思想引领

视频讲解

第3章 列表与适配器

3.1 下拉列表

3.1.1 任务说明

本任务的演示效果如图 3-1 所示。在该应用中，活动页面视图根节点为垂直的 LinearLayout，在布局中依次放置 3 个 UI：1 个 TextView，用于显示个人信息；1 个 TextView（id 为 tv_result），显示 Spinner（下拉列表）的选择结果；1 个 Spinner，包含 3 个可供选择的数据“杭州”“宁波”和“温州”。单击 Spinner，会弹出下拉列表显示候选数据，选中某项数据后，Spinner 会收缩，tv_result 则显示选中的数据。

图 3-1 任务的演示效果

3.1.2 任务相关知识点

1. 适配器 ArrayAdapter

在 Android 中，很多 UI 需要展示批量化的数据。例如，新闻应用中每条新闻列表的格式一样，都有相同样式的标题、发布时间、图片等，对于应用层开发程序员，不会像 C 语言一

样对这些数据进行循环处理，生成一个个视图并渲染数据，而是将这些数据封装成一个类，存储于数组或列表中，并改写某个适配器将这些数据批量化地转换为视图。

在众多适配器中，最简单的是 ArrayAdapter，它接收字符串的列表(List)或者数组，并使用 Android 内置的样式进行数据到视图的转换，非常容易使用。适配器的角色就是将批量化的数据按预定义的格式，显示到 Spinner、ListView(列表视图)以及 GridView(网格视图)等组件上。ArrayAdapter 常用的构造方法有以下两种。

(1) public ArrayAdapter(Context context, @LayoutRes int resource, T[] objects)。

(2) public ArrayAdapter(Context context, @LayoutRes int resource, List < T > objects)。

其中，传参 context 是上下文，一般指使用该适配器的 Activity 对象，在 MainActivity 类里指 MainActivity. this(在非匿名方法中可直接用 this)。传参 resource 是样式控制参数，常用的样式有 android. R. layout. simple_list_item_1 和 android. R. layout. simple_list_item_single_choice(效果见图 3-2)，不同的样式具有不同的视图格式。传参 objects 是数据，类型为数组 T[]或者列表 List < T >，其中，T 是泛型，表示 objects 数据元素是 T 类型，T 可以是任何类型，若是非 String 类型，则会自动调用 toString()方法将其转换为 String 类型，若 T 类型没有实现 toString()方法，则显示该数据的内存引用信息。

图 3-2　适配器参数为 simple_list_item_single_choice 的样式效果

泛型是 Java 的一个特点，在没有泛型之前，对某个类使用 getter()方法，返回数据是个棘手问题。例如，使用 findViewById()方法，若定义的是 TextView 类型变量，则希望 findViewById()方法返回的也是 TextView 类型；若定义的是 Button 类型变量，则希望 findViewById()方法返回的是 Button 对象。若没有泛型，则不可能知道返回的数据是什么类型，因此早期 SDK 中，findViewById()方法返回的是所有 UI 的父类，即 View 对象，此时编程极为不便，需要对 findViewById()方法返回的数据进行强制类型转换成所需的数据类型。例如，假设定义了 TextView 和 Button 两个对象，在没有使用泛型时，则需要对返回结果进行强制类型转换，具体如下代码所示。

```
TextView tv = (TextView)findViewById(R. id. xxx);
Button bt = (Button)findViewById(R. id. yyy);
```

现行的 Android Studio 和 SDK 中，findViewById()方法采用了类似泛型的技术，返回变量用< T >描述，使用其匹配声明时的变量类型，从而避免了强制类型转换。

ArrayAdapter 适配器提供了获取指定位置的对象的方法：public T getItem(int position)。该方法返回的数据类型 T 与构造方法中的数据类型 T 一致，这样才能使构造方法中能传入任何类型的数据，使取数方法中得到对应类型的数据时无须强制类型转换。若没有泛型，则 getItem()方法只能返回 Object 类型，在返回数据时需强制类型转换成期望的数据类型。

Spinner 设置的适配器对象 adapter 还可以使用 setDropDownViewResource(int resource)

方法进一步设置弹出的下拉列表样式,在该方法中,resource 是文本样式参数,与构造方法的第 2 个参数类似。

数组 T[]和列表 List<T>在使用场合上有些区别,Java 中的数组动态特性较弱,只适合存储不需要增删操作的数据,反之,列表数据动态特性支持性好,适合存储动态数据。Java 中的数组没有提供对应的增删方法,若涉及增删操作非常烦琐。例如,在数组中间插入一个数据,需要考虑原定义数组长度够不够用,插入一个数据后,该位置之后的所有数据还需要进行移位操作。列表 List<T>本身是抽象类,常用 ArrayList 具体类。列表能弥补数组动态特性方面的不足,ArrayList 常用的操作有:add()方法,支持将数据插入末端,也可以将数据插入指定位置;addAll()方法支持批量数据添加;remove()方法支持删除数据;get()方法取指定位置的数据。ArrayList 不用用户定义列表长度,当长度不够后会自动申请增加长度,因此列表类型特别适合未知数据长度的动态数据以及需要对数据进行增删操作的使用场合。

适配器通过构造方法创建后,将批量的数据与指定的样式适配,构造方法所返回的适配器对象可以提供给 Spinner、ListView 和 GridView 等组件,组件对象通过 setAdapter()方法,将适配器作为方法的参数,使得视图组件拥有适配器,进而实现批量化数据的展示。

2. Spinner 的侦听回调

Spinner 常用于数据选择,具有展开和收缩两种视图形态。当用户不选择数据时,Spinner 处于收缩状态,显示的是被选中的数据;当用户单击 Spinner 时,Spinner 会展开视图,弹出下拉列表,展现适配器中的所有数据,供用户选择。因此,Spinner 设置完适配器后,往往需要设置对应的选中侦听事件,对用户选择行为进行响应处理。

Spinner 的选中侦听和回调如下述代码所示。

```
setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
@Override
public void onItemSelected(AdapterView<?> parent, View view,
                                                int position, long id) {
      //该回调方法在选项被选中时触发,Spinner 初始化后默认选中第 0 项,也会触发此回调
      //parent 是对应的组件对象,这里指 Spinner 对象
      //view 是被单击触发的视图
      //position 是被单击的位置索引,从 0 开始
      //id 是被单击数据的 id,在数据库 Cursor 中才会用到
    }
    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        //没有选项被选中时触发,一般用不到
    }
});
```

Spinner 的选中侦听回调在 Spinner 视图生成后才会被触发。在 MainActivity 中,onCreate()方法执行完毕时,视图并没有生成,Spinner 的选中侦听不会被触发。因此,在 MainActivity 的 onCreate()方法中,Spinner 选中侦听设置方法写在设置适配器之前还是之后均可以,读者不用担心代码书写顺序。

3.1.3　任务实现

1. 实现 UI 布局

新建布局文件 my_main.xml，如代码 3-1 所示。布局以垂直 LinearLayout 为根节点，在布局中依次拖入 1 个 TextView，用于显示个人信息；1 个 TextView，将 id 设置为 tv_result，用于显示 Spinner 的选择结果；1 个 Spinner，用于显示下拉列表数据和选中侦听事件处理。在 UI 面板中拖入的 Spinner，默认高度是 wrap_content，在组件树视图中有感叹号警示，其原因是高度值包围内容时是 24dp，认为在高密度屏幕中尺寸过小，不利于触控操作，并建议通过 android:minHeight 属性将最小高度设置为 48dp。此时，即使开发者接受 fix 按钮的修改建议，增加了对应属性，该 UI 依然有警示符号，提醒当前 Spinner 内容为空。这些警示只是 Android 的一些建议，不会引起程序崩溃，可忽略。

代码 3-1　布局文件 my_main.xml

```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:orientation = "vertical"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent">
    <TextView
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:text = "Your name and ID" />
    <TextView
        android:id = "@ + id/tv_result"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:text = "" />
    <Spinner
        android:id = "@ + id/spinner"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content" />
</LinearLayout>
```

2. 实现 MainActivity

MainActivity 的实现如代码 3-2 所示。在 onCreate()方法中初始化了 1 个 String 数组，作为适配器构造方法的数据。适配器声明时采用了泛型 ArrayAdapter<String>，表示该适配器只接收 String 类型的数组或列表数据，当调用适配器的 getItem()方法获取指定位置数据时，其返回的数据就是泛型指定的类型(本任务中是 String 类型)。

在代码 3-2 中，故意将 Spinner 的选中侦听设置方法和适配器设置方法调换顺序(习惯上先给 Spinner 对象设置适配器，再写相应的选中侦听)，以测试程序的运行情况。如前述分析，Spinner 的选中侦听和响应需要视图渲染之后才能生效，而 MainActivity 的 onCreate()方法并没有真正完成渲染，因此在 onCreate()方法执行结束之时，Spinner 的选中侦听方法不会被回调，因此 Spinner 的 setOnItemSelectedListener()方法与 setAdapter()方法先后顺序不敏感。

代码 3-2　MainActivity.java

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

```
3    import android.view.View;
4    import android.widget.AdapterView;
5    import android.widget.ArrayAdapter;
6    import android.widget.Spinner;
7    import android.widget.TextView;
8    public class MainActivity extends AppCompatActivity {
9        @Override
10       protected void onCreate(Bundle savedInstanceState) {
11          super.onCreate(savedInstanceState);
12          setContentView(R.layout.my_main);
13          TextView tv = findViewById(R.id.tv_result);
14          Spinner sp = findViewById(R.id.spinner);
15          String[] cities = new String[]{"杭州","宁波","温州"};
16          ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
17                                        android.R.layout.simple_list_item_1,cities);
18          sp.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
19            //设置选中侦听
20            @Override
21            public void onItemSelected(AdapterView<?> parent, View view,
22                                                        int position, long id){
23                string city = cities[position];
24                tv.setText(city);
25            }
26            @Override
27            public void onNothingSelected(AdapterView<?> parent) {
28            }
29         });
30         sp.setAdapter(adapter);          //设置适配器
31     }
32   }
```

3. 程序调试

程序调试是编程生涯中最重要的技巧之一,也是很多初学者最欠缺的技能之一。为了进一步提高读者的调试能力,在本任务中增加一个环节:如何验证行 Spinner 的 onItemSelected()回调方法中传参 parent 和 Spinner 对象 sp 是同一个对象?

最好的验证方式是将变量 sp 加入变量窗口,并在代码 3-2 的第 24 行设置断点,程序运行至第 24 行断点处,在变量窗口中查看变量 sp 和传参 parent 是否是同一个内存引用,若是,则这两个变量是同一个对象。此外,还可以弹出表达式窗口,直接在线运行对象的某些方法用于验证对象。

验证变量 sp 和传参 parent 的调试操作如下。

步骤 1:在代码第 14 行选中 sp 对象后右击,在弹出的快捷菜单中选择 Add to Watches 选项,即可将该变量加入变量窗口;

步骤 2:在代码第 24 行设置断点;

步骤 3:单击 Debug 图标进入调试模式。

若程序中还有其他断点,则应用在调试模式下运行至第一个断点处暂停,假如此时不在代码的第 24 行处,若此时采用单步调试(Step Over,F8 快捷键)运行 onCreate()方法所有代码后,则程序将处于"失联"状态。其原因是,onCreate()方法执行完毕,系统会继续执行 MainActivity 的其他方法,但这些方法不在当前代码区,若要单步运行至代码第 24 行是非常漫长且不现实的。此时,需要利用全速继续运行功能(Resume Program,F9 快捷键),使

之全速运行，直至遇到断点暂停。在全速继续运行过程中，当 Spinner 触发选中事件时，会在代码第 24 行处暂停，此时观察到的变量窗口如图 3-3 所示。遗憾的是，在变量窗口中，变量 sp 无法被观测，其原因是该变量是 onCreate()方法中的局部变量，而 onItemSelected()回调方法虽然写在 onCreate()中，但它是匿名方法，本质上在 onCreate()方法之外，在选中事件触发回调时，onCreate()方法早已运行完毕，此时，onCreate()方法中的局部变量已消亡。

图 3-3　变量窗口的各个变量

对于本任务而言，即使对变量 sp 添加 final 关键字修饰，也无法在 onItemSelected()方法中通过调试窗口观察到该变量。另一种做法是，将变量 sp 修改成 MainActivity 的成员变量(全局变量)，使得变量 sp 能被 MainActivity 类中的所有方法访问，具体操作如代码 3-3 所示。注意，变量 sp 被声明成类成员变量后，在 onCreate()方法中初始化时，不能写成 Spinner sp＝findViewById(R.id.spinner)，若是这样写，则 onCreate()方法中的变量 sp 是局部变量，与类成员变量 sp 无关，此时成员变量 sp 依然是 null 对象。

代码 3-3　修改 Spinner 对象为类成员变量

```
1    public class MainActivity extends AppCompatActivity {
2        Spinner sp;
3        @Override
4        protected void onCreate(Bundle savedInstanceState){
5            …
6            sp = findViewById(R.id.spinner);
7            //不要写成 Spinner sp = findViewById(R.id.spinner);
8            //若这样写,onCreate()方法中的 sp 是局部变量,成员变量 sp 依然是 null 对象
9        …
10       }
11   }
```

按代码 3-3 修改代码后，在同样的地方设置断点，重新调试程序至断点处，变量窗口如图 3-4 所示，Spinner 对象 sp 和 onItemSelected()方法的传参 parent 均指向同一个引用(@9647)，因此两者是同一个对象。当适配器采用 R.layout.simple_list_item_1 样式参数时，还可看到传参 view 和变量 tv 均为 MaterialTextView，说明采用 ArrayAdapter 和

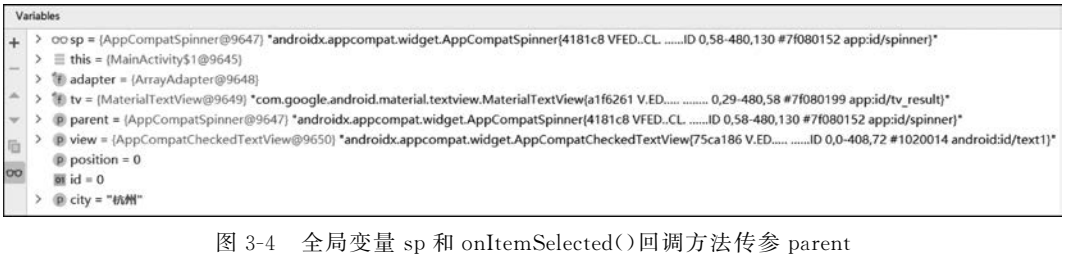

图 3-4　全局变量 sp 和 onItemSelected()回调方法传参 parent

R.layout.simple_list_item_1 内置样式时，得到的单元视图是一个文本对象，此时也可以从传参 view 调用相关方法获得对应文本。此外，还可以通过单击 Evaluate Expression 图标(或者 Alt＋F8 快捷键)，得到表达式运算对话框，如图 3-5 所示，在对话框中可调用对象的相关方法以获得更多的信息。

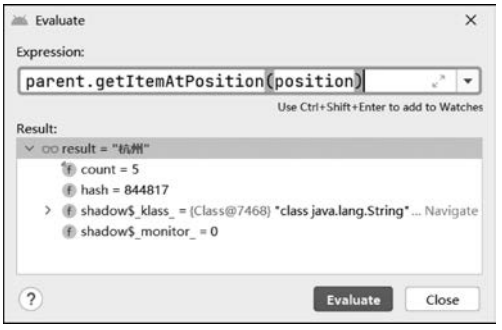

图 3-5　表达式运算对话框

4. 接口回调中的数据获取方法

Spinner 对象在 onItemSelected()回调方法中会回传 AdapterView 对象 parent、当前视图对象 view、触发的位置 position。此时若要在回调方法中取位置 position 对应的数据，可用以下方法实现。

(1) 利用适配器的数据源对象(数组 T[]或者列表 List＜T＞)，取得对应位置的数据。例如，代码 3-2 中，通过数组 cities[position]获得选中的数据。在编程实践中，可先写 cities[position]，并利用 Ctrl＋Alt＋V 快捷键生成局部变量，开发环境非常智能，知道 cities 是复数，获得的数据是单数，并且知道该数据的类型，会自动填充代码变成：String city ＝ cities[position]，这一功能能大大提高编程效率。

(2) 利用适配器对象所提供的 public T getItem(int position)方法。适配器构造时采用了泛型 ArrayAdapter＜T＞声明，因此能保证 getItem()方法返回的数据与声明时的数据源类型一致，调用该方法可使用 Ctrl＋Alt＋V 快捷键生成局部变量。

(3) 利用使用了适配器的组件对象(本任务中是 Spinner 对象 sp)所提供的 public Object getItemAtPosition(int position)方法。遗憾的是，getItemAtPosition()方法返回的是 Object 对象，需要声明对象类型并将返回数据进行强制类型转换。

(4) 利用回调方法中的传参 view。若传参 view 是容器(例如，复杂视图是一个 LinearLayout，包含了若干个 UI)，还需要进一步对传参 view 分离 UI，利用 view.findViewById()方法，找出相关 id 的 UI，再调用 UI 对象获得相关数据；若传参 view 是单个 UI，则可直接调用 UI 相关方法取数。在本任务中，传参 view 是 TextView 对象，可直接将传参 view 强制类型转换成 TextView 对象，再利用 TextView 对象的方法获得数据。

以上几种方法对应的实现如代码 3-4 所示，综合而言，笔者不太建议使用方法 3 和方法 4，尤其是方法 4，对于复杂视图的操作会非常烦琐。方法 1 和方法 2 均能通过 Ctrl＋Alt＋V 快捷键对返回结果生成局部变量，比较符合笔者的编程习惯。

代码 3-4　Spinner 对象在回调方法中获取触发位置对应数据的若干方法

```
1    public void onItemSelected(AdapterView<?> parent, View view,
2                               int position, long id) {
3        //String city = cities[position];              //可用 Ctrl + Alt + V 快捷键
4        String city = adapter.getItem(position);       //可用 Ctrl + Alt + V 快捷键
5        //String city = (String) parent.getItemAtPosition(position);
6        //String city = (String) sp.getItemAtPosition(position);
7        //TextView temp = (TextView) view;
8        //View 对象本身不提供 getText()方法，需要强制类型转换
9        //String city = (String) temp.getText();
```

```
10          tv.setText(city);
11      }
```

3.2　使用 Spinner 控制文本颜色

3.2.1　任务说明

本任务的演示效果如图 3-6 所示。在该应用中，活动页面视图根节点为垂直的 LinearLayout，在布局中依次有 1 个 TextView(id 为 tv)，用于显示个人信息；1 个 Spinner，下拉列表中有 XML 资源文件所自定义的 3 个颜色数据“红色”“绿色”和“蓝色”，单击选项，则 tv 文本框的文本颜色被更改为对应的颜色。

图 3-6　Spinner 控制文本颜色演示效果

本任务要求颜色的值和颜色的名称均定义在 XML 资源文件中，方便后期替换更改。完成项目后，在不更改任何代码的情况下，将资源文件里的颜色名称和颜色值修改为 4 个数据，运行应用观察是否正常。资源文件在项目的 res/values 文件夹下，可以是自定义 XML 文件，也可以在项目所生成的默认文件上增加数据。

3.2.2　任务实现

1. 项目的资源文件

Android 编程中，不建议将值和界面文字直接定义在代码中，而是将它们定义在 XML 资源文件中，以方便后期的管理。例如，在程序修改后需要将界面中所有的中文字符改成英文字符。若将相关的文字直接写在代码里，必然使程序难以维护，需要程序员在代码中逐一替换，而利用 XML 资源文件统一管理，只需要一个专业的翻译将对应文字进行翻译即可。考虑到本书的初衷只是为了帮助初学者更好地理解代码和编程方法，因此在很多例子中并没有采用这个规范，而是直接将相关文字和值定义在代码中，减少代码篇幅。

在本任务中将使用 XML 资源文件管理文字、颜色常量值，以便读者对 Android 项目的资源文件有初步的认识。若仅仅是为了完成图 3-6 所示的演示效果，显然将这些数据直接定义到代码中会更紧凑，读者不妨自行实现。

XML 资源文件存放于项目的 res/values 文件夹中，可用 color 标签定义颜色，string 标签定义字符串，dimen 标签定义尺寸，drawable 标签定义绘图资源及图片(引用图片资源)，integer 标签定义整型数据。这些标签的 name 属性给出资源的名称，以方便资源之间的引用，在 XML 中的引用格式为@xxx/yyy，其中，xxx 为数据类型，yyy 为 name 值；在代码中的引用格式为 R.xxx.yyy，同理，xxx 为数据类型，yyy 为 name 值。标签数据可直接赋值，也可以引用资源中其他数据的值。

除了单个数据的定义，XML 中还可以定义数组，使用 string-array 定义字符串数组，使用 integer-array 定义整型数组，使用 array 定义通用数组。数组除了 name 属性，还有 item

标签定义子节点,用于表示数组中1个元素的值,该值可直接赋值也可以使用引用值。数组在代码中的引用方式是R.array.yyy,其中,yyy是数组的name属性值。在程序中获取XML资源还要利用Activity对象提供的getResources()方法获得Resources对象。使用Resources对象在XML资源中取数时,若是string-array数据,则可用getStringArray()方法;若是integer-array,则用getIntArray()方法;若是通用型array,则须通过obtainTypedArray()方法获得通用型数组,再通过TypedArray对象的getColor()、getDimension()、getDrawable()等方法分别进行颜色、尺寸或绘图对象的二次解析。

本任务对res/values/colors.xml资源文件进行修改,如代码3-5所示,对颜色值数组my_color_values增加了红、绿和蓝3个颜色值,采用的是通用型array数组。考虑到颜色数组和对应颜色名称数组放在同一个文件更便于管理,因此,将颜色名称数组my_color_names也放在colors.xml资源文件中。

代码3-5　对res/values/colors.xml资源文件增加颜色值数组和颜色名称数组

```
<array name = "my_color_values">
      <item>#f00</item>
      <item>#0f0</item>
      <item>#00f</item>
</array>
<string-array name = "my_color_names">
      <item>红色</item>
      <item>绿色</item>
      <item>蓝色</item>
</string-array>
```

2. 实现UI布局

MainActivity的布局文件my_main.xml如代码3-6所示,布局中TextView需要赋予id值,以便于在代码中对其进行属性控制。

代码3-6　布局文件my_main.xml

```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
      android:orientation = "vertical"
      android:layout_width = "match_parent"
      android:layout_height = "match_parent">
      <TextView
            android:id = "@ + id/tv"
            android:layout_width = "match_parent"
            android:layout_height = "wrap_content"
            android:text = "Your name and ID" />
      <Spinner
           android:id = "@ + id/spinner"
           android:layout_width = "match_parent"
           android:layout_height = "wrap_content" />
</LinearLayout>
```

3. 实现MainActivity

MainActivity的实现如代码3-7所示。在程序中,需要注意字符串数组和通用性数组在处理方式上的区别。颜色名称数组colorNames是string-array类型,可直接通过Resources对象的getStringArray()方法获得;颜色值数组colorValues是通用型array数组,需通过obtainTypedArray()方法得到TypedArray数组,再通过调用getColor()方法进

行颜色值的二次解析。

代码 3-7　MainActivity.java

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.res.Resources;
import android.content.res.TypedArray;
import android.graphics.Color;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_main);
        TextView tv = findViewById(R.id.tv);
        Spinner sp = findViewById(R.id.spinner);
        Resources res = getResources();  //XML 中定义的资源数据需要 Resources 对象获取
        String[] colorNames = res.getStringArray(R.array.my_color_names);
        //String[]类型的数组可直接调用 getStringArray()方法获得
        TypedArray colorValues = res.obtainTypedArray(R.array.my_color_values);
        //资源没有提供颜色数组,只能用通过 TypedArray 获得数组对象
        ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
                                    android.R.layout.simple_list_item_1,colorNames);
        sp.setAdapter(adapter);
        sp.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
           @Override
           public void onItemSelected(AdapterView<?> parent, View view,
                                                        int position, long id){
               int color = colorValues.getColor(position, Color.BLACK);
               //第 2 个参数是获取失败时替换的默认颜色
               //第 2 个参数使用了 Color 类里提供的颜色常量,Color.BLACK 为黑色
               tv.setTextColor(color);
           }
           @Override
           public void onNothingSelected(AdapterView<?> parent) {
           }
       });
    }
}
```

3.3　使用 Spinner 控制文本大小

3.3.1　任务说明

本任务的演示效果如图 3-7 所示。在该应用中,活动页面布局根节点为垂直的 LinearLayout,在布局中依次有 1 个 TextView(id 为 tv),用于显示个人信息;1 个 Spinner,Spinner 中有下拉选项“小号”、“中号”和“大号”,对应字体大小的值分别为 10sp、16sp 和 24sp。Spinner 选择字号选项后,则 tv 的字体大小更改为对应字号。字体大小相关数据采用 XML 资源文件管理,并且需要利用 Logcat 打印字号对应的真实值。

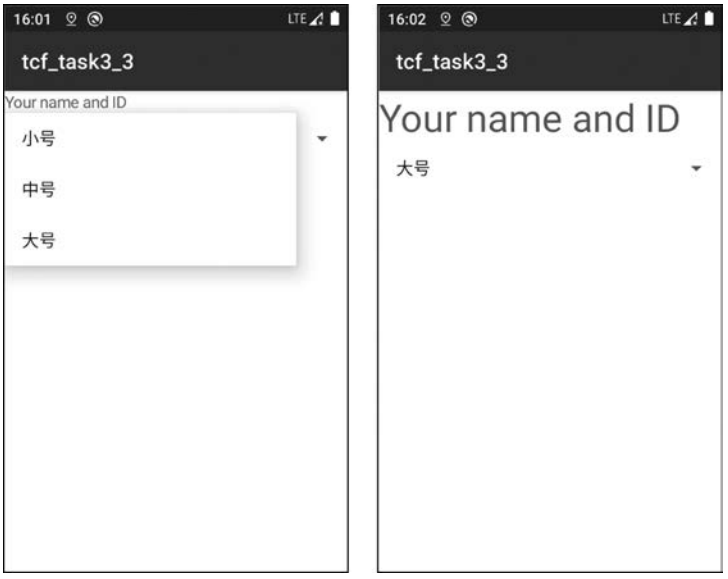

图 3-7　Spinner 控制文本大小演示效果

利用 Logcat 窗口查看打印输出也是 Android 编程中常用的技巧,其对应的调用类是 android. util. Log。Log 类提供了打印输出的多种静态方法,常用的输出方法如下所示。

(1) public static int d(String tag, String msg),用于打印 Debug 日志信息。

(2) public static int e(String tag, String msg),用于打印 Error 错误信息,以红色文字显示。

(3) public static int i(String tag, String msg),用于打印 Info 普通信息。

(4) public static int w(String tag, String msg),用于打印 Warn 警告信息,以蓝色文字显示。

在以上各个输出方法中,第 1 个参数 tag 是标签,可以是任意文字,用于过滤信息或者通过标签告知用户,打印的是哪个信息; 第 2 个参数 msg 是所需要打印的消息值,若是非 String 类型的数据,可用字符串双引号+变量的方式,将后面的变量转换为文本输出。日志打印可帮助开发者了解过程变量,在调试中经常使用。

3.3.2　任务实现

1. 创建资源文件与数据

本任务中所涉及的尺寸值采用 XML 资源文件进行管理,可在 res/values 文件夹下创建 dimens. xml 文件,文件名只要符合 XML 命名规则即可(即名称可以包含小写英文字符、阿拉伯数字和短下画线,以小写字母开头)。创建资源文件可右击 values 文件夹,在弹出的快捷菜单中选择 New→Values Resource File 选项,利用向导创建,如图 3-8 所示。

资源文件 dimens. xml 如代码 3-8 所示,字号数组 my_dimen_values 采用通用型 array,字号名称数组 my_dimen_names 采用 string-array。

代码 3-8　资源文件 res/values/dimens. xml

```
1    <resources>
2        <array name="my_dimen_values">
```

```
        <item>10sp</item>
        <item>16sp</item>
        <item>24sp</item>
    </array>
    <string-array name="my_dimen_names">
        <item>小号</item>
        <item>中号</item>
        <item>大号</item>
    </string-array>
</resources>
```

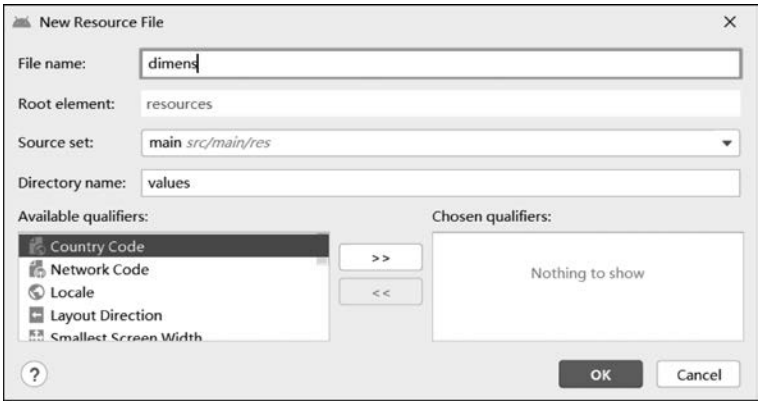

图 3-8　利用向导创建资源文件

2. 实现 MainActivity

MainActivity 的布局文件与 3.2 节中的项目相同，可直接将 3.2 节项目中的 my_main.xml 文件复制至本项目。MainActivity 的实现如代码 3-9 所示。对比 3.2 节中的 MainActivity 程序，本项目的适配器采用了不同的生成方法。适配器常用构造方法创建，但若数据资源定义在 XML 文件中，还有更简便的方法，即直接调用 ArrayAdapter.createFromResource()静态方法从 XML 资源中取数并创建适配器。注意，createFromResource()方法的参数顺序与构造方法略有不同，其第 2 个参数是数据资源(构造方法中第 2 个参数是样式)，可直接采用 R.array.yyy 的方式直接引用 XML 资源中的数组数据，第 3 个参数是样式(构造方法中第 3 个参数是数据)。此外，该静态方法返回的适配器泛型参数是 CharSequence(String 类继承自 CharSequence)，若直接用 ArrayAdapter<String>定义适配器，开发环境会提示类型不匹配，可通过编译器自动修复(Alt+Enter 快捷键)，即可自动改成 ArrayAdapter<CharSequence>。

CharSequence 是一个接口，它只包括 length()、charAt(int index)、subSequence(int start,int end)等几个常用的方法，String、StringBuilder 和 StringBuffer 等类均来自于 CharSequence(如图 3-9 所示)。若是对字符串少量的拼接操作可用 String，若是大量的拼接操作，则建议用 StringBuilder(非线程安全)或 StringBuffer(线程安全)，其原因是采用 String 类拼接字符串，对待拼接的 String 对象，会生成对应的 String 对象加入常量池再对其引用，拼接操作越多，所产生的内存小碎片越多，不利于系统优化。

代码 3-9　MainActivity.java

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.res.Resources;
```

```
import android.content.res.TypedArray;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_main);
        Spinner sp = findViewById(R.id.spinner);
        TextView tv = findViewById(R.id.tv);
        ArrayAdapter < CharSequence > adapter = ArrayAdapter.createFromResource(this,
              R.array.my_dimen_names,android.R.layout.simple_list_item_1);
        //直接从资源文件取数得到适配器,注意第 2 参数和第 3 参数的参数顺序
        //第 2 参数是资源数据引用,第 3 参数是样式,并且定义所用的泛型是 CharSequence
        Resources res = getResources();
        TypedArray dimenValues = res.obtainTypedArray(R.array.my_dimen_values);
        sp.setAdapter(adapter);
        sp.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView <?> parent, View view,
                                                    int position, long id){
                float dimension = dimenValues.getDimension(position, 10.0f);
                //尺寸数据是浮点数,第 2 个参数是默认值(取数解析失败时取默认值返回)
                Log.d("Dimension","" + dimension);
                //Log.d 可打印 Debug 日志,在 Logcat 窗口中可查看
                Log.e("Dimension","" + dimension);      //打印 Error 信息
                Log.i("Dimension","" + dimension);      //打印 Info 信息
                Log.w("Dimension","" + dimension);      //打印 Warn 信息
                tv.setTextSize(dimension);
            }
            @Override
            public void onNothingSelected(AdapterView <?> parent) {
            }
        });
    }
}
```

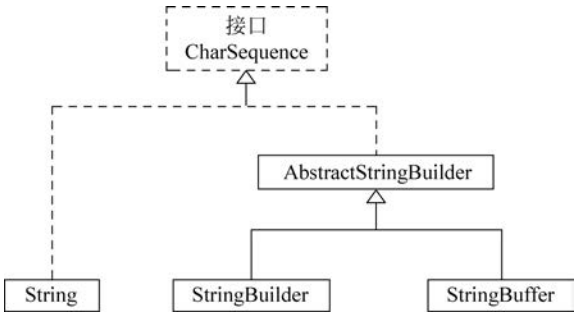

图 3-9　String 和 CharSequence 的关系

3. 使用 Logcat 日志

在代码 3-9 中，onItemSelected()回调方法利用 Log 类打印输出四种类型的日志，用于验证字体尺寸单位 sp 是否与浮点数值等值。Log 类打印日志的各个方法中，第 1 个参数可以是任何字符串，便于信息查看的标识或过滤；第 2 个参数是打印的内容信息，若是传入浮点数，需要加空字符串拼接，将浮点数转换为字符串。Logcat 窗口打印输出如图 3-10 所示，其输出的信息很多，可结合若干技巧加以运用：①如果不关心历史信息，可单击日志窗口左上方的删除按钮，将之前的日志信息清除；②在日志类型窗口中可选择对应的日志类型，Verbose 是所有类型，Debug 只显示 Log. d()方法的输出，Info 显示 Log. i()方法的输出，Warn 显示 Log. w()方法的输出，Error 显示 Log. e()方法的输出；Assert 显示断言信息；③可在过滤检索栏输入关键词，只显示与日志第 1 个参数(标签值)相匹配的日志信息。

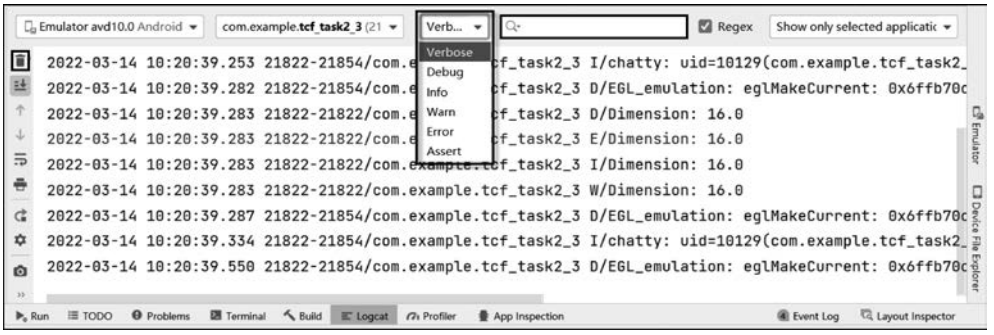

图 3-10　Logcat 日志窗口

过滤日志如图 3-11 所示，在过滤栏输入 dimen，则日志窗口显示符合过滤条件的日志信息，能帮助开发者更快得到相关信息。

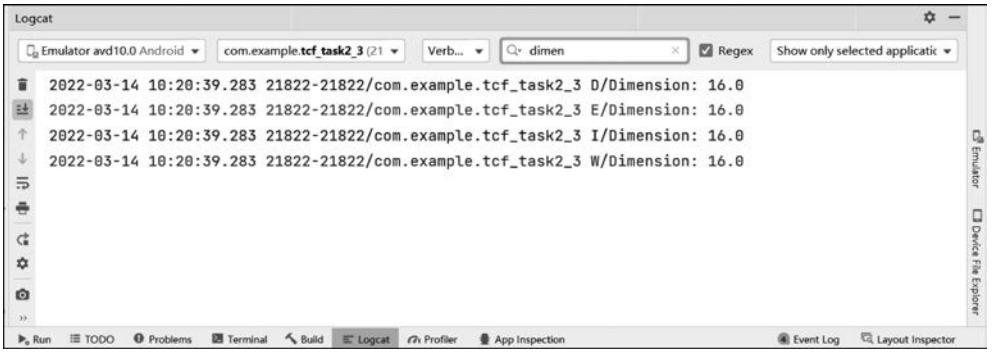

图 3-11　Logcat 窗口通过过滤功能获得日志信息

3.4　使用 ListView 切换 ImageView 图片

3.4.1　任务说明

本任务的演示效果如图 3-12 所示。在该应用中，活动页面视图根节点为垂直的 LinearLayout，在布局中依次有 1 个 TextView，用于显示个人信息；1 个 ImageView(图像视图)，高和宽分别为 200dp；1 个 ListView(列表视图)，显示的列表项为“杭州”“宁波”“温

州”。单击 ListView 的列表项,ImageView 的图片会切换为对应城市的图片。

图 3-12　使用 ListView 切换 ImageView 的图片资源

ListView 是显示相同格式内容的一种组件,常用于新闻、商品等数据展示,其使用方法与 Spinner 类似,也是通过适配器绑定数据。区别在于,ListView 使用 OnItemClickListener 接口侦听列表项的单击事件,而 Spinner 则使用 OnItemSelectedListener 接口侦听下拉列表选项的选中事件。

3.4.2　任务实现

1. 项目中的图片

在 Android 项目中,可放置 jpg、gif 和 png 等图片资源,但是其文件命名必须符合规范:不能有中文、空格,以及大写英文字符等不符合资源命名方式的字符,合法的是小写字母、短下画线(_)和阿拉伯数字,且首字符必须是小写字母。因此,若将网络图片复制到项目资源中,则需将其重命名为符合规范的文件名才能被使用。

复制图片时,可单击 res/drawable 文件夹,直接按 Ctrl+V 快捷键粘贴图片,即可将图片文件复制到 drawable 文件夹,并得到图 3-13 所示的文件夹选择提示,有 drawable 和 drawable-v24 两个类型。如果开发者的虚拟机是 Android 5.0(API 21),不能选择 drawable-v24 文件夹,否则会出现因找不到图片资源而导致的程序崩溃。Android 设备不同屏幕有不同的屏幕密度,为了更好地匹配效果,会有 mipmap-mdpi、mipmap-hdpi、mipmap-xhdpi 等不同的文件夹匹配不同尺寸的图片,使设备能根据屏幕密度选择合适文件夹下的图片资源以获得更好的显示效果。同理,drawable-v24 文件夹用于适配 Android 7.0 以上设备(API 24),若虚拟机是 Android

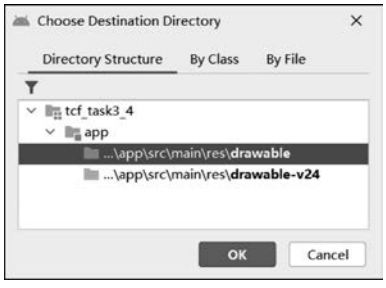

图 3-13　粘贴图片时选择 drawable 图片文件夹

5.0，则访问不了该文件夹下的图片资源，反之，Android 7.0 以上设备可以访问 drawable 和 drawable-v24 文件夹。复制图片时，虽然开发环境默认选择 drawable-v24 文件夹，开发者需要手动切换到 drawable 文件夹，使低于 Android 7.0 的设备也能访问图片资源。

若不小心将图片复制到 drawable-v24 文件夹，如图 3-14 所示，对于 v24 资源，会在资源名称后面加“(v24)”，其修正方法可以先删除相关资源，再重新复制。这里介绍一种兼容性强的处理方法，首先将项目视图从 Android 切换到 Project 或 Packages。在 Android 视图下，资源比较紧凑，但不是物理文件夹视图，Project 视图则更接近物理文件夹视图，以方便不同文件夹之间的资源复制，具体操作如图 3-15 所示。在 Project 视图模式下，将 drawable-v24 文件夹中的相关资源复制到 drawable 文件夹，切换到 Android 视图模式，各个图片资源以文件夹形式出现，并且具有两个不同版本的文件。在实践中，可根据需要，将不同分辨率的图片以相同的名称分别复制到 drawable 和 drawable-v24 文件夹，让系统根据 Android 版本进行匹配，Android 7.0 以下设备使用低分辨率图片，而 Android 7.0 以上设备则使用高分辨率图片。

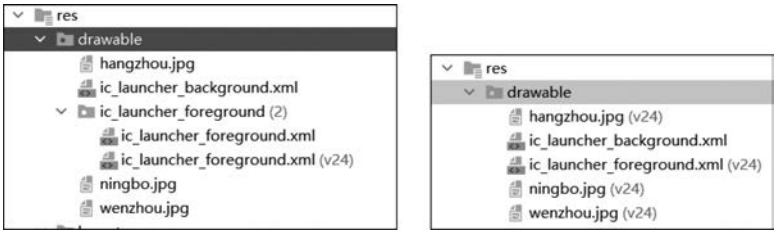

图 3-14　图片在不同 drawable 文件夹的区别

对于 drawable 文件夹中的图片，XML 中的引用方式为@drawable/xxx，其中，xxx 为图片的文件名称。例如，@drawable/hangzhou 引用的是 hangzhou.jpg，资源引用不用加文件的扩展名，系统会自动识别。在 Java 代码中的引用方式为 R.drawable.xxx，其中，xxx 为图片文件名称，同样不能有扩展名。例如，R.drawable.hangzhou 引用的同样是 hangzhou.jpg。Java 中图片资源的引用本质上是 int 型数据，多张图片可以用 int[]数组管理，也可以在 XML 中定义通用型数组 array，将图片以引用的方式作为 array 标签中一个 item 的值，再在 Java 中获取 array 数组加以利用。

图 3-15　将项目视图从 Android 切换至 Project

2. 实现 UI 布局

MainActivity 的布局文件 my_main.xml 如代码 3-10 所示，建议在设置布局文件之前，先将图片资源复制到项目中，以方便拖入 ImageView 组件时可通过向导指定图片资源。ImageView 图片控制属性是 app:srcCompat="@drawable/xxx"，其中，xxx 为图片资源的文件名。

ImageView 是图像视图，用于显示图片资源。本任务中，ImageView 的宽度和高度设置为 200dp，其目的是指定图片的宽高，当不同图片资源原始尺寸不一样时，显示在 ImageView 中具有相同的尺寸。若 ImageView 的尺寸属性值是 wrap_content，则 UI 会随

图 3-16　将 drawable-v24 资源复制到 drawable 文件夹

着图片原始尺寸改变大小,从而导致切换图片时产生 UI 位置的跳跃感,影响应用体验。ImageView 通过 android:layout_gravity="center"设置 UI 在 LinearLayout 中居中显示,此时 center 属性值本质上是垂直和水平居中,由于 LinearLayout 是垂直方向的,子视图的 layout_gravity 属性只和父容器的正交方向起作用,因而 ImageView 在容器中实际上是水平居中。ImageView 可用 android:scaleType 控制图片宽高的比例属性,若值是 centerCrop,则从图片中心点裁剪缩放,使其宽高符合 ImageView 尺寸;若没有使用 android:scaleType 属性,则保持原图片资源的宽高比例进行缩放,使得最长边符合 ImageView 所定义的尺寸。ListView 默认没有创建 id,可通过向导创建对应 id,该组件默认宽度和高度均为 match_parent,可根据需要将高度设置为 wrap_content。当 ListView 中的内容高度超过容器时,可上下滑动列表视图查看剩余内容。

代码 3-10　布局文件 my_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Your name and ID" />
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_gravity="center"
        app:srcCompat="@drawable/hangzhou" />
<!--    建议先复制图片资源到项目,再拖入 ImageView,通过向导设置资源文件 -->
    <ListView
        android:id="@+id/listview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

3. 实现 MainActivity

本任务的 MainActivity 如代码 3-11 所示。图片资源使用 R.drawable.xxx 方式引用，采用 int[]数组管理，并且与数组 cities 一一对应。ListView 的使用方式与 Spinner 类似，需要定义一个适配器并为组件设置适配器。两者的区别是侦听事件不同，ListView 通过 OnItemClickListener 接口捕捉用户单击列表项行为，并在 onItemClick(AdapterView <?> parent, View view, int position, long id)回调方法中进行处理。onItemClick()方法的参数中，parent 是组件对象(ListView 对象)；view 是所单击的行视图；position 是单击列表项的位置，从 0 开始索引；id 是对应数据库游标(Cursor)的 id 值，若没有用到数据库则可忽略。在 onItemClick()回调方法中，通过传参 position 可获得数组 images 对应位置的图片资源，并通过 ImageView 对象(iv)的 setImageResource()方法更换图片。ImageView 提供了非常多的更换图片方法，如 setImageBitmap(Bitmap bm)、setImageDrawable(Drawable drawable)、setImageURI(Uri uri)、setImageResource(int resId)等，可根据对象类别选择对应的方法对 ImageView 更换图片。

代码 3-11　MainActivity.java

```
1    import androidx.appcompat.app.AppCompatActivity;
2    import android.os.Bundle;
3    import android.view.View;
4    import android.widget.AdapterView;
5    import android.widget.ArrayAdapter;
6    import android.widget.ImageView;
7    import android.widget.ListView;
8    public class MainActivity extends AppCompatActivity {
9        @Override
10       protected void onCreate(Bundle savedInstanceState) {
11           super.onCreate(savedInstanceState);
12           setContentView(R.layout.my_main);
13           ImageView iv = findViewById(R.id.imageView);
14           ListView lv = findViewById(R.id.listview);
15           String[] cities = new String[]{"杭州","宁波","温州"};
16           int[] images = new int[]{R.drawable.hangzhou,R.drawable.ningbo,
17                                    R.drawable.wenzhou};
18           //图片资源引用本质上是 int 数据,用 int[]数组管理若干张图片资源引用
19           ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
20                                              android.R.layout.simple_list_item_1,cities);
21           lv.setAdapter(adapter);  //ListView 对象的使用方式：构造适配器,设置适配器
22           lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
23             // ListView 对象的列表单击事件处理是 setOnItemClickListener()方法
24             // Spinner 对象的选中事件处理是 setOnItemSelectedListener()方法
25             @Override
26             public void onItemClick(AdapterView<?> parent, View view,
27                                     int position, long id){
28                 iv.setImageResource(images[position]);
29                 //ImageView 对象设置图片的方式有多种,根据数据类型选择对应方法
30             }
31         });
32     }
33   }
```

4. 通过 XML 资源管理图片

本任务还可以通过 XML 资源文件实现数据管理。开发者通过向导在 res/values 文件

夹创建 images.xml 用于管理图片资源,如代码 3-12 所示,创建的数组 image_values 用于存放图片资源,该数组可以是 integer-array 也可以是通用型 array,子标签 item 通过@drawable/xxx 的方式引用图片资源,作为数组元素的值;创建的数组 image_names 用于管理图片名称,使之与数组 image_values 一一对应。

代码 3-12　图片资源管理文件 images.xml

```
<resources>
    <integer-array name="image_values">
        <item>@drawable/hangzhou</item>
        <item>@drawable/ningbo</item>
        <item>@drawable/wenzhou</item>
    </integer-array>
    <string-array name="image_names">
        <item>杭州</item>
        <item>宁波</item>
        <item>温州</item>
    </string-array>
</resources>
```

通过 XML 资源实现的逻辑控制如代码 3-13 所示,在 MainActivity 中,通过 ArrayAdapter 的静态方法 createFromResource()直接从 XML 资源数组 R.array.image_names 创建适配器。尽管图片资源在 XML 中定义的是 integer-array 数组,但直接取出的 int 值不能使用 setImageResource()方法设置图片,其解决方法是采用 obtainTypedArray()方法获得通用型数组,再调用该数组的 getDrawable()方法将图片资源二次解析成 Drawable 对象,并通过 ImageView 对象的 setImageDrawable()方法设置图片资源。

代码 3-13　通过 XML 资源实现的 MainActivity.java

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.res.TypedArray;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.ListView;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_main);
        ImageView iv = findViewById(R.id.imageView);
        ListView lv = findViewById(R.id.listview);
        ArrayAdapter adapter = ArrayAdapter.createFromResource(this,
                R.array.image_names, android.R.layout.simple_list_item_1);
        //不用从 adapter 中取数,因此可忽略泛型
        lv.setAdapter(adapter);
        TypedArray images = getResources()
                                .obtainTypedArray(R.array.image_values);
        lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
```

```
26                                                          int position, long id){
27                  Drawable drawable = images.getDrawable(position);
28                  iv.setImageDrawable(drawable);
29              }
30          });
31      }
32  }
```

3.5　使用 SimpleAdapter 生成复杂视图

3.5.1　任务说明

本任务的实现效果如图 3-17 所示。在该应用中，活动页面视图根节点为垂直的 LinearLayout，在布局中有 1 个 TextView，用于显示个人信息；1 个 ListView，用于显示城市信息。ListView 一行的视图较为复杂，左边是 1 个 ImageView，宽和高均为 80dp，采用中心点裁剪方式，使不同尺寸的图片在 ListView 中均具有相同的尺寸；ListView 行视图的中间是两个上下放置的 TextView，文字与左图有 10dp 的间距，上部 TextView 显示城市名称，文字颜色为黑色，字体大小为 20sp，下部 TextView 显示热线电话，默认字体；ListView 行视图的右边是一个拨打电话图标。单击 ListView 列表项，会弹出 Toast，显示所单击的城市名称。

图 3-17　SimpleAdapter 实现效果

3.5.2　任务实现

1. SimpleAdapter 适配器

在本任务中，ListView 不再是简单的一行行文字，而是采用自定义布局实现特定格式的样式，并具有复杂的数据。在本任务中，ListView 的一行视图具有图片资源、城市名称和电话号码等数据，需要自定义视图进行数据渲染。ArrayAdapter 以及内置的样式实现不了本任务的效果，若无复杂的逻辑需求，可通过 SimpleAdapter 实现。

SimpleAdapter 的构造方法如下：

```
public SimpleAdapter(android.content.Context context,
                     java.util.List <? extends java.util.Map < String, ?>> data,
                     @LayoutRes int resource, String[] from, @IdRes int[] to)
```

其中，context 是上下文；data 是复杂数据的表达，采用 List 类型数据，实际实现的时候可采用 ArrayList(List 的具体类)；resource 是自定义布局文件的引用，以 R.layout.xxx 的方式引用(xxx 是 res/layout/文件夹中 xml 布局文件名)；数组 from 指定了 data 元素中哪些字段的数据取出渲染适配器行视图中数组 to 指向的 UI，与数组 to 一一对应；数组 to 指向 resource 布局文件中需要渲染数据的 UI 的 id。

SimpleAdapter 的构造方法中,需要注意第 2 参数 data 是 List 数据类型,且该 List 的泛型是"<? extends java.util.Map<String,? >>",表示 List 中的元素数据是 Map 的继承类(具体实现可用 HashMap),"? extends java.util.Map"的问号表示某种类型,该类型通过 extends 继承自 Map 类型,而 Map 类型具有泛型<String,?>,表示该 Map 的 key 是 String 类型,value 可以是任何数据类型。传参 data 通过 Map 类型可封装任何数据,将多个不同种类的数据打包成一个元素,使之能对应适配器一行的视图。数组 from 和数组 to 是一一对应的关系,是具有相同长度的数组,数组 from 第 i 个 key 从 Map 对象中取出 value 渲染到数组 to 中第 i 个 id 指向的 UI。

2. 实现页面布局和自定义行视图

将本任务所需的图片复制到 res/drawable 文件夹中。MainActivity 对应的布局文件 my_main.xml 比较简单,如代码 3-14 所示。

适配器所需的自定义视图效果如图 3-18 所示,具体布局 row_view.xml 详见代码 3-15。在 row_view.xml 文件中,根节点是水平的 LinearLayout,宽度匹配父容器,高度包围内容,为了让左边组件(ImageView)、中间组件(嵌入的垂直 LinearLayout),以及右边组件(ImageView)均能垂直居中对齐,对根节点 LinearLayout 设置了 android:gravity 属性。行视图中所有 UI 的 id,均按"row_view_"前缀命名,以便于在编写代码时使得开发者更清楚该布局文件中的 UI。行视图左边的 ImageView 宽高均为 80dp,并通过 android:scaleType="centerCrop"属性使该 ImageView 从中心点缩放裁剪,保证不同尺寸的图片在行视图中均具有相同的尺寸,且不留白边。行视图中间是一个嵌入的垂直 LinearLayout,通过 android:paddingLeft 属性设置左内边距,通过 android:layout_weight 属性使之占据剩余的容器宽度,从而使右边的 ImageView 靠右对齐。在垂直的 LinearLayout 中,两个 TextView 默认情况下在布局中顶部对齐,可对 LinearLayout 设置 android:gravity 属性,使容器内部的 UI 为垂直居中对齐。行视图右边的 ImageView 在刚拖入容器时,有设置图片资源向导,如图 3-19 所示,可在搜索框中输入 call,查找 Android 内置的与搜索关键词相关的图标。ImageView 后期更改时,也可在 Design 界面中单击该组件,并在属性栏中单击 Pick a Resource 按钮调出资源设置向导,设置图片资源。

图 3-18　适配器自定义行视图效果

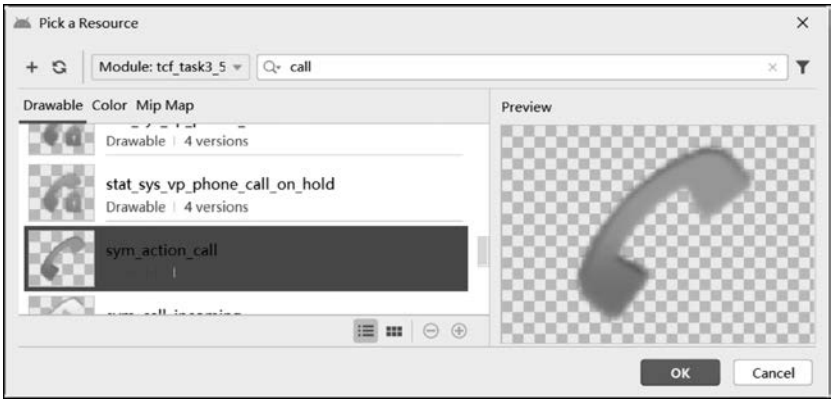

图 3-19　图片资源设置向导与搜索过滤

代码 3-14　活动页面布局文件 my_main. xml

```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    xmlns:app = "http://schemas.android.com/apk/res - auto"
    android:orientation = "vertical"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent">
    <TextView
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:text = "Your name and ID" />
    <ListView
        android:id = "@ + id/listview"
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content" />
</LinearLayout >
```

代码 3-15　适配器自定义视图 row_view. xml

```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    xmlns:app = "http://schemas.android.com/apk/res - auto"
    android:layout_width = "match_parent"
    android:gravity = "center_vertical"
    android:layout_height = "wrap_content">
<!--    根节点为水平的 LinearLayout,高度 wrap_content,设置所有内容垂直居中 -->
    <ImageView
        android:id = "@ + id/row_view_iv"
        android:layout_width = "80dp"
        android:layout_height = "80dp"
        android:scaleType = "centerCrop"
        app:srcCompat = "@drawable/hangzhou" />
<!--    所有的 id 以 row_view 作为前缀,便于查找 -->
<!--    ImageView 设置缩放为中线点裁剪 -->
    <LinearLayout
        android:layout_width = "wrap_content"
        android:layout_height = "match_parent"
        android:paddingLeft = "10dp"
        android:layout_weight = "1"
        android:gravity = "center_vertical"
        android:orientation = "vertical">
<!--    内嵌一个垂直的 LinearLayout,放置两个 TextView,设置权重使之占据剩余空间 -->
<!--    内嵌的 LinearLayout 设置左边距,并设置垂直居中 -->
        <TextView
            android:id = "@ + id/row_view_tv_name"
            android:layout_width = "match_parent"
            android:layout_height = "wrap_content"
            android:textSize = "20sp"
            android:textColor = "#000"
            android:text = "Name" />
        <TextView
            android:id = "@ + id/row_view_tv_phone"
            android:layout_width = "match_parent"
            android:layout_height = "wrap_content"
            android:text = "Phone" />
    </LinearLayout >
    <ImageView
        android:id = "@ + id/row_view_iv_call"
```

```
39              android:layout_width = "40dp"
40              android:layout_height = "40dp"
41              app:srcCompat = "@android:drawable/sym_action_call" />
42  <!--     row_view_iv_call 的图标可在图片资源设置向导中输入 call 搜索 -->
43  </LinearLayout >
```

3. 实现 MainActivity

MainActivity 的实现如代码 3-16 所示。HashMap 以 key-value 方式操作,尽管 key 可以直接输入字符串,但是这种方式很容易造成拼写错误,无形中给程序"挖坑"。例如,赋值的 key 采用 name,取数时 key 误拼为 mane,则存数取数 key 不一致,会造成取数时无相关 key 的错误。正规的写法是将 HashMap 所需要的 key 定义成常量(使用 static final 关键字修饰),并根据实际需要添加 public 或者 private 关键字修饰。public 关键字修饰表示该变量在其他类中能被访问。例如,外部类可通过 MainActivity. KEY _ IMAGE 访问 MainActivity 中定义的 public 常量 KEY_IMAGE,若该常量不想被外部类引用,可修改为 private 常量。static 关键字修饰表示该变量是静态的,不管类被生成多少个实例,均共享同一个静态变量,final 关键字修饰则表示不能重定向该变量,一般用纯大写表示常量。

在 ListView 一行的视图中,需要用城市图片、城市名称和市民热线电话渲染对应 UI,因此定义了数组 images 用于存储 3 张图片资源,数组 cities 用于存储城市名称,数组 phones 用于存储热线电话,但是这 3 个数组是独立的,可使用 HashMap 通过 key-value 的方式将不同数据打包成一种数据,并放到 List 列表中,使 List 一个数据对应 ListView 一行的视图。考虑到数组 images 的元素是 int 型,而数组 cities 和数组 phones 的元素是 String 型,在定义 HashMap 泛型时,value 对应的数据类型只能是 Object,用于涵盖不同的数据类型。本任务中,通过 for 循环,将 3 个数组的元素按 key-value 方式放入 HashMap 对象,最后将该对象添加到 ArrayList 对象中。

循环体的使用技巧:在开发环境中输入 fori 后按 Enter 键即可调出循环模板,完成部分代码自动填充;或者输入 images. fori 后按 Enter 键即可对 images 数组展开循环,并自动填充部分代码。

构造 SimpleAdapter 适配器时,数组 from 指定从 HashMap 对象的哪些 key 中取数,并将数据渲染到数组 to 对应的 UI 上。在 ListView 对象的 onItemClick()回调方法中,需要对列表 list 取指定位置的数据,编程时可先输入"list. get(position);",再利用 Ctrl+Alt+V 快捷键生成局部变量。HashMap 对象取数可通过 get()方法传入对应 key,取出 key 所对应的 value,在泛型约束中,value 被定义成 Object 类型,若想取出 String 型变量,则须对取出值进行强制类型转换。

代码 3-16　MainActivity. java

```
1   import androidx.appcompat.app.AppCompatActivity;
2   import android.os.Bundle;
3   import android.view.View;
4   import android.widget.AdapterView;
5   import android.widget.ListView;
6   import android.widget.SimpleAdapter;
7   import android.widget.Toast;
8   import java.util.ArrayList;
```

```
import java.util.HashMap;
public class MainActivity extends AppCompatActivity {
    public static final String KEY_IMAGE = "key_image";       //Ctrl + D 可复制当前行
    public static final String KEY_NAME = "key_name";
    //若 KEY_IMAGE 被定义成 private static final String,则外部类不能访问
    public static final String KEY_PHONE = "key_phone";
    //定义 3 个常量用于 HashMap 的 key
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_main);
        int[] images = new int[]{R.drawable.hangzhou,R.drawable.ningbo,
                                 R.drawable.wenzhou};
        String[] cities = new String[]{"杭州","宁波","温州"};
        String[] phones = new String[]{"0571 - 12345","0574 - 12345","0577 - 12345"};
        ListView lv = findViewById(R.id.listview);
        ArrayList < HashMap < String,Object >> list = new ArrayList <>();
        //SimpleAdapter 接受的数据是列表数据,且元素是 Map 对象
        //在 HashMap 中需要对数据封装,泛型约定 key 和 value 的数据类型
        //由于 value 中有 String 和 int 数据,所以用 Object 定义 value 类型
        //key 一般用 String 类型,用字符串常量,常量定义用纯大写
        for (int i = 0; i < images.length; i++) {
            HashMap < String,Object > hashMap = new HashMap <>();
            hashMap.put(KEY_IMAGE,images[i]);
            hashMap.put(KEY_NAME,cities[i]);
            hashMap.put(KEY_PHONE,phones[i]);
            //将图片、城市名称和电话号码通过 key - value 打包在同一个 HashMap 对象中
            list.add(hashMap);       //将 hashmap 添加到 list 末尾
        }
        String[] from = new String[]{KEY_IMAGE,KEY_NAME,KEY_PHONE};
        //from 定义 SimpleAdapter 从 HashMap 的哪些字段取数据
        int[] to = new int[]{R.id.row_view_iv,R.id.row_view_tv_name,
                             R.id.row_view_tv_phone};
        //HashMap 从 from 所定义 key 中取值,填充到 to 对应的 id 的 UI 上
        //from 和 to 是一一对应的,from 定义数据源字段,to 定义渲染的 UI 的 id
        SimpleAdapter adapter = new SimpleAdapter(this,list,R.layout.row_view,from,to);
        lv.setAdapter(adapter);
        lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView <?> parent, View view,
                                    int position, long id){
                HashMap < String, Object > hashMap = list.get(position);
                //从 list 对象中取数
                String city = (String) hashMap.get(KEY_NAME);
                //get()方法返回的是 Object
                Toast.makeText(getApplicationContext(),city,
                                                    Toast.LENGTH_SHORT).show();
                //Toast 显示单击列表项的城市名称
            }
        });
    }
}
```

SimpleAdapter 虽然能实现复杂的图文视图,但在实际使用中并不灵活。首先,数据源需要 HashMap 对数据封装,若数据是多种不同类型的,取数需要强制类型转换,非常不方便,此外,开发者还需要了解 key 和 value 的对应关系,开发便捷程度远不如直接将数据封

装成一个独立类。其次,数据源与视图只能机械地进行一一映射,无法处理根据内容调整视图的复杂逻辑。例如,在本任务中,若某个城市没有电话数据,则拨打电话图标不显示,该功能直接调用 SimpleAdapter 的构造方法很难实现。再次,一行复杂视图中,难以实现单击不同的 UI,使之能处理不同的逻辑。例如,在本任务中,单击电话图标使之显示电话(真实场景是拨打电话),单击其他地方则显示城市名称(真实场景是跳转到该城市的详情页面),使用 SimpleAdapter 难以实现。因此在实践中,更倾向于对复杂数据进行类封装,并对适配器改写,使得适配器能使用自定义类数据,生成自定义视图。

3.6 改写 ArrayAdapter 生成复杂视图

3.6.1 任务说明

本任务的演示效果如图 3-20 所示,活动页面的主布局和适配器的自定义行视图均与 3.5 节的任务相同,并在 3.5 节基础上增加了更多的逻辑处理。本任务所增加的逻辑处理:当某城市无电话号码时,对应行视图则无电话图标;单击列表项,Toast 显示城市名称;单击电话图标,Toast 显示电话号码。为了使得 ListView 能滚动,城市数据多于一屏幕能显示的数据数目(数据可重复)。

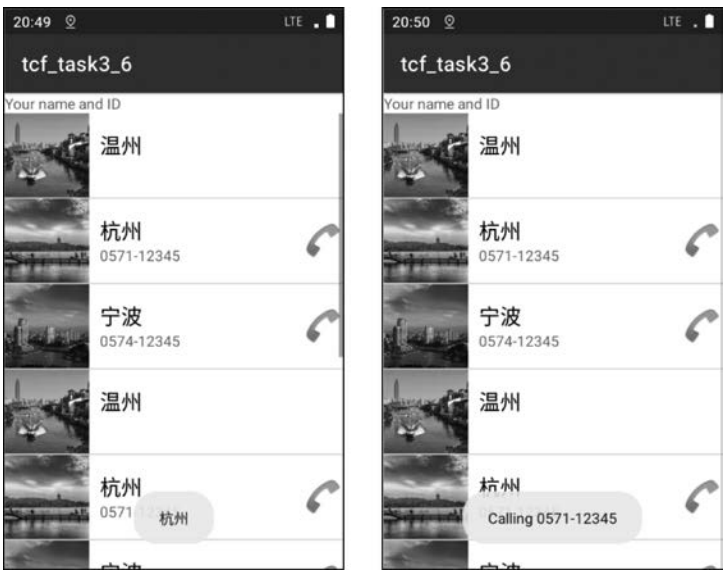

图 3-20 任务的演示效果

3.6.2 任务实现

本任务可重复利用 3.5 节项目的图片和布局资源,将所需图片从 3.5 节项目复制至本项目的 res/drawable 文件夹,将 3.5 节项目的 my_main.xml 和 row_view.xml 文件复制至本项目的 res/layout 文件夹。

1. 实现自定义数据类 City

由图 3-20 的适配器行视图可知,其一行的数据需要图片资源(可用 int 数据引用)、城市

名称(String)和热线电话(String),因此可对这些数据创建一个对应类,通过构造方法初始化数据,实现相关的 getter 和 setter 方法对类成员进行取数和修改数据。

右击项目包,在快捷菜单中选择 New→Java Class 选项创建新的类 City.java,如图 3-21 所示,并定义类的相关成员变量。注意,定义成员变量的顺序不同,会影响自动创建的构造方法的参数顺序,在定义之前须确定构造方法参数的顺序,再按顺序定义类成员变量,一般而言,类成员变量使用 private 关键字修饰。

完成类成员变量的定义后,即可利用开发环境自动生成构造方法和成员变量的存取方法。在 City.java 文件代码区右击,然后在快捷菜单中选择 Generate 选项(Alt+Insert 快捷键),在弹出的列表中选择 Constructor 选项,进而在弹出的对话框中选择相关变量(如图 3-22 所示),即可根据这些变量生成对应的构造方法,其生成的构造方法如代码 3-17 所示。继续使用 Generate 功能,选择 Getter and Setter 选项,选中相关变量,生成对应的取数存数方法,这些自动生成的方法非常规范,对非布尔值,生成 getXxx()和 setXxx()方法,布尔值则生成 isXxx()和 setXxx()方法,各方法采用驼峰写法,传入参数和返回值自动匹配对应变量类型。

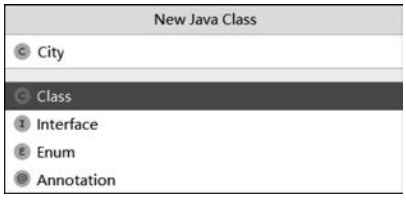

图 3-21　通过向导创建新的类

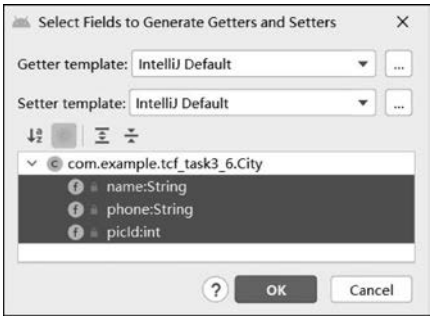

图 3-22　利用向导选择成员变量生成类的构造方法

自定义类 City 具体见代码 3-17,除了成员变量的定义,其他方法均是利用 Generate 功能自动生成代码。后期还可以根据需要,利用 Generate 功能生成 toString()方法,使得类在某些场合能根据需要自动调用 toString()方法,产生相关字符串,供打印日志等需求。

代码 3-17　自定义类 City.java

```
1    public class City {
2        private String name;
3        private String phone;
4        private int picId;
5        //Alt + Insert 快捷键,选择 Constructor→Getter and Setter 选项
6        //由向导创建构造方法和取数存数方法
7        public City(String name, String phone,int picId) {
8            this.name = name;
9            this.phone = phone;
10           this.picId = picId;
11       }
12       public int getPicId() {
13           return picId;
14       }
15       public void setPicId(int picId) {
16           this.picId = picId;
17       }
```

```
18          public String getName() {
19              return name;
20          }
21          public void setName(String name) {
22              this.name = name;
23          }
24          public String getPhone() {
25              return phone;
26          }
27          public void setPhone(String phone) {
28              this.phone = phone;
29          }
30      }
```

2. 改写 ArrayAdapter

通过自定义类 City.class 实现复杂数据的封装后,需要改写 ArrayAdapter,使之能根据自定义布局生成行视图,并将 City 类的对应数据渲染到行视图各 UI 上。改写 ArrayAdapter 的核心工作有两个:①通过构造方法传递必要的参数,使适配器能得到所需操作的对象,例如,上下文 Context 和数据源;②改写 getView()方法,在该方法中,通过 LayoutInflater 类将自定义布局生成 View 对象,并利用传参 position 取得数据源对应位置的单元数据将其渲染到行视图上。

自定义适配器 CityAdapter 如代码 3-18 所示。利用向导创建 CityAdapter 后,在类文件中,通过 extends 关键字,使之继承 ArrayAdapter,并给定泛型类型为 City。适配器需要数据源,为了提高通用性,可用 ArrayList 的父类 List 定义数据源,通过泛型指明接收的是 City 类数据。构造方法可用 Generate 向导生成,但是会额外增加很多参数,此时,可根据需要将多余的传参删除,精简自定义适配器构造方法的传参。在构造方法中,super 指的是本类的父类,即 ArrayAdapter,这里必须调用 super()方法即父类的构造方法生成适配器,让它能根据 UI 的视图和数据源,反复自动调用 getView()方法生成行视图并渲染数据。super()方法的 3 个参数就是 ArrayAdapter 构造方法的参数,其中,第 2 个参数是样式参数,会被 getView()方法覆盖,本质上不会起什么作用,但必须存在且是合理的参数,第 3 个参数是数据源。若采用双参数的 super()构造方法(没有传递数据源),编译器不会报错,但在实际运行中,不会生成视图,此时可对 CityAdapter 改写 getCount()方法,告知适配器数据的长度,并自动调用 getView()方法渲染行视图,具体实现如下所示。

```
@Override
public int getCount() {
    return list.size();
}
```

适配器的核心工作是反复调用 getView()方法,注意,该方法并不是一次性循环调用完毕,而是根据 Android 设备的屏幕和现状按需调用。假如数据长度是 1000,屏幕中最多能存放 10 行视图,则一开始,只会调用 10 次 getView()方法,生成 10 行视图,当用户滑动屏幕,系统会计算哪些行视图需要生成,并调用 getView()方法按需生成对应行视图。

在 getView()方法的参数中,传参 position 为行视图的位置索引,与数据源的位置一一对应;传参 convertView 是行视图对象,若此行视图需要重新生成,则为 null,若行视图可重

复利用则非空；传参 parent 为适配器所依赖的组件，若该适配器用于 ListView，则 parent 为 ListView 对象。行视图的重复利用指适配器生成一屏幕的行视图后，当用户滑动屏幕后，适配器调用 getView()方法更新某些行视图，可直接从旧行视图取得对象，此时 convertView 非空，指向某个旧行视图，只需将视图中的 UI 重新进行数据渲染，即可当成新的行视图使用。getView()方法也可以使用传参 parent 的 getContext()方法获得上下文，此时，所改写的适配器不需要传递上下文参数。

适配器经常需要滑动页面加载信息，为了提高效率，往往会对 getView()方法进行优化，若传参 convertView 非空，则直接根据传参 position 索引的数据渲染 convertView 中的各个 UI；若传参 convertView 为空，则利用 LayoutInflater 从布局文件生成 View 对象，再对 View 对象中的各 UI 进行数据渲染。LayoutInflater 为布局填充器，调用 from()静态方法传入上下文参数，再调用 inflate()方法将自定义布局生成视图赋给 View 对象。注意，在代码 3-18 中，行视图中的 UI 须通过 View 对象的 findViewById()方法将布局文件 R.layout.row_view 中的 UI 找出赋给对应 UI 对象。UI 不能直接调用 findViewById()方法，其原因是 Activity 类或者 View 对象支持该方法，但 ArrayAdapter 类没有对应方法。在 MainActivity 类中，findViewById()方法是由 Activity 对象提供的，从 Activity 布局文件中找出 UI，在代码 3-18 中，所使用的方法是适配器提供的，而适配器类本身并不提供 findViewById()方法，需要从 View 对象提供的方法中找出相关的 UI。

行视图数据渲染由 getView()方法完成，该方法的功能是：将数据源 list 中 position 索引位置的数据取出，并根据各 UI 对应的方法将数据渲染到行视图的相关 UI 上。判断文本是否为空最好使用 TextUtils.isEmpty()方法，它考虑了文本对象为 null 或者空字符串("")两种情况。控制 ImageView 对象不可见可通过 setVisibility(View.GONE)方法实现，控制 ImageView 对象可见则通过 setVisibility(View.VISIBLE)方法实现。适配器内部也可以实现特定 UI 的单击事件处理，代码 3-18 对电话图标的 ImageView 设置了单独的单击响应，并用 Toast 显示对应行数据的电话号码。当用户单击 ListView 的电话图标时，会响应适配器中的指定方法；当用户单击 ListView 其他地方时，则响应 ListView 的列表单击事件，与适配器的电话图标事件无关，从而实现了行视图不同 UI 有不同的响应处理。适配器 getView()方法需要返回行视图对象，使得经数据渲染后的行视图能在适配器的载体上显示。

代码 3-18　改写的适配器 CityAdapter.java

```
1    import android.content.Context;
2    import android.text.TextUtils;
3    import android.view.LayoutInflater;
4    import android.view.View;
5    import android.view.ViewGroup;
6    import android.widget.ArrayAdapter;
7    import android.widget.ImageView;
8    import android.widget.TextView;
9    import android.widget.Toast;
10   import androidx.annotation.NonNull;
11   import androidx.annotation.Nullable;
12   import java.util.List;
13   public class CityAdapter extends ArrayAdapter<City>{
```

```
14      private Context context;
15      private List<City> list;
16      //上下文参数 context(用于 LayoutInflater),数据源 list 用于行视图数据渲染
17      public CityAdapter(@NonNull Context context, List<City> list) {
18          super(context, android.R.layout.simple_list_item_1, list);
19          //super 就是父类 ArrayAdapter,因此构造方法使用 ArrayAdapter 相同的方法
20          //android.R.layout.simple_list_item_1 不会起作用,会被 getView()重新处理
21          //若使用 super(context, android.R.layout.simple_list_item_1)两个参数
22          //由于没有传数据,默认数据长度为 0,不会生成视图,此时需要改写 getCount()方法
23          this.context = context;
24          this.list = list;
25      }
26      @NonNull
27      @Override
28      public View getView(int position, @Nullable View convertView,
29                          @NonNull ViewGroup parent) {
30          //position 行位置索引
31          //convertView 为一行的视图,若不为空可对其利用,取出 UI,直接数据渲染
32          View v = convertView;
33          if(v == null){
34              v = LayoutInflater.from(context).inflate(R.layout.row_view,
35                      null,false);
36              //inflate()固定用法,false 不绑定到根视图上,根视图 root = null
37          }
38          ImageView iv = v.findViewById(R.id.row_view_iv);
39          //注意是 v 对象上的 findViewById()方法
40          TextView tv_name = v.findViewById(R.id.row_view_tv_name);
41          TextView tv_phone = v.findViewById(R.id.row_view_tv_phone);
42          ImageView iv_call = v.findViewById(R.id.row_view_iv_call);
43          City city = list.get(position);
44          iv.setImageResource(city.getPicId());
45          String phone = city.getPhone();
46          if(TextUtils.isEmpty(phone)){
47              //TextUtils.isEmpty(phone)相当于 phone == null||phone.equals("")
48              iv_call.setVisibility(View.GONE);     //让 iv_call 消失
49          }else {
50              iv_call.setVisibility(View.VISIBLE);
51              //注释本行,并上下滑动 ListView,行视图图片会超预期消失
52          }
53          tv_name.setText(city.getName());
54          tv_phone.setText(phone);
55          iv_call.setOnClickListener(new View.OnClickListener() {
56              @Override
57              public void onClick(View v) {
58                  Toast.makeText(context,"Calling " + phone,
59                                  Toast.LENGTH_SHORT).show();
60                  //对 iv_call 单独设置单击事件处理,与 ListView 的列表单击事件不冲突
61              }
62          });
63          return v;
64       }
65  }
```

3. 实现 MainActivity

MainActivity 的实现如代码 3-19 所示,通过 City 和 CityAdapter 分别对数据进行类封装和自定义适配器渲染数据后,主程序的代码要比 3.5 节中采用 HashMap 和 SimpleAdapter 的实

现方式简洁。在 onCreate()方法中,生成 list 数据时,通过 for 循环创建了较多的重复数据,使得 ListView 的数据多于一屏幕所能显示的数据数目。将代码 3-18 第 50 行注释掉,对 ListView 反复上下滑动屏幕,使其不断调用 getView()方法更新行视图,会发现行视图的电话图标逐渐消失。造成该现象的原因是,getView()方法对 convertView 重复利用,被重复利用的旧视图中,消失了的电话图标没有调用 setVisibility(View. VISIBLE)方法显现视图,导致最终所有被重复利用的行视图的电话图标均消失。若 getView()方法每次都直接调用 LayoutInflater 生成行视图,不对旧视图 convertView 重复利用,则不会出现反复滑动 ListView 导致的电话图标消失现象,但这种做法每次均要从布局中生成视图,降低了效率。

代码 3-19　MainActivity. java

```
1   import androidx.appcompat.app.AppCompatActivity;
2   import android.os.Bundle;
3   import android.view.View;
4   import android.widget.AdapterView;
5   import android.widget.ListView;
6   import android.widget.Toast;
7   import java.util.ArrayList;
8   public class MainActivity extends AppCompatActivity {
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.my_main);
13         ListView lv = findViewById(R.id.listview);
14         ArrayList<City> list = new ArrayList<>();
15         list.add(new City("温州","",R.drawable.wenzhou));
16         //电话用""或 null 分别定义数据
17         for (int i = 0; i<3 ; i++) {
18             list.add(new City("杭州","0571-12345",R.drawable.hangzhou));
19             list.add(new City("宁波","0574-12345",R.drawable.ningbo));
20             list.add(new City("温州",null,R.drawable.wenzhou));
21         }
22         CityAdapter adapter = new CityAdapter(this,list);
23         //使用 CityAdapter,主程序代码更简洁,同时也提高了代码的复用性
24         lv.setAdapter(adapter);
25         lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
26             @Override
27              public void onItemClick(AdapterView<?> parent, View view,
28                                                      int position, long id) {
29                 Toast.makeText(getApplicationContext(),
30                       list.get(position).getName(), Toast.LENGTH_SHORT).show();
31              }
32          });
33       }
34  }
```

3.7　使用网格视图

3.7.1　任务说明

本任务的演示效果如图 3-23 所示。在该应用中,活动页面视图根节点为垂直的 LinearLayout,在布局中依次有 1 个 TextView,用于显示个人信息;1 个 TextView,用于显

示 GridView(网格视图)的单击结果；1 个 GridView，具有 3 列，每列等宽，3 列占据屏幕的整个宽度。GridView 的数据使用循环程序生成 100 个数据，依次为 Item00 至 Item99。

图 3-23　GridView 演示效果

3.7.2　任务实现

1. GridView 简介

GridView 的使用方法与 ListView 类似，需要适配器，相应的单击事件亦与 ListView 相同。GridView 作为网格视图，支持一行显示多列，每个单元格对应数据源的一个数据，因此 GridView 的显示内容往往比 ListView 更紧凑。

在布局中，GridView 相比 ListView 需要使用更多的属性，如下所示。

(1) android:numColumns，控制 GridView 的列数，属性值可以是直接的数字，表示有多少列，也可以是 auto_fit，由系统根据 GridView 的列宽和父容器宽度自动计算列数。

(2) android:columnWidth，控制列宽，如果 android:numColumns 有确定的数值，列宽属性一般上不再使用。

(3) android:gravity，用于控制 GridView 单元格内的对齐方式。

(4) android:horizontalSpacing 和 android:verticalSpacing，分别控制单元格之间的水平间距(列间距)和垂直间距(行间距)。

(5) android:stretchMode，控制 GridView 分配完列空间后，对剩余空间的处理方式。若属性值是 none，则剩余空间不扩展到列宽或列间距上；若属性值为 spacingWidth，则将剩余空间分配到列间距上，但是首列左边和末列右边不分配；若属性值为 columnWidth，则将剩余空间平均分配到列宽上；若属性值为 spacingWidthUniform，则将剩余空间分配到列间距上，且首列左边和末列右边均分配。

2. 实现 UI 布局

本任务的布局文件比较简单，主布局 my_main. xml 如代码 3-20 所示，GridView 仅用了两个额外属性：android:numColumns，设置列数为 3；android:stretchMode，将剩余空间分配给列宽。

代码 3-20　布局文件 my_main. xml

```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:orientation = "vertical"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent">
    <TextView
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:text = "Your name and ID" />
    <TextView
        android:id = "@ + id/tv_result"
        android:layout_width = "match_parent"
```

```
            android:layout_height = "wrap_content"
            android:text = "" />
        < GridView
            android:id = "@ + id/gridView"
            android:layout_width = "match_parent"
            android:numColumns = "3"
            android:stretchMode = "columnWidth"
            android:layout_height = "match_parent" />
<!--    android:numColumns = "3" GridView 有 3 列 -->
<!--    android:stretchMode = "columnWidth" 剩余空间平均分配给列宽 -->
</LinearLayout >
```

3. 实现 MainActivity

MainActivity 的实现如代码 3-21 所示。在 onCreate()方法中,通过 for 循环和 String.format()方法,生成 100 个字符串数据赋给 ArrayList 对象,作为 GridView 的数据源。适配器使用 ArrayAdapter,单元格单击事件与 ListView 相同,均采用 OnItemClickListener 接口。

代码 3-21　MainActivity.java

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.GridView;
import android.widget.TextView;
import java.util.ArrayList;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_main);
        ArrayList < String > list = new ArrayList <>();
        for (int i = 0; i < 100; i++) {
            String item = String.format("Item % 02d", i);
            //用 String.format()打印格式化的字符串
            list.add(item);
        }
        ArrayAdapter < String > adapter = new ArrayAdapter <>(this,
                android.R.layout.simple_list_item_1,list);
        GridView gv = findViewById(R.id.gridView);
        TextView tv = findViewById(R.id.tv_result);
        gv.setAdapter(adapter);
        //使用方法与 ListView 相似
        gv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView <?> parent, View view,
                                                    int position, long id) {
                String item = list.get(position);
                tv.setText(item);
            }
        });
    }
}
```

3.8 列表视图与网格视图的动态切换

3.8.1 任务说明

本任务的演示效果如图 3-24 所示。在该应用中,活动页面视图根节点为垂直的 LinearLayout,在布局中依次有 1 个 TextView,用于显示个人信息;1 个 Switch 组件,控制视图类型;1 个 ListView 或者 GridView。Switch 组件对视图的控制逻辑为:当 Switch 组件处于开的状态,使用 ListView 显示数据,且 Switch 组件的文字为"列表视图";当 Switch 组件处于关的状态,使用 GridView 显示数据,且 Switch 组件的文字为"网格视图"。

图 3-24 通过 Switch 组件动态切换视图

3.8.2 任务实现

1. 实现适配器布局

由图 3-24 可知,在本任务中,ListView 和 GridView 对应的单元视图略有不同。ListView 行视图中各 UI 是左右关系,GridView 单元格视图中则是上下关系,因此适配器所需的视图应分开设计。ListView 适配器的行视图文件 row_view_list_view.xml 如代码 3-22 所示,根节点为水平的 LinearLayout;GridView 适配器的单元格视图文件 row_view_grid_view.xml 如代码 3-23 所示,根节点为垂直的 LinearLayout。两个布局文件对应的 UI 使用了相同的 id,以便于使用同一个适配器进行统一处理。

代码 3-22 ListView 的行视图布局文件 row_view_list_view.xml

```
1    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2        xmlns:app="http://schemas.android.com/apk/res-auto"
3        android:layout_width="match_parent"
4        android:layout_height="wrap_content"
```

```
        android:gravity = "center">
        < ImageView
              android:id = "@ + id/row_view_iv"
              android:layout_width = "80dp"
              android:layout_height = "80dp"
              app:srcCompat = "@drawable/hangzhou" />
        < TextView
              android:id = "@ + id/row_view_tv"
              android:layout_width = "match_parent"
              android:layout_margin = "5dp"
              android:layout_height = "wrap_content"
              android:textColor = "#000"
              android:text = "Name" />
</LinearLayout >
```

代码 3-23　GridView 的单元格布局文件 row_view_grid_view. xml

```
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
      xmlns:app = "http://schemas.android.com/apk/res - auto"
      android:layout_width = "match_parent"
      android:layout_height = "wrap_content"
      android:orientation = "vertical"
      android:gravity = "center">
      < ImageView
            android:id = "@ + id/row_view_iv"
            android:layout_width = "80dp"
            android:layout_height = "80dp"
            app:srcCompat = "@drawable/hangzhou" />
      < TextView
            android:id = "@ + id/row_view_tv"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:gravity = "center"
            android:textColor = "#000"
            android:text = "Name" />
</LinearLayout >
```

2. 实现自定义适配器

本任务通过 Switch 组件控制内容的展现方式，能在 ListView 和 GridView 之间动态切换，两种视图使用的是同一个数据源。考虑到 ListView 和 GridView 均需要使用适配器，不妨直接设计一个适配器，使之既能适用于 ListView 又能用于 GridView，但是两种视图的单元视图的布局会存在一些细微差异。因此，对两种不同视图采用的是两个对应的单元视图布局文件，同时为了提高复用性，两个布局文件中的 UI 使用相同的 id，使得适配器在调用 findViewById()方法时能传递相同的参数进行统一处理。

视图中单元格数据只有两个字段：图片的 id 值和城市名称，对应的数据封装类 City 如代码 3-24 所示。

代码 3-24　自定义类 City. java

```
public class City {
    private String name;
    private int picId;
    public City(String name, int picId) {
```

```
5            this.name = name;
6            this.picId = picId;
7        }
8        public String getName() {
9            return name;
10       }
11       public void setName(String name) {
12            this.name = name;
13       }
14       public int getPicId() {
15            return picId;
16       }
17       public void setPicId(int picId) {
18            this.picId = picId;
19       }
20   }
```

自定义适配器 CityAdapter 如代码 3-25 所示。在 CityAdapter 中使用了如下两种构造方法。

(1) public CityAdapter(@NonNull Context context, List<City> list, int type)。

(2) public CityAdapter(@NonNull Context context, List<City> list)。

两种构造方法中,前两个参数相同,分别是上下文 Context 对象和 List 对象。在第 1 个构造方法中,通过传参 type 传递控制视图类型的参数,若 type 值为 CityAdapter.TYPE_LIST_VIEW,则在 getView()方法中,按 ListView 的行布局文件生成单元视图;若 type 值为 CityAdapter.TYPE_GRID_VIEW,则在 getView()方法中按 GridView 的单元格布局文件生成单元格视图。为了在其他类中能引用 type 的两种值,在 CityAdapter 中定义了 public 关键字修饰的两个 int 常量值 TYPE_LIST_VIEW 和 TYPE_GRID_VIEW,取值可以随意定义(但必须不同),这里简单地分别取值 1 和 2。

CityAdapter 类所改写的 getView()方法中,根据传参 type 的值分别加载对应的视图布局文件,由于两个布局文件中对应的 UI 使用了相同的 id 值,在使用 findViewById()方法关联 UI 时可用相同的代码,不用区分处理。本任务中,CityAdapter 具备一定的灵活性,使之能通过传参 type 控制,既能用于 ListView 视图,也能用于 GridView 视图。

代码 3-25　自定义适配器 CityAdapter.java

```
1    import android.content.Context;
2    import android.view.LayoutInflater;
3    import android.view.View;
4    import android.view.ViewGroup;
5    import android.widget.ArrayAdapter;
6    import android.widget.ImageView;
7    import android.widget.TextView;
8    import androidx.annotation.NonNull;
9    import androidx.annotation.Nullable;
10   import java.util.List;
11   public class CityAdapter extends ArrayAdapter<City> {
12       private Context context;
13       private List<City> list;
14       private int type;              //通过该参数控制适配器用于 ListView 或 GridView
15       public static final int TYPE_LIST_VIEW = 1;
```

```
    public static final int TYPE_GRID_VIEW = 2;
    public CityAdapter(@NonNull Context context, List<City> list, int type) {
        //type 参数控制视图类型
        super(context, android.R.layout.simple_list_item_1,list);
        this.context = context;
        this.list = list;
        this.type = type;
    }
    public CityAdapter(@NonNull Context context, List<City> list) {
        //使用该构造方法默认用于 ListView
        super(context, android.R.layout.simple_list_item_1,list);
        this.context = context;
        this.list = list;
        type = TYPE_LIST_VIEW;              //type 默认值为 TYPE_LIST_VIEW
    }
    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView,
                        @NonNull ViewGroup parent) {
        View v = convertView;
        if(v == null){
           if(type == TYPE_LIST_VIEW) {
              v = LayoutInflater.from(context)
                          .inflate(R.layout.row_view_list_view, null, false);
           }else {
              v = LayoutInflater.from(context)
                           .inflate(R.layout.row_view_grid_view, null, false);
           }
           //根据 type 类型,从对应的布局文件生成视图对象 v
        }
        ImageView iv = v.findViewById(R.id.row_view_iv);
        TextView tv = v.findViewById(R.id.row_view_tv);
        //两个不同的布局文件对应的 UI 使用相同的 id,避免了区分处理
        City city = list.get(position);
        iv.setImageResource(city.getPicId());
        tv.setText(city.getName());
        return v;
     }
}
```

3. 实现活动页面的布局

MainActivity 的布局文件 my_main.xml 如代码 3-26 所示，在布局中，Switch 组件具有二值状态，由 android:checked 属性控制，若值是 true 则为开的状态，若值是 false 则为关的状态。Switch 使用 android:layout_gravity 控制其相对父容器的对齐方式，实现了居中对齐。根节点 LinearLayout 中嵌入一个子 LinearLayout，id 为 container，需要在逻辑代码中找出 container 容器，并对其控制，实现动态加载 ListView 或者 GridView。

代码 3-26　MainActivity 的布局文件 my_main.xml

```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
      android:orientation = "vertical"
      android:layout_width = "match_parent"
      android:layout_height = "match_parent">
      <TextView
           android:layout_width = "match_parent"
```

```
            android:layout_height = "wrap_content"
            android:text = "Your name and ID" />
        < Switch
            android:id = "@ + id/switch1"
            android:layout_width = "wrap_content"
            android:layout_height = "wrap_content"
            android:layout_gravity = "center"
            android:checked = "true"
            android:text = "列表视图" />
<!--     使用 Switch 组件切换视图,设置默认选中状态为列表视图 -->
        < LinearLayout
            android:id = "@ + id/container"
            android:layout_width = "match_parent"
            android:layout_height = "match_parent"
            android:orientation = "vertical">
<!--        使用 container 线性布局动态生成 ListView 或者 GridView -->
<!--        根据 Switch 取值,动态加载 list_view.xml 或者 grid_view.xml -->
        </LinearLayout >
</LinearLayout >
```

Switch 控制 MainActivity 加载 ListView 或者 GridView,在 my_main.xml 布局中不适合直接放入 ListView 或者 GridView 组件,否则在 Switch 切换时对 ListView 或 GridView 进行卸载和加载时会很烦琐。可以在 my_main.xml 布局中嵌入 id 为 container 的 LinearLayout,并使用逻辑代码对 container 进行加载或卸载 UI。因此,除了 my_main.xml 布局文件,还需要额外创建两个布局文件分别将 ListView 和 GridView 放到单独的布局文件中,以便于利用 XML 控制对应 UI 的属性。ListView 的单独布局文件为 list_view.xml,如代码 3-27 所示;GridView 的单独布局文件为 grid_view.xml,如代码 3-28 所示。ListView 和 GridView 在两个布局文件中均使用了相同的 id 值 adapterView,以便于逻辑代码的处理。

代码 3-27　ListView 的单独布局文件 list_view.xml

```
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:orientation = "vertical"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent">
    < ListView
        android:id = "@ + id/adapterView"
        android:layout_width = "match_parent"
        android:layout_height = "match_parent" />
</LinearLayout >
```

代码 3-28　GridView 的单独布局文件 grid_view.xml

```
< LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:orientation = "vertical"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent">
    < GridView
        android:id = "@ + id/adapterView"
        android:layout_width = "match_parent"
        android:numColumns = "3"
        android:stretchMode = "columnWidth"
        android:layout_height = "match_parent" />
</LinearLayout >
```

4. 实现 MainActivity

MainActivity 的实现如代码 3-29 所示。在 onCreate()方法中，调用了 updateView()方法实现视图的动态加载。updateView()方法需要访问列表数据 list、动态加载的容器 container，以及控制数据视图类型的参数 type，因此相关的 3 个变量应声明为 MainActivity 的成员变量。

Switch 组件的使用相对比较简单，其开关状态事件与 CheckBox 类似。在 onCheckedChanged()回调方法中，根据传参 isChecked 的值，更改 Switch 组件本身的文本和 type 值，传参 buttonView 即为 Switch 组件的对象实例。

在 updateView()方法中，根据变量 type 的值将变量 layout 赋为对应的布局文件，用于指向 ListView 或 GridView 视图。容器 container 是 1 个垂直的 LinearLayout，定义在主布局文件 my_main.xml 中。反复调用 updateView()方法的过程中，容器 container 会存在之前加载的 ListView 或 GridView，需要调用 removeAllViews()方法将 container 中的 UI 移除，再重新加载。加载视图时，使用 LayoutInflater 从 ListView 或者 GridView 布局文件中生成 View 对象 v，并且运用 ListView 和 GridView 均继承自 AdapterView 这一特点，直接定义了 AdapterView 对象，AdapterView 对象泛型参数是适配器。鉴于 list_view.xml 和 grid_view.xml 布局文件中，ListView 和 GridView 均使用了相同的 id 值，逻辑处理时，可统一使用 findViewById()方法找出 UI，并对其设置适配器，实现代码的统一。若 ListView 和 GridView 使用不同的 id 值，则须根据变量 type 的值分别调用 findViewById()方法关联 UI，并对其分别设置适配器。

代码 3-29　MainActivity.java

```
1   import androidx.appcompat.app.AppCompatActivity;
2   import android.os.Bundle;
3   import android.view.LayoutInflater;
4   import android.view.View;
5   import android.widget.AdapterView;
6   import android.widget.CompoundButton;
7   import android.widget.LinearLayout;
8   import android.widget.Switch;
9   import java.util.ArrayList;
10  public class MainActivity extends AppCompatActivity {
11      ArrayList<City> list;
12      int type = CityAdapter.TYPE_LIST_VIEW;
13      LinearLayout container;
14      @Override
15      protected void onCreate(Bundle savedInstanceState) {
16          super.onCreate(savedInstanceState);
17          setContentView(R.layout.my_main);
18          Switch sw1 = findViewById(R.id.switch1);
19          list = new ArrayList<>();
20          for (int i = 0; i < 4; i++) {
21              list.add(new City("杭州",R.drawable.hangzhou));
22              list.add(new City("宁波",R.drawable.ningbo));
23              list.add(new City("温州",R.drawable.wenzhou));
24          }
25          container = findViewById(R.id.container);
26          updateView();
27          sw1.setOnCheckedChangeListener(
```

```
28                          new CompoundButton.OnCheckedChangeListener() {
29              @Override
30              public void onCheckedChanged(CompoundButton buttonView,
31                                                      boolean isChecked) {
32                  if(isChecked){
33                      type = CityAdapter.TYPE_LIST_VIEW;
34                      buttonView.setText("列表视图");
35                  }else {
36                      type = CityAdapter.TYPE_GRID_VIEW;
37                      buttonView.setText("网格视图");
38                  }
39                  updateView();
40              }
41          });
42      }
43      private void updateView() {
44          //动态加载视图到 container
45          int layout;
46          if(type == CityAdapter.TYPE_LIST_VIEW){
47          //根据 type 类型,给 layout 赋对应布局文件的引用
48              layout = R.layout.list_view;        //list_view.xml 中有 1 个 ListView 组件
49          }else {
50              layout = R.layout.grid_view;        //grid_view.xml 中有 1 个 GridView 组件
51          }
52          container.removeAllViews();
53          //先清除 container 中的所有视图
54          View v = LayoutInflater.from(MainActivity.this)
55                                  .inflate(layout, null, false);
56          CityAdapter adapter = new CityAdapter(MainActivity.this,list,type);
57          AdapterView<CityAdapter> adapterView = v.findViewById(R.id.adapterView);
58          //要从视图对象 v 中找 adapterView
59          //利用 ListView 和 GridView 均继承自 AdapterView 的特点,实现统一的对象操作
60          adapterView.setAdapter(adapter);
61          container.addView(v);                   //将视图 v 加载到容器 container 上
62      }
63  }
```

3.9 使用 RecyclerView

3.9.1 任务说明

本任务的演示效果如图 3-25 所示。在该应用中,活动页面视图根节点为垂直的 LinearLayout,在布局中依次有 1 个 TextView,用于显示个人信息;1 个 Switch 组件,用于切换视图布局;1 个 RecyclerView,用于显示城市图片和城市名称。当 Switch 组件开关状态为开,开关文本为“水平列表”,RecyclerView 的内容为可水平方向左右滑动的列表视图;当 Switch 组件开关状态为关,开关文本为“网格布局”,RecyclerView 的内容为 3 列的网格视图。单击 RecyclerView 上的内容,Toast 显示城市名称。

3.9.2 任务实现

1. 实现 UI 布局

RecyclerView 的单元视图为 row_view.xml 文件,具体如代码 3-30 所示。在视图文件

图 3-25　使用 RecyclerView 实现视图布局的动态切换

中，出于美观考虑，对根节点的 LinearLayout 增加了外边距和对齐方式，并且单元视图的根节点宽度和高度都是 wrap_content，ImageView 则使用了 android:scaleType 属性使之从图片中心点裁剪缩放。

代码 3-30　单元视图布局文件 row_view.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:orientation="vertical"
    android:gravity="center">
    <ImageView
        android:id="@+id/row_view_iv"
        android:layout_width="80dp"
        android:layout_height="80dp"
        android:scaleType="centerCrop"
        app:srcCompat="@drawable/hangzhou" />
    <TextView
        android:id="@+id/row_view_tv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="#000"
        android:text="Name" />
</LinearLayout>
```

MainActivity 的布局文件 my_main.xml 如代码 3-31 所示。在布局中，Switch 组件的处理方式与 3.8 节的任务类似。RecyclerView 组件可通过 UI 面板直接拖入，注意该 UI 使用了全名 androidx.recyclerview.widget.RecyclerView。RecyclerView 使用 android:layout_gravity 属性设置其相对父容器居中对齐。

代码 3-31　MainActivity 布局文件 my_main. xml

```
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
    android:orientation = "vertical"
    android:layout_width = "match_parent"
    android:layout_height = "match_parent">
    <TextView
        android:layout_width = "match_parent"
        android:layout_height = "wrap_content"
        android:text = "Your name and ID" />
    <Switch
        android:id = "@ + id/switch1"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_gravity = "center"
        android:checked = "true"
        android:text = "水平列表" />
    <androidx.recyclerview.widget.RecyclerView
        android:id = "@ + id/recyclerView"
        android:layout_gravity = "center"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content" />
</LinearLayout>
```

2. RecyclerView 简介

RecyclerView 相比 ListView 和 GridView,功能要强大很多,但使用它所涉及的代码也会更多。RecyclerView 对应的包根据不同版本的 Android Studio 以及 SDK 有较大区别。早期的 RecyclerView 需要在 Gradle 中配置相关依赖,非常不友好。近年来的 Android Studio 已将 RecyclerView 统一到 androidx. recyclerview. widget. RecyclerView,无须在 Gradle 中配置对应的依赖,提高了项目对不同开发环境的适用性。

Android 提供的 RecyclerView 能支持线性布局、网格布局,以及瀑布流布局等布局管理器,支持横向滚动(ListView 默认只能实现纵向滚动),并且数据渲染效率也高于 ListView 和 GridView。

3. 实现 RecyclerView 适配器

使用 RecyclerView 没有像 ListView 那样方便,有直接支持的适配器可用,RecyclerView 需要用户自己写适配器,并且适配器需要改写的方法也更多,此外,RecyclerView 没有提供类似 ListView 的单击侦听接口,需要用户自行实现单击侦听。在实际使用中,若将单击侦听放在适配器中实现,其适用性较差,当用户对单击回调所要实现的功能因场景不同而不同时,适配器不可能将这些场景都考虑进去并一一实现。最佳的实践方式是在适配器中自定义一个接口和回调方法,将单击项的数据和位置通过回调方法回传给用户处理。用户使用该适配器的同时,需要实现适配器的自定义接口和回调方法,从而能根据应用需要,进行特定场景处理。

本任务所用的 City 类定义同 3.8 节,具体参考代码 3-24,项目中的图片也使用了相同的资源。

本任务针对 RecyclerView 和 City 类,自定义了对应的适配器:CityRecyclerAdapter,如代码 3-32 所示。CityRecyclerAdapter 需要继承 RecyclerView. Adapter,RecyclerView. Adapter 具有泛型参数 RecyclerView. ViewHolder(或者 ViewHolder 的继承类),在本任务

中，泛型参数使用继承自 ViewHolder 的自定义类 CityViewHolder，为了方便使用，CityViewHolder 是直接定义在 CityRecyclerAdapter 内部的公开类。回顾 3.6 节内容，ArrayAdapter 继承类改写过程中，当适配器在生成单元视图时，需要调用单元视图对象的 findViewById()方法找出相关 UI，再对这些 UI 进行数据渲染，这意味着每个新生成的单元视图均要调用 findViewById()方法关联 UI，效率不高。RecyclerView 的适配器对此做了优化，将单元视图交给 onCreateViewHolder()方法和 onBindViewHolder()方法进行处理，其中，onCreateViewHolder()方法负责单元视图的生成以及 UI 绑定，onBindViewHolder()方法则负责单元视图的数据渲染。鉴于实际运行中，onCreateViewHolder()方法的调用次数远少于 onBindViewHolder()方法，从而使适配器的性能得到优化。相比之下，若是改写 ArrayAdapter，则每次调用 getView()方法均要绑定 UI 和渲染数据，性能劣于 RecyclerView 的适配器。

RecyclerView 适配器的优化机制分析。CityRecyclerAdapter 利用 Logcat 在 onCreateViewHolder()方法和 onBindViewHolder()方法中打印日志，当运行应用并反复滑动 RecyclerView 时，便可观察两个方法各自的打印次数，用于验证各方法的调用次数。基于单元视图生成的次数远少于单元视图数据渲染的次数，以及 View 对象调用 findViewById()方法会增加系统消耗的事实，要优化 RecyclerView 的效率，就要尽可能减少 findViewById()方法的调用次数。显然在 onBindViewHolder()方法中，做数据渲染时调用 findViewById()方法是不合适的，达不到优化的目的，进一步而言，通过 onBindViewHolde()方法传递单元视图对象也没有意义，因为所传递的单元视图对象依然需要调用 findViewById()方法关联 UI。针对这个问题，RecyclerView 适配器设计得非常巧妙，通过 ViewHolder 构造单元视图，并把单元视图中的 UI 作为 ViewHolder 的成员变量，使之能从 ViewHolder 对象中直接取出，从而，在 onBindViewHolder()方法中传递进来的不是单元视图，而是对应的 ViewHolder 对象，进而实现了 onBindViewHolder()方法做数据渲染时，只需要从 ViewHolder 对象里取出 UI 成员直接渲染数据，避免了反复调用 findViewById()方法，达到提升效率的目的。RecyclerView.ViewHolder 并没有用户自定义的单元视图 UI 成员，因此需要写一个自定义类去继承 ViewHolder，并定义相关的 UI 成员以及实现 UI 成员的初始化。本任务中，ViewHolder 的继承类为 CityViewHolder，定义在 CityRecyclerAdapter 类中，作为内部类使用。鉴于 RecyclerView 需要对单元视图设置单击侦听，因此在 CityViewHolder 中增加了单元视图对象 rowView，使之作为成员变量，并且能在 onBindViewHolder()方法中被设置单击侦听。

通过以上分析可知，onCreateViewHolder()方法生成单元视图时，并不是简单地返回该单元视图，而是将之作为 CityViewHolder 的构造参数，让 CityViewHolder 把单元视图中的 UI 抽取出来成为自己的成员变量，进而，在 onBindViewHolder()方法进行数据渲染时可方便地从 onCreateViewHolder()方法所返回的 CityViewHolder 对象中取出 UI，大大减少了 findViewById()方法的调用次数。适配器还需要改写 getItemCount()方法返回数据长度，使适配器知晓需要生成以及渲染数据的单元视图总数。

CityRecyclerAdapter 实现了自定义接口 OnItemClickListener 用于单击侦听，在该接口中定义一个回调方法 onItemClick(CityViewHolder holder, int position, City city)，将 ViewHolder 对象 holder、位置索引 position 和单元视图所对应的数据 city 通过回调方法传

出,供调用者实现接口方法时取得所需数据,进行具体的单击事件场景化处理。CityRecyclerAdapter 的设计者根本不用关心调用者要对视图单击事件做什么,只需要将该传的数据传出去,实现代码解耦,方便后期维护和代码复用。单击事件侦听和回调触发在 onBindViewHolder()方法中实现,处理过程中,先对接口对象判断是否为空,若非空,则对 CityViewHolder 的成员变量 rowView(单元视图)设置单击侦听,并在单击侦听中调用自定义接口的回调方法。当用户使用 CityRecyclerAdapter 时实现了 OnItemClickListener 接口中的 onItemClick()方法,就能按用户的意愿实现场景化处理。此外,onBindViewHolder()方法的传参 position 没有用 final 关键字修饰,用户单击行视图时与 onBindViewHolder()方法执行数据渲染的时间点可能相差很大,此时传参 position 已经被回收消亡,为非可靠变量,因此在 OnItemClickListener 接口的 onItemClick()方法中,传递 ViewHolder 对象的 getAdapterPosition()方法所获得的位置信息比传参 position 更可靠。

代码 3-32　自定义适配器 CityRecyclerAdapter.java

```
import android.content.Context;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;
import java.util.List;
public class CityRecyclerAdapter extends
        RecyclerView.Adapter<CityRecyclerAdapter.CityViewHolder>{
    private Context context;
    private List<City> list;
    private int createViewHolderCounter=0;        //测试变量,实际中应删除
    private int bindViewHolderCounter=0;          //测试变量,实际中应删除
    private OnItemClickListener onItemClickListener;
    //自定义的单元视图单击侦听接口
    public CityRecyclerAdapter(Context context, List<City> list){
        this.context = context;
        this.list = list;
    }
    interface OnItemClickListener{
        //自定义接口,让用户使用适配器时能处理单元视图单击事件
        public void onItemClick(CityViewHolder holder,int position,City city);
        //回调方法将对象 holder、位置 position,以及数据 city 传出去
    }
    @NonNull
    @Override
    public CityViewHolder onCreateViewHolder(@NonNull ViewGroup parent,
                                             int viewType){
        //创建一行视图时调用,即将布局文件通过 LayoutInflater 转为对应的视图
        //再通过 CityViewHolder 包装单元视图,找出相关 UI,作为 Holder 的成员变量
        View v = LayoutInflater.from(context)
                               .inflate(R.layout.row_view, parent, false);
        CityViewHolder holder = new CityViewHolder(v);
        createViewHolderCounter++;
        Log.d("onCreateViewHolder",
              "createViewHolderCounter=" + createViewHolderCounter);
```

```
40          //Log.d()打印 createViewHolderCounter 的值,观察该方法被调用的次数
41          return holder;
42      }
43      @Override
44      public void onBindViewHolder(@NonNull CityViewHolder holder, int position) {
45          //若没有对应的行视图,先调用 onCreateViewHolder()生成行视图,返回 holder
46          //若已有可重复利用的行视图,则直接返回 holder
47          //onCreateViewHolder()的调用次数比 onBindViewHolder()要少很多
48          bindViewHolderCounter++;
49          Log.d("onBindViewHolder",
50                  "bindViewHolderCounter = " + bindViewHolderCounter);
51          //观察 onBindViewHolder()被调用的次数,实际中应删除
52          City city = list.get(position);
53          holder.iv.setImageResource(city.getPicId());
54          holder.tv.setText(city.getName());
55          //实现自定义类 CityViewHolder 的目的是方便从类中取 UI 成员变量进行数据渲染
56          if(onItemClickListener != null){//单击侦听接口
57              holder.rowView.setOnClickListener(new View.OnClickListener() {
58                  @Override
59                  public void onClick(View v) {
60                      onItemClickListener.onItemClick(holder,
61                              holder.getAdapterPosition(),city);
62                      //holder.getAdapterPosition()比 position 更可靠
63                      //将相关数据通过接口方法传递出去
64                      //用户调用接口并实现接口的方法时,可对这些数据进行处理
65                  }
66              });
67          }
68      }
69      public void setOnItemClickListener(OnItemClickListener l) {
70          this.onItemClickListener = l;
71          //用户调用适配器的 setOnItemClickListener(),可实现单元视图单击的个性化处理
72      }
73      @Override
74      public int getItemCount() {
75          return list.size();       //返回数据长度,告知适配器总共有多少条数据
76      }
77      public class CityViewHolder extends RecyclerView.ViewHolder{
78      //先写 CityViewHolder 类,再对 CityRecyclerAdapter 的泛型参数设置 CityViewHolder
79      //写在改写适配器的相关方法 onCreateViewHolder()和 onBindViewHolder()之前
80          ImageView iv;
81          TextView tv;
82          View rowView;
83          public CityViewHolder(@NonNull View itemView) {
84              //构造方法会把行视图 itemView 传入
85              //利用 itemView 的 findViewById()将相关的 UI 绑定到 Holder 的成员变量上
86              super(itemView);
87              iv = itemView.findViewById(R.id.row_view_iv);
88              tv = itemView.findViewById(R.id.row_view_tv);
89              //绑定后,可利用 CityViewHolder 的成员变量(UI)进行数据渲染
90              rowView = itemView;
91              //通过成员变量 rowView 绑定行视图,用于设置单击侦听接口
92          }
93      }
94  }
```

自定义适配器 CityRecyclerAdapter 的实现逻辑如下。

步骤 1：考虑传入的参数，如上下文 Context 和数据源，并构造对应方法。

步骤 2：实现内部类 CityViewHolder，将单元视图中的 UI 找出，成为 CityViewHolder 的成员。

步骤 3：将 CityViewHolder 作为 CityRecyclerAdapter 继承适配器时的泛型参数。

步骤 4：利用开发环境的提示，实现所需要改写的方法，开发环境会自动对所需改写的方法设置相关的传参和返回类型。

步骤 5：根据需要定义自定义接口和回调方法。

自定义适配器改写注意事项如下。

(1) 改写 onCreateViewHolder()方法，利用 LayoutInflater 将单元布局文件生成单元视图，并构造自定义 ViewHolder 对象作为返回参数。

(2) 改写 onBindViewHolder()方法，利用 ViewHolder 对象和位置信息，从数据源中取出单元数据渲染到 ViewHolder 对象的 UI 成员上，根据需要，对单元视图或者个别视图做单击事件侦听，在侦听中调用自定义接口的方法，实现事件触发和数据回传。

(3) 改写 getItemCount()方法，返回数据长度。

4. 实现 MainActivity

MainActivity 的实现如代码 3-33 所示。RecyclerView 的使用除了设置适配器以外，还需要通过 setLayoutManager()方法设置布局管理器。常用的布局管理器有如下 3 个。

(1) LinearLayoutManager，线性布局管理器，可通过第 2 参数设置垂直方向或水平方向，第 3 参数用于控制布局方向是否为反向顺序，在水平方向时，若第 3 参数为 true 则布局方向从右到左，若第 3 参数为 false 则布局方向从左到右。

(2) GridLayoutManager，网格布局管理器，第 2 参数用于控制列数。

(3) StaggeredGridLayoutManager，瀑布流布局管理器，在瀑布流布局中，每个单元格的内容长度或宽度随内容调整，允许不一样的尺寸。

RecyclerView 的单击事件通过自定义适配器所提供的接口实现，本任务中采用匿名写法，使其使用方式与 ListView 和 GridView 所提供的原生方法类似。RecyclerView 支持布局动态调整，可通过 RecyclerView 提供的布局管理器，在不更改适配器的前提下，直接实现布局的动态调整。与之相比较，3.8 节项目则通过 ListView 和 GridView 之间的切换实现布局更改，切换的视图属于两个不同的 UI，需要动态加载 UI 以及重新生成相关适配器，不仅涉及的代码多，而且在效率方面也比 RecyclerView 低很多。

代码 3-33　MainActivity.java

```
1   import androidx.appcompat.app.AppCompatActivity;
2   import androidx.recyclerview.widget.GridLayoutManager;
3   import androidx.recyclerview.widget.LinearLayoutManager;
4   import androidx.recyclerview.widget.RecyclerView;
5   import androidx.recyclerview.widget.StaggeredGridLayoutManager;
6   import android.os.Bundle;
7   import android.widget.CompoundButton;
8   import android.widget.Switch;
9   import android.widget.Toast;
10  import java.util.ArrayList;
11  public class MainActivity extends AppCompatActivity {
12      @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_main);
        ArrayList<City> list = new ArrayList<>();
        Switch sw1 = findViewById(R.id.switch1);
        for (int i = 0; i < 4; i++) {
            list.add(new City("杭州",R.drawable.hangzhou));
            list.add(new City("宁波",R.drawable.ningbo));
            list.add(new City("温州",R.drawable.wenzhou));
        }
        RecyclerView recyclerView = findViewById(R.id.recyclerView);
        RecyclerView.LayoutManager layoutManager = new LinearLayoutManager(this,
                                    LinearLayoutManager.HORIZONTAL,false);
        recyclerView.setLayoutManager(layoutManager);
        CityRecyclerAdapter adapter = new CityRecyclerAdapter(this,list);
        recyclerView.setAdapter(adapter);
        adapter.setOnItemClickListener(
                                new CityRecyclerAdapter.OnItemClickListener() {
          //RecyclerView 不支持单击事件,须在自定义适配器中通过接口实现
          @Override
          public void onItemClick(CityRecyclerAdapter.CityViewHolder holder,
                                                    int position, City city) {
              Toast.makeText(MainActivity.this,city.getName(),
                                                Toast.LENGTH_SHORT).show();
          }
        });
        sw1.setOnCheckedChangeListener(
                                new CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView,
                                        boolean isChecked) {
                RecyclerView.LayoutManager layoutManager;
                if(isChecked){
                    buttonView.setText("水平列表");
                    layoutManager = new LinearLayoutManager(MainActivity.this,
                                LinearLayoutManager.HORIZONTAL,false);
                }else {
                    buttonView.setText("网格布局");
                    layoutManager = new GridLayoutManager(MainActivity.this,3);
                    //网格视图为 3 列
                    //layoutManager = new StaggeredGridLayoutManager(2,
                    //                    StaggeredGridLayoutManager.HORIZONTAL);
                    //设置 2 行的水平瀑布流布局
                }
                recyclerView.setLayoutManager(layoutManager);
          }
        });
    }
}
```

RecyclerView 的使用,不管是适配器还是 RecyclerView 本身,虽然所涉及的代码更多,同时也提高了灵活性,并且在效率方面也优于 ListView 和 GridView。本书只给了 RecyclerView 最基本的使用方法,若要获得更多的使用和控制方法,可参考 CSDN、GitHub 的一些开源项目以及 Android 的官方开发者文档。例如,使用 addItemDecoration()方法添

加分割线,使用 setSpanSizeLookup()方法设置不同行列数等,以及动画特效、拖拽排序和滑动菜单等功能。

3.10 本章综合作业

编写一个应用,布局根节点为垂直的 LinearLayout,在布局中依次有 1 个 TextView,用于显示个人信息;1 个 TextView(id 为 tv_result),用于显示城市和景点名称;1 个 ImageView(id 为 iv),用于显示景点图片;1 个 Spinner,内容是城市名称,城市不少于 3 个;1 个 ListView,用于显示 Spinner 所选择的城市的景点列表,景点不少于 4 个。

ListView 采用自定义适配器,适配器自定义视图的左边为景点的图片,右边为景点的名称。

Spinner 下拉选项改变后,ListView 内容同步更新为 Spinner 所选城市的景点,并且 tv_result 和 iv 也同步更新,tv_result 显示所选城市名称和 ListView 首行景点名称,iv 显示 ListView 首行景点图片。

ListView 响应列表项单击事件,tv_result 和 iv 更新为 ListView 所单击的景点数据,tv_result 显示的景点名称包含城市名称,而 ListView 景点名称中则不包含城市名称。