第 6 天 Android 基础

第六章 Activity

- Activity 的跳转
- Activity 的生命周期
- Activity 的启动模式
- 获取其他 Activity 销毁时设置的数据

1.1 Activity 的创建

Activity 是 Android 四大组件之一，用于展示界面。Activity 中所有操作都与用户密切相关，是一个负责与用户交互的组件，它上面可以显示一些控件也可以监听并处理用户的事件。一个 Activity 通常就是一个单独的屏幕，Activity 之间通过 Intent 进行通信。

1.1.1 如何创建自定义的 Activity

一、自定义类继承 Activity

新建一个 Android 工程，在 src 目录下新建一个类，不妨起名 MyActivity，让该类继承 Android SDK 提供的 Activity。

二、在 MyActivity 中覆写 onCreate()方法

onCreate()方法是在当前 Activity 被系统创建的时候调用的，在该方法中一般要通过 setContentView(R.layout.myactivity)方法给当前 Activity 绑定布局文件或视图。因此为了让我们的 MyActivity 能够展示界面需要在工程目录的 res/layout/目录下创建一个布局文件。比如我创建的布局文件名（注意：Android 的资源文件名中是不允许有大写字母的，必须全小写，如果有多个单词可以用下划线分开）为 myactivity.xml，见【文件 1-1】。

【文件 1-1】 myactivity.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:orientation="vertical" >
6.     <TextView
7.         android:layout_width="match_parent"
8.         android:layout_height="wrap_content"
9.         android:text="这是 MyActivityA"
10.    />
11. </LinearLayout>
```

必须要注意的就是在该方法中必须先调用父类的 onCreate() 方法，也就是 super.onCreate(savedInstanceState);该句代码必须被调用一次。这是为什么呢？我们看看父类的 onCreate()方法里面都干了些啥就知道了。如图 1-1 所示，在父类的方法中执行了一堆业务逻辑，而且在最后一行用了一个成员变量 mCalled 记录了当前方法是否被调用了。虽然我们现在还不看不同这些代码，但是我们已经知道既然父类帮我们实现了这么功能，那么我们子类就一定必须调用一次父类的该方法。

```
protected void onCreate(Bundle savedInstanceState) {
    if (DEBUG_LIFECYCLE) Slog.v(TAG, "onCreate " + this + ": " + savedInstanceState);
    if (mLastNonConfigurationInstances != null) {
        mAllLoaderManagers = mLastNonConfigurationInstances.loaders;
    }
    if (mActivityInfo.parentActivityName != null) {
        if (mActionBar == null) {
            mEnableDefaultActionBarUp = true;
        } else {
            mActionBar.setDefaultDisplayHomeAsUpEnabled(true);
        }
    }
    if (savedInstanceState != null) {
        Parcelable p = savedInstanceState.getParcelable(FRAGMENTS_TAG);
        mFragments.restoreAllState(p, mLastNonConfigurationInstances != null
                ? mLastNonConfigurationInstances.fragments : null);
    }
    mFragments.dispatchCreate();
    getApplication().dispatchActivityCreated(this, savedInstanceState);
    mCalled = true;
}
```

图 1-1　onCreate 方法

三、在 AndroidManifest.xml 中进行 Activity 的注册

Android 中共有四大组件，除了 BroadCastReceiver 之外，其他三个组件如果要使用那么必须在 AndroidManifest.xml 中进行注册（也叫声明）。

我们在上面创建的 MyActivity 如何注册呢？其实如果你不会的话完全可以将 eclipse 自动生成的 MainActivity 的注册代码给拷贝过来，然后将 android:name 的属性值修改成我们的类。见【文件 1-2】

【文件 1-2】 AndroidManifest.xml 中注册 Activity

```
1.  <activity
2.          android:name="com.itheima.android.activity.MyActivity"
3.          android:label="@string/app_name" >
4.          <intent-filter>
5.              <action android:name="android.intent.action.MAIN" />
6.
7.              <category android:name="android.intent.category.LAUNCHER" />
8.          </intent-filter>
9.  </activity>
```

上面用到的标签以及属性含义如下表。

标签/属性名	属性含义	是否必须
activity/android:name	要注册的 Activity 的全限定类名，在 eclipse 中按住 Ctrl+鼠标左键可以点击到该类的代码视图	必须
activity/android:label	Activity 左上角显示的名称，如图 1-2 所示，红色框中的内容就是。	可选
intent-filter	叫意图过滤器，一般用于隐式启动该 Activity	可选
action/android:name	意图过滤器的 Action 名称，可以自定义也可以使用系统提供的	可选

category/android:name	意图过滤器的 category 名称，只能使用系统提供的常量值	可选

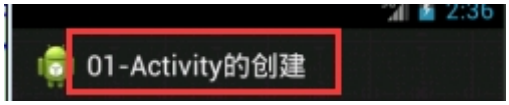


图 1-2　　Activity 的 Label 属性

在【文件 1-2】中，我将 MainActivity 的所有东西都拷贝过来了，只不过是替换了类名。对于 MainActivity 来说正是配置了特定（action 为 android:name="android.intent.action.MAIN"，category 为 android:name="android.intent.category.LAUNCHER"）的 intent-filter，那么当应用安装好以后，才会在桌面创建一个图标，点击该图标打开 MainActivity。现在我自定义的 MyActivity 也配置了一模一样的 intent-filter，那么系统也会为我的 MyActivity 创建一个图标（如图 1-3），如果用户点击了该图标，那么也可以直接打开我的 MyActivity 界面。也就是说一个 Android 工程可以创建多个图标，不过通常情况下一个 Android 应用只要给一个入口的 Activity 配置为入口 Activity 即可。

图 1-3　　一个应用多个图标

1.2 Activity 的跳转

一个 Android 应用通常有多个界面，也就说有多个 Activity，那么如何从一个 Activity 跳转到另外一个 Activity 呢？以及跳转的时候如何携带数据呢？

Activity 之间的跳转分为 2 种：

显式跳转：在可以引用到另外一个 Activity 的字节码，或者包名和类名的时候，通过字节码，或者包名+类名的方法实现的跳转叫做显示跳转。显示跳转多用于自己工程内部多个 Activity 之间的跳转，因为在自己工程内部可以很方便地获取到另外一个 Activity 的字节码。

隐式跳转：隐式跳转不需要引用到另外一个 Activity 的字节码，或者包名+类名，只需要知道另外一个 Activity 在 AndroidManifest.xml 中配置的 intent-filter 中的 action 和 category 即可。言外之意，如果你想让你的 Activity 可以被隐式意图的形式启动起来，那么就必须为该 Activity 配置 intent-filter。

Activity 之间的跳转都是通过 Intent 进行的。Intent 即意图，不仅用于描述一个的 Activity 信息，同时也是一个数据的载体。

1.2.1 显示跳转

为了方便演示，我们创建一个新的 Android 工程《Activity 的跳转》。然后创建两个 Activity 类，分别为 FirstActivity，和 SecondActivity，并在 Android 工程的清单文件中声明这两个 Activity 类。工程清单中添加 Activity 配置如【文件 1-3】所示。

【文件 1-3】 AndroidManifest.xml 中注册 Activity

```
<activity
    android:name="com.itheima.activitySkip.FirstActivity"
    android:label="@string/app_name" >
    <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>

<activity android:name="com.itheima.activitySkip.SecondActivity"/>
```

在上面的清单文件中，我们将 FirstActivity 配置成了主 Activity，SecondActivity 则为普通 Activity。这样 FirstActivity 就是我们应用的入口 Activity 了。

FirstActivity 和 SecondActivity 代码清单如下所示。

【文件 1-4】 FirstActivity.java

```
 package com.itheima.activitySkip;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class FirstActivity extends Activity {

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_first);
}
//绑定界面的 Button 控件的点击事件，点击后实现跳转到 SecondActivity
public void skip2Second(View view){
    //创建一个 Intent 对象，并传递当前对象（Context 对象）和要跳转的 Activity 类字节码
    Intent intent = new Intent(this, SecondActivity.class);
    //Intent 同时也是数据的载体，在跳转的时候可以携带数据
    //通过 intent 的 putExtra(key,value)方法可以设置数据
    //Intent 支持所有基本数据类型及其数组形式
    intent.putExtra("name", "张三");
    //启动第二个 Activity
    startActivity(intent);
```

```
25. }
26. }
27.
```

【文件 1-5】 SecondActivity.java

```
1. public class SecondActivity extends Activity {
2.  protected void onCreate(android.os.Bundle savedInstanceState) {
3.      super.onCreate(savedInstanceState);
4.      setContentView(R.layout.activity_second);
5.      /**
6.       * 获取 Intent 对象，该对象是启动了当前 Activity 的对象
7.       */
8.      Intent intent = getIntent();
9.
10.     if (intent != null) {
11.         // 从 Intent 中获取设置的数据，如果获取不到返回 null
12.         //获取 FirstActivity 中设置的数据
13.         String name = intent.getStringExtra("name");
14.         if (name!=null) {
15.             Toast.makeText(this, "name="+name, Toast.LENGTH_LONG).show();
16.         }
17.     }
18. };
19. }
```

1.2.2 显示跳转用法总结

一、显示 Intent 的写法

1、通过 Intent 的带参构造函数

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

2、通过 Intent 的 setClass 方法

```
intent.setClass(this, SecondActivity.class);
```

二、Intent 可以携带的数据类型

1、八种基本数据类型 boolean、byte、char、short、int、float、double、long 和 String 以及这 9 种数据类型的数组形式

2、实现了 Serializable 接口的对象

3、实现了 Android 的 Parcelable 接口的对象以及其数组对象

1.2.3 隐式跳转

在黑马 Android 基础的第一天我们做了一个拨打电话的功能，其实那个实现的就是通过隐式意图跳转的功能。如果想让我们的 SecondActivity 支持被隐式意图启动，那么就需要给 SecondActivity 配置意图过滤器（intent-filter）。

1、配置 intent-filter

【文件 1-6】 SecondActivity 配置 intent-filter

```
<activity android:name="com.itheima.activitySkip.SecondActivity">
            <!-- 配置意图过滤器 -->
            <intent-filter >
<!-- 在意图过滤器中设置 action 和 category，当有匹配的 action 和 category 的时候启动
该 Activity。这里使用 Android 提供的默认 category 即可 -->
<action android:name="com.itheima.activitySkip.SecondActivity"/>
 <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
            </activity>
```

在【文件 1-6】中我们给 SecondActivity 配置了 intent-filter，该标签下的 action 子标签是我们自定义的，而 category 则必须使用系统提供的。通过我们使用 android:name="android.intent.category.DEFAULT"。配置成这样的 category 的好处就是在启动 SecondActivity 的时候 category 可以设置也可以不设置。

2、通过隐式意图启动 SecondActivity

【文件 1-7】 使用隐式意图启动 SecondActivity 核心代码

```
    //创建一个 Intent 对象
    Intent intent = new Intent();
    //设置 Action，次 Action 必须和 intent-filter 中的 action 一模一样
    intent.setAction("com.itheima.activitySkip.SecondActivity");
    //对于 android.intent.category.DEFAULT 类型的信息为 Android 系统默认的信息，省略也可以
    intent.addCategory("android.intent.category.DEFAULT");
    //启动 Activity
    startActivity(intent);
```

1.2.4 案例-隐式意图打开浏览器

自己将 Android 应用的源码安装在：D:\AndroidSource_GB\AndroidSource_GB 中，打开 D:\AndroidSource_GB\AndroidSource_GB\packages\apps\Browser 目录，然后打开 AndroidManifest.xml 清单文件，找到用于启动浏览器的 intent-filter。大家可能发现有很多个 intent-filter，这么多 intent-filter 只要能匹配上一个则就可以将该 Activity 启动起来。

【文件 1-8】 Browser 浏览器的意图过滤器

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
            <data android:scheme="http" />
            <data android:scheme="https" />
            <data android:scheme="about" />
            <data android:scheme="javascript" />
</intent-filter>
```

观察上面的 intent-flter，发现有一个 data 标签，该标签中有 scheme 属性。这个标签是用于传递数据的。如果某个 intent-filter 有 data 标签，那么在通过该 intent-filter 启动 Activity 的时候就必须设置 data，data 标签的 scheme 属性相当于数据的的前缀或约束，设置数据时必须在数据前加上该前缀。

下面就给出一个打来浏览器的核心代码，如【文件 1-9】所示。

【文件 1-9】 使用隐式意图启动 SecondActivity 核心代码

```
1. //跳转到浏览器界面
2.  public void skip2Browser(View view){
3.      //创建一个 Intent 对象
4.      Intent intent = new Intent();
5.      //设置 Action
6.      intent.setAction("android.intent.action.VIEW");
7.      //设置 category
8.      intent.addCategory("android.intent.category.BROWSABLE");
9.      //设置参数
10.     intent.setData(Uri.parse("http://www.itheima.com"));
11.     //启动 Activity
12.     startActivity(intent);
13. }
```

启动后的界面如图 1-4 所示。

图 1-4　　隐式意图打开浏览器

1.2.5 通过隐式意图给自己的 Activity 传递数据

首先 Intent 本身就是数据的载体，因此不管是显示意图跳转还是隐式意图跳转都可以直接通过 Intent 的 putXXX()方法设置数据。除此之外，隐式意图还可以根据 intent-filter 中的 data 标签然后通过 Intent 的 setData(Uri)方法设置数据。后者是本节要去演示的内容。

一、修改 AndroidManifest.xml 文件中 SecondActivity 的 intent-filter 参数

【文件 1-10】　　给 SecondActivity 配置 intent-filter

```
1. <activity android:name="com.itheima.activitySkip.SecondActivity">
```

```
2.          <!-- 配置意图过滤器 -->
3.          <intent-filter >
4.              <!-- 在意图过滤器中设置 action 和 category，当有匹配的 action 和 category
5. 的时候启动该 Activity -->
6.              <action android:name="com.itheima.activitySkip.SecondActivity"/>
7.              <category android:name="android.intent.category.DEFAULT"/>
8.             <!-- 设置数据协议 -->
9.             <data android:scheme="money"/>
10.             <!-- 配置数据的 mimeType:必须为 xxx/xxx 的格式，否则会报异常  -->
11.              <data android:mimeType="data/mymime"/>
12.          </intent-filter>
13.          </activity>
```

【文件 1-10】跟【文件 1-6】相比，多了两个 data 标签，这两个标签分别配置了 scheme 和 mimeType 属性。那么对于这样的 intent-filter 我们该如何设置数据呢？

二、编写跳转的核心代码

【文件 1-11】 跳转功能的核心代码

```
1. // 发送数据给 SecondActivity
2.  public void sendData2Second(View view) {
3.      // 创建一个 Intent 对象
4.      Intent intent = new Intent();
5.      // 设置 Action
6.      intent.setAction("com.itheima.activitySkip.SecondActivity");
7.      // 对于 android.intent.category.DEFAULT 类型的信息为 Android 系统默认的信息，省略也可以
8.      intent.addCategory("android.intent.category.DEFAULT");
9.      //这里必须调用 setDataAndType 方法同时将 data 和 type 设置出来
10.     intent.setDataAndType(Uri.parse("money:转账 100 元。"), "data/mymime");
11.     // 启动 Activity
12.     startActivity(intent);
13. }
```

注意：

上面的代码是根据 SecondActivity 的 intent-fiter 的配置文件编写的，配置文件中的 data 标签声明了 scheme 和 mimeType 属性，因此这里要想成功跳转就必须使用 setDataAndType()这样的方法传递值。除此之外我们发现 Intent 还有 seteData()和 setType()两个方法，这两个方法一看便知道是设置 data 和设置 mimeType 的，那么我们为何不使用这两个方法呢？

我们查看 Intent 的 setData 和 setType 方法的源码以及 setDataAndType，见【文件 1-12】。

【文件 1-12】 setData 和 setType 源码

```
1.  public Intent setData(Uri data) {
2.          mData = data;
3.          mType = null;
4.          return this;
5.      }
6.
7.    public Intent setType(String type) {
8.          mData = null;
9.          mType = type;
```

```
10.         return this;
11.     }
12.
13.  public Intent setDataAndType(Uri data, String type) {
14.         mData = data;
15.         mType = type;
16.         return this;
17.     }
```

从上面的源码我们发现 setData()的时候给 data 赋值了，但是把 mimeType 的值给设置为 null 了，setType 的时候给 mimeType 赋值了，但是把 data 给设置为 null 了，也就是说这两个方法是互斥的，而 setDataAndType 方法则同时设置了 data 和 mimeType，因此当我们需要同时设置 data 和 mimeType 的时候只能使用 setDataAndType 方法。

三、在 SecondActivity 中接收数据

在第二步中我们的 sendData2Second 方法给 Intent 设置了数据，那么这些数据如何在 SecondActivity 中获取呢？

【文件 1-13】 获取 Intent 中的数据

```
1. public class SecondActivity extends Activity {
2.  private static final String TAG = "SecondActivity";
3.
4.  @Override
5.  protected void onCreate(Bundle savedInstanceState) {
6.      super.onCreate(savedInstanceState);
7.      setContentView(R.layout.activity_second);
8.      //获取 Intent 对象
9.      Intent intent = getIntent();
10.     if (intent!=null) {
11.         //获取 setData 方法设置的值
12.         Uri data = intent.getData();
13.         //获取 data 中的 scheme
14.         String scheme = data.getScheme();
15.         //获取 mimeType
16.         String mimeType = intent.getType();
17.         Log.d(TAG, "data:"+data.toString());
18.         Log.d(TAG, "scheme:"+scheme);
19.         Log.d(TAG, "mimeType:"+mimeType);
20.     }
21. }
22. }
```

执行完上面的代码后日志成功输入了我们的数据，见图 1-5 所示。

```
com.itheima.activity.skip    SecondActi...  data:money:转账100元。
com.itheima.activity.skip    SecondActi...  scheme:money
com.itheima.activity.skip    SecondActi...  mimeType:data/mymime
```

图 1-5 获取数据日志

1.3 Activity 的生命周期

学到这里我们有必要对 Activity 有一个全新的认识。Activity 已经不能再简单的认为就是一个普通的类，其生命周期就是类的加载，对象的的创建和对象的卸载那么简单。Activity 从创建到销毁，整个生命周期是一个非常复杂的过程，该过程由 Android 系统负责维护。

1.3.1 Activity 的 3 种状态

我们人类有婴幼儿时期、青少年时期、中老年时期，Activity 一样也有 3 种状态：Resumed、Paused、Stopped，这三种状态是 Android 官方给出的，我们翻译过来可以理解为：激活或运行状态、暂停状态、停止状态。这 3 种状态的特点如下：

1、Resumed 状态：Activity 位于前端位置，并且获取到了用户的焦点。也就是当前 Activity 完全可见也可用。

注意：在 Android 中目前只允许同时只能有一个 Activity 位于前端位置。

2、Paused 状态：如果另外一个 Activity 位于前端位置并且获取了焦点，但是该 Activity 还依然可见，那么该 Activity 就处于了 Paused 状态。比如如果另外一个 Activity 虽然位于前端，但是是透明的或者没有占满整个屏幕，那么就会出现上面的这种情况。位于 Paused 状态的 Activity 依然是“存活”着的，但是如果系统内存极端的不足，那么就有可能被系统“杀死”以便释放内存。

3、Stopped 状态：当另外一个 Activity 完全将该 Activity 遮盖住的情况下，那么该 Activity 就处于停止状态了。位于停止状态的 Activity 依然“活着”，但是它已经对用户完全不可见了，因此只要系统需要释放内存就会将该 Activity“杀死”。

我们编写的代码要根据 Activity 的不同状态让其做不同的工作，比如如果我们的 Activity 是用于播放视频的，那么当其位于 Resumed 状态时我们可以让视频正常的播放，当 Activity 位于 Paused 状态的时候，我们也应该让我们的视频暂停播放，当 Activity 位于 Stopped 状态时我们应该停止播放视频并视频资源。

那么如何让我们的程序员能够感知到 Activity 的状态变化呢？Android 系统为了将 Activity 状态的变化通知给我们在 Activity 中提供了 7 个回调方法。我们只需要在不同的回调方法中完成不同的工作即可。

1.3.2 Activity 生命周期的 7 个回调方法

【文件 1-14】 Activity 生命周期的 7 个回调方法

方法名	特点
onCreate	当 Activity 第一次创建时被调用，在该方法中我们应该执行创建视图、初始化数据等工作。该方法被调用之后紧接着就是调用 onStart 方法。
onStart	当 Activity 对用户可见前被系统调用。
onResume	当 Activity 可以跟用户交互前被调用，该方法被调用后 Activity 就处于 Resumed 状态。
onPause	当另外一个 Activity 即将成为 Resumed 状态前调用，在该方法中应该保存临时数据、停止动画、暂停视频播放器等。必须要注意的就是该方法执行必须要快，因为只有当该方法执行过之后，另外一个 Activity 才会成为 Resumed 状态，不然会造成 Activity 直接切换的“卡顿”现象。
onStop	当 Activity 对用户已经完全不可见的时候就会调用该方法。当另外一个 Activity 已经成为 Resumed 状态或者当前 Activity 被销毁的情况下会导致当前 Activity 不可见。
onDestory	当 Activity 被销毁前调用。当前 Activity 调用 finish()方法或者系统为了释放内存而将其销

	毁都会调用该方法。
onRestart	当 Activity 处于 onStop 状态之后，如果重新启动则会调用该方法。比如如果当前 Activity 位于 Resumed 状态，此时我们按了 Home 键，那么 Activity 就完全不可见了，onStop 方法就会被执行，然后再长按 Home 键再将该 Activity 重新启动起来就会调用 onRestart 方法。

如图 1-6，是 Android 官方给出的 Activity 生命周期图。

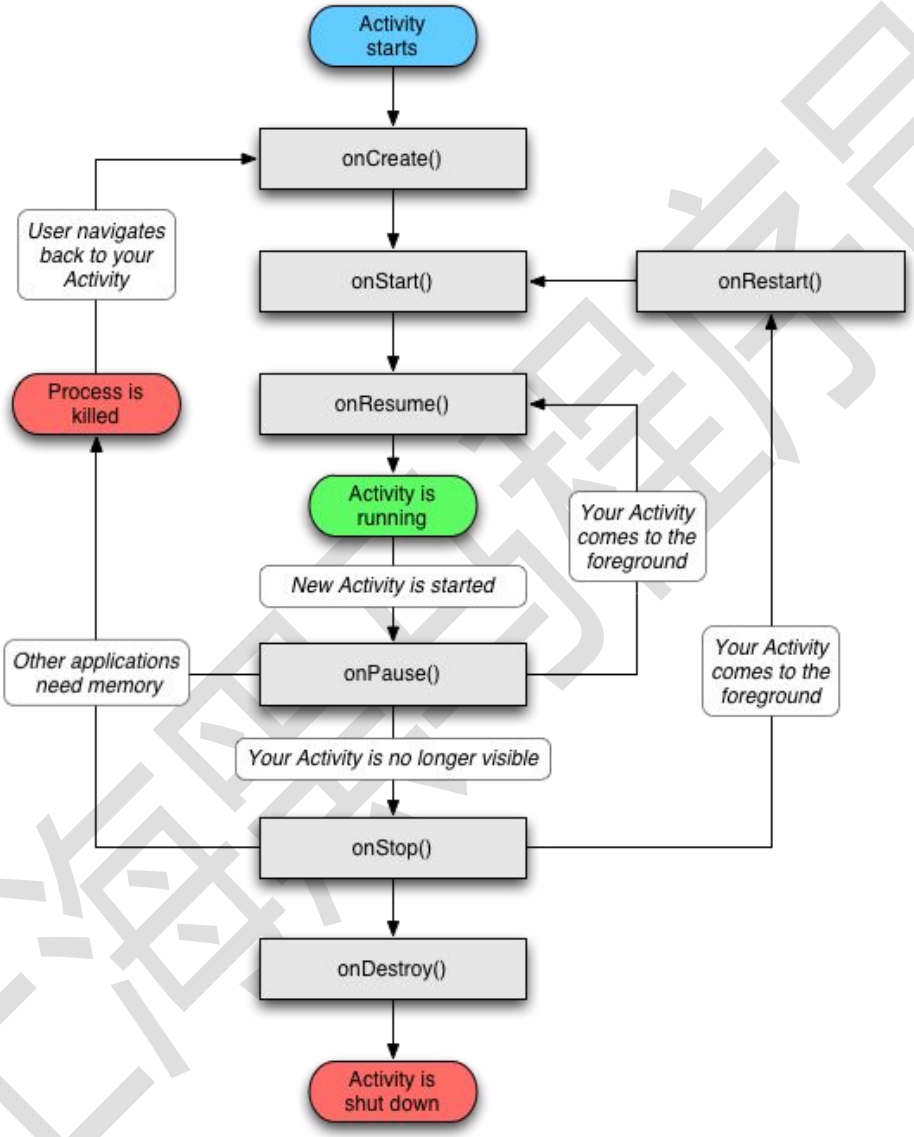


图 1-6　　Activity 生命周期

1.3.3 观察 Activity 生命周期方法的调用

为了观察 Activity 的生命周期，我们新创建一个 Android 工程《Activity 的生命周期》。创建两个 Activity，FirstActivity 和 SecondActivity，在这两个 Activity 中分别覆写 Activity 生命周期的 7 个回调方法。

FirstActivity 代码如【文件 1-15】所示。

【文件 1-15】　　FirstActivity.java

```
1. package com.itheima.activity.lifecycle;
2.
3. import android.os.Bundle;
4. import android.util.Log;
5. import android.view.View;
6. import android.app.Activity;
7. import android.content.Intent;
8. /**
9.  *
10.  * @author wzy 2015-11-12
11.  *
12.  * Activity 的生命周期
13.  */
14. public class FirstActivity extends Activity {
15. private static final String TAG = "MyTag";
16.
17. @Override
18. protected void onCreate(Bundle savedInstanceState) {
19.     super.onCreate(savedInstanceState);
20.     setContentView(R.layout.activity_main);
21.     Log.d(TAG, "FirstActivity: onCreate");
22. }
23. @Override
24. protected void onStart() {
25.     super.onStart();
26.     Log.d(TAG, "FirstActivity: onStart");
27. }
28.
29. @Override
30. protected void onResume() {
31.     super.onResume();
32.     Log.d(TAG, "FirstActivity: onResume");
33. }
34. @Override
35. protected void onPause() {
36.     super.onPause();
37.     Log.d(TAG, "FirstActivity: onPause");
38. }
39. @Override
40. protected void onStop() {
41.     super.onStop();
42.     Log.d(TAG, "FirstActivity: onStop");
43. }
44. @Override
45. protected void onRestart() {
```

```
46.     super.onRestart();
47.     Log.d(TAG, "FirstActivity: onRestart");
48. }
49. @Override
50. protected void onDestroy() {
51.     super.onDestroy();
52.     Log.d(TAG, "FirstActivity: onDestroy");
53. }
54. /**
55.  * 跳转到 SecondActivity
56.  * @param view
57.  */
58. public void click1(View view){
59.     Intent intent = new Intent(this, SecondActivity.class);
60.     startActivity(intent);
61. }
62. /**
63.  * 调用本 Activity 中的 finish()方法
64.  * @param view
65.  */
66. public void click2(View view){
67.     //调用该方法用于销毁当前 Activity
68.     finish();
69. }
70. /**
71.  * 跳转到 FirstActivity
72.  * @param view
73.  */
74. public void click3(View view){
75.     startActivity(new Intent(this, FirstActivity.class));
76. }
77.
78. }
79.
```

【文件 1-16】 SecondActivity.java

```
1. package com.itheima.activity.lifecycle;
2.
3. import android.app.Activity;
4. import android.content.Intent;
5. import android.os.Bundle;
6. import android.util.Log;
7. import android.view.View;
8. /**
9.  *
```

```
10.  * @author wzy 2015-11-12
11.  *
12.  * 观测 Activity 的生命周期
13.  */
14. public class SecondActivity extends Activity {
15. private static final String TAG = "MyTag";
16. @Override
17. protected void onCreate(Bundle savedInstanceState) {
18.     super.onCreate(savedInstanceState);
19.     setContentView(R.layout.activity_second);
20.     Log.d(TAG, "SecondActivity: onCreate");
21. }
22. @Override
23. protected void onStart() {
24.     super.onStart();
25.     Log.d(TAG, "SecondActivity: onStart");
26. }
27.
28. @Override
29. protected void onResume() {
30.     super.onResume();
31.     Log.d(TAG, "SecondActivity: onResume");
32. }
33. @Override
34. protected void onPause() {
35.     super.onPause();
36.     Log.d(TAG, "SecondActivity: onPause");
37. }
38. @Override
39. protected void onStop() {
40.     super.onStop();
41.     Log.d(TAG, "SecondActivity: onStop");
42. }
43. @Override
44. protected void onRestart() {
45.     super.onRestart();
46.     Log.d(TAG, "SecondActivity: onRestart");
47. }
48. @Override
49. protected void onDestroy() {
50.     super.onDestroy();
51.     Log.d(TAG, "SecondActivity: onDestroy");
52. }
53. /**
54.  * 跳转到 FirstActivity
```

```
55.  * @param view
56.  */
57. public void click1(View view){
58.     startActivity(new Intent(this, FirstActivity.class));
59. }
60. /**
61.  * 跳转到当前 Activity 也就是 SecondActivity
62.  * @param view
63.  */
64. public void click2(View view){
65.     startActivity(new Intent(this, SecondActivity.class));
66. }
67. }
68.
```

FirstActivity.java 的 onCreate()方法中使用到了 activity_main.xml 布局文件。

【文件 1-17】 activity_main.xml

```
1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:orientation="vertical">
6.
7.     <Button
8.        android:onClick="click1"
9.        android:layout_width="match_parent"
10.        android:layout_height="wrap_content"
11.        android:text="跳转到 SecondActivity" />
12.     <Button
13.        android:layout_width="match_parent"
14.        android:layout_height="wrap_content"
15.        android:text="调用 finish 方法"
16.        android:onClick="click2"
17.        />
18.     <Button
19.        android:layout_width="match_parent"
20.        android:layout_height="wrap_content"
21.        android:text="跳转到 FirstActivity"
22.        android:onClick="click3"
23.        />
24. </LinearLayout>
```

FirstActivity 的效果图如图 1-7 所示。

图 1-7　　FirstActivity 预览图

FirstActivity 和 SecondActivity 都必须在 AndroidManifest.xml 中进行注册，这里就不给出清单文件代码了，比较简单。

下面我分别以多种情况来分析 Activity 的生命周期调用情况。

一、启动一个新的 Activity 时的调用情况

当新启动 FirstActivity 时生命周期如下图所示：

调用顺序为：onCreate->onStart->onResume

tag:mytag

TID	Application	Tag	Text
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onCreate
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onStart
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onResume

二、点击 back 键，关闭 Activity 时的调用情况

在上面的基础上点击 back 键，生命周期新增了 onPause->onStop->onDestory;

tag:mytag

TID	Application	Tag	Text
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onCreate
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onStart
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onResume
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onPause
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onStop
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onDestroy

三、启动 Activity 后点击 Home 键时的调用情况

启动 FirstActivity，然后点击 Home 键，然后再通过长按 Home 键弹出的快速图标（如图 1-8）启动 FirstActivity。

启动 FirstActivity 到按 Home 键生命周期方法为：onCreate->onStart->onResume->onPause->onStop;如图 1-9 所示。

当再次启动 FirstActivity 时新增加的生命周期为：onRestart->onStart->onResume;如图 1-10 所示。

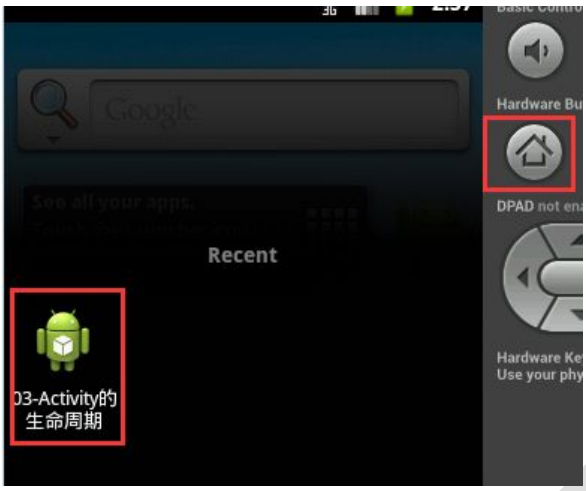


图 1-8　快速启动栏

tag:mytag

TID	Application	Tag	Text
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onCreate
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onStart
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onResume
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onPause
1415	com.itheima.activity.lif...	MyTag	FirstActivity: onStop

图 1-9　按 Home 键后 FirstActivity 的生命周期

tag:mytag

PID	TID	Application	Tag	Text
1415	1415	com.itheima.activity.lif...	MyTag	FirstActivity: onCreate
1415	1415	com.itheima.activity.lif...	MyTag	FirstActivity: onStart
1415	1415	com.itheima.activity.lif...	MyTag	FirstActivity: onResume
1415	1415	com.itheima.activity.lif...	MyTag	FirstActivity: onPause
1415	1415	com.itheima.activity.lif...	MyTag	FirstActivity: onStop
1415	1415	com.itheima.activity.lif...	MyTag	FirstActivity: onRestart
1415	1415	com.itheima.activity.lif...	MyTag	FirstActivity: onStart
1415	1415	com.itheima.activity.lif...	MyTag	FirstActivity: onResume

图 1-10　重新启动 FirstActivity 时的生命周期

四、两个 Activity 之间切换的调用情况

这里要讨论的是两个 Activity 的生命周期。先启动 FirstActivity 然后 FirstActivity 再通过 startActivity 方法启动 SecondActivity，在这个过程中我们观察这两个 Activity 的生命周期调用情况。

1、启动 FirstActivity 时，执行的方法：

FirstActivity:onCreate->FirstActivity:onStart->FirstActivity:onResume->FirstActivity;

2、点击 FirstActivity 的按钮通过 startActivity 启动 SecondActivity 时，新增执行的方法：

FirstActivity:onPause->SecondActivity:onCreate->SecondActvity:onStart->SecondActivity:onResume ->FirtActivity:onStop;这个过程如图 1-11 所示。这里一定要注意的是 FirstActivity 和 SecondActivity 的生命周期是交替进行的。FirstActivity 先暂停，然后等 SecondActivity 完全启动起来了，FirstActivity 再执

行 onStop 方法。

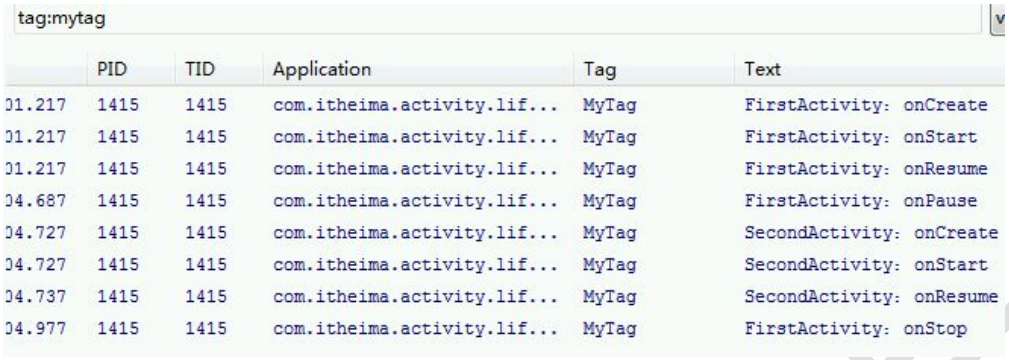


图 1-11　Activity 切换时的生命周期

五、启动一个对话框样式的 Activity 时的调用情况

在该案例中我们先启动 FirstActivity，然后通过点击 FirstActivity 的按钮调用 startActivity 方法，启动一个对话框样式的 Activity，不妨起名 DialogActivity。

1、首先创建一个对话框样式的 Activity

【文件 1-18】　DialogActivity.java

```
1. package com.itheima.activity.lifecycle;
2.
3. import android.app.Activity;
4. import android.os.Bundle;
5.
6. /**
7.  *
8.  * @author wzy 2015-11-13
9.  *
10.  * 对话框样式的 Activity 跟普通样式的 Activity 没有任何差别，
11.  * 唯一的差别是在 AndroidManifest.xml 中配置时需要给对话框样式的 Activity
12.  * 配置一个对话框的主题。
13.  *
14.  */
15. public class DialogActivity extends Activity {
16.
17. @Override
18. protected void onCreate(Bundle savedInstanceState) {
19.     super.onCreate(savedInstanceState);
20.     setContentView(R.layout.activity_dialog);
21. }
22.
23. }
24.
```

activity_dialog.xml 非常简单，就一个 TextView 显示“我是 DialogActivity”，因此不再给出代码。

2、在 AndroidManifest.xml 中给 DialogActivity 配置为对话框样式

【文件 1-19】　AndroidManifest.xml 中配置 DialogActivity。

```
1. <activity android:name="com.itheima.activity.lifecycle.DialogActivity"
```

```
2.          android:theme="@android:style/Theme.Dialog"
3.          />
```

3、观察 FirstActivity 的生命周期

启动 FirstActivity，执行 onCreate->onStart->onResume 方法。

点击按钮，执行 startActivity 方法，启动 DialogActivity（如图 1-12）时走的生命周期为：onPause，仅仅执行了 onPause，这个很好理解因为 DialogActivity 只占据了屏幕一小部分，FirstActivity 依然可见，只不过是不可跟用户交互。

当点击回退键，将 DialogActivity 退出时，走的生命周期为：onResume；

上面的过程见图 1-13。

图 1-12　DialogActivity

TID	Application	Tag	Text
6354	com.itheima.activity.lif...	MyTag	FirstActivity: onCreate
6354	com.itheima.activity.lif...	MyTag	FirstActivity: onStart
6354	com.itheima.activity.lif...	MyTag	FirstActivity: onResume
6354	com.itheima.activity.lif...	MyTag	FirstActivity: onPause
6354	com.itheima.activity.lif...	MyTag	FirstActivity: onResume

图 1-13　启动 DialogActivity 时生命周期

1.3.4 横竖屏切换时 Activity 的生命周期

Android 手机在横竖屏切换时，默认情况下会把 Activity 先销毁再创建，其生命周期如图 1-14。模拟器横竖屏切换的快捷键是 Ctrl+F11。

在类似手机游戏、手机影音这一类的应用中，这个体验是非常差的。不让 Activity 在横竖屏切换时销毁，只需要在清单文件声明 Activity 时配置<activity>节点的几个属性即可，其方式如下：

图 1-14　默认情况下横竖屏切换生命周期

一、 4.0 以下版本：

```
1. android:configChanges="orientation|keyboardHidden"
```

二、 4.0 以上版本：

```
1. android:configChanges="orientation|screenSize"
```

三、 4.0 以上版本：

```
1. android:configChanges="orientation|keyboardHidden|screenSize"
```

将该参数配置到 FirstActivity 中后在切换屏幕方向，那么 Activity 就不会再销毁和重新创建了。但是配置了该参数仅仅是不让 Activity 销毁和重建，Activity 界面依然会跟着屏幕方向重新调整，那么如何固定 Activity 的方向呢？

1.3.5 固定 Activity 的方向

想固定 Activity 的方向其实比较简单，有两种方法：

1、通过配置文件

在 AndroidManifest.xml 中的 activity 节点中添加如下属性。

```
android:screenOrientation="portrait"
```

该属性通常有两个常量值，portrait：垂直方向，landscape：水平方向。

2、通过代码

在 Activity 的 onCreate 方法中执行如下方法。

```
//垂直方向
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
//水平方向
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

1.4　Activity 的启动模式

1.4.1 Activity 的任务栈

Task Stack（任务栈）是一个具有栈结构的容器，可以放置多个 Activity 实例。启动一个应用，系统就会为之创建一个 task，来放置根 Activity；默认情况下，一个 Activity 启动另一个 Activity 时，两个 Activity

是放置在同一个 task 中的，后者被压入前者所在的 task 栈，当用户按下 back 键，后者从 task 中被弹出，前者又显示在屏幕前，特别是启动其他应用中的 Activity 时，两个 Activity 对用户来说就好像是属于同一个应用；系统 task 和应用 task 之间是互相独立的，当我们运行一个应用时，按下 Home 键回到主屏，启动另一个应用，这个过程中，之前的 task 被转移到后台，新的 task 被转移到前台，其根 Activity 也会显示到幕前，过了一会之后，再次按下 Home 键回到主屏，再选择之前的应用，之前的 task 会被转移到前台，系统仍然保留着 task 内的所有 Activity 实例，而那个新的 task 会被转移到后台，如果这时用户再做后退等动作，就是针对该 task 内部进行操作了。

任务栈的设计是为了提高用户体验，但是也有其不足的地方。任务栈的缺点如下：

1、每开启一次页面都会在任务栈中添加一个 Activity，而只有任务栈中的 Activity 全部清除出栈时，任务栈被销毁，程序才会退出，这样的设计在某种程度上可能造成了用户体验差，需要点击多次返回才可以把程序退出了。

2、每开启一次页面都会在任务栈中添加一个 Activity 还会造成数据冗余，重复数据太多，会导致内存溢出的问题(OOM)。

为了解决任务栈产生的问题，Android 为 Activity 设计了启动模式，那么下面的内容将介绍 Android 中 Activity 的启动模式，这也是最重要的内容之一。

1.4.2 Activity 的启动模式

启动模式（launchMode）在多个 Activity 跳转的过程中扮演着重要的角色，它可以决定是否生成新的 Activity 实例，是否重用已存在的 Activity 实例，是否和其他 Activity 实例共用一个 task。

Activity 一共有以下四种 launchMode：standard、singleTop、singleTask、singleInstance。

我们可以在 AndroidManifest.xml 配置<activity>的 android:launchMode 属性为以上四种之一即可。

下面我们结合实例一一介绍这四种 lanchMode。

一、standard

standard 模式是默认的启动模式，不用为<activity>配置 android:launchMode 属性即可，当然也可以指定值为 standard。

我们将创建一个 Activity，命名为 FirstActivity，来演示一下标准的启动模式。FirstActivity 代码如下：

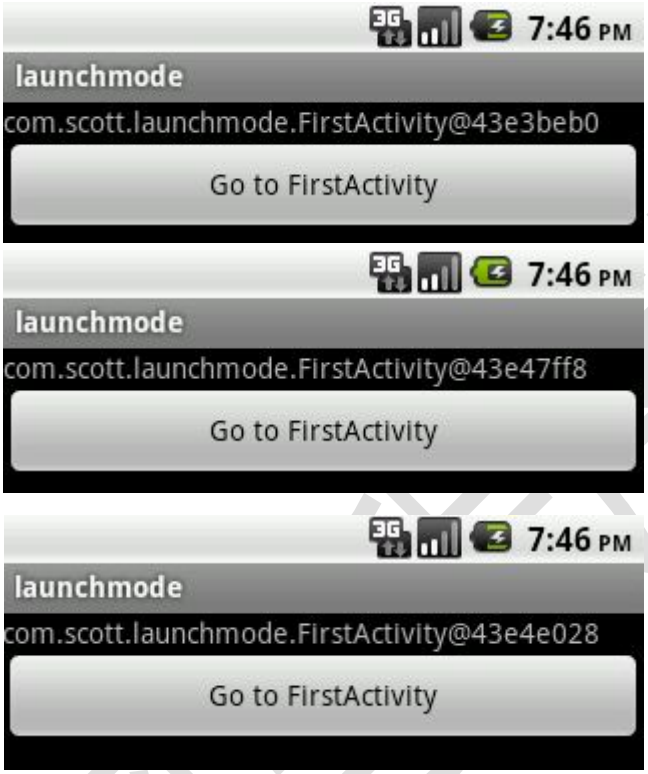
【文件 1-20】 FirstActivity.java

```
1. public class FirstActivity extends Activity {
2.     @Override
3.     public void onCreate(Bundle savedInstanceState) {
4.         super.onCreate(savedInstanceState);
5.         setContentView(R.layout.first);
6.         TextView textView = (TextView) findViewById(R.id.tv);
7.         textView.setText(this.toString());
8.         Button button = (Button) findViewById(R.id.bt);
9.         button.setOnClickListener(new View.OnClickListener() {
10.          @Override
11.          public void onClick(View v) {
12.              Intent intent = new Intent(FirstActivity.this, FirstActivity.class);
13.          startActivity(intent);
14.          }
15.      });
```

```
16.     }
17. }
```

FirstActivity 界面中的 TextView 用于显示当前 Activity 实例的序列号，Button 用于跳转到下一个 FirstActivity 界面。

然后我们连续点击几次按钮，将会出现下面的现象：



我们注意到都是 FirstActivity 的实例，但序列号不同，并且我们需要连续按后退键两次，才能回到第一个 FirstActivity。standard 模式的原理如下图所示：

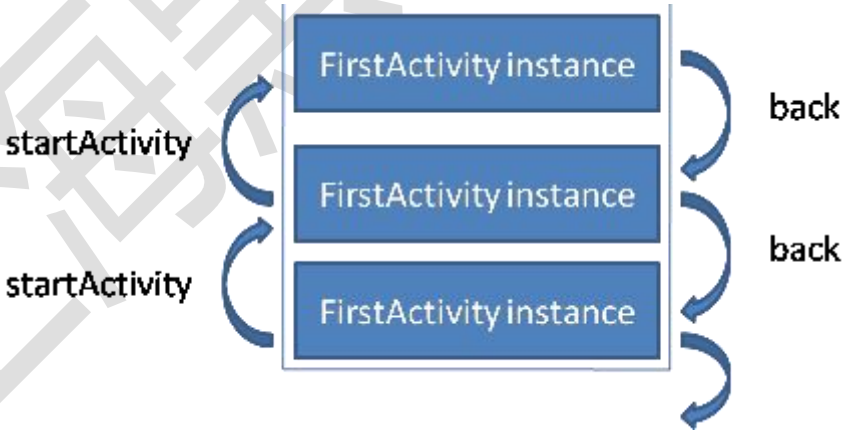


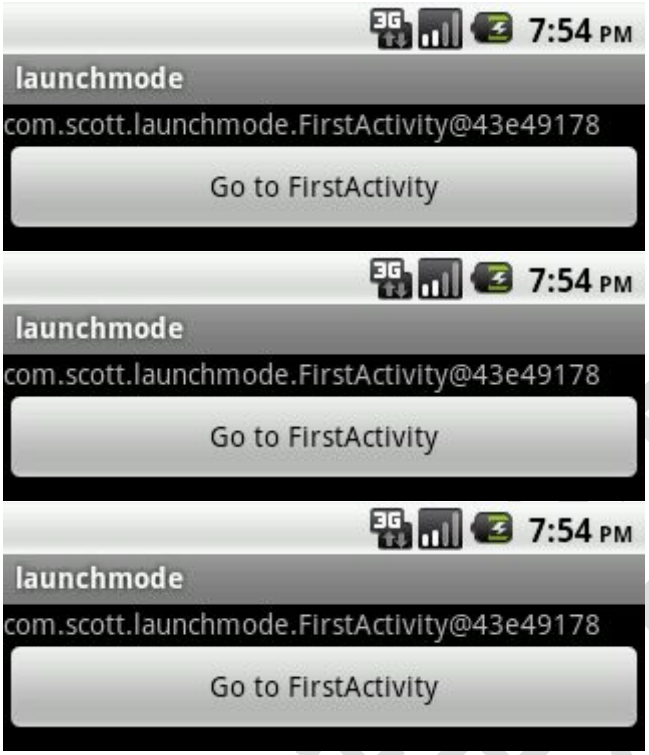
图 1-15　standard 启动模式

如图 1-15 所示，每次跳转系统都会在 task 中生成一个新的 FirstActivity 实例，并且放于栈结构的顶部，当我们按下后退键时，才能看到原来的 FirstActivity 实例。

这就是 standard 启动模式，不管有没有已存在的实例，都生成新的实例。

二、singleTop

我们在上面的基础上为<activity>指定属性 android:launchMode="singleTop"，系统就会按照 singleTop 启动模式处理跳转行为。我们重复上面几个动作，将会出现下面的现象：



我们看到这个结果跟 standard 有所不同，三个序列号是相同的，也就是说使用的都是同一个 FirstActivity 实例；如果按一下后退键，程序立即退出，说明当前栈结构中只有一个 Activity 实例。singleTop 模式的原理如下图所示：

图 1-16　singleTop 启动模式

正如图 1-16 所示，跳转时系统会先在栈结构中寻找是否有一个 FirstActivity 实例正位于栈顶，如果有则不再生成新的，而是直接使用。

也许大家会有疑问，我们只看到栈内只有一个 Activity，如果是多个 Activity 怎么办，如果不是在栈顶会如何？我们接下来再通过一个示例来解释一下大家的疑问。

我们再新建一个 Activity 命名为 SecondActivity，如下：

【文件 1-21】　SecondActivity.java

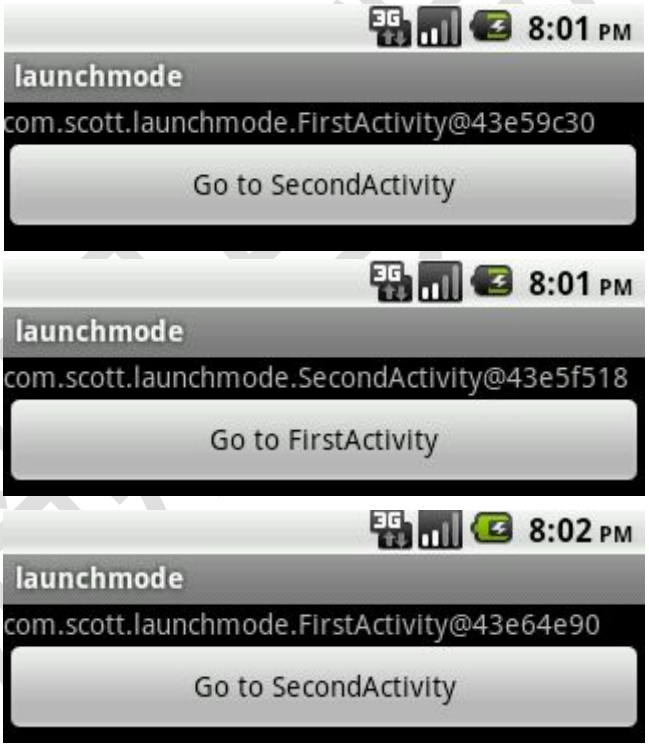
```
1. public class SecondActivity extends Activity {
2.  @Override
3.  protected void onCreate(Bundle savedInstanceState) {
4.      super.onCreate(savedInstanceState);
5.      setContentView(R.layout.second);
6.      TextView textView = (TextView) findViewById(R.id.tv);
```

```
7.        textView.setText(this.toString());
8.        Button button = (Button) findViewById(R.id.button);
9.        button.setOnClickListener(new View.OnClickListener() {
10.         @Override
11.         public void onClick(View v) {
12.             Intent intent = new Intent(SecondActivity.this, FirstActivity.class);
13.         startActivity(intent);
14.         }
15.        });
16. }
17. }
18.
```

然后将之前的 FirstActivity 跳转代码改为：

```
19. Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
20. startActivity(intent);
```

这时候，FirstActivity 会跳转到 SecondActivity，SecondActivity 又会跳转到 FirstActivity。演示结果如下：



我们看到，两个 FirstActivity 的序列号是不同的，证明从 SecondActivity 跳转到 FirstActivity 时生成了新的 FirstActivity 实例。原理图如下：

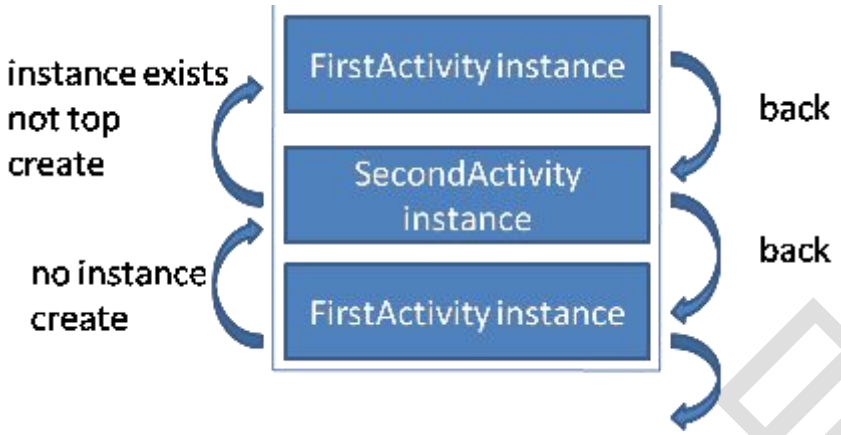


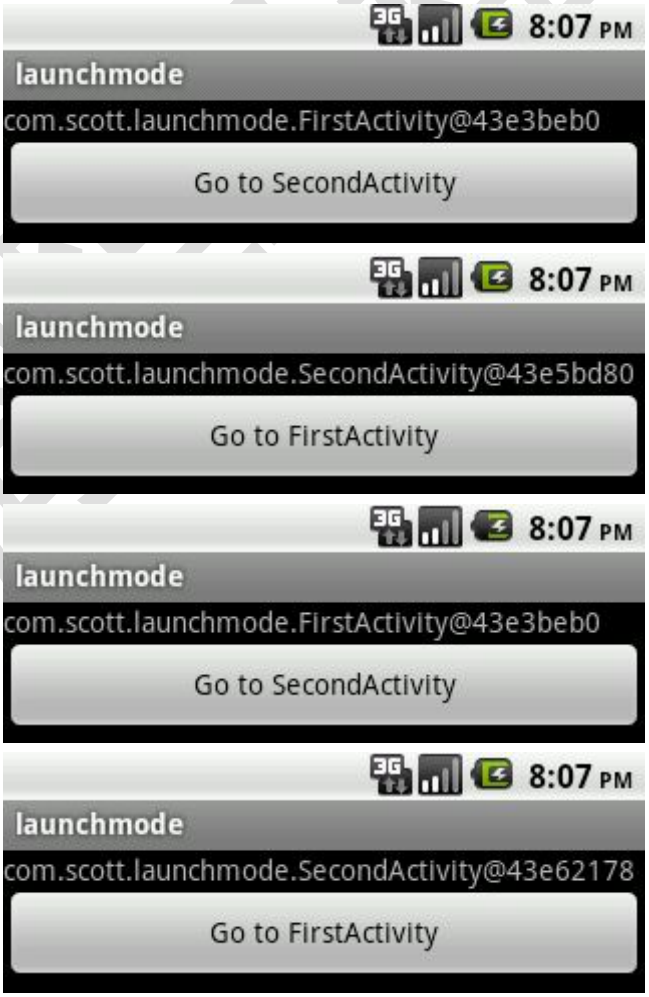
图 1-17　singleTop 启动模式 2

我们看到，当从 SecondActivity 跳转到 FirstActivity 时，系统发现存在有 FirstActivity 实例,但不是位于栈顶，于是重新生成一个实例。

这就是 singleTop 启动模式，如果发现有对应的 Activity 实例正位于栈顶，则重复利用，不再生成新的实例。

三、singleTask

在上面的基础上我们修改 FirstActivity 的属性 android:launchMode="singleTask"。演示的结果如下：



我们注意到，在上面的过程中，FirstActivity 的序列号是不变的，SecondActivity 的序列号却不是唯一

的，说明从 SecondActivity 跳转到 FirstActivity 时，没有生成新的实例，但是从 FirstActivity 跳转到 SecondActivity 时生成了新的实例。singleTask 模式的原理图如下图所示：

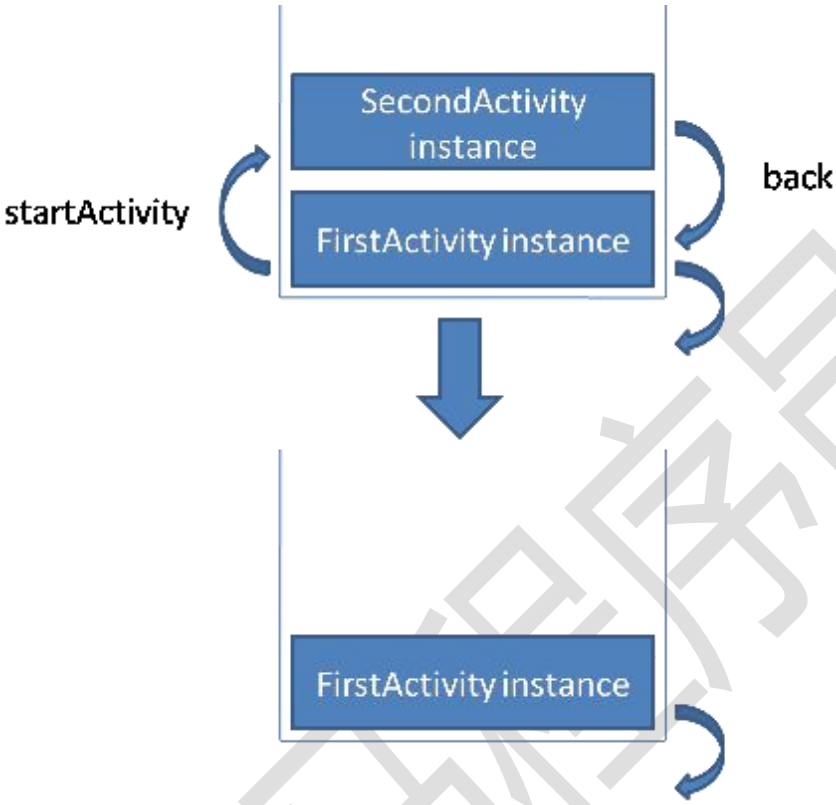


图 1-18　singleTask 启动模式

在图中的下半部分是 SecondActivity 跳转到 FirstActivity 后的栈结构变化的结果，我们注意到，SecondActivity 消失了，没错，在这个跳转过程中系统发现有存在的 FirstActivity 实例，于是不再生成新的实例，而是将 FirstActivity 之上的 Activity 实例统统出栈，将 FirstActivity 变为栈顶对象，显示到屏幕前。也许大家有疑问，如果将 SecondActivity 也设置为 singleTask 模式，那么 SecondActivity 实例是不是可以唯一呢？在我们这个示例中是不可能的，因为每次从 SecondActivity 跳转到 FirstActivity 时，SecondActivity 实例都被迫出栈，下次等 FirstActivity 跳转到 SecondActivity 时，找不到存在的 SecondActivity 实例，于是必须生成新的实例。但是如果我们有 ThirdActivity，让 SecondActivity 和 ThirdActivity 互相跳转，那么 SecondActivity 实例就可以保证唯一。

这就是 singleTask 模式，如果发现有对应的 Activity 实例，则使此 Activity 实例之上的其他 Activity 实例统统出栈，使此 Activity 实例成为栈顶对象，显示到屏幕前。

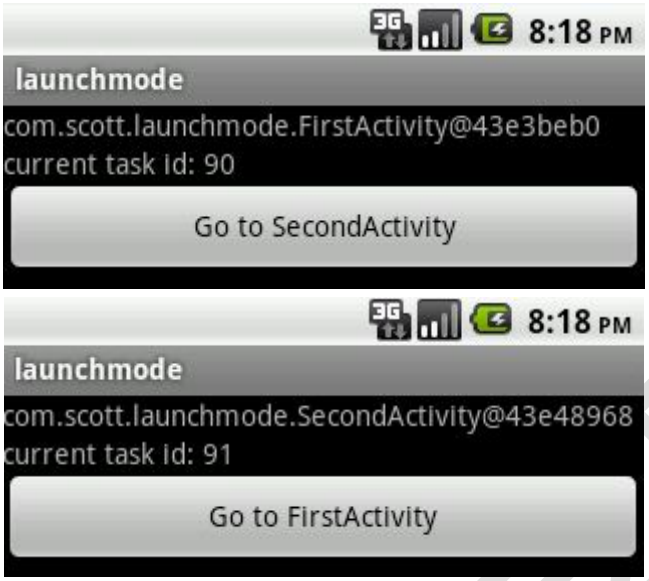
四、singleInstance

这种启动模式比较特殊，因为它会启用一个新的栈结构，将 Activity 放置于这个新的栈结构中，并保证不再有其他 Activity 实例进入。

我们修改 FirstActivity 的 launchMode="standard"，SecondActivity 的 launchMode="singleInstance"，由于涉及到了多个栈结构，我们需要在每个 Activity 中显示当前栈结构的 id，所以我们为每个 Activity 添加如下代码：

```
1. TextView taskIdView = (TextView) findViewById(R.id.taskIdView);
2. taskIdView.setText("current task id: " + this.getTaskId());
```

然后我们再演示一下这个流程：



我们发现这两个 Activity 实例分别被放置在不同的栈结构中，关于 singleInstance 的原理图如下:

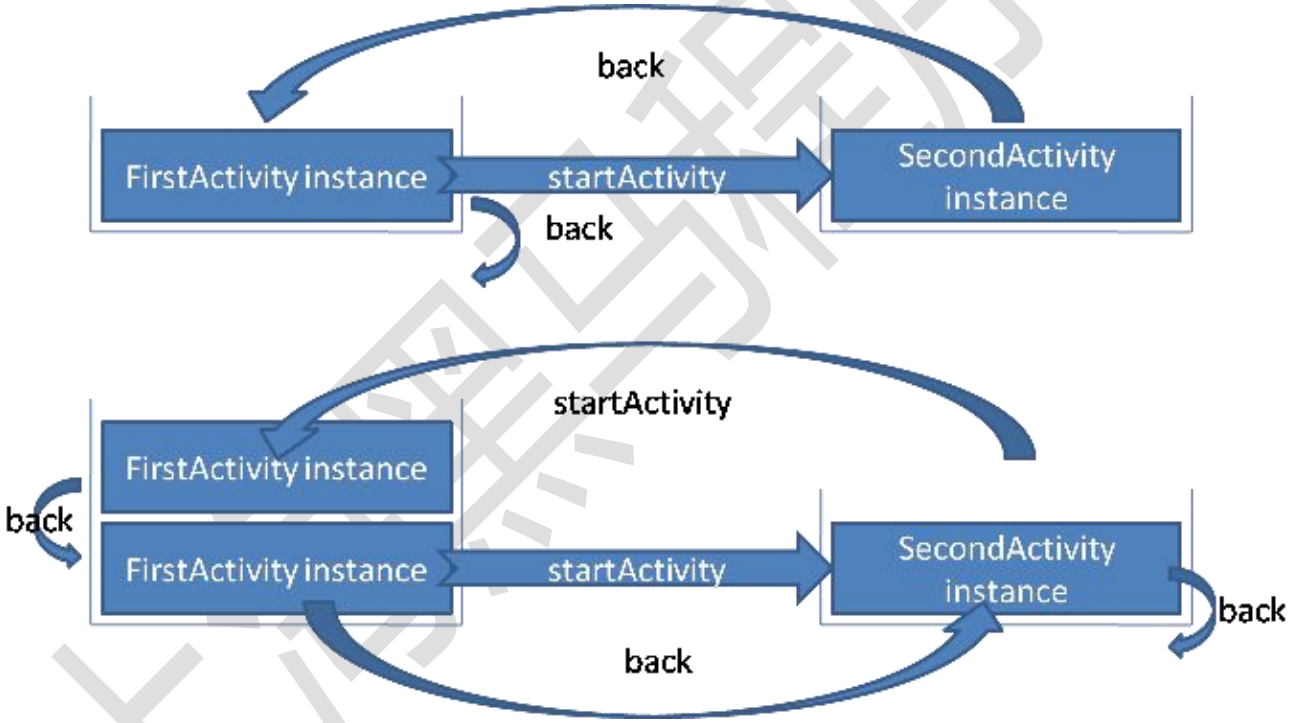
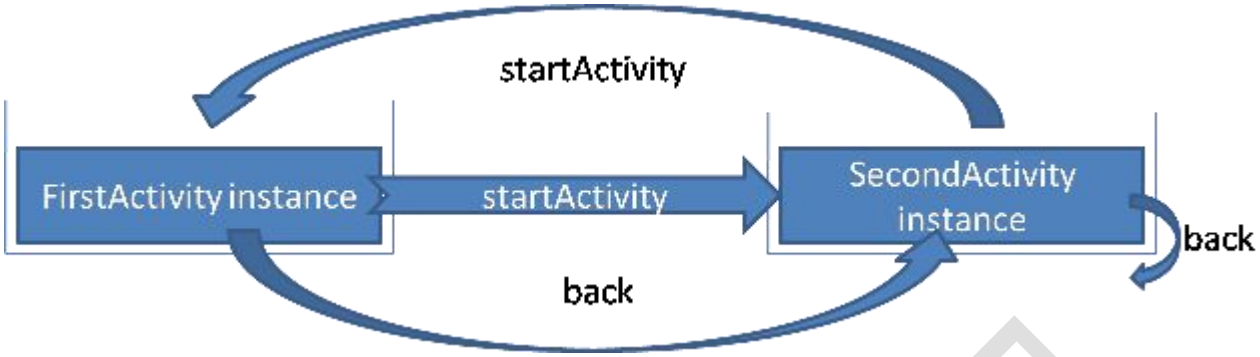


图 1-19　singleInstance 启动模式

我们看到从 FirstActivity 跳转到 SecondActivity 时，重新启用了一个新的栈结构来放置 SecondActivity 实例，然后按下后退键，再次回到原始栈结构；图中下半部分显示的在 SecondActivity 中再次跳转到 FirstActivity，这个时候系统会在原始栈结构中生成一个 FirstActivity 实例，然后回退两次，注意，并没有退出，而是回到了 SecondActivity，为什么呢？是因为从 SecondActivity 跳转到 FirstActivity 的时候，我们的起点变成了 SecondActivity 实例所在的栈结构，这样一来，我们需要“回归”到这个栈结构。

如果我们修改 FirstActivity 的 launchMode 值为 singleTop、singleTask、singleInstance 中的任意一个，流程将会如图所示：



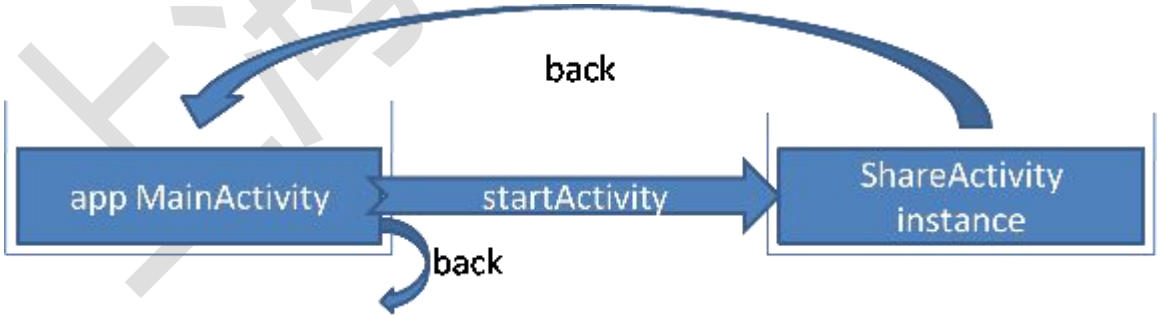
singleInstance 启动模式可能是最复杂的一种模式，为了帮助大家理解，我举一个例子，假如我们有一个 share 应用，其中的 ShareActivity 是入口 Activity，也是可供其他应用调用的 Activity，我们把这个 Activity 的启动模式设置为 singleInstance，然后在其他应用中调用。我们编辑 ShareActivity 的配置：

```
1.  <activity
2.  android:name=".ShareActivity"
3.  android:launchMode="singleInstance" >
4.   <intent-filter>
5.   <action android:name="android.intent.action.MAIN" />
6.   <category android:name="android.intent.category.LAUNCHER" />
7.           </intent-filter>
8.    <intent-filter>
9. <action android:name="android.intent.action.SINGLE_INSTANCE_SHARE" />
10. <category android:name="android.intent.category.DEFAULT" />
11.    </intent-filter>
12. </activity>
```

然后我们在其他应用中这样启动该 Activity：

```
1. Intent intent = new Intent("android.intent.action.SINGLE_INSTANCE_SHARE");
2. startActivity(intent);
```

当我们打开 ShareActivity 后再按后退键回到原来界面时，ShareActivity 做为一个独立的个体存在，如果这时我们打开 share 应用，无需创建新的 ShareActivity 实例即可看到结果，因为系统会自动查找，存在则直接利用。关于这个过程，原理图如下：



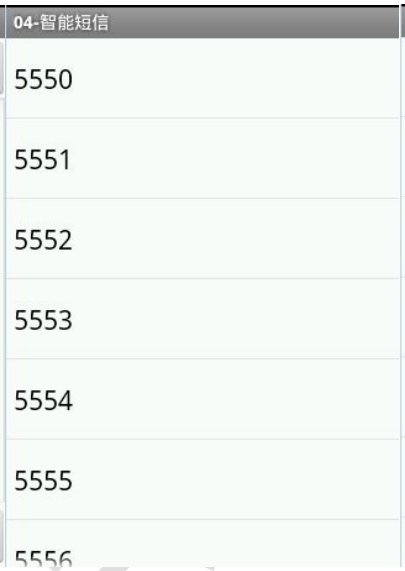
1.5 案例-智能短信

1.5.1 需求

通过该案例可以学习 startActivityForResult 的用法。该案例界面如下图所示。第一幅图是主界面，在该界面可以让用户输入电话号码和短信内容，然后发送出去。“选择号码”点击后跳转到第二个界面，该界面列出的是通讯录中的所有号码（模拟的假数据），显然该界面是用 ListView 显示的，单击 ListView 的单个条目后该 Activity 会退出，同时会将点击的数据自动填充到主界面的电话编辑框中。

点击“快速发送”按钮，打开另外一个界面，如下图第三幅，该界面使用 ListView 展示了常用的快速回复短信内容，当点击一个条目时关闭当前 Activity，同时将点击的短信内容自动填充到主界面的短信文本编辑框中。

电话号码和短信内容选择后，点击“发送”按钮就可以将短信内容给发送出去了。



1.5.2 布局

主界面使用的布局文件为 activity_main.xml。

【文件 1-22】 activity_main.xml

```
1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:orientation="vertical" >
6.
7.     <LinearLayout
8.         android:layout_width="match_parent"
9.         android:layout_height="wrap_content" >
10.
11.        <EditText
12.            android:id="@+id/et_num"
13.            android:layout_width="0dp"
14.            android:layout_height="wrap_content"
15.            android:layout_weight="1"
16.            android:hint="请输入电话号码" />
17.
18.        <Button
19.            android:layout_width="wrap_content"
20.            android:layout_height="wrap_content"
21.            android:onClick="chooseNum"
22.            android:text="选择号码" />
23.    </LinearLayout>
24.
25.    <EditText
26.        android:id="@+id/et_content"
27.        android:layout_width="match_parent"
28.        android:layout_height="0dp"
29.        android:layout_weight="1"
30.        android:gravity="top"
31.        android:hint="请输入短信内容" />
32.
33.    <LinearLayout
34.        android:layout_width="match_parent"
35.        android:layout_height="wrap_content"
36.        android:orientation="horizontal" >
37.
38.        <Button
39.            android:layout_width="0dp"
40.            android:layout_height="wrap_content"
```

```
41.            android:layout_weight="1"
42.            android:onClick="send"
43.            android:text="发送" />
44.
45.        <Button
46.            android:layout_width="0dp"
47.            android:layout_height="wrap_content"
48.            android:layout_weight="1"
49.            android:onClick="selectSend"
50.            android:text="快速发送" />
51.    </LinearLayout>
52.
53. </LinearLayout>
```

“选择号码”和“快速发送”界面是一样，用的是同样的布局，只不过数据不同而已。

【文件 1-23】 activity_list.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:orientation="vertical" >
6.
7.     <ListView
8.         android:id="@+id/lv"
9.         android:layout_width="match_parent"
10.         android:layout_height="match_parent" >
11.    </ListView>
12.
13. </LinearLayout>
```

1.5.3 代码

主界面对应的 MainActivity.java 其，代码如下：

【文件 1-24】 MainActivity.java

```
1. package com.itheima.android.smartsms;
2.
3. import java.util.ArrayList;
4. import android.os.Bundle;
5. import android.app.Activity;
6. import android.content.Intent;
7. import android.telephony.SmsManager;
8. import android.text.TextUtils;
9. import android.view.View;
10. import android.widget.EditText;
```

```
11. import android.widget.Toast;
12. /**
13.  *
14.  * @author wzy 2015-11-13
15.  *
16.  * 智能短信
17.  */
18. public class MainActivity extends Activity {
19.
20. private static final int REQUESTNUM = 1;
21. private static final int REQUESTSMS = 2;
22. private EditText et_num;
23. private EditText et_content;
24.
25. @Override
26. protected void onCreate(Bundle savedInstanceState) {
27.     super.onCreate(savedInstanceState);
28.     setContentView(R.layout.activity_main);
29.     et_num = (EditText) findViewById(R.id.et_num);
30.     et_content = (EditText) findViewById(R.id.et_content);
31. }
32.
33. /**
34.  * 选择电话号码
35.  *
36.  * @param view
37.  */
38. public void chooseNum(View view) {
39.     Intent intent = new Intent(this, ContactActivity.class);
40.
41.     // startActivity(intent);
42.     // 启动 Activity 为了请求你的数据
43.     startActivityForResult(intent, REQUESTNUM);
44. }
45.
46. @Override
47. protected void onActivityResult(int requestCode, int resultCode, Intent data) {
48.     if (data == null) {
49.         return;
50.     }
51.     switch (requestCode) {
52.     case REQUESTNUM:
53.         String num = data.getStringExtra("num");
54.         et_num.setText(num);
55.         break;
```

```
56.     case REQUESTSMS:
57.         String sms = data.getStringExtra("sms");
58.         et_content.setText(sms);
59.     default:
60.         break;
61.     }
62. }
63.
64. /**
65.  * 发送短信
66.  *
67.  * @param view
68.  */
69. public void send(View view) {
70.     // 获取数据
71.     String num = et_num.getText().toString().trim();
72.     String content = et_content.getText().toString().trim();
73.     // 校验数据
74.     if (TextUtils.isEmpty(num) || TextUtils.isEmpty(content)) {
75.         Toast.makeText(this, "号码和内容不能为空", Toast.LENGTH_SHORT).show();
76.         return;
77.     }
78.     // 发送短信
79.     SmsManager manager = SmsManager.getDefault();
80.     // 将短信切割，如果短信过长，切割成多个部分
81.     ArrayList<String> divideMessage = manager.divideMessage(content);
82.     // 发送功能的实现
83.     manager.sendMultipartTextMessage(num, null, divideMessage, null, null);
84.     Toast.makeText(this, "短信发送成功", Toast.LENGTH_SHORT).show();
85. }
86.
87. /**
88.  * 选择快速回复的内容
89.  *
90.  * @param view
91.  */
92. public void selectSend(View view) {
93.     Intent intent = new Intent(this, CommonActivity.class);
94.     startActivityForResult(intent, REQUESTSMS);
95. }
96.
97. }
98.
```

联系人界面对应的是 ContactActivity。其代码如下：

【文件 1-25】　ContactActivity .java

```
1. package com.itheima.android.smartsms;
2.
3. import android.app.Activity;
4. import android.content.Intent;
5. import android.view.View;
6. import android.widget.AdapterView;
7. import android.widget.AdapterView.OnItemClickListener;
8. import android.widget.ArrayAdapter;
9. import android.widget.ListView;
10.
11. /**
12.  *
13.  * @author wzy 2015-11-13
14.  *
15.  */
16. public class ContactActivity extends Activity implements OnItemClickListener {
17. private String[] objects;
18.
19. @Override
20. protected void onCreate(android.os.Bundle savedInstanceState) {
21.     super.onCreate(savedInstanceState);
22.     setContentView(R.layout.activity_list);
23.     ListView lv = (ListView) findViewById(R.id.lv);
24.     lv.setOnItemClickListener(this);
25.     objects = new String[50];
26.     for (int i = 0; i < 50; i++) {
27.         objects[i] = "" + (5550 + i);
28.     }
29.     lv.setAdapter(new ArrayAdapter<String>(this,
30.  android.R.layout.simple_list_item_1, objects));
31. }
32.
33. @Override
34. public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
35.     // 获取到点击的数据
36.     String data = objects[position];
37.     // 创建 Intent，并封装数据
38.     Intent intent = new Intent();
39.     intent.putExtra("num", data);
40.     // 设置返回的数据
41.     setResult(RESULT_OK, intent);
42.     // 销毁当前 Activity，并返回数据
43.     finish();
44. };
```

```
45. }
46.
```

快速发送界面对应的是 CommonActivity，其代码如下：

【文件 1-26】 CommonActivity.java

```
1. package com.itheima.android.smartsms;
2.
3. import android.app.Activity;
4. import android.content.Intent;
5. import android.os.Bundle;
6. import android.view.View;
7. import android.widget.AdapterView;
8. import android.widget.AdapterView.OnItemClickListener;
9. import android.widget.ArrayAdapter;
10. import android.widget.ListView;
11.
12. /**
13.  *
14.  * @author wzy 2015-11-13
15.  *
16.  */
17. public class CommonActivity extends Activity implements OnItemClickListener {
18. private String[] objects;
19.
20. @Override
21. protected void onCreate(Bundle savedInstanceState) {
22.     super.onCreate(savedInstanceState);
23.     setContentView(R.layout.activity_list);
24.     ListView lv = (ListView) findViewById(R.id.lv);
25.     lv.setOnItemClickListener(this);
26.     objects = new String[50];
27.     for (int i = 0; i < 50; i++) {
28.         objects[i] = "我是短信内容" + i;
29.     }
30.     lv.setAdapter(new ArrayAdapter<String>(this,
31.  android.R.layout.simple_list_item_1, objects));
32. }
33.
34. @Override
35. public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
36.     // 获取到点击的数据
37.     String data = objects[position];
38.     // 创建 Intent，并封装数据
39.     Intent intent = new Intent();
40.     intent.putExtra("sms", data);
```

```
41.     // 设置返回的数据
42.     setResult(RESULT_OK, intent);
43.     // 销毁当前 Activity，并返回数据
44.     finish();
45. }
46.
47. }
```

1.5.4 AndroidManifest.xml

【文件 1-27】 AndroidManifest.xml

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3.     package="com.itheima.android.smartsms"
4.     android:versionCode="1"
5.     android:versionName="1.0" >
6.
7.     <uses-sdk
8.         android:minSdkVersion="8"
9.         android:targetSdkVersion="18" />
10.
11.     <uses-permission android:name="android.permission.SEND_SMS" />
12.
13.     <application
14.         android:allowBackup="true"
15.         android:icon="@drawable/ic_launcher"
16.         android:label="@string/app_name"
17.         android:theme="@style/AppTheme" >
18.         <activity
19.             android:name="com.itheima.android.smartsms.MainActivity"
20.             android:label="@string/app_name" >
21.             <intent-filter>
22.                 <action android:name="android.intent.action.MAIN" />
23.
24.                 <category android:name="android.intent.category.LAUNCHER" />
25.             </intent-filter>
26.         </activity>
27.         <activity android:name="com.itheima.android.smartsms.ContactActivity" />
28.         <activity android:name="com.itheima.android.smartsms.CommonActivity" />
29.     </application>
30.
31. </manifest>
```

1.5.5 总结

在上面的案例中我们用到了两个新知识点。

一、关闭当前 Activity

在 Activity 中调用 finish()方法即可关闭当前 Activity。

二、启动一个 Activity 时获取该 Activity 销毁时的返回值

如果想让我们的 MainActivity 打开 ContactActivity，同时能够获取到 ContactActivity 销毁时返回的数据，那么就不能通过 startActivity(Intent)方法去启动 ContactActivity 了。必须使用 startActivityForResult(Intent intent, int requestCode)方法，该方法必须跟 onActivityResult(int requestCode, int resultCode, Intent data)方法结合着使用才行，当 MainActivity 启动的 ContactActivity 销毁前，如果 ContactActivity 调用了 setResult(int resultCode, Intent data)方法，那么系统就会调用 MainActivity 的 onActivityResult()方法，同时将数据 Intent 的形式传递过来。

上面用到了 3 个新的 API：

1、startActivityForResult(Intent intent, int requestCode)

启动 Activity，同时等待该 Activity 返回数据。只有该 Activity 销毁时数据才会被返回。

参数 1：意图，封装要启动的 Activity，当然也可以携带数据

参数 2：请求码，如果是大于 0 的整数，那么该请求码会在 onActivityResult 中的 requestCode 中出现，如果小于等于 0，则不会被返回。

2、onActivityResult(int requestCode, int resultCode, Intent data)

当打开的 Activity 销毁的时候该方法会被调用，从该方法中获取销毁的 Activity 设置的数据。

参数 1：startActivityForResult 方法中的 requestCode

参数 2：setResult 方法中的 resultCode

参数 3：Intent 数据

3、setResult(int resultCode, Intent data)

被打开的 Activity 如果想返回数据，则在销毁前一定要调用该方法，通过该方法设置数据。

参数 1：结果码，该结果码可以用于区分是哪个 Activity 给我们返回的数据

参数 2：设置的数据