宝贵建议请发送至：wangzhenyang@itcast.cn

黑马程序员 itheima.com -编程，始于黑马

Android 课程同步笔记

Alpha 0.01 版

Android-BroadcastReceiver

1.BroadcastReceiver 简介（★★★★）

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。

广播接收者（BroadcastReceiver）用于接收广播 Intent 的，广播 Intent 的发送是通过调用 sendBroadcast/sendOrderedBroadcast 来实现的。通常一个广播 Intent 可以被订阅了此 Intent 的多个广播接收者所接收。

1.1 实现一个 BroadcastReceiver

需求：定义一个广播接收器，用于接收 SDCard 移除时发送的广播。

1. 创建一个新的 Android 工程《广播接收器》,包名 :com.itheima.broadcastReciver。
2. 在 src 目录下新建一个 SDCardUnmountedReceiver 类继承 BroadcastReceiver 类，覆写 onReceive 方法,代码清单如下：

```
public class SDCardUnmountedReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "SDCard 已经被移除！", 0).show();
    }
}
```

3. 注册 SDCardUnmountedReceiver。

Tips：注册一个广播接收者有两种方式。

静态注册：在 AndroidManifest.xml 中注册广播

在 AndroidManifest.xml 文件中添加如下配置：

```
<receiver
android:name="com.itheima.broadcastReciver.SDCardUnmountedReceiver
">
          <intent-filter>
<action android:name="android.intent.action.MEDIA_UNMOUNTED"/>
         <data android:scheme="file"></data>
          </intent-filter>
</receiver>
```

动态注册：在 Java 代码中注册

```
public class MainActivity extends Activity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      IntentFilter intentFilter = new
IntentFilter("android.intent.action.MEDIA_UNMOUNTED");
      intentFilter.addDataScheme("file");
      registerReceiver(new SDCardUnmountedReceiver(),
intentFilter);
      System.out.println("广播接收器已经注册成功。");
   }
}
```

4 由于 Android 高版本中已经不支持 SDCard 的卸载，因此我们使用 Android 2.3 版本的模拟器。在 settings->Storage settings 中可以找到 Unmount SD Card。点击此选项可以移除 SDCard。

软件运行效果截图如下：发现 Toast 成功打印出了“SDCard 已经被移除”信息。

注意：

★ java 代码注册的广播接收者优先级要比清单文件的要高，但是当前的广播接收者的生命周期的期限和 activity 是相关联的,activity 销毁，广播接收者也就不再起作用。

★ 通过清单文件注册的广播接收者在系统中运行一次后就会被注册到系统中，以后无需运行此广播接受者，但是也可以接收到广播。

★ 接收广播时要注意在清单文件中添加对应的权限。

2.Android 中常见广播（★★★）

2.1 监听拨打电话广播

需求：监听用户拨打电话，在用户拨打电话号码前自动加上 17951 等。

拦截的广播：

```
<action
android:name="android.intent.action.NEW_OUTGOING_CALL"></action>
```

需要的权限：

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

1 新创建一个 Android 工程《IPCaller》。

2 在 src 目录下新创建一个类 IPCallerReceiver 继承 BroadcastReceiver，重写 OnReceive 方法。

```
public class IPCallReveiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //获取拨打的电话号码
        String resultData = getResultData();
        //在电话号码前加上 17951，然后返回数据
        setResultData("17951"+resultData);
    }
}
```

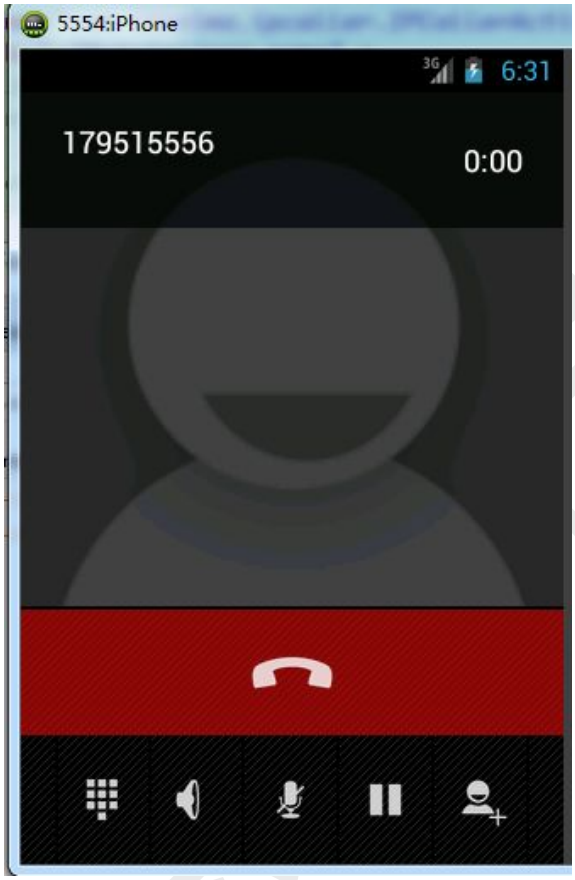
3 在 AndroidManifest.xml 中注册广播接收者。

在下面的配置文件中 `<intent-filter android:priority="1000">`属性代表着给当前接收者设置优先级，优先级越高越优先接收到广播。

```
<receiver android:name="com.itheima.ipcaller.IPCallReveiver">
            <intent-filter android:priority="1000">
                <action
android:name="android.intent.action.NEW_OUTGOING_CALL"></action>
            </intent-filter>
        </receiver>
```

4 运行上面的代码，然后拨打电话 5556，发现拨出去的号码已经变为 179515556。

运行效果图如下：



2.2 监听系统开机的广播

需求：

拦截手机开机的广播，手机开机后，弹一个提示。

拦截的广播：

```
<action
android:name="android.intent.action.BOOT_COMPLETED"></action>
```

需要的权限:

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

虚拟机重启命令：adb shell reboot

Tips：3.0 以上版本必须加权限，以下的版本可以不加,3.0 以上的版本如果用户没有启动过程序，接收不到开启启动完成的广播。

1 新创建一个 Android 工程《开机启动》。

2 在 src 目录下新创建一个类 BootReceiver 继承 BroadcastReceiver ,重写 OnReceive 方法。

```
public class BootReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "开机启动已经完成", 1).show();
    }
}
```

3 在 AndroidManifest.xml 中注册广播接收者。

```
<receiver android:name="com.itheima.bootStart.BootReceiver" >
        <intent-filter>
            <action
android:name="android.intent.action.BOOT_COMPLETED"></action>
        </intent-filter>
    </receiver>
```

4 将上面的代码部署到模拟器上，然后关机重启模拟器。

运行效果图如下：

2.3 监听安装和卸载程序的广播

需求：

监听程序的安装或者卸载，并在 LogCat 中输出提示信息。

拦截的广播：

```
<action
android:name="android.intent.action.PACKAGE_ADDED"></action>
<action
android:name="android.intent.action.PACKAGE_REMOVED"></action>
```

指定 scheme: package

1 新创建一个 Android 工程《监听应用安装与卸载》

2 新创建 InstallReceiver 类继承 BroadcastReceiver 类，覆写 onReceive 方法

```
public class InstallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //获取包名称
        String packageName = intent.getData().toString();
        String action = intent.getAction();
        //如果是卸载动作
        if ("android.intent.action.PACKAGE_REMOVED".equals(action)) {
            Toast.makeText(context, packageName + "已经被删除！",
1).show();
            System.out.println(packageName + "已经被删除！");
        //如果是安装动作
        } else if
("android.intent.action.PACKAGE_ADDED".equals(action)) {
            Toast.makeText(context, packageName + "已经被安装！",
1).show();
            System.out.println(packageName + "已经被安装！");
        }
    }

}
```

3 在 AndroidManifest.xml 清单中注册 InstallReceiver。

```
  <receiver
android:name="com.itheima.installReceiver.InstallReceiver">
          <intent-filter>
 <action android:name="android.intent.action.PACKAGE_REMOVED"/>
<action android:name="android.intent.action.PACKAGE_ADDED"/>
              <data android:scheme="package"/>
          </intent-filter>
 </receiver>
```

4 将上面的工程部署在模拟器上。然后通过系统应用管理工具，卸载一个应用程序，发

现成功接收到了应用被卸载的广播。运行图如下：

2.4 拦截短信

需求：

对用户接收的短信进行拦截，若是 10086 发来的短信，将此短信拦截。

拦截的广播：

```
<action
android:name="android.provider.Telephony.SMS_RECEIVED"></action>
```

需要的权限:

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

Tips：android 4.2 后废除了此 action。

1 新创建一个工程《短信拦截》,在 src 目录下新建 SMSReceiver 类继承 BroadcastReceiver 类

```
public class SMSReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        //获取短信数据 pdus：短信集合
        Object[] objs = (Object[]) intent.getExtras().get("pdus");
        for(Object pdu : objs){
            //通过 Android API 中的 SmsMessage 类将短信字节数组转化为短信对象
            SmsMessage message = SmsMessage.createFromPdu((byte[])pdu);
            //获取短信来源
            String address = message.getOriginatingAddress();
            //获取短信内容
            String body = message.getMessageBody();
            //短信短信内容
            System.out.println(address+"--"+body);
            Toast.makeText(context, address+"--"+body, 1).show();
            //终止广播
            abortBroadcast();
        }
    }
```

2 在 AndroidManifest.xml 中注册短信拦截器

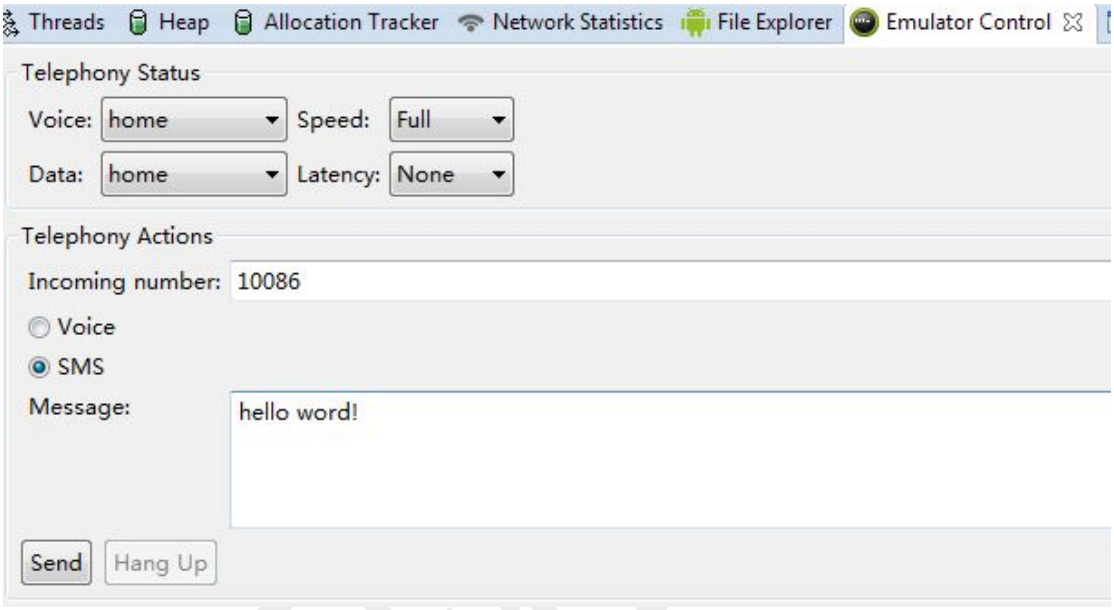
```
<receiver android:name="com.itheima.smsreceiver.SMSReceiver">
            <intent-filter android:priority="1000">
                <action
android:name="android.provider.Telephony.SMS_RECEIVED"/>
            </intent-filter>
        </receiver>
```

3 在 AndroidManifest.xml 中添加权限

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

4 将上面的工程部署在模拟器上

打开 DDMS 视图，选择 Emulator Control 选项卡，在 Incomeing number：中填写一个号码，模拟发送消息的号码。然后选择 SMS 单选按钮，在右侧的输入框输入短信的内容。然后点击 Send 按钮，发送信息。



运行截图如下：发现成功拦截了刚才发送的短信。

3.发送自定义广播（★★★★）

3.1 无序广播

- 无序广播不可以被拦截，如果被拦截的话会报错：

```
BroadcastReceiver trying to return result during a non-ordered broadcast
```

- 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 sendBroadcast 方法来发送

Tips：无序广播的实现比较简单，因此这里只给出核心代码。

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

3.2 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
- 无序广播使用 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

我们新创建一个项目，来演示无序广播的发送和接收过程。

1 新创建一个 Android 工程《广播发送和接收》，包名 com.itheima.broadcastAndreceiver。

2 在默认的 MainActivity 的布局中添加一个按钮，绑定事件，该事件的核心功能是发送一个有序广播，MainActivity 类代码清单如下：

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void sendOrder(View view) {
        // 意图
        Intent intent = new Intent();
        intent.setAction("com.itheima.data");
        /**
         * intent The Intent to broadcast; all receivers matching this
Intent
         * will receive the broadcast. receiverPermission String naming
a
         * permissions that a receiver must hold in order to receive your
         * broadcast. If null, no permission is required. resultReceiver
Your
         * own BroadcastReceiver to treat as the final receiver of the
         * broadcast. scheduler A custom Handler with which to schedule
the
         * resultReceiver callback; if null it will be scheduled in the
```

```
 * Context's main thread. initialCode An initial value for the result
        * code. Often Activity.RESULT_OK. initialData An initial value
for the
        * result data. Often null. initialExtras An initial value for
the
        * result extras. Often null.
        * 第一个参数 Intent 类型：意图
        * 第二个参数 String 类型 receiverPermission，接收器需要的权限
        * 第三个参数 BroadcastReceiver 类型，自己定义的接收器作为最终接收
器
        * 第四个参数 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
        * 第五个参数 int 类型，结果代码的初始码
        * 第六个参数初始化参数
        * 第七个参数 Bundle 类型，额外的数据
        */
        sendOrderedBroadcast(intent, null, null,
       null, RESULT_OK, "1 万元钱", null);

   }
```

3 分别编写 MyReceiver 和 MyReceiver2 类，继承 BroadcastReceiver 类

MyReceiver 类代码清单：

```
public class MyReceiver extends BroadcastReceiver {

   @Override
   public void onReceive(Context context, Intent intent) {
      String action = intent.getAction();
      String resultData = getResultData();
      Toast.makeText(context, "MyReceiver 接收到"+action+"发布的广播:
"+resultData, 1).show();
      System.out.println("MyReceiver 接收到"+action+"发布的广播:
"+resultData);
      }
}
```

MyReceiver2 类代码清单：

```
public class MyReceiver2 extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        String resultData = getResultData();
        Toast.makeText(context, "MyReceiver2 接收到"+action+"发布的广播： "+resultData, 1).show();
        System.out.println("MyReceiver2 接收到"+action+"发布的广播： "+resultData);

    }
}
```

4 在 AndroidManifest.xml 中注册 MyReceiver 和 MyReceiver2

```
 <receiver
android:name="com.itheima.broadcastAndreceiver.MyReceiver">
            <intent-filter android:priority="1000">
                <action android:name="com.itheima.data"></action>
            </intent-filter>
        </receiver>
        <receiver
android:name="com.itheima.broadcastAndreceiver.MyReceiver2">
            <intent-filter android:priority="-1000">
                <action android:name="com.itheima.data"></action>
            </intent-filter>
        </receiver>
```

Tips:在上面清单文件中我们给 MyReceiver 设置了最高优先级 1000，给 MyReceiver2 设置了最低优先级-1000。

5 下面我们分多钟情况，分别演示有序广播的接收规律

1、直接部署上面的工程到模拟器，点击发送广播按钮，控制台结果为：

```
MyReceiver接收到com.itheima.data发布的广播：1万元钱
MyReceiver2接收到com.itheima.data发布的广播：1万元钱
```

我们发现 MyReceiver 先接收到了广播，然后 MyReceiver2 才接收到广播。

2、修改 MyReceiver 类的代码：在 onReceive 方法中添加 abortBroadcast( )方法，

```
public class MyReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        String resultData = getResultData();
        Toast.makeText(context, "MyReceiver 接收到"+action+"发布的广播:
"+resultData, 1).show();
        System.out.println("MyReceiver 接收到"+action+"发布的广播:
"+resultData);
        abortBroadcast();
    }
}
```

然重新部署该工程，运行，点击发送广播按钮。这时控制台打印信息为：

```
MyReceiver接收到com.itheima.data发布的广播：1万元钱
```

我们发现只有 MyReceiver 接收到了广播，而 MyReceiver2 没有接收到广播。原因就是我们在 MyReceiver 中执行了 `abortBroadcast（）` 方法，终止了该广播。

3、在第二种情况的基础上，我们修改 MainActivity 类中 sendOrder 方法，修改 sendOrderedBroadcast 中第三个参数，指定一个最终接收器。

```
public void sendOrder(View view) {
        // 意图
        Intent intent = new Intent();
        intent.setAction("com.itheima.data");
         sendOrderedBroadcast(intent, null, new MyReceiver2(),
        null, RESULT_OK, "1 万元钱", null);
}
```

运行上面工程，然后点击发送广播按钮，发现控制台输出如下信息：

```
MyReceiver接收到com.itheima.data发布的广播：1万元钱
MyReceiver2接收到com.itheima.data发布的广播：1万元钱
```

我们发现，虽然在 MyReceiver 中我们调用了 abortBroadcast();方法，但是广播依然被 MyReceiver2 接收到。原因是我们在 sendOrderedBroadcast 方法中指定了 MyReceiver2 接收器为最终接收器，因此该广播被终止的时候 MyReceiver2 接收器依然可以接收到广播。

◆ 4、在第 3 种情形的基础上，我们修改 MyReceiver 类，我们将该类中的 onReceive 方法中的 abortBroadcast();方法去掉。然后运行上面工程。发现控制台输入如下信息：

```
MyReceiver接收到com.itheima.data发布的广播：1万元钱
MyReceiver2接收到com.itheima.data发布的广播：1万元钱
MyReceiver2接收到com.itheima.data发布的广播：1万元钱
```

我们发现 MyReceiver 第一个接收到广播，MyReceiver2 第二个接收到广播，然后 MyReceiver2 又接收到一次广播。对的，结果确实是这样，因为我们在 sendOrderedBroadcast 方法中，将 MyReceiver2 作为最终接收器，那么我们发出的广播会被所有符合条件的接收器接收，最后指定的最终接收器不管是否已经接收过信息依然会再次接收。

至此，本文档内容完！