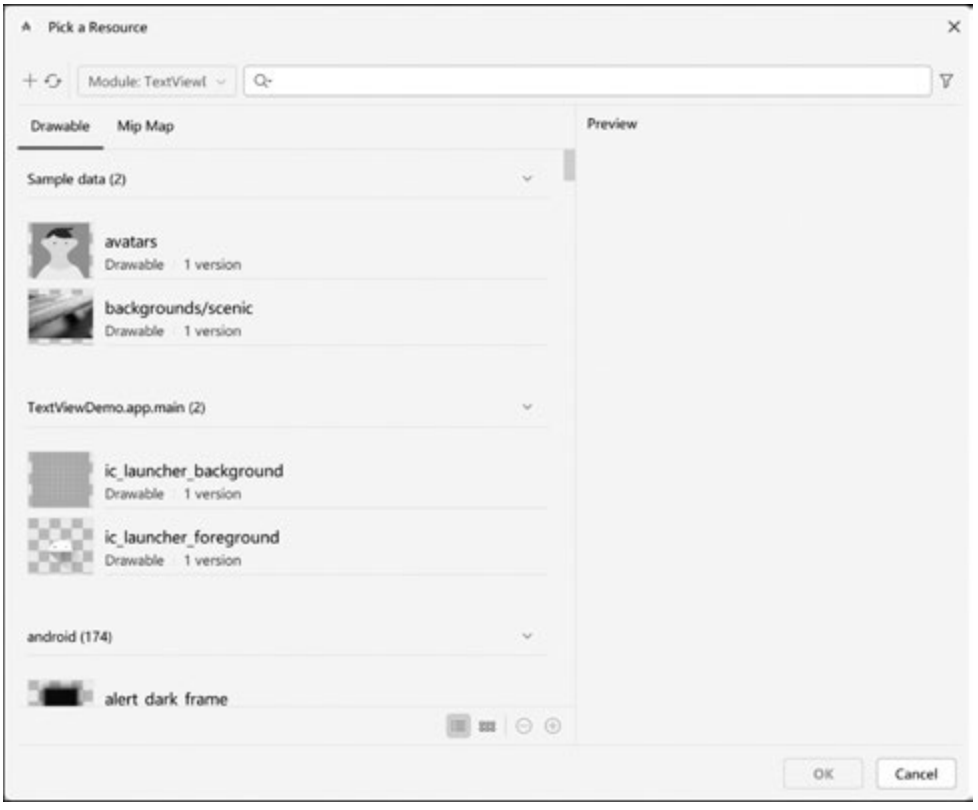

图 5.19 选择一个 drawable 资源的对话框

在该对话框中，我们可以通过选择第1个drawable资源avatars，然后单击该资源，接着单击对话框右下角的OK按钮，完成ImageView控件的添加。最后，拖曳该控件周围显示的4个小圆圈将其固定下来，如图5.20所示。完成添加以后的布局文件如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< androidx. constraintlayout. widget. ConstraintLayout xmlns:android = "http://schemas. android.
com/apk/res/android"
    xmlns:app = "http://schemas. android. com/apk/res - auto"
    xmlns:tools = "http://schemas. android. com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    tools:context = ". MainActivity">
    < ImageView
        android:id = "@ + id/imageView"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        app:layout_constraintBottom_toBottomOf = "parent"
        app:layout_constraintEnd_toEndOf = "parent"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent"
        tools:srcCompat = "@tools:sample/avatars" />
</androidx. constraintlayout. widget. ConstraintLayout >
```

在界面视图中，我们可以清晰地看到显示了一张图片的ImageView控件，如图5.20所示。运行程序，在虚拟设备中呈现的却是一个空白的Activity。这是为什么呢？

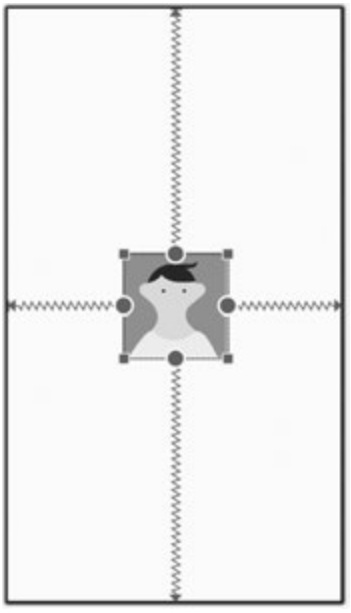

图5.20　添加了ImageView控件后的界面视图

注意看设置这张图片的XML代码：

```
tools:srcCompat = "@tools:sample/avatars"
```

属性名 srcCompat 前面的 tools 表明该属性来自 tools 命名空间。ImageView 控件的图片通常在运行时显示。若需在开发阶段预览界面视图，可以使用 tools:src 属性。这个属性在实际运行时不会展示图片，而是作为占位符，辅助程序员在设计和开发过程中查看效果。需注意：常用的属性均可使用 tools 命名空间。

为了在运行时显示图片，可以在 ImageView 的节点内增加 app:srcCompat 属性。增加以后的 XML 代码如下：

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="125dp"
    android:layout_height="138dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/lab"
    tools:srcCompat="@tools:sample/avatars" />
```

再次运行程序，可以看到如图 5.21 所示的界面。

图 5.21 添加了 ImageView 控件后的运行界面

众所周知，不同 Android 手机、平板的分辨率差异很大，因此，App 为了适配不同屏幕尺寸的手机、平板，需在资源文件夹中定义对应尺寸的界面布局。以下是常用的布局资源文件夹。

(1) layout 文件夹：用于存放一些通用布局 XML 文件，例如界面顶部和底部的布局，原因是这些布局不会随着屏幕大小变化。

(2) layout-small 文件夹：用于小屏幕设备(尺寸约 3 英寸[1 英寸＝2.54 厘米])的布局资源。

(3) layout-normal 文件夹：用于中等屏幕设备(尺寸 4～5 英寸)的布局资源。

(4) layout-large 文件夹：用于大屏幕设备(尺寸 5～7 英寸)的布局资源。

(5) layout-xlarge 文件夹：用于超大屏幕设备(尺寸 7～10 英寸)的布局资源。

另外，还需要针对不同的分辨率定义多组图片资源，以下是常用的图片资源文件夹。

(1) drawable 文件夹：用于存放适用于所有设备的图片资源。

(2) drawalbe-ldpi 文件夹：用于存放低分辨率的图片资源——QVGA(240×320)。

(3) drawable-mdpi 文件夹：用于存放中等分辨率的图片资源——HVGA(320×480)。

(4) drawable-hdpi 文件夹：用于存放高分辨率的图片资源——WVGA(480×800)和 FWVGA (480×854)。

(5) drawable-xhdpi 文件夹：用于存放分辨率至少为 960×720 的高清图片资源。

当 Android App 运行时，系统会根据设备的分辨率选择最合适的布局资源和 drawable 子目录中的图片资源。然而，这些资源会占据大量的存储空间，导致 App 的体积变大。为了解决这个问题，Android API 21(5.0)以后，引入了 Drawable 的子类 VectorDrawable 来渲染矢量图形，还引入了 AnimatedVectorDrawable 类用于播放矢量动画。从此，Android 支持可缩放矢量图形。SVG 是一种基于 XML 标记语言来描述二维矢量图形的图形格式，具有体积小、支持动画和交互、无限缩放不失真的特性。引入 SVG 以后，可以大大减小 Android App 的体积。

Android 的 ImageView 支持显示 SVG 矢量图形。为了将 SVG 矢量图形资源导入项目中，可以在 drawable 文件夹上右击，然后选择 New→Vector Asset 菜单项，如图 5.22 所示。

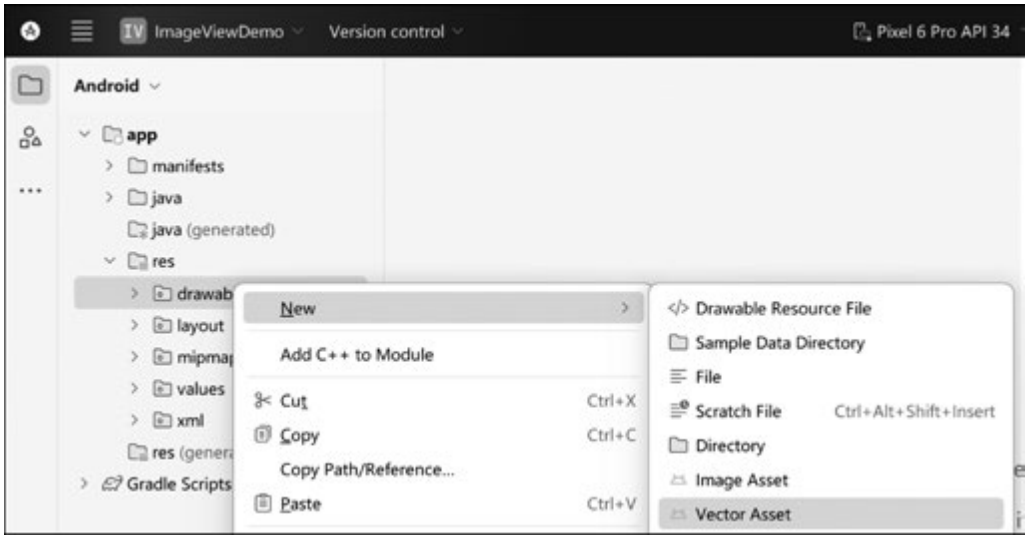

图 5.22　新建矢量图形资源

接下来，Android Studio 将显示一个对话框，让用户选择相应的设置，具体设置如图 5.23 所示。

其中，Asset type 项让用户从 Clip art(Android Studio 资源库)或者 Local file(本地文件)中选择矢量图形资源。

Name 项让用户设置矢量图形资源名称，默认以“ic_”开头。

如果选择从本地文件加载矢量图形资源，对话框会显示 Path 项，用于指定本地文件的路径名。

Size 项用于设置矢量图形的分辨率。例如 128×128dp。

Opacity 项用于指定矢量图形的不透明度。默认为 100%的不透明度，即完全不透明。

Enable auto mirroring for RTL layout 项用于在自右向左的布局中自动镜像显示矢量图形资源，该选项主要适用于一些自右向左显示的语言，例如阿拉伯文。

在图 5.23 所示的对话框中选择 Local file 选项，输入矢量图像文件的名称，然后选择本地矢量图形 SVG 或 PSD 文件，再单击 Next 按钮，系统将显示如图 5.24 所示对话框。从中

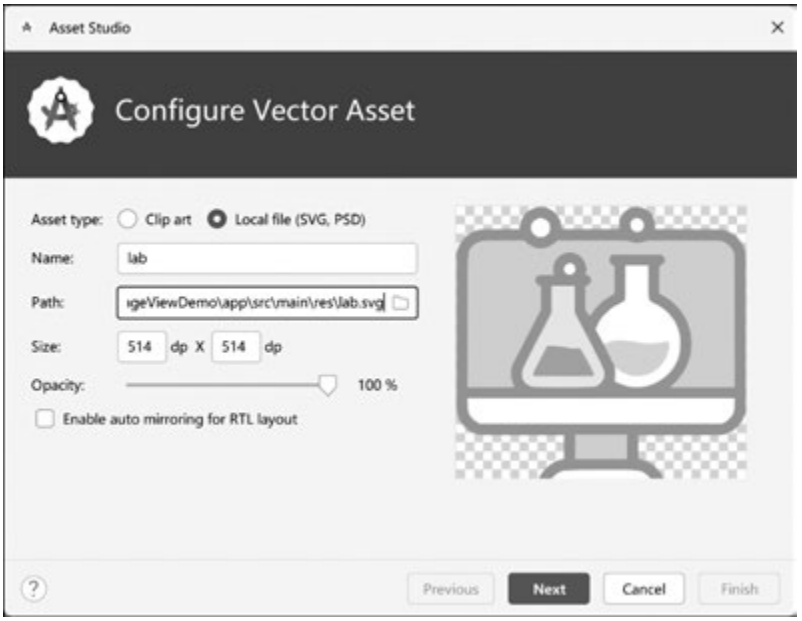

图 5.23 矢量图形参数设置对话框

可以看出,矢量图形文件资源对应的 XML 文件已经保存在 App 项目的 res/drawable 目录下。

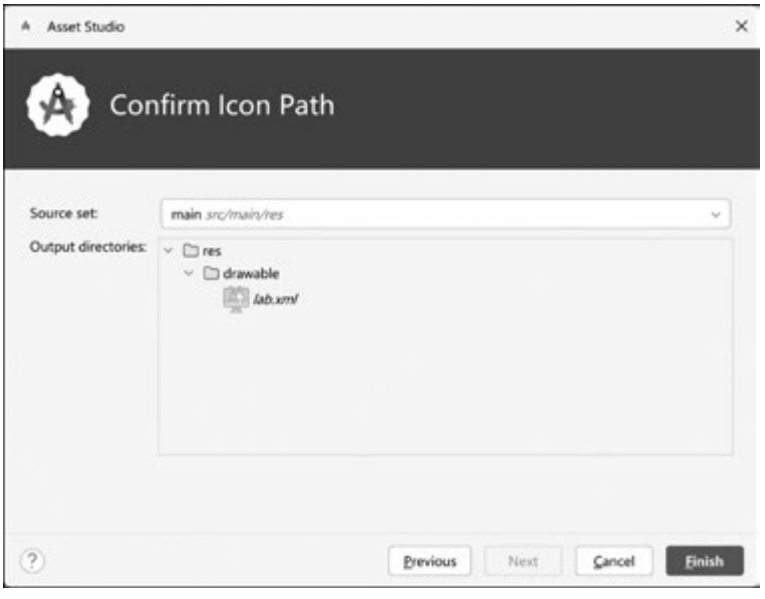

图 5.24 矢量图形参数设置对话框

修改 ImageView 控件的 app:srcCompat 属性为:

```
app:srcCompat = "@drawable/lab"
```

单击 Finish 按钮以后,再次运行 App,就可以看到如图 5.21 所示的运行界面。

5.5 对话框

对话框是 Android App 中一种常用的用户界面元素，用于提示信息或者进行其他交互，比如显示信息、警告、确认、选择和输入等。

Android SDK 中定义了一个对话框的基类——Dialog 类，可以用其创建各种类型的对话框，但开发者很少直接使用它来创建对话框，原因是不同类型对话框的创建涉及的设置项较多，操作较为烦琐。因此，Android SDK 提供了各种类型的 Dialog 类的子类，方便用户创建各种类型的对话框。

下面将介绍 Android 系统中常见的对话框。

5.5.1 提示对话框(AlertDialog)

AlertDialog 是最常用的 Android 对话框，用于显示一些信息、进行操作确认或选择。它广泛应用于弹出提示、确认操作以及输入等场景，例如删除操作之前为了防止用户误删除，一般都要弹出一个对话框让用户确认是否删除。

为了演示 AlertDialog 的用法，我们新建一个项目 DialogDemo，修改布局文件 activity_main.xml 文件为垂直线性布局，增加一个按钮控件。项目的布局文件如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns: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:orientation = "vertical"
    android:gravity = "center">
    < Button
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:id = "@ + id/btnAlertDialog"
        android:text = "AlertDialog 示例"
        android:textAllCaps = "false"/>
</LinearLayout >
```

布局文件很简单，里面就只有一个 Button 控件，id 为 btnAlertDialog。接下来修改 MainActivity 文件，增加显示 AlertDialog 对话框的相关代码：

```
public class MainActivity extends AppCompatActivity {
    private btnShowAlertDialog;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        Button btnShowAlertDialog =  findViewById(R.id.btnAlertDialog);
        btnShowAlertDialog.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                showAlertDialog();
            }
        });
    }
    private void showAlertDialog(){
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("提示");
        builder.setMessage("AlertDialog的示例");
        builder.setPositiveButton("确定", new DialogInterface. OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Toast.makeText(MainActivity.this, "单击了确定按钮", Toast.LENGTH_SHORT).
show();
            }
        });
        builder.setNegativeButton("取消", new DialogInterface. OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Toast.makeText(MainActivity.this, "单击了取消按钮", Toast.LENGTH_SHORT).
show();
            }
        });
        AlertDialog dialog = builder.create();
        dialog.show();
    }
}
```

从代码可以看出，在按钮的单击事件中调用自定义的showAlertDialog()方法。该方法并没有直接使用 AlertDialog 类来创建对话框，而是使用了 AlertDialog.Builder 类来辅助 AlertDialog 对话框的创建。许多对话框类都包含一个 Builder 静态子类来辅助完成对话框的创建。顾名思义，AlertDialog.Builder 是 AlertDialog 的创建者。

理清了这一点以后，下面的代码就容易理解了。AlertDialog.Builder 类用各种 set 方法来设置不同的对话框参数，包括标题(setTitle)、对话框显示的消息(setMessage)、“确定”按钮的文字和单击监听事件(setPositiveButton)、“取消”按钮的文字和单击监听事件(setNegativeButton)。然后使用 create()方法完成 AlertDialog 对话框的创建，再调用 show()方法来进行显示。注意：上述 set 方法的调用顺序没有固定要求，可以自由调整。

运行这个示例程序，选择相应的控件以后，单击“提交”按钮，运行结果如图 5.25 所示。

上面提到的 Builder 类是设计模式中的创建者(Builder)模式的一个应用。Android 的对话框大量采用了这种模式。

图 5.25 AlertDialog 示例程序的运行结果

使用创建者模式的原因是在面向对象的程序设计中，开发者可以通过构造方法来初始化对象的参数，但是，当参数较多时，每次调用构造方法的参数个数可能变化较大，例如参数个数可能从 2 个到 8 个不等。为了适应这种变化，需要编写多个构造方法，以满足不同的参数需求。这种方式使用起来不灵活。采用创建者模式后可以按需创建，解决使用构造方法不灵活的问题。另外，使用创建者模式时，通常采取结合链式调用(Chained invocation)的编程风格——在同一个对象上连续调用多个方法，每个方法返回该对象本身。这种编程风格可以使代码更加简洁且更具有可读性。这种风格的 API 设计也被称为 fluent API。

如果采用链式调用来实现 showAlertDialog()，则具体的代码如下：

```
private void showAlertDialog (){
          AlertDialog.Builder builder = new AlertDialog.Builder(this);
          builder.setTitle("提示")
                   .setMessage("使用链式调用运行的 AlertDialog 的示例")
                   .setPositiveButton("确定", new DialogInterface.OnClick Listener() {
             @Override
             public void onClick(DialogInterface dialog, int which) {
                 Toast.makeText(MainActivity.this, "单击了确定按钮", Toast.LENGTH_SHORT).
show();
             }
          })
                 .setNegativeButton("取消", new DialogInterface.OnClick Listener() {
             @Override
             public void onClick(DialogInterface dialog, int which) {
                  Toast.makeText(MainActivity.this, "单击了取消按钮", Toast.LENGTH_SHORT).
show();
             }
          })
                  .create()
                  .show();
    }
```

注意：在上面的代码中，除了 show()方法带有分号外，其余方法均没有带分号。这些方法其实可以写在同一行，但为了便于阅读代码，每个方法都单独占一行。

5.5.2 输入对话框

Android 的输入对话框用于向用户提供输入信息的操作界面。输入对话框通常包含一些用于接收用户输入的文本、数字等信息的功能组件，例如 EditText。用户可以在对话框中输入这些信息，并通过对话框的按钮进行确认或取消操作。然而，Android SDK 并没有提供专门的输入对话框类。开发者可以利用 5.5.1 节提到的 AlertDialog 类来实现输入对话框。

为了演示输入对话框的创建和使用，我们在 5.5.1 节创建的项目 DialogDemo 的布局文件 activity_main.xml 中再增加一个按钮控件，id 为 btnInputDialog。此按钮的 XML 代码如下：

```
<Button
     android:layout_width = "wrap_content"
     android:layout_height = "wrap_content"
     android:id = "@ + id/btnInputDialog"
     android:text = "输入对话框示例"/>
```

然后，在 MainActivity 类中增加此按钮的处理代码：

```
    Button btnInputDialog = findViewById(R.id.btnInputDialog);
   btnInputDialog.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
        showInputDialog();
   }
});
```

显示输入对话框代码定义在了 showInputDialog()方法里，具体代码如下：

```
private void showInputDialog(){
     final EditText input = new EditText(this);
     AlertDialog.Builder builder = new AlertDialog.Builder(this);
     builder.setView(input)
               .setTitle("输入对话框")
               .setPositiveButton("确定", new DialogInterface.OnClickListener() {
         @Override
         public void onClick(DialogInterface dialog, int which) {
             String text = input.getText().toString();
             Toast.makeText(MainActivity.this, "输入的内容是: " + text, Toast.LENGTH_
SHORT).show();
         }
    })
            .setNegativeButton("取消", new DialogInterface.OnClickListener() {
         @Override
         public void onClick(DialogInterface dialog, int which) {
              dialog.cancel();
         }
    })
              .show();
}
```

代码中首先创建了一个 EditText 对象实例，并将其赋值给 input，然后利用 Builder 对象和链式调用完成输入对话框的创建和显示。其中，setView()方法是实现输入对话框的关键，通过将 EditText 对象实例作为参数传递给该方法，就能实现一个简单的输入对话框了。

运行这个示例程序，单击输入对话框对应的按钮，运行结果如图 5.26 所示。

上面的例子只有一个输入框，因此可以将 EditText 对象作为参数传递到 setView()方法里。若需要在输入对话框里包含两个输入框，应首先在代码中实例化一个 LinearLayout 对象，然后将两个输入框添加到此 LinearLayout 对象中，再将此对象作为参数传递给 setView()方法，即可实现对话框中两个输入框的布局。具体的实现代码如下：

```
private void showInputDialog2(){
         LinearLayout layout = new LinearLayout(this);
         layout.setOrientation(LinearLayout.VERTICAL);
         final EditText input1 = new EditText(this);
         input1.setHint("输入框 1");
         layout.addView(input1);
         final EditText input2 = new EditText(this);
         input2.setHint("输入框 2");
```

```
            layout.addView(input2);
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setView(layout)
                    .setTitle("输入对话框")
                    .setPositiveButton("确定", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            String text1 = input1.getText().toString();
                            String text2 = input2.getText().toString();
                            Toast.makeText(MainActivity.this, "输入的内容是: " + text1 + ",
" + text2, Toast.LENGTH_SHORT).show();
                        }
                    })
                    .setNegativeButton("取消", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            dialog.cancel();
                        }
                    })
                    .show();
        }
```

图 5.26　输入对话框运行结果示意图一

在这段代码中,首先实例化一个 LinearLayout 对象,并设置它的方向为垂直排列。然后,创建两个 EditText 控件 input1 和 input2,并用 LinearLayout 类的 addView()方法将它们作为子视图添加到 LinearLayout 中。最后,调用 setView()方法将 LinearLayout 对象设置为 AlertDialog 的视图,从而实现了在一个输入对话框中包含两个输入框的功能。

接下来,在 onCreate()方法里增加单击按钮显示输入对话框的代码:

```
    Button btnInputDialog2 = findViewById(R.id.btnInputDialog2);
btnInputDialog.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        showInputDialog2();
    }
});
```

运行这个示例程序，单击输入对话框对应的按钮，运行结果如图 5.27 所示。

图 5.27 输入对话框运行结果示意图二

如果输入对话框的布局较为复杂，也可以创建一个布局文件。创建对话框布局文件的操作如图 5.28 所示。右击 layout 文件夹以后，在弹出的菜单项中层层选择，选择 layout XML File 菜单项。

接下来，在弹出的对话框中输入 XML 文件名 input_dialog_layout。创建对话框布局 XML 文件以后，修改布局文件代码，添加两个按钮和一个 Switch 开关控件，修改后的 XML 文件如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "wrap_content"
    android:orientation = "vertical">
    < EditText
        android:id = "@ + id/editText1"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:hint = "输入框 1" />
```

```
    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="输入框 2" />
    <Switch
        android:id="@+id/switch1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="信息公开" />
</LinearLayout>
```

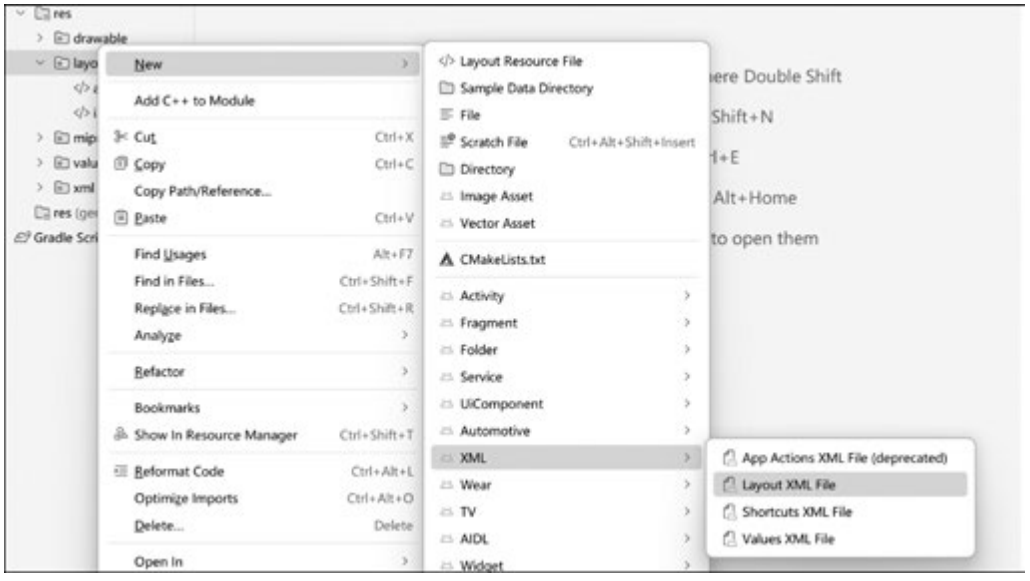

图 5.28　创建对话框布局 XML 文件的示意图

然后，在 MainActivity 类中编写相关的显示对话框方法，具体代码如下：

```
private void showInputDialog3(){
        LayoutInflater inflater = getLayoutInflater();
        View dialogView = inflater.inflate(R.layout.input_dailog_layout, null);
        final EditText editText1 = dialogView.findViewById(R.id.editText1);
        final EditText editText2 = dialogView.findViewById(R.id.editText2);
        final Switch switch1 = dialogView.findViewById(R.id.switch1);
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setView(dialogView)
                .setTitle("输入对话框")
                .setPositiveButton("确定", new DialogInterface. OnClickListener() {
                    @Override
                   public void onClick(DialogInterface dialog, int which) {
                        String input1 = editText1.getText().toString();
                        String input2 = editText2.getText().toString();
                        boolean isChecked = switch1.isChecked();
                        Toast.makeText(MainActivity.this, "输入的内容是: " + input1 + ",
" + input2 + "," + isChecked, Toast.LENGTH_SHORT).show();
                    }
```

```
                })
                .setNegativeButton("取消", new DialogInterface. OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.cancel();
                    }
                })
                .show();
    }
```

在这段代码中，首先通过 getLayoutInflater()方法获取布局加载器 LayoutInflater 的实例。LayoutInflater 是 Android 中的一个类，主要用于动态加载布局，并将布局文件实例化为 View 对象。具体方法是调用 inflate()方法将 R. layout. input_dialog_layout 对应的 XML 布局文件转换为 View 对象 dialogView。接着，调用 findViewById()方法找到界面上所有控件对象，以方便在后续的操作中访问这些控件。最后，调用 setView()方法将 dialogView 设置为 AlertDialog 的视图，从而实现了一个相对复杂的输入对话框。

接下来，在 onCreate()方法里增加单击按钮的代码内容如下：

```
Button btnInputDialog3 = findViewById(R.id.btnInputDialog3);
btnInputDialog.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
          showInputDialog3();
      }
});
```

运行这个示例程序，单击输入对话框对应的按钮，运行结果如图 5.29 所示。

图 5.29 输入对话框运行结果示意图三

AlertDialog 的功能远不止上面介绍的内容，结合 AlertDialog. Builder 类的 setItems()、setSingleChoiceItems()和 setMultiChoiceItems()方法还可以实现列表对话框、单选对话框和多选对话框，本小节不再赘述。

5.6 进度条 ProgressBar

进度条 ProgressBar 是 Android 的一种重要控件，用于展示某项任务的进展情况，被广泛应用于各种场景，例如下载文件、导入/导出数据、加载图片等。

关于 ProgressBar 控件的常用属性，参见表 5.9。

表 5.9 ProgressBar 的常用属性

属 性	功 能 描 述
android:max	设置进度条的最大值，即进度条走完时的进度值，默认是 100
android:min	设置进度条的最小值
android:progress	进度条的进度
android:secondaryProgress	第二进度，常用于展示视频播放时后台缓存视频的进度，后台缓存的进度通常大于播放进度，否则视频就会卡顿
android:visibility	设置进度条的可见性：值为 View. GONE 代表控件完全消失，不再占据屏幕空间；值为 View. VISIBLE 代表控件可见；值为 View. INVISIBLE 代表控件不可见，但还占据屏幕空间。许多控件都支持这个属性
android:style	设置进度条的样式，包括水平样式、圆形样式等
android:progressTint	设置进度条的前景色
android:backgroundTint	设置进度条的背景色
android:indeterminate	设置是否为不确定模式(indeterminate mode)的进度条，如果为 true，进度条会显示一个动画效果。部分后台任务的执行进度无法确定，例如查询数据库，此时，可以将进度条设置为不确定的进度条

这些属性大多都有相应的 set 方法与之对应，例如可以调用 setProgress()方法修改进度条的进度。

为了演示进度条 ProgressBar 控件的用法，我们新建一个项目 ProgressBarDemo，界面的 XML 代码如下：

```
<LinearLayout xmlns: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:orientation = "vertical"
    android:padding = "16dp"
    android:gravity = "center"
    tools:context = ".MainActivity">
<ProgressBar
    android:id = "@ + id/progressBar"
    style = "?android:attr/progressBarStyleHorizontal"
    android:layout_width = "match_parent"
    android:layout_height = "wrap_content" />
    <TextView
```

```
        android:id="@+id/txtProgress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="进度：0%" />
    <Button
        android:id="@+id/btnStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="开始" />
</LinearLayout>
```

上面的代码里设置进度条样式时，符号“?”表示引用系统资源，android:attr 表示系统属性，progressBarStyle Horizontal 表示进度条的水平样式。也就是将进度条的样式设置为系统定义的水平进度条样式。

接下来，在 MainActivity 中编写相应的代码：

```
public class MainActivity extends AppCompatActivity {
    private ProgressBar pbProgressBar;
    private TextView txtProgress;
    private Button btnStart;
    private int progressStatus = 0;                    //当前进度
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        pbProgressBar = findViewById(R.id.progressBar);
        txtProgress = findViewById(R.id.txtProgress);
        btnStart = findViewById(R.id.btnStart);
        btnStart.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                progressStatus = 0;
                startProgress();
                btnStart.setEnabled(false);
            }
        });
    }
private void startProgress() {
        new Thread(new Runnable() {
            public void run() {
                while (progressStatus < 100) {
                    progressStatus += 1;
                    runOnUiThread(new Runnable() {      //在UI线程中更新进度条和文本显示
                        public void run() {
                            pbProgressBar.setProgress(progressStatus);
                            txtProgress.setText("进度：" + progressStatus + "%");
                                                                    //更新进度文本显示
                        }
                    });
                    SystemClock.sleep(100);            //休眠100毫秒
```

```
                }
                runOnUiThread(() ->{            //Lamda 表达式
                    btnStart.setEnabled(true);
                });
            }
        }).start();
    }
}
```

在 MainActivity 类中，首先定义了 pbProgressBar、txtProgress 和 btnStart 这三个控件对象变量，以便于在 onCreate()方法中通过 findViewById()方法及控件 id 找到相应的控件。另外，还定义了一个整型变量 progressStatus，用于保存当前的进度。在按钮 btnStart 的 onClick()方法中，将这个整型变量赋值为 0，随后调用 startProgress()方法。在 startProgress()方法中，通过 new Thread(new Runnable())创建了一个新线程，并通过 new Runnable()创建的匿名内部类实现了 Runnable 接口，重写了该接口的 run()方法。run()方法中定义了线程的具体逻辑，当线程被启动以后，run()方法中的代码将在新线程中执行。

run()方法用一个 while 循环来模拟并展示进度条的运行。每循环一次就给 progressStatus 累加 5，然后将累加后的值传递给 ProgressBar 类的方法 setProgress()，另外，调用 EditText 类的 setText()方法来输出进度信息。为了防止进度条变动时误触按钮 btnStart，单击时调用了 setEnabled(false)将按钮禁用，然后在 startProgress()方法里，进度条进度达到 100%以后再将其启用。

这段代码中用到了一个名为 runOnUiThread()的特殊方法，它是 Activity 类提供的用于在非 UI 线程中更新 UI。原因是在 Android 中，UI 控件只能在主线程中进行更新，而不能在其他线程（例如后台线程）中直接更新。否则 App 会出现闪退或者其他不可预料的错误（感兴趣的读者可以把代码中的两处 runOnUiThread()分别去掉，试试会出现什么情况）。

图 5.30　ProgressBar 运行结果示意图

因此，尽管 startProgress()方法定义了一个后台线程，使用 runOnUiThread()仍然是必要的，以确保 UI 的更新在主线程中进行。代码的第 1 处 runOnUiThread()使用了匿名内部类，第 2 处 runOnUiThread()使用 Lamda 表达式简化了代码。比较后不难发现，Lamda 表达式的代码更为简洁易懂。

关于线程的详细内容，请参阅第 9 章。

运行这个示例程序，单击界面上的“开始”按钮后，运行结果如图 5.30 所示。

除了长条形进度条以外，Android 还有一种环形进度条。这种进度条比较适合长时间执行且进度无法精确计算的应用场景。例如从数据库查询数据、图像处理操作等。

为了演示环形进度条的用法，我们在布局文件

activity_main.xml 文件中再增加一个 ProgressBar 控件，id 为 progressBar2，以及一个按钮控件，id 为 btnStart2。在 XML 文件中，将新增加的 ProgressBar 控件置为不可见(android:visibility="invisible")，相应的 XML 代码如下：

```
<ProgressBar
   android:id="@+id/progressBar2"
   style="?android:attr/progressBarStyle"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:visibility="invisible"/>
<Button
   android:id="@+id/btnStart2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="开始"
   android:layout_gravity="center" />
```

然后，在 MainActivity 类中增加这两个控件的相关定义，具体代码如下：

```
private ProgressBar pbProgressBar2;
private Button btnStart2;
```

以及相应的处理代码如下：

```
pbProgressBar2 = findViewById(R.id.progressBar2);
btnStart2 = findViewById(R.id.btnStart2);
btnStart2.setOnClickListener(v->{
          pbProgressBar2.setVisibility(View.VISIBLE);
          btnStart2.setEnabled(false);
          startProgress2();
     });
```

单击 btnStart2 按钮以后，将进度条控件 pbProgressBar2 设置为可见，将 btnStart2 按钮设置为禁用，然后调用 startProgress2()方法，该方法的代码如下：

```
      private void startProgress2() {
     new Thread(new Runnable() {
         public void run() {
             SystemClock.sleep((int)(Math.random()*10000));
             runOnUiThread(()->{
                 btnStart2.setEnabled(true);
                 pbProgressBar2.setVisibility(View.INVISIBLE);
             });
         }
     }).start();
}
```

在该方法里，结合 SystemClock.sleep()方法和 Math.random()随机数生成方法模拟一个长时间运行的任务。当任务结束以后，将 btnStart2 按钮设置为可用，pbProgressBar2 设置为不可见。

运行上述程序，单击 btnStart2 按钮，运行结果如图 5.31 所示。

图 5.31　环形进度条运行示意图

除了上述两种进度条以外，Android 提供了其他多种进度条样式，如图 5.32 所示。感兴趣的读者可以逐一进行试验，在此就不再一一赘述了。

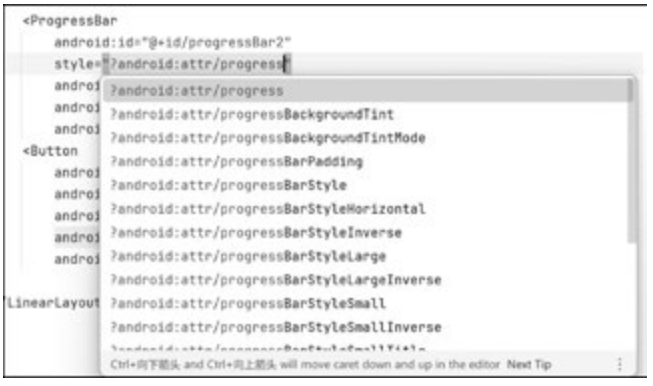

图 5.32　Android 系统提供的进度条类型

5.7　ListView

ListView 是 Android App 中一个常用的控件，用于在一个垂直滚动的列表中显示一系列的列表项(Item)，例如联系人、新闻列表等场景。ListView 中的每个列表项可以设置单击事件(OnItemClickListener)或者长单击事件(OnItemLongClickListener)。开发者可以自定义列表项的布局文件以实现相对复杂的 ListView 布局。如果 ListView 中对应的数据集合发生变化，开发者可以通过更新适配器(Adapter)来动态更新 ListView 中的项目。

关于 ListView 控件的常用属性，参见表 5.10。

表 5.10 ListView 控件的常用属性

属 性	功 能 描 述
android:divider	设置 ListView 的分隔线
android:dividerHeight	设置 ListView 的分隔线的高度
android:dividerPadding	设置分隔线与列表项的间距
android:choiceMode	设置列表项的选择模式，可以取值为 none(列表项不能选择，默认值)、singleChoice(只能选择一个列表项)、multipleChoice(可以选择多个列表项)或 multipleChoiceModal(可以选择多个列表项，且至少要选择一项)
android:scrollingCache	设置是否启用滚动缓存。启用后，可以提高 ListView 的流畅度，但会增加内存占用
android:fastScrollEnabled	设置是否启用快速滚动功能。启用后，可以提高 ListView 的用户体验

为了演示 ListView 控件的用法，我们新建一个项目 ListViewDemo，界面的 XML 代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< androidx. constraintlayout. widget. ConstraintLayout xmlns:android = "http://schemas. android.
com/apk/res/android"
    xmlns:app = "http://schemas. android. com/apk/res - auto"
    xmlns:tools = "http://schemas. android. com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:layout_marginTop = "30dp"
    tools:context = ". MainActivity">
    < ListView
        android: id = "@ + id/listView"
        android:layout_width = "match_parent"
        android:layout_height = "match_parent"
        app:layout_constraintTop_toTopOf = "parent"
        app:layout_constraintBottom_toBottomOf = "parent"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintEnd_toEndOf = "parent" />
</androidx. constraintlayout. widget. ConstraintLayout >
```

接下来，在 MainActivity 类中编写相应的代码：

```
public class MainActivity extends AppCompatActivity {
    private ListView lstvMenu;
    private String[ ] dishes = {"小炒肉", "回锅肉","青菜汤","凉拌黄瓜", "麻婆豆腐","夫妻
肺片","鸡蛋炒番茄"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super. onCreate(savedInstanceState);
        EdgeToEdge. enable(this);
        setContentView(R. layout. activity_main);
        lstvMenu = findViewById(R. id. listView);
        ArrayAdapter < String > adapter = new ArrayAdapter <>(this, android. R. layout. simple_
list_item_1, dishes);
        lstvMenu. setAdapter(adapter);
```

```
        lstvMenu.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                Toast.makeText(MainActivity.this, "你单击了第" + (position + 1) + "项,菜
肴是:" + dishes[position], Toast.LENGTH_SHORT).show();   //显示 Toast 消息
            }
        });
    }
}
```

在 MainActivity 类中,首先定义了一个 lstvMenu 控件对象变量,以便于在 onCreate()方法中通过 findViewById()方法及控件 id 找到界面上的 ListView 控件。另外,还定义了一个字符串数组 dishes,用于存储 ListView 中显示的菜肴名称。

在 ListView 中显示列表项时,不能直接将数据传递给 ListView,而是需要通过一个适配器(Adapter)。适配器充当数据与 ListView 之间的桥梁,负责将数据集合映射到视图组件上,如图 5.33 所示。

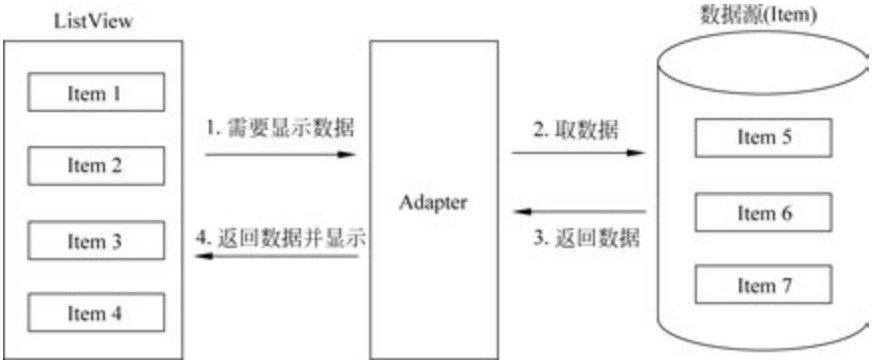

图 5.33　Adapter、ListView 和数据源之间的关系

从图 5.33 可以看出,ListView 是用于显示数据的视图容器,而 Adapter 负责从数据源中获取数据,适配成 ListView 可以展示的形式提供给 ListView。这里的数据源可以是数组、集合、数据库查询结果等。

适配器的类型很多。其中,ArrayAdapter 是最简单的一种适配器,它能够将数据与 ListView 的每个列表项进行绑定。通过将数据集合传递给 ArrayAdapter,并将 ArrayAdapter 应用于 ListView,即可自动将数据逐个填充到 ListView 的每个列表项。

ArrayAdapter 提供了一系列方法来操作数据集合,例如添加、删除和修改数据。调用这些方法,可以方便地对 ListView 中显示的数据进行增删改操作,然后调用 notifyDataSetChanged()方法通知 ListView 更新显示。

这个示例程序中,核心的代码就是创建 ArrayAdapter 对象 adapter 的这一条。它使用泛型 ArrayAdapter<String>定义了适配器中的数据类型为 String。this 是当前的上下文,这里是 MainActivity。android.R.layout.simple_list_item_1 是 Android 自带的一个简单列表项布局,用于快速显示文本数据,省去了开发者定义 ListView 列表项布局文件的工作。dishes 是一个 String 类型的数组,是 ListView 的数据源。

创建好 adapter 对象以后,就可以使用 ListView 的 setAdapter()方法将其与具体的

ListView 关联起来。接下来，调用 setOnItemClickListener()方法设置 ListView 的单击事件监听器。当用户单击 ListView 的列表项时，该监听器会触发回调方法，执行相应的操作。

运行该示例程序，单击其中的列表项，运行结果如图 5.34 所示。

需注意，ArrayAdapter 适用于简单的列表场景。对于需要更复杂的自定义列表项布局、交互或数据操作，应使用自定义的适配器类来实现更灵活的功能。

下面将展示一个相对复杂的使用自定义列表项布局的例子。这个例子完成以后的运行效果如图 5.35 所示。从界面上看，每个列表项显示了一道菜肴的图片、名称和价格。

图 5.34 ListView 运行结果示意图一

图 5.35 ListView 运行结果示意图二

新建一个项目 ListViewDemo2，主界面的 XML 代码与前面一样。新建一个列表项的布局文件 item_dish.xml，内容如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <ImageView
        android:id="@+id/imgDish"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:padding="6dp"/>
    <TextView
        android:id="@+id/txtName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
```

```
            android:textSize = "16sp"
            android:padding = "8dp"/>
        < TextView
            android:id = "@ + id/txtPrice"
            android:textStyle = "italic"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:textSize = "20sp"
            android:padding = "10dp"
            android:layout_gravity = "end"/>
    </LinearLayout >
```

该布局是一个水平线性布局，里面有三个控件：一个 ImageView 和两个 TextView，分别对应要显示的菜肴的图片、名称和价格。基于菜肴的这三个信息，新建一个 Dish 类，定义一个构造方法将菜肴的三个信息传入，完成初始化工作。完整的代码如下：

```
public class Dish {
    private int imageResId;                          //菜肴图片的 ID
    private String name;                             //菜肴图片名称
    private float price;                             //菜肴价格
    public Dish(int imageResId, String name,float price) {
        this.imageResId = imageResId;
        this.name = name;
        this.price = price;
    }
    public int getImageResId() {
        return imageResId;
    }
    public String getName() {
        return name;
    }
    public float getPrice() {
        return price;
    }
}
```

另外，再自定义一个 DishAdapter 类，该类继承自 ArrayAdapter < Dish >类，用于创建一个 Adapter 适配器。DishAdapter 类用于将 Dish 对象的数据与界面上的 ListView 进行绑定。创建 DishAdapter 对象并设置为 ListView 的适配器后，DishAdapter 适配器负责将数据源中的 Dish 对象绑定到 ListView 的列表项上，从而在 ListView 中显示 Dish 对象的数据。DishAdapter 类的代码如下：

```
public class DishAdapter  extends ArrayAdapter < Dish > {
    private Context context;
    private int layoutResourceId;
    private List < Dish > data = null;
    public DishAdapter(Context context, int layoutResourceId, List < Dish > data) {
        super(context, layoutResourceId, data);
        this.layoutResourceId = layoutResourceId;
        this.context = context;
```

```
            this.data = data;
        }
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            View row = convertView;
            ItemHolder holder = null;
            if (row == null) {
                LayoutInflater inflater = LayoutInflater.from(context);
                row = inflater.inflate(layoutResourceId, parent, false);
                holder = new ItemHolder();
                holder.imgDish = row.findViewById(R.id.imgDish);
                holder.txtName = row.findViewById(R.id.txtName);
                holder.txtPrice = row.findViewById(R.id.txtPrice);
                row.setTag(holder);
            } else {
                holder = (ItemHolder) row.getTag();
            }
            Dish item = data.get(position);
            holder.imgDish.setImageResource(item.getImageResId());
            holder.txtName.setText(item.getName());
            holder.txtPrice.setText(String.valueOf(item.getPrice()));
            return row;
        }
        static class ItemHolder {
             ImageView imgDish;
             TextView txtName;
             TextView txtPrice;
        }
    }
```

在 DishAdapter 类中，定义了三个成员变量，包括一个 Context 上下文对象变量、一个布局资源 ID(layoutResourceId)和一个 Dish 对象的 List 队列(data)。然后在构造方法里接收这些参数完成初始化工作。

DishAdapter 类重写了 getView()方法，这是 Adapter 类中的一个重要方法，在显示(渲染)每个列表项时被调用，为 ListView 的列表项提供自定义的视图。在这个方法中，首先判断是否有可重用的视图 convertView，如果没有则通过 LayoutInflater 从布局资源 layoutResourceId(实际执行时传入列表项的布局文件)中创建一个新的视图 row，作为列表项的视图容器。然后，定义了一个 ItemHolder 对象变量 holder。接下来通过 findViewById()方法找到视图中的 ImageView 和 TextView 控件，将对应的图片 id、菜肴名称和价格存储到 holder 中，再调用 setTag()方法将 holder 对象存入 Tag 标签中，以便在下次重用时通过调用 getTag()方法直接获取到该对象。如果 convertView 不为空，则说明表示有可重用的视图，直接从 convertView 的 tag 中通过 getTag()方法获取之前保存的 ItemHolder 对象。

接下来，根据 position 参数从数据源 data 中获取指定位置对应的 Dish 对象存入 item，并将 item 的数据设置到 holder 对象中。最后，返回视图 row。

ItemHolder 是一个静态内部类，它用于存储视图中的控件。这样可以避免在每次调用 getView()方法时通过相对较为耗时的 findViewById()方法来查找控件，提高 ListView 的显示性能。

MainActivity 的代码如下：

```
public class MainActivity extends AppCompatActivity {
    private ListView lstvMenu;
    private DishAdapter dishAdapter ;
    private List < Dish > dishList;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        lstvMenu = findViewById(R.id.listView);              //获取 ListView 实例
        dishList = createDishList ();                       //创建数据源
        dishAdapter = new DishAdapter(this, R.layout.item_dish, dishList); //创建适配器并设置
        lstvMenu.setAdapter(dishAdapter );
        lstvMenu.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView <?> parent, View view, int position, long id) {
                //生成一个 Toast
                Dish dish = dishList.get(position);
                Toast.makeText(MainActivity.this, "你单击了第" + (position + 1) + "项,菜
肴是: " + dish.getName() + ",价格是:" + dish.getPrice() + "元", Toast.LENGTH_SHORT).show();
            }
        });
    }
   private List < Dish > createDishList() {
        List < Dish > dishes = new ArrayList <>();
        dishes.add(new Dish(R.drawable.image1, "油爆大虾",18));
        dishes.add(new Dish(R.drawable.image2, "欧芹烩饭",21.5f));
        dishes.add(new Dish(R.drawable.image3, "凉拌沙拉",50));
        dishes.add(new Dish(R.drawable.image4, "炒空心菜",18.8f));
        dishes.add(new Dish(R.drawable.image5, "时蔬拼盘",30));
       //添加更多菜品…
        return dishes;
    }
}
```

这段代码首先定义了一个 ListView 对象变量 lstvMenu，用于引用主界面布局文件中的 ListView，还定义了一个自定义适配器类 DishAdapter 的对象变量 dishAdapter，作为 ListView 的适配器。接下来定义了一个 Dish 类的对象队列 dishList，其中包含了菜单项的图片、名称和价格等信息。在 onCreate()方法里调用自定义方法 createDishList()将一些菜肴信息加入数据源 dishList 中。

然后，定义一个 DishAdapter 对象变量，并将其实例化。在实例化该对象时，传入三个参数：当前上下文(this)，列表项布局文件的资源 ID(R. layout. listview_item)，以及数据源(dishList)。这样就将适配器、列表项布局文件和数据源进行了关联。再使用 ListView 的 setAdapter()方法将 DishAdapter 对象与具体的 ListView 关联起来。

最后，调用 setOnItemClickListener()方法来设置 ListView 的列表项单击事件监听器。当用户单击某一列表项时，会弹出一个 Toast 消息，显示相关信息，包括菜肴图片、菜名和价格。运行结果如图 5.35 所示。

5.8 GridView

在 Android 中，GridView 是一种以网格的形式显示项目的布局方式，并且可以滚动浏览显示的内容。网格中的每个单元（cell）格可以是图片、文本或其他自定义视图。和 ListView 一样，GridView 中的每个列表项可以设置单击事件或者长单击事件。开发者还可以自定义单元格的布局文件实现相对复杂的 GridView 布局。如果 GridView 中对应的数据集合发生变化，开发者还可以通过更新 Adapter 适配器来动态更新 GridView 的内容。

关于 GridView 控件的常用属性，参见表 5.11。

表 5.11 GridView 控件的常用属性

属 性	功 能 描 述
android:numColumns	设置 GridView 每行的列数
android:horizontalSpacing	设置网格单元的两列之间的间距
android:verticalSpacing	设置网格单元的两行之间的间距
android:columnWidth	设置网格单元的列宽
android:rowHeight	设置网格单元的行高
android:gravity	设置网格单元的对齐方式，默认是左上角对齐

为了演示 GridView 控件的用法，我们新建一个项目 GridViewDemo，界面的 XML 代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< androidx.constraintlayout.widget.ConstraintLayout xmlns:android = "http://schemas.android.
com/apk/res/android"
    xmlns:app = "http://schemas.android.com/apk/res - auto"
    xmlns:tools = "http://schemas.android.com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:layout_marginTop = "40dp"
    tools:context = ".MainActivity">
    < GridView
        android:id = "@ + id/gridView"
        android:layout_width = "match_parent"
        android:layout_height = "match_parent"
        android:numColumns = "3"
        android:background = "#999999"
        android:horizontalSpacing = "1dp"
        android:verticalSpacing = "1dp"
        android:padding = "2dp"
        app:layout_constraintBottom_toBottomOf = "parent"
        app:layout_constraintEnd_toEndOf = "parent"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent" />
</androidx.constraintlayout.widget.ConstraintLayout >
```

在上述布局文件中，设置 GridView 的列数为 3。然后再新建一个 GridView 单元格的布局文件 item_dish.xml，内容如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:background="@color/white">
    <ImageView
        android:id="@+id/imgDish"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:layout_gravity="center"
        android:padding="6dp"/>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
    <TextView
            android:id="@+id/txtName"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:textSize="16sp"
            android:textStyle="bold"
            android:layout_marginStart="6dp"
            android:textAlignment="viewStart" />
        <TextView
            android:id="@+id/txtPrice"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="16sp"
            android:layout_marginEnd="6dp"
            android:textStyle="italic"
            android:textAlignment="viewEnd" />
    </LinearLayout>
</LinearLayout>
```

该布局是一个垂直线性布局，里面有三个控件：ImageView 和两个 TextView，分别对应要显示的菜肴的图片、名称和价格。菜肴的 Java 类 Dish 代码沿用 5.7 节，保持不变。

另外，适配器类也可以沿用 5.7 节定义的 DishAdapter 类。只需稍作调整，将代码中定义的 View 对象 row 的名称换成 cell(换了便于理解代码，不换也可以正常运行)。DishAdapter 类用于将 Dish 对象的数据与 GridView 进行绑定。通过创建 DishAdapter 对象，并将其设置为 GridView 的适配器，由适配器负责将数据源中的 Dish 对象逐个绑定到 GridView 的每个单元上，就可以将 Dish 对象的数据显示在 GridView 中。

接下来，在 MainActivity 中编写相应的代码：

```
public class MainActivity extends AppCompatActivity {
    private GridView gridView;
    private DishAdapter adapter;
    private List<Dish> dishList;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        gridView = findViewById(R.id.gridView);                //获取 GridView 实例
        dishList = createDishList();                           //创建数据源
        adapter = new DishAdapter(this,R.layout.item_dish, dishList); //创建适配器并设置
        gridView.setAdapter(adapter);
        gridView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                //生成一个 Toast 消息
                Dish item = dishList.get(position);
                Toast.makeText(MainActivity.this, "你单击了第" + (position + 1) + "项,菜
肴是: " + item.getName() + ",价格是:" + item.getPrice() + "元", Toast.LENGTH_SHORT).show();
            }
        });
    }
    private List<Dish> createDishList() {
        List<Dish> dishes = new ArrayList<>();
        dishes.add(new Dish(R.drawable.image1, "油爆大虾",18));
        dishes.add(new Dish(R.drawable.image2, "欧芹烩饭",21.5f));
        dishes.add(new Dish(R.drawable.image3, "凉拌沙拉",50));
        dishes.add(new Dish(R.drawable.image4, "炒空心菜",18.8f));
        dishes.add(new Dish(R.drawable.image5, "时蔬拼盘",30));
        //添加更多菜品…
        return dishes;
    }
}
```

将此 MainActivity 类与 5.7 节的 ListViewDemo2 项目的 MainActivity 类进行对比不难发现，两个类几乎一模一样，区别仅在于主界面上对应的 View 对象的定义和名称有所不同。

综上不难看出，ListView 和 GridView 的功能很相似。原因是这两个类都继承自抽象类 AbsListView。该类提供了基本的 ListView 和 GridView 实现，用于显示大量数据并支持滚动浏览，有很多共同的属性和方法。因此，在使用这两个类时，代码大同小异也就不奇怪了。

这个示例代码复用了 5.7 节的很多代码，充分展现了面向对象程序设计的代码复用带来的好处。例如，可以用本节的布局文件替换 ListViewDemo2 项目的布局文件，再更改 ListViewDemo2 项目的 lvstMenu 的定义为 GridView lstvMenu。然后将 findViewById()方法的 id 对应到 GridView 的 id。程序一样可以运行。这充分展现了 Android 程序开发采用 MVC 架构模式，功能代码与界面设计分离带来的便利性。感兴趣的读者可以对此进行实验。

这里不再详细解读这些代码的功能，大家可以参照 5.7 节的内容来理解学习。

运行该示例代码，运行结果如图 5.36 所示。

图 5.36　GridView 运行结果示意图

5.9 RecyclerView

RecyclerView是Android开发中常用的一个控件，用于展示大量数据，并支持各种交互操作。它相比于传统的ListView、GridView功能更为强大，具有更好的灵活性和性能优势，表现在以下3个方面。

(1) RecyclerView具有更好的内存管理和性能优化，可以实现更流畅的滚动和更好的用户体验。

(2) RecyclerView提供了更灵活的布局管理器，可以实现各种复杂的布局效果。

(3) RecyclerView支持添加和删除动画、拖曳和滑动删除等交互效果，用户体验更好。

这些特点都是ListView和GridView所不具备的。因此，官方文档中明确建议用RecyclerView来代替ListView和GridView。虽然，开发者依然可以使用ListView和GridView，但对于新建的项目而言，推荐使用RecyclerView来代替这两个控件。

关于Recyclerview控件的常用属性，参见表5.12。

表5.12 RecyclerView控件的常用属性

属 性	功 能 描 述
android:orientation	设置RecyclerView的布局方向，水平或者垂直
android:scrollbars	设置是否显示滚动条，可以取值为none(没有滚动条)、horizontal(显示水平滚动条)和vertical(显示垂直滚动条)
app:layoutManager	设置RecyclerView的布局管理器，可以是LinearLayoutManager(线性布局管理器)、GridLayoutManager(网格布局管理器)或者StaggeredGridLayoutManager(流式布局管理器)
app:spanCount	设置GridLayoutManager或StaggeredGridLayoutManager的行数或列数

5.9.1 用RecyclerView实现ListView的功能

为了演示RecyclerView控件的用法，我们新建一个项目RecyclerViewDemo，实现和之前5.7节ListView同样的功能：显示菜品的图片、名称和价格信息，界面的XML代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< androidx. constraintlayout. widget. ConstraintLayout xmlns:android = "http://schemas. android.
com/apk/res/android"
    xmlns:app = "http://schemas. android. com/apk/res - auto"
    xmlns:tools = "http://schemas. android. com/tools"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:layout_marginTop = "30dp"
    tools:context = ". MainActivity">
 < androidx. recyclerview. widget. RecyclerView
        android:id = "@ + id/recyclerView"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        app:layout_constraintBottom_toBottomOf = "parent"
        app:layout_constraintEnd_toEndOf = "parent"
```

```
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent"/>
</androidx.constraintlayout.widget.ConstraintLayout >
```

列表项的布局文件沿用 5.7 节定义的 item_dish.xml。菜品信息类 Dish 也沿用 5.7 节定义的类。大家可以在 Android Studio 中或者 Windows 文件资源管理器里，把这两个文件复制到 RecyclerViewDemo 项目文件夹下的相应位置：item_dish.xml 复制到 layout 下，Dish 类复制到 MainActivity 类所在的文件夹下即可。

另外，再自定义一个 DishAdapter 类，这个类比起 ListView 示例代码中定义的 DishAdapter 类要稍微复杂一些。该类继承自静态抽象类 RecyclerView.Adapter 类，功能是创建一个在 RecyclerView 展示数据用的适配器(Adapter)。

RecyclerView.Adapter 提供了以下两个重要的方法。

(1) onCreateViewHolder(ViewGroup parent, int viewType)：用于创建 ViewHolder 对象。

(2) onBindViewHolder(ViewHolder holder, int position)：将数据与视图绑定，更新 ViewHolder 中的视图内容。

DishAdapter 类用于将 Dish 对象的数据与界面上的 RecyclerView 进行绑定。通过创建 DishAdapter 对象，并将其设置为 RecyclerView 的适配器，由适配器负责将数据源中的 Dish 对象逐个绑定到 RecyclerView 的每个列表项，就可以实现将 Dish 对象的数据显示在 RecyclerView 中。

DishAdapter 类的代码如下：

```
public class DishAdapter extends RecyclerView.Adapter < DishAdapter.DishViewHolder > {
     private List < Dish > dishList;
     private OnItemClickListener listener;
     public DishAdapter(List < Dish > dishList, OnItemClickListener listener) {
          this.dishList = dishList;
          this.listener = listener;
    }
    @NonNull
    @Override
    public DishViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
         View view = LayoutInflater.from(parent.getContext()).inflate (R.layout.item_dish,
parent, false);
         return new DishViewHolder(view,listener,dishList);
    }
    @Override
    public void onBindViewHolder(@NonNull DishViewHolder holder, int position) {
         Dish dish = dishList.get(position);

         holder.imgView.setImageResource(dish.getImageResId());
         holder.txtName.setText(dish.getName());
         holder.txtPrice.setText(String.valueOf(dish.getPrice()));
    }
    @Override
    public int getItemCount() {
```

```
            return dishList.size();
        }
        public interface OnItemClickListener {
            void onItemClick(Dish dish,int position);
        }
        public static class DishViewHolder extends RecyclerView.ViewHolder {
            ImageView imgView;
            TextView txtName;
            TextView txtPrice;
            public DishViewHolder(@NonNull View itemView, final OnItemClickListener listener,
    final List<Dish> dishList) {
                super(itemView);
                imgView = itemView.findViewById(R.id.imgDish);
                txtName = itemView.findViewById(R.id.txtName);
                txtPrice = itemView.findViewById(R.id.txtPrice);
                itemView.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        if (listener != null) {
                            int position = getAdapterPosition();
                            if (position != RecyclerView.NO_POSITION) {
                      listener.onItemClick(dishList.get(position),position);
                            }
                        }
                    }
                });
            }
        }
    }
```

在 DishAdapter 类中，定义了两个成员变量，包含一个 Dish 对象的 List 队列 dishList 和一个 OnItemClickListener 对象变量 listener。然后在构造方法接收这两个参数完成初始化工作。

DishAdapter 类重写了 onCreateViewHolder()方法。该方法是 RecyclerView.Adapte 类中的一个重要方法，用于为 RecyclerView 的每个列表项提供自定义的视图，通过 LayoutInflater 加载 R.layout.item_dish 布局文件以创建一个视图，并将其与 dishList、listener 一起作为参数传递给 DishViewHolder 的构造方法，创建一个新的视图持有者 DishViewHolder。DishViewHolder 是 DishAdapter 类中包含的一个静态内部类，继承自 RecyclerView.ViewHolder 类，用于存储列表项的视图。它包含了菜品的图片、名称和价格的 ImageView 和 TextView 控件：imgView、txtName 和 txtPrice。在 DishViewHolder 类的构造方法中，设置了列表项的单击事件监听器，并在单击事件发生时调用监听器的 onItemClick()方法，将单击的菜品对象和位置作为参数传递给监听器。

在 DishAdapter 类的 onBindViewHolder()方法中，根据列表项的位置获取对应的菜品对象，并将菜品的图片、名称和价格设置到视图持有者 DishViewHolder 的对应控件中。另外，DishAdapter 类中还定义了一个内部接口 OnItemClickListener，其中包含一个 onItemClick 方法，用于处理列表项的单击事件。

MainActivity类中需要编写RecyclerView相关的代码，并实现DishAdapter.OnItemClickListener接口，该类完整的代码如下：

```
public class MainActivity extends AppCompatActivity implements DishAdapter.OnItemClickListener {
    private RecyclerView recyclerView;
    private DishAdapter dishAdapter;
    private List<Dish> dishList;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        recyclerView = findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        //添加分隔线
        recyclerView.addItemDecoration(new DividerItemDecoration(this, LinearLayout.
VERTICAL));
        dishList = createDishList();                    //创建菜品列表数据
        dishAdapter = new DishAdapter(dishList, this);//创建适配器并设置给RecyclerView
        recyclerView.setAdapter(dishAdapter);
    }
    private List<Dish> createDishList() {
        List<Dish> dishes = new ArrayList<>();
        dishes.add(new Dish(R.drawable.image1, "油爆大虾",18));
        dishes.add(new Dish(R.drawable.image2, "欧芹烩饭",21.5f));
        dishes.add(new Dish(R.drawable.image3, "凉拌沙拉",50));
        dishes.add(new Dish(R.drawable.image4, "炒空心菜",18.8f));
        dishes.add(new Dish(R.drawable.image5, "时蔬拼盘",30));
        //添加更多菜品…
        return dishes;
    }
    public void onItemClick(Dish dish,int position) {
        Toast.makeText(MainActivity.this, "你单击了第" + (position + 1) + "项,菜肴是:
" + dish.getName() + ",价格是:" + dish.getPrice() + "元", Toast.LENGTH_SHORT).show();
    }
}
```

在MainActivity类中，定义了RecyclerView对象变量recyclerView，用于引用主界面布局文件中的RecyclerView控件；定义了适配器类DishAdapter的对象变量dishAdapter，作为RecyclerView的适配器；使用泛型和List定义了一个ArrayList类型的List集合dishList，用于存储一组菜肴的数据。

recyclerView对象调用setLayoutManager()方法将RecyclerView的布局管理器设置为默认的垂直方向的LinearLayoutManager。然后调用addItemDecoration()方法添加垂直方向的分隔线。

接下来通过自定义方法createDishList()创建了一个Dish对象的队列dishList，其中包含了菜单项的图片、名称和价格等信息。然后，创建一个DishAdapter实例。在实例化DishAdapter时，传入两个参数：数据源dishList和上下文this。这样就将适配器与数据源和列表项布局文件进行了关联。再使用RecyclerView的setAdapter()方法将其与具体的

RecyclerView 关联起来。

最后，实现 DishAdapter. OnItemClickListener 接口中定义的 onItemClick()方法来处理 RecyclerView 中的列表项被单击的事件。当用户单击某一列表项时，会弹出一个 Toast 消息，显示相应的菜肴名称和价格。

程序运行结果如图 5.37 所示。

图 5.37　RecyclerView 运行结果示意图一

5.9.2　用 RecyclerView 实现 GridView 的功能

为了演示 RecyclerView 控件的用法，我们新建一个项目 RecyclerViewDemo2，实现和之前 5.8 节 GridView 同样的功能：显示菜品的图片、名称和价格信息，界面的 XML 代码与 5.8.1 节完全相同，不需要修改，可以将其直接复制过来。

单元格的布局文件沿用 5.7 节定义的 item_dish. xml 保持不变。菜品信息类 Dish 也沿用 5.7 节定义的类。DishAdapter 类则沿用 5.8.1 节的代码。可以在 Android Studio 下或者 Windows 文件资源管理器把这三个文件复制到项目文件夹下的相应位置：item_dish. xml 复制到 layout 下，Dish 类和 DishAdapter 类复制到 MainActivity 类所在的文件夹下即可。

MainActivity 类中需要编写 RecyclerView 相关的代码，并实现 DishAdapter. OnItemClick Listener 接口，完整的代码如下：

```
public class MainActivity extends AppCompatActivity implements DishAdapter.OnItemClickListener {
    private RecyclerView recyclerView;
    private DishAdapter dishAdapter;
```

```
    private List<Dish> dishList;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        recyclerView = findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new GridLayoutManager(this, 3));
        //添加分隔线
        recyclerView.addItemDecoration(new DividerItemDecoration(this, LinearLayout.
VERTICAL));
        recyclerView.addItemDecoration(new DividerItemDecoration(this, LinearLayout.
HORIZONTAL));
        dishList = createDishList();                    //创建菜品列表数据
        dishAdapter = new DishAdapter(dishList, this);//创建适配器并设置给 RecyclerView
        recyclerView.setAdapter(dishAdapter);
    }
    private List<Dish> createDishList() {
        List<Dish> dishes = new ArrayList<>();
        dishes.add(new Dish(R.drawable.image1, "油爆大虾",18));
        dishes.add(new Dish(R.drawable.image2, "欧芹烩饭",21.5f));
        dishes.add(new Dish(R.drawable.image3, "凉拌沙拉",50));
        dishes.add(new Dish(R.drawable.image4, "炒空心菜",18.8f));
        dishes.add(new Dish(R.drawable.image5, "时蔬拼盘",30));
        dishes.add(new Dish(R.drawable.image1, "油爆大虾",18));
        dishes.add(new Dish(R.drawable.image2, "欧芹烩饭",21.5f));
        dishes.add(new Dish(R.drawable.image3, "凉拌沙拉",50));
        dishes.add(new Dish(R.drawable.image4, "炒空心菜",18.8f));
        dishes.add(new Dish(R.drawable.image5, "时蔬拼盘",30));
        //添加更多菜品…

        return dishes;
    }
    public void onItemClick(Dish dish,int position) {
        Toast.makeText(MainActivity.this, "你单击了第" + (position + 1) + "项,菜肴是:
" + dish.getName() + ",价格是:" + dish.getPrice() + "元", Toast.LENGTH_SHORT).show();
    }
}
```

MainActivity类的代码与5.8.1节的代码几乎一模一样。区别仅在于调用findViewById()方法获得recyclerView对象以后,接着调用setLayoutManager()方法将RecyclerView的布局管理器设置为GridLayoutManager。然后两次调用addItemDecoration()方法分别添加垂直方向和水平方向的分隔线。其他代码的功能就不在此赘述,可参考5.8.1节的叙述。

程序运行结果如图5.38所示。

对比RecyclerView、ListView和GridView的代码不难发现,使用RecyclerView可以调用setLayoutManager()方法来快速切换界面布局为ListView或者GridView,另外,它集成了ViewHolder,性能更优越。建议开发者在新项目中积极使用RecyclerView控件。

图 5.38　RecyclerView 运行结果示意图二

5.10　小结

本章介绍了 Android 开发的重要概念——事件处理机制的原理及应用，在本书的大部分示例代码中，我们都会看到事件处理代码的身影。它的重要性不言而喻，是每一个初学者应该重点学习的内容之一。

本章的另外一个重点是常用的界面控件的使用方法，对于每一个界面控件的示例代码，本书给出了完整的界面布局代码和详细的示例代码，帮助初学者尽快地掌握这些界面控件的使用。学习并掌握本章内容后，可以为已初步踏入 Android 开发的门槛。