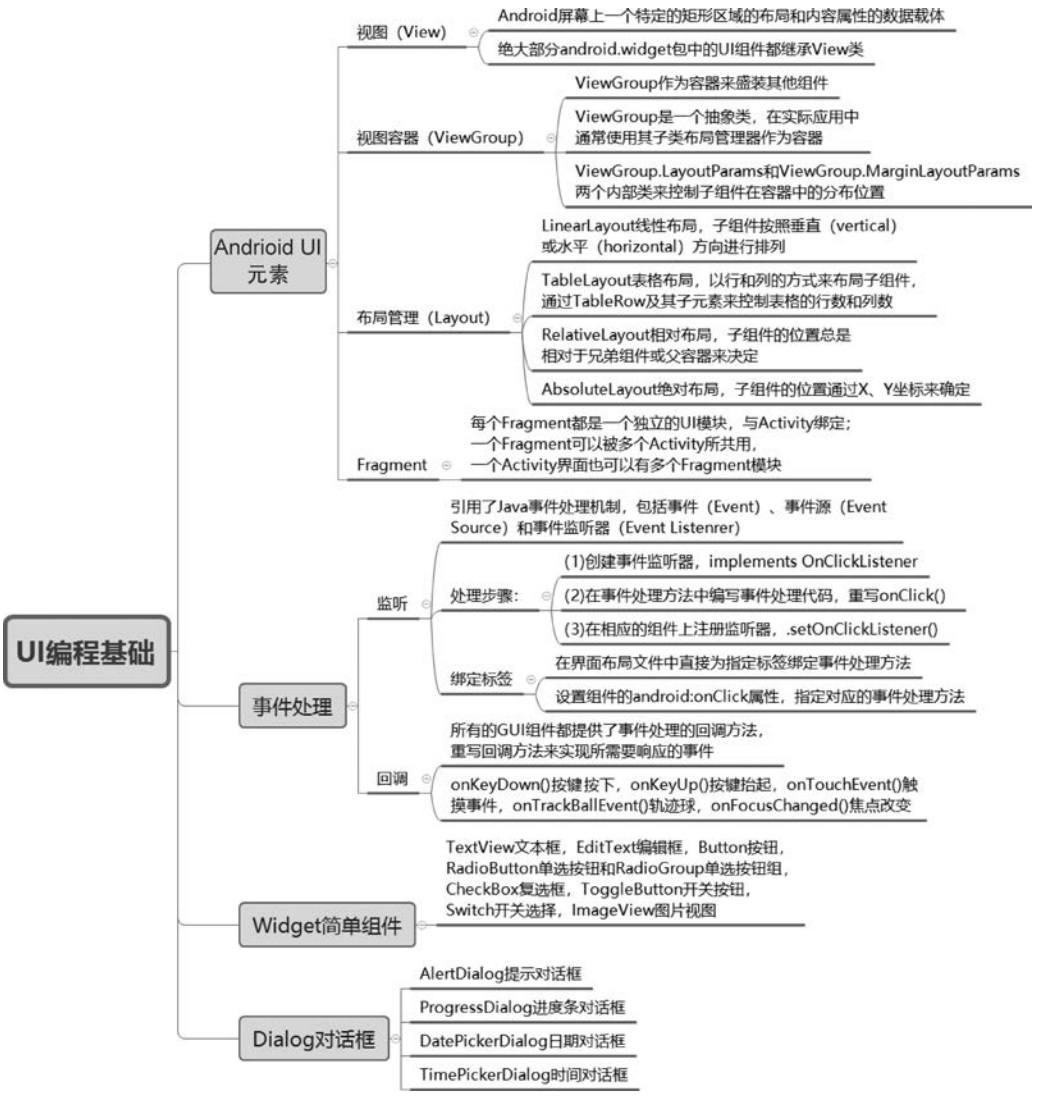
第3章 UI编程基础

本章思维导图



本章目标

- 了解 Android 中的 UI 元素；
- 能够使用布局管理器对界面进行管理；
- 掌握界面交互事件处理机制及实现步骤；
- 能够熟练使用常用的 Widget 简单组件；
- 掌握 Dialog 对话框的使用。

3.1 Android UI 元素

UI(User Interface,用户界面)设计是指对软件的人机交互、操作逻辑、界面美观的整体设计。良好的 UI 设计不仅是让软件变得更加人性化,还让软件的操作变得舒适、简单、自由、充分体现软件的定位和特点。Android 借鉴了 Java 中的 UI 设计思想,包括事件响应机制和布局管理,提供了丰富的可视化用户界面组件,例如,菜单、对话框、按钮和文本框等。

Android 中界面元素主要由以下几个部分构成。

- 视图(View):视图是所有可视界面元素(通常称为控件或小组件)的基类,所有 UI 控件都是由 View 类派生而来的。
- 视图容器(ViewGroup):视图容器是视图类的扩展,其中包含多个子视图。通过扩展 ViewGroup 类,可以创建由多个相互连接的子视图所组成的复合控件,还可以创建布局管理器从而实现 Activity 中的布局。
- 布局管理(Layout):布局管理器是由 ViewGroup 派生而来,用于管理组件的布局格式,组织界面中组件的呈现方式。
- Activity:用于为用户呈现窗口或屏幕,当程序需要显示一个 UI 界面时,需要为 Activity 分配一个视图(通常是一个布局或 Fragment)。
- Fragment:Fragment 是 Android 3.0 引入的新 API,代表了 Activity 的子模块,即 Activity 片段(Fragment 本身就是片段的意思)。Fragment 可用于 UI 的各个部分,特别适合针对不同屏幕尺寸时,优化 UI 布局以及创建可重用的 UI 元素。每个 Fragment 都包含自己的 UI 布局,并接受相应的输入事件,但使用时必须与 Activity 紧密绑定在一起(Fragment 必须嵌入到 Activity 中)。

因此,一个复杂的 Android 界面设计往往需要不同的组件组合才能实现,有时需要对这些标准视图进行扩展或者修改从而提供更好的用户体验。

3.1.1 视图

View 视图组件是用户界面的基础元素,View 对象是 Android 屏幕上一个特定的矩形区域的布局和内容属性的数据载体,通过 View 对象可实现布局、绘图、焦点变换、滚动条、屏幕区域的按键、用户交互等功能。Android 应用的绝大部分 UI 组件都放在 android.widget 包及其子包中,所有这些 UI 组件都继承了 View 类。View 的常见子类及功能如表 3-1 所示。

表 3-1　View 类的主要子类

类　名	功能描述	类　名	功能描述
TextView	文本视图	DigitalClock	数字时钟
EditText	编辑文本框	AnalogClock	模拟时钟
Button	按钮	ProgessBar	进度条
Checkbox	复选框	RatingBar	评分条
RadioGroup	单选按钮组	SeekBar	搜索条
Spinner	下拉列表	GridView	网格视图
AutoCompleteTextView	自动完成文本框	ListView	列表视图
DataPicker	日期选择器	ScrollView	滚动视图
TimePicker	时间选择器		

注意　本章后续各节将对上述 View 组件进行重点讲解。

3.1.2　视图容器

View 类还有一个非常重要的 ViewGroup 子类，该类通常作为其他组件的容器使用。View 组件可以添加到 ViewGroup 中，也可以将一个 ViewGroup 添加到另一个 ViewGroup 中。Android 中的所有 UI 组件都是建立在 View、ViewGroup 基础之上，Android 采用了“组合器”模式来设计 View 和 ViewGroup；其中，ViewGroup 是 View 的子类，因此 ViewGroup 可以当成 View 来使用。对于一个 Android 应用的图形 UI 而言，ViewGroup 又可以作为容器来盛装其他组件；ViewGroup 不仅可以包含普通的 View 组件，还可以包含其他 ViewGroup 组件。Android 图形 UI 的组件层次如图 3-1 所示。

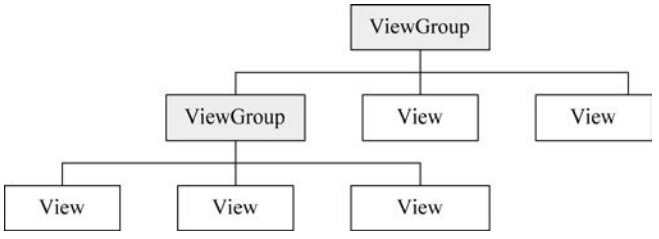


图 3-1　UI 组件层次

注意　图 3-1 来自 Android 开发文档，对于每个 Android 程序员而言，Android 提供的官方文档都需要仔细阅读。

ViewGroup 类提供的主要方法如表 3-2 所示。

表 3-2　ViewGroup 类的方法功能

方　法	功能描述
ViewGroup()	构造方法
void addView(View child)	用于添加子视图，以 View 作为参数，将该 View 增加到当前视图组中
removeView(View view)	将指定的 View 从视图组中移除

续表

方　法	功 能 描 述
updateViewLayout(View view, ViewGroup.LayoutParams params)	用于更新某个 View 的布局
void bringChildToFront(View child)	将参数所指定的视图移动到所有视图之前显示
boolean clearChildFocus(View child)	清除参数所指定的视图的焦点
boolean dispatchKeyEvent(KeyEvent event)	将参数所指定的键盘事件分发给当前焦点路径的视图。当分发事件时，按照焦点路径来查找合适的视图。若本视图为焦点，则将键盘事件发送给自己；否则发送给焦点视图
boolean dispatchPopulateAccessibilityEvent(AccessibilityEvent event)	将参数所指定的事件分发给当前焦点路径的视图
boolean dispatchSetSelected(boolean selected)	为所有的子视图调用 setSelected()方法

注意 ViewGroup 继承了 View 类，虽然可以当成普通的 View 来使用，但习惯上将 ViewGroup 当容器来使用。由于 ViewGroup 是一个抽象类，在实际应用中通常使用 ViewGroup 的子类作为容器，例如，各种布局管理器。

1. ViewGroup 继承结构

ViewGroup 的继承者大部分位于 android.widget 包中，其直接子类包括 AdapterView、AbsoluteLayout、FrameLayout、LinearLayout 和 RelativeLayout 等。以上直接子类又分别具有子类，ViewGroup 继承者的体系结构如图 3-2 所示。

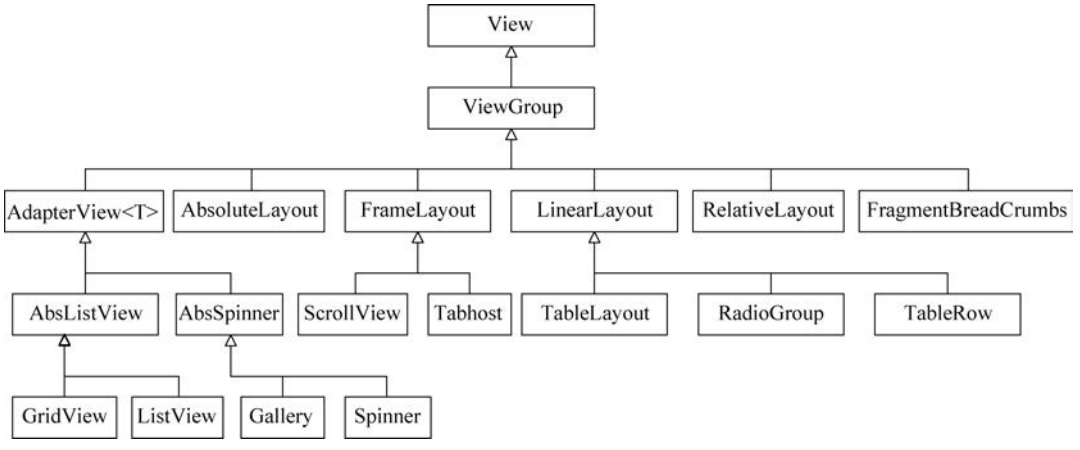


图 3-2　ViewGroup 继承者的体系结构

如图 3-2 所示，ViewGroup 直接子类均可作为容器来使用，这些类为子类提供不同的布局方法，用于设置子类之间的位置和尺寸关系。ViewGroup 类的间接子类中，有些不能作为容器来使用，仅能当作普通的组件来使用。

2. 布局参数类

在 Android 布局文件中，每个组件所能使用的 XML 属性有以下三类。

- 组件本身的 XML 属性。
- 组件祖先类的 XML 属性。

- 组件所属容器的布局参数。

其中，布局参数是包含该组件的容器（如 ViewGroup 子类）所提供的参数。在 Android 中，ViewGroup 子类都有一个相应的{XXX}. LayoutParams 静态子类，用于设置子类所使用的布局方式。这些子类继承关系和 ViewGroup 子类的继承关系具有相似性。

ViewGroup 容器使用 ViewGroup. LayoutParams 和 ViewGroup. MarginLayoutParams 两个内部类来控制子组件在其中的分布位置，这两个内部类中都提供了一些 XML 属性，ViewGroup 容器中的子组件通过指定 XML 属性来控制组件的位置，如表 3-3 所示。

表 3-3　ViewGroup 子元素支持的属性

XML 属性	功 能 描 述
android:layout_width	设定该组件的子组件布局的宽度
android:layout_height	设定该组件的子组件布局的高度

android:layout_height 和 android:layout_width 属性都支持以下三个属性值。

- fill_parent 属性用于指定子组件的高度、宽度与父容器的高度、宽度相同。
- match_parent 与 fill_parent 的功能完全相同，从 Android 2.2 开始推荐使用该属性值来代替 fill_parent。
- wrap_content 属性用于指定子组件的大小恰好能包裹其内容即可。

注意　在实际应用中，除了为组件指定高度、宽度，还需要设置布局的高度、宽度，这是由 Android 的布局机制决定的。Android 组件的大小不仅由实际的宽度、高度控制，还由布局的高度、宽度控制。例如，一个组件的宽度为 30px，如果将其布局宽度设置为 match_parent，那么该组件的宽度将会被“拉宽”并占满其所在的父容器；如果将其布局宽度设为 wrap_cotent，那么该组件的宽度才会是 30px。

ViewGroup. MarginLayoutParams 用于控制子组件周围的页边距（即组件四周的留白），所支持的 XML 属性如表 3-4 所示。

表 3-4　MarginLayoutParams 支持的属性

XML 属性	功 能 描 述
android:layout_marginTop	指定该子组件上面的页边距
android:layout_marginRight	指定该子组件右面的页边距
android:layout_marginBottom	指定该子组件下面的页边距
android:layout_marginLeft	指定该子组件左面的页边距

注意　由于 LayoutParams 也具有继承关系，因此 LinearLayout 的子类除了可以使用 LinearLayout. LayoutParams 所提供的 XML 属性外，还可以使用其祖先类 ViewGroup. LayoutParams 的 XML 属性。

3.1.3　布局管理

针对不同的手机屏幕（如手机屏幕的分辨率不同或屏幕尺寸不同等情况），当在程序中手动控制每个组件的大小和位置时，将会给编程带来巨大的困难。为了解决这个问题，Android

提供了布局管理器，使得 Android 各类组件（如按钮、文本等组件）能够适应屏幕的变化。布局管理器可以根据运行平台来调整组件的大小，开发者只需为容器选择合适的布局管理器即可。

Android 的布局管理器本身是一种 UI 组件，所有的布局管理器都是 ViewGroup 的子类，Android 布局管理器类之间的关系如图 3-3 所示。

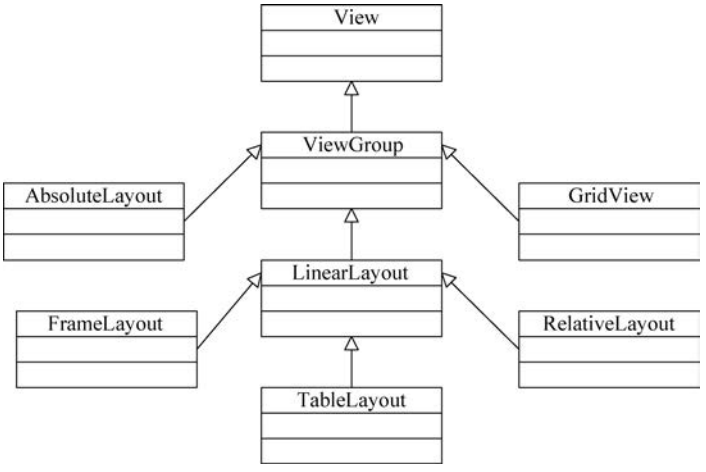


图 3-3 Android 布局管理器类之间的关系

所有布局都可以作为容器来使用，通过调用 addView()方法向布局管理器中添加组件。此外，布局管理器还继承了 View 类，在实际编程过程中可以把布局管理器作为普通的 UI 组件嵌套到其他布局管理器中。

Android 提供了多种布局，常用的布局有以下几种。

- LinearLayout（线性布局）：该布局中子元素之间成线性排列，即在水平或垂直方向上的顺序排列。
- RelativeLayout（相对布局）：该布局是一种根据相对位置排列元素的布局方式，允许子元素指定相对于其他元素或父元素的位置（一般通过 ID 指定）。在线性布局中排列子元素时，不需要特殊指定参照物，而相对布局中的子元素必须指定其参照物，只有指定参照物之后，才能定义该元素的相对位置。
- TableLayout（表格布局）：该布局将子元素的位置分配到表格的行或列中，即按照表格形式的顺序排列。一个表格布局中有多个“表格行”，而表格行中又包含多个“表格单元”。表格布局并不是真正意义上的表格，只是按照表格的方式组织元素的布局，元素之间并没有实际表格中的分界线。
- AbsoluteLayout 绝对布局：按照绝对坐标对元素进行布局。与相对布局不同，绝对布局不需要指定参照物，而是使用整个手机界面作为坐标系，通过坐标系的水平偏移量和垂直偏移量来确定其唯一位置。

3.1.4 Fragment

Fragment 允许将 Activity 拆分成多个完全独立的可重用的组件，每个组件具有自己的生命周期和 UI 布局。Fragment 最大的优点就是灵活地为不同大小屏幕的设备创建 UI 界面，例如，小屏幕的智能手机和大屏幕的平板电脑。

每个 Fragment 都是一个独立的模块，并与所绑定的 Activity 紧密地联系在一起。一个 Fragment 可以被多个 Activity 所共用，一个界面有可以有多个 UI 模块，对于像平板电脑的设备，Fragment 展现了很好的适应性和动态创建 UI 的能力，在一个 Activity 中可以添加、删除、更换 Fragment。Fragment 为不同型号、尺寸、分辨率的设备提供了统一的 UI 优化方案。

注意 有关 Fragment 的生命周期及详细使用方法参见第 5 章。

3.2 界面布局

Android 中提供了以下两种创建布局的方式。

- 在 XML 布局文件中声明：首先将需要显示的组件在布局文件中进行声明，然后在程序中通过 setContentView(R.layout.XXX)方法将布局呈现在 Activity 中。推荐使用此种方式，前面的程序也一直使用此种方式。
- 在程序中直接实例化布局及其组件：此种方式并不提倡使用，除非界面中的组件及布局需要动态改变才使用。

常见的 Android 布局包括 LinearLayout、RelativeLayout、TableLayout 和 AbsoluteLayout 等多种布局。

3.2.1 线性布局

LinearLayout 是一种线性排列的布局，布局中的组件按照垂直或者水平方向进行排列，排列方向是由 android:orientation 属性进行控制，其属性值包括垂直(vertical)和水平(horizontal)两种。LinearLayout 对应的类为 android.widget.LinearLayout。

LinearLayout 常用的 XML 属性及相关方法的说明如表 3-5 所示。

表 3-5 LinearLayout 常用的 XML 属性及对应方法

XML 属性	对应方法	功能描述
android:divider	setDividerDrawable()	设置垂直布局时两个按钮之间的分隔条
android:gravity	setGravity()	设置布局管理器内组件的对齐方式。该属性支持 top、bottom、left、right、center_vertical、fill_vertical、center_horizontal、fill_horizontal、center、fill、clip_vertical、clip_horizontal、start、end 几个属性值。也可以指定多种对齐方式的组合，例如，left|center_vertical 代表出现在屏幕左边，且垂直居中
android:orientation	setOrientation()	设置布局管理器内组件的排列方式，参数可以为 horizontal(水平排列)或 vertical(垂直排列，默认值)

此外，LinearLayout 中包含的所有子元素的位置都受 LinearLayout.LayoutParams 控制，LinearLayout 包含的子元素可以额外指定属性，如表 3-6 所示。

表 3-6 LinearLayout 子元素常用 XML 属性及说明

XML 属性	功能描述
android:layout_gravity	指定子元素在 LinearLayout 中的对齐方式
android:layout_weight	指定子元素在 LinearLayout 中所占的比重

注意 线性布局不会换行，当组件顺序排列到屏幕边缘时，剩余的组件不会被显示出来。

在项目的 res\layout 目录下创建一个线性布局文件 linearlayout.xml。如图 3-4 所示，打开 res 文件目录，右击 layout 文件夹，选择 New→XML→Layout XML File 命令，创建一个新的 XML 布局文件。

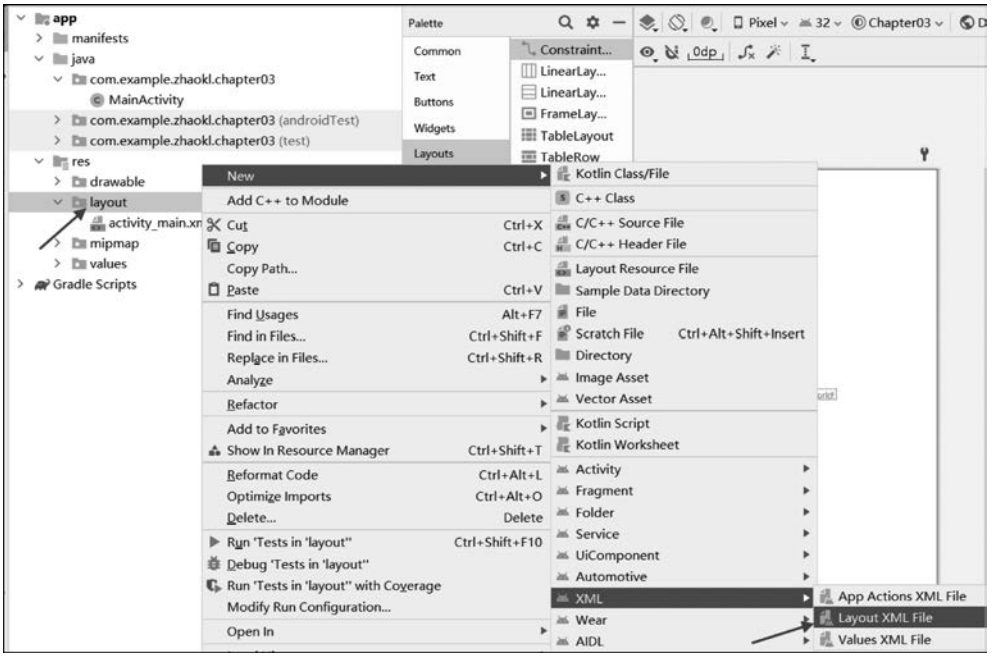


图 3-4 创建新的 XML 布局文件

LinearLayout 布局代码如下。

【案例 3-1】 linearlayout.xml

```
<?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_horizontal">
    < TextView
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "txtView1"
        android:textColor = "＃000000"
        android:textSize = "20sp"/>
    < TextView android:text = "txtView2" .../>
    < TextView android:text = "txtView3" .../>
    < TextView android:text = "txtView4" .../>
    < TextView android:text = " txtView5" .../>
</LinearLayout >
```

上述代码中，页面布局相对比较简单，仅定义了一个线性布局，并在布局中定义了 5 个 TextView；在定义线性布局时默认采用垂直排列方式，且所有组件在容器的顶部居中对齐。如图 3-5 所示，在 Android Studio 中编写、设计页面布局可以采用三种模式：Code 只显示左侧代码窗口；Split 以切分方式左侧显示代码，右侧显示设计窗口；Design 只显示右侧设计窗口。

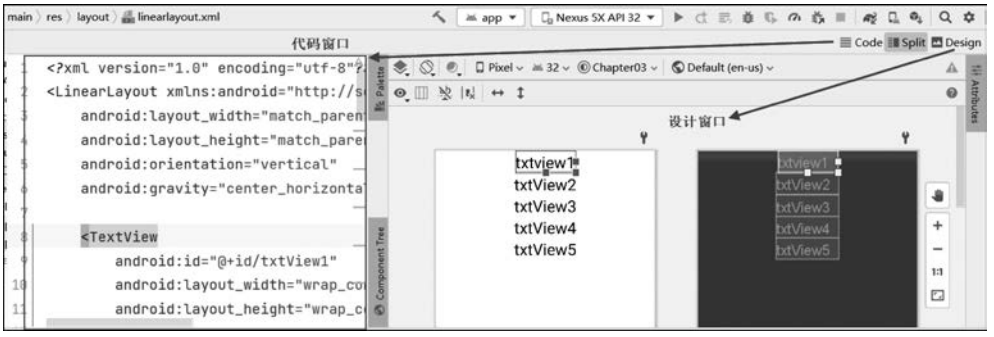


图 3-5　页面布局窗口三种模式

在 LayoutActivity 中使用 linearlayout.xml 布局，代码如下。

【案例 3-2】 LayoutActivity.java

```
public class LayoutActivity extends AppCompatActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.linearlayout);
    }
}
```

上述代码中，调用 setContentView()方法将布局设置到屏幕中，运行结果如图 3-6(a)所示。在上述布局文件中，将 LinearLayout 属性修改为 android:gravity="center"，即垂直方向居中，再次运行 LayoutActivity 结果如图 3-6(b)所示。

(a) center_horizontal水平居中　　(b) center垂直居中

图 3-6　线性布局

3.2.2 表格布局

TableLayout类似表格形式，以行和列的方式来布局子组件。TableLayout继承了LinearLayout，因此其本质上依然是线性布局。TableLayout并不需要明确地声明所包含的行数和列数，而是通过TableRow及其子元素来控制表格的行数和列数。

通常情况下，TableLayout的行数由开发人员直接指定，即TableRow对象(或View控件)的个数；TableLayout的列数等于含有最多子元素的TableRow所包含的元素个数，例如，第一个TableRow中含2个子元素，第二个TableRow中含3个，第三个TableRow中含4个，则该TableLayout的列数为4。

在TableLayout布局中，某列的宽度是由该列中最宽的那个单元格决定，整个表格布局的宽度则取决于父容器的宽度(默认总是占满父容器本身)。

在表格布局器中，可以通过以下3种方式对单元格进行设置。

- Shrinkable：如果某个列被设置为Shrinkable，那么该列中所有单元格的宽度都可以被收缩，以保证表格能适应父容器的宽度。
- Stretchable：如果某个列被设置为Stretchable，那么该列中所有单元格的宽度都可以被拉伸，以保证组件能够完全填满表格的空余空间。
- Collapsed：如果某个列被设置为Collapsed，那么该列中所有单元格都会被隐藏。

TableLayout可设置的属性包括全局属性和单元格属性，全局属性也被称为列属性。TableLayout常用的全局XML属性及相关方法如表3-7所示。

表3-7 TableLayout常用XML属性及对应方法

XML属性	对应方法	功能描述
android:shrinkColumns	setShrinkAllColumns(boolean)	设置可收缩的列。当该列子控件的内容太多，已经挤满所在行时，子控件的内容将往列方向显示，多个列之间用逗号隔开
android:stretchColumns	setStretchAllColumns(boolean)	设置可伸展的列。该列可以横向伸展，最多可占据一整行，多个列之间用逗号隔开
android:collapseColumns	setColumnCollapsed(int,boolean)	设置要隐藏的列，多个列之间用逗号隔开

下述代码演示了表格的全局属性的使用。

【示例】 全局属性的设置

```
<?xml version = "1.0" encoding = "utf - 8"?>
< TableLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:stretchColumns = "0"
    android:collapseColumns = " * "
    android:shrinkColumns = "1,2" >
</TableLayout >
```

其中：

- android:stretchColumns="0"表示第0列可伸展。
- android:shrinkColumns="1,2"表示第1、2列皆可收缩。

- android:collapseColumns="*"表示隐藏所有行。

注意 列可以同时具备 stretchColumns 和 shrinkColumns 属性；当该列的内容较多时，将以“多行”方式显示其内容。（此处所指的“多行”不是真正的多行，而是系统根据需要自动调节该行的 layout_height。）

TableRow.LayoutParams 常用的单元格 XML 属性及方法如表 3-8 所示，通常对 TableRow 的子元素进行修饰。

表 3-8 TableRow.LayoutParams 的单元格 XML 属性及对应方法

XML 属性	功 能 描 述
android:layout_column	指定该单元格在第几列显示
android:layout_span	指定该单元格占据的列数（未指定时，默认为 1）

下述代码演示了表格属性的设置。

【示例】 对表格属性进行设置

```
<?xml version = "1.0" encoding = "utf - 8"?>
< TableLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent"
    android:stretchColumns = "0"
    android:collapseColumns = " * "
    android:shrinkColumns = "1,2" >
  < TableRow >
        < Button android:layout_span = "2"/>
        < Button android:layout_column = "1"/>
  </TableRow >
</TableLayout >
```

其中：

- android:layout_span="2"表示该控件占据 2 列。
- android:layout_column="1"表示该控件显示在第 1 列。

注意 由于 TableLayout 继承了 LinearLayout，因此完全支持 LinearLayout 所支持的全部 XML 属性。

下述代码用于演示 TableLayout 的基本使用。在 res\layout 目录下创建一个表格布局文件 tablelayout.xml。

【案例 3-3】 tablelayout.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< TableLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:id = "@ + id/MorePageTableLayout_Favorite"
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:collapseColumns = "2"
    android:shrinkColumns = "0"
```

```
    android:stretchColumns = "0" >
    < TableRow
        android:id = "@ + id/more_page_row1"
        android:layout_width = "fill_parent"
        android:layout_marginLeft = "2.0dip"
        android:layout_marginRight = "2.0dip"
        android:paddingBottom = "16.0dip"
        android:paddingTop = "8.0dip" >
        < TextView
            android:layout_width = "wrap_content"
            android:layout_height = "fill_parent"
            android:drawablePadding = "10.0dip"
            android:gravity = "center_vertical"
            android:includeFontPadding = "false"
            android:paddingLeft = "17.0dip"
            android:text = "账号管理"
            android:textColor = "#ff333333"
            android:textSize = "16.0sp" />

        < ImageView
            android:layout_width = "wrap_content"
            android:layout_height = "fill_parent"
            android:layout_gravity = "right"
            android:gravity = "center_vertical"
            android:paddingRight = "20.0dip"
            android:src = "@drawable/item_arrow" />
    </TableRow >
    < TableRow android:id = "@ + id/more_page_row0" ...>
        < TextView android:text = "搜索商品" .../>
        < ImageView android:src = "@drawable/item_arrow" .../>
    </TableRow >
    < TableRow android:id = "@ + id/more_page_row2" ...>
        < TextView android:text = "浏览记录" .../>
        < ImageView android:src = "@drawable/item_arrow" .../>
    </TableRow >
</TableLayout >
```

上述代码中：

- 使用< TableLayout >元素定义了表格布局，该元素的 android:collapseColumns 属性用于指明表格的列数，此处设置表格的列数为 2。android:stretchColumns 属性用于指明表格的伸展列，将指定列进行拉伸以填满剩余的空间；注意列号从 0 开始，此处 0 表示第 1 列为伸展列。
- 使用< TableRow >元素定义了表格中的行，其他组件都放在该元素内。

在 LayoutActivity 中，使用 tablelayout.xml 布局，相关代码如下。

```
setContentView(R.layout.tablelayout);
```

运行结果如图 3-7 所示。将< TableLayout >元素中的 android:stretchColumns="0"删除，即不指定伸展列时，运行结果如图 3-8 所示。

图 3-7　第一列为延伸列

图 3-8　普通的表格布局

注意　Android 的表格布局跟 HTML 中的表格布局非常类似，TableRow 相当于 HTML 表格的< tr >标记。

3.2.3　相对布局

RelativeLayout 是一组相对排列的布局方式，在相对布局容器中子组件的位置总是相对于兄弟组件或父容器，例如，一个组件在另一个组件的左边、右边、上边或下边等位置。在相对布局容器中，当 A 组件的位置是由 B 组件来决定时，Android 要求先定义 B 组件，再定义 A 组件。

RelataiveLayout 位于 android. widget 包中，其常用 XML 属性及方法如表 3-9 所示。

表 3-9　RelativeLayout 常用 XML 属性及方法

XML 属性	对 应 方 法	功 能 描 述
android:gravity	setGravity()	设置布局管理器内组件的对齐方式。该属性支持包括 top、bottom、left、right、center_vertical、fill_vertical、center_horizontal、fill _ horizontal、center、fill、clip _ vertical、clip _ horizontal、start 和 end。也可以同时指定多种对齐方式的组合，例如，left|center_vertical 代表出现在屏幕左边且垂直居中
android:ignoreGravity	setIgnoreGravity()	设置特定的组件不受 gravity 属性的影响

为了控制该布局容器中各个子组件的布局分布，RelativeLayout 提供了一个内部类：RelativeLayout. LayoutParams，该类提供了大量的 XML 属性来控制 RelativeLayout 布局中子组件的位置分布，如表 3-10 所示，表中所列属性取值只能为 true 或 false。

表 3-10　RelativeLayout. LayoutParams 的 XML 属性及说明（一）

XML 属性	功 能 描 述
android:layout_alignParentLeft	指定该组件是否与布局容器左对齐
android:layout_alignParentTop	指定该组件是否与布局容器顶端对齐

续表

XML 属性	功 能 描 述
android:layout_alignParentRight	指定该组件是否与布局容器右对齐
android:layout_alignParentBottom	指定该组件是否与布局容器底端对齐
android:layout_centerInParent	指定该组件是否位于布局容器的中央位置
android:layout_centerHorizontal	指定该组件是否在布局容器中水平居中
android:layout_centerVertical	指定该组件是否在布局容器中垂直居中

RelativeLayout.LayoutParams 中另外一部分的属性值可以是其他 UI 组件的 ID 值，表示当前组件与指定 ID 组件的相对位置，如表 3-11 所示。

表 3-11 RelativeLayout.LayoutParams 的 XML 属性及说明(二)

XML 属性	功 能 描 述
android:layout_toLeftOf	控制该组件位于指定 ID 组件的左侧
android:layout_toRightOf	控制该组件位于指定 ID 组件的右侧
android:layout_above	控制该组件位于指定 ID 组件的上方
android:layout_below	控制该组件位于指定 ID 组件的下方
android:layout_alignLeft	控制该组件与指定 ID 组件的左边界进行对齐
android:layout_alignTop	控制该组件与指定 ID 组件的上边界进行对齐
android:layout_alignRight	控制该组件与指定 ID 组件的右边界进行对齐
android:layout_alignBottom	控制该组件与指定 ID 组件的下边界进行对齐

此外，RelativeLayout.LayoutParams 还继承了 android.view.ViewGroup.MarginLayoutParams 类，该类用于定义组件边缘的空白，具有 android:layout_marginTop、android:layout_marginLeft、android:layout_marginBottom、android:layout_marginRight 四个 XML 属性，分别表示上、左、下、右四个方向的边缘空白。

下面通过对“东西南北中”的布局，来演示 RelativeLayout 的使用。

【案例 3-4】 relativelayout.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< RelativeLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent" >
    < TextView
        android:id = "@ + id/middle"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_centerInParent = "true"
        android:text = "中" />
    < TextView android:id = "@ + id/west" android:layout_toLeftOf = "@id/middle"...
        android:text = "西" />
    < TextView android:id = "@ + id/east" android:layout_toRightOf = "@id/middle"...
        android:text = "东" />
    < TextView android:id = "@ + id/north" android:layout_above = "@id/middle"...
        android:text = "北" />
    < TextView android:id = "@ + id/south" android:layout_below = "@id/middle"...
   android:text = "南" />
</RelativeLayout >
```

上述代码使用< RelativeLayout >元素定义了一个相对布局，该布局中含有 5 个文本，分别位于“东、西、南、北、中”。由于相对布局中的组件总是由其他组件来决定分布的位置，在设计过程中首先把“中”元素放到布局容器的中间，然后以该组件为中心，依次将“东、西、南、北”四个组件分布到四周，这样就形成了“上北下南左西右东”的布局效果。文本的摆放位置具体如下。

- “西”位于文本“中”的左边，即通过 layout_toLeftOf 属性进行设置。
- “东”位于文本“中”的右边，即通过 layout_toRightOf 属性进行设置。
- “北”位于文本“中”的上边，即通过 layout_above 属性进行设置。
- “南”位于文本“中”的下边，即通过 layout_below 属性进行设置。

在 LayoutActivity 中，使用 relativelayout. xml 布局，相关代码如下。

```
setContentView(R.layout.relativelayout);
```

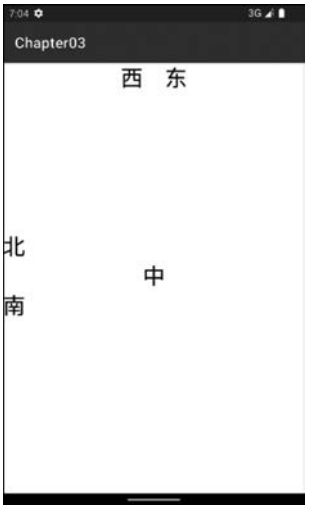


图 3-9　相对布局

运行结果如图 3-9 所示。

在图 3-9 中，以“中”为中心，所显示的“上北下南左西右东”偏离了预想的位置；如果希望“东西南北”四个元素以“中”为中心，分别位于正东、正西、正南、正北”四个方位，需要在布局文件中通过 android:layout_alignXXX 属性来指定具体的对齐方式。

- “中”的“正西”：由于两个文字的高度相同，可以通过 android:layout_alignTop 属性进行设置。
- “中”的“正东”：通过 android:layout_alignTop 属性进行设置。
- “中”的“正南”：由于两个文字的长度相同，可以通过 android:layout_alignLeft 属性进行设置。
- “中”的“正北”：通过 android:layout_alignLeft 属性进行设置。

对 relativelayout. xml 布局文件进行改进，改进后的代码如下。

【案例 3-5】 relativelayout. xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< RelativeLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent" >
    < TextView android:id = "@ + id/middle" ... android:text = "中" />
    < TextView android:id = "@ + id/west" ...
        android:layout_alignTop = "@id/middle"
        android:layout_toLeftOf = "@id/middle"
        android:text = "西" />
    ...
</RelativeLayout >
```

注意　如果五个方位的字符串长度不同，则需要选择居中对齐的方式，例如，使用 android:layout_centerHorizontal="true"来实现。

运行上述代码结果如图 3-10 所示。

图 3-10 显示了传统意义上的“上北、下南、左西、右东”各个方位，但五个方位之间过于紧凑，需要调整一下元素之间的间距，因此可以使用 android:layout_margin 等属性来设置元素的边缘空白，例如，将边缘空白设置为 10dp。

对 relativelayout.xml 布局文件进一步改进，改进后的代码如下。

【案例 3-6】 relativelayout.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< RelativeLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent" >
    < TextView
        android:id = "@ + id/middle"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_centerInParent = "true"
        android:layout_margin = "10dp"
        android:text = "中" />
    …
</RelativeLayout >
```

运行上述代码结果如图 3-11 所示。

图 3-10 相对布局(对齐)

图 3-11 相对布局(页边距)

注意 上述效果除了通过设置除了“中”之外，还可以通过设置其他四个方位对象的 android:layout_marginXX 来实现上述效果，此处不再赘述，请读者验证之。

3.2.4 绝对布局

AbsoluteLayout 通过指定组件的确切 X、Y 坐标来确定组件的位置。下述代码用于演示 AbsoluteLayout 的使用。

【案例 3-7】 absolutelayout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <AbsoluteLayout android:id="@+id/AbsoluteLayout01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button android:text="A" android:id="@+id/Button01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_x="10dp" android:layout_y="20dp"></Button>
        <Button android:text="B" android:id="@+id/Button02" ...
            android:layout_x="100dp" android:layout_y="20dp"></Button>
        <Button android:text="C" android:id="@+id/Button03" ...
            android:layout_x="10dp" android:layout_y="80dp"></Button>
        <Button android:text="D" android:id="@+id/Button04" ...
            android:layout_x="100dp" android:layout_y="80dp"></Button>
    </AbsoluteLayout>
</LinearLayout>
```

上述代码使用<AbsoluteLayout>元素来定义绝对布局，该布局中有四个按钮，每个按钮的位置都是通过 X、Y 轴坐标进行指定，其中，layout_x 属性用于指定元素的 X 轴坐标，layout_y 属性用于指定元素的 Y 轴坐标。

在 LayoutActivity 中，使用 absolutelayout.xml 布局，相关代码如下。

```
setContentView(R.layout.absolutelayout);
```

运行结果如图 3-12 所示。

图 3-12 绝对布局

3.3 事件处理

当用户在程序界面上执行各种操作时，应用程序必须为用户提供响应动作，通过响应动作来完成事件处理。在图形界面(UI)的开发中，有两个非常重要的内容：一个是控件的布局，另一个就是控件的事件处理，其中控件的布局已经在 3.2 节中进简要介绍，本节主要对事件处理进行介绍。

Android 提供了两种方式的事件处理：基于回调的事件处理和基于监听的事件处理。Android 系统充分利用这两种事件处理方式的优点，允许开发人员采用自己熟悉的事件处理方式为用户的操作提供响应动作，从而可以开发出界面友好、人机交互效果好的 Android 应用程序。

3.3.1 基于监听的事件处理

基于监听的事件处理方式和 Java Swing/AWT 的事件处理方式几乎完全相同，如果开发者具有 Java Swing 方面的编程经验，则更容易上手。

Android 系统中引用了 Java 事件处理机制，包括事件、事件源和事件监听器三个事件模型。

- 事件(Event)：是一个描述事件源状态改变的对象，事件对象不是通过 new 运算符创建的，而是在用户触发事件时由系统生成的对象。事件包括键盘事件、触摸事件等，一般作为事件处理方法的参数，以便从中获取事件的相关信息。
- 事件源(Event Source)：触发事件的对象，事件源通常是 UI 组件，例如，单击按钮时按钮就是事件源。
- 事件监听器(Event Listenrer)：当触发事件时，事件监听器用于对该事件进行响应和处理。监听器需要实现监听接口中所定义的事件处理方法。

当用户单击一个按钮或单击某个菜单选项时，这些操作就会触发一个响应事件，该事件就会调用在事件源上注册的事件监听器，事件监听器调用相应的事件处理程序并完成相应的事件处理。基于监听的事件处理流程如图 3-13 所示。

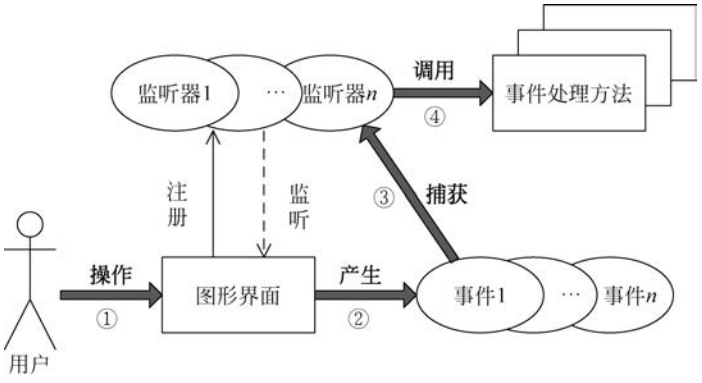


图 3-13 基于监听的事件处理流程

Android 的事件处理机制是一种委派式事件处理机制，该处理方式类似于人类社会的分工协作，例如，某个企业(事件源)进行货物采购(事件)时，企业通常不会自己运输物品，而是找特定的物流公司来运输；如果发生了火灾(事件)，则会委派给消防局(事件监听器)来处理；而消防局或物流公司也会同时监听多个企业的火灾事件或货物运输事件。委派式的处理方式将事件源和事件监听器分离，从而提供更好的程序模型，有利于提高程序的可维护性和代码的健壮性。

在 Android 应用程序中，所有的组件都可以针对特定的事件指定一个事件监听器，每个事件监听器可以监听一个或多个事件源。同一个事件源上也可能发生多个事件，例如，在按钮上可能发生单击、获取焦点等事件，委派式事件处理将事件源上的所有可能发生的事件分别委派给不同的事件监听器来处理；同时也可以让一类事件都使用同一个事件监听器来处理。

Android 中常用的事件监听器如表 3-12 所示，这些事件监听器以内部接口的形式定义在 android.view.View 中。

表 3-12 Android 中的事件监听器

事件监听器接口	事　　件	功 能 描 述
OnClickListener	单击事件	当用户单击某个组件或者按方向键时触发该事件
OnFocusChangeListener	焦点事件	当组件获得或者失去焦点时触发该事件
OnKeyListener	按键事件	当用户按下或者释放设备上的某个按键触发该事件

续表

事件监听器接口	事　　件	功 能 描 述
OnTouchListener	触摸事件	如果设备具有触摸屏功能，当触碰屏幕时触发该事件
OnCreateContextMenuListener	创建上下文菜单事件	当创建上下文菜单时触发该事件
OnCheckedChangeListener	选项改变事件	当选择改变时触发该事件

由此可知，事件监听器本质上是一个实现了特定接口的 Java 对象。在程序中实现事件监听器，通常有以下几种形式。

- Activity 本身作为事件监听器：通过 Activity 实现监听器接口，并实现事件处理方法。
- 匿名内部类形式：使用匿名内部类创建事件监听器对象。
- 内部类或外部类形式：将事件监听类定义为当前类的内部类或普通的外部类。
- 绑定标签：在布局文件中为指定标签绑定事件处理方法。

通常实现基于监听的事件处理步骤如下。

（1）创建事件监听器。

（2）在事件处理方法中编写事件处理代码。

（3）在相应的组件上注册监听器。

1. Activity 本身作为事件监听器

通过 Activity 实现监听器接口，并实现该接口中对应的事件处理方法。下述代码演示了在 Button 按钮上绑定单击事件，当单击按钮时改变文字的内容，布局代码如下。

【案例 3-8】 event_btn.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:gravity = "center_horizontal"
    android:orientation = "vertical" >
    < EditText android:id = "@ + id/showTxt" android:layout_width = "match_parent"
        android:layout_height = "wrap_content" android:editable = "false" />
    < Button android:id = "@ + id/clickBtn" android:layout_width = "wrap_content"
        android:layout_height = "wrap_content" android:text = "单击我" />
</LinearLayout >
```

上述代码定义了 Button 和 EditText 两个组件，主要用于实现 Button 的单击事件。实现监听和事件处理的 Activity 代码如下。

【案例 3-9】 EventBtnActivity.java

```
//1.实现事件监听器接口
public class EventBtnActivity extends AppCompatActivity
        implements OnClickListener{
    //单击 Button
    private Button clickBtn;
    //文字显示
    private TextView showTxt;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.event_btn);
        //初始化组件
        showTxt = (TextView) findViewById(R.id.showTxt);
        clickBtn = (Button)findViewById(R.id.clickBtn);
        //3.直接使用 Activity 作为事件监听器
        clickBtn.setOnClickListener(this);
    }
    //2. 在事件处理方法中编写事件处理代码
    @Override
    public void onClick(View v) {
        //实现事件处理方法
        showTxt.setText("btn 按钮被单击了!");
    }
}
```

运行上述代码,当单击“单击我”按钮时,TextView 文本内容将发生改变,效果如图 3-14 所示。

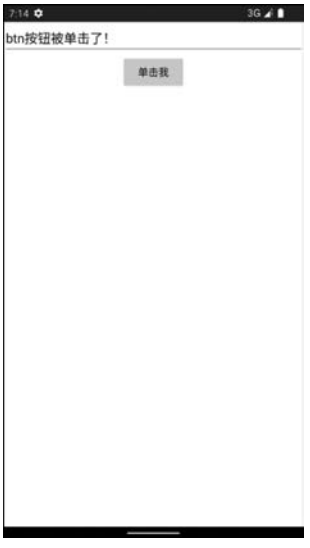


图 3-14 按钮单击效果

上述代码中,定义的 EventBtnActivity 继承了 Activity,并实现了 OnClickListener 接口,此时 Activity 对象允许作为事件监听器进行使用。代码“clickBtn.setOnClickListener(this);”用于为 clickBtn 按钮注册事件监听器。当单击 Button 按钮时,触发鼠标单击事件并调用 onClick()事件处理方法,TextView 文本内容变成“btn 按钮被单击了!”。从上面程序中可以看出,基于监听的事件的处理模型的编程步骤如下。

(1) 获取所要触发事件的事件源控件,例如本例中的 clickBtn 对象。

(2) 实现事件监听器类,本例中的监听器类是 Activity 对象本身(实现了 OnClickListener 接口)。

(3) 调用事件源的 setXxxListener()方法,将事件监听器注册给事件源对象;当事件源上发生指定事件时,Android 会触发事件监听器,由事件监听器调用相应的方法来处理事件。

2. 匿名内部类形式

Activity 的主要职责是完成界面的初始化工作,而案例 3-9 中使用 Activity 本身作为监听器类,并在 Activity 类中定义事件处理方法,易造成程序结构混乱。大部分情况下事件监听器只是临时使用一次,所以匿名内部类形式的事件监听器更合适。将案例 3-9 改为匿名内部类形式,代码如下。

【案例 3-10】 AnonymousBtnActivity.java

```
//实现事件监听器接口
public class AnonymousBtnActivity extends AppCompatActivity{
    //单击 Button
    private Button clickBtn;
    //文字显示
    private TextView showTxt;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.event_btn);
        //初始化组件
        showTxt = (TextView) findViewById(R.id.showTxt);
        clickBtn = (Button)findViewById(R.id.clickBtn);
        //使用匿名内部类创建一个监听器
        clickBtn.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                //实现事件处理方法
                showTxt.setText("btn 按钮被单击了!");
            }
        });
    }
}
```

上述代码中粗体部分使用匿名内部类创建了一个事件监听器对象，界面效果与图 3-14 一致，此处不再演示。

3. 内部类、外部类形式

“内部类”形式是指将事件监听器定义成当前类的内部类。下述代码演示使用内部类的方式实现事件监听。

【案例 3-11】 InnerClassBtnActivity.java

```
//实现事件监听器接口
public class InnerClassBtnActivity extends AppCompatActivity{
    //单击 Button
    private Button clickBtn;
    //文字显示
    private TextView showTxt;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.event_btn);
        //初始化组件
        showTxt = (TextView)findViewById(R.id.showTxt);
        clickBtn = (Button)findViewById(R.id.clickBtn);
        //直接使用 Activity 作为事件监听器
        clickBtn.setOnClickListener(new ClickListener());
    }
    //内部类方式定义一个事件监听器
    class ClickListener implements OnClickListener{
        @Override
        public void onClick(View v) {
            //实现事件处理方法
            showTxt.setText("btn 按钮被单击了!");
        }
    }
}
```

使用内部类有以下优点。

- 可以在当前类中复用内部监听器类。
- 由于监听器是当前类的内部类，所以可以访问当前类的所有界面组件。

注意 外部监听器的定义方式和内部类的定义方式相似，由于使用外部类事件监听器的形式比较少见，此处不再赘述。

4. 绑定标签

Android 还有一种更简单的绑定事件的方式，在界面布局文件中直接为指定标签绑定事件处理方法。对于大多数 Android 界面的组件标签而言，基本都支持 onClick 事件属性，相应的属性值就是一个类似 xxxMethod 形式的方法名称。

【案例 3-12】 event_tag.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:gravity = "center_horizontal"
    android:orientation = "vertical" >
    < EditText android:id = "@ + id/showTxt" android:layout_width = "match_parent"
        android:layout_height = "wrap_content" android:editable = "false" />
    < Button android:id = "@ + id/clickBtn" android:layout_width = "wrap_content"
        android:layout_height = "wrap_content" android:text = "单击我"
        android:onClick = "clickMe"/>
</LinearLayout >
```

上述代码中，粗体部分用于为 clickBtn 按钮绑定一个事件处理方法 clickMe，此时需要开发者在相应的 Activity 中定义一个名为 clickMe 的方法，该方法用于负责处理按钮的单击事件，代码如下。

【案例 3-13】 BindTagActivity.java

```
//实现事件监听器接口
public class BindTagActivity extends AppCompatActivity{
    //单击 Button
    private Button clickBtn;
    //文字显示
    private TextView showTxt;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.event_btn);
        //初始化组件
        showTxt = (TextView) findViewById(R.id.showTxt);
        clickBtn = (Button)findViewById(R.id.clickBtn);
    }
    public void clickMe(View v){
        //实现事件处理方法
        showTxt.setText("btn 按钮被单击了!");
    }
}
```

上述代码中，粗体部分定义了 clickMe()方法，其中有一个 View 类型的参数，方法的返回类型为 void。运行上述代码，界面效果与图 3-14 一致。

3.3.2 基于回调机制的事件处理

在 Android 平台中，每个 View 都拥有事件处理的回调方法，开发人员通过重写这些回调方法来实现所需要响应的事件。当某个事件没有被任何一个 View 控件处理时，便会调用 Activity 中相应的回调方法进行处理。从代码实现的角度来看，基于回调的事件处理模型要比基于监听的事件处理模型更为简单。事件监听机制是一种委托式的事件处理，而回调机制则恰好与之相反；对于基于回调的事件处理模型而言，事件源和事件监听器是统一的，当用户在 GUI 组件上触发某个事件时，组件自身的方法将会负责处理该事件。

为了实现回调机制的事件处理，Android 为所有的 GUI 组件都提供了事件处理的回调方法，例如，View 中提供了 onKeyDown()、onKeyUp()、onTouchEvent()、onTrackBallEvent()和 onFocusChanged()等事件回调方法。

1. onKeyDown()

onKeyDown()方法是 KeyEvent.Callback 接口中的抽象方法，所有的 View 都实现了该接口并重写了 onKeyDown()方法，onKeyDown()方法用来捕捉手机键盘按键被按下的事件，方法的签名如下。

【语法】

```
public boolean onKeyDown (int keyCode, KeyEvent event)
```

其中：

- 参数 keyCode 表示被按下的键值(即键盘码)，手机键盘中每个按键都有一个单独的键盘码，在应用程序中可通过键盘码的值来判断用户按下的是哪个键。在 KeyEvent 类中定义了许多常量来表示不同的 keyCode，如表 3-13 所示。
- 参数 event 用于封装按键事件的对象，其中包含触发事件的详细信息，例如，事件的状态、事件的类型以及事件触发的时间等。当用户按下某个键时，系统自动将事件封装成 KeyEvent 对象供应用程序使用。
- onKeyDown()方法的返回值为 boolean 类型，当方法返回 true 时，表示已经完整地处理了该事件，并不希望其他的回调方法再次进行处理；当方法返回 false 时，表示没有完全处理完该事件，其他回调方法可以继续对该事件进行处理，例如，Activity 中的回调方法。

表 3-13 keyCode 的部分值

常 量 名	功 能 描 述
KEYCODE_CALL	拨号键
KEYCODE_ENDCALL	挂机键
KEYCODE_HOME	按键 Home
KEYCODE_MENU	菜单键
KEYCODE_BACK	返回键
KEYCODE_SEARCH	搜索键

续表

常 量 名	功 能 描 述
KEYCODE_CAMERA	拍照键
KEYCODE_FOCUS	拍照对焦键
KEYCODE_POWER	电源键
KEYCODE_NOTIFICATION	通知键
KEYCODE_MUTE	话筒静音键
KEYCODE_VOLUME_MUTE	扬声器静音键
KEYCODE_VOLUME_UP	音量增加键
KEYCODE_VOLUME_DOWN	音量减小键
KEYCODE_CALL	拨号键
KEYCODE_ENDCALL	挂机键

注意 表 3-13 中是常见的手机键盘中的 keyCode 值；此外，keyCode 还包括控制键、组合键、基本键、符号键、小键盘键和功能键等，读者可以参见 KeyEvent 代码。

下述代码通过一个简单例子来演示 onKeyDown()方法的使用。通过用户的按键，来捕获手机键盘事件，并根据按键情况来显示相关信息。

【案例 3-14】 keydown_btn.xml

```
<?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 = "match_parent"
    android:gravity = "center_horizontal"
    android:orientation = "vertical" >
    < EditText
        android:id = "@ + id/showTxt"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:editable = "false" />
</LinearLayout >
```

上述代码定义了一个 EditText 文本框，用于显示用户按下按键时不同的文本。在相应的 Activity 中实现 onKeyDown 事件监听，代码如下。

【案例 3-15】 KeyDownActivity.java

```
public class KeyDownActivity extends AppCompatActivity {

    //自定义的 Button
    EditText showText;
    public void onCreate(Bundle savedInstanceState) { //重写的 onCreate()方法
        super.onCreate(savedInstanceState);
        setContentView(R.layout.keydown_btn);
        showText = (EditText) findViewById(R.id.showTxt);
    }
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        //重写的键盘按下监听
```

```
            switch (keyCode) {
            case KeyEvent.KEYCODE_BACK:
                showText.setText("按下了【回退键】");
                break;
            case KeyEvent.KEYCODE_0:
                showText.setText("0 键");
                break;
            case KeyEvent.KEYCODE_A:
                showText.setText("A 键");
                break;
            case KeyEvent.KEYCODE_VOLUME_DOWN:
                showText.setText("音量 - ");
                break;
            case KeyEvent.KEYCODE_VOLUME_UP:
                showText.setText("音量 + ");
                break;
            default:
                break;
            }
            return super.onKeyDown(keyCode, event);
        }
    }
```

上述代码中，粗体部分为重写的 Activity 中的 onKeyDown()方法，在该方法中实现键盘按下的事件处理，方法返回之前调用父类的同名方法并返回处理结果。当按下手机键盘上的回退键、0 键、A 键、音量－或音量＋键时，在 showText 文本框中显示对应的文本信息，例如，按下键盘上的音量＋键或时，显示结果如图 3-15 所示。

如果将 onKeyDown()方法中的返回代码“return super.onKeyDown(keyCode, event);”改为“return true;”，然后按下回退键时，回退键的按下事件会被捕获并进行处理，但界面仍然在当前页面，并没有产生回退效果，如图 3-16 所示。

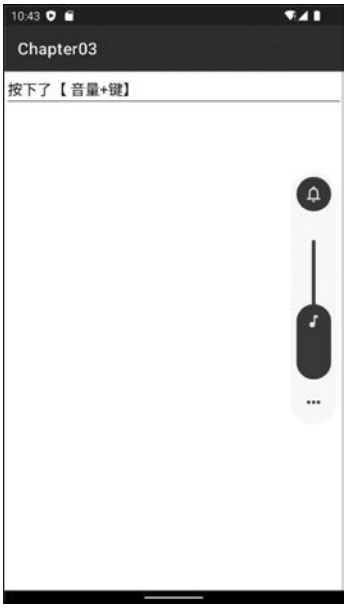


图 3-15　按下音量＋键效果

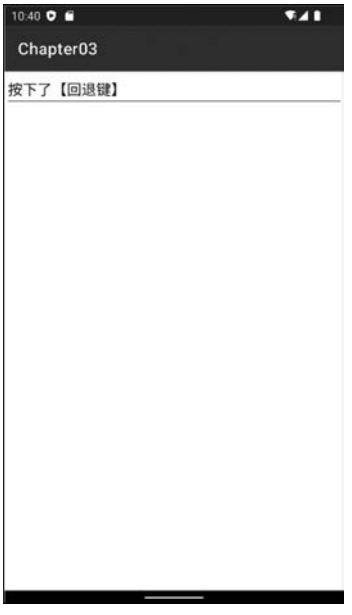


图 3-16　按下回退键效果

注意 在实际应用中，有时需要对 Home 等特殊键进行屏蔽处理，可以采用上述方式对这些键进行捕获并处理。

2. onKeyUp()

onKeyUp()方法也是接口 KeyEvent.Callback 中的一个抽象方法，并且所有的 View 都实现了该接口并重写了 onKeyUp()方法，onKeyUp()方法用来捕捉手机键盘按键抬起的事件，方法的签名如下。

【语法】

```
public boolean onKeyUp (int keyCode, KeyEvent event)
```

其中：

- 参数 keyCode 表示触发事件的按键码，需要注意的是，同一个按键在不同型号的手机中的按键码可能不同。
- 参数 event 是一个事件封装类的对象，其含义与 onKeyDown()方法中的完全相同，此处不再赘述。
- onKeyUp()方法返回值的含义与 onKeyDown()方法相同，同样通知系统是否希望其他回调方法再次对该事件进行处理。

onKeyUp()的使用方式与 onKeyDown()基本相同，只是 onKeyUp()方法在按键抬起时触发调用。如果用户需要对按键被抬起时进行事件处理，可以通过重写该方法来实现。

3. onTouchEvent()

onKeyDown()和 onKeyUp()方法主要针对手机键盘事件的处理，onTouchEvent()方法主要针对手机屏幕事件的处理。onTouchEvent()方法在 View 类中定义，并且所有的 View 都重写了该方法，应用程序可以通过 onTouchEvent()方法来处理手机屏幕的触摸事件。onTouchEvent()方法的签名如下。

【语法】

```
public boolean onTouchEvent(MotionEvent event)
```

其中：

- 参数 event 是一个手机屏幕触摸事件封装类的对象，用于封装事件的相关信息，例如，触摸的位置、触摸的类型以及触摸的时间等。在用户触摸手机屏幕时由系统创建 event 对象。
- onTouchEvent()方法的返回机制与键盘响应事件的相同，当已经完整地处理了该事件且不希望其他回调方法再次处理时返回 true，否则返回 false。

与 onKeyDown()、onKeyUp()方法不同的是，onTouchEvent()方法可以处理多种事件；一般情况下，屏幕中的按下、抬起和拖动事件均可由 onTouchEvent()方法进行处理，只是每种情况中的动作值有所不同。

- 屏幕被按下：当触摸屏幕时，会自动调用 onTouchEvent()方法来处理事件，此时 MotionEvent.getAction()的值为 MotionEvent.ACTION_DOWN，如果在应用程序中需要处理屏幕被按下的事件，只需重写该回调方法，并在方法中进行动作的判

断即可。

- 触摸动作抬起：离开屏幕时所触发的事件，该事件需要 onTouchEvent()方法来捕捉，并在该方法中进行动作判断。当 MotionEvent.getAction()的值为 MotionEvent.ACTION_UP 时，表示触发的是触摸屏幕动作抬起的事件。
- 在屏幕中拖动：onTouchEvent()方法还用于处理在屏幕上滑动的事件，根据 MotionEvent.getAction()方法的返回值来判断动作值是否为 MotionEvent.ACTION_MOVE，然后进行相应的处理。

下述代码通过一个简单例子来演示 onTouchEvent()方法的使用。在用户单击的位置绘制一个矩形，然后监测用户触摸的状态，当用户在屏幕上移动手指时，使矩形随之移动，而当用户手指离开手机屏幕时，停止绘制矩形。代码如下。

【案例 3-16】 KeyTouchActivity.java

```
public class KeyTouchActivity extends AppCompatActivity {
    TouchView touchView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //初始化自定义的 View ①
        touchView = new TouchView(this);
        //设置当前显示的用户界面 ②
        setContentView(touchView);
    }
    //重写的 onTouchEvent 回调方法
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN://手指按下 ③
            //改变 x 坐标
            touchView.x = (int) event.getX();
            //改变 y 坐标
            touchView.y = (int) event.getY() - 52;
            touchView.postInvalidate();
            //重绘
            break;
        case MotionEvent.ACTION_MOVE: //手指移动 ④
            //改变 x 坐标
            touchView.x = (int) event.getX();
            //改变 y 坐标
            touchView.y = (int) event.getY() - 52;
            touchView.postInvalidate();
            //重绘
            break;
        case MotionEvent.ACTION_UP://手指抬起⑤
            //改变 x 坐标
            touchView.x = -100;
            //改变 y 坐标
            touchView.y = -100;
            //重绘
            touchView.postInvalidate();
            break;
        }
```

```
        return super.onTouchEvent(event);
    }
    //定义 View 的子类⑥
    class TouchView extends View {
        //画笔
        Paint paint;
        //x 坐标
        int x = 300;
        //y 坐标
        int y = 300;
        //矩形的宽度
        int width = 100;
        public TouchView(Context context) {
            super(context);
            //初始化画笔⑦
            paint = new Paint();

        }
        @Override
        protected void onDraw(Canvas canvas) {
            //绘制方法⑧
            canvas.drawColor(Color.WHITE);
            //绘制背景色⑨
            canvas.drawRect(x, y, x + width, y + width, paint);
            //绘制矩形⑩
            super.onDraw(canvas);
        }
    }
}
```

代码解释如下。

- 标号①处创建了一个自定义的 TouchView 对象，标号②处将该 View 的对象设置为当前显示的用户界面。
- 标号③用于判断当前事件是否为屏幕被按下的事件，通过调用 MotionEvent 的 getX()和 getY()方法得到事件发生的 x 和 y 坐标，并赋给 TouchView 对象的 x 与 y 成员变量。
- 标号④用于判断是否为屏幕的滑动事件，同样将得到事件发生的位置并赋给 TouchView 对象的 x、y 成员变量。需要注意的是，因为此时手机屏幕并不是全屏模式，所以需要对坐标进行调整。
- 标号⑤用于判断是否为触摸屏幕动作抬起的事件，此时将 TouchView 对象的 x、y 成员变量设成－100，表示并不需要在屏幕中绘制矩形。
- 标号⑥处自定义了 TouchView 类，并重写了 View 类的 onDraw()绘制方法。在⑦处的构造方法中初始化绘制时需要的画笔，然后在第⑧～⑩行的方法中根据成员变量 x、y 的值来绘制矩形。

注意 自定义的 View 并不会自动刷新，所以每次改变数据模型时都需要手动调用 postInvalidate()方法进行屏幕的刷新操作。关于自定义 View 的使用方法，将会在后面的章节中进行详细介绍，此处只是简单地使用。

运行上述代码，效果如图 3-17 所示。

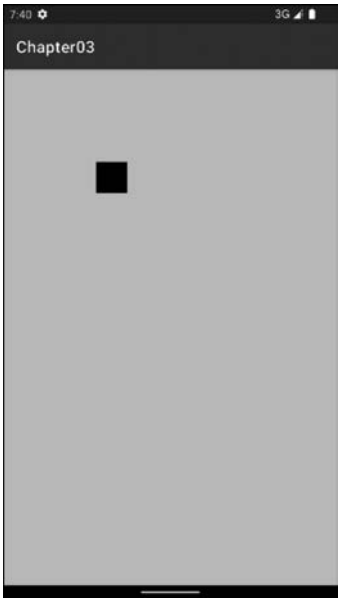


图 3-17　矩形绘制

当单击屏幕时，会在所单击的位置绘制一个矩形，当手指在屏幕中滑动时，该矩形会随之移动，而当手指离开屏幕时，取消所绘制的矩形。

注意　由于无法在图书中展示动态效果，需要读者自行验证。

4. onTrackBallEvent()

onTrackBallEvent()方法是手机中轨迹球的处理方法。所有的 View 同样全部实现了该方法，该方法的签名如下。

【语法】

```
public Boolean onTrackballEvent (MotionEvent event)
```

其中：

- 参数 event 为手机轨迹球事件封装类的对象，用于封装触发事件的相关信息，包括事件的类型、触发时间等；一般情况下，该对象会在用户操控轨迹球时由系统创建。
- onTrackBallEvent()方法的返回机制与前面介绍的各个回调方法完全相同，此处不再赘述。

轨迹球与手机键盘有一定的区别，具体如下。

- 某些型号的手机设计出的轨迹球会比只有手机键盘时更美观，可增添用户对手机的整体印象。
- 轨迹球使用更为简单，例如，在某些游戏中使用轨迹球控制会更为合理。
- 使用轨迹球会比键盘更为细化，即滚动轨迹球时，后台表示状态的数值会变得更细微、更精准。
- onTrackBallEvent()方法的使用与前面介绍过的各个回调方法基本相同，可以在 Activity 中重写该方法，也可以在 View 的子类中重写。

注意　在模拟器运行状态下，可以通过 F6 键打开模拟器的轨迹球，然后通过鼠标的移动来模拟轨迹球事件。

5. onFocusChanged()

前面介绍的各个方法都可以在 View 及 Activity 中重写，接下来介绍的 onFocusChanged()方法只能在 View 中重写。该方法是焦点改变的回调方法，在某个控件重写了该方法后，当焦点发生变化时，会自动调用该方法来处理焦点改变的事件，该方法的签名如下。

【语法】

```
protected void onFocusChanged (
    Boolean gainFocus, int direction, Rect previouslyFocusedRect)
```

其中：

- 参数 gainFocus 表示触发该事件的 View 是否获得了焦点，当该控件获得焦点时 gainFocus 为 true，否则为 false。
- 参数 direction 表示焦点移动的方向，使用数值表示，有兴趣的读者可以重写 View 中的该方法并打印该参数进行观察。
- 参数 previouslyFocusedRect 表示在触发事件的 View 的坐标系中，前一个获得焦点的矩形区域，即表示焦点是从哪里来的，如果不可用则为 null。

下述代码通过一个简单例子来演示 onFocusChanged()方法的使用。通过移动上下按键来观察屏幕中 4 个按钮在获得或失去焦点后，按钮上文字的变化情况，代码如下。

【案例 3-17】 FocusEventActivity.java

```
public class FocusEventActivity extends AppCompatActivity {
    //定义 4 个 button ①
    FocusButton focusButton1;
    FocusButton focusButton2;
    FocusButton focusButton3;
    FocusButton focusButton4;
    //声明 myButton04 的引用
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //创建 4 个 FocusButton 对象 ②
        focusButton1 = new FocusButton(this);
        focusButton2 = new FocusButton(this);
        focusButton3 = new FocusButton(this);
        focusButton4 = new FocusButton(this);
        //设置 focusButton1 上的文字 ③
        focusButton1.setText("focusButton1");
        //设置 focusButton2 上的文字
        focusButton2.setText("focusButton2");
        //设置 focusButton3 上的文字
        focusButton3.setText("focusButton3");
        //设置 focusButton4 上的文字
        focusButton4.setText("focusButton4");
        //创建一个线性布局 ④
        LinearLayout linearLayout = new LinearLayout(this);
        //设置其布局方式为垂直
        linearLayout.setOrientation(LinearLayout.VERTICAL);
        //将 focusButton1 添加到布局中 ⑤
        linearLayout.addView(focusButton1);
        //将 focusButton2 添加到布局中
        linearLayout.addView(focusButton2);
        //将 focusButton3 添加到布局中
        linearLayout.addView(focusButton3);
        //将 focusButton4 添加到布局中
        linearLayout.addView(focusButton4);
        //设置当前的用户界面 ⑥
        setContentView(linearLayout);
    }
    //自定义 Button ⑦
    class FocusButton extends Button {
```

```
        //自定义 Button
        public FocusButton(Context context) {
            //构造器
            super(context);
        }
        @Override
        protected void onFocusChanged(boolean focused, int direction,
                Rect previouslyFocusedRect) {
            String suffix = "(选中)";
            String text = getText().toString();
            //重写的焦点变化方法
            if(focused){
                //获取焦点时,添加"(选中)"文字
                if(!text.contains(suffix)){
                    this.setText(text + suffix);
                }
            }else{
                //去掉"(选中)"文字
                if(text.contains(suffix)){
                    text = text.substring(0,text.length() - suffix.length());
                    this.setText(text);
                }
            }
            super.onFocusChanged(focused, direction, previouslyFocusedRect);
        }
    }
}
```

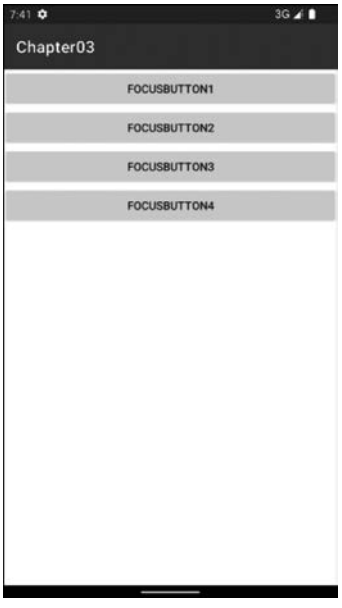


图 3-18　焦点获取

上述代码解释如下。

- 标号①处声明了 4 个自定义的按钮变量。
- 标号②处初始化标号①所声明的 4 个自定义的按钮控件，然后在标号③处分别设置了各个按钮上的文字。
- 标号④处创建一个线性布局，并设置其布局方式为垂直。
- 标号⑤处用于将 4 个按钮控件依次添加到线性布局中，然后在⑥处将该线性布局设置成当前显示的用户界面。
- 标号⑦处为自定义的 FocusButton 类，在该类中重写了 onFocusChanged() 方法，并在该方法内判断是否获取焦点，如果按钮获取焦点则为该按钮添加“(选中)”文字，否则取消“(选中)”文字。

运行上述代码，效果如图 3-18 所示。

读者可以通过向上向下键来控制各个按钮的焦点切换，并观察界面的变化情况。

注意 每按下一次按键，会调用两次 onFocusChanged()方法，第一次是某个按钮失去焦点时调用，第二次是另一个按钮获得焦点时调用。同时方向 direction 的值会根据情况的不同而有所不同。此外，默认情况下 Android 5.0.1 模拟器的方向键可能不起作用，需要读者自己重新设置，将 C:\Users\xxuser\.android\avd\android_720p.avd\文件夹下的 config.ini 中的 hw.dPad 属性值改为 yes；其中，xxuser 表示当前系统用户，android_720p 表示自定义的模拟器名称。

在介绍 onFocusChanged()方法时，提到了焦点的概念。焦点描述了按键事件(或者是屏幕事件等)的接受者，每次按键事件都发生在拥有焦点的 View 上。在应用程序中，开发人员可以对焦点进行控制，例如，将焦点从一个 View 移动到另一个 View 上。下面列出一些与焦点有关的方法，如表 3-14 所示，读者可以进一步进行学习。

表 3-14 常见的焦点相关方法

方 法	功 能 描 述
setFocusable()	用于设置 View 是否可以拥有焦点
isFocusable()	用于判断 View 是否可以拥有焦点
setNextFocusDownId()	用于设置 View 的焦点向下移动后获得焦点 View 的 ID
hasFocus()	用于判断 View 的父控件是否获得了焦点
requestFocus()	用于尝试让此 View 获得焦点
isFocusableTouchMode()	用于设置 View 是否可以在触摸模式下获得焦点，默认情况下不可用

3.4 Widget 简单组件

本节将要介绍的是 Android 的基本组件。一个易操作、美观的 UI 界面，都是从界面布局开始，然后不断地向布局容器中添加界面组件。掌握这些最基本的用户界面组件是学好 Android 编程的基础。Android 几乎所有的用户界面组件都定义在 android.widget 包中，如 Button、TextView、EditText、CheckBox、RadioGroup 和 Spinner 等。

3.4.1 Widget 组件通用属性

对 Widget 组件进行 UI 设计时，既可以采用 XML 布局方式，也可以采用编写代码的方式。其中，XML 布局文件方式由于简单易用，被广泛使用。Widget 组件几乎都属于 View 类，因此大部分属性在这些组件之间是通用的，如表 3-15 所示。

表 3-15 Widget 组件通用属性

属 性 名 称	功 能 描 述
android:id	设置控件的索引，Java 程序可通过 R.id.<索引>形式来引用该控件
android:layout_height	设置布局高度，使用以下三种方式来指定高度：fill_parent(和父元素相同)、wrap_content(随组件本身的内容调整)、通过指定 px 值来设置高度
android:layout_width	设置布局宽度，也可以采用三种方式：fill_parent、wrap_content、指定 px 值
android:autoLink	设置是否当文本为 URL 链接时，文本显示为可单击的链接。可选值为 none、web、email、phone、map 和 all
android:autoText	如果设置，将自动执行输入值的拼写纠正
android:bufferType	指定 getText()方式取得的文本类别

续表

属性名称	功能描述
android:capitalize	设置英文字母大写类型。需要弹出输入法才能看得到
android:cursorVisible	设定光标为显示/隐藏，默认为显示
android:digits	设置允许输入哪些字符，如 1234567890.＋－＊/%\n()
android:drawableBottom	在 text 的下方输出一个 drawable
android:drawableLeft	在 text 的左边输出一个 drawable
android:drawablePadding	设置 text 与 drawable（图片）的间隔，与 drawableLeft、drawableRight、drawableTop、drawableBottom 一起使用，可设置为负数，单独使用没有效果
android:drawableRight	在 text 的右边输出一个 drawable 对象
android:inputType	设置文本的类型，用于帮助输入法显示合适的键盘类型
android:cropToPadding	是否截取指定区域用空白代替；单独设置无效果，需要与 scrollY 一起使用
android:maxHeight	设置 View 的最大高度

3.4.2 TextView 文本框

TextView 文本框直接继承了 View 类，位于 android.widget 包中。TextView 定义了操作文本框的基本方法，是一个不可编辑的文本框，多用于在屏幕中显示静态字符串。从功能上来看，TextView 实际上是一个文本编辑器，只是 Android 关闭了其文字编辑功能，如果开发者想要定义一个可以编辑内容的文本框，可以使用其子类 EditText 来实现，EditText 允许用户编辑文本框的内容。此外，TextView 还是 Button 的父类，TextView 类及其子类的继承关系如图 3-19 所示。

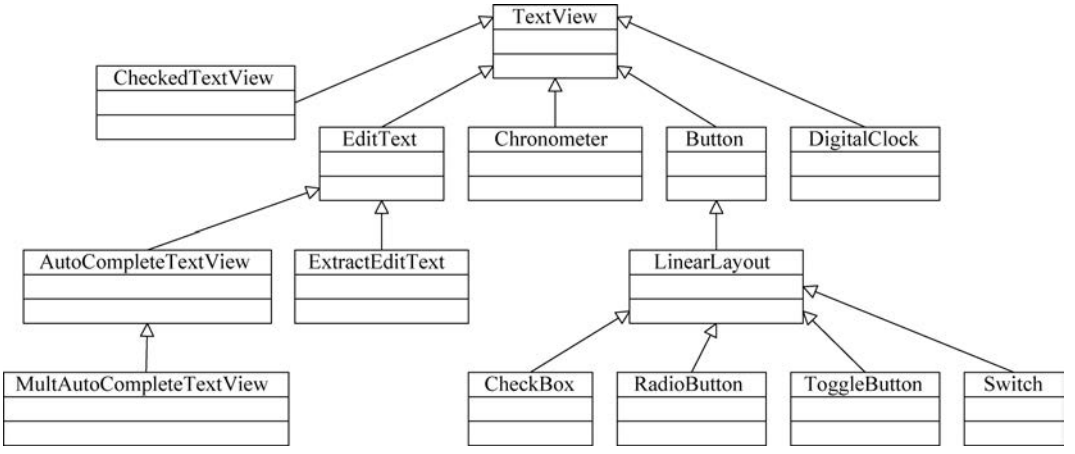


图 3-19　TextView 及其子类

TextView 提供了大量的 XML 属性，这些属性不仅可以适用于 TextView，还可以适用于其子类；TextView 所支持的 XML 属性及说明如表 3-16 所示。

表 3-16　TextView 类的 XML 属性及描述

XML 属性	功能描述
android:autoLink	设置是否当文本为 URL 链接（例如 email、电话号码、map）时，文本显示为可单击的链接。可选值有 none、web、email、phone、map 和 all

续表

XML 属性	功 能 描 述
android:autoText	如果设置,将自动执行输入值的拼写纠正。此处无效果,在显示输入法并输入的时候起作用
android:digits	设置允许输入哪些字符。如 1234567890.+-*/%\n()
android:drawableLeft	在 text 的左边输出一个 drawable,如图片
android:drawablePadding	设置 text 与 drawable(图片)的间隔,与 drawableLeft、drawableRight、drawableTop、drawableBottom 一起使用,可设置为负数,单独使用没有效果
android:drawableRight	在 text 的右边输出一个 drawable,如图片
android:drawableTop	在 text 的正上方输出一个 drawable,如图片
android:ellipsize	设置当文字过长时如何显示该控件,取值情况如下。 start:省略号显示在开头。 end:省略号显示在结尾。 middle:省略号显示在中间。 marquee:以跑马灯的方式显示(动画横向移动)
android:gravity	设置文本位置,例如,center 表示文本将居中显示
android:hint	文本为空时显示的提示信息,可通过 textColorHint 设置提示信息的颜色。此属性在 TextView 和 EditView 中使用
android:ems	设置 TextView 的宽度为 N 个字符的宽度
android:maxEms	设置 TextView 的宽度为最长为 N 个字符的宽度,与 ems 同时使用时将覆盖 ems 选项
android:minEms	设置 TextView 的宽度为最短为 N 个字符的宽度,与 ems 同时使用时将覆盖 ems 选项
android:maxLength	限制显示的文本长度,超出部分不显示
android:lines	设置文本的行数,设置两行就显示两行,即使第二行没有数据
android:maxLines	设置文本的最大显示行数,与 width 或者 layout_width 结合使用,超出部分自动换行,超出行数将不显示
android:minLines	设置文本的最小行数,与 lines 类似
android:linksClickable	设置链接是否可以单击,即使设置了 autoLink
android:lineSpacingExtra	设置行间距
android:lineSpacingMultiplier	设置行间距的倍数,例如 1.2
android:numeric	如果被设置,该控件将有一个数字输入法。 此处无用,设置后唯一效果是 TextView 有单击效果,此属性将在 EdtiView 中详细说明
android:password	以“.”显示文本
android:phoneNumber	设置为电话号码的输入方式
android:scrollHorizontally	设置文本超出 TextView 的宽度的情况下,是否出现横向滚动条
android:selectAllOnFocus	如果文本是可选的,使其获取焦点,而不是将光标移动到文本的开始位置或者末尾位置。TextView 中设置后无效果
android:shadowColor	指定文本阴影的颜色,需要与 shadowRadius 一起使用
android:shadowDx	设置阴影横向坐标开始位置
android:shadowDy	设置阴影纵向坐标开始位置
android:shadowRadius	设置阴影的半径。设置为 0.1 就变成字体的颜色,一般设置为 3.0 的效果较好

续表

XML 属性	功 能 描 述
android:singleLine	设置单行显示。如果和 layout_width 一起使用，当文本不能全部显示时，后面用“...”来表示，例如，android:text＝"test_singleLine" android:singleLine＝"true" android:layout_width＝"20dp"则显示的文本为“t...”，而不是 test_singleLine。 如果不设置 singleLine 或者设置为 false，文本将自动换行
android:text	设置显示文本
android:textAppearance	设置文字外观。如“? android: attr/textAppearanceLargeInverse” 此处引用的是系统自带的一个外观，“?”表示系统是否有这种外观，否则使用默认的外观。取值情况如下：textAppearanceButton、textAppearanceInverse、textAppearanceLarge、textAppearanceLargeInverse、textAppearanceMedium、textAppearanceMediumInverse、textAppearanceSmall、textAppearanceSmallInverse
android:textColor	设置文本颜色
android:textColorHighlight	被选中文字的底色，默认为蓝色
android:textColorHint	设置提示信息文字的颜色，默认为灰色。与 hint 一起使用
android:textColorLink	文字链接的颜色
android:textScaleX	设置文字缩放，默认为 1.0f。分别设置 0.5f/1.0f/1.5f/2.0f
android:textSize	设置文字大小，推荐度量单位“sp”，如“15sp”
android:textStyle	设置字形[bold(粗体) 0，italic(斜体) 1，bolditalic(又粗又斜) 2]。可以设置一个或多个，用“|”隔开
android:height	设置文本区域的高度，支持度量单位：px(像素)、dp、sp、in、mm(毫米)
android:maxHeight	设置文本区域的最大高度
android:minHeight	设置文本区域的最小高度
android:width	设置文本区域的宽度，支持度量单位：px(像素)、dp、sp、in、mm(毫米)

注意 表 3-16 中介绍了 TextView 最常用的属性，其他不常用的属性此处没有介绍，读者可在实际使用时再具体查询。

TextView 提供了大量的 XML 属性，通过这些属性开发人员可以控制 TextView 中文本的行为，下面通过简单示例讲解 TextView 的基本用法，代码如下。

【案例 3-18】 textview_demo.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:orientation = "vertical" >
    <!-- 设置字号为 20sp -->
    < TextView android:layout_width = "match_parent"
        android:layout_height = "wrap_content" android:text = "TextView 演示"
        android:textSize = "20sp" />
    <!-- 设置中间省略，所有字母大写，字号为 20sp，内容一行 -->
    < TextView android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:singleLine = "true" android:ellipsize = "middle"
        android:text = "TextView 演示 TextView 演示 TextView 演示 TextView
                演示 TextView 演示 TextView 演示 TextView 演示"
```

```
        android:textAllCaps = "true" android:textSize = "20sp" />
    <!-- 邮件、电话添加链接 -->
    <TextView android:layout_width = "match_parent"
        android:layout_height = "wrap_content" android:singleLine = "true"
        android:text = "邮件: zkl@163.com 电话: 053212345678"
        android:autoLink = "email|phone" android:textSize = "20sp" />
    <!-- 测试密码框 -->
    <TextView android:layout_width = "match_parent"
        android:layout_height = "wrap_content" android:text = "TextView 演示"
        android:password = "true" android:textSize = "20sp" />
</LinearLayout>
```

代码解释如下。

- 第一个 TextView 通过设置 android:textSize = "20sp"指定了字号为 20sp；其中，sp(scaled pixels，比例像素)主要用于处理字体的大小，可以根据用户的字体大小首选项进行缩放。
- 第二个 TextView 设置了 android:ellipsize = "middle"属性，当文本内容大于文本框宽度时，从中间省略文本。通过设定 android:textAllCaps = "true"属性，将该文本框中的所有字母都以大写形式进行显示。
- 第三个 TextView 设置了 android:autoLink = "email|phone"属性，该文本框会自动为文本框内的 email、电话号码添加超链接。
- 第四个 TextView 设置了 android:password = "true"属性，指定了该文本框会用“.”来显示所有字符。

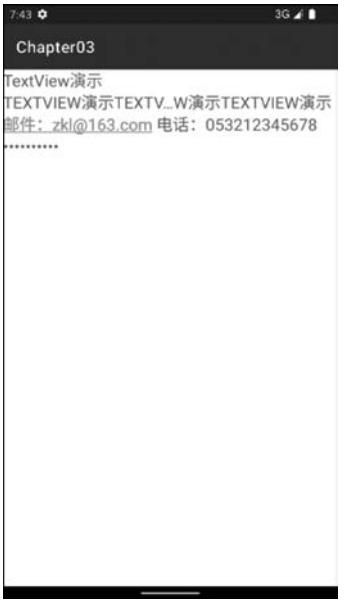


图 3-20 TextView 效果演示

通过运行 TextViewDemoActivity，效果如图 3-20 所示。

3.4.3 EditText 编辑框

EditText 是 TextView 的子类，继承了 TextView 的 XML 属性和方法，EditText 和 TextView 的最大区别是：EditText 可以接受用户的输入。EditText 作为用户与系统之间的文本输入接口，用于接收用户输入的数据并传给系统，从而使系统获取所需要的数据。EditText 组件最重要的是 inputType 属性，该属性用于指定在 EditText 输入值时所启动的虚拟键盘风格，例如，经常需要虚拟键盘只提供字符或数字。在开发过程中，常用的 inputType 属性值如表 3-17 所示。

表 3-17 inputType 属性值

属 性 值	功 能 描 述	属 性 值	功 能 描 述
text	普通文本，默认	textEmailAddress	电子邮件地址
textCapCharacters	字母大写	textEmailSubject	邮件主题
textCapWords	每个单词的首字母大写	textShortMessage	短信

续表

属 性 值	功 能 描 述	属 性 值	功 能 描 述
textLongMessage	长信息	numberSigned	带符号数字格式
textPassword	密码	numberDecimal	带小数点的浮点格式
number	数字	phone	拨号键盘
textAutoCorrect	自动完成	datetime	时间日期
textMultiLine	多行输入	date	日期键盘
textNoSuggestions	不提示	time	时间键盘
textUri	网址		

下面通过简单示例演示 inputType 属性的使用，代码如下。

【案例 3-19】 edittext_demo. xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:orientation = "vertical" >
    <!-- 多行效果 -->
    < EditText android:layout_width = "match_parent"
        android:layout_height = "wrap_content" android:inputType = "textMultiLine"
        android:hint = "多行效果" android:textSize = "20sp" />
    <!-- 拨号键盘 -->
    < EditText android:layout_marginTop = "15dp"...
        android:inputType = "phone" android:textSize = "20sp" />
     <!-- 密码类型 -->
    < EditText android:layout_marginTop = "15dp"...
        android:inputType = "textPassword"
        android:hint = "输入密码" android:textSize = "20sp" />
</LinearLayout >
```

代码解释如下。

- 上述代码中，三个 EditText 都通过 android:hint 属性指定了文本框的提示信息；当用户输入内容之前，文本框内默认显示指定的提示信息，当用户输入信息时，提示信息被清除。
- 第一个 EditText 通过设置 android:inputType = "textMultiLine"属性来指定该文本框允许多行输入。
- 第二个 EditText 通过设置 android:inputType = "phone"属性来指定该文本框只能接受数值的输入。
- 第三个 EditText 通过设置 android:inputType = "textPassword"属性来指定该文本框是一个密码框。

通过 Activity 运行上述代码，效果如图 3-21 所示。

图 3-21 EditText 效果演示

3.4.4 Button 按钮

Button 继承了 TextView，主要用于在 UI 界面上生成一个按钮，当用户单击按钮时，会触发一个 OnClick 事件。按钮的使用相对比较容易，通过 android:backgroud 属性为按钮

指定背景颜色或背景图片，使用各式各样的背景图片可以实现各种不规则形状的按钮。

Button 类通过继承父类的方法来实现对按钮组件的操作，表 3-18 列举了 Button 类的常用方法。

表 3-18 Button 类的方法

方 法	功 能 描 述
onKeyDown()	当用户按键时，该方法被调用
onKeyUp()	当用户按键弹起后，该方法被调用
onKeyLongPress()	当用户保持按键时，该方法被调用
onKeyMultiple()	当用户多次按键时，该方法被调用
invalidateDrawable()	用于刷新 Drawable 对象
onPreDraw()	用于设置视图显示，例如，在视图显示之前调整滚动轴的边界
setOnKeyListener()	用于设置按键监听器
setOnClickListener()	用于设置单击监听器

下述代码以 setOnClickListener() 为例，通过一个模拟登录操作来演示 Button、TextView 和 EditView 的使用。界面布局的代码如下。

【案例 3-20】 login.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:orientation = "vertical" >
    <!-- 标题 ① -- >
    < TextView android:layout_width = "wrap_content"
        android:layout_height = "wrap_content" android:layout_gravity = "center"
        android:text = "用户登录" android:textSize = "35sp" />
    <!-- 用户名 ② -- >
    < LinearLayout android:layout_width = "match_parent"
        android:layout_height = "wrap_content" android:layout_marginTop = "10dp" >
        < TextView android:layout_width = "wrap_content"
            android:layout_height = "wrap_content" android:text = "用户名: " />
        < EditText android:id = "@ + id/userNameTxt" .../>
    </LinearLayout >
    <!-- 密码 ③ -- >
    < LinearLayout android:layout_width = "match_parent"...>
        < TextView android:text = "密 码: " .../>
        < EditText android:id = "@ + id/passwordTxt"
            android:inputType = "textPassword".../>
    </LinearLayout >
   <!-- "登录"按钮 ④ -- >
    < Button android:id = "@ + id/loginBtn" android:text = "登录" .../>
    <!-- 成功或失败提示 ⑤ -- >
    < TextView android:id = "@ + id/tipsTxt"
        android:text = "显示成功或失败" android:visibility = "gone" .../>
</LinearLayout >
```

代码解释如下。

- 上述代码中，标号①处用于显示的是当前界面的标题，并将 TextView 的字号设为 35sp，相对比较醒目。
- 标号②处通过 LinearLayout 将 TextView 和 EditText 组合起来，用于接收用户名的输入。

- 标号③处通过 LinearLayout 将 TextView 和 EditText 组合起来，用于接收密码的输入，其中，EditText 的 inputType 设置为 textPassword，表示该文本框当作密码框使用。
- 标号④定义了一个“登录”按钮，在 Activity 中用于实现登录的业务逻辑处理。
- 标号⑤定义了一个 TextView，用于显示用户登录成功或失败后的提示，例如，用户名不存在、密码错误等。通过设置 android:visibility="gone"，默认隐藏该提示。

接下来在相应的 Activity 中实现登录的业务逻辑，此处仅实现一个简单的登录验证，并不做真正的登录跳转，代码如下。

【案例 3-21】 LoginActivity.java

```
public class LoginActivity extends AppCompatActivity {
    //用户名 ①
    EditText userNameTxt;
    //密码
    EditText passwordTxt;
    //"登录"按钮
    Button loginBtn;
    //提示
    TextView tipsTv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login);
        //初始化各个组件 ②
        userNameTxt = (EditText)findViewById(R.id.userNameTxt);
        passwordTxt = (EditText) findViewById(R.id.passwordTxt);
        tipsTv = (TextView) findViewById(R.id.tipsTxt);
        loginBtn = (Button)findViewById(R.id.loginBtn);
        //实现单击 Button 的业务逻辑 ③
        loginBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //获取用户名
                String userName = userNameTxt.getText().toString();
                //获取密码
                String password = passwordTxt.getText().toString();
                //判断
                //判断用户名
                if(!"admin".equals(userName)) {
                    tipsTv.setText("用户名不存在!");
                    tipsTv.setVisibility(View.VISIBLE);
                    return;
                }
                if(!"1".equals(password)) {
                    tipsTv.setText("密码不正确!");
                    tipsTv.setVisibility(View.VISIBLE);
                    return;
                }
                if("admin".equals(userName)&&"1".equals(password)) {
                    tipsTv.setText("登录成功!");
```

```
                    tipsTv.setVisibility(View.VISIBLE);
                }
            }
        });
    }
}
```

代码解释如下。

- 上述代码中，标号①处定义了一个 EditText 类型的属性变量 userNameTxt，用于获取界面传递过来的对象，其他属性的定义功能类似，此处不再赘述。
- 标号②处对标号①处所定义的各个属性变量进行初始化，通过对属性变量赋值，使其能够进行后续的业务逻辑操作。
- 标号③处实现了“登录”按钮 loginBtn 的业务逻辑。逻辑相对比较简单：如果用户名无效，则显示“用户名不存在”；如果密码不对，则显示“密码不正确!”；如果用户输入的用户名和密码都正确，则显示“登录成功!”。

注意 本案例仅演示 Button、TextView 和 EditText 的使用，未涉及更复杂的应用逻辑。在实际开发中，用户登录时需要查询后台数据库来验证用户名和密码是否正确。

运行上述代码，当用户输入错误时进行相应的错误提示，例如，用户名输入“admi”，密码任意输入“1”，显示的界面如图 3-22 所示。当用户输入正确的用户名和密码时，显示的界面的如图 3-23 所示。

读者可以进一步完善该案例，例如，可以增加用户名和密码的空校验等功能。

图 3-22 用户输入错误时的情况

图 3-23 用户输入正确时的情况

3.4.5 单选按钮和单选按钮组

在一组按钮中有且仅有一个按钮能够被选中，当选择按钮组中某个按钮时会取消其他按钮的选中状态。上述效果需要 RadioButton 和 RadioGroup 配合使用才能实现。RadioGroup 是单选按钮组，是一个允许容纳多个 RadioButton 的容器。在没有 RadioGroup 的情况下，RadioButton 可以分别被选中；当多个 RadioButton 同在一个 RadioGroup 按钮组中，RadioButton 只允许选择其中之一。RadioButton 和 RadioGroup 的关系如下。

- RadioButton 表示单个圆形单选框，而 RadioGroup 是一个可以容纳多个 RadioButton 的容器。
- 同一个 RadioGroup 中，只能有一个 RadioButton 被选中。
- 不同的 RadioGroup 中，RadioButton 互不影响，即如果组 A 中有一个被选中，组 B 中依然可以有一个被选中。
- 通常情况下，一个 RadioGroup 中至少有两个 RadioButton。
- 大部分应用场景下，建议一个 RadioGroup 中的 RadioButton 默认会有一个被选中，并将其放在 RadioGroup 中的起始位置。

RadioGroup 类是 LinearLayout 的子类，其常用的设置和控制单选按钮组的方法如表 3-19 所示。

表 3-19 RadioButton 相关方法

方 法	功 能 描 述
getCheckedRadioButtonId()	获取被选中按钮的 ID
clearCheck()	清除选中状态
check (int id)	通过参数 ID 来设置该选项为选中状态；如果传入－1 则清除单选按钮组的选中状态，相当于调用 clearCheck()操作
setOnCheckedChangeListener (RadioGroup.OnCheckedChangeListener listener)	在一个单选按钮组中，当该单选按钮选中状态发生改变时所要调用的回调函数。当 RadioButton 的 checked 属性为 true 时，check(id)方法不会触发 onCheckedChanged 事件
addView(View child,int index,ViewGroup.LayoutParams params)	使用指定的布局参数添加一个子视图。 • child：所要添加的子视图。 • index：将要添加子视图的位置。 • params：所要添加的子视图的布局参数
getText()	用于获取单选框的值

此外，通过 OnCheckedChangeListener 监听器对单选按钮的状态切换进行监听并处理。

下面通过一个简单的实例演示 RadioButton 和 RadioGroup 的使用，界面布局的代码如下。

【案例 3-22】 radiobutton_demo.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:layout_marginRight = "5dp"
    android:orientation = "vertical" >
    <!-- 显示选择的内容 ① -->
    < TextView android:id = "@ + id/chooseTxt" ...
```

```
        android:text = "我选择的是...?" android:textSize = "30sp" />
    <!-- 单选按钮组 ② -->
    < RadioGroup android:id = "@ + id/radioGroup" ...>
        < RadioButton android:id = "@ + id/radioButton1" android:text = "按钮 1" .../>
        < RadioButton android:id = "@ + id/radioButton2" android:text = "按钮 2" .../>
    </RadioGroup >
    <!-- 清除所有选中状态 ③ -->
    < Button android:id = "@ + id/radio_clearBtn" android:text = "清除选中" .../>
    <!-- 往按钮组中添加新的单选按钮 ④ -->
    < Button android:id = "@ + id/radio_addBtn" android:text = "添加子项" .../>
</LinearLayout >
```

代码解释如下。

- 上述代码中,标号①处用于显示当前选中按钮的标题。
- 标号②处定义了一个单选按钮组,并为该按钮组添加了两个单选按钮。
- 标号③处定义了一个“清除选中”按钮,用于清除按钮组中所有单选按钮的选中状态。
- 标号④处定义了一个“添加子项”按钮,用于向按钮组中添加新的互斥的单选按钮。

接下来在对应的 Activity 中演示按钮组的使用,实现清除按钮组中所有按钮的选中状态,以及向按钮组中添加新的单选按钮的功能,代码如下。

【案例 3-23】 RadioButtonActivity.java

```
public class RadioButtonActivity extends AppCompatActivity {
    //显示选择的单选按钮文本 ①
    private TextView chooseTxt;
    //按钮组
    private RadioGroup radioGroup;
    //多个单选按钮
    private RadioButton radioButton1;
    private RadioButton radioButton2;
    //清除按钮
    private Button radioClearBtn;
    //新增按钮
    private Button radioAddBtn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.radiobutton_demo);
        //初始化按钮 ②
        chooseTxt = (TextView)findViewById(R.id.chooseTxt);
        radioGroup = (RadioGroup)findViewById(R.id.radioGroup);
        radioButton1 = (RadioButton)findViewById(R.id.radioButton1);
        radioButton2 = (RadioButton)findViewById(R.id.radioButton2);
        radioGroup.setOnCheckedChangeListener(onCheckedChangeListener);
        //清除选中状态
        radioClearBtn = (Button)findViewById(R.id.radio_clearBtn);
        radioClearBtn.setOnClickListener(onClickListener);
        //增加子选项
        radioAddBtn = (Button)findViewById(R.id.radio_addBtn);
        radioAddBtn.setOnClickListener(onClickListener);
```

```
    }
    /**
     * 定义按钮组的监听事件 ③
     */
    private OnCheckedChangeListener onCheckedChangeListener
        = new OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(RadioGroup group, int checkedId) {
        int id = group.getCheckedRadioButtonId();
            switch (group.getCheckedRadioButtonId()) {//获取当前选中的按钮的 ID
            case R.id.radioButton1:
                chooseTxt.setText("我选择的是:" + radioButton1.getText());
                break;
            case R.id.radioButton2:
                chooseTxt.setText("我选择的是:" + radioButton2.getText());
                break;
            default:
                chooseTxt.setText("我选择的是:新增");
                break;
            }
        }
    };
    //定义清除状态按钮和新增按钮的单击事件 ④
    private OnClickListener onClickListener = new OnClickListener() {
        @Override
        public void onClick(View view) {
            switch (view.getId()) {
            case R.id.radio_clearBtn:
                radioGroup.check(-1); //清除选项
                chooseTxt.setText("我选择的是...?");
                break;
            case R.id.radio_addBtn:
                //新增子选项
                RadioButton newRadio = new RadioButton(RadioButtonActivity.this);
                newRadio.setLayoutParams(new LayoutParams(
                        LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
                newRadio.setText("新增");
                radioGroup.addView(newRadio,radioGroup.getChildCount());
                break;
           default:
                break;
            }
        }
    };
}
```

代码解释如下。

- 上述代码中，标号①处定义了一个 TextView 类型的属性变量 chooseTxt，用于获取当前被选中按钮的文本，其他属性的定义请见注释，此处不再赘述。
- 标号②处对标号①处所定义的各个属性变量进行初始化，通过对属性变量的赋值，使其可以进行后续的业务逻辑操作。

- 标号③处定义了一个按钮组监听器对象，用于获取当前在按钮组中选中的单选按钮对象，并将文本显示在 chooseTxt 对象上。
- 标号④处定义一个普通按钮监听器对象，用于实现 radioClearBtn 和 radioAddBtn 的业务逻辑功能，当用户单击 radioClearBtn 按钮时，按钮组中被选中的单选按钮状态被清空；当用户单击 radioAddBtn 时，系统会在 radioGroup 对象中增加一个单选按钮对象。

运行上述代码，选中一个单选按钮的效果如图 3-24 所示。当用户单击“添加子项”按钮后，并选中新增的选项，效果如图 3-25 所示。

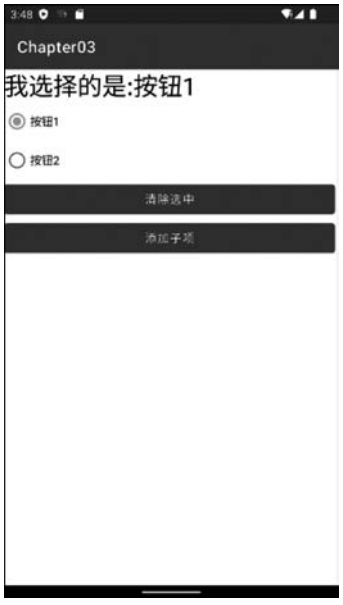


图 3-24 选中单选按钮

图 3-25 添加子项

可以通过 android:drawableRight 属性使单选按钮在文本的右边显示，对布局文件进行修改，代码如下。

【案例 3-24】 radiobutton_demo. xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:layout_marginRight = "5dp" android:orientation = "vertical" >
   <!-- 显示选择的内容 -->
    < TextView android:id = "@ + id/chooseTxt" ...
        android:text = "我选择的是...?" android:textSize = "30sp" />
    <!-- 单选按钮组 -->
    < RadioGroup android:id = "@ + id/radioGroup" ...>
        < RadioButton android:id = "@ + id/radioButton1" ...
            android:button = "@null"
            android:drawableRight = "@android:drawable/btn_radio"
            android:text = "按钮 1" />
     ...
```

运行修改后的代码，效果如图 3-26 所示。

图 3-26　改变 RadioButton 样式

3.4.6　CheckBox 复选框

CheckBox 复选框是一种具有双状态的按钮，具有选中和未选中两种状态。在布局文件中定义复选框时，对每一个复选框注册 OnCheckedChangeListener 事件监听，然后在 onCheckedChanged()事件处理方法中根据 isChecked 参数来判断选项是否被选中。

CheckBox 和 RadioButton 的主要区别如下。

- RadioButton 单选按钮被选中后，再次单击时无法改变其状态，而 CheckBox 复选框被选中后，可以通过单击来改变其状态。
- 在 RadioButton 单选按钮组中，只允许选中一个；而在 CheckBox 复选框组中，允许同时选中多个。
- 在大部分 UI 框架中，RadioButton 默认都以圆形表示，CheckBox 默认都以矩形表示。

下述代码通过一个简单的示例演示 CheckBox 的用法，以“体育爱好”的多选为例，人们的“体育爱好”可能有足球、篮球等，而人的性别选择有所不同，性别只能选择“男”或“女”，且两者互斥。示例的界面布局代码如下。

【案例 3-25】 checkbox_demo. xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:orientation = "vertical" >
    <!-- 基本显示 ① -->
    < TextView android:text = "@string/title" android:textSize = "20sp" ...
        android:textStyle = "bold" android:textColor = "#FFFFFF" />
    <!-- 足球 ② -->
    < CheckBox android:id = "@ + id/checkbox1" ...
```

```
        android:text = "@string/football" android:textSize = "16sp" />
    <!-- 篮球 ③ -->
    < CheckBox android:id = "@ + id/checkbox2" ...
        android:text = "@string/basketball" android:textSize = "16sp" />
    <!-- 排球 ④ -->
    < CheckBox android:id = "@ + id/checkbox3" ...
        android:text = "@string/volleyball" android:textSize = "16sp" />
</LinearLayout >
```

代码解释如下。

- 上述代码中,标号①处的 TextView 用于显示用户的标题。
- 标号②处定义的是“足球”复选框。
- 标号③处定义的是“篮球”复选框。
- 标号④处定义的是“排球”复选框。

上述代码中,复选框的文本部分使用了字符串资源,例如,“足球”的文本是引用的 strings. xml 文件中的字符串,其中,strings. xml 中的字符串定义如下。

【案例 3-26】 strings. xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< resources >
    < string name = "title">你喜欢的运动是: </string >
    < string name = "app_name">复选框测试</string >
    < string name = "football">足球</string >
    < string name = "basketball">篮球</string >
    < string name = "volleyball">排球</string >
</resources >
```

通常在开发过程中使用 strings. xml 文件的目的如下。

- 国际化: Android 建议将屏幕中显示的文字定义在 strings. xml 中,如果今后需要进行国际化时仅需要修改 string. xml 文件即可。例如,原本开发的应用是面向国内用户的,在屏幕上使用中文,当需要将应用国际化时,用户希望屏幕上所显示的内容是英文,此时如果没有把文字信息定义在 string. xml 中,就需要修改程序的内容来实现。但如果把所有屏幕上出现的文字信息都集中存放在 string. xml 文件中,在需要国际化时只需修改 string. xml 中所定义的字符串资源即可,Android 操作系统会根据用户手机的语言环境和国家来自动选择相应的 string. xml 文件,实现起来更加方便。
- 为了减少应用的体积,降低数据的冗余,例如,在应用中要使用“我们一直在努力”这段文字 1000 次,如果不将“我们一直在努力”定义在 string. xml 文件中,而是在每次使用时直接使用该字符串,这样会浪费大量的空间,并且维护起来较为麻烦。

注意 关于各种类型的资源文件的讲解会在后面的章节中介绍,此处不再赘述。

下面在相应的 Activity 中演示复选框的使用,当用户选择不同的“爱好”时,在屏幕上显示用户的选择结果,代码如下。

【案例 3-27】 CheckBoxDemoActivity.java

```
public class CheckBoxDemoActivity extends AppCompatActivity {
    //声明复选框 ①
    private CheckBox footballChx;
    private CheckBox basketballChx;
    private CheckBox volleyballChx;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.checkbox_demo);
        //通过 findViewById 获得 CheckBox 对象 ②
        footballChx = (CheckBox) findViewById(R.id.footballChx);
        basketballChx = (CheckBox) findViewById(R.id.basketballChx);
        volleyballChx = (CheckBox) findViewById(R.id.volleyballChx);
        //注册事件监听器 ③
        footballChx.setOnCheckedChangeListener(listener);
        basketballChx.setOnCheckedChangeListener(listener);
        volleyballChx.setOnCheckedChangeListener(listener);
    }

    //响应事件 ④
    private OnCheckedChangeListener listener = new OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView,
                boolean isChecked) {

            switch (buttonView.getId()) {
            case R.id.footballChx:
                //选择足球
                if (isChecked) {
                    //Toast 的使用 ⑤
                    Toast.makeText(CheckBoxDemoActivity.this, "你喜欢足球",
                            Toast.LENGTH_LONG).show();
                }
                break;
            case R.id.basketballChx:
                //选择篮球
                if (isChecked) {
                    Toast.makeText(CheckBoxDemoActivity.this, "你喜欢篮球",
                            Toast.LENGTH_LONG).show();
                }
            case R.id.volleyballChx:
                //选择排球
                if (isChecked) {
                    Toast.makeText(CheckBoxDemoActivity.this, "你喜欢排球",
                            Toast.LENGTH_LONG).show();
                }
            default:
                break;
            }
        }
    };
}
```

代码解释如下。

- 上述代码中，标号①处定义了三个 CheckBox 复选框，供用户进行选择。
- 标号②处对标号①处所定义的各个属性变量初始化，通过对属性变量赋值，使其可以进行后续的业务逻辑操作。
- 标号③处分别为三个 CheckBox 对象设置监听器，用于监听各自的选中或取消事件。
- 标号④处定义了一个监听器对象，用于监听并实现三个 CheckBox 的业务逻辑功能，当用户单击不同的 CheckBox 时，屏幕上会通过 Toast 对象显示相应的文本信息。

运行上述 Activity，并选择“篮球”复选框时，界面效果如图 3-27 所示。

图 3-27 爱好的选择

注意 Toast 是 Android 中用来显示提示信息的一种机制，与 Dialog 不同的是：Toast 提示没有焦点且时间有限，在一定的时间后会自动消失。Toast 使用简单，主要用于向用户显示提示消息，在后续章节会有详细的介绍。

3.4.7 开关控件

ToggleButton、Switch、CheckBox 和 RadioButton 组件均继承自 android.widget.CompoundButton，都是选择类型的按钮，因此这些组件的用法非常相似。CompoundButton 按钮共有两种状态：选中(checked)和未选中(unchecked)状态。而 Switch 控件是 Android 4.0 后出现的控件。

ToggleButton 所支持的 XML 属性和方法如表 3-20 所示。

表 3-20 ToggleButton 的 XML 属性和相关方法

XML 属性	对应方法	功能描述
android:checked	setChecked(boolean)	设置该按钮是否被选中
android:textOff	setTextOff(CharSequence)	设置按钮的状态关闭时所显示的文本
android:textOn	setTextOn(CharSequence)	设置按钮的状态打开时所显示的文本

下述代码通过一个简单的示例演示 ToggleButton 的用法；当 ToggleButton 处于选中的状态时，文本显示“已开启”，当 ToggleButton 处于未选中状态时，文本显示“已关闭”。首先实现界面布局，代码如下。

【案例 3-28】 togglebutton_demo.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:orientation = "horizontal" >
     <!-- 文本展示 ① -->
     < TextView android:id = "@ + id/tvSound" ...
```

```
            android:text = "已开启" android:textColor = "@android:color/black"
            android:textSize = "14.0sp" />
    <!-- 定义 ToggleButton 对象 ② -->
    < ToggleButton android:id = "@ + id/tglSound"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:layout_marginLeft = "10dp"
            android:checked = "true"
            android:text = ""
            android:textOff = "OFF"
            android:textOn = "ON" />
</LinearLayout >
```

代码解释如下。

- 上述代码中，标号①处的 TextView 用于显示 ToggleButton 按钮的 On/Off 时的标题。
- 标号②处定义了一个 ToggleButton，用于测试按钮的开启或关闭。

然后，在 Activity 中展示 ToggleButton 的使用：当用户选择了 On/Off 时，在屏幕上显示用户的选择。对应的 Activity 代码如下。

【案例 3-29】 ToggleButtonDemoActivity.java

```
public class ToggleButtonDemoActivity extends AppCompatActivity {
    //声明 XML 中定义的组件 ①
    private ToggleButton mToggleButton;
    private TextView tvSound;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.toggleswitch_demo);
        initView();
    }
    //初始化控件方法 ②
    private void initView() {
        //获取控件
        mToggleButton = (ToggleButton) findViewById(R.id.tglSound);
        tvSound = (TextView) findViewById(R.id.tvSound);
        //注册监听器，添加监听事件 ③
        mToggleButton.setOnCheckedChangeListener(new OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView,
                    boolean isChecked) {
                if (isChecked) {
                    tvSound.setText("已开启");
                } else {
                    tvSound.setText("已关闭");
                }
            }
        });
    }
}
```

代码解释如下。

- 上述代码中，标号①处分别声明了 ToggleButton 类型和 TextView 类型的属性变量。
- 标号②处对标号①处所定义的各个属性变量进行初始化，通过对属性变量的赋值，使其可以进行后续的业务逻辑操作。
- 标号③处对 ToggleButton 对象注册监听器，用来监听按钮的开启或关闭事件。

运行上述代码后，当用户单击 ON 按钮后，显示效果如图 3-28 所示。

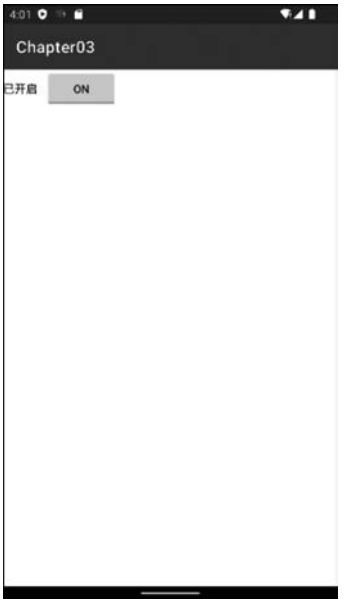


图 3-28 选中状态

如图 3-28 所示，当用户切换按钮变为 ON 状态时，显示的文本变为“已开启”。从图中可以看出，默认的 ToggleButton 比较难看，在实际开发中可以通过设置按钮的背景图片来实现较为美观的开/关。本示例中提供了用于切换 ON/OFF 的较为美观的图片，通过设置按钮的背景图片来实现较为美观的效果，修改后的代码如下。

【案例 3-30】 togglebutton_demo.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:orientation = "horizontal" >
      <!-- 文本展示 ① -- >
      < TextView android:id = "@ + id/tvSound"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:text = "已开启"
            android:textColor = "@android:color/black"
            android:textSize = "14.0sp" />
      <!-- 定义 ToggleButton 对象 ② -- >
        < ToggleButton android:id = "@ + id/tglSound"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:layout_marginLeft = "10dp"
            android:background = "@drawable/selector_btn_toggle"
            android:checked = "true"
            android:text = ""
            android:textOff = ""
            android:textOn = "" />
</LinearLayout >
```

代码解释如下。

- 上述代码中，标号①处的 TextView 用于显示 ON/OFF 后的标题。
- 标号②处定义了一个 ToggleButton 按钮，用于显示开启或关闭；此时，需要将属性 android:textOff 和 android: textOn 设置为空，否则将会覆盖在图片上。然后将 android:backgroud 的属性值设置为 selector_btn_toggle，该值对应的并不是一幅图片，而是一个资源文件 selector_btn_toggle.xml。

在资源目录 res\drawable 下，创建 selector_btn_toggle.xml 资源文件，代码如下。

【案例 3-31】 selector_btn_toggle.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< selector xmlns: android = " http://schemas. android. com/
apk/res/android">
    < item android:state_checked = "true" android:drawable =
"@drawable/btn_open" />
    < item android:drawable = "@drawable/btn_close" />
</selector >
```

在上述资源文件中，指定了 ToggleButton 在默认状态下的背景图片和选中状态下的图片背景。运行更改后的代码，展示效果如图 3-29 所示。

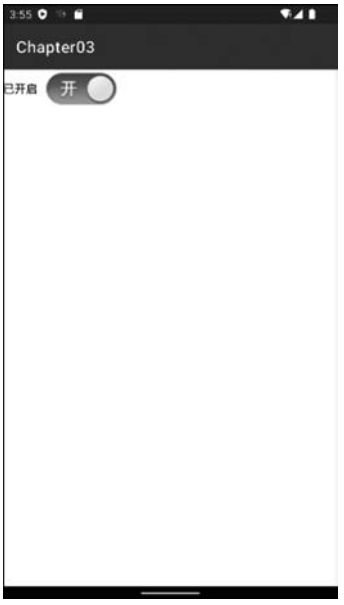


图 3-29　ToggleButton 背景优化

通过效果可以看出，美化后的 ToggleButton 更注重用户的体验。

Switch 的使用方式与 ToggleButton 类似，Switch 所支持的 XML 属性和方法如表 3-21 所示。

表 3-21　Switch 的 XML 属性及对应方法

XML 属性	对应方法	功能描述
android:checked	setChecked(boolean)	设置当前按钮的状态，选中或未选中
android:textOff	setTextOff(CharSequence)	设置按钮关闭状态所显示的文本
android:textOn	setTextOn(CharSequence)	设置按钮打开状态所显示的文本
android:switchMinWidth	setSwitchMinWidth(int)	设置开关的最小宽度
android:textStyle	setSwitchTypeface (Typeface , int)	设置开关的文本风格
android:typeface	setSwitchTypeface(Typeface)	设置开关的文本的字体风格
android:switchPadding	setSwitchPadding(int)	设置开关与标题文本之间的空白
android:thumb	setThumbResource(int)	使用自定义的 Drawable 来绘制开关的开关按钮
android:track	setTrackResource(int)	使用自定义的 Drawable 来绘制开关的开关轨道

下述代码通过一个示例演示 Switch 的使用。通过改变 Switch 状态来实现界面布局中的 LinearLayout 布局的方向在水平和垂直布局之间切换，布局文件代码如下。

【案例 3-32】 switch_demo.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:orientation = "vertical" >
    <!-- 定义一个 Switch 组件 ① -->
    < Switchandroid:id = "@ + id/switcher" android:checked = "true"
        android:textOff = "横向排列"android:textOn = "纵向排列"
        android:showText = "ture"android:thumb = "@drawable/check" />
    <!-- 定义一个可以动态改变方向的线性布局 ② -->
    < LinearLayout android:id = "@ + id/test" android:orientation = "vertical" ...>
```

```
        <TextView android:text="测试文本 1" .../>
        <TextView android:text="测试文本 2" .../>
        <TextView android:text="测试文本 3" .../>
    </LinearLayout>
</LinearLayout>
```

代码解释如下：

- 上述代码中，标号①处定义了一个 Switch 组件，并将 android:thumb 的属性设置为“@drawable/check”。
- 标号②处定义了一个可以动态改变方向的线性布局，其中包含 3 个文本框用于显示效果。

下面在 Activity 中演示 Switch 的使用：当用户选择了“横向排列”/“纵向排列”时，界面布局发生相应的变化。Activity 代码的实现如下。

【案例 3-33】 SwitchDemoActivity.java

```
public class SwitchDemoActivity extends AppCompatActivity {
    //定义变量 ①
    Switch switcher;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.switch_demo);
        //初始化组件②
        switcher = (Switch) findViewById(R.id.switcher);
        final LinearLayout test = (LinearLayout) findViewById(R.id.test);
        OnCheckedChangeListener listener = new OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton button,
                    boolean isChecked) {
                if (isChecked) {
                    //设置 LinearLayout 垂直布局
                    test.setOrientation(LinearLayout.VERTICAL);
                    switcher.setChecked(true);
                } else {
                    //设置 LinearLayout 水平布局
                    test.setOrientation(LinearLayout.HORIZONTAL);
                    switcher.setChecked(false);
                }
            }
        };
        //为 switch 组件添加事件监听器 ③
        switcher.setOnCheckedChangeListener(listener);
    }
}
```

代码解释如下。

- 上述代码中，标号①分别声明了一个 Switch 类型的属性变量。
- 标号②处用于初始化标号①处所声明的属性变量，通过对属性变量的赋值，使其可以进行后续的业务逻辑操作。

- 标号③处对 Switch 对象设置监听器，用于监听按钮的开启或关闭事件。当事件发生时，判断按钮的开启/关闭状态，并切换界面的布局。

运行上述代码后，界面效果如图 3-30 所示。当用户再次单击“纵向排列”按钮时，系统会自动切换到“横向排列”状态，效果如图 3-31 所示。

图 3-30　Switch 实现纵向布局

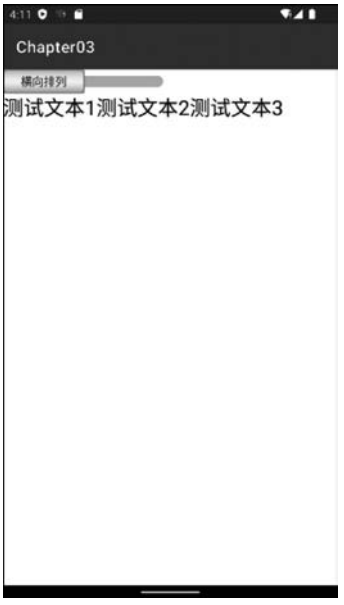


图 3-31　Switch 实现横向布局

3.4.8　图片视图(ImageView)

ImageView 继承自 View 组件，主要用于显示图像资源(例如图片等)。ImageView 可以定义所显示的尺寸等。此外，ImageView 还派生了 ImageButton、ZoomButton 等组件。ImageView 所支持的 XML 属性和方法如表 3-22 所示。

表 3-22　ImageView 的 XML 属性及方法

XML 属性	对 应 方 法	功 能 描 述
android:adjustViewBounds	setAdjustViewBounds(boolean)	是否保持宽高比。需要与 maxWidth、MaxHeight 一起使用，单独使用没有效果
android:cropToPadding	setCropToPadding(boolean)	截取指定区域是否使用空白代替。单独设置无效果，需要与 scrollY 一起使用
android:maxHeight	setMaxHeight(int)	设置 View 的最大高度，单独使用无效，需要与 setAdjustViewBounds()方法一起使用。如果想设置图片固定大小，又想保持图片宽高比，需要如下设置。 (1) 设置 setAdjustViewBounds 为 true。 (2) 设置 maxWidth 和 MaxHeight 属性。 (3) 设置 layout_width 和 layout_height 均为 wrap_content
android:maxWidth	setMaxWidth(int)	设置 View 的最大宽度。具体描述与 SetMaxHeight()方法类似

续表

XML 属性	对 应 方 法	功 能 描 述
android:src	setImageResource(int)	设置 ImageView 所显示的 Drawable 对象
android:scaleType	setScaleType (ImageView.ScaleType)	设置所显示的图片如何缩放或移动以适应 ImageView 的大小

ImageView 的 android:scaleType 属性可以指定如下属性值。

- matrix：用矩阵来绘图。
- fitXY：拉伸图片(不按比例)以填充 View 的宽高。
- fitStart：按比例拉伸图片,图片拉伸后的高度为 View 的高度,且显示在 View 的左边。
- fitCenter：按比例拉伸图片,图片拉伸后的高度为 View 的高度,且显示在 View 的中间。
- fitEnd：按比例拉伸图片,图片拉伸后的高度为 View 的高度,且显示在 View 的右边。
- center：按原图大小显示图片,当图片宽高大于 View 的宽高时,截图图片中间部分显示。
- centerCrop：按比例放大原图,直至等于 View 某边的宽高。
- centerInside：当原图宽高等于 View 的宽高时,按原图大小居中显示；反之将原图缩放至 View 的宽高居中显示。

此外,为了控制 ImageView 所显示的图片,该组件提供了如下方法。

- setImageBitmap(Bitmap)：使用 Bitmap 位图来设置 ImageView 所显示的图片。
- setImageDrawable(Drawable)：使用 Drawable 对象来设置 ImageView 所显示的图片。
- setImageResource(int)：使用图片资源 ID 来设置 ImageView 所显示的图片。
- setImagURI(Uri)：使用图片的 URI 来设置 ImageView 所显示的图片。

下面通过一个简单示例演示 ImageView 的使用。通过单击“下一页”/“上一页”按钮,实现国旗的切换,对应的布局文件代码如下。

【案例 3-34】 imageview_demo.xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:orientation = "vertical" >
    <!-- 国旗及文字 ① -->
    < LinearLayout android:layout_gravity = "center"
        android:orientation = "vertical" ...>
        < ImageView android:id = "@ + id/guoqiImageView"
            android:layout_width = "200dp" android:layout_height = "wrap_content"
            android:layout_gravity = "center" android:layout_marginTop = "30dp"
            android:background = "@android:color/white"
            android:scaleType = "fitCenter" android:src = "@drawable/china" />
        < TextView android:id = "@ + id/guoqiTxt" ...
            android:layout_gravity = "center" android:text = "中国" />
    </LinearLayout >
    <!-- 分页 ② -->
    < LinearLayout android:gravity = "center"
        android:orientation = "horizontal" ...>
        < ImageButton android:id = "@ + id/backImageBtn"
            android:layout_width = "50dp" android:layout_height = "50dp"
```

```
            android:layout_gravity = "center" android:src = "@drawable/back" />
        < ImageButton android:id = "@ + id/forwardImageBtn"
            android:layout_marginLeft = "20dp" android:layout_width = "50dp"
            android:layout_height = "50dp" android:layout_gravity = "center"
            android:src = "@drawable/forward" />
    </LinearLayout >
</LinearLayout >
```

代码解释如下。

- 上述代码中，标号①处定义了一个 ImageView 组件，用来显示国旗的图片；通过设置 android:scaleType＝"fitCenter"属性，使用拉伸后图片的高度作为 View 的高度，且在 View 的中间显示。同时还定义了一个 TextView 组件用来显示国别。
- 标号②处定义了两个 ImageButton 组件，分别用于显示“上一页”和“下一页”的国旗和国别。

下面在 Activity 中演示了图片分页效果：当用户分别单击“上一页”/“下一页”按钮时，在屏幕上会显示相应的国旗和国别。对应的 Activity 代码实现如下。

【案例 3-35】 ImageViewDemoActivity.java

```
public class ImageViewDemoActivity extends AppCompatActivity {
    //定义变量 ①
    //国旗对应的 ImageView
    ImageView flagImageView;
    TextView flagTxt;
    //上一页
    ImageButton backImageBtn;
    //下一页
    ImageButton forwardImageBtn;
    //国旗数组 中国 德国 英国 ②
    int[ ]flag = {R.drawable.china,R.drawable.germany,R.drawable.britain};
    String[ ]flagNames = {"中国","德国","英国"};
    //当前页为默认第一页
    int currentPage = 0;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.imageview_demo);
        //初始化组件 ③
        flagImageView = (ImageView) findViewById(R.id. flagImageView);
        //国旗名称
        guoqiTxt = (TextView)findViewById(R.id. flagTxt);
        //上一页、下一页
        backImageBtn = (ImageButton)findViewById(R.id.backImageBtn);
        forwardImageBtn = (ImageButton)findViewById(R.id.forwardImageBtn);
        //注册监听器
        backImageBtn.setOnClickListener(onClickListener);
        forwardImageBtn.setOnClickListener(onClickListener);
    }
    //定义单击事件监听器 ④
    private OnClickListener onClickListener = new OnClickListener() {
        @Override
```

```
        public void onClick(View v) {
            switch (v.getId()) {
            case R.id.backImageBtn:
                if(currentPage == 0){
                    Toast.makeText(ImageViewDemoActivity.this,
                            "第一页,前面没有了", Toast.LENGTH_SHORT).show();
                    return;
                }
                //上翻一页
                currentPage-- ;
                //设置国旗图片
                flagImageView.setImageResource(flag[currentPage]);
                //设置国旗名字
                flagTxt.setText(flagNames[currentPage]);
                break;
            case R.id.forwardImageBtn:
                if(currentPage == (flag.length-1)){
                    Toast.makeText(ImageViewDemoActivity.this,
                            "最后一页,后面没有了", Toast.LENGTH_SHORT).show();
                    return;
                }
                //下翻一页
                currentPage++ ;
                //设置国旗图片
                flagImageView.setImageResource(flag[currentPage]);
                //设置国旗名字
                flagTxt.setText(flagNames[currentPage]);
                break;
            default:
                break;
            }
        }
    };
}
```

代码解释如下。

- 上述代码中,标号①分别声明了 ImageView 类型和 ImageButton 类型的属性变量。
- 标号②处定义了两个数组并进行初始化,分别表示国旗图片资源和国旗名称。
- 标号③处用于初始化标号①处所声明的属性变量;并对 backImageBtn 和 forwardImageBtn 对象设置监听器,来监听各自的单击事件。
- 标号④定义了一个 OnClickListener 类型的监听器,用于处理按钮单击事件;当单击按钮时根据所单击的按钮不同来显示不同的国旗和国别。

运行上述代码后,默认的界面效果如图 3-32 所示。当用户单击“>”按钮后,界面显示效果如图 3-33 所示。

注意 在实际开发中,如果要实现页面的切换功能,通常使用 ViewPager 类;该类是 Android Support Liberary 中自带的一个附加包中的一个类,用来实现屏幕间的切换。

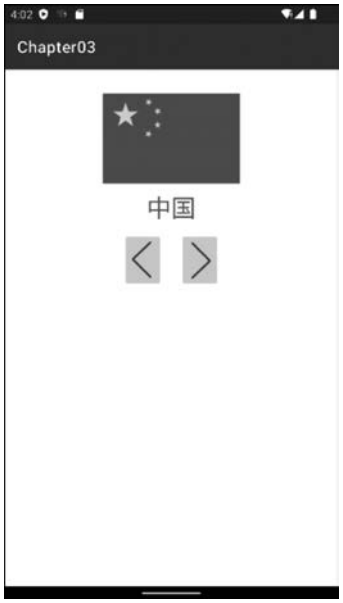


图 3-32　切换国旗一

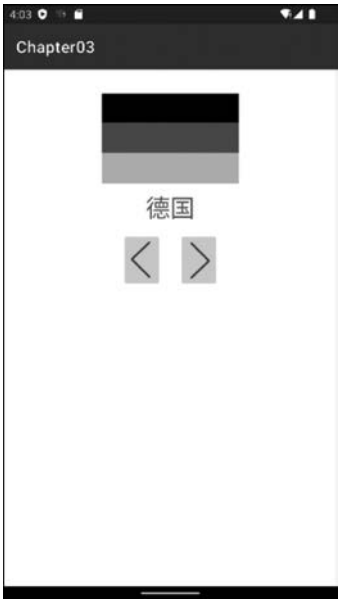


图 3-33　切换国旗二

3.5　Dialog 对话框

Dialog 对话框对于应用程序而言是一个必不可少的组件，在 Android 中，对话框对于一些重要的提示信息或者需要用户额外的交互是很有帮助的。所有的对话框都直接或间接继承自 Dialog 类，其中，AlertDialog 直接继承 Dialog 类，而其他的几个类则继承自 AlertDialog 类。

对话框就是一个小窗口，并不会填满整个屏幕，通常是以模态显示，需要用户采取行动才能进行后续操作。Android 提供了丰富的对话框支持，其中常用的对话框有以下 4 种。

- AlertDialog 提示对话框：是一种使用广泛且功能丰富的对话框。AlertDialog 不仅可以包含 0～3 个响应按钮，还可以包含一个单选框、复选框或列表。警告对话框通常用于创建交互式界面，是最常用的对话框类型。
- ProgressDialog 进度条对话框：只是对进度条进行了简单的封装，用于显示一个进度环或进度条，由于 ProgressDialog 是 AlertDialog 的扩展，所以也支持按钮选项。
- DatePickerDialog 日期对话框：用于用户选择日期的对话框。
- TimePickerDialog 时间对话框：用于用户选择时间的对话框。

3.5.1　AlertDialog 提示对话框

AlertDialog 继承自 Dialog 类，可以包含一个标题、一个内容消息或者一个选择列表以及 0～3 个按钮。在创建 AlertDialog 时推荐使用 AlertDialog 的 Builder 内部类来创建。首先使用 Builder 对象来设置 AlertDialog 的各种属性，然后通过 Builder. create()方法生成一个 AlertDialog 对象；如果只是显示一个 AlertDialog 对话框，可以直接使用 Builder. show()方法返回一个 AlertDialog 对象并显示。

当仅提示一段信息时，可以直接使用 AlertDialog 的属性设置提示信息，相关方法如表 3-23 所示。

表 3-23 AlertDialog 的相关方法

方 法	功 能 描 述
void create()	根据设置的属性，创建一个 AlertDialog
void show()	根据设置的属性，显示已创建的 AlertDialog
AlertDialog. Builder setTitle()	设置标题
AlertDialog. Builder setIcon()	设置标题的图标
AlertDialog. Builder setMessage()	设置标题的内容
AlertDialog. Builder setCancelable()	设置是否模态；一般设置为 false，表示采用模态形式，要求用户必须采取行动才能继续进行剩下的操作
AlertDialog setPositiveButton()	为对话框添加 Yes 按钮
AlertDialog setNegativeButton	为对话框添加 No 按钮

注意 AlertDialog. Builder 的大部分设置属性的方法返回是此 AlertDialog. Builder 对象，所以可以使用链式方式编写代码，这样更方便。

当一个对话框调用 show()方法将其展示到屏幕上时，如果需要消除该对话框，可以使用 DialogInterface 接口所提供的 cancel()方法来取消或者 dismiss()方法来消除对话框。cancel()和 dismiss()方法的作用是一样的，但推荐使用 dismiss()方法。Dialog 和 AlertDialog 都实现了 DialogInterface 接口，因此所有的对话框都可以使用这两个方法来消除对话框。对话框使用的场景较多，主要分为如下几种形式。

1. 普通对话框

下面通过一个简单的示例，演示使用 AlertDialog 如何提示信息，布局文件代码如下。

【案例 3-36】 dialog_demo. xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:orientation = "vertical" >
    <!-- 普通对话框 ① -->
    < Button android:id = "@ + id/normalBtn" android:text = "普通对话框" .../>
</LinearLayout >
```

上述代码中，标号①处定义了一个 Button 组件，当用户单击时，弹出一个普通对话框。

接下来在对应的 Activity 中实现按钮事件的业务逻辑：当用户单击“普通对话框”按钮时，在屏幕上显示对话框；当用户单击对话框中的“确认”按钮时，将退出应用程序；当用户单击“取消”按钮时，程序返回到主界面。代码实现如下。

【案例 3-37】 DialogDemoActivity. java

```
public class DialogDemoActivity extends AppCompatActivity {
    //普通对话框 ①
    Button normalBtn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.dialog_demo);
        //初始化组件 ②
        normalBtn = (Button) findViewById(R.id.normalBtn);
        //设置监听器对象
        normalBtn.setOnClickListener(onClickListener);
    }
    //定义单击事件监听器 ③
    private OnClickListener onClickListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            switch (v.getId()) {
            case R.id.normalBtn: {
                //普通对话框 ④
                AlertDialog.Builder builder = new AlertDialog.Builder(
                        DialogDemoActivity.this);
                builder.setMessage("确认退出吗?")
                        .setTitle("提示");
                //单击“确认”按钮后触发事件
                builder.setPositiveButton("确认",
                        new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog,
                                    int which) {
                                dialog.dismiss();
                                DialogDemoActivity.this.finish();
                            }
                        });
                //单击“取消”按钮后触发事件
                builder.setNegativeButton("取消",
                        new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog,
                                    int which) {
                                dialog.dismiss();
                            }
                        });
                builder.create().show();
                break;
            }
            default:
                break;
            }
        }
    };
}
```

代码解释如下。

- 上述代码中，标号①声明了 Button 类型的属性变量，当用户单击该按钮时，弹出普通对话框。
- 标号②处对标号①处所声明的属性变量进行初始化，通过对属性变量的赋值，使其可以进行后续的业务逻辑操作。

- 标号③处创建了一个监听器，用来监听用户单击按钮时所触发的事件。
- 标号④处实现了 OnClickListener 事件处理的业务逻辑；当事件触发时，弹出一个带有“确认”和“取消”的对话框；单击“确认”按钮退出当前应用，单击“取消”按钮返回程序的主界面。

运行上述代码后，在屏幕上单击“普通对话框”按钮时，打开提示对话框，如图 3-34 所示。

图 3-34 普通对话框

2. 内容型对话框

下面在上述实例的基础上，演示内容型对话框的使用。在 dialog_demo. xml 文件中添加一个“内容型对话框”按钮，当用户单击该按钮时，弹出一个含有三个按钮的对话框。以观众对电影的喜好为例，实现上述功能；首先，在布局文件中添加“内容型对话框”按钮，代码如下。

【案例 3-38】 dialog_demo. xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:orientation = "vertical" >
    <!-- 普通对话框 省略 -->
    <!-- 内容型对话框 -->
    < Button android:id = "@ + id/contentBtn" android:text = "内容型对话框" .../>
</LinearLayout >
```

然后，在 DialogDemoActivity. java 中按照“普通对话框”的编写步骤，添加“内容型对话框”对应的代码，代码如下。

【案例 3-39】 DialogDemoActivity. java

```
...
public class DialogDemoActivity extends AppCompatActivity {
    //普通对话框
    Button normalBtn;
    //内容型对话框
    Button contentBtn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.dialog_demo);
        //初始化组件
        normalBtn = (Button) findViewById(R.id.normalBtn);
        // 设置监听器对象
        normalBtn.setOnClickListener(onClickListener);
        //
        contentBtn = (Button)findViewById(R.id.contentBtn);
        contentBtn.setOnClickListener(onClickListener);
    }

    private OnClickListener onClickListener = new OnClickListener() {
```

```
        @Override
        public void onClick(View v) {
            switch (v.getId()) {
            case R.id.normalBtn: {
             …
            }
            case R.id.contentBtn:{
                //处理内容型的对话框
                AlertDialog.Builder builder
                         = new AlertDialog.Builder(DialogDemoActivity.this);
                builder.setIcon(android.R.drawable.btn_star)
                    .setTitle("喜欢度调查").setMessage("你喜欢成龙的电影吗?")
                    .setPositiveButton("很喜欢",
                        new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            Toast.makeText(DialogDemoActivity.this,
                                "我很喜欢他的电影.",Toast.LENGTH_LONG).show();
                        }
                    });
                //不喜欢
                builder.setNegativeButton("不喜欢",
                    new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            Toast.makeText(DialogDemoActivity.this,
                             "我不喜欢他的电影.", Toast.LENGTH_LONG).show();
                            }
                    });
                builder.setNeutralButton("一般",
                        new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            Toast.makeText(DialogDemoActivity.this,
                                "一般吧,谈不上喜欢.", Toast.LENGTH_LONG).show();
                         }});
                //显示对话框
                builder.show();
            }
            default:
                break;
            }
        }
    };
}
```

运行上述代码后，在屏幕上单击“内容型对话框”按钮，弹出一个内容型对话框，效果如图 3-35 所示。

注意 除了上述两种类型的对话框之外，开发人员还可以在对话框中实现一组单选框、多选框或列表项等多种形式，限于篇幅，此处不再赘述。

图 3-35 内容型对话框

3.5.2 ProgressDialog 进度对话框

在用户使用 App 的过程中,有些操作需要提示用户等待,比如在执行耗时较多的操作时,可以使用进度对话框显示一个进度信息来提示用户等待,此时可以使用 ProgressDialog 组件来实现。

ProgressDialog 有以下两种显示方式。

- 滚动的环状图标,是一个包含标题和提示内容的等待对话框。
- 带刻度的进度条,和常规进度条的用法一致。

上述两种方式的显示样式可以通过 ProgressDialog. setProgressStyle()方法进行设置,该方法的参数取值情况如下。

- STYLE_HORIZONTAL——刻度滚动。
- STYLE_SPINNER——图标滚动,默认选项。

其中,图标滚动可以使用两种方式来实现,一种是使用构造方法创建 ProgressDialog 对象,再设置对象的属性;另外一种是直接使用 ProgressDialog. show()静态方法返回一个 ProgressDialog 对象,然后调用 show()方法来显示。

下面通过一个简单示例演示 ProgressDialog 的使用。当用户单击"滚动等待对话框"按钮时,弹出滚动等待类型的对话框;当用户单击"进度条对话框"按钮时,弹出进度条类型的对话框,对应的布局文件代码如下。

【案例 3-40】 progress_demo. xml

```
<?xml version = "1.0" encoding = "utf - 8"?>
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android" ...
    android:orientation = "vertical" >
    <!-- 滚动等待对话框 -->
```

```
    <Button android:id="@+id/progressCircleBtn"
        android:text="滚动等待对话框" .../>
    <!-- 进度条对话框 -->
    <Button android:id="@+id/progressBarBtn"
        android:text="进度条对话框" .../>
</LinearLayout>
```

上述代码中定义了两个按钮，分别用于实现弹出“滚动等待对话框”和“进度条对话框”。

下面在相应的 Activity 中实现以下功能：当用户单击“滚动等待对话框”按钮时，屏幕上显示滚动等待对话框；当用户单击“进度条对话框”按钮时，屏幕上会显示进度条对话框。代码实现如下。

【案例 3-41】 ProgressDemoActivity.java

```
public class ProgressDemoActivity extends AppCompatActivity {
    //滚动等待对话框
    Button progressCircleBtn;
    //进度条对话框
    Button progressBarBtn;
    //存储进度条当前值,初始为 0
    int count = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.progress_demo);
        //初始化组件
        progressCircleBtn = (Button) findViewById(R.id.progressCircleBtn);
        //设置监听器对象
        progressCircleBtn.setOnClickListener(onClickListener);
        //
        progressBarBtn = (Button) findViewById(R.id.progressBarBtn);
        progressBarBtn.setOnClickListener(onClickListener);
    }
    private OnClickListener onClickListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            switch (v.getId()) {
            case R.id.progressCircleBtn: {
                //滚动等待对话框
                final ProgressDialog progressDialog = new ProgressDialog(
                        ProgressDemoActivity.this);
                progressDialog.setIcon(R.drawable.ic_launcher);
                progressDialog.setTitle("等待");
                progressDialog.setMessage("正在加载....");
                progressDialog.show();
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        try {
```

```
                        Thread.sleep(5000);
                    } catch (Exception e) {
                        e.printStackTrace();
                    } finally {
                        progressDialog.dismiss();
                    }
                }
            }).start();
        }
        case R.id.progressBarBtn: {
            //滚动等待对话框
            final ProgressDialog progressDialog = new ProgressDialog(
                    ProgressDemoActivity.this); //得到一个对象
            //设置为矩形进度条
            progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
            progressDialog.setTitle("提示");
            progressDialog.setMessage("数据加载中,请稍后...");
            progressDialog.setIcon(R.drawable.ic_launcher);
            //设置进度条是否为不明确
            progressDialog.setIndeterminate(false);
            progressDialog.setCancelable(true);
            //设置进度条的最大值
            progressDialog.setMax(200);
            //设置当前默认进度为 0
            progressDialog.setProgress(0);
            //设置第二条进度值为 1000
            progressDialog.setSecondaryProgress(1000);
            progressDialog.show(); //显示进度条
            new Thread() {
                public void run() {
                    while (count <= 200) {
                        progressDialog.setProgress(count++);
                        try {
                            Thread.sleep(100); //暂停 0.1 秒
                        } catch (Exception e) {
                        }
                    }
                }
            }.start();
        }
        default:
            break;
        }
    }
};
}
```

运行上述代码,在屏幕上单击“滚动等待对话框”按钮,效果如图 3-36 所示。当用户单击“进度条对话框”按钮后,效果如图 3-37 所示。

图 3-36　滚动等待对话框

图 3-37　进度条对话框

小结

（1）Android 应用的绝大部分 UI 组件都放在 android. widget 包及其子包中，Android 应用程序的所有 UI 组件都继承了 View 类。

（2）Android 中的界面元素主要由以下几个部分构成：视图、视图容器、Fragment、Activity 和布局管理器。

（3）Android 的所有 UI 组件都是建立在 View、ViewGroup 基础之上的，Android 采用了“组合器”模式来设计 View 和 ViewGroup，其中，ViewGroup 是 View 的子类。

（4）布局管理器可以根据运行平台来调整组件的大小，程序员的工作只是为容器选择合适的布局管理器即可。

（5）Android 提供了多种布局，常用的布局有以下几种：LinearLayout（线性布局）、RelativeLayout（相对布局）、TableLayout（表格布局）和 AbsoluteLayout（绝对布局）。

（6）Android 提供了两种方式的事件处理：基于回调的事件处理和基于监听的事件处理。

（7）Android 系统中引用 Java 的事件处理机制，包括事件、事件源和事件监听器三个事件模型。

（8）Android 的事件处理机制是一种委派式事件处理方式，该处理方式类似于人类社会的分工协作。这种委派式的处理方式将事件源和事件监听器分离，从而提供更好的程序模型，有利于提高程序的可维护性和代码的健壮性。

（9）对于基于回调的事件处理模型而言，事件源和事件监听器是统一的，当用户在 GUI 组件上触发某个事件时，组件自身的方法将会负责处理该事件。

（10）对 Widget 组件进行 UI 设计时既可以采用 XML 布局方式，也可以采用编码方式来实现，其中，XML 布局方式更加简单易用，被广泛使用。

习题

1. 下列可作为 EditText 编辑框的提示信息的是_____。

A. android:inputType　　B. android:text

C. android:digits　　D. android:hint

2. (多选)关于 widget 组件属性的写法,下列正确的是_____。

A. android:id="@+id/tv_username"　　B. android:layout_width="100px"

C. android:src="@drawable/icon"　　D. android:id="@id/tabhost"

3. 下列不是 Android SDK 中的 ViewGroup(视图容器)的是_____。

A. LinearLayout　　B. ListView

C. GridView　　D. Button

4. Android 提供了_____和_____两种事件处理方式。

5. Android 中的所有 UI 组件都是建立在_____基础之上。

6. 使用_____来设置 AlterDialog 的各种属性。

7. 简述 Android 中常用的几种布局方式。

8. 利用线性布局或相对布局实现一个加、减、乘、除及数字 1～9 的计算器完整布局,并编写相应的处理事件来完善整个计算功能。

9. 利用网格布局或相对布局实现一个注册界面,注册信息包括用户名、密码、确认密码和邮箱。