Activity

1. 本章概述

本章讲解 Android 的组件式开发思想、Android 的四大组件、Android 中的 Activity 的基本概念、生命周期函数以及 Activity 之间的传值。

2. 本章重点与难点及学习方法

1）重点

（1）Android 的四大组件。

（2）Activity 的生命周期。

（3）Activity 的基本用法。

（4）Activity 页面之间的传值。

2）难点

（1）Activity 生命周期。

（2）Activity 页面之间的传值。

3. 重难点学习建议

本章将重点学习 Activity 的创建方法、Activity 生命周期函数。理解 Android 程序结构及组件式开发思想，通过反复练习实践掌握 Activity 的用法，同时通过开发环境的 Logcat 观察 Android 生命周期函数的执行时机。

3.1 Android 四大组件

Android 应用开发是组件式开发，所谓组件，可以理解为装修房屋的一个组成元素，一个 Android 应用就像一间房子，房子里的一张桌子、一把椅子、一张床就相当于 Android 的组件，除了这种可以直观上看得到的组件外，还有一些组件负责服务功能，例如，房子中的水管道、煤气管道、电力管道等，这些不是直接呈现在视觉中，但起着很重要的作用，就相当于 Android 的 Service 组件。要实现一个 Android 应用，就可以把一堆接口标准、封装完整的组件拿来用，组件搭配使用，就形成了一间装修好的房子，也就是一个 Android 应用了。Android 的四大组件如下。

Activity：表示一个可视化的用户界面，在应用程序中是一个单独的屏幕。每个屏幕都是通过继承和扩展基类 Activity 实现的。

Service：表示服务，没有可见的用户界面，只提供服务，能够长时间运行于后台，通过继承和扩展基类 Service 来实现。在后台运行于应用程序进程的主线程中，因此 Service 不会阻塞其他组件或者用户界面。

ContentProvider：表示内容提供者，可以将应用程序特定的数据提供给另一个应用程序使用，其数据存储方式可以是 Android 文件系统、SQLite 数据库或者其他方式。

BroadcastReceiver：表示广播接收器，自身并不实现图形用户界面，但是当收到某个广播后，BroadcasetReceiver 可以启动 Activity 作为响应，或者通过 NotificationManager 提醒用户，或者调用 Service 处理长时间事务。

除了以上四大组件外，Intent 也是一个非常重要的组件，它在不同组件之间传递信息，将一个组件的请求意图传给另一个组件，可以实现组件之间调用，还可以通过 Intent 进行组件间传递数据。Android 会根据意图的内容选择适当的组件来调用。

3.2 Activity 的创建

Activity 的中文意思是活动，可以显示由几个 View 组件组成的用户接口，并且可以对事件进行相应的处理。在 Android 中，Activity 代表手机屏幕的一屏，或是平板电脑中的一个窗口。它是 Android 应用程序与用户交互的窗口，几乎每一个 Android 应用程序都离不开 Activity，它就像一个网站的页面一样，每个页面都可以通过一个独立的类来表示，这个独立的类继承于 Activity 这个基类。

创建 Activity，大致可以分为以下两个步骤。

(1) 创建一个 Activity 一般是继承 android.app 包中的 AppCompatActivity 类，不过在不同的应用场景下，也可以继承 Activity 的子类。创建一个继承 AppCompatActivity 的类，名称为 MainActivity 的具体代码如下。

```
import android.app.Activity;
public class MainActivity extends AppCompatctivity {
}
```

(2) 重写需要的回调方法。通常情况下，都需要重写 onCreate()方法，并且在该方法中调用 setContentView()方法设置要显示的视图。例如，在步骤(1)中创建的 Activity 中，重写 onCreate()方法，并且设置要显示的视图的具体代码如下。

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
}
```

创建 Activity 后，还需要在 AndroidManifest.xml 文件中配置该 Activity，具体的配置方法是在＜application＞＜/application＞标记中添加＜activity＞＜/activity＞标记实现。＜activity＞标记的基本格式如下。

```
< activity
    android:icon="@drawable/图标文件名"
    android:name="实现类"
    android:label="说明性文字"
android:theme="要应用的主题"
…
>
…
</activity>
```

(1) 启动 Activity 步骤如下。

① 生成一个意图对象 Intent。

② 调用 setClass 方法设置所要启动的 Activity。

③ 调用 startActivity 方法启动 Activity，其语句如下。

```
public void startActivity (Intent intent)
```

(2) 关闭 Activity，其语句如下。

```
public void finish ()
```

3.3 Activity 的生命周期

在 Activity 的生命周期中，有表 3.1 所示的 4 个重要状态。

表 3.1 Activity 生命周期

状 态	描 述
活动状态	当前 Activity 位于 Activity 栈顶，用户可见，并且可以获得焦点
暂停状态	失去了焦点的 Activity，仍然可见，但是在内存低的情况下，它不能被系统 killed(杀死)
停止状态	该 Activity 被其他 Activity 所覆盖，不可见，但是它仍然保存所有的状态和信息，不过，在内存低的情况下，它可能会被系统 killed(杀死)
销毁状态	该 Activity 结束，或 Activity 所在的 Dalvik 进程被结束

图 3.1 描述了 Activity 从创建到销毁整个生命周期方法的调用过程。

从最初调用 onCreate()到最终调用 onDestroy()称为完整生命周期。Activity 会在 onCreate()中进行所有“全局”状态的设置，在 onDestroy()中释放所有持有的资源。例如，如果它有一个从网络上下载数据的后台线程，那就可以在 onCreate()中创建这个线程

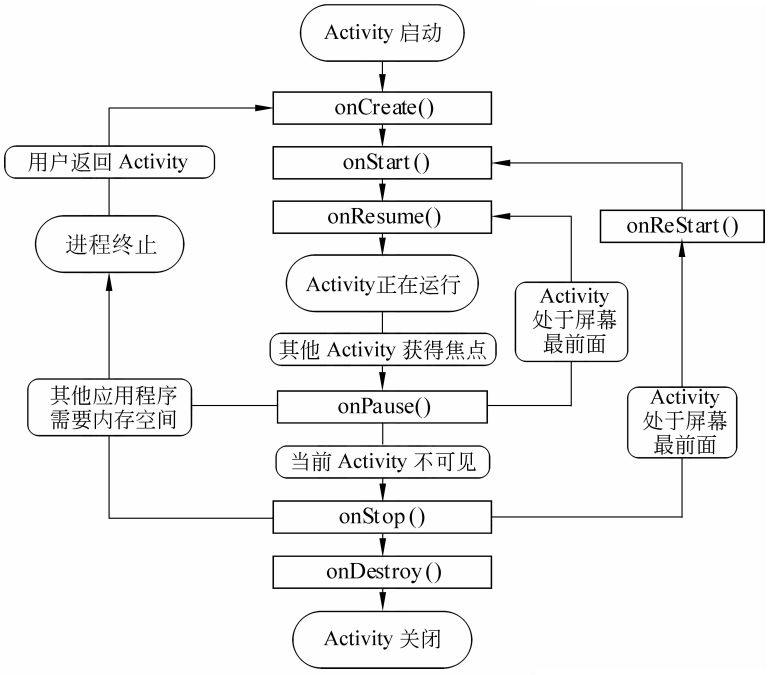

图 3.1　Activity 的生命周期

并在 onDestroy()中停止这个线程。

从 Activity 调用 onStart()开始，到调用对应的 onStop()为止，称为可见生命周期。在这段时间内尽管此 Activity 并不一定是在屏幕的最前方，也不一定可以和用户交互，但是用户可以在屏幕上看到这个 Activity。在这两个方法运行周期之间可以根据需求维护 Activity。例如，可以在 onStart()中注册一个 IntentReceiver(意图接收器)来监控那些可以对 UI 产生影响的环境改变，当 UI 不继续在用户面前显示时，可以在 onStop() 中注销这个 IntentReceiver。

每当 Activity 在用户面前显示或者隐藏时都会调用相应的方法，所以 onStart()和 onStop()方法在整个生命周期中可以多次被调用。

从 Activity 调用 onResume()开始，到调用对应的 onPause()为止称为前景生命周期，这段时间 Activity 处于其他所有 Activity 的前面，且与用户交互。一个 Activity 可以经常在 resumed 和 paused 状态之间转换，例如，手机进入休眠时、Activity 的结果返回时或者新的 Intent 到来时，所以这两个方法中的代码应该非常简短。

总之，所有 Activity 都应该实现自己的 onCreate(Bundle)方法来进行初始化设置，大部分还应该实现 onPause()方法提交数据的修改，并且准备终止与用户的交互。

【例 3.1】 Activity 生命周期回调函数执行时机练习。定义两个 Activity，第一个 Activity 中有个标签，标签的内容为“第一个 Activity”，还有一个 Button，Button 显示的内容为“跳到第二个 Activity”，第二个 Activity 有一个标签，标签内容为“第二个 Activity”，还有一个 Button，Button 显示的内容为“返回第一个 Activity”。在每个 Activity 中添加 Activity 的回调函数，运行观察 Logcat 的输出，理解 Activity 各个回调函

数的执行时机。

本程序需要创建两个 Activity，分别为 MainActivity 和 Main2Activity，创建 Activity 的步骤如下。

(1) 在使用 Android 项目结构时，选择 Java 目录下的包名，右击，在弹出的快捷菜单中选择 New→Activity→Empty Activity，出现图 3.2 所示的对话框。

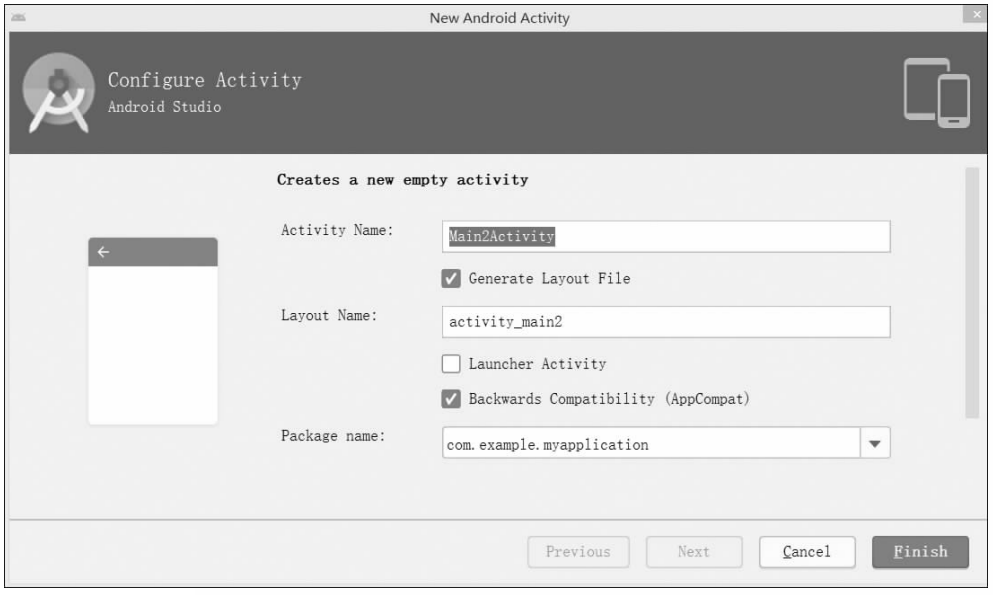

图 3.2 新建 Activity 对话框

(2) 在 MainActivity 类中重写 Activity 的几个生命周期方法，在每个方法的方法体里使用 Logcat 输出如下内容：当前所在 Activity 名：当前所在方法名。例如，在 onRestart 方法中加入如下输出语句：

```
Log.d(TAG,"onCreate: ");
```

在 MainActivity 对应的布局文件上放置一个按钮，设置按钮的 id 为“go”，在后台代码中通过 findViewById 方法找到该按钮，设置按钮的单击事件，Android 事件处理模型和 Java 类似，所有实现 OnClickListener 接口的类，都可作为按钮事件的监听器，设置监听器的方法为 setOnClickListener，每当单击按钮时，都会自动回到 OnClickListener 接口的 onClick 方法。MainActivity 对应代码如下。

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
```

```
import com.example.jason.demo20181126.R;
public class MainActivity extends AppCompatActivity {
//定义静态变量 TAG,用于 Logcat 输出的标签
    public static final String TAG="MainActivity";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main1);
        Log.d(TAG, "onCreate: ");
        System.out.println();

        Button startActivity=(Button) findViewById(R.id.go);
        //设置启动按钮的监听器,匿名类作监听器
        startActivity.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //使用 Intent 进行跳转
                Intent intent=new Intent(MainActivity.this, Main2Activity.
                class);
                startActivity(intent);
            }
        });

    }
   //重写 onStart 方法
    @Override
    protected void onStart() {
        super.onStart();
        Log.d(TAG, "onStart");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.d(TAG, "onResume");
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.d(TAG, "onPause");
    }
```

```
    @Override
    protected void onStop() {
        super.onStop();
        Log.d(TAG, "onStop");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy");
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d(TAG, "onRestart");
    }

}
```

MainActivity 对应的布局文件如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/start_normal_activity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="第一个 Activity" />
    <Button
        android:id="@+id/go"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="跳到第二个 Activityy" />
</LinearLayout>
```

同样地，在 Main2Activity 中，重写 Main2Activity 类中的几个生命周期方法，在每个方法的方法体里输出如下内容：当前所在 Activity 名：当前所在方法名。同时设置按钮监听器方法，单击按钮时返回，结束当前 Activity 即是返回，返回语句为：

```
finish();
```

Main2Activity 对应的代码如下。

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import com.example.jason.demo20181126.R;
public class Main2Activity extends AppCompatActivity {
  public static final String TAG="Main2Activity";
  private Button btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main2);
        btn=findViewById(R.id.back);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();
            }
        });
    }

    @Override
    protected void onStart() {
        super.onStart();
        Log.d(TAG, "onStart: ");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.d(TAG, "onResume: ");
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.d(TAG, "onPause: ");
    }

    @Override
    protected void onStop() {
```

```
        super.onStop();
        Log.d(TAG, "onStop: ");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy: ");
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d(TAG, "onRestart: ");
    }

}
```

MainActivity 对应的布局文件 activity_main 代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="第二个 Activity"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="返回第一个 Activity"
        android:id="@+id/back"/>

</LinearLayout>
```

第一个 Activity 运行效果，如图 3.3 所示。
第二个 Activity 运行效果，如图 3.4 所示。
使用 Logcat 观察输出结果，如图 3.5 所示。

图 3.3　第一个 Activity 运行效果

图 3.4　第二个 Activity 运行效果

```
2019-03-25 10:50:27.217 6678-6678/com.example.jason.demo20181126 D/MainActivity: onCreate:
2019-03-25 10:50:27.218 6678-6678/com.example.jason.demo20181126 D/MainActivity: onStart
2019-03-25 10:50:27.220 6678-6678/com.example.jason.demo20181126 D/MainActivity: onResume
2019-03-25 10:50:27.946 1622-1642/? I/ActivityManager: Displayed com.example.jason.demo20181126/.lifecycle.MainActivity: +874ms
2019-03-25 10:50:29.510 6678-6678/com.example.jason.demo20181126 D/MainActivity: onPause
2019-03-25 10:50:30.027 6678-6678/com.example.jason.demo20181126 D/MainActivity: onStop
2019-03-25 10:50:30.037 1622-2726/? I/WindowManager: Destroying surface Surface(name=com.example.jason.demo20181126/com.example.jason.demo20181126.lifecycle.MainActivity) ca
2019-03-25 10:50:31.404 6678-6678/com.example.jason.demo20181126 D/MainActivity: onRestart
2019-03-25 10:50:31.404 6678-6678/com.example.jason.demo20181126 D/MainActivity: onStart
2019-03-25 10:50:31.405 6678-6678/com.example.jason.demo20181126 D/MainActivity: onResume
2019-03-25 11:46:07.176 6678-6678/com.example.jason.demo20181126 D/MainActivity: onPause
2019-03-25 11:46:07.674 6678-6678/com.example.jason.demo20181126 D/MainActivity: onStop
2019-03-25 11:50:20.192 7611-7611/com.example.jason.demo20181126 D/Main2Activity: onStart:
2019-03-25 11:50:20.193 7611-7611/com.example.jason.demo20181126 D/Main2Activity: onResume:
2019-03-25 11:50:20.276 1622-1642/? I/ActivityManager: Displayed com.example.jason.demo201811
2019-03-25 11:50:23.029 7611-7611/com.example.jason.demo20181126 D/Main2Activity: onPause:
2019-03-25 11:50:23.474 7611-7611/com.example.jason.demo20181126 D/Main2Activity: onStop:
2019-03-25 11:50:23.474 7611-7611/com.example.jason.demo20181126 D/Main2Activity: onDestroy:
```

图 3.5　Logcat 查看 MainActivity 生命周期函数执行顺序

3.4　Activity 间的信使 Intent

Intent 的中文意思是“意图，意向”，Intent 机制在 Android 中用来协助应用间的交互与通信，Intent 负责对应用中一次操作的动作、动作涉及的数据和附加数据进行描述，Android 则根据此 Intent 的描述，负责找到对应的组件，将 Intent 传递给调用的组件，并完成组件的调用。Intent 不仅可用于应用程序之间，也可用于应用程序内部的 Activity/Service 之间的交互。因此，可以将 Intent 理解为不同组件之间通信的“媒介”专门提供组件互相调用的相关信息。所以将 Intent 翻译为“信使”。

3.4.1 显式调用和隐式调用

1. 显式调用

直接调用 Activity 的 Class 类，例如 MainActivity 调用 NewActivity。

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

2. 隐式调用

顾名思义，隐式调用就是在不明确设置激活对象的前提下寻找最匹配的组件，隐式不明确指定启动哪个 Activity，而是设置 Action、Data、Category，让系统来筛选出合适的 Activity，筛选是根据所有的＜intent-filter＞来筛选。例如有甲乙丙三人：甲 160cm，乙 170cm，丙 180cm。如果是显式意图的话，要指明选择乙的话会说："我选择乙"，但是如果是隐式调用，则会说："我要选择 170cm 的人"，虽然没有指明要选乙，但会寻找条件最匹配的人。

在 intent 过滤器中类似于上面例子中的"身高"条件的匹配条件如下。

(1) action：行为。

(2) category：提供将要执行的 action 的额外信息。

(3) data：scheme、host、path、type，data 属性匹配信息。

当在程序中设置了这些激活组件的条件，程序就会去寻找最匹配的组件，但是注意：只要有一点不匹配，就是不匹配。

隐式 Intent 的核心代码模式，首先是在 AndroidManifest. xml 中为某个 Activity 设置意图过滤器。

```
<activity>
    <intent-filter>
        <action android:name="..."/>
        <category android:name="..."/>
        <category android:name="android.intent.category.DEFAULT"/>
<!--此句一般都要加，激活寻找最匹配的组件-->
        <data android:scheme="..." android:host="..." android:path="/..."
        android:type="..."/>
    </intent-filter>
</activity>
```

以上设置是设置 Activity 本身的属性，接下来在程序中要设置的是要寻找时匹配的条件。

```
Intent intent=new Intent();
intent.setAction("....");
intent.addCategory("....");
intent.setData(Uri.parse("...."));
//设置 data 的 scheme、host、path 条件
intent.setDataAndType(Uri.parse(""),String type);
//同时设置 data 的 scheme、host、path、type 条件
startActivity(intent);
//调用 intent.addCategory("android.intent.category.DEFAULT");
```

例如，在 MainActivity 启动时直接进行 Activity 隐式跳转，将进行隐式 Intent 匹配，最后寻找到并激活 NewActivity。

```
<activity
         android:name=".NewActivity"
         android:label="@string/title_activity_new"
         android:theme="@style/AppTheme.NoActionBar" >
         <intent-filter>
             <action android:name="com.neuedu.action" />
             <category android:name="android.intent.category.DEFAULT" />
             <category android:name="com.neuedu.category" />
             <data
                 android:host="www.baidu.com"
                 android:scheme="baidu"/>
         </intent-filter>
     </activity>
```

在 NewActivity 中设置了 action 为 com. neuedu. action，category 为 com. neuedu. category，data 的属性 host、scheme 分别为 www. baidu. com 和 baidu。

在 MainActivity 中进行隐式跳转，隐式调用如下代码。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    //Activity 跳转，隐式调用
    Intent intent=new Intent();
    intent.setAction("com.neuedu.action");
    intent.addCategory("com.neuedu.category");
    intent.setData(Uri.parse("neuedu://www.my12306.com/"));
    startActivity(intent);
    //此方法中调用 intent.addCategory("android.intent.category.DEFAULT");
}
```

3.4.2 向下一个 Activity 传递数据

当在一个 Activity 中启动另一个 Activity 时，经常需要传递一些数据过去。这时就可以通过 Intent 来实现，因为 Intent 通常被称为是两个 Activity 之间的信使，通过将要传递的数据保存在 Intent 中，就可以将其传递到另一个 Activity 中了。利用 Intent 的 putExtra()来存储需要传递的数据，可以传送 int、long、char 等一些基础类型，对复杂的对象建议采用 Bundle 进行传递。putExtra("A",B)中，AB 为键值对，第一个参数为键名，第二个参数为键对应的值。如果取出 Intent 对象中的这些值，则需要在另一个 Activity 中调用 getStringExtra()等方法，注意需要使用对应类型的方法，参数为键名。假设 Intent 就像一封邮件，里面有送信人地址（原始 Activity），也有收信人地址（目标 Activity），参数也可看作是信件内容。

Intent 在传递复杂数据时，可以借助 Bundle 来实现。Bundle 只是一个信息的载体，将内部的数据以键值对形式存储。在传复杂参数前，需要将对象封装起来，在 Android 系统中，通过 Parcelable 和 Serializable 接口实现在进程间和网络中封装传递对象。

Parcelable 的性能优于 Serializable，在内存开销方面较小，所以在内存间数据传输时推荐使用 Parcelable，如 Activity 间传输数据。而 Serializable 可将数据持久化方便保存，所以在需要保存或网络传输数据时选择 Serializable。

下面通过一个具体的实例介绍如何使用 Intent 在 Activity 之间交换数据，传递复杂数据对象时，需要对象是可序列化的，这里的 Data 就是一个实现 Serializable 接口的可序列化类。

【例 3.2】 使用 Intent 从 TransmitDataActivity 传递一个字符串、一个整数和一个 Data 对象到 MyActivity 中，在 MyActivity 中把传递过来的数据显示在 TextView 上。

将数据保存到 Intent 对象，并跳到另一个用来显示这些数据的 Activity 的代码如下。

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import cn.edu.neusoft.software.mobile.activitydemo.R;
public class TransmitDataActivity extends AppCompatActivity {
private Button btn;
    @Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_transmit_data);
btn=(Button) findViewById(R.id.btnValue);
btn.setOnClickListener(new View.OnClickListener() {
            @Override
public void onClick(View v) {
```

```
                Intent intent=new Intent(TransmitDataActivity.this,
                MyActivity.class);
                intent.putExtra("intent_string","使用 intent 传值");
                intent.putExtra("intent_integer",100);
                Data data=new Data();
                data.id=1000;
                data.name="Android";
                intent.putExtra("intent_object",data);
                startActivity(intent);
            }
        });
    }

}
```

上面的代码涉及一个 Data 类，这个类是可序列化的，也就是实现了 java. io. Serializable 接口的类。Data 类的代码如下。

```
public class Data implements Serializable{
    String name;
    int id;
}
```

在 MyActivity 类中获取通过 Intent 对象传递来的 3 个值(String、Integer 和 Data 类型的值)的代码如下。

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.TextView;
import cn.edu.neusoft.software.mobile.activitydemo.R;
public class MyActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);
        TextView textView= (TextView)findViewById(R.id.textResult);

        String intentString=getIntent().getStringExtra("intent_string");
        /* String intentString=getIntent().getExtras().getString("intent_
           string"); */
        int intentInteger=getIntent().getExtras().getInt("intent_integer");
        Data data= (Data)getIntent().getExtras().get("intent_object");
```

```
        StringBuffer sb=new StringBuffer();
        sb.append("intent_string");
        sb.append(intentString);
        sb.append("\n");
        sb.append("intent_integer");
        sb.append(intentInteger);
        sb.append("\n");
        sb.append("data.id");
        sb.append(data.id);
        sb.append("\n");
        sb.append("data.name");
        sb.append(data.name);
        sb.append("\n");
        textView.setText(sb.toString());
    }
}
```

程序分析：使用 Intent 传递数据，是最常用的一种数据传递的方法。通过 Intent.putExtra 方法可以将简单类型的数据或可序列化的对象保存在 Intent 对象中，然后在目标 Activity 中使用 getXxx(getInt、getString 等)方法获得这些数据。

运行程序后，单击图 3.6 所示界面的按钮，会显示图 3.7 所示的输出信息。

图 3.6　TransmitData 程序的主界面

图 3.7　显示从 Intent 对象中获取的数据

3.4.3　返回数据给上一个 Activity

在开发应用时，不仅需要传递数据给其他 Activity，也要从其他 Activity 中返回数据。返回数据，一般采用 Intent 对象的方式来返回数据，采用这种方式，需要使用 startActivityForResult(Intent intent,int requestCode)，requestCode 的值是自定义的，用于识别跳转的目标 Activity。跳转的目标 Activity 所要做的就是返回数据/结果，setResult(int resultCode)只返回结果不带数据，或者 setResult(int resultCode,Intent data)两者都返回。而接收返回的数据/结果的处理函数是 onActivityResult(int requestCode,int resultCode,Intent data)，这里的 requestCode 就是 startActivityForResult 的 requestCode，resultCode 就是 setResult 里面的 resultCode，返回的数据在 data 里面。

【例 3.3】 使用 startActivityForResult 从目标 Activity 返回值，创建 3 个 Activity，名字分别为 FirstActivity、SecondActivity 和 ThirdActivity，在 FirstActivity 布局上放置两个按钮，单击分别跳到 SecondActivity 和 ThirdActivity，在 SecondActivity 和 ThirdActivity

上放置按钮，单击返回到 FirstActivity，同时分别返回一个字符串给 FirstActivity，在 FirstActivity 中通过对话框显示返回的字符串。

在 FirstActivity 类中获取通过 startActivityForResult 启动 SecondActivity 和 ThirdActivity，在 onActivityResult 中处理返回结果。FirstActivity 的代码如下。

```
import android.content.Intent;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
public class FirstActivity extends AppCompatActivity {
private Button btnSecond,btnThird;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_first);
        btnSecond=findViewById(R.id.btnSecond);
        btnSecond.setOnClickListener(new ButtonListener());
        btnThird=findViewById(R.id.btnThird);
        btnThird.setOnClickListener(new ButtonListener());
    }
  class ButtonListener implements View.OnClickListener{

      @Override
      public void onClick(View v) {
          switch (v.getId()){
              case R.id.btnSecond:
                  Intent intent=new Intent(FirstActivity.this,SecondActivity.
                  class);
                  startActivityForResult(intent,1);
                  break;
              case R.id.btnThird:
                  Intent intent1=new Intent(FirstActivity.this,ThirdActivity.
                  class);
                  startActivityForResult(intent1,2);
                  break;
          }
      }
  }
```

```
    @Override
    protected void onActivityResult(int requestCode, int resultCode, @Nullable
    Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        switch (requestCode) {
            case 1:
                if(resultCode==RESULT_OK) {
                    String returnedData=data.getStringExtra("data_return");
                    Toast.makeText(this,returnedData+" ",Toast.LENGTH_
                    SHORT).show();
                    Log.d("FirstActivity", returnedData);
                }
                break;
            case 2:
                if(resultCode==RESULT_OK) {
                    String returnedData=data.getStringExtra("data_return");
                    Toast.makeText(this,returnedData,Toast.LENGTH_LONG).
                    show();
                    Log.d("FirstActivity", returnedData);
                }
                break;
            default:
        }
    }
}
```

FirstActivity 对应的布局文件如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FirstActivity">
    <Button
        android:id="@+id/btnSecond"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="跳到第二个"

      />
```

```
    <Button
        android:id="@+id/btnThird"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="跳到第三个"
         />
</LinearLayout>
```

SecondActivity 的代码如下。

```
import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.Nullable;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
    public class SecondActivity extends AppCompatActivity {
        private Button btn;
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        btn=findViewById(R.id.back);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent=new Intent();
                intent.putExtra("data_return", "Hello FirstActivity,I am
                secondActivity");
                setResult(RESULT_OK, intent);
                finish();

            }
        });
    }
}
```

SecondActivity 对应的布局文件如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
   <Button
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/back"
        android:text="返回"/>
</LinearLayout>
```

ThirdActivity 的后台代码和对应的布局请参考 SecondActivity 自行完成。程序运行效果如图 3.8 所示，在 MainActivity 中单击“跳到第二个”按钮跳到第二个 Activity 中，在 SecondActivity 单击“返回”按钮返回到 MainActivity 中，同时在 MainActivity 中显示返回的结果，如图 3.9 和图 3.10 所示。

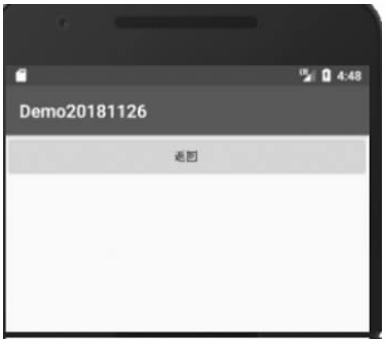

图 3.8　SecondActivity 运行效果

图 3.9　带返回值的运行效果 1

图 3.10　带返回值的运行效果 2

3.5 知识拓展：Activity/ActionBarActivity/AppCompatActivity

在 AS 中创建的 MainActivity 类默认是继承自 AppCompatActivity，它是 Activity 的子类。ActionBarActivity 与 AppCompatActivity 都是 Activity 的子类，ActionBarActivity 在 AS 中已经是过期类。在 AS 中如果创建的类继承自 Activity，则不带 ActionBar，如果要在 AS 中也使用 ActionBar，并且不使用已经过时的 ActionBarActivity，有什么办法呢？就是使用 AppCompatActivity。在 AS 中把 MainActivity 继承自 AppCompatActivity，并导入对应的包。在以后的项目中，可以通过手动修改 Activity 的继承父类，来决定是否显示 ActionBar，并且对程序没有其他影响。

小　　结

本章给出了一个简单的例子来作为 Android 入门学习的第一个程序。通过这个程序读者可以掌握建立 Android 程序的方法，理解 Android 程序的基本结构，理解 Android 程序的组件式开发方法，掌握 Android 的四大组件及 Activity 的基本概念与用法。

习　　题

1. 两个 Activity 之间跳转时必然会执行的是哪几个方法？

2. 简述 Android 系统的 4 种基本组件 Activity、Service、BroadcastReceiver 和 ContentProvider 的用途。

3. Intent 的作用是什么？Intent 传递数据时，可以传递哪些类型数据？

4. 简要说明 gradle 文件的主要作用。

5. Activity 对象的 7 个生命周期方法是什么？说明各方法的执行时机。

6. Activity 生命周期中，第一个需要执行的方法是什么？