

第 5 章

Activity

Activity 是与用户交互的接口,提供了图形界面供用户操作。前文中每个应用程序都用到了 Activity: 通过调用 setContentView()方法,为 Activity 生成一个图形界面。因此,Acrtivity 是 Android 系统最重要的功能之一。

5.1 生命周期

在一个 Android 应用中可以有多个 Activity,这些 Activity 组成了 Activity 栈(Stack),当前活动的 Activity 位于栈顶,之前的 Activity 被压入下面成为非活动 Activity,等待是否可能被恢复为活动状态。Activity 的 4 个重要状态见表 5-1。

表 5-1 Activity 的 4 个重要状态

序号	状 态	说 明
1	运行状态	当前 Activity,用户可见,可以获得焦点
2	暂停状态	失去焦点的 Activity,仍然可见
3	停止状态	该 Activity 被其他 Activity 覆盖,不可见,会保存所有状态和信息
4	销毁状态	该 Activity 结束

一个 Activity 从创建到消亡叫作一个生命周期,其中有很多有用的回调函数,方便截获添加有价值的处理功能。Activity 常用的回调函数见表 5-2。

表 5-2 Activity 常用的回调函数

序号	函 数 名 称	说 明
1	onCreate()	创建 Activity 时被回调,执行更多的初始化功能,如添加消息响应

续表

序号	函数名称	说 明
2	onStart()	启动 Activity 时被回调,也就是当 Activity 变为可见时被回调
3	onResume()	Activity 由暂停态变为运行态时被调用
4	onPause()	暂停 Activity 时被调用,通常用于持久保存数据
5	onRestart()	重新启动 Activity 时被调用,在 onStop()后运行
6	onStop()	停止 Activity 时被回调
7	onDestroy()	销毁 Activity 时被回调

对 Activity 来说,onCreate()与 onDestroy()函数都运行一次,其他回调函数可运行多次,主要有 4 个流程,如下所示。

流程 1: 若焦点一直在 Activity 上,则流程为 onCreate()->onStart()->onResume()->正常运行界面的其他功能->onDestroy()正常销毁。

流程 2: 焦点有切换,假设有两个 Activity,名称为 a、b。初始时焦点在 a 上,则流程为 onCreate()->onStart()->onResume()->正常运行界面的其他功能;当切换到名称为 b 的 Activity 时,对 a 来说,运行 onPause->onStop();当焦点再切回到名称为 a 的 Activity 时,对 a 来说,运行 onRestart()->onStart()->onResume->正常运行界面的其他功能。

流程 3: 从 onPause()->onResume()函数,例如,当在某 Activity 中弹出某模态对话框后,Activity 运行 onPause()函数,当关闭模态对话框后,Activity 运行 onResume()函数。

流程 4: 由 App 进程管理器直接销毁该 Activity 及对应的应用程序。

【例 5-1】 验证 Active 各回调函数的执行顺序。

本示例仅能验证上文所述的流程 1、流程 2 的情况。许多 Android 书都用日志监测 Activity 各回调函数的执行流程,在集成开发环境中看日志较方便,但将该应用安装在真实手机上看日志却并不方便。本示例利用图形用户界面显示 Activity 各回调函数响应的各种情况,在集成开发平台与在真实手机中均非常方便。思路是: 将 Activity 响应函数名称显示在 TextView 控件中。各关键代码如下所示。

① 主布局文件: main.xml。

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

```
        android:layout_height="match_parent">
        <TextView
            android:id="@+id/mytext"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:textSize="30sp"/>
    </LinearLayout>
```

以上代码采用线性布局,定义了 id 为 mytext 的 TextView 控件,用于显示 Activity 各回调函数的响应名称。

② MainActivity.java。

```
public class MainActivity extends AppCompatActivity {
    void show(String name) {
        TextView tv = (TextView) findViewById(R.id.mytext);
        String s = tv.getText().toString();
        tv.setText(s + "\n" + name);
    }
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show("onCreate");
    }
    protected void onStart() {super.onStart();show("onStart");}
    protected void onDestroy() {super.onDestroy();show("onDestroy");}
    protected void onStop() {super.onStop();show("onStop");}
    protected void onResume() {super.onResume();show("onResume");}
    protected void onPause() {super.onPause();show("onPause");}
    protected void onRestart() {super.onRestart();show("onRestart");}
}
```

关键理解 show(String name)函数,name 即回调函数字符串名称。该函数首先获得 TextView 组件上已有的内容,将其与当前 name 字符串相加,再重新设置回 TextView。

5.2 建立 Activity

5.2.1 入口 Activity 类

一个应用可以包含多个 Activity 活动页面,其入口 Activity 类对应应用的第一个界

面,之前经常用的 MainActivity.java 就是入口 Activity 类,如何区分入口 Activity 类与其他普通 Activity 类? 这涉及应用配置文件 Androidmanifest.xml(在 res 目录下),打开该文件,可以看到与 Activity 类 MainActivity 对应的配置信息在<activity>标签内,如下所示。

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

< activity > 属性 name = “. MainActivity”,标识了该 Active 类的名称为 MainActivity,在工程默认包下。

<activity>子标签<intent-filter>定义了过滤器信息,对初学者而言,只要知道<intent-filter>子标签<action>属性 name=“android.intent.action.MAIN”,<category>属性 name=“android.intent.category.LAUNCHER”,那么它对应的 Activity 类就位于优先级最高级,最先执行。

5.2.2 普通 Activity 类

从入口 Active 类类推,普通 Activity 类也应有如下内容: Activity Java 类、界面 XML 布局文件、Androidmanifest.xml 配置文件中增加的<activity>子标签内容。

【例 5-2】 最简单的 Activity 启动程序。一般来说,已有工程已经包含入口 Activity 类 MainActivity,现在要求再建立一个普通 Activity 类 SecondActivity。MainActivity 对应的主界面(main.xml)有一按钮 OK,当单击 OK 按钮时,启动 SecondActivity 对应的界面(main2.xml);SecondActivity 对应的界面显示“This is second activity”,详细内容如下所示。

① Activity 类界面布局文件。

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

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="ok"/>
</LinearLayout>
//main2.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is second activity"/>
</LinearLayout>

```

② 配置文件 Androidmanifest.xml, 以下仅列出与 Activity 有关的<activity>节点内容。

```

<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name=".SecondActivity"></activity>

```

可以看出, 每增加一个普通 Activity 类, 在配置文件中就增加一个<activity>节点, 而且内容较简单, 设定其 name 属性等于 Activity 类名称即可。

③ 对应的两个 Activity 类。

```

//SecondActivity.java
public class SecondActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
    }
}
//MainActivity.java
public class MainActivity extends AppCompatActivity {
    protected void onCreate(Bundle savedInstanceState) {

```

```

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button b = (Button) findViewById(R.id.myok);
        b.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent in = new Intent(MainActivity.this, SecondActivity.class);
                startActivity(in);
            }
        });
    }
}

```

可以看出,启动另一个 Activity 的关键函数是 `startActivity(Intent intent)`,因此必须先产生 Intent 对象。Intent 一个常用的构造方法是 `Intent(Context ctx, Class<?> cls)`,ctx 是上下文对象,cls 可以是待启动的 Activity 类的类名。



5.3 Activity 通信

Activity 类间数据通信要应用 Intent 对象,Intent 保存数据的本质是 map 映射(键-值),键一般是字符串,值可以是多种类型,Intent 可读、可写,常用的函数如下所示。

- `void putExtra(String key, XXX value)`,写单值函数,XXX 可以是 8 个基本数据类型: byte、short、int、long、float、double、char、boolean。
- `void putExtra(String key, XXX value[])`,写数组函数,XXX 可以是 8 个基本数据类型: byte、short、int、long、float、double、char、boolean。
- `void putExtra(String key, CharSequence value)`,写字符串函数。
- `void putExtra(String key, CharSequence value[])`,写字符串数组函数。
- `void putExtra(String key, Serializable value)`,写实现序列化对象函数。
- `void putExtras(Bundle b)`,写捆绑对象。
- `xxx getXxxExtra(String key, xxx default)`,读基本数据类型,3 个小写的“xxx”表示 8 种基本数据类型之一;首字符大写其余小写的“Xxx”,表示 8 种基本数据类型名称首字符大写,其余小写。该函数的含义是:若读成功,则返回读的值,若读不成功,则返回 default 值。
- `xxx[] getXxxArrayExtra(String key)`,读基本数据类型数组,若不成功,则返回 null。
- `String getStringExtra(String key)`,读字符串数据。
- `String[] getStringArrayExtra(String key)`,读字符串数组数据。

- Serializable getSerializableExtra(String key),读序列化对象数据。
- Bundle getExtras(String key),读捆绑对象。

假设有两个 Activity 类 A、B,A 启动 B,A、B 间传参有以下情况：A 向 B 传送基本数据类型;A 向 B 传送类对象类型;A 接收 B 销毁返回给 A 的数据。下面以关键代码的形式一一加以说明。

① A 向 B 传送基本数据类型,A 发送数据,B 解析数据,见表 5-3。

表 5-3 基本数据类型发送解析关键代码

	A 发送数据	B 解析数据
方法 1	<pre>Intent out =new Intent(A,this,B.class); int no=1000; String name = "zhang"; out.putExtra("no",no); out.putExtra("name", name); startActivity(out);</pre>	<pre>Intent in =B.this.getIntent(); int no=in.getIntExtra("no",0); String name=in.getStringExtra("name");</pre>
方法 2	<pre>Bundle b =new Bundle(); int no=1000;String name="zhang"; b.putInt("no", no); b.putString("name", name); Intent out = new Intent(A,this,B.class); out.putExtras(b); startActivity(out);</pre>	<pre>Intent in =B.this.getIntent(); Bundle b = in.getExtras(); int no = b.getInt("no"); String name = b.getString("name");</pre>

② A 向 B 传送类对象类型,A 发送数据、B 解析数据。

假设自定义 Data 类如下所示。

```
class Data implements Serializable{
    int no;String name;
    Data(int no,String name){this.no=no; this.name=name;}
}
```

之所以实现 Serializable 接口,是因为要用到 putExtra(String key,Serializable value) 函数,value 对应的类要求必须实现 Serializable 接口,其相应的发送、接收关键代码见表 5-4。

表 5-4 类对象数据发送、接收关键代码

A 发送数据	B 解析数据
<pre>Data d =new Data(1000,"zhang"); Intent out = new Intent(A,this,B.class); out.putExtra("data",d); startActivity(out);</pre>	<pre>Intent in =this.getIntent(); Data d = (Data) in.getSerializableExtra("data");</pre>

利用 Bundle 捆绑数据也能实现类对象数据在 Activity 间的发送和解析,本题中的自定义类也无须实现 Serializable 接口。请读者参照表 5-3 中的方法 2 实现相应的发送和接收数据。

③ A 接收 B 销毁返回给 A 的数据。

为了更好地理解此部分知识,还需要掌握系统 Activity 类的 3 个基本函数,如下所示。

- void startActivityForResult(Intent intent, int requestCode),与 startActivity()函数不同,本函数是具有接收返回结果的 Activity 启动函数,requestcode 是请求识别码。
- void finish(),结束当前 Activity 对象,返回到前一个 Active 对象。
- void setResult(int resultCode, Intent data),resultCode 是结果识别码,data 是准备返回前一级 Activity 对象的数据。

另外,还要知道:若某 Activity 对象接收下一级 Activity 对象的返回数据,则必须重写如下函数。

- protected void onActivityResult(int requestCode, int resultCode, Intent data);很明显,data 是返回的数据,requestCode 是请求码,resultCode 是结果码。

实现 A 接收 B 销毁返回给 A 的数据的关键代码见表 5-5。

表 5-5 实现 A 接收 B 销毁返回给 A 的数据的关键代码表

A 中的关键代码	B 中的关键代码
<pre>//A 启动 B 代码 Intent out = new Intent (MainActivity. this, SecondActivity. class); startActivityForResult(out,1); //A 接收 B 的返回值,必须重载下面函数 protected voidonActivityResult(int requestCode, intresultCode, Intent data) { super.onActivityResult(requestCode, resultCode, data); if(requestCode==1 && resultCode==2){ int no = data.getIntExtra("no", 0); } }</pre>	<pre>//B 主动销毁时,直接调用以下函数 onBackPressed() //重写 onBackPressed()函数 public voidonBackPressed() { //super.onBackPressed(); Intent out = new Intent(); out.putExtra("no",1000); setResult(2,out); finish(); }</pre>

B 销毁有两种情况:一是响应 B 中的界面子组件事件,如按钮事件等;二是响应手机固有的“返回”事件。这两种情况的响应函数本质上功能是一致的,因此本例重写了“返回”事件函数 onBackPressed(),完成了向上一级 Activity 对象设置返回值及销毁本级

Activity 对象功能。在 `onBackPressed()` 函数内,注释了一行语句 `super.onBackPressed()`,若把注释打开,运行程序后会发现 A 中的 `onActivityResult()` 函数不会接收 B 中传过来的数据。这是因为 B 中的 `super.onBackPressed()` 函数已经完成了销毁 B 的功能,在该行代码下再设置向 A 发送的数据当然是无效了。

通过此例,再加深理解 `startActivityForResult(intent, requestCode)` 及 `setResult(requestCode, intent)` 中申请码 `requestCode`、结果码 `resultCode` 的作用,这两个值最终都体现在 `onActivityResult(requestCode, resultCode, data)` 函数参数内:根据 `requestCode` 可知道是从哪个 Activity 对象返回的;根据 `resultCode`,可以知道该如何解析 `data`,因为不同的 `resultCode` 可能代表不同的、有用的数据类型。

5.4 隐式启动 Activity

前文所述启动 Activity 都是显示方式,主要体现在建立 Intent 对象时明确体现了启动的 Activity 对象。例如, `Intent in = new Intent(A.this, B.class)`,则 A 是源 Activity 类,B 是待启动的 Activity 类。隐式启动 Activity 的含义是:在程序中看不出启动的 Activity 类,它是通过匹配查找配置文件 `AndroidManifest.xml` 中与 Activity 有关的内容,找出应该启动哪个 Activity 类。

5.4.1 intent-filter

在 `AndroidManifest.xml` 中,与 Activity 有关的主要是 `<activity>` 标签的子标签 `<intent-filter>` 的内容,对隐式启动的 Activity 来说,必须配置 `<intent-filter>` 的内容。它主要包含 `<action>`、`<category>`、`<data>` 3 个子标签的内容设置,下面一一加以说明。

① `<action>` 标签。

`action` 属性的值为一个字符串,它代表系统中已经定义了一系列常用的动作,形如 `<action android:name="XXX">`,XXX 可以自定义,也可以应用系统定义的常量,如下所示。

- 自定义动作字符串。
- `ACTION_MAIN`: Android Application 的入口,每个 Android 应用必须且只能包含一个此类型的 Action 声明。
- `ACTION_VIEW`: 系统根据不同的 Data 类型,通过已注册的对应 Application 显示数据。

- ACTION_EDIT: 系统根据不同的 Data 类型,通过已注册的对应 Application 编辑数据。
- ACTION_DIAL: 打开系统默认的拨号程序,如果 Data 中设置了电话号码,则自动在拨号程序中输入此号码。
- ACTION_CALL: 直接呼叫 Data 中所带的号码。
- ACTION_ANSWER: 接听来电。
- ACTION_SEND: 由用户指定发送方式进行数据发送操作。
- ACTION_SENDTO: 系统根据不同的 Data 类型,通过已注册的对应 Application 进行数据发送操作。

所有系统定义的常量都要加前缀“android.intent.action.”,形如 `<action android:name="android.intent.action.ACTION_MAIN">`。

② `<category>` 标签

`<category>` 标签用于指定当前 intent 被处理的环境,一个 intent-filter 可以包含多个 category 属性。

- 自定义种类字符串。
- CATEGORY_DEFAULT: Android 系统中默认的执行方式,按照普通 Activity 的执行方式执行。
- CATEGORY_HOME: 设置该组件为 Home Activity。
- CATEGORY_PREFERENCE: 设置该组件为 Preference。
- CATEGORY_LAUNCHER: 设置该组件为在当前应用程序启动器中优先级最高的 Activity,通常与入口 ACTION_MAIN 配合使用。
- CATEGORY_BROWSABLE: 设置该组件可以使用浏览器启动。
- CATEGORY_GADGET: 设置该组件可以内嵌到另外的 Activity 中。

所有系统定义的常量都要加前缀“android.intent.category.”,形如 `<category android:name="android.intent.category.CATEGORY_DEFAULT">`。

③ `<data>` 标签。

每个 `<data>` 标签都可以指定一个 URI 结构以及 data 的 MIME 类型。一个完整的 URI 由 scheme、host、port 和 path 组成,其结构如下所示。

```
<scheme>://<host>:<port>/<path>
```

其中,scheme 既可以是 Android 中常见的协议,也可以是自定义的协议。Android 中常见的协议包括 content 协议、http 协议、file 协议等,自定义协议可以使用自定义字符串。

`<data>` 标签格式形如: `<data android:scheme="XXX" android:host="XXX">`。