- https://developer.android.com/guide/components/broadcasts?hl=zh-cn
- 基础总结篇之五：BroadcastReceiver应用详解
- Oreo怎么收不到广播了？

BroadcastReceiver

Android 系统和其他 Android 应用之间可以相互首发广播，这与“发布-订阅”设计模式相似，下面介绍下广播相关内容。

- 系统广播
- 接收广播
  - 清单文件注册
  - 上下文注册
- 对进程状态的影响
  - 使用 goAsync 执行耗时异步任务
- 发送广播
- 通过权限限制广播
- 安全事项和最佳做法

系统广播

系统广播都有一个与其相关的 action，例如飞行模式 `android.intent.action.AIRPLANE_MODE` 它有一个 extra 字段用来表示飞行模式的开和关，另外还有开机广播，充电广播，灭屏，网络变化等，完整的系统广播列表可以在 SDK 中搜索 `BROADCAST_ACTIONS.TXT` 文件

接收广播

清单文件注册

在清单文件注册广播，不管应用有么有启动都可以收到广播，但是 android API26 之后，不能为隐式广播声明接收器，但有一些不受限制的广播例外。

在清单文件中注册广播需要如下步骤

- 1.在清单文件中添加 `<receiver>` 元素

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
...
<receiver android:name=".MyBroadcastReceiver"  android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <action android:name="android.intent.action.INPUT_METHOD_CHANGED" />
    </intent-filter>
</receiver>
```

- 2.创建 `BroadcastReceiver` 子类并实现 `onReceive(Context, Intent)`

```
private const val TAG = "MyBroadcastReceiver"

class MyBroadcastReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        StringBuilder().apply {
            append("Action: ${intent.action}\n")
            append("URI: ${intent.toUri(Intent.URI_INTENT_SCHEME)}\n")
            toString().also { log ->
                Log.d(TAG, log)
                Toast.makeText(context, log, Toast.LENGTH_LONG).show()
            }
        }
    }
}
```

系统软件包管理器会在应用安装的时候注册接收器，所以在应用未启动时，系统会启动应用并发送广播。

> 可以通过命令 `adb shell am start broadcast -a actionName` 来发送一条广播

上下文注册

上下文注册广播需要下面几步

- 1.创建 `BroadcastReceiver` 实例

```
val br: BroadcastReceiver = MyBroadcastReceiver()
```

- 2.创建 `IntentFilter` 并调用 `registerReceiver(BroadcastReceiver, IntentFilter)` 来注册接收器：

```
val filter = IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION).apply {
    addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED)
}
registerReceiver(br, filter)
```

- 3.停止接收广播，`unregisterReceiver(android.content.BroadcastReceiver)`,需要注意注销的位置，如果您使用 `Activity` 上下文在 `onCreate(Bundle)` 中注册接收器，则应在 `onDestroy()` 中注销，以防接收器从 `Activity` 上下文中泄露出去。如果您在 `onResume()` 中注册接收器，则应在 `onPause()` 中注销，以防多次注册接收器（如果您不想在暂停时接收广播，这样可以减少不必要的系统开销）

对进程状态的影响

当 `onReceive()` 的方法正在执行时，它被认为是前台进程，除非系统资源非常紧张才会被终结。但是一旦 `onReceiver()` 方法返回时，系统会将其视为低优先级进程，并可能会终结它，释放资源给其他应用。因此你不应该在其中启动长时间运行的线程，要避免优先级被降低这种情况，可以使用 `goAsync()` 方法，或者也可以使用 `Jobschedule` 调度 `Jobservice`。

如果你在 `onReceive()` 中进行的工作需要的时间长，会导致界面丢帧(> 16ms),则可以使用 `goAsync` 方法。

```
private const val TAG = "MyBroadcastReceiver"

class MyBroadcastReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        val pendingResult: PendingResult = goAsync()
        val asyncTask = Task(pendingResult, intent)
        asyncTask.execute()
    }

    private class Task(
            private val pendingResult: PendingResult,
            private val intent: Intent
    ) : AsyncTask<String, Int, String>() {

        override fun doInBackground(vararg params: String?): String {
            val sb = StringBuilder()
            sb.append("Action: ${intent.action}\n")
            sb.append("URI: ${intent.toUri(Intent.URI_INTENT_SCHEME)}\n")
            return toString().also { log ->
                Log.d(TAG, log)
            }
        }

        override fun onPostExecute(result: String?) {
            super.onPostExecute(result)
            // 必须调用 finish() 让 BroadcastReceiver 可以被回收
            pendingResult.finish()
        }
    }
}
```

发送广播

Android 有三种方式来发送广播

- `sendOrderedBroadcast(Intent, String)` 方法一次向一个接收器发送广播。当接收器逐个顺序执行时，接收器可以向下传递结果，也可以完全中止广播，使其不再传递给其他接收器（`abortBroadcast()`）。接收器的运行顺序可以通过匹配的 `intent-filter` 的 `android:priority` 属性来控制；**具有相同优先级的接收器将按随机顺序运行。**

- `sendBroadcast(Intent)` 方法会按随机的顺序向所有接收器发送广播。这称为常规广播。这种方法效率更高，但也意味着接收器无法从其他接收器读取结果，无法传递从广播中收到的数据，也无法中止广播。

```
Intent().also { intent ->
    intent.setAction("com.example.broadcast.MY_NOTIFICATION")
    intent.putExtra("data", "Notice me senpai!")
    sendBroadcast(intent)
}
```

- `LocalBroadcastManager.sendBroadcast` 方法会将广播发送给与发送器位于**同一应用中**的接收器。如果您不需要跨应用发送广播，请使用本地广播。这种实现方法的效率更高（无需进行进程间通信），而且您无需担心其他应用在收发您的广播时带来的任何安全问题。

通过权限限制广播

带权限的发送

在发送广播时, 有一个字段可以指定权限，只有声明了这些权限的应用可以接收到广播, `sendBroadcast(Intent, String)` 或 `sendOrderedBroadcast(Intent, String, BroadcastReceiver, Handler, int, String, Bundle)` 例如:

```
sendBroadcast(Intent("com.example.NOTIFY"), Manifest.permission.SEND_SMS)
```

要接收此广播，接收方必须声明

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

带权限的接收

如果你在注册广播接收器的时候指定了权限参数（通过 `registerReceiver(BroadcastReceiver, IntentFilter, String, Handler)` 或清单中的 `<receiver>` 标记指定），广播放必须通过其清单中请求该权限，例如接收方有如下声明的接收器

```
<receiver android:name=".MyBroadcastReceiver"
              android:permission="android.permission.SEND_SMS">
    <intent-filter>
        <action android:name="android.intent.action.AIRPLANE_MODE"/>
    </intent-filter>
</receiver>
```

或者你的接收器在上下文中使用如下方法注册了接收器

```
var filter = IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED)
    registerReceiver(receiver, filter, Manifest.permission.SEND_SMS, null )
```

那么发送方程序必须请求如下权限，才能向这些接收器发送广播

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

> 上面两个一个限制的是接收方需要申请权限，一个限制的是发送方需要申请权限，是不一样的哦

安全事项和最佳做法

- 如果您不需要向应用以外的组件发送广播，则可以使用支持库中提供的 `LocalBroadcastManager` 来收发本地广播。`LocalBroadcastManager` 效率更高（无需进行进程间通信），并且您无需考虑其他应用在收发您的广播时带来的任何安全问题。本地广播可在您的应用中作为通用的发布/订阅事件总线，而不会产生任何系统级广播开销。

- 如果有许多应用在其清单中注册接收相同的广播，可能会导致系统启动大量应用，从而对设备性能和用户体验造成严重影响。为避免发生这种情况，请优先使用上下文注册而不是清单声明。有时，Android 系统本身会强制使用上下文注册的接收器。例如，`CONNECTIVITY_ACTION` 广播只会传送给上下文注册的接收器。

- 请勿使用隐式 intent 广播敏感信息。任何注册接收广播的应用都可以读取这些信息。您可以通过以下三种方式控制哪些应用可以接收您的广播：

  - 您可以在发送广播时指定权限。
  - 在 Android 4.0 及更高版本中，您可以在发送广播时使用 `setPackage(String)` 指定软件包。系统会将广播限定到与该软件包匹配的一组应用。
  - 您可以使用 `LocalBroadcastManager` 发送本地广播。

- 当您注册接收器时，任何应用都可以向您应用的接收器发送潜在的恶意广播。您可以通过以下三种方式限制您的应用可以接收的广播：

  - 您可以在注册广播接收器时指定权限。
  - 对于清单声明的接收器，您可以在清单中将 android:exported 属性设置为“false”。这样一来，接收器就不会接收来自应用外部的广播。
  - 您可以使用 `LocalBroadcastManager` 限制您的应用只接收本地广播。

- 广播操作的命名空间是全局性的。请确保在您自己的命名空间中编写操作名称和其他字符串，否则可能会无意中与其他应用发生冲突。

- 由于接收器的 `onReceive(Context, Intent)` 方法在主线程上运行，因此它会快速执行并返回。如果您需要执行长时间运行的工作，请谨慎生成线程或启动后台服务，因为系统可能会在 `onReceive()` 返回后终止整个进程。建议：

  - 在接收器的 `onReceive()` 方法中调用 `goAsync()`，并将 `BroadcastReceiver.PendingResult` 传递给后台线程。这样，在从 `onReceive()` 返回后，广播仍可保持活跃状态。不过，即使采用这种方法，系统仍希望您非常快速地完成广播（在 10 秒以内）
  - 使用 `JobScheduler` 调度作业

- 请勿从广播接收器启动 `Activity`，否则会影响用户体验，尤其是有多个接收器时。相反，可以考虑使用通知。