第 3 章　Android 的 Activity 组件

Activity(活动)组件是 Android 的四大组件之一,是 Android 移动应用的基本组件,用于设计移动应用界面的屏幕显示。Android 移动应用往往需要多个界面,因此移动应用中必须定义一个或多个 Activity 以实现界面的交互。在 Android 3.0 以后的版本中出现的 Fragment(碎片)可以嵌入 Activity 的图形用户界面中。目前,采用单个 Activity 和多个 Fragment 结合的方式已成为移动应用界面定义的首选。本章将对 Activity 组件以及嵌入 Activity 组件的 Fragment 进行介绍。

3.1　Activity 的创建

Activity 用于创建移动界面的屏幕。它的主要作用就是实现移动界面和用户之间的交互。Activity 类用于创建和管理用户界面,一个应用程序可以有多个 Activity,但在同一个时间内只有一个 Activity 处于激活状态。在移动应用中,Activity 之间的依赖关系低。

创建 Activity 时,需要完成如下几个步骤。

(1) 定义 Activity 类。

(2) 定义 Activity 对应的布局文件,使得布局文件成为 Activity 的界面定义。

(3) 在配置清单文件 AndroidManifest.xml 中声明并配置 Activity 的相应属性。

1. 定义 Activity 类

Activity 必须是自己的子类或扩展子类的子类。因为 Activity 的一个子类为 AppCompatActivity,它可以定义 Material Design 风格的界面,所以在当前 Android 中创建 Activity 一般是定义为 AppComaptActivity 的子类。形式如下:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

在 MainActivity(主活动)中定义了 onCreate()函数,该函数是必须定义的回调函数,它会在系统创建 Activity 时被调用。在 Activity 中必须使用 setContentView()函数来指定 Activity 的界面布局。这个布局通过资源引用 R.layout.activity_main 来引用资源目录下的 res/layout/activity_main.xml 的布局文件。布局文件(如 activity_main.xml)用于定义 Activity 的外观,Activity 可以显示并对布局文件的界面组件进行控制,实现交互行为,如图 3-1 所示。

2. 创建 Activity 的布局文件

布局配置文件保存在 res/layout 目录下,是 XML 文件。通过配置布局的相关属性和

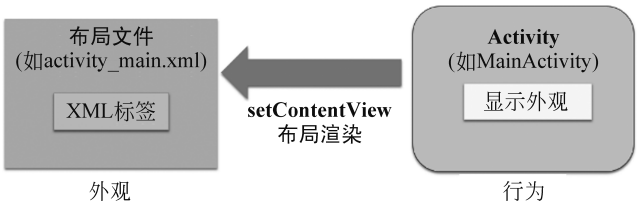


图 3-1　Activity 和布局之间的关系示意

元素可实现布局。布局文件中所有的元素必须配置 android:layout_width 和 android: layout_height 属性,它们分别表示元素在屏幕的宽度和高度,一般取值如下。

(1) match_parent: 表示使视图扩展至父元素大小。

(2) wrap_content: 表示自适应大小,将组件元素的全部内容进行显示。

当然,也可以设置尺寸大小。Activity 类通过 setContentView()来引用布局资源,实现布局在 Activity 中的渲染和展示,代码如下:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView android: id="@+id/textView"
    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: text="Hello World!"
    android: layout_width="match_parent"
    android: layout_height="match_parent"
    tools: context=".MainActivity" />
```

在上述文件中,根元素是 TextView,是一个文本标签。如果希望在 Activity 中对整个 TextView 可以进行引用和处理,可以在布局文件中使用 android: id="@+id/textView" 来标识资源。其中,"@+id"表示增加一个资源 id。如果希望在布局文件中引用这个 TextView,可以通过"@id/textView"实现。但是在 Activity(例如 MainActivity 代码)中需要通过这个 TextView 的资源 id 来引用,形式如下:

```
val textView: TextView=findViewById(R.id.textView)
```

如果在模块的构建文件 build.gradle 中增加插件 kotlin-android-extensions,形如:

```
apply plugin: 'kotlin-android-extensions'
```

则可以直接在 Activity 中通过资源编号引用 View(视图)组件:

```
textView.text="Hello"          //直接设置布局资源编号为 textView 的文本为 Hello
```

但是由于 kotlin-android-extensions 插件只能对 Kotlin 支持,目前已经被弃用。

3. 在配置清单文件 AndroidManifest.xml 中注册 Activity

要让创建的 Activity 能让移动应用识别和调用,就必须在 AndroidManifest.xml 系统配置清单文件中进行声明和配置。AndroidManifest.xml 文件保存在项目的 manifests 目

录中。代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns: android="http://schemas.android.com/apk/res/android"
    package="chenyi.book.android.ch03_01">
    <application android: allowBackup="true"
        android: icon="@mipmap/ic_launcher"
        android: label="@string/app_name"
        android: roundIcon="@mipmap/ic_launcher_round"
        android: supportsRtl="true"
        android: theme="@style/Theme.Chapter03">
        <activity android: name=".MainActivity"android: exported="true">
        <intent-filter>
        <action android: name="android.intent.action.MAIN" />
        <category android: name="android.intent.category.LAUNCHER" />
        </intent-filter>
       </activity>
   </application>
</manifest>
```

在 application 元素(表示移动应用)中增加下级元素 activity,并通过 android: name 属性指定活动的类名。在 manifest 元素的 package 属性中指定了包名,所以在 activity 元素的 android: name 中只需要定义成

```
android: name=".MainActivity"
```

即可。在 activity 元素下定义了 intent-filter 意图过滤器。intent-filter 定义的意图过滤器非常强大,通过它可以根据显式请求启动 Activity,也可以根据隐式请求启动 Activity。下面代码用于指定 MainActivity(主活动),它是整个移动应用的入口活动,表示应用程序可以显示在程序列表中。

```
action android: name="android.intent.action.MAIN"
category android: name="android.intent.category.LAUNCHER"
```

注意,在 Android 12 及以上版本中使用 intent-filter 元素配置时,必须时 Activity 增加配置 android:exported 属性。

3.2 Activity 和 Intent

一个应用可以定义多个 Activity,这些 Activity 可以相互切换和跳转。要实现这些交互功能,就必须了解什么是 Intent(意图)。Intent 表示封装执行操作的意图,是应用程序启动其他组件的 Intent 对象启动组件。这些组件可以是 Activity,也可以是其他基本组件,如 Service(服务)组件、BroadcastReceiver(广播接收器)组件。从一个基本组件导航到另一个组件。通过 Intent 实现组件之间的跳转和关联。Intent 分为显式 Intent 和隐式 Intent 两种。

3.2.1 显式 Intent

显式 Intent 就是在 Intent()函数启动组件时,需要明确指定的激活组件名称,例如：

```
Intent(Context packageContext, Class<?>c)
```

其中，Intent()函数包含两个参数，packageContext 用于提供启动 Activity 的上下文，c 用于指定想要启动的目标组件类。

例 3-1 显式 Intent 的应用。在本应用中定义 3 个 Activity 类：MainActivity、FirstActivity 和 SecondActivity。其中，MainActivity 用于定义菜单，具体的菜单项包括“第一个活动”“第二个活动”“退出”。根据菜单项，可跳转至不同的活动或退出应用。选中“第一个活动”菜单项，可跳转到 FirstActivity，选中“第二个活动”菜单项，可跳转到 SecondActivity，选中“退出”菜单项，则从移动应用中退出。

在新建移动应用模块时，因为本例涉及菜单的应用，所以需要先定义相关的资源文件。

(1) 定义相关资源。

移动应用往往需要大量的字符串来配置相关的组件，因此可以将字符串统一定义在 res/values 目录下的 strings.xml 配置文件中，通过设置 string 元素配置字符串。string 元素的 name 属性表示字符串的名字，string 元素的内容表示字符串的取值，代码如下：

```
<!--模块 Ch03_02 定义字符串资源文件 res/values/strings.xml-->
<resources>
    <string name="app_name">Ch03_02</string>
    <string name="title_first_activity">第一个活动</string>
    <string name="title_second_activity">第二个活动</string>
    <string name="title_exit_app">退出</string>
</resources>
```

在 Activity 中创建菜单时，必须依赖菜单资源。在 res 目录下创建资源目录 menu，然后在 res/menu 目录下创建 menu.xml 文件，可以直接利用编辑器在 menu.xml 中编辑菜单的内容，也可以通过编辑器的 Design 界面来实现。在这个例题中，res/menu 目录下的 menu.xml 的内容如下：

```
<!--模块 Ch03_02 定义菜单资源文件 res/menu/menu.xml-->
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns: android="http://schemas.android.com/apk/res/android">
    <item android: id="@+id/firstItem" android: icon=
    "@android: drawable/arrow_up_float"
    android: title="@string/title_first_activity"/>
<item android: id="@+id/secondItem"
   android: icon="@android: drawable/arrow_down_float"
   android: title="@string/title_second_activity" />
<item android: id="@+id/exitItem"
   android: icon="@android: drawable/ic_lock_power_off"
   android: title="@string/title_exit_app" />
</menu>
```

在 menu.xml 文件中，menu 元素定义菜单，item 元素定义菜单项。在菜单项中使用 android：id 指定菜单的 id 资源编号，android：icon 表示菜单的图标，android：title 定义菜单对应的文字标签。

如果在移动应用中需要使用具体尺寸大小的相关资源，可以采用类似的方法创建dimens.xml来保存尺寸资源。具体做法是，选中res/values目录并右击，在弹出的快捷菜单中选中New|Values Resource File选项，然后指定资源名，如dimens，单击OK按钮，创建dimens.xml的尺寸大小的资源文件，代码如下：

```
<!--模块 Ch03_02 定义尺寸资源文件 res/values/dimens.xml -->
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="size_text">40sp</dimen>
</resources>
```

指定的尺寸采用了sp单位，表示比例无关的像素单位，常常表示字体的大小。表示布局的尺寸往往使用dp单位，dp单位代表密度无关的像素，是基于屏幕的物理密度的抽象或虚拟单元。

(2) 定义MainActivity。MainActivity是当前应用的入口。它对应的布局文件activity_main.xml文件如下：

```
<!--模块 Ch03_02 主活动对应的布局文件 activity_main.xml -->
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns: android="http://schemas.android.com/apk/res/android"
    xmlns: tools="http://schemas.android.com/tools"
    android: layout_width="wrap_content"
    android: layout_height="wrap_content"
    android: layout_gravity="center"
    android: text="Hello World!"
    tools: context=".MainActivity"/>
```

在MainActivity中不但利用activity_main.xml渲染生成显示的界面，而且定义菜单，并根据菜单实现不同活动的跳转。代码如下：

```
//模块 Ch03_02 主活动的定义文件 MainActivity.kt
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
    override fun onCreateOptionsMenu(menu: Menu?): Boolean {     //定义菜单
        menuInflater.inflate(R.menu.menu,menu)
                                    //引用 res/menu 目录的 menu.xml 生成菜单对象 menu
        return true
    }
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
                                    //根据选中的菜单项进行处理
        when(item.itemId){
            R.id.firstItem->turnTo(FirstActivity: : class.java)
            R.id.secondItem->turnTo(SecondActivity: : class.java)
            R.id.exitItem->{
```

```
                AlertDialog.Builder(this).apply{          //定义对话框
                    title =resources.getString(R.string.app_name)
                    setMessage("退出应用?")
                    setPositiveButton("确定") { _, _->
                        exitProcess(0)
                    }
                    setNegativeButton("取消",null)
                }.create().show()
            }
        }
        return super.onOptionsItemSelected(item)
    }
    private fun <T>turnTo(c: Class<T>){                   //通过显式意图跳转
        val intent =Intent(MainActivity@this,c)
        startActivity(intent)
    }
}
```

在 MainActivity 中定义两个回调函数 onCreateOptionsMenu()和 onOptionsItemSelected()分别用于创建菜单和菜单项选择的处理。

onCreateOptionsMenu()函数采用 menuInflater.inflate(R.menu.menu,menu)函数引用菜单资源 res/menu/menu.xml 文件渲染菜单对象 menu。

onOptionsItemSelected()函数对菜单项的 id 值进行判断并处理,并调用了通用的显式 Intent 跳转的 turnTo()函数,使得从当前的 MainActivity 跳转到指定类的 Activity 中。在退出菜单项的处理中增加了 AlertDialog 对话框的显示,如果选中“确定”动作,则执行 exitProcess(0)退出整个移动应用。注意,在 setPositiveButton 处理部分使用了两个“_”表示匿名的参数。如果在 Lambda 表达式中传递的参数没有使用,可以用“_”来替换。

turnTo()函数是自定义的函数,在这个函数中使用了泛型,即定义了类型变量 T,将类型变量传递给 Class<T>中,表示各种 Class 类型。在函数体中,通过创建显式 Intent 对象,并启动 startActivity()函数实现从 MainActivity 跳转到指定类的 Activity 中。如果调用的是 turnTo(FirstActivity::class.java)函数,则等价执行的代码如下:

```
//创建从当前的 MainActivity 跳转到 FirstActivity 活动的意图对象
val intent =Intent(MainActivity@this, FirstActivity: : class.java)
//在当前 Activity 中启动 Intent
MainActivity@this.startActivity(intent)
```

其中,MainActivity@this 表示 MainActivity 的当前对象。

(3) 定义其他的 Activity。移动模块中还定义了 FirstActivity 和 SecondActivity,它们的布局文件分别为 activity_first.xml 和 activity_second.xml,均保存在 res/layout 目录下。它们都定义了一个文本标签中显示字符串。布局文件类似 activity_main.xml 布局文件的定义。这两个 Activity 的任务就是引用各自的布局文件显示界面内容。

```
//模块 Ch03_02 第一个活动的定义文件 FirstActivity.kt
class FirstActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_first)
    }
}
//模块 Ch03_02 第二个活动的定义文件 SecondActivity.kt
class SecondActivity: AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_second)
    }
}
```

(4) 配置清单文件中注册 Activity,代码如下:

```
<!--模块 Ch03_02 配置清单文件 AndroidManifest.xml -->
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns: android="http://schemas.android.com/apk/res/android"
    package="chenyi.book.android.ch03_02">
    <application .......>    <!--其他配置省略 -->
        <!--配置 SecondActivity -->
        <activity android: name=".SecondActivity"
            android: label="@string/title_second_activity"/>
        <!--配置 FirstActivity -->
        <activity android: name=".FirstActivity"
            android: label="@string/title_first_activity"/>
        <!--配置 MainActivity -->
        <activity android: name=".MainActivity"
           android: exported="true">
           <intent-filter>
        <action android: name="android.intent.action.MAIN" />
        <category android: name="android.intent.category.LAUNCHER" />
        </intent-filter>
        </activity>
    </application>
</manifest>
```

在配置清单文件中,配置了 FirstActivity、SecondActivity 和 MainActivity。其中,MainActivity 作为入口,在程序加载时首先启动和显示,如图 3-2 所示。

运行结果中菜单项对应的图标在下列菜单列表并没有显示。有一个编程技巧,可以将所有的菜单项向下降一个等级。将 res/menu 目录下的 menu.xml 修改成如下形式:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns: android="http://schemas.android.com/apk/res/android">
    <item android: id="@+id/menu" android: title="Menu">
        <menu>
            <item android: id="@+id/firstItem"
            android: icon="@android: drawable/arrow_up_float"
```

图 3-2　菜单处理的运行结果

```
            android: title="@string/title_first_activity" />
        <item android: id="@+id/secondItem"
            android: icon="@android: drawable/arrow_down_float"
            android: title="@string/title_second_activity" />
        <item android: id="@+id/exitItem"
            android: icon="@android: drawable/ic_lock_power_off"
            android: title="@string/title_exit_app" />
    </menu>
</item>
</menu>
```

这时，可以出现显示二级菜单，菜单的图标可以显示。

3.2.2　隐式 Intent

隐式 Intent 没有明确地指定要启动哪个 Activity，而是通过 Android 系统分析 Intent，并根据分析结果来启动对应的 Activity。执行过程如图 3-3 所示，即某个 Activity A 通过 startActivity()函数操作某个隐式 Intent 调用另外的 Activity。Android 系统先分析应用配置清单 AndroidManifest.xml 中声明的 Intent Filter 内容，再搜索应用中与 Intent 相匹配的 Intent Filter 内容，若匹配成功后，则 Android 系统调用匹配的 Activity，用 Activity B 的 onCreate()函数启动新的 Activity，实现从 Activity A 跳转到 Activity B。

一般情况下，隐式 Intent 需要在配置清单文件 AndroidManifest.xml 中指定 Intent Filter 的 action、category 和 data 3 个属性。Android 系统会对 Intent Filter 的 3 个属性进行分析和匹配，以搜索对应 Activity 或其他组件。

(1) action：表示该 Intent 所要完成的一个抽象的 Activity。

(2) category：用于为 action 增加额外的附加类别信息。

(3) data：向 action 提供操作的数据。

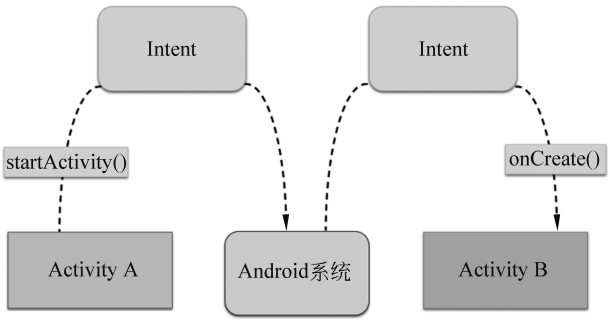


图 3-3　通过隐式 Intent 通过 Android 系统启动活动

但是这 3 个属性并不是必须全部配置，可以根据具体的需要进行组合配置。例如：

```
<activity android: name=".CustomedActivity" android: exported="true">
    <intent-filter>
        <action android: name="chenyi.book.android.ch03_03.ACTION" />
    <!--自定义动作的名称-->
        <category android: name="android.intent.category.DEFAULT" />
    <!--指定默认类别-->
        <category android: name="chenyi.book.android.ch03_03.MyCategory" />
    <!--指定自定义的类别-->
    </intent-filter>
</activity>
```

例 3-2　使用隐式 Intent 的应用实例。MainActivity 可以分别调用自定义的 Activity 和拨打电话。

在本应用定义两个活动：一个是自定义的 CustomedActivity；另一个表示 MainActivity，用于移动应用的入口。在 MainActivity 中提供了两个按钮，单击后都可以实现调用 CustomedActivity 和拨打电话的功能。在系统的配置清单文件中，首先配置并注册这些 Activity。

(1) 在配置清单 AndroidManifest.xml 中注册 Activity，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns: android="http://schemas.android.com/apk/res/android"
    package="chenyi.book.android.ch03_03">
    <application.....>        <!--其他配置省略 -->
        <activity android: name=".CustomedActivity"
        android: exported="true"><!--注册 CustomedActivity -->
          <intent-filter>
              <action android: name="chenyi.book.android.ch03_03.ACTION" />
              <category android: name="android.intent.category.DEFAULT" />
              <category android: name="chenyi.book.android.ch03_03.MyCategory" />
          </intent-filter>
       </activity>
       <activity android: name=".MainActivity"
          android: exported="true"><!--注册 MainActivity -->
          <intent-filter>
```

```
                <action android: name="android.intent.action.MAIN" />
                <category android: name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

(2) 定义 CustomedActivity,代码如下:

```
<!--模块 Ch03_03 定义 CustomedActivity 对应的布局 activity_customed.xml -->
<?xml version="1.0" encoding="utf-8"?>
<ImageView android: id="@+id/imageView"
    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"
    app: srcCompat="@mipmap/ic_launcher"
    tools: context=".CustomedActivity" />
```

在上述的布局中定义了 ImageView 组件,用于显示图片,通过 app: srcCompat 直接引用图片 res/mipmap 目录下的 ic_launcher 图片资源。

```
//模块 Ch03_03 定义 CustomedActivity 的文件 CustomedActivity.kt
class CustomedActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_customed)
    }
}
```

(3) 定义 MainActivity,代码如下:

```
<!--模块 Ch03_03 定义 MainActivity 对应的布局 activity_main.xml -->
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns: android="http://schemas.android.com/apk/res/android"
    xmlns: app="http://schemas.android.com/apk/res-auto"
    xmlns: tools="http://schemas.android.com/tools"
    android: layout_width="match_parent"
    android: layout_height="match_parent"
    tools: context=".MainActivity">
    <Button android: id="@+id/firstBtn"
        android: layout_width="wrap_content"
        android: layout_height="wrap_content"
        android: text="@string/title_customed_activity"
        app: layout_constraintBottom_toBottomOf="parent"
        app: layout_constraintHorizontal_bias="0.544"
        app: layout_constraintLeft_toLeftOf="parent"
        app: layout_constraintRight_toRightOf="parent"
        app: layout_constraintTop_toTopOf="parent"
```

```
            app: layout_constraintVertical_bias="0.345" />
        <Button android: id="@+id/secondBtn"
            android: layout_width="wrap_content"
            android: layout_height="wrap_content"
            android: text="@string/title_action_view"
            app: layout_constraintBottom_toBottomOf="parent"
            app: layout_constraintHorizontal_bias="0.524"
            app: layout_constraintLeft_toLeftOf="parent"
            app: layout_constraintRight_toRightOf="parent"
            app: layout_constraintTop_toTopOf="parent"
            app: layout_constraintVertical_bias="0.461" />
    </androidx.constraintlayout.widget.ConstraintLayout>
```

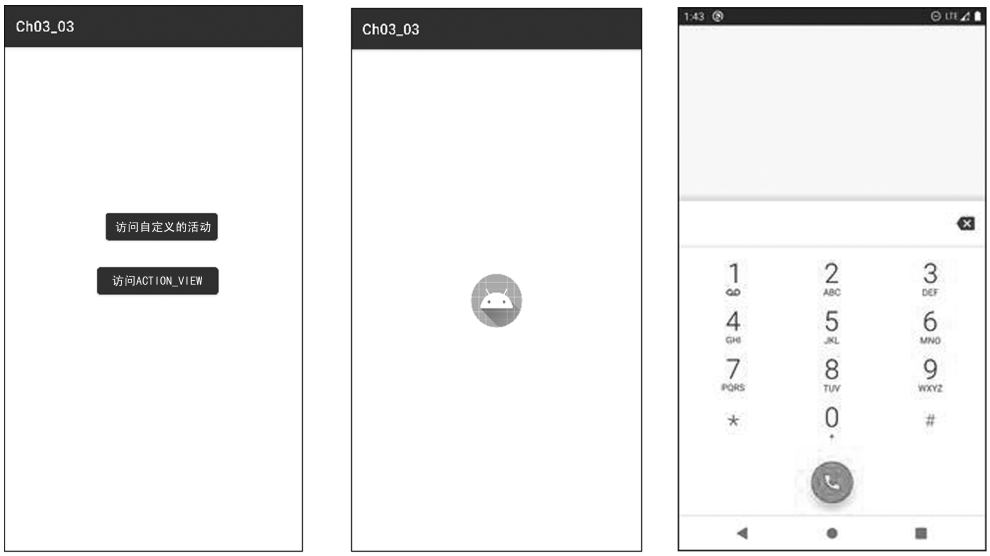
在 MainActivity 布局文件中定义了两个按钮，目的是实现不同 Activity 的跳转的接口。

```
    //模块 Ch03_03 定义 MainActivity 的文件 MainActivity.kt
    class MainActivity : AppCompatActivity() {
        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_main)
            val firstBtn: Button =findViewById(R.id.firstBtn)
            firstBtn.setOnClickListener {
                val intent1 =Intent("chenyi.book.android.ch03_03.ACTION")
                intent1.addCategory("chenyi.book.android.ch03_03.MyCategory")
                MainActivity@this.startActivity(intent1)
            }
            val secondBtn: Button =findViewById(R.id.secondBtn)
            secondBtn.setOnClickListener {
                val intent2 =Intent(Intent.ACTION_DIAL)
                startActivity(intent2)
            }
        }
    }
```

运行结果如图 3-4 所示。

MainActivity 是移动应用中第一个调用和加载的界面，在配置清单文件 AndroidManifest.xml 中也注册了 CustomedActivity，在 activity 元素定义了下级标签 intent-filter，指定了 CustomedActivity 的动作名称，并添加了两个类别：android.intent.category.DEFAULT 和 chenyi.book.android.ch03_03.MyCategory。android.intent.category.DEFAULT 是默认的类别。在调用 startActivity()函数时，Android 系统会自动将 android.intent.category.DEFAULT 添加到 Intent 中。chenyi.book.android.ch03_03.MyCategory 是自定义的类别，因此，需要调用 addCategory()函数将这个自定义的类别添加到 Intent 中，即单击第一个按钮会通过隐式 intent1 跳转到 CustomedActivity。

注意，第二个按钮单击实现拨打电话。Intent.ACTION_DIAL 内置的系统动作，对应 android.intent.action.DIAL。Android 系统中定义常见的内置动作如表 3-1 所示。



(a) MainActivity　　(b) CustomedActivity　　(c) 系统的拨号界面

图 3-4　隐式 Intent 实例的运行结果

表 3-1　Intent 内置的动作

动　作	说　明
ACTION_ANSWER	处理来电
ACTION_CALL	拨打电话，使用 Intent 的号码，需要设置 android.permission.CALL_PHONE
ACTION_DIAL	调用拨打电话的程序，使用 Intent 的号码，没有直接打出
ACTION_EDIT	编辑 Intent 中提供的数据
ACTION_VIEW	查看动作，可以浏览网页、短信、地图路肩规划，根据 Intent 的数据类型和数据值决定
ACTION_SENDTO	发送短信、电子邮件
ACTION_SEND	发送信息、电子邮件
ACTION_SEARCH	搜索，通过 Intent 的数据类型和数据判断搜索的动作

调用 Intent 的 setData()函数可以设置 Intent 的数据，用 Kotlin 的表达是 intent.data；调用 Intent 的 setType()函数可以设置 Intent 数据的类型；调用 setDataAndType()函数可以来设置数据和数据的类型。

3.3　Activity 之间的数据传递

Activity 之间往往需要传递数据。数据的传递可以借助 Intent 来实现。Intent 提供了两种实现数据传递的方式：一种是通过 Intent 的 putExtra 传递数据；另一种是通过 Bundle 传递多类型数据。

3.3.1 传递常见数据

在这里常见的数据类型包括 Int(整型)、Short(短整型)、Long(长整型)、Float(单精度实型)、Double(双精度实型)、数组、ArrayList(数组列表)等多种类型。在 Activity 之间,可以将这些类型的数据进行传递。

首先,Intent 提供了一系列的 putExtra()函数实现数据传递。在数据的发送方,通过 putExtra()函数种指定键值对,然后启动 startActivity 将意图从当前的 Activity 跳转到 intent 指定的活动,代码如下:

```
val intent =Intent(this,OtherActivity: : class.java)
                                                    //定义跳转到 OtherActivity 的 Intent
intent.putExtra("intValue",23)                      //配置传递数据的键值对
startActivity(intent)                               //根据 Intent 启动 Activity
```

数据接收方(设为 OtherActivity)调用 getIntent()函数获得启动该 Activity 的 Intent,代码如下:

```
val intent=getIntent()
```

在 Kotlin 中也可以直接表示成

```
val intent=intent
```

然后,根据数据的关键字,调用接收数据类型对应 get×××Extra()函数来获得对应的取值。例如:

```
val received=intent.getIntExtra("intValue")     //根据关键字 intValue 获取对应的 Int 数据
```

第二种方式结合 Bundle 数据包和 putExtra()函数实现数据的传递。具体的执行过程与第一种方式类似。数据的发送方创建 Bundle 数据包,代码如下:

```
val bundle: Bundle=Bundle()
```

然后,调用 Bundle 对象的对应的 put×××函数,设置不同类型的数据,例如:

```
bundle.putInt("bundleIntValue",1000)
bundle.putString("bundleStringValue","来自 MainActivity 的问候!")
```

在发送方通过调用 Intent 的 putExtra()函数,设置要传递的数据,再调用 startActivity 实现 Activity 的跳转和数据的发送,代码如下:

```
val intent=Intent(this,OtherActivity: : class.java)
intent.putExtra(bundle)
startActivity(intent)
```

在数据的接收方,例如 OtherActivity 有两种方式接收数据:一种是通过 Bundle 对象的对应的 get×××()函数来获得;另一种是直接通过调用启动该 Activity 的 Intent 的 get

×××Extra()函数,根据关键字获得对应的取值,代码如下:

```
val intent =intent
val bundle: Bundle? =intent.extras
//根据 Bundle 来获得数据
val intValue =bundle?.getInt("bundleIntValue",0)    //0 为默认值,取值失败时会赋值
val strValue =bundle?.getString ("bundleStringValue")
//或根据 Intent 直接获得数据
val intValue =intent.getIntExtra("bundleIntValue")
val strValue =intent.getStringExtra("bundleStringValue")
```

例 3-3 常见数据类型数据的传递的应用实例 MainActivity 分别使用两种不同的方式传递数据到其他两个 Activity,代码如下:

```
<!--模块 Ch03_04 定义 MainActivity 的布局文件 activity_main.xml -->
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns: android="http://schemas.android.com/apk/res/android"
    xmlns: app="http://schemas.android.com/apk/res-auto"
    xmlns: tools="http://schemas.android.com/tools"
    android: layout_width="match_parent"
    android: layout_height="match_parent"
    tools: context=".MainActivity">
    <Button  android: id="@+id/firstBtn"
        android: layout_width="wrap_content"
        android: layout_height="wrap_content"
        android: text="@string/title_send_int"
        app: layout_constraintBottom_toBottomOf="parent"
        app: layout_constraintEnd_toEndOf="parent"
        app: layout_constraintLeft_toLeftOf="parent"
        app: layout_constraintRight_toRightOf="parent"
        app: layout_constraintStart_toStartOf="parent"
        app: layout_constraintTop_toTopOf="parent"
        app: layout_constraintVertical_bias="0.327" />
    <Button  android: id="@+id/secondBtn"
        android: layout_width="wrap_content"
        android: layout_height="wrap_content"
        android: text="@string/title_send_array"
        app: layout_constraintBottom_toBottomOf="parent"
        app: layout_constraintEnd_toEndOf="parent"
        app: layout_constraintLeft_toLeftOf="parent"
        app: layout_constraintRight_toRightOf="parent"
        app: layout_constraintStart_toStartOf="parent"
        app: layout_constraintTop_toTopOf="parent"
        app: layout_constraintVertical_bias="0.453" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

在 MainActivity 的布局文件定义了两个按钮,并分别为这两个按钮定义了相应的处理动作,代码如下:

```
//模块 Ch03_04 定义数据的发送方的主活动 MainActivity.kt
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val firstBtn: Button =findViewById(R.id.firstBtn)
        val secondBtn: Button =findViewById(R.id.secondBtn)
        firstBtn.setOnClickListener {
            val intent =Intent(MainActivity@this,FirstActivity::class.java)
            intent.putExtra("intValue",999)
            intent.putExtra("strValue","来自 MainActivity 的问候")
            val strArr =arrayOf("Kotlin","Java","Scala")
            intent.putExtra("arrValue",strArr)
            startActivity(intent)
        }
        secondBtn.setOnClickListener {
            val bundle =Bundle()
            bundle.putInt("bundleIntValue",1000)
            bundle.putString("bundleStringValue","来自 MainActivity 的问候!!")
            val intArr =IntArray(10)
            for(i in intArr.indices)
                intArr[i] =i+1
            bundle.putIntArray("bundleIntArray",intArr)
            val intent =Intent(MainActivity@this,SecondActivity::class.java)
            intent.putExtras(bundle)
            startActivity(intent)
        }
    }
}
```

作为数据的发送方,MainActivity 提供了两种数据发送的方法。第一种方法是直接利用 Intent 的 put×××Extra()函数将数据传递出去;第二种方法是将数据通过键值对放置在 Bundle 对象中,然后利用 Intent 将 Bundle 数据包整体传递出去。有数据发送,必须有一个接收方来接收数据,否则发送数据没有任何意义。在此处,定义了 FirstActivity 和 SecondActivity 分别接收数据,代码如下:

```
//模块 Ch03_04 定义数据接收方 FirstActivity.kt
class FirstActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_first)
        val intent =intent
        val receivedInt =intent.getIntExtra("intValue",0) //获得整数,0 为默认值
        val receivedStr =intent.getStringExtra("strValue")     //获得字符串
        val receivedStrArr =intent.getStringArrayExtra("arrValue") //获得字符串数组
        val result =StringBuilder()
```

```
        result.append("接收: ${receivedInt}\n${receivedStr}\n${Arrays.
          toString(receivedStrArr)}")
        Toast.makeText(this,"$result",Toast.LENGTH_LONG).show()
    }
}
```

FristActivity 中接收的数据是利用 Intent 调用 get×××Extra()函数,通过关键字来获得对应的数据。在代码中,接收了字符串数组,调用 java.util.Arrays 的 toString()函数,将数组转换成字符串的表达形式。另外,在代码中使用了 Toast 信息提示框。通过调用 Toast.makeText()函数可以将文本在信息提示中显示出来,代码如下:

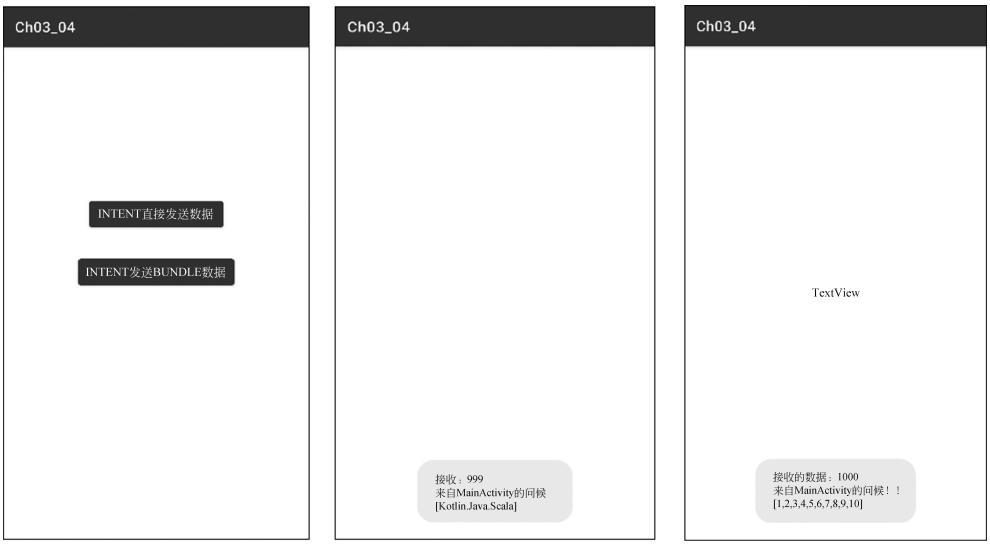
```
//模块 Ch03_04 定义数据接收方 SecondActivity.kt
class SecondActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_second)
        val intent =intent                                    //获得发送数据的意图
        val bundle: Bundle? =intent.extras                    //获得 Bundle 对象
        val receivedIntValue =bundle?.getInt("bundleIntValue")
                                                              //接收整数数据
        val receivedStrValue =intent.getStringExtra("bundleStringValue")
                                                              //接收字符串数据
        val receivedIntArr =bundle?.getIntArray("bundleIntArray")
                                                              //接收整型数组数据
        val result=StringBuilder()
    result.append("接收的数据: ${receivedIntValue}\n${receivedStrValue}\n"
        ${Arrays.toString(receivedIntArr)}")
    Toast.makeText(this,"$result",Toast.LENGTH_LONG).show()     //显示数据
    }
}
```

SecondActivity 接收数据的方式有两种:一种方式是启动该活动的意图获得 Bundle 对象,然后从 Bundle 对象中获得关键字对应的数据;另一种方式是通过调用 Intent 的 get×××Extra()函数根据传递数据的关键字获得对应的数据,如图 3-5 所示。

3.3.2 Serializable 对象的传递

在 Activity 之间传递数据有一种特殊情况,就是传递自定义类的对象或者自定义类的对象数组或集合类型中包含自定义类的对象。需要对这些自定义类进行可序列化处理。可序列化处理,就是将对象处理成可以传递或存储的状态,使得对象可以从临时瞬间状态保存持久状态,即存储到文件、数据库等中。要在 Java 中实现可序列化,就需要用类实现 java.io.Serializable 接口。作为基于 JVM 的语言,Kotlin 也需要实现 java.io.Serializable 接口,这个类的对象就可以进行序列化的处理。形式如下:

```
class DataType: Serializable{
  …
}
```



(a) 数据的发送方　　(b) 数据的接收方一　　(c) 数据的接收方二

图 3-5　活动传递数据实例的运行结果

通过 Serializable 传递对象的过程，如图 3-6 所示。发送方将数据序列化成字节流发送，在接收方，将字节流重组成对应的对象。

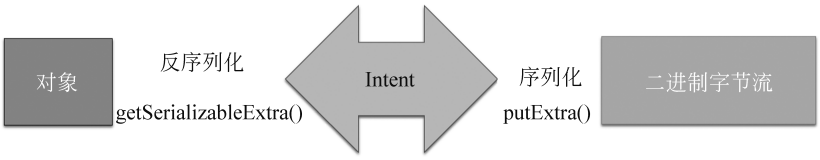


图 3-6　Serializable 方式传递对象的原理

要在 Activity 之间发送 Serializable 对象，仍需要通过 Intent 实现。Intent 通过调用 putExtra()函数，设置“关键字”和对象之间的键值对，形式如下：

```
val intent =Intent(this,OtherActivity…class.java)
val object: Type =…
intent.putExtra("object",object)
startActivity(intent)
```

Activity 的接收方 OtherActivity 在接收数据时，首先启动该活动的意图，再通过意图的 getSerializableExtra()函数通过关键字来获得对应的 Serializable 对象，最后利用 as 操作符转换成对应的对象。实现了根据接收的状态数据反序列化生成对应的对象。形式如下：

```
val intent =intent
val object =intent.getSerializableExtra("object") as Type
```

例 3-4　在 Activity 之间传递 Serializable 对象应用实例。实现将学生的信息(学号、姓名、出生日期)从一个 Activity 传递到另外一个 Activity。

(1) 定义实体类 Student,代码如下:

```
//模块 Ch03_05 定义数据实体类 Student
data class Student(val no: String,val name: String,
    val birthday: LocalDate): Serializable
```

(2) 定义数据的发送方 MainActivity,代码如下:

```
<!--模块 Ch03_05 MainActivity 对应的布局文件 activity_main.xml -->
<?xml version="1.0" encoding="utf-8"?>
<Button android: id="@+id/sendBtn"
    xmlns: android="http://schemas.android.com/apk/res/android"
    xmlns: tools="http://schemas.android.com/tools"
    android: layout_width="wrap_content"
    android: layout_height="wrap_content"
    android: layout_gravity="center"
    android: text="@string/title_send_object"
    android: textSize="30sp"
    tools: context=".MainActivity"/>
```

MainActivity 发送 Student 对象到 FirstActivity。调用 Intent 的 putExtra()函数设置发送数据的键值对。关键字是 student,取值是 Student 对象 student。然后通过 startActivity(intent)启动 Intent 封装活动,实现数据通过 Intent 发送 FirstActivity,代码如下:

```
//模块 Ch03_05 发送方的主活动定义 MainActivity.kt
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val sendBtn: Button =findViewById(R.id.sendBtn)
        sendBtn.setOnClickListener {
            val student: Student=Student("60012322","张三",
            LocalDate.of(2008,2,12))                          //定义对象
            val intent =Intent(this,FirstActivity::class.java)  //定义 Intent
            intent.putExtra("student",student)                 //Intent 设置键值对
            MainActivity@this.startActivity(intent)            //启动 Activity
        }
    }
}
```

(3) 定义数据接收方的 FirstActivity,代码如下:

```
//模块 Ch03_05 接收方定义 FirstActivity.kt
class FirstActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_first)
        val intent =intent
        val receivedStu: Student =intent.getSerializableExtra("student") as Student
                                                          //将类型强制转换成 Student
```

```
        Toast.makeText(this,"${receivedStu.toString()}",Toast.LENGTH_LONG).show()
    }
}
```

接收方的 FirstActivity 用于接收对象。由于 FirstActivity 中 Intent 对象的 getSerializableExtra()函数会返回一个 Serializable 类型的对象,因此需要对接收对象进行处理,即结合 as 操作符执行类型转换。形式如下:

```
val receivedStu: Student =intent.getSerializableExtra("student") as Student
```

通过这种方式获得接收的对象后,整个实例的运行结果如图 3-7 所示。

(a) 对象发送方　　(b) 对象接收方

图 3-7　活动传递 Serializable 对象的运行结果

3.3.3　Parcelable 对象的传递

通过 Java 的 java.io.Serializable 方式传递对象,将对象整体数据序列化处理,将对象转换可传递状态进行发送。接收方将接收的状态数据重新反序列化重新生成对象。由于这种处理方式效率低下,因此 Android 提供了自带的 Parcelable(打包)方式实现对象的传递。

以 Parcelable 方式传递对象的原理是,通过一套机制,将一个完整的对象进行分解,分解后的每一部分数据都属于 Intent 支持的数据类型。可以将分解后的可序列化的数据写入一个共享内存中,其他进程通过 Parcel 从这块共享内存中读出字节流,并反序列化成对象,通过这种方式来实现传递对象的功能,如图 3-8 所示。

要实现 Parcelable 方式传递对象,需要定义的类必须实现 Parcelable 接口。必须对其中的方法进行覆盖重写,具体如下。

(1) writeToParcel():实现将对象的数据进行分解写入。

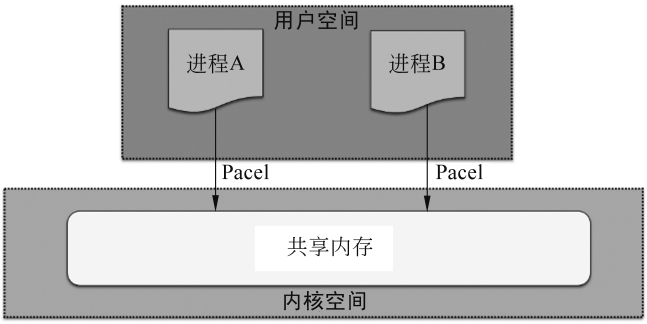


图 3-8　Parcelable 方式传递对象

(2) describeContent()：内容接口的描述，默认只需要返回为 0。

(3) Parcelable.Creator：需要定义静态内部对象实现 Parcelable.Createor 接口的匿名类。该接口提供了两个函数：一个是 createFromParcelable()函数实现从 Parcel 容器中读取传递数据值，读取数据的顺序与写入数据的顺序必须保持一致，然后封装成 Parcel 对象返回逻辑层，另一个是 newArray()函数创建一个指定类型指定长度的数组，供外部类反序列化本类数组使用。下面定义了实现 Parcelable 接口的 Student 类，代码如下：

```
data class Student(val no: String,val name: String,val birthday: LocalDate):
    Parcelable {
    constructor(parcel: Parcel):
        this(parcel.readString()!!,parcel.readString()!!,
        parcel.readSerializable() as LocalDate)
    override fun writeToParcel(parcel: Parcel, flags: Int) {       //分解数据
            parcel.writeString(no)
            parcel.writeString(name)
            parcel.writeSerializable(birthday)
    }
    override fun describeContents(): Int =0
        companion object CREATOR : Parcelable.Creator<Student>{   //重组生成对象
              override fun createFromParcel(parcel: Parcel): Student {
                  return Student(parcel)
        }
        override fun newArray(size: Int): Array<Student?>{
              return arrayOfNulls(size)
        }
    }
}
```

但是，这种表示方式非常复杂，在 Kotlin 中提供了一种更简单的表示方式，即利用@Parcelize 标注实体类。形式如下：

```
@Parcelize
data class Student(val no: String,val name: String,
    val birthday: LocalDate): Parcelable
```