《Android 应用程序开发与典型案例》

作者：华清远见

第 6 章 组件间通信

本章简介

在第 5 章的学习中 ,主要了解了 Android 程序界面的开发 ,包括用户界面基础、用户界面控件的使用、界面布局的特点及使用方法、菜单的使用方法、界面事件的处理方法等。在此基础上，本章将对 Android 组件间的通信进行学习，包括 Intent 进行组件通信的原理、Intent 启动 Activity 的方法、获取 Activity 返回值的方法、Intent 过滤器的原理及其匹配机制、发送和接收广播消息的方法等。

6.1 Intent 对象及其属性

Intent 是一个动作的完整描述，包含了动作的产生组件、接收组件和传递的数据信息。Android 则根据 Intent 的描述，在不同组件间传递消息，负责找到对应的组件，将 Intent 传递给调用的组件，组件接收到传递的消息，执行相关动作，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。Intent 为 Activity、Service 和 BroadcastReceiver 等组件提供交互能力，还可以启动 Activity 和 Service，在 Android 系统上发布广播消息。这里的广播消息是指可以接收到的特定数据或消息，也可以是手机的信号变化或电池的电量过低等信息。

因此，Intent 在这里起着一个媒体中介的作用，专门提供组件互相调用的相关信息，实现调用者与被调用者之间的解耦。在 SDK 中给出了 Intent 作用的表现形式。

- 通过 Context.startActivity() or Activity.startActivityForResult()启动一个 Activity。
- 通过 Context.startService() 启动一个服务，或者通过 Context.bindService() 和后台服务交互。
- 通过广播方法（比如 Context.sendBroadcast(),Context.sendOrderedBroadcast(), Context.sendStickyBroadcast()）发给 broadcast receivers。

一般情况下，Intent 对某操作的抽象描述包含下面几个部分。

- 对执行动作的描述：操作（action）。
- 对这次动作相关联的数据进行描述：数据（data）。
- 对数据类型的描述：数据类型（type）。
- 对执行动作的附加信息进行描述：类别（category）。
- 其他一切附加信息的描述：附件信息（extras）。
- 对目标组件的描述：目标组件（component）。

6.1.1 Intent 的 action 属性

action 是要执行的动作，也可以是在广播 Intent 中已发生且正被报告的动作。action 部分是一个字符串对象。它描述了 Intent 会触发的动作。Android 系统中已经预定义了一些 action 常量，可以参看 SDK 帮助文档，下表给出了一些标准的 action 常量，如表 6-1 所示。

表 6-1 SDK 中定义的标准动作

常量	目标组件	描述
ACTION_CALL	activity	初始化一个电话呼叫
ACTION_EDIT	activity	显示可供用户编辑的数据
ACTION_MAIN	activity	将该 Activity 作为 task 的第一个 Activity，没有数据输入，也没有数据返回
ACTION_SYNC	activity	使服务器上的数据与移动设备上数据同步
ACTION_BATTERY_LOW	broadcast receiver	提示电池电量低
ACTION_HEADSET_PLUG	broadcast receiver	提示耳机塞入或拔出
ACTION_SCREEN_ON	broadcast receiver	屏幕已点亮
ACTION_TIMEZONE_CHANGED	broadcast receiver	时区设置改变

问：除了 SDK 中定义的标准动作外，可以使用自定义动作吗？

答：当然，可以自定义动作。

自定义的动作在使用时，一般要加上包名作为前缀（为防止重复定义），如“com.example.project.SHOW_COLOR”，并可定义相应的 Activity 来处理自定义动作。

除上表介绍的 action 常量外，开发者也可以定义自己的 action 描述。一般来讲，定义自己的 action 字符串应该以应用程序的包名为前缀（防止重复定义）。由于 action 部分很大程度上决定了一个 Intent 的内容，特别是数据（data）和附加（extras）字段，就像一个方法名决定了参数和返回值。正是这个原因，应该尽可能明确指定动作，并紧密关联到其他 Intent 字段。即应该定义组件能够处理的 Intent 对象的整个协议，而不仅仅是单独地定义一个动作。一个 Intent 对象的动作通过 setAction()方法设置，通过 getAction()方法读取。

6.1.2 Intent 的 data 属性

data，即执行动作要操作的数据。

data 描述了 Intent 的动作所能操作数据的 MIME 类型和 URL，不同的 Action 用不同的操作数据。例如，如果 Activity 字段是 ACTION_EDIT，data 字段将显示包含用于编辑的文档的 URI；如果 Activity 是 ACTION_CALL，data 字段是一个 tel://URI 和将拨打的号码；如果 Activity 是 ACTION_VIEW，data 字段是一个 http://URI，接收活动将被调用去下载和显示 URI 指向的数据。在许多情况下，数据类型能够从 URI 中推测出来，特别是 content://URIs，它表示位于设备上的数据且被内容提供者（Content Provider）控制。但是类型也能够显示设置，setData()方法指定数据的 URI，setType()指定 MIME 类型，setDataAndType()指定数据的 URI 和 MIME 类型。通过 getData()读取 URI，getType()读取类型。

匹配一个 Intent 到一个能够处理 data 的组件，知道 data 的类型（它的 MIME 类型）和它的 URI 很重要。例如，一个组件能够显示图像数据就不应该被调用去播放音频文件。

6.1.3 Intent 的 type 属性

数据类型（type），显式指定 Intent 的数据类型（MIME）。一般 Intent 的数据类型能够根据数据本身进行判定，但是通过设置这个属性，可以强制采用显式指定的类型而不再进行推导。

6.1.4 Intent 的 category 属性

category（类别），被执行动作的附加信息。例如，LAUNCHER_CATEGORY 表示 Intent 的接受者应该在 Launcher 中作为顶级应用出现；而 ALTERNATIVE_CATEGORY 表示当前的 Intent 是一系列的可选动作中的一个，这些动作可以在同一块数据上执行。其他的如表 6-2 所示。

表 6-2 SDK 中定义的标准动作

常量	描述
CATEGORY_BROWSABLE	目标 Activity 可通过浏览器安全启动以显示一个链接相关的数据，如图片或邮件信息
CATEGORY_GADGET	Activity 可被嵌入另外一个拥有 gadget 的 Activity 中
CATEGORY_HOME	Activity 显示主页，即设备打开时用户看到的第一个界面或是用户按 Home 键时的界面
CATEGORY_LAUNCHER	Activity 是一个 task 的初始 Activity，是程序启动的高优先级 Activity
CATEGORY_PREFERENCE	目标 Activity 为 preference panel.

通过 addCategory()方法添加一个种类到 Intent 对象中；通过 removeCategory()方法删除一个之前添加的种类；通过 getCategories()方法获取 Intent 对象中的所有种类。

6.1.5 Intent 的 extras 属性

extras（附加信息）是一组键值对，包含了需要传递给目标组件并有其处理的一些附加信息。

就像动作关联的特定种类的数据 URIs，也关联到某些特定的附加信息。例如，一个 ACTION_TIMEZONE_CHANGE intent 有一个“time-zone”的附加信息，标识新的时区，ACTION_HEADSET_PLUG 有一个“state”附加信息，标识头部现在是否塞满或未塞满；有一个“name”

附加信息，标识头部的类型。如果你自定义了一个 SHOW_COLOR 动作，颜色值将可以设置在附加的键值对中。例如，如果要执行“发送电子邮件”这个动作，可以将电子邮件的标题、正文等保存在 extras 里，传给电子邮件发送组件。

Intent 有一系列 putXXX()方法用于插入各种附加数据，有一系列 getXXX()方法可以取出一系列数据。

使用 Extras 可以为组件提供扩展信息。

6.1.6 Intent 的 ComponentName 属性

ComponentName（组件），指定 Intent 的目标组件的类名称。ComponentName 包含两个 String 成员，分别代表组件的全称类名和包名，包名必须和 AndroidManifest.xml 文件标记中的对应信息一致。ComponentName 通过 setComponent()、setClass()或 setClassName()设置，通过 getComponent()读取。

通常 Android 会根据 Intent 中包含的其他属性的信息（如 action、data/type、category）进行查找，最终找到一个与之匹配的目标组件。但是，如果 ComponentName 这个属性有指定，将直接使用指定的组件，而不再执行上述查找过程。指定了这个属性以后，Intent 的其他所有属性都是可选的。

对于 Intent，组件名并不是必需的。如果一个 Intent 对象添加了组件名，则称该 Intent 为“显式 Intent”，这样的 Intent 在传递时会直接根据组件名去寻找目标组件。如果没有添加组件名，则称为“隐式 Intent”，Android 会根据 Intent 中的其他信息来确定响应该 Intent 的组件。

总之，action、data/type、category 和 extras 一起使系统能够理解诸如“查看某联系人的详细信息”或“给某人打电话”之类的短语。随着应用不断地加入系统中，Android 系统可以添加新的 action、data/type、category 来扩展功能。当然，最受益的还是应用本身，可以利用这套语言机制来处理不同的动作和数据。

6.2 系统标准 ActivityAction 应用

6.2.1 启动 Activity

在 Android 系统中，应用程序一般都有多个 Activity，Intent 可以实现不同 Activity 之间的切换和数据传递。

启动 Activity 方式有以下两种方式。

- 显式启动，必须在 Intent 中指明启动的 Activity 所在的类。
- 隐式启动，Android 系统根据 Intent 的动作和数据来决定启动哪一个 Activity，即在隐式启动时，Intent 中只包含需要执行的动作和所包含的数据，而无须指明具体启动哪一个 Activity，选择权由 Android 系统和最终用户来决定。

1. 显式启动

使用 Intent 显式启动 Activity，首先需要创建一个 Intent，指定当前的应用程序上下文及要启动的 Activity，并把创建好的 Intent 作为参数传递给 startActivity()方法。代码如代码清单 6-1 所示。

代码清单 6-1 显式启动

```
Intent intent = new Intent(IntentDemo.this, Activity2.class);
startActivity(intent);
```

以下将通过一个 IntentDemo 示例来详细讲解如何使用 Intent 显式启动新的 Activity。

IntentDemo 示例中包含 IntentDemo 和 Activity2 这两个 Activity 类，程序启动是默认启动 IntentDemo 这个 Activity，启动画面如图 6-1 所示。

在图 6-1 的界面中，单击“跳转到 Activity2”按钮后，程序启动 Activity2 这个 Activity，界面如图 6-2 所示。

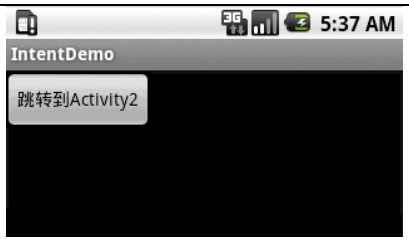


图 6-1　名为 IntentDemo 的 Activity 界面

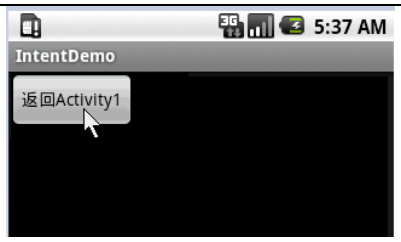


图 6-2　名为 Activity2 的 Activity 界面

为使程序达到上述效果，首先，在 AndroidManifest.xml 文件中注册上面两个 Activity，应使用<activity>标签，嵌套在<application>标签内部。代码如代码清单 6-2 所示。

代码清单 6-2　AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.IntentDemo"
  android:versionCode="1"
  android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
   <activity android:name=".IntentDemo"
       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=".Activity2"
       android:label="@string/app_name">
   </activity>
  </application>
<uses-sdk android:minSdkVersion="3" />
</manifest>
```

在 Android 应用程序中，用户使用的每个组件都必须在 AndroidManifest.xml 文件中的<application>节点内定义，所以<application>节点下共有两个<activity>节点，分别代表应用程序中所使用的两个 Activity：IntentDemo 和 Activity2。

在 IntentDemo.java 文件中，包含了显示使用 Intent 启动 Activity2 的核心代码，如代码清单 6-3 所示。

代码清单 6-3　IntentDemo.java

```
Button button = (Button)findViewById(R.id.btn);
button.setOnClickListener(new OnClickListener(){
 public void onClick(View view){
      Intent intent = new Intent(IntentDemo.this, Activity2.class);
      startActivity(intent);
 }
});
```

在单击事件的处理函数中，Intent 构造函数的第 1 个参数是应用程序上下文，应用程序上下文是 IntentDemo；第 2 个参数是接收 Intent 的目标组件，使用的是显式启动方式，直接指明了需要启动的 Activity。

同理，在 Activity2.java 文件中，包含了显示使用 Intent 启动 IntentDemo 的核心代码，如代码清单 6-4 所示。

代码清单 6-4　Activity2.java

```
Button button = (Button)findViewById(R.id.btn);
button2.setOnClickListener(new OnClickListener(){
 public void onClick(View view){
      Intent intent = new Intent(Activity2.this, IntentDemo.class);
      startActivity(intent);
 }
});
```

2．隐式启动

隐式启动 Activity 时，Android 系统在应用程序运行时解析 Intent，并根据一定的规则对 Intent 和 Activity 进行匹配，使 Intent 上的动作、数据与 Activity 完全吻合。而匹配的 Activity 可以是应用程序本身的，也可以是 Android 系统内置的，还可以是第三方应用程序提供的。因此，这种方式更强调了 Android 应用程序中组件的可复用性。

由此可以看出，隐式启动不需要指明需要启动哪一个 Activity，而由 Android 系统来决定，有利于使用第三方组件。

在默认情况下，Android 系统会调用内置的 Web 浏览器，代码如代码清单 6-5 所示。

代码清单 6-5　隐式启动默认情况调用内置 Web 浏览器

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.com"));
startActivity(intent);
```

在上述代码中，Intent 的动作是 Intent.ACTION_VIEW，是根据 Uri 的数据类型来匹配动作；数据部分的 Uri 是 Web 地址，使用 Uri.parse(urlString)方法，可以简单地把一个字符串解释成 Uri 对象。

Intent 的语法如代码清单 6-6 所示。

代码清单 6-6　Intent 语法

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(urlString));
```

Intent 构造函数的第 1 个参数是 Intent 需要执行的动作；第 2 个参数是 Uri，表示需要传递的数据。

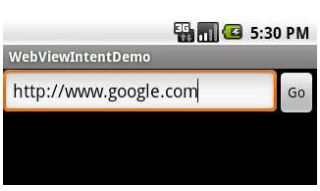
Android 系统支持的常见动作字符串常量表，如表 6-3 所示。

表 6-3　Android 系统支持的常见动作字符串常量表

动　作	说　明
ACTION_ANSWER	打开接听电话的 Activity，默认为 Android 内置的拨号盘界面
ACTION_CALL	打开拨号盘界面并拨打电话，使用 Uri 中的数字部分作为电话号码
ACTION_DELETE	打开一个 Activity，对所提供的数据进行删除操作
ACTION_DIAL	打开内置拨号盘界面，显示 Uri 中提供的电话号码
ACTION_EDIT	打开一个 Activity，对所提供的数据进行编辑操作
ACTION_INSERT	打开一个 Activity，在提供数据的当前位置插入新项
ACTION_PICK	启动一个子 Activity，从提供的数据列表中选取一项
ACTION_SEARCH	启动一个 Activity，执行搜索动作
ACTION_SENDTO	启动一个 Activity，向数据提供的联系人发送信息
ACTION_SEND	启动一个可以发送数据的 Activity
ACTION_VIEW	最常用的动作，对以 Uri 方式传送的数据，根据 Uri 协议部分以最佳方式启动相应的 Activity 进行处理。对于 http:address 将打开浏览器查看；对于 tel:address 将打开拨号呼叫指定的电话号码
ACTION_WEB_SEARCH	打开一个 Activity，对提供的数据进行 Web 搜索

以下将通过一个 WebViewIntentDemo 示例来了解如何隐式启动 Activity。

如图 6-3 所示，当用户在文本框中输入要访问的网址后，通过单击“Go”按钮，程序根据用户输入的网址生成一个 Intent，并以隐式启动的方式调用 Android 内置的 Web 浏览器，并打开指定的 Web 页面。本例输入的网址是 google 主页的主站地址，地址是：http://www.google.com。



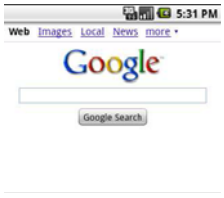


图 6-3　隐式启动 Activity

其中，main.xml 代码如代码清单 6-7 所示。

代码清单 6-7　main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="horizontal"
android:layout_width="fill_parent"
android:layout_height="fill_parent" >
<EditText
 android:id="@+id/url_field"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_weight="1.0"
 android:lines="1"
 android:inputType="textUri"
 android:imeOptions="actionGo" />
<Button
 android:id="@+id/go_button"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/go_button" />
</LinearLayout>
```

其中，strings.xml 代码如代码清单 6-8 所示。

代码清单 6-8　strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name"> WebViewIntentDemo </string>
<string name="go_button">Go</string>
</resources>
```

WebViewIntentDemo.java 代码如代码清单 6-9 所示。

代码清单 6-9　WebViewIntentDemo.java

```
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnKeyListener;
import android.widget.Button;
import android.widget.EditText;

public class WebViewIntentDemo extends Activity {
private EditText urlText;
private Button goButton;

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

 // Get a handle to all user interface elements
 urlText = (EditText) findViewById(R.id.url_field);
 goButton = (Button) findViewById(R.id.go_button);
```

```
    // Setup event handlers
    goButton.setOnClickListener(new OnClickListener() {
         public void onClick(View view) {
              openBrowser();
         }
    });

    urlText.setOnKeyListener(new OnKeyListener() {
         public boolean onKey(View view, int keyCode, KeyEvent event) {
              if (keyCode == KeyEvent.KEYCODE_ENTER) {
                   openBrowser();
                   return true;
              }
              return false;
         }
    });
  }
  }
```

在上述代码中第 26 行对按钮“Go”添加监听，当触摸单击或通过手机按键单击该按钮时，触发 openBrowser()方法。其中，openBrowser()方法代码如代码清单 6-10 所示。

代码清单 6-10　openBrowser()方法

```
/** Open a browser on the URL specified in the text box */
private void openBrowser() {
Uri uri = Uri.parse(urlText.getText().toString());
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
}
```

6.2.2　获取 Activity 返回值

在 6.2.1 节的 IntentDemo 示例中，通过使用 startActivity（Intent）方法启动 Activity 后，两个 Activity 之间相互独立，没有任何关联。然而，在很多情况下，后启动的 Activity 是为了让用户对特定信息进行选择，在关闭这个 Activity 后，用户的选择信息需要返回给未关闭的那个 Activity。由此，按照 Activity 启动的先后顺序，先启动的称为父 Activity，后启动的称为子 Activity，如果需要将子 Activity 的部分信息返回给父 Activity，则可以使用 Sub-Activity 的方式去启动子 Activity。

获取子 Activity 的返回值，一般分以下 3 个步骤。

1．以 Sub-Activity 的方式启动子 Activity

首先，调用 startActivityForResult(Intent, requestCode)函数，其中，参数 Intent 用于决定启动哪个 Activity，参数 requestCode 是唯一的标识子 Activity 的请求码。

显式启动子 Activity 的代码如代码清单 6-11 所示。

代码清单 6-11　显式启动子 Activity

```
int SUBACTIVITY1 = 1;
Intent intent = new Intent(this, SubActivity1.class);
startActivityForResult(intent, SUBACTIVITY1);
```

隐式启动子 Activity 的代码如代码清单 6-12 所示。

代码清单 6-12　隐式启动子 Activity

```
int SUBACTIVITY2 = 2;
Uri uri = Uri.parse("content://contacts/people");
Intent intent = new Intent(Intent.ACTION_PICK, uri);
startActivityForResult(intent, SUBACTIVITY2);
```

2．设置子 Activity 的返回值

在子 Activity 调用 finish()函数关闭前，调用 setResult()函数将所需的数据返回给父 Activity。其中，setResult()函数有两个参数：结果码和返回值。结果码表明了子 Activity 的返回状态，通常为 Activity.RESULT_OK 或 Activity.RESULT_CANCELED，或自定义的结果码，结果码均为整数类型；返回值封装在 Intent 中，子 Activity 通过 Intent 将需要返回的数据传递给父 Activity。数据主要是 Uri 形式，可以附加一些额外信息，这些额外信息用 Extra 的集合表示。

代码清单 6-13 所示代码说明了如何在子 Activity 中设置返回值。

代码清单 6-13　在子 Activity 中设置返回值

```
Uri data = Uri.parse("tel:" + tel_number);
Intent result = new Intent(null, data);
result.putExtra("address", " ");
setResult(RESULT_OK, result);
finish();
```

3．在父 Activity 中获取返回值

当子 Activity 关闭时，启动它的父 Activity 的 onActivityResult()函数将被调用；如果需要在父 Activity 中处理子 Activity 的返回值，则覆盖此函数即可。此函数的语法如代码清单 6-14 所示。

代码清单 6-14　onActivityResult()语法

```
public void onActivityResult(int requestCode, int resultCode, Intent data);
```

在上述代码中，第 1 个参数 requestCode，用来表示是哪一个子 Activity 的返回值；第 2 个参数 resultCode 用于表示子 Activity 的返回状态；第 3 个参数 data 是子 Activity 的返回数据，返回数据类型是 Intent。返回数据的用途不同，Uri 数据的协议也就不同。也可以使用 Extra 方法返回一些原始类型的数据。

代码清单 6-15 所示代码说明如何在父 Activity 中处理子 Activity 的返回值。

代码清单 6-15　在父 Activity 中处理子 Activity 的返回值

```
private static final int SUBACTIVITY1 = 1;
private static final int SUBACTIVITY2 = 2;

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data){
 Super.onActivityResult(requestCode, resultCode, data);
 switch(requestCode){
      case SUBACTIVITY1:
           if (resultCode == Activity.RESULT_OK){
                Uri uriData = data.getData();
           }else if (resultCode == Activity.RESULT_CANCEL){
           }
      break;
      case SUBACTIVITY2:
           if (resultCode == Activity.RESULT_OK){
                Uri uriData = data.getData();
           }
      break;
     }
 }
```

在上述代码中，第 1 行代码和第 2 行代码是两个子 Activity 的请求码；第 7 行代码对请求码进行匹配；第 9 行和第 11 行代码对结果码进行判断，如果返回的结果码是 Activity.RESULT_OK，则在代码的第 10 行使用 getData()函数获取 Intent 中的 Uri 数据；反之，若返回的结果码是 Activity.RESULT_CANCELED，则不进行任何操作。

ActivityCommunication 示例说明了如何以 Sub-Activity 方式启动子 Activity，以及使用 Intent 进行组件间的通信。

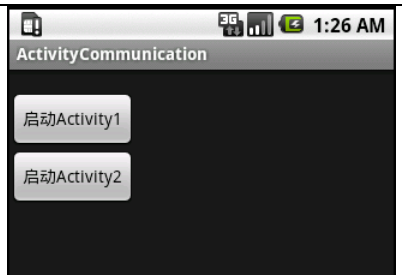


图 6-4　ActivityCommunication 主界面

在如图 6-4 所示的主界面中，当用户单击“启动 Activity1”和“启动 Activity2”按钮时，程序将分别启动两个不同的子 Activity：子 SubActivity1 和 SubActivity2，它们的界面分别如图 6-5 所示。

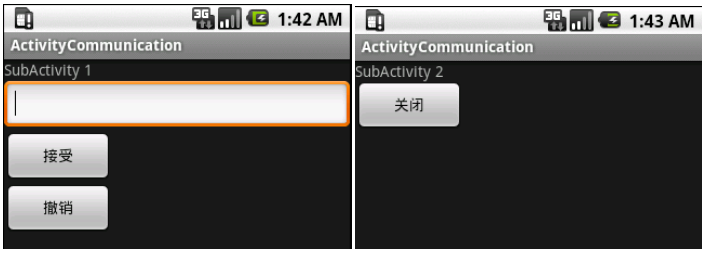


图 6-5　SubActivity1 和 SubActivity2 界面

SubActivity1 提供了一个输入框，以及“接受”和“撤销”两个按钮，如果在输入框中输入信息后单击“接受”按钮，程序会把输入框中的信息传递给父 Activity，并在父 Activity 界面上显示；如果用户单击“撤销”按钮，则程序不会向父 Activity 传递任何信息。

SubActivity2 主要为了说明如何在父 Activity 中处理多个子 Activity，因此仅提供了用于关闭 SubActivity2 的“关闭”按钮。

下面介绍 ActivityCommunication 的核心代码。

首先是 ActivityCommunication.java 文件，如代码清单 6-16 所示。

代码清单 6-16　ActivityCommunication.java

```
public class ActivityCommunication extends Activity {
  private static final int SUBACTIVITY1 = 1;
  private static final int SUBACTIVITY2 = 2;
  TextView textView;
  @Override
  public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
     textView = (TextView)findViewById(R.id.textShow);
     final Button btn1 = (Button)findViewById(R.id.btn1);
     final Button btn2 = (Button)findViewById(R.id.btn2);

     btn1.setOnClickListener(new OnClickListener(){
         public void onClick(View view){
         Intent intent = new Intent(ActivityCommunication.this, SubActivity1.class);
         startActivityForResult(intent, SUBACTIVITY1);
         }
      });

       btn2.setOnClickListener(new OnClickListener(){
          public void onClick(View view){
          Intent intent = new Intent(ActivityCommunication.this, SubActivity2.class);
          startActivityForResult(intent, SUBACTIVITY2);
          }
      });
   }

   @Override
     protected void onActivityResult(int requestCode, int resultCode, Intent data) {
```

```
          super.onActivityResult(requestCode, resultCode, data);

         switch(requestCode){
             case SUBACTIVITY1:
               if (resultCode == RESULT_OK){
                     Uri uriData = data.getData();
                     textView.setText(uriData.toString());
              }
              break;
              case SUBACTIVITY2:
              break;
        }
     }
}
```

在上述代码中，第 2 行和第 3 行分别定义了两个子 Activity 的请求码，而在第 16 行和第 23 行以 Sub-Activity 的方式分别启动两个子 Activity。

第 29 行代码是子 Activity 关闭后的返回值处理函数，其中 requestCode 是子 Activity 返回的请求码，应与第 2 行和第 3 行定义的两个请求码相匹配；resultCode 是结果码，在第 32 行代码对结果码进行判断，如果等于 RESULT_OK，则在第 35 行代码获取子 Activity 的返回值中的数据，其中，data 是返回值，子 Activity 需要返回的数据就保存在 data 中。

SubActivity1.java 的核心代码如代码清单 6-17 所示。

代码清单 6-17　SubActivity1.java

```
public class SubActivity1 extends Activity {
 @Override
 public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.subactivity1);
      final EditText editText = (EditText)findViewById(R.id.edit);
      Button btnOK = (Button)findViewById(R.id.btn_ok);
      Button btnCancel = (Button)findViewById(R.id.btn_cancel);

      btnOK.setOnClickListener(new OnClickListener(){
           public void onClick(View view){
                String uriString = EditText.getText().toString();
                Uri data = Uri.parse(uriString);
                Intent result = new Intent(null, data);
                setResult(RESULT_OK, result);
                finish();
           }
      });

      btnCancel.setOnClickListener(new OnClickListener(){
           public void onClick(View view){
                setResult(RESULT_CANCELED, null);
                finish();
           }
       });
  }
}
```

上述 SubActivity1.java 的核心代码中，第 13 行代码将 EditText 控件的内容作为数据保存在 Uri 中；第 14 行代码中使用这个 Uri 构造 Intent；第 15 行代码中，将 Intent 作为返回值，RESUIT_OK 作为结果码，通过调用 setResult()函数，将返回值和结果码传递给父 Activity；第 16 行代码调用 finish()函数关闭当前的子 Activity。

SubActivity2.java 的核心代码如代码清单 6-18 所示。

代码清单 6-18　SubActivity2.java

```
public class SubActivity2 extends Activity {
 @Override
 public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.subactivity2);

      Button btnReturn = (Button)findViewById(R.id.btn_return);
      btnReturn.setOnClickListener(new OnClickListener(){
           public void onClick(View view){
                setResult(RESULT_CANCELED, null);
                finish();
           }
      });
 }
}
```

上述 SubActivity2.java 的核心代码中，第 10 行的 setResult()函数仅设置了结果码，第 2 个参数为 null，表示数据需要传递给父 Activity。

6.3 Intent 过滤器

Intent 过滤器是一种根据 Intent 中的动作（Action）、类别（Categorie）和数据（Data）等内容，对适合接收该 Intent 的组件进行匹配和筛选的机制。它可以匹配数据类型、路径和协议，还可以用来确定多个匹配项顺序的优先级（Priority）。

应用程序的 Activity 组件、Service 组件和 BroadcastReceiver 都可以注册 Intent 过滤器,这些组件在特定的数据格式上就可以产生相应的动作。

6.3.1 注册 Intent 过滤器

注册 Intent 过滤器的方法如下：

在 AndroidManifest.xml 文件的各个组件的节点下定义<intent-filter>节点，然后在<intent-filter>节点中声明该组件所支持的动作、执行的环境和数据格式等信息。

<intent-filter>节点支持<action>标签、<category>标签和<data>标签，其中：<action>标签定义 Intent 过滤器的“类别”；<category>标签定义 Intent 过滤器的“动作”；<data>标签定义 Intent 过滤器的“数据”。

<intent-filter>节点支持的标签和属性如表 6-4 所示。

表 6-4 <intent-filter>节点支持的标签和属性

标 签	属 性	说 明
<action>	Android:name	指定组件所能响应的动作，用字符串表示，通常使用 Java 类名和包的完全限定名构成
<category>	Android:category	指定以何种方式去服务 Intent 请求的动作
<data>	Android:host	指定一个有效的主机名
	Android:mimetype	指定组件能处理的数据类型
	Android:path	有效的 Uri 路径名
	Android:port	主机的有效端口号
	Android:scheme	所需要的特定的协议

<category>标签用来指定 Intent 过滤器的服务方式，每个 Intent 过滤器可以定义多个<category>标签，程序开发人员可使用自定义的类别，或使用 Android 系统提供的类别。其中，Android 系统提供的类别如表 6-5 所示。

表 6-5　Android 系统提供的类别

值	说　明
ALTERNATIVE	Intent 数据默认动作的一个可替换的执行方法
SELECTED_ALTERNATIVE	和 ALTERNATIVE 类似，但替换的执行方法不是指定的，而是被解析出来的
BROWSABLE	声明 Activity 可以由浏览器启动

续表

值	说　明
DEFAULT	为 Intent 过滤器中定义的数据提供默认动作
HOME	设备启动后显示的第一个 Activity
LAUNCHER	在应用程序启动时首先被显示

AndroidManifest.xml 文件中的每个组件的<intent-filter>都被解析成一个 Intent 过滤器对象。当应用程序安装到 Android 系统时，所有的组件和 Intent 过滤器都会注册到 Android 系统中。这样，Android 系统便知道了如何将任意一个 Intent 请求通过 Intent 过滤器映射到相应的组件上。

6.3.2　Intent 解析

Intent 到 Intent 过滤器的映射过程称为“Intent 解析”。Intent 解析可以在所有组件中找到一个可与请求的 Intent 达成最佳匹配的 Intent 过滤器。

Intent 解析的匹配规则。

- Android 系统把所有应用程序包中的 Intent 过滤器集合在一起，形成一个完整的 Intent 过滤器列表。
- 在 Intent 与 Intent 过滤器进行匹配时，Android 系统会将列表中所有 Intent 过滤器的“动作”和“类别”与 Intent 进行匹配，任何不匹配的 Intent 过滤器都将被过滤掉。没有指定“动作”的 Intent 过滤器可以匹配任何的 Intent，但没有指定“类别”的 Intent 过滤器只能匹配没有“类别”的 Intent。
- 把 Intent 数据 Uri 的每个子部与 Intent 过滤器的<data>标签中的属性进行匹配，如果<data>标签指定了协议、主机名、路径名或 MIME 类型，那么这些属性都要与 Intent 的 Uri 数据部分进行匹配，任何不匹配的 Intent 过滤器均被过滤掉。
- 如果 Intent 过滤器的匹配结果多于一个，则可以根据在<intent-filter>标签中定义的优先级标签来对 Intent 过滤器进行排序，优先级最高的 Intent 过滤器将被选择。

在此以 6.2.1 节中隐式启动 Activity 的例子 WebViewIntentDemo 为基础，在 AndroidManifest.xml 文件中注册 Intent 过滤器，以及设置<intent-filter>节点属性来捕获指定的 Intent。

在 AndroidManifest.xml 中添加如代码清单 6-19 所示。

代码清单 6-19　AndroidManifest.xml 中添加代码

```
<intent-filter>
<action android:name="android.intent.action.VIEW" />
<data android:schema="http" />
</intent-filter>
```

利用 <intent-filter> 可以把应用程序的操作注册到系统中，当用户调用 Intent 时，可以根据输入的 ACTION 和 Uri 参数来找到这个应用程序。例如，在上述代码中，以 http 协议为例，打开 Google Map 可以用 URI: geo:38.899533,-77.036476。

此外，“协议”在 Android 里都可以随便定义。例如，写一个打开文件的关联 Intent，如 file:///sdcard/abc.txt。也可以用 type 进行关联，其代码如代码清单 6-20 所示。

代码清单 6-20 用 type 进行关联

```
<intent-filter>
<action android:name="android.intent.action.VIEW" />
<type android:value="test.item"/>
</intent-filter>
```

然后，通过代码清单 6-21 所示代码就能定位到应用。

代码清单 6-21 用 type 定位到应用代码

```
Intent it = new Intent(Intent.ACTION_VIEW);
it.setType("test.item”);
startActivity(it)
```

6.4 广播消息

应用程序和 Android 系统都可以使用 Intent 发送广播消息。其中，广播消息的内容可以是与应用程序密切相关的数据信息，也可以是 Android 的系统信息，例如，网络连接变化、电池电量变化、接收短信和系统设置变化等。如果应用程序注册了 BroadcastReceiver，则可以接收到指定的广播消息。

下面将介绍广播信息的使用方法。

首先，创建一个 Intent。调用 sendBroadcast()函数，就可把 Intent 携带的消息广播出去，如果要在 Intent 传递额外数据，可以用 Intent 的 putExtra()方法。

注意：

在构造 Intent 时必须用全局唯一的字符串标识其要执行的动作，通常使用应用程序包的名称。

利用 Intent 发送广播消息，并添加了额外的数据，然后调用 sendBroadcast()发送广播消息的代码如代码清单 6-22 所示。

代码清单 6-22 利用 Intent 发送广播消息

```
String UNIQUE_STRING = "com.example.BroadcastReceiverDemo";
Intent intent = new Intent(UNIQUE_STRING);
intent.putExtra("key1", "value1");
intent.putExtra("key2", "value2");
sendBroadcast(intent);
```

广播消息发送后，利用 BroadcastReceiver 监听广播消息。具体方法如下：在 AndroidManifest.xml 文件或在代码中注册一个 BroadcastReceiver，并在其中使用 Intent 过滤器指定要处理的广播消息。在 BroadcastReceiver 接收到与之匹配的广播消息后，onReceive()方法会被调用；onReceive()方法必须要在 5 秒内执行完毕，否则 Android 系统会认为该组件失去响应，并提示用户强行关闭该组件。

创建 BroadcastReceiver 需继承 BroadcastReceiver 类，并重载 onReceive()方法。代码如代码清单 6-23 所示。

代码清单 6-23 利用 BroadcastReceiver 监听广播消息

```
public class MyBroadcastReceiver extends BroadcastReceiver {
 @Override
 public void onReceive(Context context, Intent intent) {
      //TODO: React to the Intent received.
 }
}
```

BroadcastReceiver 的应用程序不需要一直运行，当 Android 系统接收到与之匹配的广播消息时，会自动启动此 BroadcastReceiver。基于以上特征，BroadcastReceiver 适合做一些资源管理的工作。

如图 6-6 所示，BroadcastReceiverDemo 示例说明了如何在应用程序中注册 BroadcastReceiver，并接收指定类型的广播消息。

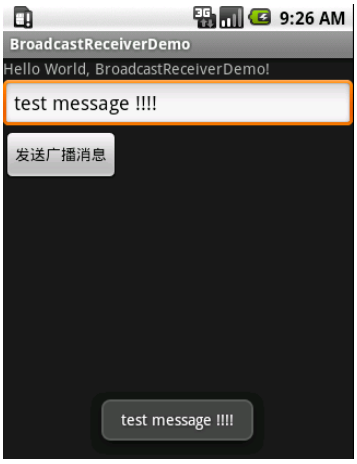


图 6-6　BroadcastReceiverDemo 示例

如图 6-6 所示，在单击“发送广播消息”按钮后，EditText 控件中内容将以广播消息的形式发送出去，示例内部的 BroadcastReceiver 将接收这个广播消息，并显示在用户界面的下方。

BroadcastReceiverDemo.java 文件中包含发送广播消息的代码，其关键代码如代码清单 6-24 所示。

代码清单 6-24　BroadcastReceiverDemo.java

```
button.setOnClickListener(new OnClickListener(){
 public void onClick(View view){
      Intent intent = new Intent("com.example.BroadcastReceiverDemo");
      intent.putExtra("message", entryText.getText().toString());
      sendBroadcast(intent);
 }
});
```

在上述代码中，第 3 行代码创建 Intent，将 com.example.BroadcastReceiverDem 作为识别广播消息的字符串标识；第 4 行代码添加了额外信息；第 5 行代码调用 sendBroadcast()函数发送广播消息。

为了能够使应用程序中的 BroadcastReceiver 接收指定的广播消息，首先要在 AndroidManifest.xml 文件中添加 Intent 过滤器，声明 BroadcastReceiver 可以接收的广播消息。其中，AndroidManifest.xml 文件的完整代码如代码清单 6-25 所示。

代码清单 6-25　AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
 package="com.example.BroadcastReceiverDemo"
 android:versionCode="1"
 android:versionName="1.0">
 <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".BroadcastReceiverDemo"
                android:label="@string/app_name">
           <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
      </activity>
      <receiver android:name=".MyBroadcastReceiver">
          <intent-filter>
               <action android:name="com.example.BroadcastReceiverDemo" />
          </intent-filter>
     </receiver>
 </application>
 <uses-sdk android:minSdkVersion="3" />
</manifest>
```

在上述代码中，第 14 行代码中创建了一个<receiver>节点；在第 15 行中声明了 Intent 过滤器的动作为“com.example.BroadcastReceiverDemo”，这与 BroadcastReceiverDemo.java 文件中 Intent 的动作相一致，表明这个 BroadcastReceiver 可以接收动作为“com.example.
BroadcastReceiverDemo”的广播消息。

MyBroadcastReceiver.java 文件创建了一个自定义的 BroadcastReceiver，其核心代码如代码清单 6-26 所示。

代码清单 6-26　MyBroadcastReceiver.java

```
public class MyBroadcastReceiver extends BroadcastReceiver {
 @Override
 public void onReceive(Context context, Intent intent) {
      String msg = intent.getStringExtra("message");
      Toast.makeText(context, msg, Toast.LENGTH_SHORT).show();
 }
}
```

在上述代码中，第 1 行代码首先继承了 BroadcastReceiver 类；第 3 行代码重载了 onReceive()函数，当接收到 AndroidManifest.xml 文件定义的广播消息后，程序将自动调用 onReceive()函数。

6.5　本章小结

本章主要对 Android 组件间的通信进行学习，包括 Intent 进行组件通信的原理、Intent 启动 Activity 的方法、获取 Activity 返回值的方法、Intent 过滤器的原理及其匹配机制、发送和接收广播消息的方法。

关键知识点测评

1．以下有关 Intent 的说法，不正确的一个是（　　）。

A．Intent 不可以用于应用程序之间交互

B．Intent 可以通过 Context.startActivity()启动一个 Activity

C．Intent 可以启动 Activity 和 Service，在 Android 系统上发布广播消息

D．Intent 描述了动作的产生组件、接收组件和传递的数据信息

2．以下有关 Intent 的叙述，正确的一个是（ ）。

A．Intent 显式启动时，构造函数的第 2 个参数是应用程序上下文

B．隐式启动 Activity 时，匹配的 Activity 不可以是第三方应用程序提供的

C．Android 系统可以在 Intent 中指明启动的 Activity 所在的类，也可以根据 Intent 的动作和数据来决定启动哪一个 Activity

D．只有 Activity 组件可以注册 Intent 过滤器

联系方式

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-22193762

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218