第 7 天 Android 基础

第七章 BroadCastReceiver

- BroadCastReceiver 的使用
- 发送无序广播
- 发送有序广播
- 特殊的广播

1.1 BroadCastReceiver 基本概念

在 Android 中，Broadcast 是一种广泛运用的在应用程序之间传输信息的机制。而 BroadcastReceiver 是对发送出来的 Broadcast 进行过滤接受并响应的一类组件，是 Android 四大组件之一。

广播接收者（BroadcastReceiver）用于接收广播的，广播的发送是通过调用 sendBroadcast（Intent）/sendOrderedBroadcast（Intent）来实现的。通常一个广播可以被多个广播接收者所接收。

广播被分为两种不同的类型：“普通广播（Normal Broadcasts）”也叫无序广播和“有序广播（Ordered Broadcasts）”。

1、普通广播是完全异步（就是不会被某个广播接收者终止）的，可以在同一时刻（逻辑上）被所有接收者接收到（其实被接收者接收到也是由顺序的，接收者配置的优先级越高，越先接收到，也就是说广播接收者的优先级对于无序广播也是有用的），消息传递的效率比较高，但缺点是：接收者不能将处理结果传递给下一个接收者，并且无法终止广播的传播。

2、有序广播是按照接收者声明的优先级别，被接收者依次接收广播。如：A 接收者的级别高于 B，B 的级别高于 C，那么，广播先传给 A，再传给 B，最后传给 C 。在传递的过程中如果有某个接收者终止（abortBroadCast）了该广播，那么后面的接收者就接收不到该广播。

3、广播接收者属于四大组件之一，因此通常需要在 AndroidManifest.xml 中进行注册，优先级别声明在 intent-filter 元素的 android:priority 属性中，数越大优先级别越高，取值范围:-1000 到 1000，优先级别也可以调用 IntentFilter 对象的 setPriority()进行设置。

4、有序广播的接收者可以终止广播的传播，广播的传播一旦终止，后面的接收者就无法接收到广播，有序广播的接收者可以将数据传递给下一个接收者，如：A 得到广播后，可以往它的结果对象中存入数据，当广播传给 B 时，B 可以从 A 的结果对象中得到 A 存入的数据。

5、Context.sendBroadcast() 发送的是普通广播，所有订阅者都有机会获得并进行处理。

6、Context.sendOrderedBroadcast() 发送的是有序广播，系统会根据接收者声明的优先级别按顺序逐个执行接收者，前面的接收者有权终止广播(BroadcastReceiver.abortBroadcast())，如果广播被前面的接收者终止，后面的接收者就再也无法获取到广播。对于有序广播，前面的接收者可以将数据通过 setResultExtras(Bundle)方法存放进结果对象，然后传给下一个接收者，下一个接收者通过代码：Bundle bundle = getResultExtras(true))可以获取上一个接收者存入在结果对象中的数据。

1.2 Android 系统常见的广播

Android 为了将系统运行时的各种“事件”通知给其他应用（或者说通知给我们程序员，让我们程序员好做出相应的反应。举个生活中的例子：比如我们坐火车，当前方到达某站的时候，火车乘务员会给所有乘客发送即将到站的广播，这样乘客收到广播后就可以提前准备下车），因此内置了多种广播，比如：系统电量的改变、屏幕的锁屏、网络状态的改变、接收到新的短信、拨打电话事件、sdcard 的挂载和移除、应用的安装和卸载等等。比如我们开发的在线播放视频类的 APP，那么我们就有必要监听网络转态改变的事件广播，如果用户的网络状态从 wifi 改变为了 4G 上网，那么应该提示用户是否使用 4G 网络继续播放视频，如果不提示用户，那么就可能导致用户流量被大量使用，一会儿功夫，用户可能就要停机了。

接下来我们会用 4 个案例来演示广播接收者的使用。

1.2.1 案例-IP 拨号器

一、需求

什么是 IP 拨号服务？我们为什么要用 IP 服务？所谓的 IP 拨号就是通过接入数据网络来传播语音信息。IP 拨号的目的在于转接至其他频道，减少话费等用处。移动 17951，联通 17911，打长途时在电话号码前加上这个就便宜了，如果你的手机上有这个键的话，那么打电话时输入长途电话号码后，直接按那个键就拨出去了，它会自动加上 IP。通俗的说就是打长途便宜。

例如手机拨打长途电话：

移动拨区号+电话号=0.25/分市话+0.7/分长途=0.95/分；

移动拨 17951+区号+电话号=0.25/分市话+0.3/分长途=0.55/分。

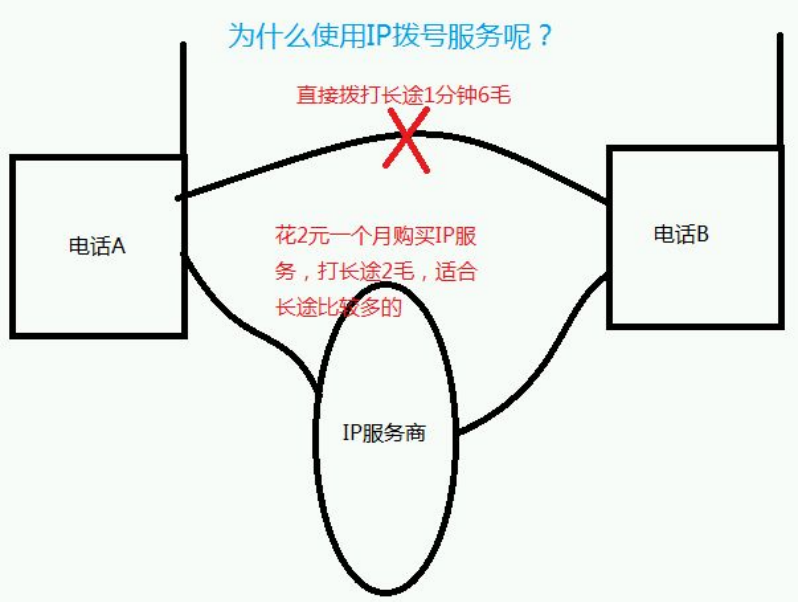

图 1-1　　IP 播放器原理

了解了 IP 拨号的用途之后，接下来，我们通过程序在用户拨出去的号码前自动加上一个 IP 号码，为用户省钱。

之所以能实现这样的功能，是因为拨号的时候 Android 系统会发送一个有序广播，该广播中携带了用户拨打的号码，我们通过注册广播接收者就可以获取到该广播，同时将该广播中的数据进行修改。从而实

现了用户号码自动加 IP 号的功能。

为了能让用户自己决定 IP 号码，我们需要一个界面（如图 1-2），让那个用户输入 IP 号码，然后将该 IP 号码保存到 SharedPreferences 中。

图 1-2　　IP 拨号器界面

二、布局

图 1-2 所示的布局代码如下所示。

【文件 1-1】 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.    <EditText
8.       android:id="@+id/et_ip"
9.       android:hint="请输入 IP 号码，默认 17951"
10.       android:layout_width="match_parent"
11.       android:layout_height="wrap_content" />
12.    <Button
13.       android:onClick="saveIP"
14.       android:layout_width="wrap_content"
15.       android:layout_height="wrap_content"
16.       android:text="保存"
17.       />
18.
19. </LinearLayout>
20.
```

三、代码

在该案例中总共用到了两个类一个是 MainActivity.java 负责让用户输入 IP 号码，另外一个是自定义的

广播接收者 IPCallerReceiver 负责监听用户的拨打电话事件。

【文件 1-2】 MainActivity.java

```
1. package com.itheima.android.ipcaller;
2.
3. import android.os.Bundle;
4. import android.app.Activity;
5. import android.content.SharedPreferences;
6. import android.text.TextUtils;
7. import android.view.View;
8. import android.widget.EditText;
9. import android.widget.Toast;
10. /**
11.  * 保存用户的 IP 号码
12.  *
13.  * @author wzy 2015-11-18
14.  *
15.  */
16. public class MainActivity extends Activity {
17.
18. private EditText et_ip;
19. private SharedPreferences sp;
20. @Override
21. protected void onCreate(Bundle savedInstanceState) {
22.     super.onCreate(savedInstanceState);
23.     setContentView(R.layout.activity_main);
24.     //文本编辑控件
25.     et_ip = (EditText) findViewById(R.id.et_ip);
26.     //获取 sp 对象
27.     sp = getSharedPreferences("config", MODE_PRIVATE);
28. }
29. //保存 IP 号码
30. public void saveIP(View view){
31.     String ipNum = et_ip.getText().toString().trim();
32.     //如果为空则保存默认值
33.     if (TextUtils.isEmpty(ipNum)) {
34.         sp.edit().putString("ip", "17951").commit();
35.     }else {
36.         sp.edit().putString("ip", ipNum).commit();
37.     }
38.     Toast.makeText(this, "IP 号码保存成功", Toast.LENGTH_SHORT).show();
39. }
40.
41. }
42.
```

编写自定义广播接收者需要自定义一个类然后继承系统提供的 BroadCastReceiver 类，然后覆写抽象方

法 onReceive。

【文件 1-3】 IPCallerReceiver.java

```
1. package com.itheima.android.ipcaller;
2.
3. import android.content.BroadcastReceiver;
4. import android.content.Context;
5. import android.content.Intent;
6. import android.content.SharedPreferences;
7. import android.text.TextUtils;
8. import android.util.Log;
9. /**
10.  * 自定义广播接收者
11.  *
12.  * @author wzy 2015-11-18
13.  *
14.  */
15. public class IPCallerReceiver extends BroadcastReceiver {
16.
17. @Override
18. public void onReceive(Context context, Intent intent) {
19.     //获取数据
20.     String resultData = getResultData();
21.     Log.d("tag", "接收到广播："+resultData);
22.     //从 SharedPreferences 中获取用户保存的 IP 号码
23.     SharedPreferences sp =
24. context.getSharedPreferences("config", Context.MODE_PRIVATE);
25.     String ipNum = sp.getString("ip", "17951");
26.     if (!TextUtils.isEmpty(ipNum)) {
27.         //修改数据
28.         resultData = ipNum+resultData;
29.     }
30.     //将修改后的数据设置出去
31.     setResultData(resultData);
32. }
33.
34. }
```

四、在清单文件中进行注册

广播是 Android 四大组件之一，因此需要在 AndroidManifest.xml 中进行注册。同时监听用户的拨打电话行为也属于侵犯用户隐私的行为，因此需要添加权限。

1、注册广播

【文件 1-4】 注册广播

```
1.  <receiver android:name="com.itheima.android.ipcaller.IPCallerReceiver">
2.          <intent-filter >
3.      <action android:name="android.intent.action.NEW_OUTGOING_CALL"></action>
4.          </intent-filter>
5.       </receiver>
```

大家可以发现广播接收者的注册也需要通过 intent-filter 来监听特定的广播，如果是监听 Android 系统的，那么在 action 中就需要配置系统提供的常量。如果监听自定义发送的广播，那么就需要配置自定义广播设置的 action。

2、声明权限

【文件 1-5】 添加权限

```
1. <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
```

五、总结和注意事项

通过这个案例大家是不是已经发现实现一个广播接收者是如此的简单。上面代码启动起来后，在如图 1-3 界面输入 17951，然后点击保存。之后拨打电话，那么我们想要的效果就出现了，如图 1-4。我拨打的是 110，那么拨打出去后就成了 17951110 了。

图 1-3　　程序主界面

图 1-4　　运行效果

实现一个广播的简单步骤如下：

1、自定义一个类继承 BroadCastReceiver

2、覆写 BroadCastReceiver 类中的 onReceive（Content，Intent）方法，在该方法中实现我们的业务逻辑。

```
    @Override
    public void onReceive(Context context, Intent intent) {
        //获取数据
        String resultData = getResultData();
        Log.d("tag", "接收到广播："+resultData);
        //从 SharedPreferences 中获取用户保存的 IP 号码
        SharedPreferences sp =
context.getSharedPreferences("config", Context.MODE_PRIVATE);
        String ipNum = sp.getString("ip", "17951");
        if (!TextUtils.isEmpty(ipNum)) {
            //修改数据
            resultData = ipNum+resultData;
        }
        //将修改后的数据设置出去
        setResultData(resultData);
    }
```

3、在 AndroidManifest.xml 中注册我们编写的广播接收者

```
    <receiver android:name="com.itheima.android.ipcaller.IPCallerReceiver">
          <intent-filter >
             <action android:name="android.intent.action.NEW_OUTGOING_CALL"></action>
          </intent-filter>
     </receiver>
```

4、在 AndroidManifest.xml 中声明用到的权限（如果需要的话，该步骤是可选的）

```
    <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
```

注意：

在使用上面案例的时候需要注意的事项比较多：

1、要想让我们的广播接收者生效，必须将我们的应用启动一次

这一点儿我们可能感觉不到，是因为当我们给模拟器安装应用的时候，系统会将其自动启动。

2、上面的代码在部分国产手机上是无法达到效果（比如我正在使用的魅族 note2）的，因为国产手机已经修改了 Android 系统源码，将该功能给阉割了（暂且认为是为了更安全吧）。

3、当我们的应用程序退出后，只要有广播进来，那么我们应用的进程会被系统自动启动起来，这也是我们的部分应用实现开机启动的原理，因为开机事件也会发送一个广播，我们只需要监听该广播即可。

1.2.2 案例-短信监听器

一、需求

系统接收到短信时会将该事件以有序广播（部分自定义的 ROM 可能已经修改了这个策略，比如：小米的 MIUI 系统）的形式发送出去，因此我们只需要自定义一个 BroadCastReceiver 监听该广播（android.provider.Telephony.SMS_RECEIVED）即可监听到短信的到来。由于该广播是有序的，因此如果将我们自定义的 BroadCastReceiver 配置了较高的优先级，那么我们就能先于系统短信 app 接收到该广播，然后终止该广播，从而就实现了短信拦截功能。

通过该案例我们可以学到：

1、什么是有序广播？

2、如何终止有序广播

3、如何从广播中获取短信

4、广播的优先级概念

在该案例中我们要做一个类似短信黑名单的应用，主界面提供一个 EditText 和一个 Button，让用户输入一个“黑名单”，点击保存之后，如果该号码发短信过来，那么我们的应用就将其拦截。

图 1-5　短信监听器界面&项目结构图

二、布局

布局界面很简单，如【文件 1-6】所示。

【文件 1-6】 activity_main.xml

```
1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     android:layout_width="match_parent"
3.     android:layout_height="match_parent"
4.     android:orientation="vertical">
```

```
5.
6.    <EditText
7.        android:id="@+id/et_num"
8.        android:layout_width="match_parent"
9.        android:layout_height="wrap_content"
10.        android:hint="请输入要拦截的号码" />
11.    <Button
12.        android:onClick="save"
13.        android:layout_width="wrap_content"
14.        android:layout_height="wrap_content"
15.        android:text="确定"
16.        ></Button>
17.
18. </LinearLayout>
```

三、代码

在该工程中总共用到了两个类，一个是主界面对应的 MainActivity，另外一个是广播接收者 SmsListenerReceiver。

【文件 1-7】 MainActivity.java

```
1. package com.itheima.smslistener;
2.
3. import android.os.Bundle;
4. import android.text.TextUtils;
5. import android.view.View;
6. import android.widget.EditText;
7. import android.widget.Toast;
8. import android.app.Activity;
9. import android.content.SharedPreferences;
10. /**
11.  * 让用户输入要拦截的号码
12.  *
13.  * @author wzy 2015-11-19
14.  *
15.  */
16. public class MainActivity extends Activity {
17.
18. private EditText et_num;
19. private SharedPreferences sp;
20.
21. @Override
22. protected void onCreate(Bundle savedInstanceState) {
```

```
23.     super.onCreate(savedInstanceState);
24.     setContentView(R.layout.activity_main);
25.     et_num = (EditText) findViewById(R.id.et_num);
26.     sp = getSharedPreferences("config", MODE_PRIVATE);
27. }
28. /**
29.  * 保存用户输入的号码
30.  * @param view
31.  */
32. public void save(View view){
33.     String num = et_num.getText().toString().trim();
34.     if (TextUtils.isEmpty(num)) {
35.         Toast.makeText(this, "请输入要拦截的号码！", Toast.LENGTH_SHORT).show();
36.         return;
37.     }
38.     //保存用户输入的号码
39.     sp.edit().putString("blackNum", num).commit();
40.     Toast.makeText(this, "黑名单已设置："+num, Toast.LENGTH_SHORT).show();
41. }
42. }
43.
```

上面代码的业务逻辑比较简单，当用户点击保存按钮的时候，判断用户是否已经输入黑名单号码，如果黑名单号码不为空则将该号码保存到 sp 文件中。

【文件 1-8】 SmsListenerReceiver.java

```
1. package com.itheima.smslistener;
2.
3. import android.content.BroadcastReceiver;
4. import android.content.Context;
5. import android.content.Intent;
6. import android.content.SharedPreferences;
7. import android.telephony.SmsMessage;
8. import android.util.Log;
9. /**
10.  * 自定义广播接收者
11.  *
12.  * @author wzy 2015-11-19
13.  *
14.  */
15. public class SmsListenerReceiver extends BroadcastReceiver {
16.
17. @Override
18. public void onReceive(Context context, Intent intent) {
19.     //从 sp 中获取保存的黑名单
20.     SharedPreferences sp =
21. context.getSharedPreferences("config", Context.MODE_PRIVATE);
```

```
22.     /*
23.      * 获取短信数据
24.      * 短信数据获取是固定的写法
25.      * 返回值是 Object[]，每个对象代表一个短信
26.      */
27.     Object[] objects = (Object[]) intent.getExtras().get("pdus");
28.     for(Object obj : objects){
29.         /*
30.          * 使用 SmsMessage 创建短信对象
31.          * 这里需要将 obj 强转为 byte[]数组
32.          */
33.         SmsMessage message = SmsMessage.createFromPdu((byte[]) obj);
34.         //获取短信发送人号码
35.         String address = message.getOriginatingAddress();
36.         //获取短信内容
37.         String body = message.getMessageBody();
38.         //从 sp 中获取黑名单
39.         String num = sp.getString("blackNum", "");
40.         //如果该短信号码跟黑名单相匹配则拦截
41.         if (address.equals(num)) {
42.             //终止该广播继续往下传递
43.             abortBroadcast();
44.             Log.d("tag", "拦截了一条短信："+"address="+address+"/body="+body);
45.         }
46.         Log.d("tag", "收到一条短信："+"address="+address+"/body="+body);
47.     }
48. }
49.
50. }
```

在上面的代码中使用了比较多的新 API。

1、从 Intent 中获取短信内容

```
Object[] objects = (Object[]) intent.getExtras().get("pdus");
for(Object obj : objects){
    /*
     * 使用 SmsMessage 创建短信对象
     * 这里需要将 obj 强转为 byte[]数组
     */
    SmsMessage message = SmsMessage.createFromPdu((byte[]) obj);
    //获取短信发送人号码
    String address = message.getOriginatingAddress();
    //获取短信内容
    String body = message.getMessageBody();
}
```

注：协议数据单元（Protocol Data Unit）是指对等层次之间传递的数据单位。在 Android 里用作短信的

传输协议。

因为短信可能会有多段（如果一条的短信内容超过一定的长度，比如在中国超过 70 个汉字，那么该短信就会被拆分多段分条发送），因此上面的代码需要使用 for 循环进行遍历。

2、终止一个有序广播

```
abortBroadcast();
```

终止有序广播只需要一句代码，该代码是 BroadCastReceiver 类中的方法，因此这里可以直接使用。这里需要注意的是如果 abortBroadCast 是在一个有序广播中执行的，那么就会报如下异常：

```
11-19 06:58:21.440: E/BroadcastReceiver(1705): BroadcastReceiver trying to return result during a non-ordered broadcast
11-19 06:58:21.440: E/BroadcastReceiver(1705): java.lang.RuntimeException: BroadcastReceiver trying to return result during a non-ordered broadcast
11-19 06:58:21.440: E/BroadcastReceiver(1705): at android.content.BroadcastReceiver.checkSynchronousHint(BroadcastReceiver.java:771)
11-19 06:58:21.440: E/BroadcastReceiver(1705): at android.content.BroadcastReceiver.abortBroadcast(BroadcastReceiver.java:682)
```

注意：在低版本的手机上比如 Android2.3 上是不会报这样的异常的。

为了防止我们终止一个无序广播导致报异常，我们可以先判断接收到的广播类型。优化后的代码如下：

```
if (isOrderedBroadcast()) {
        abortBroadcast();
        Log.d("tag", "拦截了一条短信："+"address="+address+"/body="+body);
}
```

isOrderedBroadcast 方法是 BroadCastReceiver 类提供的，用于判断当前的广播类型。返回 true 为有序广播，返回 false 为无序广播。

四、在清单文件中进行注册

1、注册 BroadCastReceiver

【文件 1-9】 注册 BroadCastReceiver

```
<receiver android:name="com.itheima.smslistener.SmsListenerReceiver" >
    <intent-filter android:priority="1000">
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
    </intent-filter>
 </receiver>
```

注意：在上面的 intent-filter 标签中配需了 android:priority="1000"属性和值，该属性用于配置广播接收者的优先级，该值越大代表优先级越高，越早接收到广播。一般认为优先级的范围为[-1000,1000]，但是如果你配置一个大于 1000 的优先级也是生效的。

在这里将优先级配置为 1000 很重要，这样可以保证我们的广播接收者第一个接收到广播，这样赶在系统短信应用收到之前将短信拦截掉。

2、声明权限

因为拦截短信是赤裸裸的侵犯用户隐私行为，因此必须声明权限才行。

【文件 1-10】 添加权限

```
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
```

五、总结

运行效果如图 1-6 和 1-7 所示。

图 1-6　　设置黑名单

Tag	Text
tag	拦截了一条短信：address=5556/body=123143142
tag	收到一条短信：address=5556/body=123143142

图 1-7　　日志输出

我们已经成功“拦截”了 5556 发送过来的短信了！但是这里依然需要注意的是上面的运行效果是在我的 Android4.1 模拟器上实现了，如果你使用的非官方的 ROM，比如 MIUI，那么可能是无权拦截短信的。监听短信实现步骤总结如下：

1、编写 activity_main.xml 和 MainActivity 代码

在 MainActivity 代码中保存用户输入的黑名单。

2、自定义 BroadCastReceiver 类

自定义 BroadCastReceiver 类，在该类中判断短信的号码是否跟黑名单一致，如果一致则进行拦截。

3、注册 BroadCastReceiver

因为我们是为了拦截短信，因此一定要加上优先级，而且需要配置一个较高的优先级，建议配置为 1000。

4、声明权限

1.2.3 案例-监听应用的安装和卸载

一、需求

在 Android 系统中，安装应用和卸载应用事件也都会发送特定的广播，我们可以通过监听这些广播间接获取到用户新安装了什么软件，卸载了哪些软件，进而可以统计用户的偏好，或统计某个软件的存留率。

需求很简单，监听应用的安装和卸载，并将其报名打印出来即可。

该应用不需要界面，代码也很简单，只需要一个自定义广播接收这就可以了。

二、代码

【文件 1-11】 PackageStateReceiver .java

```
1. package com.itheima.packages.receiver;
2.
3. import android.content.BroadcastReceiver;
4. import android.content.Context;
5. import android.content.Intent;
6. import android.util.Log;
7. /**
8.  * 监听应用的安装和卸载
9.  *
10.  * @author wzy 2015-11-19
11.  *
12.  */
13. public class PackageStateReceiver extends BroadcastReceiver {
14.
15. @Override
16. public void onReceive(Context context, Intent intent) {
17.     String packageName = intent.getDataString();
18.     if (intent.getAction().equals(Intent.ACTION_PACKAGE_ADDED)) {
19.         Log.d("tag", "安装了：+packageName);
20.     }else if (intent.getAction().equals(Intent.ACTION_PACKAGE_REMOVED)) {
21.         Log.d("tag", "卸载了：+packageName);
22.     }
23. }
24. }
25.
```

三、注册 BroadCastReceiver

【文件 1-12】　AndroidManifest.xml 片段

```
1.  <receiver android:name="com.itheima.packages.receiver.PackageStateReceiver">
2.          <intent-filter >
3.              <data android:scheme="package"/>
4.              <action android:name="android.intent.action.PACKAGE_ADDED"/>
5.              <action android:name="android.intent.action.PACKAGE_REMOVED"/>
6.          </intent-filter>
7.  </receiver>
```

监听应用的安装和卸载不需要额外声明权限。因此只需要在 AndroidManifest.xml 中注册 BroadCastReceiver 即可。

在上面配置的 intent-filter 标签中，必须添加 data 子标签才能接收到广播。

四、总结

运行上面的代码，我分别卸载和安装了一个应用，日志输入如下图

Application	Tag	Text
com.itheima.packages.receiver	tag	卸载了：package:com.itheima.smslistener
com.itheima.packages.receiver	tag	安装了：package:com.itheima.android.ipcaller

图 1-8　监听应用安装和卸载日志

1.3 发送无序广播

无序广播不可以被拦截，在高版本的系统上拦截会报异常。所有接收无序广播的广播接收者在此广播被发送时均能接收到此广播无序广播使用 Context.sendBroadcast 方法来发送：无序广播的实现比较简单，因此这里只给出核心代码。

【文件 1-13】　发送无序广播代码片段

```
1. public void sendBroadcast(View view){
8.      //定义一个意图
9.      Intent intent = new Intent();
10.     //设置 Action
11.     intent.setAction("com.itheima.broadcast");
12.     //绑定数据
13.     intent.putExtra("data", "我是无序广播数据");
14.     //发送无序广播
15.     sendBroadcast(intent);
16. }
```

接收我们自定义无序广播的代码也很简单，主要需要两个步骤：

1、编写自定义 BroadCastReceiver

【文件 1-14】 自定义广播接收者

```
1. public class MyReceiver extends BroadcastReceiver {
2.
3.  @Override
4.  public void onReceive(Context context, Intent intent) {
5.      String msg =
6.  intent.getAction()+"/"+intent.getStringExtra("data")+"/"+intent.getData();
7.      Log.d("tag", msg);
8.      Toast.makeText(context, msg, Toast.LENGTH_LONG).show();
9.  }
10. }
```

2、在 AndroidManifest.xml 中进行注册

【文件 1-15】 AndroidManifest.xml 片段

```
1.  <receiver android:name="com.itheima.android.receive.mybroadcast.MyReceiver">
2.          <intent-filter >
3.             <action android:name="com.itheima.broadcast"/>
4.          </intent-filter>
5.   </receiver>
```

1.4 发送有序广播

有序广播可以被拦截，且优先级高的接收者可以拦截优先级低的。

- 广播接收者的优先级的推荐取值范围是: 1000(最高) ~ -1000(最低)
- 相同优先级下，接收的顺序要看在清单文件中声明的顺序，先声明的接收者比后声明的要先收到广播
- 有序广播使用 sendOrderedBroadcast 方法来发送,使用 abortBroadcast 方法拦截
- 广播接收者的优先级在清单文件中声明接收者时，在<intent-filter>标签下通过设置”android:property”属性来设置

1.4.1 需求

创建一个应用，在该应用中模拟一个广播发送者，多个广播接收者。给这些接收者配置不同的优先级。然后观察当发送广播的时候这些接收者接收到广播的顺序和数据又什么不同。

1.4.2 代码

在该工程中使用了一个 MainActivity，提供一个按钮用于点击发送有序广播，三个 BroadCastReceiver 类，分别是 MyReceiver1、MyReceiver2、MyReceiver3。分别给这三个 BroadCastReceiver 配置优先级为 100/200/300。

【文件 1-16】 MainActivity.java

```
1. package com.itheima.orderedbroadcast;
2.
3. import android.os.Bundle;
4. import android.view.View;
5. import android.app.Activity;
6. import android.content.Intent;
7. /**
8.  * 发送无序广播
9.  *
10.  * @author wzy 2015-11-19
11.  *
12.  */
13. public class MainActivity extends Activity {
14.
15. @Override
16. protected void onCreate(Bundle savedInstanceState) {
17.     super.onCreate(savedInstanceState);
18.     setContentView(R.layout.activity_main);
19. }
20.
21. /**
22.  * 发送无序广播
23.  *
24.  * @param view
25.  */
26. public void send(View view) {
27.     Intent intent = new Intent();
28.     intent.setAction("com.itheima.broadcast");
29.     /**
30.      * 参数 1 Intent 类型：意图
31.      * 参数 2 String 类型 receiverPermission，接收器需要的权限
32.      * 参数 3 BroadcastReceiver 类型，自己定义的接收者作为最终接收者
33.      * 参数 4 Handler 类型，用于执行接收器的回调，如果为 null 则在主线程中执行
34.      * 参数 5 int 类型，结果代码的初始码
35.      * 参数 6 初始化参数
36.      * 参数 7Bundle 类型，额外的数据
37.      */
```

```
38.     sendOrderedBroadcast(intent, null, null, null, 0, "转账 10000 元", null);
39. }
40.
41. }
42.
```

3 个 MyReceiver 代码逻辑是完全一样的，因此这里只给出第一个的源码。

【文件 1-17】 MyReceiver1.java

```
1. package com.itheima.orderedbroadcast;
2.
3. import android.content.BroadcastReceiver;
4. import android.content.Context;
5. import android.content.Intent;
6. import android.util.Log;
7. /**
8.  * 接收自己发送的广播
9.  *
10.  * @author wzy 2015-11-19
11.  *
12.  */
13. public class MyReceiver1 extends BroadcastReceiver {
14.
15. @Override
16. public void onReceive(Context context, Intent intent) {
17.     String action = intent.getAction();
18.     String resultData = getResultData();
19.     Log.d("tag", "MyReceiver1 接收到" + action + "发布的广播：" + resultData);
20. }
21. }
```

注册 BroadCastReceiver

【文件 1-18】 AndroidManifest.xml 中接收者配置情况

```
1.  <receiver android:name="com.itheima.orderedbroadcast.MyReceiver1">
2.          <intent-filter android:priority="100">
3.             <action android:name="com.itheima.broadcast"></action>
4.          </intent-filter>
5.       </receiver>
6.       <receiver android:name="com.itheima.orderedbroadcast.MyReceiver2">
7.          <intent-filter android:priority="200">
8.             <action android:name="com.itheima.broadcast"></action>
9.          </intent-filter>
10.       </receiver>
11.       <receiver android:name="com.itheima.orderedbroadcast.MyReceiver3">
12.          <intent-filter android:priority="300">
13.             <action android:name="com.itheima.broadcast"></action>
14.          </intent-filter>
15.       </receiver>
```

1.4.3 结果分析总结

一、发送普通的有序广播

图 1-9　图主界面

如图 1-9 所示，点击按钮后，观测日志如图 1-10 所示。

Application	Tag	Text
com.itheima.orderedbroadcast	tag	MyReceiver3接收到com.itheima.broadcast发布的广播：转账10000元
com.itheima.orderedbroadcast	tag	MyReceiver2接收到com.itheima.broadcast发布的广播：转账10000元
com.itheima.orderedbroadcast	tag	MyReceiver1接收到com.itheima.broadcast发布的广播：转账10000元

图 1-10　日志输出 1

我们发现 MyReceiver3 第一个接收到，MyReceiver2 第二个接收到，MyReceiver1 最后一个才接收到。这个原因很简单，因为这三个的优先级关系是 MyReceiver3>MyReceiver2>MyReceiver1。

二、让 MyReceiver3 终止广播

修改 MyReceiver3 中的 onReceive 方法，让 MyReceiver3 终止广播的传递，然后观察 MyReceiver1 和 MyReceiver2 的活动情况。

【文件 1-19】　MyReceiver3.java 修改后的代码

```
1. public class MyReceiver3 extends BroadcastReceiver {
2.
3.  @Override
4.  public void onReceive(Context context, Intent intent) {
5.      String action = intent.getAction();
6.      String resultData = getResultData();
7.      Log.d("tag", "MyReceiver3 接收到" + action + "发布的广播：" + resultData);
8.      //终止该广播的传播
9.      abortBroadcast();
10. }
11. }
```

重新运行该项目，打印日志如 1-11 所示：

tag:tag　　verbose

Application	Tag	Text
com.itheima.orderedbroadcast	tag	MyReceiver3接收到com.itheima.broadcast发布的广播：转账10000元

图 1-11　日志输出 2

我们发现 MyReceiver1 和 MyReceiver2 没有日志输出，这正是因为优先级最高的 MyReceiver3 将该有序广播总结了。

三、发送带有最终广播接收者的有序广播

修改 MainActivity 中的 send（View）方法，在 sendOrderedBroadcast 的时候我们指定 MyReceiver1 为最终广播接收者。那么就算是之前的 MyReceiver3 将该广播终止了，MyReceiver1 依然可以接收到该广播。这就是最终广播的特点。

【文件 1-20】　MainActivity.java 修改后的代码

```
1. /**
2.   * 发送无序广播
3.   *
4.   * @param view
5.   */
6.  public void send(View view) {
7.      Intent intent = new Intent();
8.      intent.setAction("com.itheima.broadcast");
9.      //创建 MyReceiver1 对象
10.     MyReceiver1 receiver1 = new MyReceiver1();
11.
12.     /**
13.      * 参数 1 Intent 类型：意图
14.      * 参数 2 String 类型 receiverPermission，接收器需要的权限
15.      * 参数 3 BroadcastReceiver 类型，自己定义的接收器作为最终接收器
16.      * 参数 4 Handler 类型，用于执行接收器的回调，如果为 null 则在主线程中执行
17.      * 参数 5 int 类型，结果代码的初始码
18.      * 参数 6 初始化参数
19.      * 参数 7Bundle 类型，额外的数据
20.      */
21.     sendOrderedBroadcast(intent, null, receiver1, null, 0, "转账 10000 元", null);
22. }
```

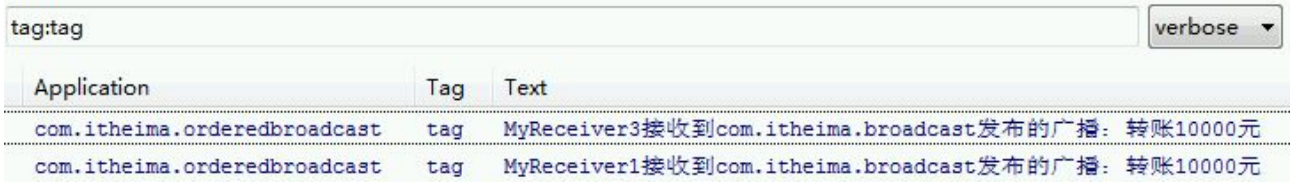

图 1-12　日志输出 3

四、让接收者修改发送的数据

将 MyReceiver3 的 onReceive 方法的 abortBroadCast()方法给去掉，不让其终止广播。然后修改 MyReceiver3 和 MyReceiver2 其代码如下文件所示：

【文件 1-21】 MyReceiver3.java 修改后的代码

```
1. public class MyReceiver3 extends BroadcastReceiver {
2.
3.  @Override
4.  public void onReceive(Context context, Intent intent) {
5.      String action = intent.getAction();
6.      String resultData = getResultData();
7.      Log.d("tag", "MyReceiver3 接收到" + action + "发布的广播：" + resultData);
8.      /*//终止该广播的传播
9.      abortBroadcast();*/
10.     //设置结果数据未 5000 元
11.     setResultData("转账 5000 元");
12. }
13. }
14.
```

【文件 1-22】 MyReceiver2java 修改后的代码

```
1. public class MyReceiver2 extends BroadcastReceiver {
2.
3.  @Override
4.  public void onReceive(Context context, Intent intent) {
5.      String action = intent.getAction();
6.      String resultData = getResultData();
7.      Log.d("tag", "MyReceiver2 接收到" + action + "发布的广播：" + resultData);
8.      //设置结果数据
9.      setResultData("转账 3000 元");
10. }
11. }
```

重新执行上面的代码，运行结果如下：

图 1-13　日志输出 4

观察日志输出，我们发现 MyReceiver3 接收到的数据是 10000 元，然后其将结果修改为了 5000 元，因此 MyReceiver2 接收到的数据就是 5000 元，MyReceiver2 又将数据修改为 3000 元，因此 MyReceiver1 接收到的数据就为 3000 元。这个过程很好理解，注意 MyReceiver1 的日志打印了两次，这是为什么呢？

这是因为我们在发送有序广播的时候将 MyReceiver1 作为了最终广播，因此 MyReceiver1 就有两个身份了，第一次执行是普通的广播接收者接收到数据，第二次是最为最终广播接收者接收到数据。

1.5 特殊的广播接收者-锁屏与解屏

在 Android 中一些操作比较频繁的事件，比如锁屏解屏和电量的变化，也会发送特定的广播。但是此类广播的注册是无法注册在 AndroidManifest.xml 中，只能在代码中进行注册。

BroadCastReceiver 的注册方式有两种：1、静态注册（就是通过 AndroidManifest.xml 注册）2、动态注册（就是通过代码注册）。在本文中前面使用到的 BroadCastReceiver 全部都使用的是静态注册方式，其实也可以使用动态注册，但是对于锁屏解屏和电量变化的监听只能通过动态注册。

接下来做一个锁屏和解屏监听的案例。其实很简单，因此只给出核心部分。

使用到的 MainActivity 类的代码如【文件 1-23】所示。

【文件 1-23】 MainActivity.java

```
1. package com.itheima.screenbroadcast;
2.
3. import android.os.Bundle;
4. import android.app.Activity;
5. import android.content.Intent;
6. import android.content.IntentFilter;
7. /**
8.  * 动态注册屏幕的锁屏和解屏广播
9.  *
10.  * @author wzy 2015-11-19
11.  *
12.  */
13. public class MainActivity extends Activity {
14.
15. private ScreenReceiver mReceiver;
16. @Override
17. protected void onCreate(Bundle savedInstanceState) {
18.     super.onCreate(savedInstanceState);
19.     setContentView(R.layout.activity_main);
20.     //1、创建一个广播接收者对象
21.     mReceiver = new ScreenReceiver();
22.     //2、创建 IntentFilter 对象
23.     IntentFilter filter = new IntentFilter();
24.     //3、添加要监听的 Action
25.     //添加屏幕点亮 Action
26.     filter.addAction(Intent.ACTION_SCREEN_ON);
27.     //添加屏幕关闭 Action
28.     filter.addAction(Intent.ACTION_SCREEN_OFF);
29.     //动态注册广播接收者
30.     registerReceiver(mReceiver, filter);
31. }
```

```
32. @Override
33. protected void onDestroy() {
34.     super.onDestroy();
35.     /*
36.      * 取消注册广播接收者
37.      * 在 onCreate 中注册了，广播接收者，在 onDestory 中就必须反注册，目的是为了防止内存的泄露
38.      */
39.     unregisterReceiver(mReceiver);
40. }
41.
42. }
```

自定义类继承 BroadCastReceiver 类。代码如【文件 1-24】所示。

【文件 1-24】 ScreenReceiver.java

```
1. package com.itheima.screenbroadcast;
2.
3. import android.content.BroadcastReceiver;
4. import android.content.Context;
5. import android.content.Intent;
6. import android.util.Log;
7. /**
8.  * 监听屏幕的点亮和关闭
9.  *
10.  * @author wzy 2015-11-19
11.  *
12.  */
13. public class ScreenReceiver extends BroadcastReceiver {
14.
15. @Override
16. public void onReceive(Context context, Intent intent) {
17.     if (intent.getAction().equals(Intent.ACTION_SCREEN_ON)) {
18.         Log.d("tag", "监听到屏幕点亮了");
19.     }else if (intent.getAction().equals(Intent.ACTION_SCREEN_OFF)) {
20.         Log.d("tag", "监听到屏幕关闭了");
21.     }
22. }
23. }
24.
```

监听锁屏和解屏不需要额外的权限，因此可以直击运行。

测试上面的代码日志输入如图 1-14 所示。

图 1-14　锁屏解屏日志

注意：如果我们的应用退出了，那么就无法监听到该广播，可能有人说是因为退出的时候调用了 MainActivity 的 onDestory（）方法，而 onDestory（）方法中调用了 unregisterReceiver（BroadCastReceiver）方法，确实如此的。那么如果我们将 onDestory（）方法中的 unregisterReceiver（）方法给去掉会是什么样的情况呢？那我们就一起试一下吧！

将 MainActivity.java 中的 unregisterReceiver（）；去掉之后重新运行上面的项目，然后点击 back 键，发现 logcat 输入了如下错误信息。

```
11-19 14:02:01.062: E/ActivityThread(2336): Activity
com.itheima.screenbroadcast.MainActivity has leaked IntentReceiver
com.itheima.screenbroadcast.ScreenReceiver@b55ff358 that was originally registered here. Are
you missing a call to unregisterReceiver()?
11-19 14:02:01.062: E/ActivityThread(2336): android.app.IntentReceiverLeaked: Activity
com.itheima.screenbroadcast.MainActivity has leaked IntentReceiver
com.itheima.screenbroadcast.ScreenReceiver@b55ff358 that was originally registered here. Are
you missing a call to unregisterReceiver()?
```

上面的信息说我们的 MainActivity 导致了 IntentReceiver 的泄露。ScreenReceiver 在 MainActivity 中被注册了但是当 MainActivity 销毁的时候没有被反注册。

可见要想让我们的代码更加完善，必须在 onDestory 中或者其他地方（必须保证 Activity 销毁前）将接收者给反注册掉。