



上海建桥学院

SHANGHAI JIAN QIAO UNIVERSITY



# 广播

万永权

# 学习目标/Target



熟悉广播机制的概述，能够归纳广播机制的实现流程

掌握广播接收者的创建方式，能够独立创建广播接收者

掌握自定义广播的方式，能够通过自定义广播实现饭堂小广播案例

熟悉广播的类型，能够归纳有序广播与无序广播的工作流程

# 章节概述 / Summary



在Android系统中，广播是一种运用在组件之间传递消息的机制，例如电池电量低时会发送一条提示广播。如果要接收并过滤广播中的消息，则需要使用BroadcastReceiver（广播接收者），广播接收者是Android四大组件之一，通过广播接收者可以监听系统中的广播消息，实现在不同组件之间的通信，本章将针对广播及广播接收者进行详细讲解。



# Contents

## 主要内容

- 1. 广播机制的概述
- 2. 广播接收者
- 3. 自定义广播与广播的类型





# 广播机制的概述

# 什么是广播？

- Android中的广播和我们传统意义上的广播有很多相似之处。
- 之所以叫广播是因为发送者只负责“说”而不管接受者“听不听”。



发送信号



接收信号



# // 广播机制的概述

- 为了便于发送和接收系统级别的消息通知，Android系统也引入了一套类似广播的消息机制。
- Android中的广播(Broadcast)机制用于进程/线程间通信，该机制使用了观察者模式，观察者模式是一种软件设计模式，该模式是基于消息的发布/订阅事件模型，该模型中的消息发布者是广播机制中的广播发送者，消息订阅者是广播机制中的广播接收者。
- 每个应用程序都可以对自己感兴趣的广播进行注册，这样该程序就只会收到自己所关心的广播内容。
- 这些广播可能是来自于系统的，也可能是来自于其他应用程序的。
- Android提供了一套完整的API，允许应用程序自由地发送和接收广播。

# // 广播机制的概述

广播机制的具体实现流程，如下图所示。





# // 广播机制的概述

广播作为Android组件间的通信方式，可以使用的场景有以下几种：

在同一个APP内部的同一组件内进行消息通信。

第一种场景

在同一个APP具有多个进程的不同组件之间进行消息通信。

第二种场景

Android系统与APP之间进行消息通信。

第三种场景

在同一个APP内部的不同组件之间进行消息通信。

第四种场景

第五种场景

在不同APP的组件之间进行消息通信。



# 广播接收者

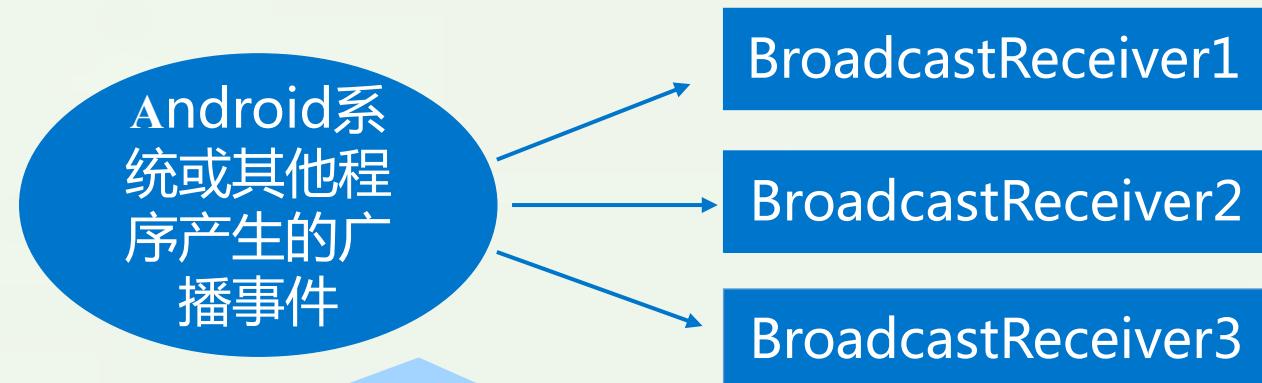
- 熟悉广播接收者的概念，能够归纳多个广播接收者接收广播的过程
- 掌握广播接收者的创建方式，能够独立创建广播接收者



# // 什么是广播接收者

Android系统中内置了很多广播，例如手机开机完成、电池电量不足时都会发送一条广播。

为了监听来自系统或者应用程序的广播事件，Android系统提供了BroadcastReceiver（广播接收者）组件。



当Android系统产生一个广播事件时，可以有多个对应的广播接收者接收并进行处理。这些广播接收者只需要在清单文件或者代码中进行注册并指定要接收的广播事件即可。



# // 创建广播接收者

广播接收者的创建方式有两种，具体如下：

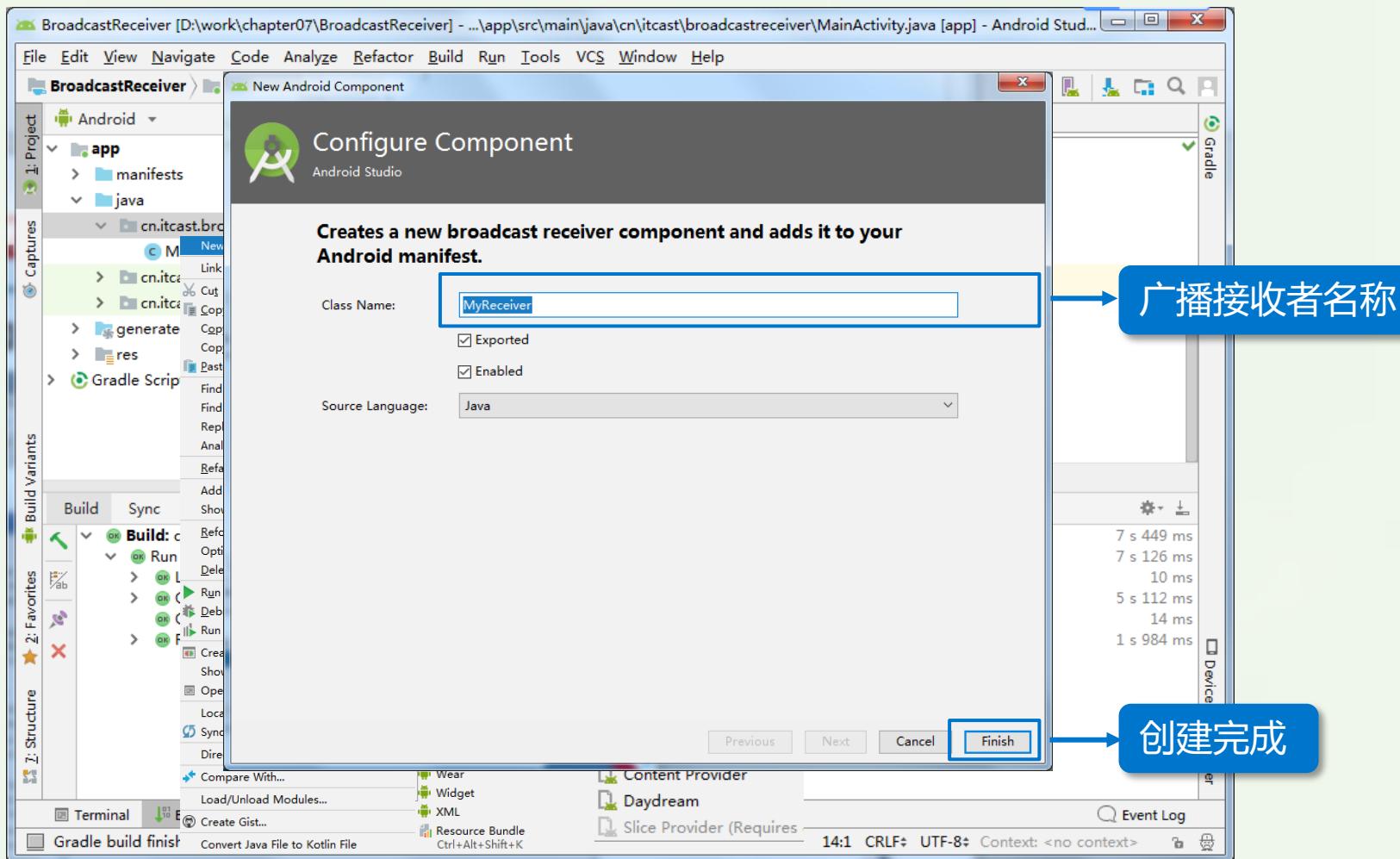
一种是通过在应用程序的包中创建一个类继承BroadcastReceiver并重写onReceive()方法来实现的。

一种是通过选中应用程序中的包，右击选择【New】→【Other】→【Broadcast Receiver】选项来创建的。

注意：创建完广播接收者之后还需要对广播接收者进行注册才可以接收广播。

# // 创建广播接收者

选择【New】→【Other】→【Broadcast Receiver】选项来创建广播接收者





# // 创建广播接收者

创建完成的广播接收者MyReceiver 的代码如下：

```
public class MyReceiver extends BroadcastReceiver {  
    public MyReceiver() {  
    }  
    @Override  
    public void onReceive (Context context, Intent intent) {  
        throw new UnsupportedOperationException("Not yet implemented");  
    }  
}
```

在该方法中实现广播接收者的相关操作

Android规定BroadcastReceiver类中的onReceiver()方法必须在5秒内执行完成，否则系统会认为该组件失去响应，并会提示用户强行关闭组件。



# // 创建广播接收者

广播接收者的注册方式有两种，分别是动态注册和静态注册。

1

动态注册：在Activity中通过代码注册广播接收者

2

静态注册：在清单文件中配置广播接收者

## // 注册广播

### □ 两种注册方式的区别：

代码注册的接受器不是常驻型接收器，也就是说当应用程序结束后该接受器也就失效了。

在配置文件中注册的接收器，不管程序有没有运行，只要有广播发送过来，程序的广播接受器会被系统调用自动运行。



# // 创建广播接收者

## 动态注册

```
receiver = new MyBroadcastReceiver(); //实例化广播接收者  
//实例化过滤器并设置要过滤的广播  
String action = "android.provider.Telephony.SMS_RECEIVED";  
IntentFilter intentFilter = new IntentFilter();  
intentFilter.addAction(action);  
registerReceiver(receiver,intentFilter); //注册广播
```



动态注册的广播接收者是否被注销依赖于注册广播的组件，当组件销毁时，广播接收者也随之被注销。



# // 创建广播接收者

## 动态注册

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    MyReceiver receiver = new MyReceiver();  
    String action = "android.provider.Telephony.SMS_RECEIVED";  
    IntentFilter intentFilter = new IntentFilter();  
    intentFilter.addAction(action);  
    registerReceiver(receiver,intentfilter);  
}  
protected void onDestroy() {  
    super.onDestroy();  
    unregisterReceiver(receiver);  
}
```

实例化过滤器并设置要过滤的action

注册广播

当Activity销毁时，取消注册



# // 创建广播接收者

## 静态注册

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ..... >
    <application ..... >
        <receiver
            android:name=".MyReceiver"
            android:enabled="true"
            android:exported="true" >
        </receiver>
    </application>
</manifest>
```

注意：在Android 8.0之后，使用静态注册的广播接收者将无法接收到广播。



# 注意：

- 其实从理论上来说，动态注册能监听到的系统广播，静态注册也应该能监听到，在过去的Android系统中确实是这样的。
- 但是由于大量恶意的应用程序利用这个机制在程序未启动的情况下监听系统广播，从而使任何应用都可以频繁地从后台被唤醒，严重影响了用户手机的电量和性能，因此Android系统几乎每个版本都在削减静态注册BroadcastReceiver的功能。
- 在Android 8.0系统之后，所有隐式广播都不允许使用静态注册的方式来接收了。隐式广播指的是那些没有具体指定发送给哪个应用程序的广播，大多数系统广播属于隐式广播，但是少数特殊的系统广播目前仍然允许使用静态注册的方式来接收。
- 这些特殊的系统广播列表详见
  - ❖ <https://developer.android.google.cn/guide/components/broadcastexceptions.html>。
- 在这些特殊的系统广播当中，有一条值为**android.intent.action.BOOT\_COMPLETED**的广播，这是一条开机广播，那么就使用它来举例学习吧。



# // 常见标准广播

常量	意义
android.intent.action.ANSWER	呼入电话
android.intent.action.Send	发送邮件
android.provider.Telephony.SMS_RECEIVED	接收短信
android.intent.action. ACTION_POWER_CONNECTED	外部电源连接
android.intent.action. ACTION_POWER_DISCONNECTED	外部电源断开
android.intent.action.BATTERY_LOW	电池电量低
android.intent.action.BOOT_COMPLETED	系统启动

# 标注广播案例

## ■ 接收网络状态变化广播

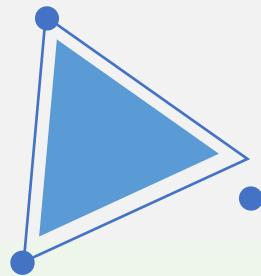
```
107     class NetworkChangeReceiver extends BroadcastReceiver {  
108         public void onReceive(Context context, Intent intent){  
109             Log.i( tag: "broad", msg: "receive 网络变化");  
110             Toast.makeText(context, text: "network is unavailable", Toast.LENGTH_SHORT).show();  
111         }  
112     }  
113 }  
114 }
```

```
34     networkChangeReceiver = new NetworkChangeReceiver();  
35     IntentFilter intentFilter2 = new IntentFilter();  
36     intentFilter2.addAction("android.net.conn.CONNECTIVITY_CHANGE");  
37     registerReceiver(networkChangeReceiver,intentFilter2);  
38 }
```

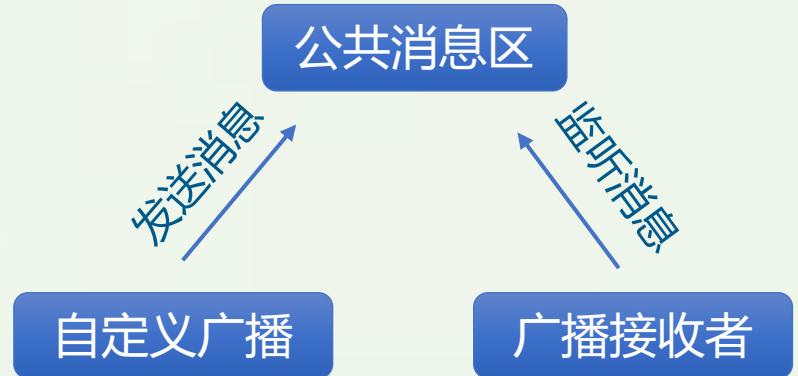




# 自定义广播与广播的类型



当系统提供的广播不能满足实际需求时，可以自定义广播，同时需要编写对应的广播接收者。



当自定义广播发送消息时，会储存到公共消息区中，而公共消息区中如果存在对应的广播接收者，就会及时的接收这条信息。

# // 实战演练—饭堂小广播

本节将通过一个饭堂小广播发送吃饭信号的案例来演示[自定义广播的发送和接收](#)。本案例的界面效果如下图所示。

- ① 创建名为CanteenRadio的程序
- ② 指定包名为cn.itcast.canteenradio
- ③ 导入界面图片
- ④ 放置界面控件

**1 搭建界面布局：**⑤ 修改默认标题栏名称

- ① 创建广播接收者MyBroadcastReceiver

**2 实现界面功能：**② 实现发送开饭消息的广播功能

- ① 运行程序
- ② 点击喇叭图片

**3 运行程序：**





# // 广播的类型

Android系统提供了**两种广播类型**，**有序广播**和**无序广播**，开发者可根据需求为程序设置不同的广播类型。

## 无序广播

无序广播是完全异步执行，发送广播时所有监听这个广播的广播接收者都会接收到此消息，但接收的顺序不确定。

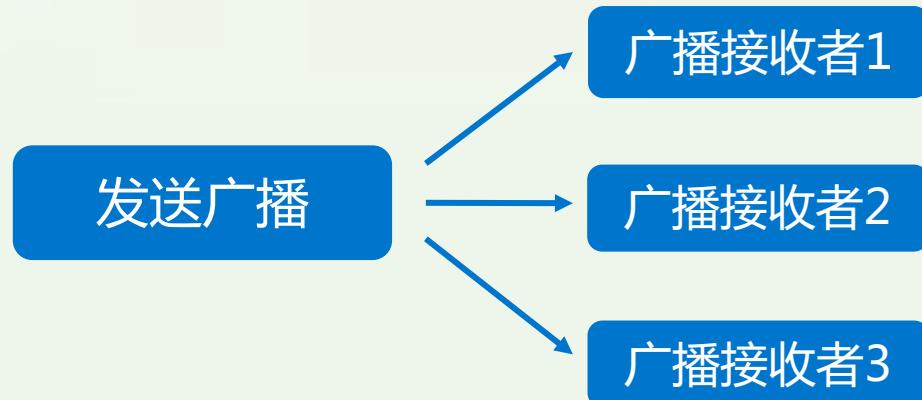
## 有序广播

按照接收者的优先级接收，只有一个广播接收者能接收消息，在此广播接收者中逻辑执行完毕后，才会继续传递。

# // 广播的类型

## 无序广播

无序广播的效率比较高，但无法被拦截，当发送一条广播消息时，所有的广播接收者都会接收到此消息。无序广播的工作流程如下：



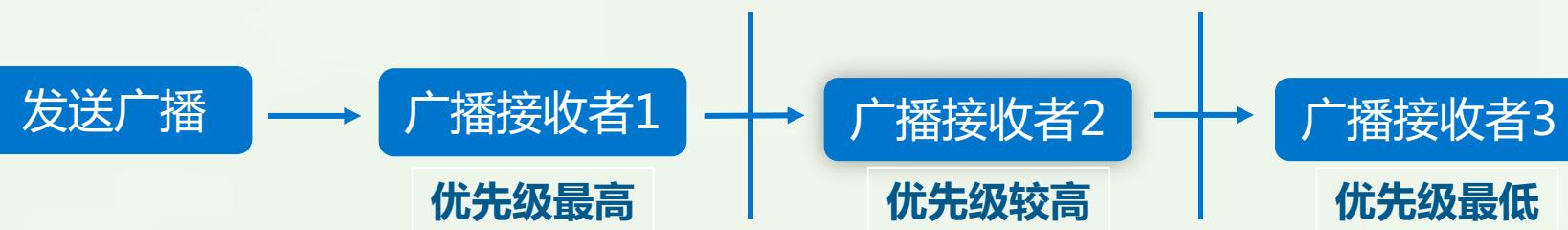
也称为：标准广播 (normal broadcasts)



# // 广播的类型

## 有序广播

相比无序广播，有序广播的广播效率较低，但此类型是有先后顺序的，并可被拦截。有序广播的工作流程如下：



# // 广播的类型

- 按照接收方声明的优先级别进行，该声明在intent-filter元素的android:priority属性中，数值越大优先级别越高，也可以通过IntentFilter对象的setPriority()方法进行设置。
- 接收方依次接收广播，同时前面的接收方有权结束广播的传播。
- 在广播发出之后，同一时刻只会有一个BroadcastReceiver能够收到这条广播消息；当这个BroadcastReceiver中的逻辑执行完毕后，广播才会继续传递。



## 有序广播的优先级

```
//动态注册MyReceiver广播  
MyReceiver one = new MyReceiver ();  
IntentFilter filter = new IntentFilter();  
filter.setPriority(1000);  
filter.addAction("Intercept_S");  
registerReceiver(one,filter);
```

数值越大，优先级越高。如果两个广播接收者的优先级相同，则先注册的广播接收者优先级高。



# // 多学一招



## 广播接收者优先级

在动态注册广播接收者时，可以使用IntentFilter对象的setPriority()方法设置优先级别，例如：intentFilter.setPriority(1000)。这里需要说明的是，属性值越大，优先级越高。

如果两个广播接收者的优先级相同，则先注册的广播接收者优先级高。也就是说，如果两个程序监听了同一个广播事件，同时设置了相同的优先级，则先安装的程序优先接收。

# // 实战演练—数鸭子

本节将通过一个有序数鸭子的案例来演示如何发送有序广播、根据广播接收者的优先级顺序接收广播、拦截广播，本案例的界面效果如下图所示。

- ① 创建名为CountDucks的程序
- ② 导入界面图片
- ③ 创建图片与控件的样式
- ④ 放置界面控件

**1 搭建界面布局：**⑤ 修改默认标题栏名称

- ① 创建3个广播接收者
- ② 动态注册广播接收者

**2 实现界面功能：**③ 实现发送有序广播的功能

- ① 运行程序
- ② 点击喇叭图片



# // 实战演练一数鸭子



# // 实战演练—数鸭子

若将广播接收者MyBroadcastReceiverTwo的优先级设置为1000，并将注册MyBroadcastReceiverTwo的语句放在注册MyBroadcastReceiverOne的语句前面，修改完MainActivity中注册的广播接收者代码后，运行程序，点击界面中的喇叭图片，效果如下。





# // 总结

✓ 本章详细地讲解了广播机制的相关知识，首先介绍了广播机制的概述，然后讲解了什么是广播接收者、广播接收者的创建、自定义广播以及广播的类型。通过本章的学习，要求初学者能够熟练掌握广播机制的使用，便于以后在实际开发中进行应用。