**Wi-Fi P2P 无网络传输技术**

**---Chris**

# 一、简介

**（1）定义**：Wi-Fi P2P也称为WiFi直连（WiFi Direct），是一套无线网络互连协议。wifi设备之间不再需要无线网络接入点（Access Point），而是以点对点的方式，直接与另一个wifi设备连线，进行高速数据传输。

**（2）优点**：WiFi直连具有传输速度快（传输速度是近乎蓝牙速度的100倍）、效率高、不影响手机网络等优点。

**（3）应用：**手机互传、手机投屏等。

**（4）硬件要求：**Android 4.0及更高系统版本的设备。

# 二、主要组成

## （1）P2P Device

支持Wifi P2P协议的设备，比如手机、电脑、平板等，它是以下角色的实体，没成为Group Owner或Group Client前的设备都是“P2P Device”。

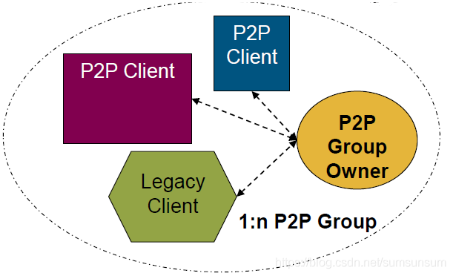
## （2）P2P Group Owner

简称GO，是协议中的一种角色（群主），相当于AP，一个组里只有一个GO。

## （3）P2P Group Client:

简称GC，是另一种角色（组员），一个组里可以有多个GC。

通过以上介绍可知，Wifi P2P的拓扑结构是1:n的，其中多个GC连接一个GO，这样的一个结构可以称为一个组，如下图所示。

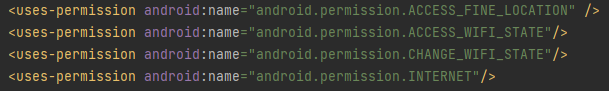


WiFi P2P Group结构图

# 三、实现流程

## （1）设置应用程序权限

在AndroidManifest.xml文件中加入以下权限：

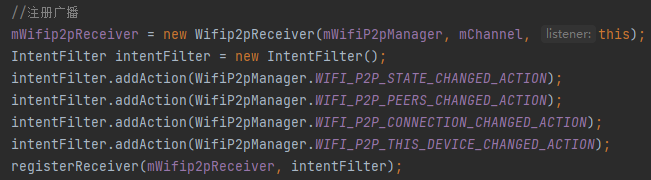


## （2）初始化WifiP2pManager

在BaseActivity文件中，获取系统P2P服务WifiP2pManager的实例，并通过调用initialize()获取Channel实例，大多数API的调用都将使用到mWifiP2pManager和mChannel：



## （3）创建广播接收器

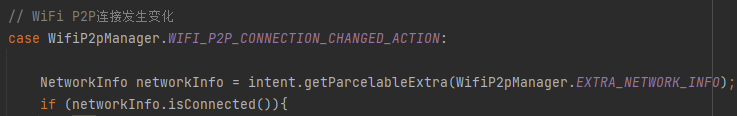
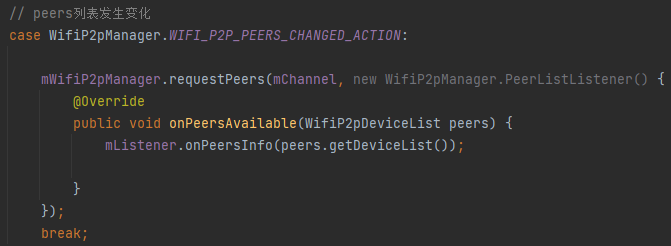
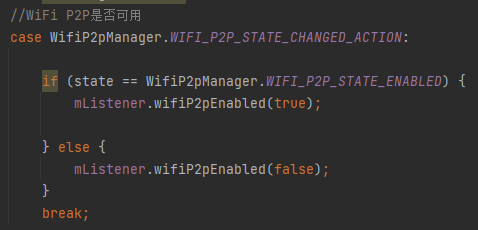


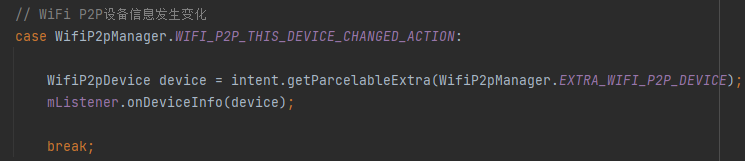
***WifiP2pManager.WIFI\_P2P\_STATE\_CHANGED\_ACTION***：检查Wi-Fi P2P是否已启用。

***WifiP2pManager.WIFI\_P2P\_PEERS\_CHANGED\_ACTION*：**对等设备发生变化，一般是在调用 discoverPeers方法后发送此广播。在此广播中，可以调用 requestPeers 方法，获得扫描到的对等设备列表。

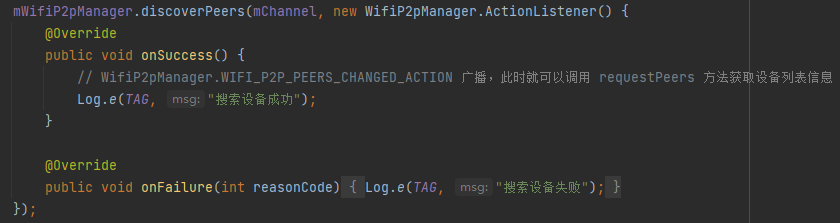
***WifiP2pManager.WIFI\_P2P\_CONNECTION\_CHANGED\_ACTION*：**连接状态发生变化，一般在调用connect或 cancelConnect 方法时会发送此广播。

***WifiP2pManager.WIFI\_P2P\_THIS\_DEVICE\_CHANGED\_ACTION*：**此设备的WiFi状态更改回调，应用可使用 requestDeviceInfo() 来检索当前连接信息。





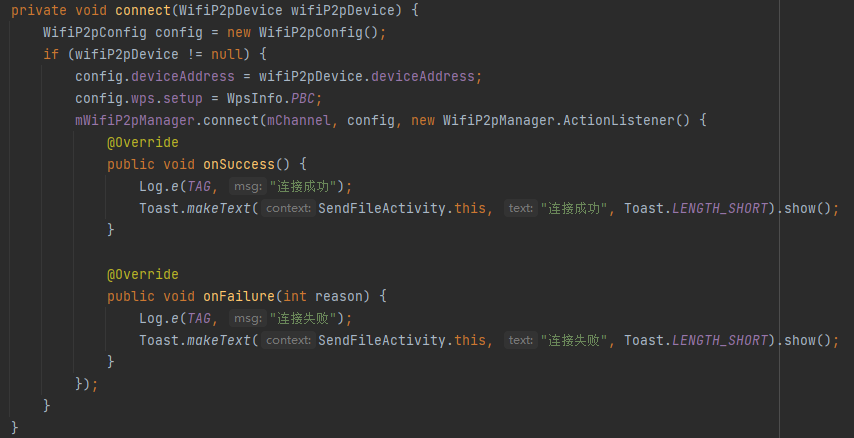
## （4）发现对等设备



## （5）查看设备信息



## （6）建立连接



# 四、注意事项

（1）当首次打开或重启APP后，点击“接收文件”选项时会自动创建群组，并且是该设备就是Group Owner；

（2）非首次打开或重启APP，谁创建群组则谁就是Group Owner，这也就意味着确定了谁是服务端（GO），谁是客户端（GC）；

（3）如果存在两台或多台设备都创建了群组，很明显这时就会有多个GO矛盾出现。因此，用户必须只选择其中某台设备作为GO，即其他设备（GC）的群组必须删除，然后GC设备点击发送文件-----建立配对连接-----点击对应GO设备，这样才可以正常建立连接。否则，不能进行正常连接和通信；

（4）从上述可以看出，GO设备只有负责接收数据，GC设备负责发送数据。原因是当成功建立连接后，从WifiP2pInfo中只能获取GO的ipAddress，并没有API可以提供其他Device的ipAdress。这个会导致上层应用无法获得对端ip地址，而无法传递数据等。

*思考：我们除了能获取GroupOwner的Ip Address外，还希望能获取对端以及Group中所有成员的Ip Address，如果可以成功获取所有设备的ip，是否GO也可以实现发送数据？*

（5）WifiP2pDevice 的意思是【本机设备信息】，不要误以为是获取到的是另外一台设备的设备信息。

其中常用有：

wifiP2pDevice.deviceName 本设备名字

wifiP2pDevice.deviceAddress 本设备的MAC地址

wifiP2pDevice.status 设备连接状态

# 五、问题汇总及解决

## （1）相关配置修改

1. Gradle Version 4.1 🡪 Gradle Version 4.10.1

2. Android Gradle plugin Version 3.0.0 🡪 Android Gradle plugin Version 3.3.0 (google())

3. compileSdkVersion 26 🡪 compileSdkVersion 28

4. buildToolsVersion "26.0.2" 🡪 buildToolsVersion "28.0.3"

5. targetSdkVersion 26 🡪 targetSdkVersion 28

6. com.android.support:appcompat-v7:26.+🡪 com.android.support:appcompat-v7:+

7. Grade JDK 1.8 ( version1.8.0)





## （2）调用WifiP2pWfdInfo类和WifiP2pManager. setWFDInfo()

Miracast服务发现中需要调用setWFDInfo()来启动WFD的设备发现功能，其中WifiP2pWfdInfo类与setWFDInfo()方法都是@hide的，在标准的sdk中无法直接进行访问，都需要采用反射方式进行调用。

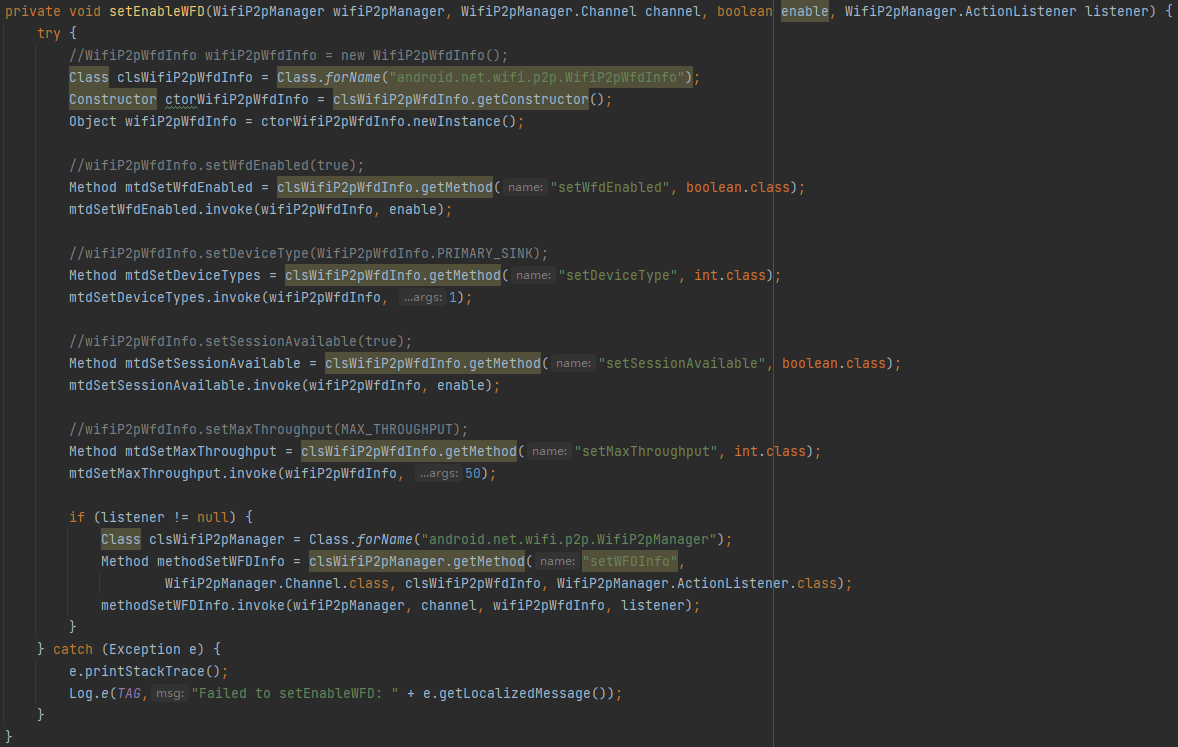
### 遇到的Error类型：

Attempt to invoke virtual method java.util.List “XXX“ on a null object reference 并且APP强制闪退

### 解决过程：

出现该问题原因是setWFDInfo()虽然是采用了反射调用的方法，但是设置WFD的相关参数有些在WifiP2pWfdInfo类中，比如wifiP2pWfdInfo.setWfdEnabled()等。刚开始尝试是将整个WifiP2pWfdInfo类从系统文件夹拷贝到工程文件夹，但经过实验证明该方法不起任何效果。之后尝试每个相关参数都采用反射方式调用，并且打开WFD的相关函数应该在广播注册时进行调用，问题成功解决。

反射调用代码如图：



### 反射调用主要流程说明：

Class clsWifiP2pWfdInfo = Class.forName("android.net.wifi.p2p.WifiP2pWfdInfo"); //返回给定的字符串名称相关联类或接口的Class对象；

Method mtdSetWfdEnabled = clsWifiP2pWfdInfo.getMethod("setWfdEnabled", boolean.class);//获取方法名称和参数列表；

mtdSetWfdEnabled.invoke(wifiP2pWfdInfo, enable);//调用method类代表的方法，其中wifiP2pWfdInfo是对象名，args是传入setWfdEnabled方法的参数；

## (3) 从Group Owner获取多个Group Client设备的ip地址

与设备建立连接的所有接口信息都存储在"/proc/net/arp"中，即ARP表。因此为了从程序内部获取GC端各个设备的ip，需要从程序内部读取arp表，并且在表内部进行相关信息分类与筛选，最终得到所需设备的IP。表中参数如图：



### 具体流程：

1. 读ARP表------*br = new BufferedReader(new FileReader("/proc/net/arp"));*
2. 分割ARP表中每行内容------String[] splitted = line.split(" +");
3. 内容分割后，获取表中设备的mac地址；------ String mac = splitted[3];
4. 将上面mac地址与程序中得到的device.deviceAddress（Mac地址）进行比较；若上述地址一致，则该设备就是要获取的GC设备；------ if (str1.equals(str2))
5. 从分割内容中找到mac地址对应设备的IP地址。------ String GCip = splitted[0];

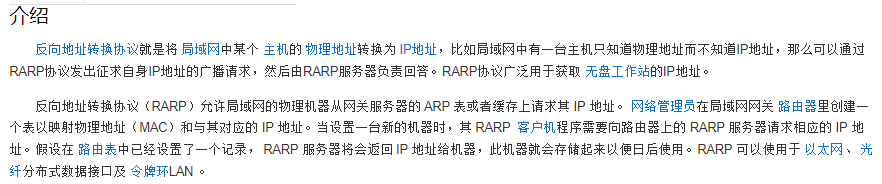
### 问题及注意事项：

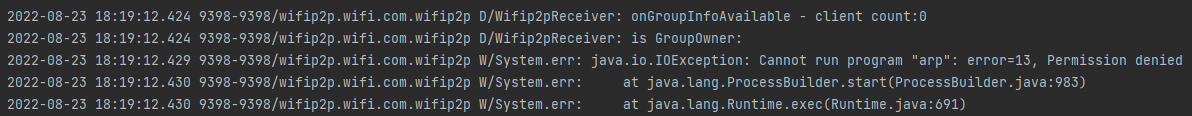
1. 调用getIPFromMac（String Mac）函数时，参数Mac是device.deviceAddress（Mac地址）。并且在读取ARP表之前需要有一段时间的延迟，这是因为当某个GC设备连接到GO 设备后，ARP表不会立马更新，而需要一段时间的缓冲。如果没有进行延时再读取，则读到的ARP表只会是上一次的状态。（一开始没有意识到该问题，导致读取的信息总是有一行误差）。

2. 表的标题也算一行，由于本次读表采用的是BufferedReader中FileReader("/proc/net/arp")方式，该方式是一次只读取一行，因此在进行读有用信息前，需要先读一遍表头，这样可以避免信息错位而产生误差。

3. 由于需要进行mac（表中）和Mac（程序中）的比较，即需要用到参数device.deviceAddress，而且如果有设备连接，则连接队列会发生变化，因此程序中在BaseActivity文件的onPeersInfo函数中调用getIPFromMac（）。

4. 利用Mac地址反解析IP地址的方法比较繁琐，而且有一定局限性，不建议采用。通常都是正向解析（由IP获取Mac）。





5.Q: Galaxy S9的mac地址读出来不一致（未找到原因）



### 主要的代码：





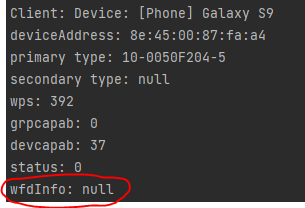
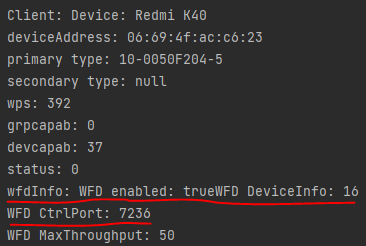
## (4) 从Group Owner获取多个Group Client设备的端口号

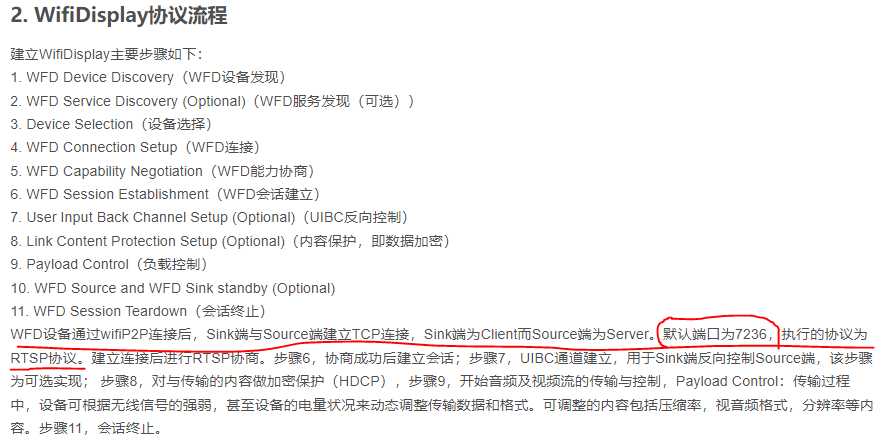
### 实验现象：

当小米手机RedmiK40连接GO后，GO端所得到的端口号为7236，而三星Galaxy S9连接GO后，GO端所得到的端口号为0，程序会将0调整为默认值7236。









### 具体流程：

1. 获取WifiP2pDevice类中声明的WifiP2pWfdInfo对象wfdInfo，相当于获取连接设备的WFD相关信息；

------Field field1=Class.forName(WifiP2pDevice.class.getName()).getDeclaredField("wfdInfo");

1. 获得device对象上此field1表示的字段值；

------ Object obj=field1.get(device);

1. 获得WifiPpWfdInfo类所对应的Class对象；

------ Class myClass=Class.forName(obj.getClass().getName());

1. 获取WifiP2pWfdInfo类中getControlPort方法；

------Method method=myClass.getMethod("getControlPort");

1. 调用上面所获取的方法，得到getControlPort的返回值；

------port= (int) method.invoke(obj);

### 主要的代码：



## (5) 将WifiP2p（app）打包成系统应用，获取系统签名。

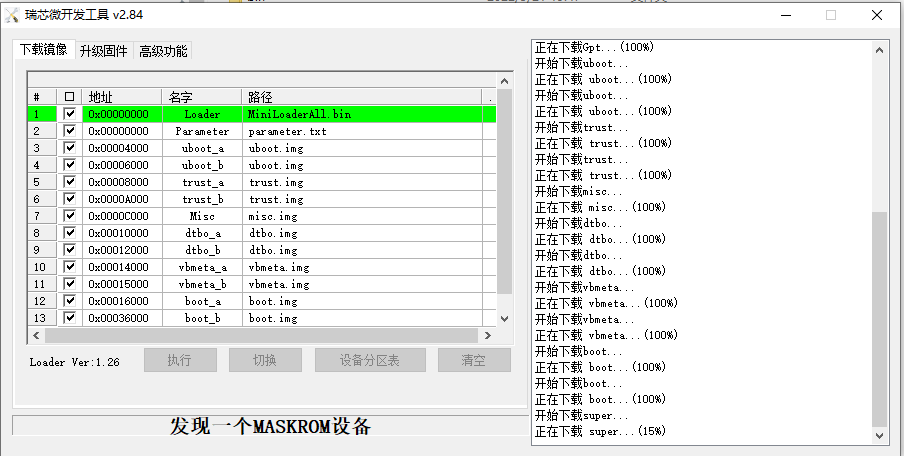
### WFD权限限制：

自Android 8.0及以后，官方已经限制了对setWFDInfo()接口的调用（这个接口本来也是@hide的，因此官方在高版本中对其进行限制也是理所当然），普通app已经没有权限进行调用了，也就是第三方app已经不能实现Sink端了。所以市面上的一些投屏软件，如：AirScreen，在高版本中会弹窗提示功能已被Google禁用。除非你是系统应用或者有系统签名才能突破此限制，此时你会收到类似如下的报错：Wifi Display Permission denied for uid = 10104。

因此在Android 8.0之后，第三方应用程序已经无法实现Sink接收端。如果你是厂商App，应用有系统权限，则可以绕过此限制。或者找厂商修改frameworks源码，增加app白名单开放此权限，也可以绕过，具体可在getWfdPermission()方法中增加过滤判断，然后返回true即可。本次实验采用的是在应用中加入系统权限。

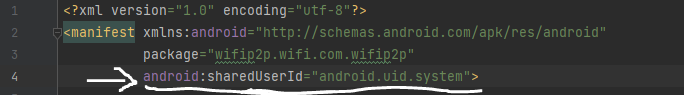
### 烧录详情：

更换Android版本（7、8、10），NonaT4-RK3399烧录详情链接：[NanoPC-T4/zh - FriendlyELEC WiKi](https://wiki.friendlyelec.com/wiki/index.php/NanoPC-T4/zh" \l "Windows.E4.B8.8B.E9.80.9A.E8.BF.87Type-C.E6.95.B0.E6.8D.AE.E7.BA.BF.E8.BF.9B.E8.A1.8C.E7.83.A7.E5.86.99) （目录4.4.2 Windows下通过Type-C数据线烧写所需文件及工具）。

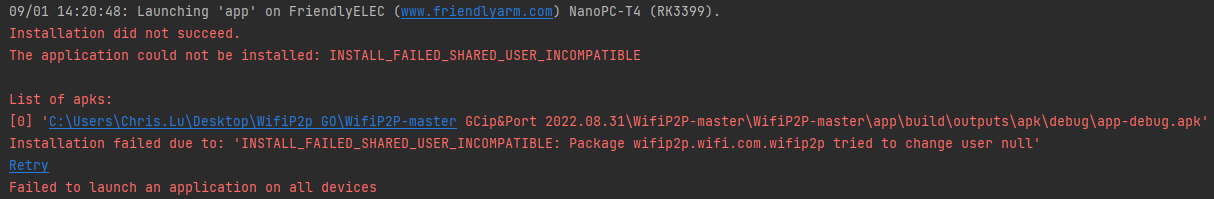
****

### 系统签名：

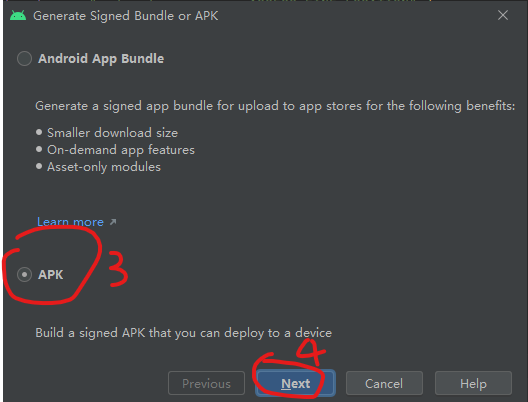
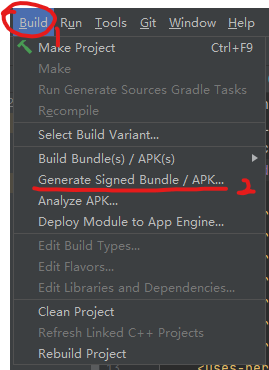
1. 在AndroidManifest.xml文件中加入 android:sharedUserId="android.uid.system"，如下：



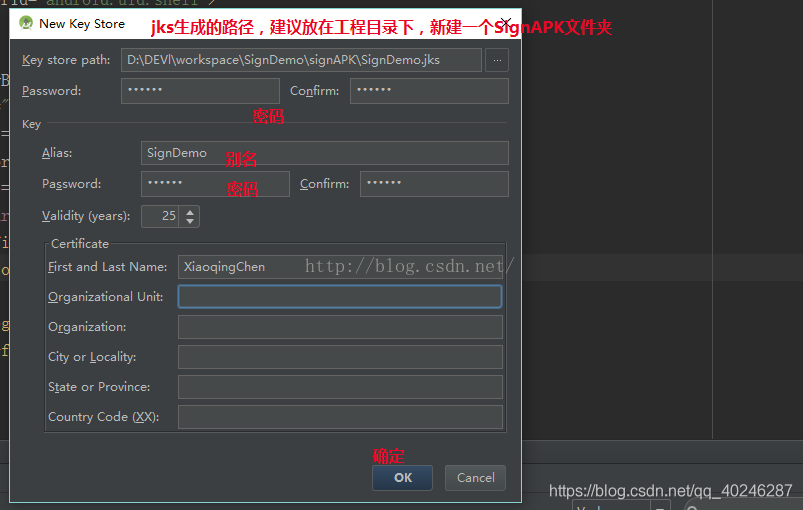
1. 添加android:sharedUserId="android.uid.system"编译成功后的apk是不能安装的 ，会出现如下错误提示，这是因为系统应用使用了统一的签名，而应用的签名和系统的签名不一样，所以接下来的事情就是怎么将自己的应用签名改成系统应用签名了。



1. 在Android studio中生成普通开发者签名的jks文件：



然后点击creat new--------点击next--------按下图提示完成设置------最后点击finsh



1. 需要准备的文件：

**·**keytool-importkeypair （[下载](https://github.com/getfatday/keytool-importkeypair)：[github.com](https://github.com/getfatday/keytool-importkeypair)），下载好解压（我是将解压后的文件夹直接放在工程根目录中）；

**·**platform.x509.pem、platform.pk8（源码中位于/build/target/product/security，本次实验是由徐老师提供已经下载好的）；

1. 使用keytool-importkeypair对jks文件引入系统签名（关键一步）：

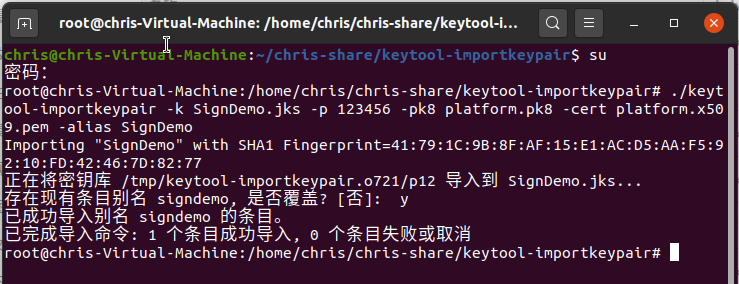
**·** 这一步本质的原理是给apk加上开发者签名（jks文件），使用keytool-importkeypair 对jks文件进行系统签名，在出包的时候，直接使用带有系统签名的jks对apk进行签名，这样编译生成的apk文件就自带系统签名了。

**·**将上述生成的SignDemo.jks(普通开发者签名的jks文件)、platform.x509.pem、platform.pk8都复制到keytool-importkeypair解压后的文件夹中（在工程目录下）。

**·**打开终端（注：这里终端是指linux系统终端，MAC终端应该也可以，但windows终端不行），终端进入超级用户模式，再进入到刚刚放置上述文件的keytool-importkeypair文件夹中，执行如下命令：

./keytool-importkeypair -k [jks文件名] -p [jks的密码] -pk8 platform.pk8 -cert platform.x509.pem -alias [jks的别名]

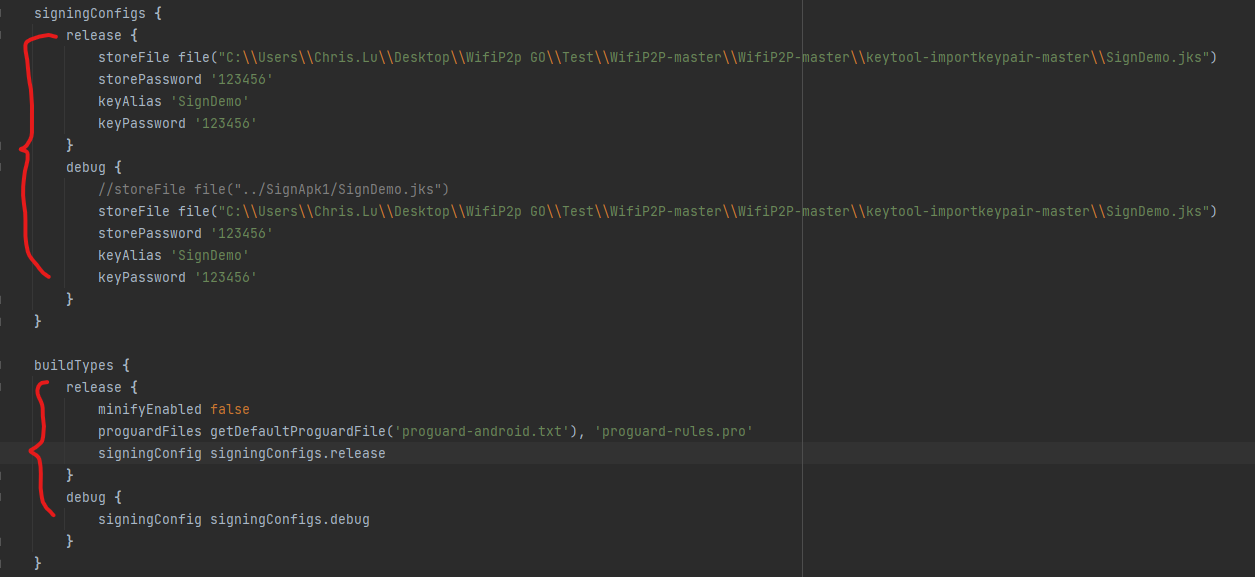
（如：./keytool-importkeypair -k SignDemo.jks -p 123456 -pk8 platform.pk8 -cert platform.x509.pem -alias SignDemo）



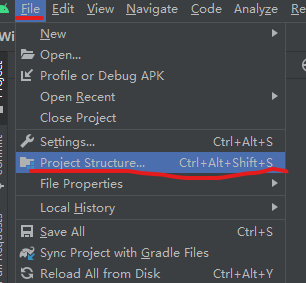
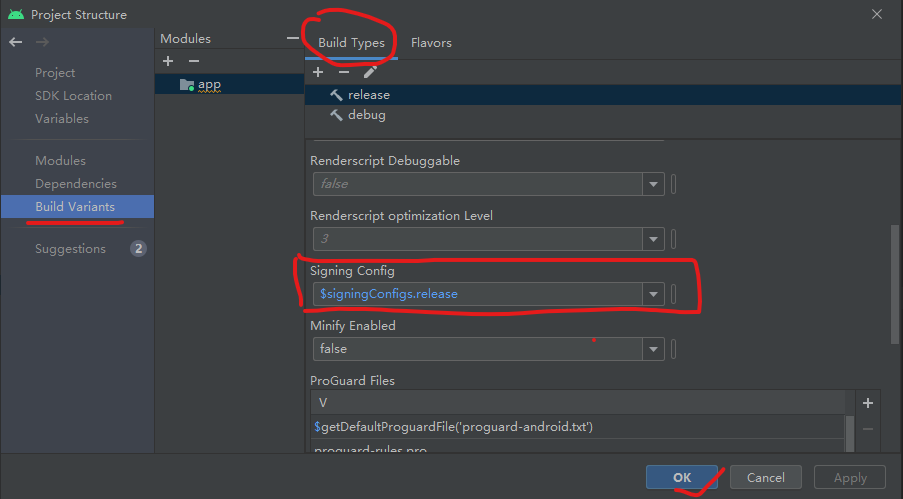
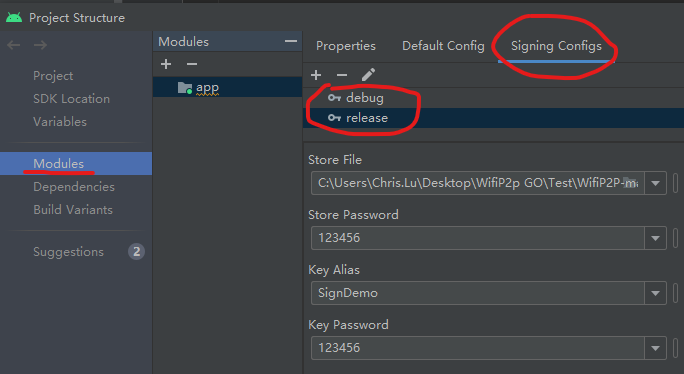
* 如上图，“1个条目成功导入”说明关键的一步执行成功，去对应共享文件夹查看，发现之前的SignDemo.jks(普通开发者签名的jks文件)被新生成的SignDemo.jks(写入了系统签名的jks文件)覆盖**；**

1. 配置gradle（app）

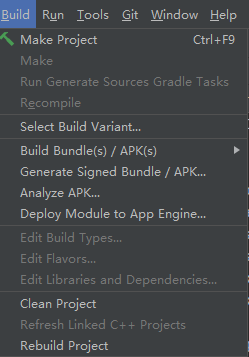
* 在android区域下(与defaultConfig同级)添加signingConfigs配置，依次填写jks的路径，密码，别名等。



* 也可手动设置，自动生成代码：

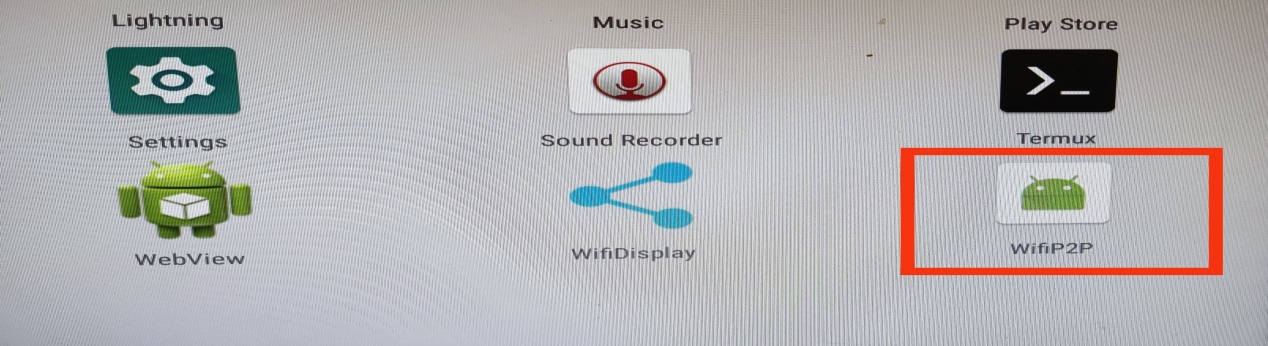
 

1. 最在Andriod studio中选择Build------Rebuild Project------Run------成功在开发板安装APP

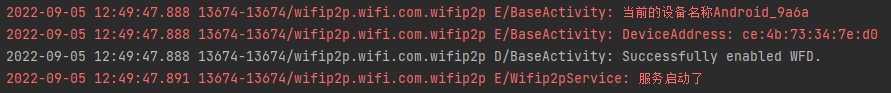


### 实验结果Android 8：

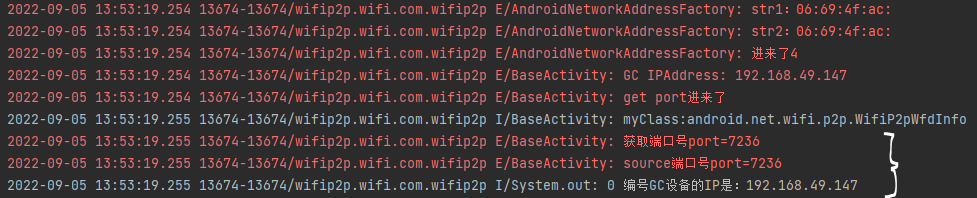
app成功在Android 8版本的板子中成功安装，运行结果如下：



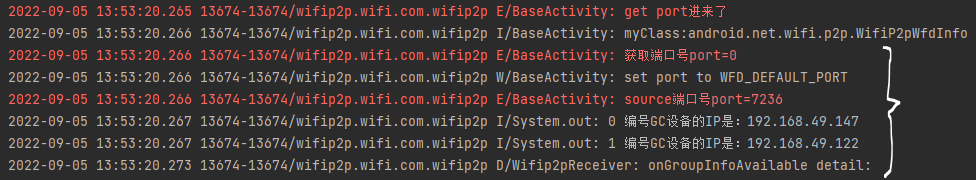
1. 成功调用setWFDInfo：在GC设备端的WifiDisplay功能中成功搜索到该设备；



1. 当某台GC设备连接到GO时，GO设备可以成功获取到该GC设备的IP及端口号；

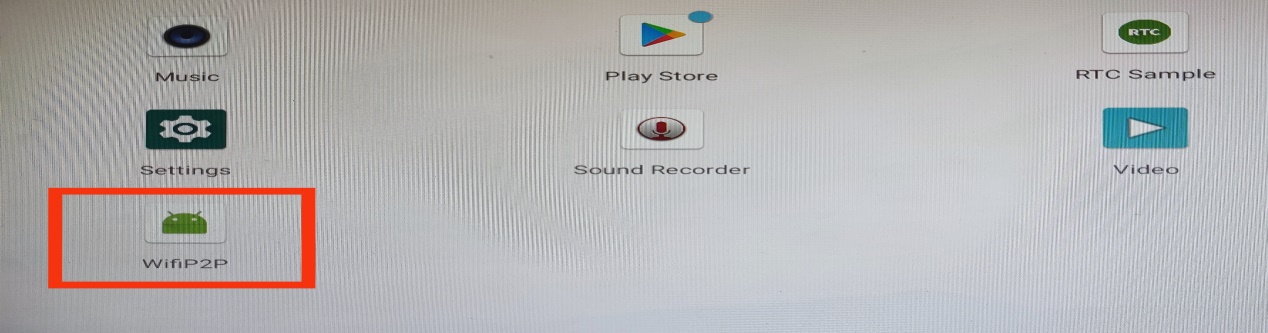


1. 当多台GC设备连接到GO设备时，GO设备能够知道组内所有成员的IP，并将各个GC设备的IP存放在列表list中。

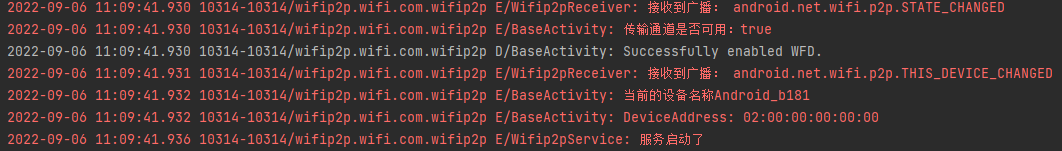


### 实验结果Android 10：

app成功在Android 10版本的板子中成功安装，运行结果如下：



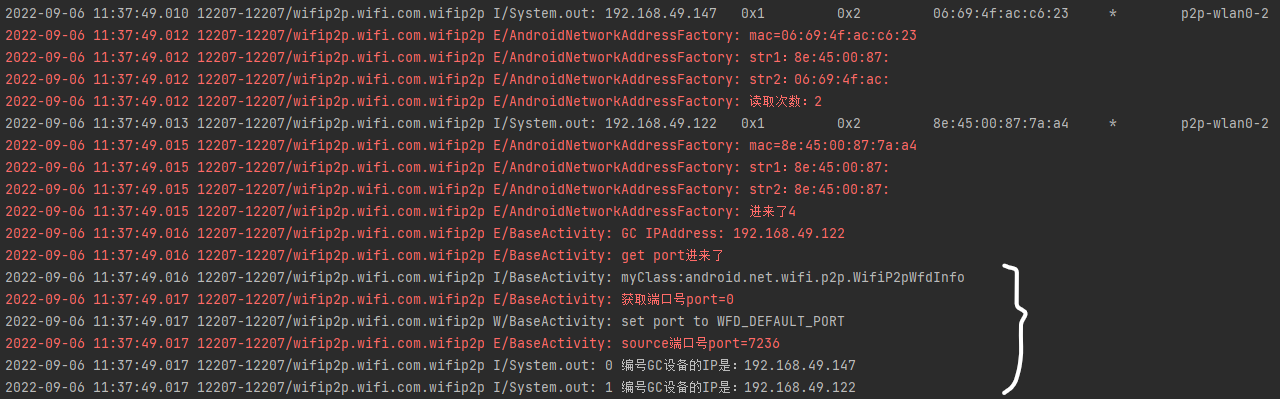
1. 成功调用setWFDInfo：在GC设备端的WifiDisplay功能中成功搜索到该设备；



1. 当某台GC设备连接到GO时，GO设备可以成功获取到该GC设备的IP及端口号；



1. 当多台GC设备连接到GO设备时，GO设备能够知道组内所有成员的IP，并将各个GC设备的IP存放在列表list中。

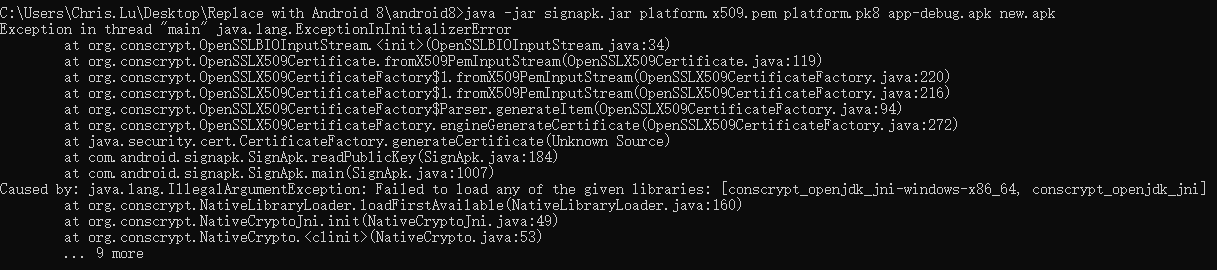


在手机投屏列表中可以成功发现上述设备：



### 问题思考：

Q**------**刚开始尝试使用signapk.jar生成新的apk方法时，当执行java -jar signapk.jar platform.x509.pem platform.pk8 app-debug.apk new.apk命令时，会报错（暂未深入研究该方法）。



# 六、参考

## （1）开发板源码网址

[Files · nanopc-t4-nougat · FriendlyELEC / rk3399-nougat · GitLab](https://gitlab.com/friendlyelec/rk3399-nougat/-/tree/nanopc-t4-nougat)

[Files · nanopc-t4-nougat · FriendlyELEC / rk3399-nougat · GitLab](https://gitlab.com/friendlyelec/rk3399-nougat/-/tree/nanopc-t4-nougat)

## （2）开发板使用手册

[NanoPC-T4/zh - FriendlyELEC WiKi](https://wiki.friendlyelec.com/wiki/index.php/NanoPC-T4/zh#Windows.E4.B8.8B.E9.80.9A.E8.BF.87Type-C.E6.95.B0.E6.8D.AE.E7.BA.BF.E8.BF.9B.E8.A1.8C.E7.83.A7.E5.86.99)

## （3）Miracast技术解析

[(60条消息) Miracast技术详解（一）：Wi-Fi Display\_放大的EZ的博客-CSDN博客](https://blog.csdn.net/qq_27061049/article/details/125546677)

## （4）指针空间源码分析

[Android 4.2 Wifi Display 之 Settings 源码分析 - 指针空间 - 博客园 (cnblogs.com)](https://www.cnblogs.com/seven-sky/p/4875711.html)

## （5）Wifi P2P --- 无网络传输文件

<https://github.com/yoonerloop/WifiP2P.git>

<http://blog.csdn.net/yoonerloop/article/details/79454389>

## （6）Android studio对APK系统签名教程

[(62条消息) Android studio对apk系统签名教程\_laitj的博客-CSDN博客\_android studio 对apk签名](https://blog.csdn.net/qq_40246287/article/details/90511816)

# 七、相关功能文件路径

**权限申请**

app—src—main—res—androidmanifest.xml

app—src—main—java—mainActivity—androidmanifest.xml

**初始化wifip2pManager**

app—src—main—java—activity—BaseActivity

**接收关键广播**

app—src—main—java—Wifip2pReceiver

**发现对等设备**

app—src—main—java—activity—SendFileActivity

**设置wifiP2pWfdInfo类**

android—net—wifi—p2p—WifiP2pWfdInfo

Q: setWFDInfo()方法，采用反射方式调用 ? app—src—main—java—WifiP2pManager--1319

Q: 上述方法在哪个文件下进行的？app—src—main—java—activity—BaseActivity

**初始化WifiP2pwfdInfo**

app—src—main—java—WifiDisplayController--286

**createGroup()**

app—src—main—java—activity—ReceiveFileActivity--96

**获取Group Owner的ip**

app—src—main—java—activity—WiFiServiceDiscoveryActivity--310

**获取Source端RTSP端口号**

app—src—main—java—WifiDisplayController--435

**反射方式获取wifiP2pwfdInfo,通过getControPort()方法获取source端的RTSP端口号**