BroadcastReceiver的定义

广播是一种广泛运用的在应用程序之间传输信息的机制，主要用来监听系统或者应用发出的广播信息，然后根据广播信息作为相应的逻辑处理，也可以用来传输少量、频率低的数据。

在实现开机启动服务和网络状态改变、电量变化、短信和来电时通过接收系统的广播让应用程序作出相应的处理。

BroadcastReceiver 自身并不实现图形用户界面，但是当它收到某个通知后， BroadcastReceiver 可以通过启动 Service 、启动 Activity 或是 NotificationMananger 提醒用户。

BroadcastReceiver使用注意

当系统或应用发出广播时，将会扫描系统中的所有广播接收者，通过action匹配将广播发送给相应的接收者，接收者收到广播后将会产生一个广播接收者的实例，执行其中的onReceiver()这个方法；特别需要注意的是这个实例的生命周期只有10秒，如果10秒内没执行结束onReceiver()，系统将会报错。 在onReceiver()执行完毕之后，该实例将会被销毁，所以不要在onReceiver()中执行耗时操作，也不要在里面创建子线程处理业务（因为可能子线程没处理完，接收者就被回收了，那么子线程也会跟着被回收掉）；正确的处理方法就是通过in调用activity或者service处理业务。

BroadcastReceiver的注册

BroadcastReceiver的注册方式有且只有两种，一种是静态注册（推荐使用），另外一种是动态注册，广播接收者在注册后就开始监听系统或者应用之间发送的广播消息。

接收短信广播示例：

定义自己的BroadcastReceiver 类

```
public class MyBroadcastReceiver extends BroadcastReceiver {

// action 名称
String SMS_RECEIVED = "android.provider.Telephony.SMS_RECEIVED" ;

    public void onReceive(Context context, Intent intent) {

       if (intent.getAction().equals( SMS_RECEIVED )) {
           // 一个receiver可以接收多个action的，即可以有多个intent-filter，需要在onReceive里面对intent.getActio
       }
    }
}
```

静态方式

在AndroidManifest.xml的application里面定义receiver并设置要接收的action。

```
< receiver android:name = ".MyBroadcastReceiver" >

 < intent-filter android:priority = "777" >
<action android:name = "android.provider.Telephony.SMS_RECEIVED" />
</ intent-filter >

</ receiver >
```

这里的priority取值是 -1000到1000，值越大优先级越高，同时注意加上系统接收短信的限权。

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

静态注册的广播接收者是一个常驻在系统中的全局监听器，当你在应用中配置了一个静态的BroadcastReceiver，安装了应用后而无论应用是否处于运行状态，广播接收者都是已经常驻在系统中了。同时应用里的所有receiver都在清单文件里面，方便查看。 要销毁掉静态注册的广播接收者，可以通过调用PackageManager将Receiver禁用。

动态方式

在Activity中声明BroadcastReceiver的扩展对象，在onResume中注册，onPause中卸载.

```
public class MainActivity extends Activity {
    MyBroadcastReceiver receiver;
    @Override
     protected void onResume() {
        // 动态注册广播 (代码执行到这才会开始监听广播消息，并对广播消息作为相应的处理)
        receiver = new MyBroadcastReceiver();
        IntentFilter intentFilter = new IntentFilter( "android.provider.Telephony.SMS_RECEIVED" );
        registerReceiver( receiver , intentFilter);
        super.onResume();
    }
    @Override
    protected void onPause() {
        // 撤销注册 (撤销注册后广播接收者将不会再监听系统的广播消息)
        unregisterReceiver(receiver);
        super.onPause();
    }
}
```

静态注册和动态注册的区别

1、静态注册的广播接收者一经安装就常驻在系统之中，不需要重新启动唤醒接收者；　　动态注册的广播接收者随着应用的生命周期，由registerReceiver开始监听，由unregisterReceiver撤销监听，如果应用退出后，没有撤销已经注册的接收者应用应用将会报错。

2、当广播接收者通过intent启动一个activity或者service时，如果intent中无法匹配到相应的组件。　　动态注册的广播接收者将会导致应用报错　　而静态注册的广播接收者将不会有任何报错，因为自从应用安装完成后，广播接收者跟应用已经脱离了关系。

发送BroadcastReceiver

发送广播主要有两种类型：

普通广播

应用在需要通知各个广播接收者的情况下使用，如 开机启动

使用方法：sendBroadcast()

```
Intent intent = new Intent("android.provider.Telephony.SMS_RECEIVED");
//通过intent传递少量数据
intent.putExtra("data", "finch");
// 发送普通广播
sendBroadcast(Intent);
```

普通广播是完全异步的，可以在同一时刻（逻辑上）被所有接收者接收到，所有满足条件的 BroadcastReceiver 都会随机地执行其 onReceive() 方法。　　同级别接收是先后是随机的；级别低的收到广播；　　消息传递的效率比较高，并且无法中断广播的传播。

有序广播

应用在需要有特定拦截的场景下使用，如黑名单短信、电话拦截。

使用方法：

`sendOrderedBroadcast(intent, receiverPermission);`

`receiverPermission` ：一个接收器必须持以接收您的广播。如果为 null ，不经许可的要求（一般都为null）。

```
//发送有序广播
 sendOrderedBroadcast(intent, null);
```

在有序广播中，我们可以在前一个广播接收者将处理好的数据传送给后面的广播接收者，也可以调用abortBroadcast()来终结广播的传播。

```
public void onReceive(Context arg0, Intent intent) {
    //获取上一个广播的bundle数据
    Bundle bundle = getResultExtras(true);//true：前一个广播没有结果时创建新的Bundle；false：不创建Bundle
    bundle.putString("key", "777");
    //将bundle数据放入广播中传给下一个广播接收者
    setResultExtras(bundle);

    //终止广播传给下一个广播接收者
    abortBroadcast();
}
```

高级别的广播收到该广播后，可以决定把该广播是否截断掉。　　同级别接收是先后是随机的，如果先接收到的把广播截断了，同级别的例外的接收者是无法收到该广播。

异步广播

使用方法：`sendStickyBroadcast()`：

发出的Intent当接收Activity（动态注册）重新处于onResume状态之后就能重新接受到其Intent.（the Intent will be held to be re-broadcast to future receivers）。就是说sendStickyBroadcast发出的最后一个Intent会被保留，下次当Activity处于活跃的时候又会接受到它。

发这个广播需要权限：

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

卸载该广播：

```
removeStickyBroadcast(intent);
```

在卸载之前该intent会保留，接收者在可接收状态都能获得。

异步有序广播

使用方法：`sendStickyOrderedBroadcast(intent, resultReceiver, scheduler, initialCode, initialData, initialExtras)`：

这个方法具有有序广播的特性也有异步广播的特性；　　同时需要限权：

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

总结

- 静态广播接收的处理器是由PackageManagerService负责，当手机启动或者新安装了应用的时候，PackageManagerService会扫描手机中所有已安装的APP应用，将AndroidManifest.xml中有关注册广播的信息解析出来，存储至一个全局静态变量当中。
- 动态广播接收的处理器是由ActivityManagerService负责，当APP的服务或者进程起来之后，执行了注册广播接收的代码逻辑，即进行加载，最后会存储在一个另外的全局静态变量中。需要注意的是：

1、 这个并非是一成不变的，当程序被杀死之后，已注册的动态广播接收器也会被移出全局变量，直到下次程序启动，再进行动态广播的注册，当然这里面的顺序也已经变更了一次。

2、这里也并没完整的进行广播的排序，只记录的注册的先后顺序，并未有结合优先级的处理。

- 广播发出的时候，广播接收者接收的顺序如下：

1．当广播为**普通广播**时，有如下的接收顺序：

无视优先级　　动态优先于静态　　同优先级的动态广播接收器，**先注册的大于后注册的**　　同优先级的静态广播接收器，**先扫描的大于后扫描的**

2．如果广播为**有序广播**，那么会将动态广播处理器和静态广播处理器合并在一起处理广播的消息，最终确定广播接收的顺序：
优先级高的先接收　　同优先级的动静态广播接收器，**动态优先于静态**　　同优先级的动态广播接收器，**先注册的大于后注册的**　　同优先级的静态广播接收器，**先扫描的大于后扫描的**

一些常用的系统广播的action 和permission

- 开机启动

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

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

- 网络状态

```
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
```

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

网络是否可用的方法：

```
  public static boolean isNetworkAvailable(Context context) {
        ConnectivityManager mgr = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_S
        NetworkInfo[] info = mgr.getAllNetworkInfo();
        if (info != null) {
            for (int i = 0; i < info.length; i++) {
      if (info[i].getState() == NetworkInfo.State.CONNECTED) {
                    return true;
                }
            }
        }
        return false;
    }
```

- 电量变化

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

BroadcastReceiver 的onReceive方法：

```
public void onReceive(Context context, Intent intent) {
        int currLevel = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, 0);  //当前电量

        int total = intent.getIntExtra(BatteryManager.EXTRA_SCALE, 1);     //总电量
        int percent = currLevel * 100 / total;
        Log.i(TAG, "battery: " + percent + "%");
    }
```