第3章 Android应用程序结构


视频讲解

作为一个移动设备的开发平台，Android 软件层次结构包含操作系统(OS)、中间件(MiddleWare)和应用程序(Application)。其中，Android 的应用程序通常涉及用户界面和用户交互，这类程序是用户实实在在能感受到的。目前 Android 本身提供了桌面、联系人、电话和浏览器等众多的核心应用，同时还允许开发者使用应用程序框架层的 API 实现自己的程序。

3.1 应用程序基本组成

Android 系统没有使用常见的应用程序入口点的方法(例如 main()方法)，应用程序是由组件组成的，组件可以调用相互独立的基本功能模块，根据完成的功能不同，Android 划分了 4 类核心组件，即 Activity、Service、BroadcastReceiver 和 ContentProvider，各组件之间的消息传递通过 Intent 完成。

3.1.1 Activity

Activity 是 Android 应用程序核心组件中最基本的一种，是用户和应用程序交互的窗口。在 Android 应用程序中，一个 Activity 通常对应一个单独的视图。一个 Android 应用程序是由一个或多个 Activity 组成的，这些 Activity 相当于 Web 应用程序中的网页，用于显示信息，并且相互之间可以进行跳转。和网页跳转不同的是，Activity 之间的跳转可以有返回值。

当新打开一个视图时，之前的那个视图会被置为暂停状态，并且压入历史堆栈中，用户可以通过回退操作返回以前打开过的视图。Activity 是由 Android 系统进行维护的，它有自己的生命周期，即“产生、运行、销毁”，但是在这个过程中会调用许多方法，如创建 onCreate()、激活 onStart()、恢复 onResume()、暂停 onPause()、停止 onStop()、销毁 onDestroy()和重启 onRestart()等。

3.1.2 Service

Service 是一种类似于 Activity 但是没有视图的程序，它没有用户界面，可以在后台运行很长时间，相当于操作系统中的一个服务。Android 定义了两种类型的 Service，即本地 Service 和远程 Service。本地 Service 是只能由承载该 Service 的应用程序访问的组件，而远

程 Service 是供在设备上运行的其他应用程序远程访问的 Service。

3.1.3 BroadcastReceiver

BroadcastReceiver 的意思是“广播接收者”，顾名思义，它用来接收来自系统和其他应用程序的广播，并做出回应。在 Android 系统中，当有特定事件发生时就会产生相应的广播。广播体现在方方面面，例如，当开机过程完成后，系统会产生一条广播，接收到这条广播就能实现开机启动服务的功能；当网络状态改变时，系统会产生一条广播，接收到这条广播就能及时地做出提示和保存数据等操作；当电池电量改变时，系统会产生一条广播，接收到这条广播就能在电量低时告知用户及时保存进度等。这些广播可以被 BroadcastReceiver 接收到，进而进行处理。

3.1.4 ContentProvider

文件、数据库等数据在 Android 系统内是私有的，仅允许被特定应用程序直接使用。在两个程序之间，数据的交换或共享由 ContentProvider 实现。

ContentProvider 类实现了一组标准方法的接口，从而能够让其他的应用保存或读取 ContentProvider 提供的各种数据类型。

3.1.5 Intent

Intent 并不是 Android 应用程序四大核心组件之一，但是其重要性无可替代，因此在这里我们简单介绍。

Android 应用程序核心组件中的三大核心组件——Activity、Service、BroadcastReceiver，通过消息机制被启动激活，而所使用的消息就是 Intent。Intent 是对即将要进行的操作的抽象描述，承担了 Android 应用程序三大核心组件相互之间的通信功能。

3.2 Activity

Activity 是 Android 组件中最基本也是最为常见的组件。Activity 是用户接口程序，原则上它会提供给用户一个交互式的接口功能，几乎所有的 Activity 都要和用户打交道，也有人把它比喻成 Android 的管理员。需要在屏幕上显示什么、用户在屏幕上做什么、处理用户的不同操作等都由 Activity 来管理和调度。

Activity 提供用户与 Android 系统交互的接口，用户通过 Activity 来完成自己的目的，例如打电话、拍照、发送 E-mail、查看地图等。每个 Activity 都提供一个用户界面窗口，一般情况下，该界面窗口会填满整个屏幕，但是也可以比屏幕小，或者浮在其他的窗口之上。

一个 Android 应用程序通常由多个 Activity 组成，但是其中只有一个为主 Activity，其作用相当于 Java 应用程序中的 main 函数，当应用程序启动时，作为应用程序的入口首先呈现给用户。Android 应用程序中的多个 Activity 可以直接相互调用以完成不同工作。当新的 Activity 被启动的时候，之前的 Activity 会停止，但是不会被销毁，而是被压入“后退栈”(Back Stack)的栈顶，新启动的 Activity 获得焦点，显示给用户。“后退栈”遵循“后入先出”的原则。当新启动的 Activity 被使用完毕，用户单击 Back 按钮时，当前的 Activity 会被销

毁，原先的 Activity 则会从“后退栈”的栈顶弹出并且被激活。

当 Activity 状态发生改变时，都会通过状态回调函数通知 Android 系统。程序编写人员可以通过这些回调函数对 Activity 进行进一步的控制。

下面对 Activity 生命周期及其涉及的回调函数进行简单介绍。

3.2.1 Activity 的生命周期

从本质上讲，Activity 在生命周期中共存在 3 个状态，分别如下。

(1) 运行态。运行态指 Activity 运行于屏幕的最上层并且获得了用户焦点。

(2) 暂停态。暂停态是指当前 Activity 依然存在，但是没有获得用户焦点。在其之上有其他的 Activity 处于运行态，但是由于处于运行态的 Activity 没有遮挡住整个屏幕，当前 Activity 有一部分视图可以被用户看见。处于暂停态的 Activity 保留了自己所使用的内存和用户信息，但是在系统极度缺乏资源的情况下，有可能会被杀死以释放资源。

(3) 停止态。停止态是指当前 Activity 完全被处于运行态的 Activity 遮挡住，其用户界面完全不能被用户看见。处于停止态的 Activity 依然存活，也保留了自己所使用的内存和用户信息，但是一旦系统缺乏资源，停止态的 Activity 就会被杀死以释放资源。

Activity 在生命周期中从一种状态到另一种状态时会激发相应的回调方法，这些回调方法包括以下几种。

(1) onCreate(Bundle savedInstanceState)。创建 Activity 时调用。设置在该方法中，还以 Bundle 的形式提供对以前存储的任何状态的访问。其中参数 savedInstanceState 对象是用于保存 Activity 的对象的状态。

(2) onStart()。Activity 变为在屏幕上对用户可见时调用。

(3) onResume()。Activity 开始与用户交互时调用(无论是启动还是重启一个活动，该方法总是被调用)。

(4) onPause()。当 Android 系统要激活其他 Activity 时，该方法被调用，在暂停或收回 CPU 和其他资源时被调用。

(5) onStop()。Activity 被停止并转为不可见阶段时调用。

(6) onRestart()。重新启动已经停止的 Activity 时调用。

(7) onDestroy()。Activity 被完全从系统内存中移除时调用，该方法被调用可能是因为有人直接调用 finish()方法或者系统决定停止该活动以释放资源。

上面 7 个生命周期方法分别在 4 个阶段按照一定的顺序进行调用，这 4 个阶段如下。

(1) 启动 Activity。在这个阶段依次执行 3 个生命周期方法：onCreate、onStart 和 onResume。

(2) Activity 失去焦点。如果在 Activity 获得焦点的情况下进入其他的 Activity 或应用程序，这时当前的 Activity 会失去焦点。在这一阶段，会依次执行 onPause 和 onStop 方法。

(3) Activity 重获焦点。如果 Activity 重新获得焦点，会依次执行 3 个生命周期方法：onRestart、onStart 和 onResume。

(4) 关闭 Activity。当 Activity 被关闭时，系统会依次执行 3 个生命周期方法：onPause、onStop 和 onDestroy。

Activity 生命周期中方法的调用过程如图 3.1 所示。

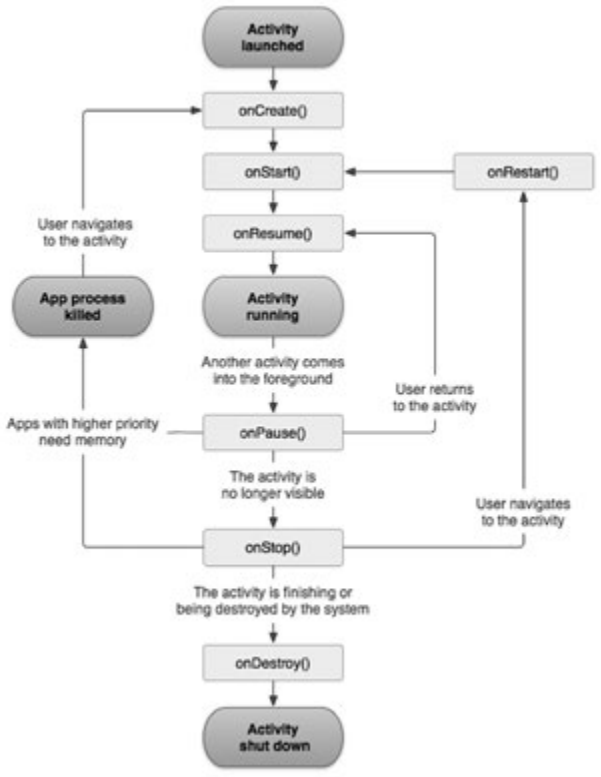

图 3.1　Activity 生命周期

通过图 3.1，可以很直观地了解到 Activity 的整个生命周期。Activity 的生命周期表现在 3 个层面，如图 3.2 所示。

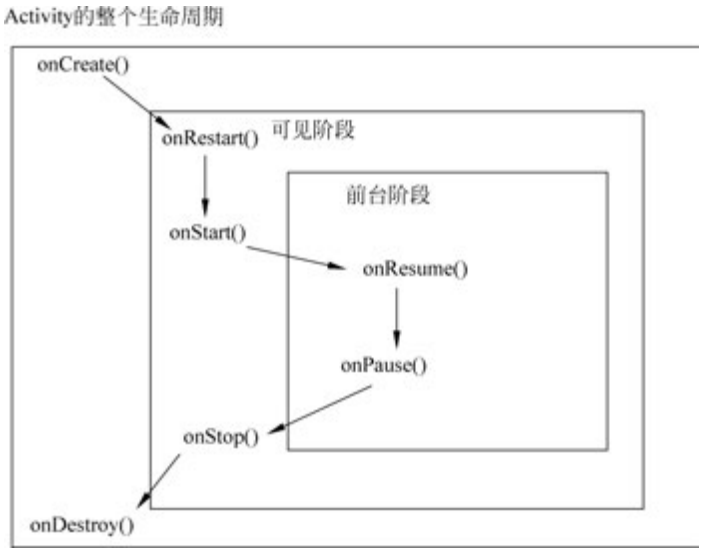

图 3.2　Activity 的整个生命周期

通过图 3.2 可以更清楚地了解 Activity 的运行机制。如果 Activity 离开可见阶段，长时间失去焦点，就很可能被系统销毁以释放资源。当然，即使该 Activity 被销毁掉，用户对该 Activity 所做的更改也会被保存在 Bundle 对象中，当用户需要重新显示该 Activity 时，Android 系统会根据之前保存的用户更改信息将该 Activity 重建。

3.2.2 Activity 的创建

在一个 Android 工程中，创建 Activity 的步骤如下。

步骤 01 新建类。创建一个 Activity，必须创建 Android.app.Activity(或者它的一个已经存在的子类)的一个子类，并重写 onCreate()方法。

步骤 02 关联布局 XML 文件。在新建的 Activity 中设置其布局方式，需要在 res/layout 目录中新建一个 XML 布局文件，可以通过 setContentView()来指定 Activity 的用户界面的布局文件。

步骤 03 注册。在 AndroidManifest.xml 文件中对建立的 Activity 进行注册，即在<application>标签下添加<activity>标签。例如，注册 ExampleActivity 的代码如下：

```
<application ...>
    <activity android:name = ".ExampleActivity" android:exported = "true"/>
     ...
</application ...>
```

其中“android:name = ".ExampleActivity"”指明当前 Activity 的名字，而“android:exported="true"”表示是否允许该 Activity 被外部应用访问。如果该属性为 false，那该 Activity 只能在程序内部被访问。

对于主 Activity，要为其添加<intent-filter>标签，代码如下：

```
<activity Android:name = ".ExampleActivity" Android:icon = "@drawable/app_icon"
android:exported = "true" >
  <intent - filter >
      <action Android:name = "Android.intent.action.MAIN" />
      <category Android:name = "Android.intent.category.LAUNCHER" />
  </intent - filter >
</activity >
```

其中，<action Android:name="Android.intent.action.MAIN" />表示该 Activity 作为主 Activity 出现；<category Android:name="Android.intent.category.LAUNCHER" />表示该 Activity 会被显示在最上层的启动列表中。

3.2.3 启动 Activity

在 Android 系统中，除了主 Activity 由系统启动外，其他 Activity 都要由应用程序来启动。

(1) 通常情况下，通过 startActivity()方法来启动 Activity，要启动的 Activity 的信息则由 Intent 对象来传递，例如：

```
Intent intent = new Intent(this, AnotherActivity.class);
startActivity(intent);
```

表示通过当前的 Activity 启动名为 AnotherActivity 的 Activity。

有时，用户不需要知道要启动的 Activity 的名字，可以仅制订要完成的行为，由 Android 系统来为用户挑选合适的 Activity，例如：

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

其中，Intent.EXTRA_EMAIL 放置的是 recipientArray 中存储的要发送的 E-mail 的目标地址。该 Intent 对象被 startActivity()启动后，Android 系统会启动相应的 E-mail 处理应用程序，并将 Intent.EXTRA_EMAIL 中的内容放置到邮件的目标地址中。

(2) 当需要从启动的 Activity 获取返回值时，需要使用 startActivityForResult()方法代替 startActivity()方法，并运用 onActivityResult()方法来获取返回值。

例如，在发送短信的时候，用户需要从联系人列表中获取联系人的信息，然后返回到短信发送界面，代码如下：

```
Intent intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);
startActivityForResult(intent, PICK_CONTACT_REQUEST);
```

当用户选择了联系人后，相关信息会被存储到 Intent 对象中，并返回到 onActivityResult()方法中。

3.2.4 关闭 Activity

关闭 Activity 使用 finish()方法。关闭之前启动的其他 Activity 可以使用 finishActivity()方法。

需要注意的是，虽然 Android SDK 提供了关闭 Activity 的方法，但是通常情况下，程序员不应该使用这些方法去强制关闭 Activity。因为 Android 系统在为用户维护 Activity 的生命周期，并且提供了完备的资源回收机制和资源重建机制，可以动态地回收和重建 Activity，因此 Activity 应用交由 Android 系统来管理，除非已确定用户不再需要当前的 Activity，并且不允许用户回退到当前 Activity。

3.2.5 Activity 数据传递

Activity 数据传递共有 3 种：

- 通过 Intent 传递一些简单的数据。
- 通过 Bundle 传递相对复杂的数据或者对象。
- 通过 startActivityForResult 可以更方便地进行来回传递，当然前两种方法也可以来回传递。

假设由 Activity1 向 Activity2 传递数据，利用 3 种方式实现的实例代码如下。

1) 利用 Intent 传递数据

(1) 在传递数据的 Activity1 中。

```
Intent intent = new Intent(Activity1.this,Activity2.class);
intent.putExtra("author","leebo");  //在 Intent 中加入键值对数据，键为"author"，值为"leebo"
Activity1.this.startActivity(intent);
```

(2) 在取出数据的 Activity2 中。

```
Intent intent = getIntent();                    //获得传过来的 Intent
String value = intent.getStringExtra("author");
//根据键名 author 取出对应键值为"leebo"
```

2) 利用 Bundle 传递数据

(1) 在传递数据的 Activity1 中。

```
Intent intent = new Intent(Activity1.this,Activity2.class);
Bundle myBundle = new Bundle();
myBundle.putString("author","leebo");
intent.putExtras(myBundle);
Activity1.this.startActivity(intent);
```

(2) 在取出数据的 Activity2 中。

```
Intent intent = getIntent();
Bundle myBundle = intent.getExtras();
String value = myBundle.getString("author");       //根据键名 author 取出对应键值为"leebo"
```

3) 利用 startActivityForResult()传递数据

startActivityForResult()方法不但可以把数据从 Activity1 传递给 Activity2,还可以把数据从 Activity2 传回给 Activity1。

(1) 在 Activity1 中。

```
final int REQUEST_CODE = 1;
Intent intent = new Intent(Activity1.this,Activity2.class);
Bundle mybundle = new Bundle();
mybundle.putString("author", "leebo");                    //把数据传过去
intent.putExtras(mybundle);
startActivityForResult(intent, REQUEST_CODE);
```

重载 onActivityResult 方法,用来接收传过来的数据(接收 b 中传过来的数据):

```
protected void onActivityResult(int requestCode, int resultCode,Intent intent){
    if(requestCode == this.REQUEST_CODE){
  switch(resultCode){
  case RESULT_OK:
    Bundle b = intent.getExtras();
    String str = b.getString("Result"); //获取 Result 中的值,为"from Activity2"
     break;
  default:
      break;
 }
 }
}
```

(2)在 Activity2 中。

```
Intent intent = getIntent();
Bundle myBundle = getIntent().getExtras();
```

```
String author = getBundle.getString("author");
Intent intent = new Intent();
Bundle bundle = new Bundle();
bundle.putString("Result","from Activity2");
intent.putExtras(bundle);
Activity02.this.setResult(RESULT_OK,intent);        //通过 intent 将数据返回给 Activity1,
                                                    //RESULT_ OK 是结果码
finish();                                           //结束当前的 Activity
```

本质上，这 3 种数据传递方式都是通过 Intent 来完成的。

【实例 3-1】 DataTransDemo 演示了第三种传递方式，运行效果如图 3.3 所示。

图 3.3 DataTransDemo 运行效果

在 DataTransDemoActivity 的布局中单击“发送数据”按键，会发送数据给 Activity2。单击“发回数据”按键，Activity2 会发送数据给 DataTransDemoActivity。

DataTransDemoActivity 的布局文件 main.xml 代码如下：

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

    < TextView
        android:id = "@ + id/textView1"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_marginStart = "86dp"
        android:layout_marginTop = "93dp"
        android:text = "TextView"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent" />

    < Button
        android:id = "@ + id/button1"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_marginStart = "94dp"
        android:layout_marginTop = "190dp"
        android:text = "发送数据"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent" />
</androidx.constraintlayout.widget.ConstraintLayout >
```

DataTransDemoActivity.java 代码如下：

```
package android.introduction.datatransdemo;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class DataTransDemoActivity extends Activity {
   /** Called when the activity is first created. */
    final int REQUEST_CODE = 1;
    TextView tv;
    Button btn;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      tv = (TextView) this.findViewById(R.id.textView1);
      btn = (Button)this.findViewById(R.id.button1);
      btn.setOnClickListener(new OnClickListener(){

            @Override
            public void onClick(View v) {
                  // TODO Auto-generated method stub
                  Intent intent = new Intent(DataTransDemoActivity.this,Activity2.class);

              intent.putExtra("version", "16");
              Bundle myBundle = new Bundle();
              myBundle.putString("bookname", "android");
              intent.putExtras(myBundle);
                 startActivityForResult(intent, REQUEST_CODE);
           }

      });

   }
   @Override
   protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        // TODO Auto-generated method stub
        super.onActivityResult(requestCode, resultCode, data);
        if(requestCode == this.REQUEST_CODE){
            switch(resultCode){
            case RESULT_OK:
                String str = data.getStringExtra("Activity2");
                tv.setText("string from activity2: " + str);
                break;
            default:
                break;
            }
```

```
        }
    }
}
```

Activity2 的布局文件 layout2.xml 代码如下：

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

    < TextView
        android:id = "@ + id/textView"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_marginStart = "114dp"
        android:layout_marginTop = "91dp"
        android:text = "TextView"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent" />

    < Button
        android:id = "@ + id/button"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:layout_marginStart = "103dp"
        android:layout_marginTop = "189dp"
        android:text = "发回数据"
        app:layout_constraintStart_toStartOf = "parent"
        app:layout_constraintTop_toTopOf = "parent" />
</androidx.constraintlayout.widget.ConstraintLayout >
```

Activity2.java 代码如下：

```
package android.introduction.datatransdemo;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class Activity2 extends Activity {
    private TextView tv;
    Button btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto - generated method stub
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout2);
        tv = this.findViewById(R.id.textView);
        btn = findViewById(R.id.button);
        Intent intent = Activity2.this.getIntent();
        Bundle myBundle = intent.getExtras();
        String bookname =  myBundle.getString("bookname");
        String author = intent.getStringExtra("version");
        StringBuilder str = new StringBuilder("string received: " + bookname);
        str.append(author);
        tv.setText(str);

      btn.setOnClickListener(new OnClickListener(){

                @Override
                public void onClick(View v) {
                    Intent intent = new Intent();
                    intent.putExtra("Activity2", "activity2");
                    Activity2.this.setResult(RESULT_OK, intent);
                    finish();
                }
            });

    }
}
```

3.3 AppCompatActivity

在 Android Studio 中创建 Activity 会默认继承自 AppCompatActivity。本节将进行简要说明。

Activity 是 Android 系统提供的一个基础类，用于创建用户界面的活动。Activity 来自 android.app 包，它是 Android 应用开发中最基本的组件之一，提供了活动的管理生命周期、用户界面显示等基本功能。

AppCompatActivity 是 Activity 类的一个间接子类，属于 Android 扩展库（AndroidX），来自 androidx.appcompat.app 包。AppCompatActivity 提供了与 Activity 相同的基本功能，但通过 AndroidX 扩展了更多功能，例如支持 Material Design 主题和样式，内置对 ActionBar 的支持等。

总而言之，AppCompatActivity 提供了比 Activity 更多的功能，并且具有更好的兼容性。但是在本书中，因为讲授的主要内容会集中在 Activity 上，代码中的 Activity 都可以替换为 AppCompatActivity，而代码正常运行。

需要说明的是，由于 AppCompatActivity 对于主题样式 Theme 有要求，在某些版本的 Android Studio 中创建的工程中，由于默认生成的主题样式和 AppCompatActivity 不匹配，会出现“java.lang.IllegalStateException：You need to use a Theme.AppCompat theme(or descendant)with this activity.”错误提示。在这种情况下，只要在 AndroidManif.xml 文件中的< application >标签中，将 android:theme 属性的值修改为 AppCompatActivity 兼容的

主题即可，例如“@style/Theme. Design. Light. NoActionBar”。

3.4 资源

在 Android 层次结构中，资源扮演着重要的角色。Android 支持字符串、位图以及其他很多种类型的资源。每种资源的语法、格式以及存放的位置都会根据其类型的不同而不同。一般来讲，共有 3 种类型的资源文件：XML 文件、位图文件（图像）和 RAW 文件（声音等）。

Android 工程目录中，用于存放资源文件的文件夹有两个，分别为 res 和 assets。其中，res 文件夹不支持深度子目录，其中的资源最终将被打包到编译后的 Java 文件中，可以直接通过 R 资源类访问，利用率较高；而 assets 中存放的资源是用于打包到应用程序中的静态文件，这些文件不会被编译，最终会直接部署到目标设备中，可以使用任意深度的子目录进行存储。assets 文件夹中的文件不能直接通过 R 资源类读取，只能使用流的形式读取，其利用率相对较低。

Android 的资源编译器 AAPT（Android Asset Packaging Tool）会依照资源所在的子目录及其格式对其进行编译。

3.5 Manifest 文件

每个 Android 项目都包含一个清单（Manifest）文件 AndroidManifest. xml，它是 XML 格式的 Android 程序声明文件，包含 Android 系统运行程序前所必须掌握的重要信息，这些信息包含应用程序名称、图标、包名称、模块组成、授权和 SDK 最低版本等，而且每个 Android 程序必须在根目录下包含一个 AndroidManifest. xml。

例如，Manifest 文件可以使用如下代码声明一个 Activity：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< manifest ... >
   < application android:icon = "@drawable/app_icon.png" ... >
      < activity android:name = "com.example.project.ExampleActivity"
               android:label = "@string/example_label" ... >
      </activity >
      ...
   </application >
</manifest >
```

AndroidManifest. xml 中可包含的所有标签元素如以下代码所示，其中除了< manifest >和< application >标签是必需的，其他所有标签都可根据情况添加。

```
<?xml version = "1.0" encoding = "utf - 8"?>
< manifest >
  < uses - permission />
  < permission />
  < permission - tree />
  < permission - group />
  < instrumentation />
```

```
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
        <activity-alias>
            <intent-filter>...</intent-filter>
            <meta-data />
        </activity-alias>
        <service>
            <intent-filter>...</intent-filter>
            <meta-data/>
        </service>
        <receiver>
            <intent-filter>...</intent-filter>
            <meta-data />
        </receiver>
        <provider>
            <grant-uri-permission />
            <meta-data />
        </provider>
        <uses-library />
    </application>
</manifest>
```

在此，仅对几种常见的标签进行简单介绍。

1）manifest 标签

manifest 标签是 AndroidManifest.xml 文件的根标签，该标签用于设置与项目相关的一些属性，如用于唯一标识应用程序的 package 属性，用于记录应用程序版本的 Android:versionName 属性，等等。其中的 xmlns:Android 属性必须被定义为 http://schemas.Android.com/apk/res/Android。

2）application 标签

manifest 标签仅能包含一个 application 标签，它使用各种属性来指定应用程序的各种元数据（包括标题、图标和主题）。它还可以作为一个包含活动（Activity）、服务（Service）、内容提供器（Provider）和广播接收器（BroadcastReceiver）标签的容器，用来指定应用程序组件。

（1）activity 标签。应用程序显示的每一个 Activity 都要求有一个 activity 标签，并使用 Android:name 属性来指定类的名称。这必须包含核心的启动 Activity 和其他所有可以

显示的屏幕或者对话框。启动任何一个没有在清单中定义的 Activity 时都会抛出一个运行时异常。每个 Activity 节点都允许使用 intent-filter 子标签来指定哪个 Intent 启动该活动。

（2）service 标签。和 activity 标签一样，应用程序中使用的每一个 Service 类都要创建一个新的 service 标签（service 标签也支持使用 intent-filter 子标签来允许后面的运行时绑定）。

（3）provider 标签。provider 标签用来说明应用程序中的每个内容提供器。内容提供器是用来管理数据库访问以及程序内和程序间共享的。

（4）receiver 标签。通过添加 receiver 标签，可以注册一个广播接收器，而不用事先启动应用程序。广播接收器就像全局事件监听器一样，一旦注册了之后，无论何时，只要与它相匹配的 Intent 被应用程序广播出来，它就会立即执行。通过在声明中注册一个广播接收器，可以使这个进程实现完全自动化。如果一个匹配的 Intent 被广播了，应用程序就会自动启动，并且注册的广播接收器也会开始运行。

3）uses-permission 标签

作为安全模型的一部分，uses-permission 标签声明了那些自己定义的权限，而这些权限是应用程序正常执行所必需的。在安装程序时，设定的所有权限将会告诉用户，由他们来决定同意与否。对很多本地 Android 服务来说，权限都是必需的，特别是那些需要付费或者有安全问题的服务（例如拨号、接收 SMS 或者使用基于位置的服务）。第三方应用程序，包括你自己的应用程序，也可以在对共享的程序组件进行访问之前指定权限。

4）permission 标签

在可以限制访问某个应用程序组件之前，需要在清单中定义一个 permission。可以使用 permission 标签来创建这些权限定义。然后，应用程序组件就可以通过添加 Android:permission 属性来要求这些权限。其他的应用程序需要在它们的清单中包含 uses-permission 标签（并且通过授权），之后才能使用这些受保护的组件。在 permission 标签内，可以详细指定允许的访问权限的级别（normal、dangerous、signature 和 signatureOrSystem）、一个 label 属性和一个外部资源，这个外部资源应该包含对授予这种权限的风险的描述。

5）instrumentation 标签

instrumentation 类提供一个框架，用来在应用程序运行时在活动或者服务上运行测试。它们提供了一些方法来监控应用程序及其与系统资源的交互。对于为自己的应用程序所创建的每一个测试类，都需要创建一个新的节点。

习题 3