Android 部分 Broadcast 篇

1.广播是什么

1.1 定义

在 Android 中，它是一种广泛运用在应用程序之间传输信息的机制，Android 中我们发送广播内容是一个 Intent,这个 Intent 中可以携带我们要发送的数据。

1.2 广播的使用场景

a.同一 app 内有多个进程的不同组件之间的消息通信。

b.不同 app 之间的组件之间消息的通信。

2 广播的种类

2.1 无序广播

context.sendBroadcast(Intent)方法发送的广播，不可被拦截，当然发送的数据，接收者是不能进行修改的。

2.2 有序广播

context.sendOrderBroadcast(Intent)方法发送的广播，可被拦截，而且接收者是可以修改其中要发送的数据，修改和添加都是可以的，这就意味着优先接收者对数据修改之后，下一个接收者接受的数据是上一个接收者已经修改了的，这必须明白。

2.3 本地广播

localBroadcastManager.sendBroadcast(Intent)，只在 app 内传播。

本地广播的发送和注册广播接收器都需要使用到 LocalBroadcastManager 类，如下所示为本地广播的发送和本地广播接收器注册的代码：

本地广播的发送：

```
1.  public static void sendLocalBroadcast(Context context,String action){
2.
3.      Intent intent = new Intent(action);
4.      LocalBroadcastManager localBroadcastManager = LocalBroadcastManager.getI
    nstance(context);
5.      localBroadcastManager.sendBroadcast(intent);
6.
7.  }
```

本地广播的接收器的注册：

```
1.  ntentFilter intentFilter = new IntentFilter();
2.      LocalBroadcastManager localBroadcastManager = LocalBroadcastManager.getI
    nstance(context);
3.
4.      intentFilter.addAction(new BroadcastUtil().action_next);
5.      nasbr = new NextAndStartBroadcastReceiver();
6.      localBroadcastManager.registerReceiver(nasbr, intentFilter);//注册本地广
    播接收器
```

3. 广播接收器

广播接收器是专门用来接收广播信息的，它可分为静态注册和动态注册：

3.1 静态注册

首先你要创建一个广播接收器类，实例代码如下：

```
1.  public class BootCompleteReceiver extends BroadcastReceiver {
2.      @Override
3.      public void onReceive(Context context, Intent intent) {
4.          Toast.makeText(context, "Boot Complete", Toast.LENGTH_LONG).show();

5.      }
6.  }
```

代码非常简单，我们只是在 onReceive()方法中使用 Toast 弹出一段信息提示信息。另外，静态的广播接收器一定要在 AndroidManifest.xml 文件中注册才可以使用，AndroidManifest.xml 文件中注册静态广播代码如下：

```
1.  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2.          package="com.example.broadcasttest"
3.          android:versionCode="1"
4.          android:versionName="1.0" >
5.          ……
6.        <uses-
    permission android:name="android.permission.ACCESS_NETWORK_STATE" />
7.        <uses-
    permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
8.
9.        <application
10.           android:allowBackup="true"
11.           android:icon="@drawable/ic_launcher"
12.           android:label="@string/app_name"
13.           android:theme="@style/AppTheme" >
14.           ……
```

```
15.
16.            <receiver
17.                android:name=".BootCompleteReceiver" >
18.                <intent-filter>
19.                    <action android:name="android.intent.action.BOOT_COMPLETE
    D" />
20.                </intent-filter>
21.            </receiver>
22.        </application>
23. </manifest>
```

可以看到，标签内出现了一个新的标签，所有静态的广播接收器都是在这里进行注册的。它的用法其实和标签非常相似，也是通过 android:name 来指定具体注册哪一个广播接收器，而 enabled 和 exported 属性则是根据我们刚才勾选的状态自动生成的，也可以自己添加，没有就自己添加嘛！

3.2 动态注册

如何创建一个广播接收器呢？其实就需要新建一个类，让它继承自 BroadcastReceiver，并重写父类的 onReceive()方法就行了。这样有广播到来时，onReceive()方法就会得到执行，具体的逻辑就可以在这个方法中处理。来个简单的例子来理解以下，如何监听网络变化呢？其实就是新建一个广播接收器去接收来自系统网络变化的广播即可，代码如下所示：

```
1.  public class MainActivity extends AppCompatActivity{
2.
3.      private IntentFilter intentFilter;
4.
5.      private NetWorkChangeReceiver netWorkChangeReceiver;
6.
7.      @Override
8.      protected void onCreate(Bundle savedInstanceState){
9.          super.Oncreata(savedInstanceState);
10.         setContentView(R.layout.activity_main);
11.
12.         intentFilter = new IntentFilter();
13.         intentFilter.addAction("android.net.conn.CONNECTIVITY_CHANGE");
14.         networkChangeReceiver = new NetWorkChangeReceiver();
15.         registerReceiver(networkChangeReceiver, intentFilter);//注册广播接收
    器
16.
17.     }
18.
19.     @Overrid
20.     protected void onDestroy(){
```

```
        unregisterReceiver(networkChangeReceiver);//一定要记得取消广播接收器的注册
        super.onDestroy();

    }

    class NetworkChangeReceiver extends BroadcastReceiver{//广播接收器类

        @Override
        public void onReceiver(Context context,Intent intent){

            //这里需要权限，需要在 AndroidManifest.xml 中进行网络访问权限申请:
            //<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
            ConnectivityManager connectionManager = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo networkInfo = connectionManager.getActiveNetworkInfo();

            if (networkInfo != null && networkInfo.isAvailable()) {

                //有网
                Toast.makeText(context, "network is available",Toast.LENGTH_SHORT).show();

            } else {

                //无网
                Toast.makeText(context, "network is unavailable",
                Toast.LENGTH_SHORT).show();
            }
        }

    }


}
```

动态注册广播接收器的优点以及缺点：动态注册的广播接收器可以自由地控制注册与注销，在灵活性方面有很大优势，但是它也存在着一个缺点，即必须要在程序启动之后才能接收到广播，因为注册的逻辑是写在 onCreate()方法中的。那么有没有广播能在程序未启动的情况下就能接收到广播呢？静态注册的广播接收器就可以做到。

3.3 那些系统发送的广播有哪些？

- 监听网络变化 android.net.conn.CONNECTIVITY_CHANGE
- 关闭或打开飞行模式 Intent.ACTION_AIRPLANE_MODE_CHANGED
- 充电时或电量发生变化 Intent.ACTION_BATTERY_CHANGED
- 电池电量低 Intent.ACTION_BATTERY_LOW
- 电池电量充足（即从电量低变化到饱满时会发出广播 Intent.ACTION_BATTERY_OKAY
- 系统启动完成后(仅广播一次) Intent.ACTION_BOOT_COMPLETED
- 按下照相时的拍照按键(硬件按键)时 Intent.ACTION_CAMERA_BUTTON
- 屏幕锁屏 Intent.ACTION_CLOSE_SYSTEM_DIALOGS
- 设备当前设置被改变时(界面语言、设备方向等)
  Intent.ACTION_CONFIGURATION_CHANGED
- 插入耳机时 Intent.ACTION_HEADSET_PLUG
- 未正确移除 SD 卡但已取出来时(正确移除方法:设置－SD 卡和设备内存－卸载 SD 卡)
  Intent.ACTION_MEDIA_BAD_REMOVAL
- 插入外部储存装置（如 SD 卡） Intent.ACTION_MEDIA_CHECKING
- 成功安装 APK Intent.ACTION_PACKAGE_ADDED
- 成功删除 APK Intent.ACTION_PACKAGE_REMOVED
- 重启设备 Intent.ACTION_REBOOT
- 屏幕被关闭 Intent.ACTION_SCREEN_OFF
- 屏幕被打开 Intent.ACTION_SCREEN_ON
- 关闭系统时 Intent.ACTION_SHUTDOWN
- 重启设备 Intent.ACTION_REBOOT
- ……

4.源码角度分析广播机制

4.1 系统广播的源码角度分析

a.自定义广播接收者 BroadcastReceiver,并且重写 onReceiver()方法。

b.通过 Binder 机制向 AMS(Activity Manager Service)进行注册。

c.广播发送者通过 Binder 机制向 AMS 发送广播。

d.AMS 查找符合条件(IntentFilter/Permission 等)的 BroadcastReceiver，将广播发送到相应的 BroadcastReceiver(一般情况下是 Activity)的消息队列中。

e.消息循环执行拿到此广播，回调 BroadcastReceiver 中的 onReceiver()方法。

4.2 本地广播的源码角度分析

相比于系统广播而言，本地广播更加安全，更加高效，以下是本地广播的特点以及内部的实现机制：

特点：

a.使用它发送的广播将只在自身 app 内传播，因此你不必担心泄漏隐私的数据。

b.其他 app 无法对你的 app 发送该广播，因此你的 app 根本不可能收到非自身 app 发送的该广播，因此你不必担心有安全漏洞可以利用。

c.比系统广播更加高效。

内部实现机制：

a.LocalBroadcast 高效的原因：因为它内部是通过 Handler 实现的，它的 sendBroadcast()方法含义并非和系统的 sendBroadcast()一样，它的 sendBroadcast()方法其实就是通过 Handler 发送了一个 Message 而已。

b.LocalBroadcast 安全的原因：既然它是通过 Handler 实现广播发送的，那么相比系统广播通过 Binder 机制实现那肯定更加高效，同时使用 Handler 来实现，别的 app 无法向我们应用发送该广播，而我们 app 内部发送的广播也不会离开我们的 app。

LocalBroadcast 内部协作主要是靠两个 Map 集合：mReceivers 和 mActions,当然还有一个 List 集合 mPendingBroadcasts,这个主要存储待接收的广播对象。