|  |
| --- |
| 1、广播，是一个全局的监听器，属于Android四大组件之一。Android广播分为两个角色：广播发送者、广播接收者。  2、监听/接收应用App发出的广播消息，并做出响应。  3. 应用场景  Android不同组件间的通信（含 ：应用内 / 不同应用之间）；  多线程通信；  与 Android 系统在特定情况下的通信；如：电话呼入时、网络可用时。  4. 实现原理  Android中的广播使用了设计模式中的观察者模式：基于消息的发布/订阅事件模型。因此，Android将广播的发送者和接收者解耦，使得系统方便集成，更易扩展。  5、模型中有3个角色：  消息订阅者（广播接收者）  消息发布者（广播发布者）  消息中心（AMS，即Activity Manager Service）  6、（1）自定义广播接收者BroadcastReceiver，继承BroadcastReceivre基类。  必须复写抽象方法onReceive()方法。  广播接收器运行在 UI 线程，因此，onReceive()方法不能执行耗时操作，否则将导致ANR。  （2）广播接收器注册  静态注册：在AndroidManifest.xml里通过<receive>标签声明。  动态注册：在代码中调用Context.registerReceiver（）方法  动态广播最好在Activity 的 onResume()注册、onPause()注销 unregisterReceiver(mBroadcastReceiver);    （3）广播发送者向AMS发送广播  广播的发送：广播是用”意图（Intent）“标识”，通过sendBroadcast（）方法发送出去。  广播的类型主要分为5类：  普通广播（Normal Broadcast）  系统广播（System Broadcast）  有序广播（Ordered Broadcast）  粘性广播（Sticky Broadcast）  App应用内广播（Local Broadcast）  7、有序广播：发送出去的广播被广播接收者按照先后顺序接收。  广播接受者接收广播的顺序规则，按照Priority属性值从大-小排序。  Priority属性相同者，动态注册的广播优先；  先接受到的可以进行拦截。  8、App应用内广播（Local Broadcast）  App应用内广播可理解为一种局部广播，广播的发送者和接收者都同属于一个App。  相比于全局广播（普通广播），App应用内广播优势体现在：安全性高 & 效率高。 |

BroadcastReceiver（广播接收器），属于 Android 四大组件之一。在 Android 开发中，BroadcastReceiver 的应用场景非常多。

# 1. 定义

即 广播，是一个全局的监听器，属于Android四大组件之一。

Android 广播分为两个角色：广播发送者、广播接收者。

# 2. 作用

监听 / 接收 应用 App 发出的广播消息，并 做出响应。

# 3. 应用场景

Android不同组件间的通信（含 ：应用内 / 不同应用之间）；

多线程通信；

与 Android 系统在特定情况下的通信；

如：电话呼入时、网络可用时。

# 4. 实现原理

## 4.1 采用的模型

Android中的广播使用了设计模式中的**观察者模式**：基于消息的发布 / 订阅事件模型。因此，Android将广播的**发送者 和 接收者 解耦**，使得系统方便集成，更易扩展。

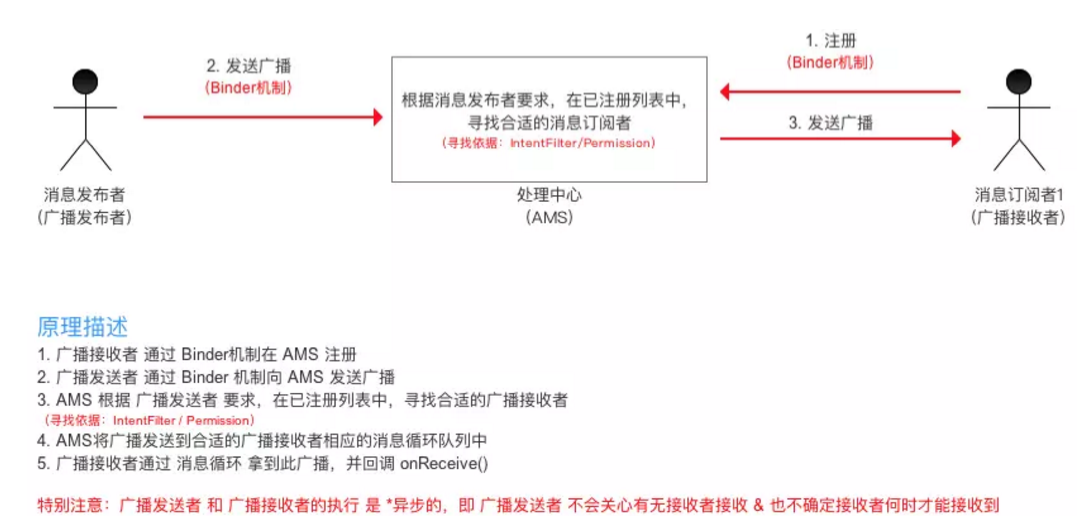
## 4.2 模型讲解

模型中有3个角色：

消息订阅者（广播接收者）

消息发布者（广播发布者）

消息中心（AMS，即Activity Manager Service）



# 5. 使用流程

使用流程如下：

## IMG_2565.1 自定义广播接收者BroadcastReceiver

继承BroadcastReceivre基类；

必须复写抽象方法onReceive()方法。

广播接收器接收到相应广播后，会自动回调 onReceive() 方法。

一般情况下，onReceive方法会涉及 与 其他组件之间的交互，如发送Notification、启动Service等。

默认情况下，广播接收器运行在 UI 线程，因此，onReceive()方法不能执行耗时操作，否则将导致ANR。

代码范例  
 mBroadcastReceiver.java

// 继承BroadcastReceivre基类public class mBroadcastReceiver extends BroadcastReceiver {

// 复写onReceive()方法

// 接收到广播后，则自动调用该方法

@Override

public void onReceive(Context context, Intent intent) {

//写入接收广播后的操作

}

}

## 5.2 广播接收器注册

注册的方式分为两种：静态注册、动态注册

### 5.2.1 静态注册

注册方式：在AndroidManifest.xml里通过**<receive>**标签声明

属性说明：

<receiver

android:enabled=["true" | "false"]

//此broadcastReceiver能否接收其他App的发出的广播

//默认值是由receiver中有无intent-filter决定的：如果有intent-filter，默认值为true，否则为false

android:exported=["true" | "false"]

android:icon="drawable resource"

android:label="string resource"//继承BroadcastReceiver子类的类名

android:name=".mBroadcastReceiver"//具有相应权限的广播发送者发送的广播才能被此BroadcastReceiver所接收；

android:permission="string"//BroadcastReceiver运行所处的进程//默认为app的进程，可以指定独立的进程//注：Android四大基本组件都可以通过此属性指定自己的独立进程

android:process="string" >

//用于指定此广播接收器将接收的广播类型//本示例中给出的是用于接收网络状态改变时发出的广播

<intent-filter><action android:name="android.net.conn.CONNECTIVITY\_CHANGE" />

</intent-filter></receiver>

注册示例

<receiver

//此广播接收者类是mBroadcastReceiver

android:name=".mBroadcastReceiver" >

//用于接收网络状态改变时发出的广播

<intent-filter>

<action android:name="android.net.conn.CONNECTIVITY\_CHANGE" />

</intent-filter></receiver>

当此 App首次启动时，系统会**自动**实例化mBroadcastReceiver类，并注册到系统中。

### 5.2.2 动态注册

注册方式：在代码中调用Context.registerReceiver（）方法

具体代码如下：

// 选择在Activity生命周期方法中的onResume()中注册

@Override

protected void onResume(){

super.onResume();

// 1. 实例化BroadcastReceiver子类 & IntentFilter

mBroadcastReceiver mBroadcastReceiver = new mBroadcastReceiver();

IntentFilter intentFilter = new IntentFilter();

// 2. 设置接收广播的类型

intentFilter.addAction(android.net.conn.CONNECTIVITY\_CHANGE);

// 3. 动态注册：调用Context的registerReceiver（）方法

registerReceiver(mBroadcastReceiver, intentFilter);

}

// 注册广播后，要在相应位置记得销毁广播

// 即在onPause() 中unregisterReceiver(mBroadcastReceiver)

// 当此Activity实例化时，会动态将MyBroadcastReceiver注册到系统中

// 当此Activity销毁时，动态注册的MyBroadcastReceiver将不再接收到相应的广播。

@Override

protected void onPause() {

super.onPause();

//销毁在onResume()方法中的广播

unregisterReceiver(mBroadcastReceiver);

}

}

特别注意

动态广播最好在Activity 的 onResume()注册、onPause()注销。

原因：

对于动态广播，有注册就必然得有注销，否则会导致**内存泄露；**

重复注册、重复注销也不允许。

**在onResume()注册、onPause()注销是因为onPause()在App死亡前一定会被执行，从而保证广播在App死亡前一定会被注销，从而防止内存泄露。**

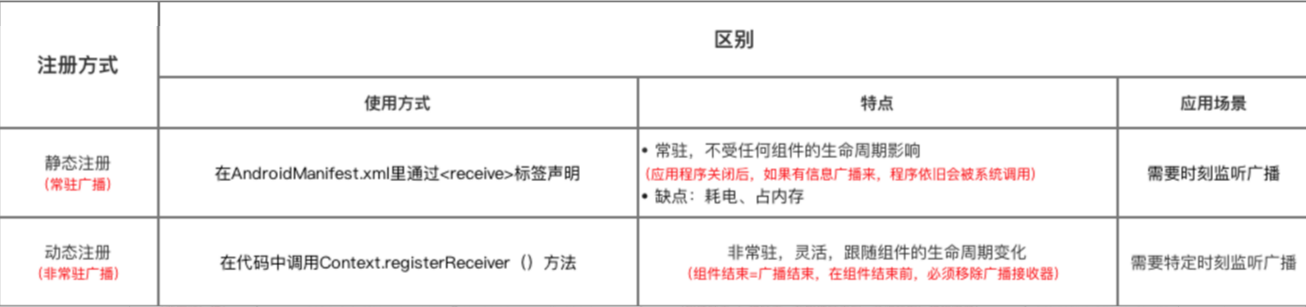
1、不在onCreate() & onDestory() 或 onStart() & onStop()注册、注销是因为：  
当系统因为内存不足（优先级更高的应用需要内存，请看上图红框）要回收Activity占用的资源时，Activity在执行完onPause()方法后就会被销毁，有些生命周期方法onStop()，onDestory()就不会执行。当再回到此Activity时，是从onCreate方法开始执行。

2、假设我们将广播的注销放在onStop()，onDestory()方法里的话，有可能在Activity被销毁后还未执行onStop()，onDestory()方法，即广播仍还未注销，从而导致内存泄露。

3、但是，onPause()一定会被执行，从而保证了广播在App死亡前一定会被注销，从而防止内存泄露。

（问题是，如果在onResume()执行的话，会不会过度执行，导致开销过大？不会。而是要考虑业务逻辑，当它退到后台之后stop之后就不应该接收到广播了。）

### 5.2.3 两种注册方式的区别



## 5.3 广播发送者向AMS发送广播

### 5.3.1 广播的发送

广播 是 用”意图（Intent）“标识

定义广播的本质 = 定义广播所具备的“意图（Intent）”

广播发送 = 广播发送者 将此广播的“意图（Intent）”通过**sendBroadcast（）方法**发送出去

### 5.3.2 广播的类型

广播的类型主要分为5类：

普通广播（Normal Broadcast）

系统广播（System Broadcast）

有序广播（Ordered Broadcast）

粘性广播（Sticky Broadcast）

App应用内广播（Local Broadcast）

具体说明如下：  
**1. 普通广播（Normal Broadcast）**  
即 开发者自身定义 intent的广播（最常用）。发送广播使用如下：

Intent intent = new Intent();//对应BroadcastReceiver中intentFilter的action

intent.setAction(BROADCAST\_ACTION);//发送广播

sendBroadcast(intent);

若被注册了的广播接收者中注册时intentFilter的action与上述匹配，则会接收此广播（即进行回调onReceive()）。如下mBroadcastReceiver则会接收上述广播

<receiver

//此广播接收者类是mBroadcastReceiver

android:name=".mBroadcastReceiver" >

//用于接收网络状态改变时发出的广播

<intent-filter>

<action android:name="BROADCAST\_ACTION" />

</intent-filter></receiver>

若发送广播有相应权限，那么广播接收者也需要相应权限

**2. 系统广播（System Broadcast）**

Android中内置了多个系统广播：只要涉及到手机的基本操作（如开机、网络状态变化、拍照等等），都会发出相应的广播

每个广播都有特定的Intent - Filter（包括具体的action），Android系统广播action如下：

| **系统操作** | **action** |
| --- | --- |
| 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 |

注：当使用系统广播时，只需要在注册广播接收者时定义相关的action即可，并不需要手动发送广播，当系统有相关操作时会自动进行系统广播

**3. 有序广播（Ordered Broadcast）**

定义  
 发送出去的广播被广播接收者按照先后顺序接收

有序是针对广播接收者而言的

广播接受者接收广播的顺序规则（同时面向静态和动态注册的广播接受者）

按照Priority属性值从大-小排序；

Priority属性相同者，动态注册的广播优先；

特点

接收广播按顺序接收

先接收的广播接收者可以对广播进行截断，即后接收的广播接收者不再接收到此广播；

先接收的广播接收者可以对广播进行修改，那么后接收的广播接收者将接收到被修改后的广播

具体使用  
有序广播的使用过程与普通广播非常类似，差异仅在于广播的发送方式：

sendOrderedBroadcast(intent);

**4. App应用内广播（Local Broadcast）**

背景  
Android中的广播可以跨App直接通信（exported对于有intent-filter情况下默认值为true）

冲突  
可能出现的问题：

其他App针对性发出与当前App intent-filter相匹配的广播，由此导致当前App不断接收广播并处理；

其他App注册与当前App一致的intent-filter用于接收广播，获取广播具体信息；  
 即会出现安全性 & 效率性的问题。

解决方案  
使用App应用内广播（Local Broadcast）

App应用内广播可理解为一种局部广播，广播的发送者和接收者都同属于一个App。

相比于全局广播（普通广播），App应用内广播优势体现在：安全性高 & 效率高。

具体使用1 - 将全局广播设置成局部广播

注册广播时将exported属性设置为false，使得非本App内部发出的此广播不被接收；

在广播发送和接收时，增设相应权限permission，用于权限验证；

发送广播时指定该广播接收器所在的包名，此广播将只会发送到此包中的App内与之相匹配的有效广播接收器中。

通过**intent.setPackage(packageName)**指定报名

具体使用2 - 使用封装好的LocalBroadcastManager类  
使用方式上与全局广播几乎相同，只是注册/取消注册广播接收器和发送广播时将参数的context变成了LocalBroadcastManager的单一实例

注：对于LocalBroadcastManager方式发送的应用内广播，只能通过LocalBroadcastManager动态注册，不能静态注册。

//注册应用内广播接收器

//步骤1：实例化BroadcastReceiver子类 & IntentFilter mBroadcastReceiver

mBroadcastReceiver = new mBroadcastReceiver();

IntentFilter intentFilter = new IntentFilter();

//步骤2：实例化LocalBroadcastManager的实例

localBroadcastManager = LocalBroadcastManager.getInstance(this);

//步骤3：设置接收广播的类型

intentFilter.addAction(android.net.conn.CONNECTIVITY\_CHANGE);

//步骤4：调用LocalBroadcastManager单一实例的registerReceiver()方法进行动态注册

localBroadcastManager.registerReceiver(mBroadcastReceiver, intentFilter);

//取消注册应用内广播接收器

localBroadcastManager.unregisterReceiver(mBroadcastReceiver);

//发送应用内广播

Intent intent = new Intent();

intent.setAction(BROADCAST\_ACTION);

localBroadcastManager.sendBroadcast(intent);

**5. 粘性广播（Sticky Broadcast）**  
由于在Android5.0 & API 21中已经失效，所以不建议使用，在这里也不作过多的总结。

# 6. 特别注意

对于不同注册方式的广播接收器回调OnReceive（Context context，Intent intent）中的context返回值是不一样的：

对于静态注册（全局+应用内广播），回调onReceive(context, intent)中的context返回值是：ReceiverRestrictedContext；

对于全局广播的动态注册，回调onReceive(context, intent)中的context返回值是：Activity Context；

对于应用内广播的动态注册（LocalBroadcastManager方式），回调onReceive(context, intent)中的context返回值是：Application Context。

对于应用内广播的动态注册（非LocalBroadcastManager方式），回调onReceive(context, intent)中的context返回值是：Activity Context.

广播发送后，我们注册的广播接收器就能接收到广播，并调用接收器中的onReceive()方法，整个广播的发送与接收流程也就完成了。要注意的是，静态注册广播接收器的特点是即使应用销毁后也会接收广播，而动态注册广播接收器则相反，它的生命周期十分短暂，大概在10秒左右。每次当有匹配的广播发送过来时，它会创建BroadcastReceiver，并执行里面的onReceive()方法，然后10秒左右就会被销毁，所以我们在onReceive()方法种尽量不要进行太长的耗时操作。此外，我们也最好不要再BroadcastReceiver中创建子线程进行耗时操作，因为BroadcastReceiver在执行onReceive()结束任务后，它的进程优先级会处于空进程的优先级，随时随地会被系统给回收销毁，而依赖于该进程的子线程也会随即被销毁。所以最好的处理方法是在onReceive()中启动Service，将耗时操作交给Service去处理。

（这个10s问题，处理方案有待考证？）

//静态注册  
 <receiver android:name=".widget.MyBroadcastReceiver">

<intent-filter android:priority="100"> //设置优先级

<action android:name="com.weimore.example.MY\_BROADCAST"/>

</intent-filter>

</receiver>

[IMG_256](https://www.cnblogs.com/weimore/p/javascript:void(0);)

//动态注册   
 IntentFilter filter = new IntentFilter("com.weimore.example.MY\_BROADCAST");

filter.setPriority(100);

如果某个广播接收器想要拦截该广播，只需要在onReceive()方法中调用abortBroadcast()方法就可以了。

@Override

public void onReceive(Context context, Intent intent) {

//拦截广播 abortBroadcast();

}