

2022年1月3日11:33:50

1 驱动移植

- 1.1 参考资料
- 1.2 环境介绍
- 1.3 驱动移植
- 1.4 内核配置
- 1.5 简单测试

2 命令编译

- 2.1 参考资料
- 2.2 环境介绍
- 2.3 简要说明
 - 2.3.1 iw
 - 2.3.2 wpa_cli
 - 2.3.3 wpa_passphrase
 - 2.3.4 wpa_supplicant
 - 2.3.5 hostapd
 - 2.3.6 hostapd_cli
 - 2.3.7 依赖库
- 2.4 交叉编译
 - 2.4.1 openssl
 - 2.4.2 libnl
 - 2.4.3 iw
 - 2.4.4 wpa_supplicant
 - 2.4.5 hostapd
- 2.5 测试验证
 - 2.5.1 命令替换
 - 2.5.2 连接 WiFi

3 连接路由

- 3.1 参考资料
- 3.2 环境介绍
- 3.3 连接路由
 - 3.3.1 固定 IP
 - 3.3.2 动态 DHCP
 - 3.3.3 脚本文件
- 3.4 资源记录
- 3.5 小结

4 开启热点

- 4.1 参考资料
- 4.2 环境介绍
- 4.3 配置过程
- 4.4 注意事项

5 驱动移植思考

- 5.1 简介
- 5.2 整体流程
 - 5.2.1 基础移植
 - 5.2.2 命令交叉编译
 - 5.2.3 业务开发
- 5.3 问题汇总
- 5.4 小结

6 HTTP 服务器

- 6.1 应用场景
- 6.2 方案论证

- 6.2.1 Lighttpd
 - 6.2.2 cpp-httpplib
 - 6.2.3 httpserver
- 6.3 实现方式
- 6.4 注意事项
- 6.5 小结
- 7 联网优先级
 - 7.1 需求场景
 - 7.2 方案分析
 - 7.3 场景举例
 - 7.4 方案实现
 - 7.4.1 方案原理
 - 7.4.2 基础命令
 - 7.4.3 实际验证
 - 7.5 问题说明
 - 7.5.1 天线问题
 - 7.5.2 局域网连接
 - 7.6 小结
- 8 总结

之前在微信公众号【编码小二】总结的系列文章，特地整理成 PDF 文档，方便阅读、查找，也更有条理性。

原文链接：https://mp.weixin.qq.com/mp/appmsgalbum?__biz=Mzk0MDE3NzU5NA==&action=get_album&album_id=1801100510683299844#wechat_redirect

1 驱动移植

1.1 参考资料

- 1、芯片数据手册：TL8188FCA.pdf
- 2、驱动移植手册：Quick_Start_Guide_for_Driver_Compilation_and_Installation.pdf
- 3、【正点原子】I.MX6U嵌入式Linux驱动开发指南V1.3.pdf
- 4、Linux 添加 WiFi 驱动：<https://blog.csdn.net/sbddd/bfm/article/details/101222266>
- 5、还有一些其他前辈的教程，十分感谢！

1.2 环境介绍

- 1、使用的新唐的 NUC980 系列 MCU ；
- 2、Linux 内核版本：4.4.179 ；
- 3、交叉编译链版本

```
1 arm-none-linux-gnueabi-gcc
2
3 gcc version 4.8.3 20140320 (prerelease) (Sourcery CodeBench Lite 2014.05-29)
```

- 4、供应商提供的 WiFi 模块驱动源码；

1.3 驱动移植

1、将 RTL8188FU 驱动添加到 Linux 内核中;

- 在 `drivers/net/wireless` 路径新建文件夹 `realtek_new` ;
- 将驱动源码
`RTL8188FU_Linux_v5.7.4_33085.20190626\driver\rtl8188FU_linux_v5.7.4_33085.20190419` 解压后, 放到 `realtek_new` 文件夹中;
- 修改文件夹名字为 `rtl8188fu` ;

2、修改 `drivers/net/wireless/Kconfig`

- 打开 `drivers/net/wireless/Kconfig` , 在文件最后一行添加

```
1 source "drivers/net/wireless/realtek_new/Kconfig"
```

3、修改 `drivers/net/wireless/Makefile`

- 打开 `drivers/net/wireless/Makefile` , 在文件最后一行添加

```
1 obj-y += realtek_new/
```

4、新增文件 `drivers/net/wireless/realtek_new/Makefile`

- 新增文件 `Makefile` , 添加以下内容, 保存

```
1 obj-$(CONFIG_RTL8188FU) += rtl8188fu/
```

5、新增文件 `drivers/net/wireless/realtek_new/Kconfig`

- 新增文件 `Kconfig` , 添加以下内容, 保存

```
1 menuconfig REALTEK_WIFI
2     tristate "Realtek wifi"
3
4     if REALTEK_WIFI
5
6     choice
7         prompt "select wifi type"
8         default RTL8189FU
9
10    config RTL8188FU
11        depends on REALTEK_WIFI
12        tristate "Realtek 8188FU USB WiFi"
13    endchoice
14    endif
```

6、关闭 Debug 功能

- 打开文件 `drivers/net/wireless/realtek_new/rtl8188fu/Makefile` ;
- 第 88 行, 将宏定义 `CONFIG_RTW_DEBUG=y` 改为 `CONFIG_RTW_DEBUG=n` ;
- 否则会打印很多调试信息, 影响正常使用;
- 修改之后如下图所示:

```
1 87 ##### Debug #####
2 88 CONFIG_RTW_DEBUG = n
3 89 # default log level is _DRV_INFO_ = 4,
4 90 # please refer to "How_to_set_driver_debug_log_level.doc" to set the
   available level.
5 91 CONFIG_RTW_LOG_LEVEL = 4
6 92 ##### Wake On Lan #####
```

1.4 内核配置

1、配置 USB 支持设备

```
1 Device Drivers --->
2   [*] USB support --->
3       <*>   Support for Host-side USB
4       <*>   EHCI HCD (USB 2.0) support
5       <*>   OHCI HCD (USB 1.1) support
6       <*>   ChipIdea Highspeed Dual Role Controller
7       [*]   ChipIdea host controller
8       [*]   ChipIdea driver debug
```

2、配置支持 WiFi 设备

```
1 Device Drivers --->
2   [*] Network device support --->
3       [*] wireless LAN --->
4           <*>   IEEE 802.11 for Host AP (Prism2/2.5/3 and WEP/TKIP/CCMP)
5
6           [*]   support downloading firmware images with Host AP driver
7
6           [*]   support for non-volatile firmware download
```

```

--- Wireless LAN
< >  USB ZD1201 based Wireless device support
< >  Wireless RNDIS USB support
< >  Atheros Wireless Cards  ----
< >  Broadcom IEEE802.11n embedded FullMAC WLAN driver
<*>  IEEE 802.11 for Host AP (Prism2/2.5/3 and WEP/TKIP/CCMP)
[*]   Support downloading firmware images with Host AP driver
[*]   Support for non-volatile firmware download
< >  Marvell 8xxx Libertas WLAN driver support
[ ]   Mediatek Wireless LAN support  ----
[ ]   TI Wireless LAN support  ----
< >  Marvell WiFi-Ex Driver
< >  Realtek wifi  ----

```

3、配置支持 IEEE 802.11

```

1  [*] Networking support  --->
2      *- wireless  --->
3          [*]      cfg80211 wireless extensions compatibility
4          <*>      Generic IEEE 802.11 Networking Stack (mac80211)

```

```

--- Wireless
<*>  Nuvoton external WiFi driver support
<*>  cfg80211 - wireless configuration API
[ ]   nl80211 testmode command
[ ]   enable developer warnings
[ ]   cfg80211 regulatory debugging
[*]   enable powersave by default
[ ]   cfg80211 DebugFS entries
[*]   cfg80211 wireless extensions compatibility
[ ]   lib80211 debugging messages
<*>  Generic IEEE 802.11 Networking Stack (mac80211)
Default rate control algorithm (Minstrel) --->
[ ]   Enable mac80211 mesh networking (pre-802.11s) support (NEW)
[ ]   Export mac80211 internals in DebugFS (NEW)
[ ]   Trace all mac80211 debug messages (NEW)
[ ]   Select mac80211 debugging features (NEW) ----

```

<Select> <Exit> <Help> <Save> <Load>

4、配置 RTL8188FU 驱动

- 由于 RTL8188FU 无需加载额外的固件文件，故无需将驱动配置成模块；
- 直接将驱动编译进内核，这样开机就会自动初始化 RTL8188FU，不需要启动后再手工加载驱动模块；

```

1 Device Drivers --->
2   [*] Network device support --->
3     [*] wireless LAN --->
4       <*> Realtek wifi --->
5         --- Realtek wifi
6           <*> select wifi type (Realtek 8188FU USB WiFi) --->

```

5、在内核源码文件夹根路径，运行命令：

```
1 make uImage -j16
```

6、编译完成；

1.5 简单测试

1、将 WiFi 模块接入控制板；

- 2、使用新编译的内核，重新启动程序；
- 3、使用命令 `ifconfig -a` 测试；
- 4、如果可以看到 wlan0 网卡，则代表驱动可以正常工作；否则，请检查之前配置步骤是否有误；
- 5、我的测试结果如下：

```
1 [root]#ifconfig -a
2 dummy0    Link encap:Ethernet  HWaddr 12:DE:F0:8F:D4:1B
3           BROADCAST NOARP  MTU:1500  Metric:1
4           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
5           TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
6           collisions:0 txqueuelen:1000
7           RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
8
9 eth0      Link encap:Ethernet  HWaddr 40:00:02:B3:D2:34
10          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
11          RX packets:336 errors:0 dropped:138 overruns:0 frame:0
12          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
13          collisions:0 txqueuelen:1000
14          RX bytes:95476 (93.2 KiB)  TX bytes:0 (0.0 B)
15
16 lo       Link encap:Local Loopback
17          inet addr:127.0.0.1  Mask:255.0.0.0
18          UP LOOPBACK RUNNING  MTU:65536  Metric:1
19          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
20          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
21          collisions:0 txqueuelen:1
22          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
23
24 wlan0    Link encap:Ethernet  HWaddr 68:B9:D3:B2:90:C8
25          BROADCAST MULTICAST  MTU:1500  Metric:1
26          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
27          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
28          collisions:0 txqueuelen:1000
29          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
30
```

2 命令编译

2.1 参考资料

- 1、编译过程，部分参考资料如下：

- [【推荐】Linux 添加 WiFi 驱动 \(https://blog.csdn.net/sbddbfm/article/details/101222266\)](https://blog.csdn.net/sbddbfm/article/details/101222266)
- [linux下wifi工具iw的交叉编译 \(https://blog.csdn.net/weixin_42432281/article/details/85786866\)](https://blog.csdn.net/weixin_42432281/article/details/85786866)
- [rtl8188 驱动移植 wifi工具移植 \(https://blog.csdn.net/wmdshhzsmile/article/details/102792811\)](https://blog.csdn.net/wmdshhzsmile/article/details/102792811)
- [rtl8188etv 无线网卡驱动移植 \(https://blog.csdn.net/imlsq/article/details/72844532\)](https://blog.csdn.net/imlsq/article/details/72844532)

- 2、解决问题，部分参考资料如下：

- [【问题1】 line 1: syntax error: unexpected word \(expecting "_"\)](https://blog.csdn.net/qg_26093511/article/details/78932765)
- [【问题2】 wifi模块配置ap模式下出现的问题](https://blog.csdn.net/qg_41877422/article/details/103194360)

3、还有一些其他前辈的教程，十分感谢！

2.2 环境介绍

1、Ubuntu18.04，**不加密环境**，版本信息如下：

```
1 zhaoc@Ubuntu1804:~$ lsb_release -a
2 No LSB modules are available.
3 Distributor ID: Ubuntu
4 Description: Ubuntu 18.04.5 LTS
5 Release: 18.04
6 Codename: bionic
7
8
9 zhaoc@Ubuntu1804:~$ uname -a
10 Linux Ubuntu1804 5.4.0-54-generic #60~18.04.1-Ubuntu SMP Fri Nov 6 17:25:16
    UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
```

2、Linux 内核版本：4.4.179；

3、交叉编译链版本

```
1 arm-none-linux-gnueabi-gcc
2
3 gcc version 4.8.3 20140320 (prerelease) (Sourcery CodeBench Lite 2014.05-29)
```

2.3 简要说明

1、简要记录驱动移植过程中，交叉编译的命令、依赖库。

2、简要说明作用及版本等信息。

2.3.1 iw

1、使用的命令版本：**4.9**

目前发现 5.0 及以上版本用 arm-linux-gcc 编译都会报错，疑似交叉编译工具的 gcc 版本太低；

2、**iw** 命令是 **iwconfig** 命令的替代者，是 Linux 系统上的一款无线配置工具；

(1) 参考资料：https://blog.csdn.net/qg_26602023/article/details/106115823

3、文件系统路径：**/sbin**

4、教程按照 **v0.8.x_rtw_r24647.20171025** 版本进行编译；

2.3.2 wpa_cli

- 1、使用的命令版本：**wpa_cli v0.8.x_rtw_r24647.20171025**
- 2、`wpa_cli` 用来查看设备当前连接 WiFi 状态；
- 3、文件系统路径：`/usr/sbin`

2.3.3 wpa_passphrase

- 1、使用的命令版本：未知；（应该与 `wpa_cli` 一致，即 `v0.8.x_rtw_r24647.20171025`，使用的同一个源码包）
- 2、暂时未用到此命令；
- 3、文件系统路径：`/usr/sbin`

2.3.4 wpa_supplicant

- 1、使用的命令版本：**wpa_supplicant v2.9**
- 2、命令作用：
 - （1）用于连接 `WPA/WPA2` 加密方式的 WiFi ；
 - （2）非加密方式的 WiFi ，也可使用此命令连接；
- 3、文件系统路径：`/usr/sbin`

2.3.5 hostapd

- 1、使用的命令版本：**hostapd v2.9**
- 2、命令作用：
 - （1）配置 WiFi 开启 AP 模式；
 - （2）参考帮助文档，描述如下：

```
1 | User space daemon for IEEE 802.11 AP management,  
2 | IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator
```

- 3、文件系统路径：`/bin`

2.3.6 hostapd_cli

- 1、使用的命令版本：**hostapd_cli v2.9**
- 2、暂未使用；
- 3、文件系统路径：`/bin`

2.3.7 依赖库

1、libnl-3.so.200

- (1) 使用的库版本: **libnl-3.5.0.tar.gz**
- (2) 作用: WiFi 相关命令依赖库;
- (3) 文件系统路径: `/lib`

2、libnl-genl-3.so.200

- (1) 使用的库版本: **libnl-3.5.0.tar.gz**
- (2) 作用: WiFi 相关命令依赖库;
- (3) 文件系统路径: `/lib`

3、libssl.so.1.1

- (1) 使用的库版本: **openssl-1.1.1h.tar.gz**
- (2) 作用: WiFi 相关命令依赖库;
- (3) 文件系统路径: `/lib`

4、libcrypto.so.1.1

- (1) 使用的库版本: **openssl-1.1.1h.tar.gz**
- (2) 作用: WiFi 相关命令依赖库;
- (3) 文件系统路径: `/usr/lib`

5、补充说明

- (1) 需要在实际使用命令时, 查看缺少的依赖库;
- (2) 上述 4 个依赖库, 是我自己在测试时, 发现缺少的;
- (3) 测试方法: 直接运行命令, 查看错误提示, 根据错误提示添加相应的库文件;
- (4) 示例代码如下:

```
1 [root]#hostapd
2 hostapd: error while loading shared libraries: libssl.so.1.1: cannot open
  shared object file: No such file or directory
3 [root]#
```

2.4 交叉编译

进行依赖库、命令的交叉编译。

2.4.1 openssl

1、解压并进入对应路径

```
1 # 解压文件
2 tar -zxvf openssl-1.1.1h.tar.gz
3
4 # 进入文件路径
5 cd openssl-1.1.1h/
```

2、配置环境变量文件

(1) 创建并编辑 env_nuc980 文件

```
1 vim env_nuc980
```

(2) 填入如下内容

```
1 export MACHINE=armv4
2 export RELEASE=4.4.179
3 export SYSTEM=linux2
4 export ARCH=arm
5 export CROSS_COMPILE="arm-none-linux-gnueabi-" # 指定交叉编译链
6 export HOSTCC=gcc # 指定gcc
```

(3) 保存退出

(4) 使用 source 命令使环境变量生效

```
1 source env_nuc980
```

3、修改相关编译配置选项

```
1 ./config no-asm no-async shared --prefix=/home/zhaoc/11-Soft/13-NUC980/12-
  armCompileLib_2/11-install/openssl-1.1.1h_nuc980
```

(1) 相关配置项说明

- **no-asm**: 在交叉编译过程中不使用汇编代码加速编译过程;
- **shared**: 生成动态连接库。
- **no-async**: 交叉编译工具链没有提供 GNU C 的 ucontext 库
- **--prefix=**: 安装路径

(2) 安装路径需要修改为自己的安装路径;

4、make 编译; -j6 : 使用 6 个逻辑内核同时编译, 速度更快;

```
1 make -j6
```

5、安装编译后的文件, 到对应路径

```
1 make install
```

2.4.2 libnl

1、解压并进入对应路径

```
1 # 解压文件
2 tar -zxvf libnl-3.5.0.tar.gz
3
4 # 进入对应路径
5 cd libnl-3.5.0/
```

2、Ubuntu 需安装必备工具，否则会报错

```
1 sudo aptitude install bison
2 sudo aptitude install flex
```

(1) 没有安装对应软件时，报错内容如下

```
1 checking for dlfcn.h... (cached) yes
2 checking for pthread_mutex_lock in -lpthread... yes
3 checking for strerror_l... no
4 configure: WARNING: bison not found. Please install before continuing.
5 configure: WARNING: flex not found. Please install before continuing.
6 configure: error: Required packages are missing. Please install them and
  rerun ./configure
7 root@qddytt:/opt/IPC3516EV200/libnl-3.5.0#
```

3、修改相关编译配置选项；注意修改为自己的安装路径

```
1 ./configure --host=arm-none-linux-gnueabi --prefix=/home/zhaoc/11-Soft/13-
  NUC980/12-armCompileLib_2/11-install/libnl-3.5.0_nuc980
```

(1) 相关配置项说明

- **--host**: 指定交叉编译链
- **--prefix=**: 安装路径

(2) 安装路径需要修改为自己的安装路径；

4、make 编译；-j6：使用 6 个逻辑内核同时编译，速度更快；

```
1 make -j6
```

5、安装编译后的文件，到对应路径

```
1 make install
```

2.4.3 iw

1、解压并进入对应路径

```
1 # 解压文件
2 tar -zxvf iw-4.9.tar.gz
3
4 # 进入对应路径
5 cd iw-4.9/
```

2、配置环境变量；对应 libnl 的安装路径下的 lib/

```
1 export PKG_CONFIG_PATH=/home/zhaoc/11-Soft/13-NUC980/12-armCompileLib_2/11-
install/libnl-3.5.0_nuc980/lib/pkgconfig:$PKG_CONFIG_PATH
```

3、使用指定交叉编译链编译

```
1 make CC=arm-none-linux-gnueabi-gcc
```

4、编译完成后，可以使用 file 命令，查看下 iw 文件的属性

(1) 本次使用 arm-none-linux-gnueabi-gcc 的属性如下

```
1 zhaoc@Ubuntu1804:~/11-Soft/13-NUC980/12-armCompileLib_2/12-sourcecode/iw-4.9$
file iw
2 iw: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically
linked, interpreter /li
```

(2) 之前使用 arm-linux-gcc 时（错误版本），类似文件（wpa_supplicant）属性如下

```
1 wpa_supplicant: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 3.2.0,
BuildID[sha1]=abd215c0dc5a490e088db5fc57f26b1efa8b670e, not stripped
```

(3) 命令属性，必须为 ARM 类型的才可以，否则 不能在板子上正常运行。

2.4.4 wpa_supplicant

1、解压并进入对应路径

```
1 # 解压文件
2 tar -zxvf wpa_supplicant_hostapd-0.8_rtw_r24647.20171025.tar.gz
3
4 # 进入对应路径
5 cd wpa_supplicant_hostapd-0.8_rtw_r24647.20171025/wpa_supplicant/
```

2、编辑 Makefile 文件，修改使用的交叉编译链（两处修改），保存退出；

```

1  ifndef CC
2  CC=arm-none-linux-gnueabi-gcc
3  endif
4
5  ifndef CFLAGS
6  CFLAGS = -MMD -O2 -Wall -g
7  endif
8
9  CC=arm-none-linux-gnueabi-gcc
10
11 export LIBDIR ?= /usr/local/lib/

```

make 编译; -j6 : 使用 6 个逻辑内核同时编译, 速度更快;

```
1  make -j6
```

2.4.5 hostapd

1、解压并进入对应路径

```

1  # 解压文件
2  tar -zxvf hostapd-2.9.tar.gz
3
4  # 进入对应路径
5  cd hostapd-2.9/hostapd/

```

2、使用默认配置文件, 创建 config 文件

```
1  cp defconfig .config
```

3、使用 vim 修改 .config , (取消注释)

```
1  #CONFIG_IEEE80211N=y
```

改为

```
1  CONFIG_IEEE80211N=y
```

否则后续会报错:

```

1  ~ # hostapd ./hostapd.conf
2  Configuration file: ./hostapd.conf
3  Line 7: unknown configuration item 'ieee80211n'
4  1 errors found in configuration file './hostapd.conf'
5  Failed to set up interface with ./hostapd.conf
6  Failed to initialize interface

```

显示行号路径如下:

```
1 156
2 157 # IEEE 802.11n (High Throughput) support
3 158 CONFIG_IEEE80211N=y
4 159
```

4、备份 Makefile 文件，免得误操作之后还要重新解压一份文件；

```
1 cp Makefile Makefile.bak
```

5、编辑 Makefile 文件，新增如下内容，注意是 **新增内容**；

```
1 CC=arm-none-linux-gnueabi-gcc
2
3 CFLAGS += -I /home/zhaoc/11-Soft/13-NUC980/14-armCompileLib_20201201/11-
install/libn1-3.5.0_nuc980/include
4 LIBS += -L /home/zhaoc/11-Soft/13-NUC980/14-armCompileLib_20201201/11-
install/libn1-3.5.0_nuc980/lib
5
6 CFLAGS += -I /home/zhaoc/11-Soft/13-NUC980/14-armCompileLib_20201201/11-
install/openssl-1.1.1h_nuc980/include
7 LIBS += -L /home/zhaoc/11-Soft/13-NUC980/14-armCompileLib_20201201/11-
install/openssl-1.1.1h_nuc980/lib -lcrypto -lssl
8
9 LDFLAGS += -lpthread
10 LDFLAGS += -lm
```

6、添加环境变量，如下所示；

```
1 export PKG_CONFIG_PATH=/home/zhaoc/11-Soft/13-NUC980/14-
armCompileLib_20201201/11-install/libn1-
3.5.0_nuc980/lib/pkgconfig:$PKG_CONFIG_PATH
```

7、make 编译；-j6：使用 6 个逻辑内核同时编译，速度更快；

```
1 make -j6
```

8、编译完成后，就会在当前目录生成所需的 `hostapd`，`hostapd_cli` 文件，不需要 `make install` 即可；

9、使用 `file` 命令，查看编译后的文件属性，如果属性为 ARM，即代表可以在开发板上运行；

```
1 zhaoc@ubuntu1804:~/11-Soft/13-NUC980/13-armCompileLib_20201130/12-
sourcecode/hostapd-2.9/hostapd$ file hostapd
2 hostapd: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically
linked, interpreter /lib/ld-linux.so.3, for GNU/Linux 2.6.16, with
debug_info, not stripped
```

至此，所有使用到的命令均已编译完毕。

2.5 测试验证

分两步进行，第一步将命令放到文件系统中；第二步测试连接 WiFi 是否正常。

2.5.1 命令替换

1、替换 `wpa_cli wpa_passphrase wpa_supplicant`

- 进入对应目录

```
1 # 进入对应目录
2 cd /usr/sbin/
```

- 查看当前目录文件

```
1 # 查看当前目录文件
2 ls -l
```

- 删除文件

```
1 # 删除文件
2 rm wpa_cli wpa_passphrase wpa_supplicant
```

- 拷贝交叉编译好的命令到当前目录（其他方式均可）

```
1 # 拷贝交叉编译好的命令到当前目录
2 lrz
```

- 赋予可执行权限

```
1 # 赋予可执行权限
2 chmod 777 wpa_cli wpa_passphrase wpa_supplicant
```

- 直接输入命令，测试是否缺少依赖库

```
1 # wpa_supplicant 测试
2 wpa_supplicant
3
4 # wpa_passphrase 测试
5 wpa_passphrase
6
7 # wpa_cli 测试
8 wpa_cli
```

2、替换 `iw`

- 进入对应目录


```
1 # 进入对应目录
2 cd /sbin/
```

- 拷贝 iw 到此目录（其他方式均可）

```
1 # 拷贝 iw 到此目录
2 lrz
```

- 可以看到拷贝进来的文件没有执行权限，修改文件权限

```
1 chmod 777 iw
```

- 直接输入命令，测试是否缺少依赖库

```
1 # 输入命令
2 iw
```

- 我的开发板提示缺少库文件：libnl-genl-3.so.200；
- 则去编译好的 libnl 安装路径中拷贝对应库文件进来即可；
- 相对路径：11-install\libnl-3.5.0_nuc980\lib
- 修改新拷贝的库文件权限

```
1 chmod 755 libnl-genl-3.so.200
```

- 接着再次从命令行输入 iw 测试命令，提示缺少库文件：libnl-3.so.200；
- 参考上述方法，拷贝文件到对应路径，同样修改库文件权限为 755；
- 再次使用 iw 测试命令，发现没有异常提示，可以正常打印相关配置信息；
- 此时代表命令已可以正常运行；

2.5.2 连接 WiFi

1、新增连接 WiFi 的配置文件

(1) 创建新文件

```
1 # 开发板没有 vim
2 vi r8000.conf
```

(2) 填入如下内容

```
1 ctrl_interface=/var/run/wpa_supplicant
2 ap_scan=1
3 network={
4     ssid="WIFI名称"
5     psk="WIFI密码明文"
6 }
```

2、配置连接 WiFi

```
1 # 查看网卡连接状态
2 wpa_cli -iwlan0 status
3
4 # 使用配置文件连接 WiFi
5 wpa_supplicant -iwlan0 -c ./r8000.conf -B
6
7 # 开启网卡（可选）
8 ifconfig wlan0 up
9
10 # 再次查看网卡连接状态
11 wpa_cli -iwlan0 status
12
13 # 设置固定 IP 地址，子网掩码
14 ifconfig wlan0 192.168.60.1 netmask 255.255.255.0
15
16 # 设置默认网关
17 route add default gw 192.168.60.254
18
19 # 修改 DNS ，根据参考资料进行配置；
20 echo nameserver 223.5.5.5 > /etc/resolv.conf
```

3、使用 ping 测试 WiFi ，验证是否正常；

```
1 ping baidu.com
```

4、如果可以 ping 通，则表示命令可以正常运行。

3 连接路由

3.1 参考资料

1、配置过程，部分参考资料如下：

- [通讯之一—ARM Linux下以太网的访问外网方法](https://blog.csdn.net/qq_27977257/article/details/53130151)
(https://blog.csdn.net/qq_27977257/article/details/53130151)
- [开发板linux连接wifi的方法（一）](https://blog.csdn.net/qq_29630271/article/details/72751076)
(https://blog.csdn.net/qq_29630271/article/details/72751076)
- [linux下ifconfig, DNS以及route配置](http://www.cnitblog.com/201/archive/2009/08/20/60887.html)
(<http://www.cnitblog.com/201/archive/2009/08/20/60887.html>)
- [Linux环境下使用WIFI模块：使用DHCP工具动态获得IP地址](https://blog.csdn.net/yunlong654/article/details/88680543)
(<https://blog.csdn.net/yunlong654/article/details/88680543>)
- [linux 添加删除- 默认网关 的方法（route add详解）](https://www.xuebuyuan.com/1256289.html)
(<https://www.xuebuyuan.com/1256289.html>)
- [无线网卡（RTL8188EU）驱动编译、使用DHCP配置无线网络](https://blog.csdn.net/jifengzhiling/article/details/80983006)
(1 (<https://blog.csdn.net/jifengzhiling/article/details/80983006>))
- [用wpa_supplicant wpa_cli连不同加密方式的ap](https://blog.csdn.net/weixin_37193849/article/details/53911579) (https://blog.csdn.net/weixin_37193849/article/details/53911579)

2、解决问题，部分参考资料如下：

- [开发板可以ping通百度IP, 但是不能ping百度域名 提示ping: bad address 'www.baidu.com' \(https://blog.csdn.net/mick_ye/article/details/50844539\)](https://blog.csdn.net/mick_ye/article/details/50844539)

3、还有一些其他前辈的教程，十分感谢！

3.2 环境介绍

1、Ubuntu18.04，**不加密环境**，版本信息如下：

```
1 zhaoc@Ubuntu1804:~$ lsb_release -a
2 No LSB modules are available.
3 Distributor ID: Ubuntu
4 Description:    Ubuntu 18.04.5 LTS
5 Release:       18.04
6 Codename:      bionic
7
8
9 zhaoc@Ubuntu1804:~$ uname -a
10 Linux Ubuntu1804 5.4.0-54-generic #60~18.04.1-Ubuntu SMP Fri Nov 6 17:25:16
    UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
```

2、Linux 内核版本：4.4.179；

3、交叉编译链版本

```
1 arm-none-linux-gnueabi-gcc
2
3 gcc version 4.8.3 20140320 (prerelease) (Sourcery CodeBench Lite 2014.05-29)
```

3.3 连接路由

在我实际调试过程中，最开始是通过命令行配置，到代码成熟后，使用脚本文件连接路由。

因此有三种方法：

- 手动连接，固定 IP 方式；
- 手动连接，动态 DHCP 方式；
- 脚本连接，动态 DHCP 方式；

3.3.1 固定 IP

命令配置过程，记录如下。

1、开启无线网卡，wlan0

```
1 ifconfig wlan0 up
```

2、使用配置文件自动连接，r8000.conf 配置文件可参考文章 [WiFi 命令编译过程](#)，或者参考文章末尾资料。

```
1 | wpa_supplicant -iwlan0 -c ./r8000.conf -B
```

3、查看 WiFi 连接状态

```
1 | wpa_cli -iwlan0 status
```

4、设置固定 IP 地址，子网掩码

```
1 | ifconfig wlan0 192.168.60.1 netmask 255.255.255.0
```

5、查看 IP 地址

```
1 | ifconfig
```

6、设置默认网关

```
1 | route add default gw 192.168.60.254
```

备注：删除默认网关

```
1 | route del default
```

7、查看默认网关

```
1 | # 命令
2 | route
3 |
4 | # 测试结果
5 | [root]#route
6 | Kernel IP routing table
7 | Destination      Gateway           Genmask          Flags Metric Ref    Use Iface
8 | default          192.168.60.254   0.0.0.0          UG    0      0      0 wlan0
9 | 192.168.60.0     *                255.255.255.0    U     0      0      0 wlan0
```

8、设置 DNS

```
1 | # 编辑配置文件
2 | vi /etc/resolv.conf
3 |
4 | # 添加 DNS ，可自行尝试
5 | nameserver 8.8.8.8
6 | nameserver 223.5.5.5
7 | nameserver 223.6.6.6
```

命令行修改 DNS 方法

```
1 | # 修改 DNS ，根据参考资料进行配置；
2 | echo nameserver 223.5.5.5 > /etc/resolv.conf
```

3.3.2 动态 DHCP

1、确定能正常检测到无线网卡（可正常看到网卡名）

```
1 | ifconfig -a
```

2、停止有线网卡

```
1 | ifconfig eth0 down
```

3、重启无线网卡

```
1 | ifconfig wlan0 down
2
3 | ifconfig wlan0 up
```

4、设置无线网卡使用配置文件连接 WiFi

```
1 | wpa_supplicant -iwlan0 -c ./r8000.conf -B
```

5、使用无线网卡扫描 WiFi 测试

```
1 | iwlist wlan0 scanning
```

6、配置无线网卡使用 DHCP 功能

```
1 | udhcpc -i wlan0
```

7、查看无线网卡 IP 地址，是否获取成功

```
1 | ifconfig -a
```

8、测试是否能正常联通外网

```
1 | ping baidu.com
```

9、查看当前 WiFi 连接状态

```
1 | wpa_cli -iwlan0 status
```

3.3.3 脚本文件

启动、停止过程，基本与命令行配置类似，只不过是全部封装为脚本，便于操作。

1、开启 WiFi 连接脚本

```
1 | #/bin/sh
```

```

2  # 日志打印
3  echo "===== $0 start ====="
4
5  # 先杀死所有相关进程
6  killall hostapd udhcpd wpa_supplicant udhcpc
7
8  # 关闭无线网卡
9  ifconfig wlan0 down
10
11 # 打开无线网卡
12 ifconfig wlan0 up
13
14 # 延时1秒
15 sleep 1
16
17 # 开启WiFi连接
18 wpa_supplicant -iwlan0 -c /root/App/wifi.conf -B
19
20 # 配置自动获取IP
21 udhcpc -i wlan0
22
23 # 日志打印
24 echo "===== $0 stop ====="

```

2、关闭 WiFi 连接，仅杀死跟 WiFi 连接路由，有关的命令

```

1  #/bin/sh
2  echo "[root] killall wpa_supplicant udhcpc"
3  killall wpa_supplicant udhcpc

```

3、WiFi 基本配置文件：/root/App/wifi.conf

```

1  ctrl_interface=/var/run/wpa_supplicant
2  ap_scan=1
3  network={
4      ssid="WiFi名称"
5      psk="WiFi密码"
6  }

```

3.4 资源记录

1、r8000.conf 配置文件

(1) 注意事项：

- 字符严格缩进；
- 不能多/少字符；
- 此文件，对格式要求很严，如有问题，请先检查是否此文件有误；
- 一定要手打，不要复制（赋值的话格式很容易出错）

(2) 配置文件内容如下：

```
1 [root]#cat r8000.conf
2 ctrl_interface=/var/run/wpa_supplicant
3 ap_scan=1
4 network={
5     ssid="WiFi名称"
6     psk="WiFi密码"
7 }
```

3.5 小结

- 1、刚开始调试时，由于不知道参数是否合适，就是用命令行，一句一句敲出来；直到代码基本确定了，才编写脚本。
- 2、现在想想，真是有点傻啊，为什么一开始不直接使用脚本？有问题，直接修改脚本里边的配置项，不就行了？
- 3、WiFi 连接路由，是最常使用的一个功能，期间遇到两个问题
 - (1) WiFi 连接路由后，网速很慢。经过测试，有几方面的原因：
 - 没有安装 WiFi 天线；
 - WiFi 天线不匹配；
 - (2) WiFi 连接路由后，不稳定。具体表现：ping 百度，延时忽高忽低。暂未发现具体原因。思考方向：
 - WiFi 模块自身问题（性能差、工艺问题等）
 - WiFi 驱动问题（版本低？不适配？）

4 开启热点

4.1 参考资料

按照惯例，先呈上参考资料，这样可以先打开，再结合文章一块食用，体验更佳！

首先强烈推荐 **周立功** 网站提供的两个参考教程，详细介绍了 WiFi 模组使用方法。

其他地方也很难找到这么详细的教程

- **【WiFi-BL使用说明】** https://manual.zlg.cn/web/#/30?page_id=1157
- **【蓝牙使用】** https://manual.zlg.cn/web/#/45?page_id=4819

然后是我自己在网上找到的参考资料

- **【linux编译wifi驱动RTL8188EUS模块 AP模式】**
<https://blog.csdn.net/zh1204190329/article/details/79942137>

最后，还有一些其他前辈的教程，十分感谢！

4.2 环境介绍

1、Ubuntu18.04，**不加密环境**，版本信息如下：

```
zhaoc@Ubuntu1804:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.5 LTS
Release:       18.04
Codename:      bionic

zhaoc@Ubuntu1804:~$ uname -a
Linux Ubuntu1804 5.4.0-54-generic #60~18.04.1-Ubuntu SMP Fri Nov 6 17:25:16 UTC 2020 x86_64 x86_64 x86_64
GNU/Linux
```

```
1  zhaoc@Ubuntu1804:~$ lsb_release -a
2  No LSB modules are available.
3  Distributor ID: Ubuntu
4  Description:    Ubuntu 18.04.5 LTS
5  Release:       18.04
6  Codename:      bionic
7
8
9  zhaoc@Ubuntu1804:~$ uname -a
10 Linux Ubuntu1804 5.4.0-54-generic #60~18.04.1-Ubuntu SMP Fri Nov 6 17:25:16
    UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
```

2、Linux 内核版本：4.4.179；

3、交叉编译链版本

```
arm-none-linux-gnueabi-gcc

gcc version 4.8.3 20140320 (prerelease) (Sourcery CodeBench Lite 2014.05-29)
```

```
1  arm-none-linux-gnueabi-gcc
2
3  gcc version 4.8.3 20140320 (prerelease) (Sourcery CodeBench Lite 2014.05-29)
```

4.3 配置过程

参考周立功的资料，记录开启 AP 的脚本文件，内容如下


```

#!/bin/sh
killall hostapd udhcpd wpa_supplicant udhcpc 2> /dev/null
#insmod /opt/bcmdhd.ko # bcmdhd.ko驱动模块放在其它目录，则需要对应修改
rmmod 8188fu
insmod ./8188fu.ko
ifconfig wlan0 down
#echo /lib/firmware/bcm/SN8000_BCM43362/fw_bcmdhd_apsta.bin > /sys/module/bcmdhd/parameters/firmware_path
#echo /lib/firmware/bcm/SN8000_BCM43362/bcmdhd.SN8000.SDIO.cal > /sys/module/bcmdhd/parameters/nvram_path
ifconfig wlan0 up
sleep 1
#echo "1" > /proc/sys/net/ipv4/ip_forward
#iptables -t nat -I POSTROUTING -o eth0 -j MASQUERADE #eth0是联网网卡名
#iptables -A FORWARD -s 192.168.5.0/24 -j ACCEPT #wlan0的ip
#iptables -A FORWARD -d 192.168.151.0/24 -j ACCEPT #联网网卡的ip
ifconfig wlan0 192.168.5.1
udhcpd ./udhcpd.conf # udhcpd.conf放在其它目录，则需要对应修改
echo "hostapd -B ./hostapd.conf"
hostapd -B ./hostapd.conf # hostapd.conf放在其它目录，则需要对应修改

```

```

1  #!/bin/sh
2  killall hostapd udhcpd wpa_supplicant udhcpc 2> /dev/null
3  #insmod /opt/bcmdhd.ko # bcmdhd.ko驱动模块放在其它目录，则需要
   对应修改
4  rmmod 8188fu
5  insmod ./8188fu.ko
6  ifconfig wlan0 down
7  #echo /lib/firmware/bcm/SN8000_BCM43362/fw_bcmdhd_apsta.bin >
   /sys/module/bcmdhd/parameters/firmware_path
8  #echo /lib/firmware/bcm/SN8000_BCM43362/bcmdhd.SN8000.SDIO.cal >
   /sys/module/bcmdhd/parameters/nvram_path
9  ifconfig wlan0 up
10 sleep 1
11 #echo "1" > /proc/sys/net/ipv4/ip_forward
12 #iptables -t nat -I POSTROUTING -o eth0 -j MASQUERADE #eth0是联网网卡名
13 #iptables -A FORWARD -s 192.168.5.0/24 -j ACCEPT #wlan0的ip
14 #iptables -A FORWARD -d 192.168.151.0/24 -j ACCEPT #联网网卡的ip
15 ifconfig wlan0 192.168.5.1
16 udhcpd ./udhcpd.conf # udhcpd.conf放在其它目录，则需要对应修改
17 echo "hostapd -B ./hostapd.conf"
18 hostapd -B ./hostapd.conf # hostapd.conf放在其它目录，则需要对应
   修改

```

参考上述脚本示例，就能有一个大概的思路。

结合自身需求，整理命令如下：



查看wlan0网卡是否存在

ifconfig -a

重启无线网卡

ifconfig wlan0 down

ifconfig wlan0 up

配置wlan0网卡IP

ifconfig wlan0 192.168.5.1

使能DHCP，自动分配IP服务

udhcpd /etc/udhcpd.conf

开启WiFi

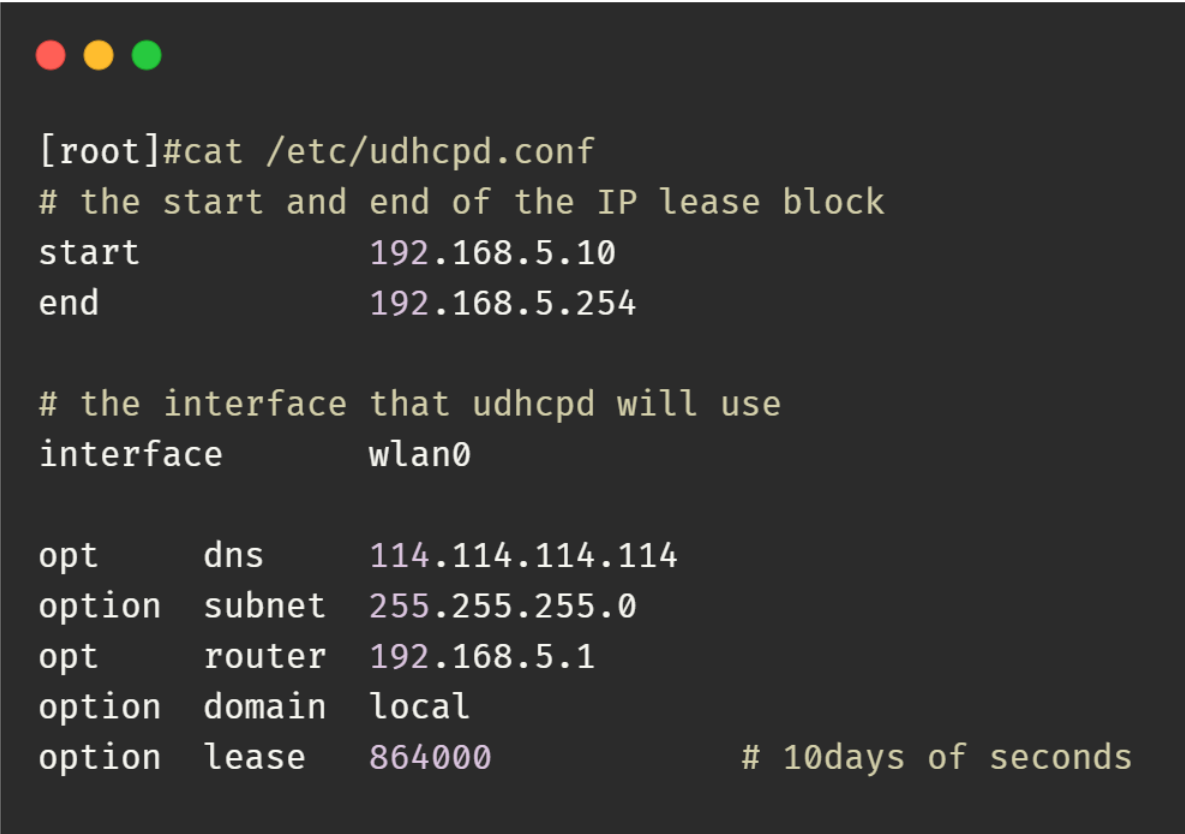
hostapd -B /etc/hostapd.conf

```
1 # 查看wlan0网卡是否存在
2 ifconfig -a
3
4 # 重启无线网卡
5 ifconfig wlan0 down
6 ifconfig wlan0 up
7
8 # 配置wlan0网卡IP
9 ifconfig wlan0 192.168.5.1
10
11 # 使能DHCP，自动分配IP服务
```

```
12 | udhcpd /etc/udhcpd.conf
13 |
14 | # 开启WiFi
15 | hostapd -B /etc/hostapd.conf
```

udhcpd 命令、hostapd 命令，需要使用对应配置文件启动，整理文件内容如下：

/etc/udhcpd.conf 文件内容如下：



```
[root]#cat /etc/udhcpd.conf
# the start and end of the IP lease block
start          192.168.5.10
end            192.168.5.254

# the interface that udhcpd will use
interface      wlan0

opt    dns    114.114.114.114
option subnet 255.255.255.0
opt    router 192.168.5.1
option domain local
option lease  864000          # 10days of seconds
```

```
1 | [root]#cat /etc/udhcpd.conf
2 | # the start and end of the IP lease block
3 | start          192.168.5.10
4 | end            192.168.5.254
5 |
6 | # the interface that udhcpd will use
7 | interface      wlan0
8 |
9 | opt    dns    114.114.114.114
10 | option subnet 255.255.255.0
11 | opt    router 192.168.5.1
12 | option domain local
13 | option lease  864000          # 10days of seconds
```

/etc/hostapd.conf 文件内容如下：



```
[root]#cat /etc/hostapd.conf
# WPA2-PSK authentication with AES encryption
interface=wlan0
driver=nl80211
ctrl_interface=/var/run/hostapd
ssid=ZHAOC
channel=6
ieee80211n=1
hw_mode=g
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=12345678
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
```

```
1 [root]#cat /etc/hostapd.conf
2 # WPA2-PSK authentication with AES encryption
3 interface=wlan0
4 driver=nl80211
5 ctrl_interface=/var/run/hostapd
6 ssid=ZHAOC
7 channel=6
8 ieee80211n=1
9 hw_mode=g
10 ignore_broadcast_ssid=0
11 wpa=2
12 wpa_passphrase=12345678
13 wpa_key_mgmt=WPA-PSK
14 rsn_pairwise=CCMP
```

其中包含 WiFi 名称、密码等信息

- WiFi 名称 (ssid) : ZHAOC
- WiFi 密码 (wpa_passphrase) : 12345678

在熟悉了基本的命令、配置文件后，就可以编写脚本，自动化执行了，这样可以大大节省调试时间。

开启 WiFi AP 模式，脚本文件如下，并附有详细注释

```
#!/bin/sh
# 先杀死所有相关进程
killall hostapd udhcpd wpa_supplicant udhcpd

# 重启无线网卡
ifconfig wlan0 down
ifconfig wlan0 up

# 延时1秒
sleep 1

# 配置无线网卡固定IP
ifconfig wlan0 192.168.43.1

# 启动服务端，DHCP自动分配IP地址
udhcpd /App/udhcpd.conf

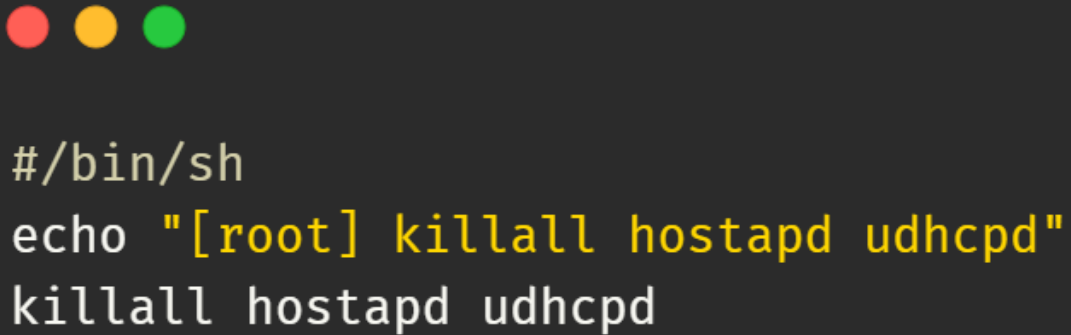
# 开启设备热点
echo "hostapd -B /App/hostapd.conf"
hostapd -B /App/hostapd.conf
```

```
1  #/bin/sh
2  # 先杀死所有相关进程
3  killall hostapd udhcpd wpa_supplicant udhcpd
4
5  # 重启无线网卡
6  ifconfig wlan0 down
7  ifconfig wlan0 up
8
9  # 延时1秒
10 sleep 1
11
12 # 配置无线网卡固定IP
```

```
13 | ifconfig wlan0 192.168.43.1
14 |
15 | # 启动服务端，DHCP自动分配IP地址
16 | udhcpd /App/udhcpd.conf
17 |
18 | # 开启设备热点
19 | echo "hostapd -B /App/hostapd.conf"
20 | hostapd -B /App/hostapd.conf
```

基本与单纯的命令行操作差不多。主要是集合到脚本文件当中，要求格式统一等。

下面看看停止 WiFi AP 模式脚本，其实就是 killall 命令



```
#!/bin/sh
echo "[root] killall hostapd udhcpd"
killall hostapd udhcpd
```

```
1 | #/bin/sh
2 | echo "[root] killall hostapd udhcpd"
3 | killall hostapd udhcpd
```

下面就是自己配置的 App/hostapd.conf，附有详细注释



```
# WPA2-PSK authentication with AES encryption
# 使用网卡
interface=wlan0

# 网卡驱动
driver=nl80211

# 未知
ctrl_interface=/var/run/hostapd

# 热点名称
ssid=ZHAOC

# 热点使用信道，频段
channel=11

# 未知
ieee80211n=1
hw_mode=g
ignore_broadcast_ssid=0

# 下面四项，当配置为加密模式时，需开启；
# 即连接 WiFi 热点时，需要输入密码；
#wpa=2
#wpa_passphrase=12345678
#wpa_key_mgmt=WPA-PSK
#rsn_pairwise=CCMP
```

```
1 # WPA2-PSK authentication with AES encryption
2 # 使用网卡
3 interface=wlan0
4
5 # 网卡驱动
```

```
6 driver=n180211
7
8 # 未知
9 ctrl_interface=/var/run/hostapd
10
11 # 热点名称
12 ssid=ZHAOC
13
14 # 热点使用信道，频段
15 channel=11
16
17 # 未知
18 ieee80211n=1
19 hw_mode=g
20 ignore_broadcast_ssid=0
21
22 # 下面四项，当配置为加密模式时，需开启：
23 # 即连接 WiFi 热点时，需要输入密码：
24 #wpa=2
25 #wpa_passphrase=12345678
26 #wpa_key_mgmt=WPA-PSK
27 #rsn_pairwise=CCMP
```

最后则是更新后的，DHCP 自动分配 IP 配置文件，App/udhcpd.conf

```
# the start and end of the IP lease block
start          192.168.43.10
end            192.168.43.254

# the interface that udhcpd will use
interface      wlan0
opt    dns     114.114.114.114
option subnet  255.255.255.0
opt    router  192.168.43.1
option domain  local
option lease   864000          # 10days of seconds
```



```
1 # the start and end of the IP lease block
2 start      192.168.43.10
3 end        192.168.43.254
4
5 # the interface that udhcpd will use
6 interface  wlan0
7 opt    dns  114.114.114.114
8 option subnet 255.255.255.0
9 opt    router 192.168.43.1
10 option domain local
11 option lease 864000          # 10days of seconds
```

4.4 注意事项

- 1、hostapd 需要使用新版本的，目前用的 2.9 版本的；
- 2、driver 最好使用 nl80211，即如下内容

```
1 driver=nl80211
```

原因：目前看到的所有成功的教程，都是使用 nl80211，最好保持一致。

- 3、为什么原来用 nl80211 不可以？

回答：

原来将 driver 配置为 nl80211，运行命令报错，是因为使用的工具版本不支持导致的，后来换了 2.9 版本的 hostapd 后，就正常了。

- 4、目前 WiFi 开启热点后，使用的信道 11，经过自测，此信道在当前环境下，相对比较稳定；

```
1 channel=11
```

5 驱动移植思考

本节打算进行一次系统性的总结、复盘，分享一下自己的思考过程，遇到的问题等。希望对您有点帮助 😊😊😊😊

备注：由于自己目前仅使用了一款 WiFi，因此不能代表所有的 WiFi 种类，仅以此为代表，谈谈自己的理解与思考。如有问题，欢迎一块讨论交流。可关注微信公众号「**编码小二**」，通过公众号后台，加我微信好友，一起学习！

5.1 简介

WiFi 是什么？能用来做什么？

都 2021 年了，相信您对 WiFi 一定不陌生了。

可以参考这里的百度百科释义：[【百度百科 - WiFi \(https://baike.baidu.com/item/Wi-Fi/151036?fromtitle=WIFI&fromid=803834&fr=aladdin\)】](https://baike.baidu.com/item/Wi-Fi/151036?fromtitle=WIFI&fromid=803834&fr=aladdin)

那 WiFi 能用来干什么呢？

以手机为例，很明显有两个功能：

- 开启 WiFi，连接到路由器；
- 开启热点（AP），让别人连接到由自己设备 WiFi 开出来的路由；

这两个场景，在实际中已经很常见了。

本次 WiFi 驱动移植，应用场景是什么呢？

简单的说，就是我有一块控制板，现在需要增加 WiFi 模块，让设备可以通过 WiFi，正常连接到网络。

应用场景也有很多，比如这些：

- 对安装环境要求较高的地方（尽可能少的接线）；
- 不易连接有线网络的地方；
- 低成本，不想用 4G 的地方；
- 仅用作局域网通信；
- ...

说到这里，还有一个比较重要的场景，就是设备端作为一个 Web 服务器，可以接入其他设备，比如手机 APP 等，进行功能参数的配置。这块考虑后边单独写一篇文章总结。

设备怎么与 WiFi 模块通信？有哪几种方式？

目前主流的方案有两种：USB 和 SDIO；

不过由于 USB 方案，适配相对简单，因此选用 USB 方式的芯片。

生活中常见的 USB 无线网卡，类似的也是这种方式。

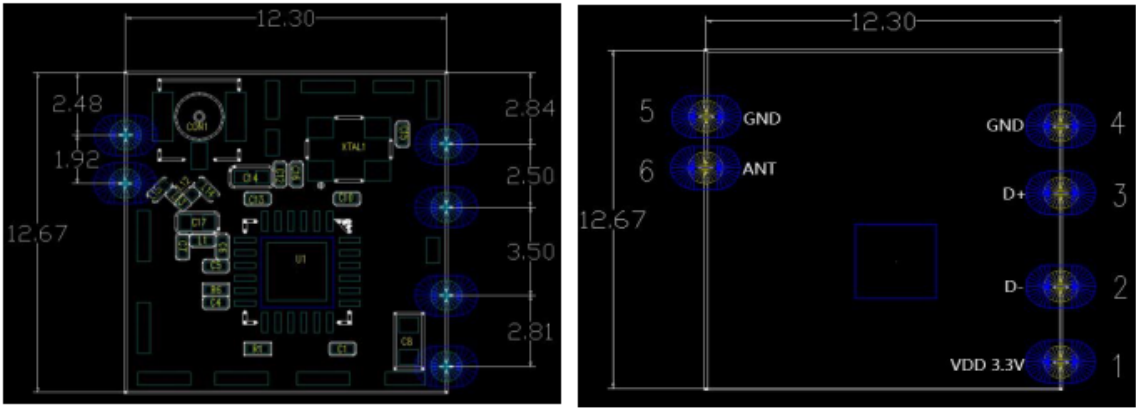
备注：具体使用哪种接口，还要结合实际情况选择。

关于 USB 方式和 SDIO 方式，可以看看这份参考资料，[WIFI的USB和SDIO接口是什么？？
\(https://zhidao.baidu.com/question/117152584.html\)](https://zhidao.baidu.com/question/117152584.html)

硬件怎么连接？是否需要天线？如果不需要天线有什么问题？天线不适配又有什么问题？

使用 USB 方式的 WiFi 模组，硬件接线较为方便，如下图所示

4.5 Mechanical Dimensions



1	VDD 3.3V	4	GND
2	D-	5	GND
3	D+	6	ANT

Product pictures



由上图可以看到，引脚功能如下：

- D+：USB接口+
- D-：USB接口-
- ANT：天线接口

而且整体尺寸很小，图中尺寸单位为 **毫米 (mm)**。

下一个问题，肯定是需要天线的，否则 WiFi 无法发射出来信号。现象就是：AP 模式下，搜索不到路由。

如果天线不适配，则不能达到最大利用率。实际现象：WiFi 网速慢，网络连接不稳定，ping 丢包等。

5.2 整体流程

这一部分，就回顾一下，从拿到 WiFi 模块，到最后产品交付的全流程，然后复盘总结一下。

我把整个过程大致分为三个阶段，图示如下：

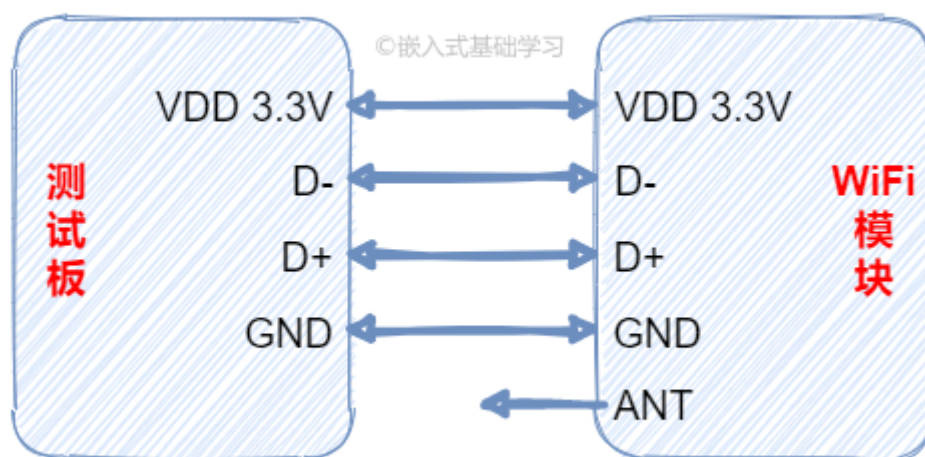


其实在这之前，还应该有一个硬件环境准备阶段。为了尽量详细，简要描述如下：

(如果您已了解，可直接看下一部分☺)



硬件连接图，简要图示如下：



5.2.1 基础移植

此阶段的终极目的：**使系统检测到 WiFi 网卡**。为什么呢？

- USB WiFi 的基本操作，都是通过 **命令行** 实现，因此必须要先检测到 WiFi 网卡，才能针对此网卡，进行特定操作。
- 本质与操作有线网卡 eth0 类似。

这里首先要做的就是 **驱动移植**，将官方的驱动，移植到现有内核中，这样才能在内核菜单中看到对应模块。

接着需要修改内核选项，使能所用的 WiFi 模块。

可以参考文章 [「详细记录 | Realtek RTL8188FU WiFi 驱动移植」](#)

最后肯定是要编译，生成新的内核文件，可以使用挂载内核的方式，验证一下新编译的文件是否正常运行。

如果可以看到 WiFi 网卡驱动，则表示内核编译无误；否则，需要重复上述步骤，进行检查；

PS：一定要确保硬件连线无误，没有断路、短路、接反等问题。

5.2.2 命令交叉编译

经过上个阶段，我们已经可以正常检测到网卡了。

这个阶段，主要是准备一些 **工具**，也就是一些用来操作 WiFi 的命令、依赖的库文件等；

这个阶段可以参考文章：[「详细记录 | Realtek RTL8188FU WiFi 命令编译」](#)

最后记得把编译生成的命令，放到文件系统上的对应路径。

注意命令权限，正常应该是 755，一定要有可执行权限！！

当我们准备好所需命令后，就可以编写一些自动化的脚本，这样在程序中，只需调用对应脚本文件即可。

当然了，一些特殊文件，比如保存 WiFi 名称密码的文件，由于需要经常修改，因此还需要在程序中进行写文件操作。

我的做法是先在文件系统中，给一个默认的初始文件，当参数有改变时，重新再写一次文件。

其实也可以单独修改文件中的某一部分，但是经过评估，发现数据量不大，直接擦了重写，更加简单方便！

5.2.3 业务开发

第一个应用场景：**连接路由**

即需要控制 WiFi 模块，连接到特定的路由，进行网络通信、数据交互等。

此时可以参考文章：[「详细记录 | Realtek RTL8188FU WiFi 连接路由」](#)

第二个应用场景：**开启 AP 热点**

也就是设备本身，需要发射出来信号，以供其他设备连接。

此时可以参考文章：[「详细记录 | Realtek RTL8188FU WiFi 开启热点」](#)

其实还有另外一个应用场景：**Web服务器**，不过我想后边单独出一篇文章来写。

是什么呢？简单说就是一个 `HTTP Server`，这个 `Server` 可以处理 `POST` 请求，进而实现与其他移动端设备的数据交互。

结合本项目，是用于 APP 给设备配置功能参数。

5.3 问题汇总

由于自己在做这一块的时候，遇到太多坑了，因此觉得很有必要再总结一下，希望能帮到您~

WiFi 配置为 AP 模式需要做什么？注意什么？

WiFi 开启 AP 模式，说白了，就两点要求：**找得到，连得上**；

- **找得到**：我们总得能看到这个 WiFi 热点吧？
- **连得上**：我们即使看到了，也得能正常连上去吧？

这里进一步延伸，就有两个技术点

- 必须要选择合适的天线，否则信号较难发射出来，又或者信号不好；
- 服务端需要有 DHCP 功能，确保设备可以正常连接，正常获取到 IP 地址；

其实脚本文件中的这一步，就是自动分配 IP 地址的过程。

```
# 启动服务端，DHCP自动分配IP地址
udhcpd /App/udhcpd.conf
```

如果没有开启 DHCP 功能，有什么异常？

我实际遇到的异常是：手机打开 WiFi 开关，点击连接 WiFi，发现一直处于 **获取 IP 地址的过程中**，始终无法连接上去，后经查找资料，才发现还需要服务端自动分配 IP 地址。

WiFi 开启 AP 模式，信道选择有什么注意的？

关于这个问题，强烈推荐这篇教程 [为什么WiFi自动信道选到的信道多数在1/6/11 \(https://blog.csdn.net/linuxjourney/article/details/39828553\)](https://blog.csdn.net/linuxjourney/article/details/39828553)

经过实际测试，发现修改为信道 11，效果要好一点。当然，这个还是要结合实际情况来定 😊

具体表现就是：可以更快的找到 WiFi 热点，连接也比较稳定。

PS：自我感觉了解的不是很充分，如果您有不同见解，欢迎留言指出，十分感谢！比心 ❤️❤️❤️

如果信道选择的不对，造成的问题，具体表现有这几种

- WiFi 信号强度不好；
- WiFi 信号稳定性不好；
- 找到 WiFi 路由，需要的时间较久；

WiFi 模块天线选择问题

这个问题，极其重要。

刚开始拿到样品后，我们使用现有的其他天线，有好几种不同的，发现测试结果总是不尽人意。

一个 8M 的文件，传输过程需要好几分钟；实际测到的速度，也就几十 kb，用起来也很糟心。

后来我们的硬件工程师，单独把我们的样机，送到天线厂商那做了适配，回来后，发现效果还真不错。

因此，**一款合适的天线，还是极其重要的！！**

WiFi 模块相关问题，是否跟驱动有关系？

这个问题，我目前还没有测试。

但是之前把问题反馈给供应商的时候，他们是给了一个新的 WiFi 驱动程序，让再测试一下。

因此，自我感觉，还是有一定影响的。

天线安装位置问题

由于我们的设备，是集成在一个很小的外壳里边，而且有大部分面积，都是金属材料。

然后设备还有 4G、WiFi、两个蓝牙模块，这些无线模块之间，信号频段又会互相造成干扰。

因此安装位置一定要找好，多考虑考虑。

5.4 小结

第一次做这样的总结复盘，如有不适之处，烦请提出您的宝贵意见，谢谢您！

我一直相信「**兼听则明，偏听则暗**」，所以有问题请一定记得给我留言哈~

然后这篇文章，也算断断续续写了两周时间，期间在忙工作、也在忙一些其他的事情。经过这几天的加班加点，总算赶出来了。

时间：2021年4月26日19:54:28

时间：2021年5月8日23:56:50

时间：2021年5月9日01:48:25

时间：2021年5月9日14:21:11

文章标题：WiFi 驱动移植过程总结思考

6 HTTP 服务器

您好呀，我是小二。

本期为大家带来一个 WiFi 应用的实际场景，其实在之前「[我对 WiFi 驱动移植过程，做了一次总结复盘](#)」这篇文章中有简单提过，但由于内容较多，就单独摘出来了。

来自读者的催更🙏🙏🙏，别着急，小二在努力了！



杜·MR.

期待WebSever精彩出炉！

👍 1



嵌入式基础学习(作者)

感谢支持，最近工作太忙了，可能要晚点😓😓😓

👍 1

6.1 应用场景

我喜欢讲一个东西，先来探讨下他的应用场景。

毕竟知道了用在哪，怎么用，才能写好代码嘛，哈哈😄

本次项目，实际的应用场景：**通过手机 APP，连接到设备 WiFi 热点，进行设备的参数配置；**

接着结合实际框架，拆分需求如下：

- 采用 HTTP 协议，使用 POST 方法；
- 设备端需要作为一个 HTTP 服务器；
- 不需要支持 CGI，通过 APP 展示界面即可；

好了，明白了需求，接下来就是实现方案了。

6.2 方案论证

刚开始的时候，一点头绪都没有。

组长指导的一种方案，就是在新唐数据手册中看到的，采用 Lighttpd 的方案。

后来去查资料，发现一篇介绍不同 HTTP 服务器的文章，挺不错的，如下

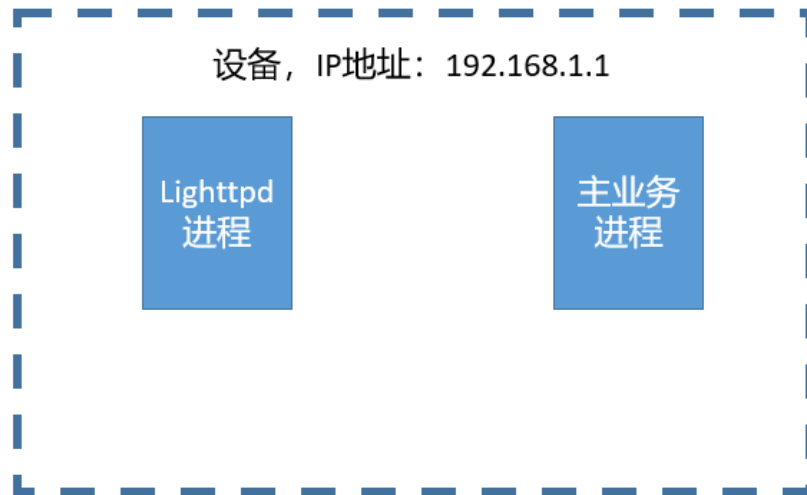
[常见的几种嵌入式web服务器 \(https://www.cnblogs.com/quliuliu2013/p/12786301.html\)](https://www.cnblogs.com/quliuliu2013/p/12786301.html)

6.2.1 Lighttpd

这块刚开始是同事在搞，后来我这边的需求规划出来后，就在想能否使用这种方案。

毕竟方案是现成的，可以节省不少时间。

后来经过分析，发现 Lighttpd 需要一个单独的进程执行，如下图所示



Lighttpd 提供了 CGI 接口，支持 IE 访问固定界面，然后进行参数配置。

实际实现方案，我认为比较好的方法是，Lighttpd 进程接收到参数变更时，写入配置文件 A，主业务进程监测文件 A 是否有改变，如果检测到改变，则读一次数据。

具体实现方式，参考下图



结合实际情况分析，

- 1、目前只维护一个主业务进程，如果再增加额外的进程，则维护成本将大大增加。
- 2、实际不需要 CGI 接口，不需要支持 IE，APP 做界面显示即可。

经过论证，此方案较复杂，暂且当做最后的备选方案。

6.2.2 cpp-httplib

接着我就去 GitHub 上寻找 HTTP Server，发现 `cpp-httplib` 这个比较好用的库。

[GitHub 链接: cpp-httplib \(https://github.com/yhirose/cpp-httplib\)](https://github.com/yhirose/cpp-httplib)

在查看了 ReadMe 文件后，很遗憾，我用不了 😞 😞 😞

提示说 GCC 4.8 及以下版本无法正常编译，因为 `<regex>` 文件已损坏。。。

NOTE

g++

g++ 4.8 and below cannot build this library since `<regex>` in the versions are broken.

我去找了找解决方法，发现在 GCC 4.9 版本修复了这个问题，参考 Stack Overflow 回答如下

<https://stackoverflow.com/questions/12530406/is-gcc-4-8-or-earlier-buggy-about-regular-expressions>

没办法了，我们当前 GCC 版本是 4.8.3 的，肯定不能因为这个库就更换编译器呀，那只能再去找找看咯。

6.2.3 httpserver

然后就接着去搜索，发现了 `httpserver` 这个库，只有一个 `.h` 头文件，感觉很简单。

[GitHub 链接: httpserver \(https://github.com/jeremycw/httpserver.h\)](https://github.com/jeremycw/httpserver.h)

分析本质需求，发现只需要在主进程中，跑一个 HTTP Server 的线程，监听固定端口，然后采用 HTTP 协议进行通信即可。

简要功能，如下图所示



从上图可以看出

- 1、主进程中，单独跑一个 HTTP Server 的线程，监听固定端口 8888；
- 2、此线程同时进行数据处理，将参数写入文件 A；
- 3、其他业务线程，在需要参数时，直接去文件 A 获取最新参数即可；

6.3 实现方式

具体实现方式，参考 ReadMe 文件，也很方便实现。

简述一下主要流程：

- 绑定监听端口号，绑定回调函数；
- 死循环监听端口；
- 当需要关闭 HTTP 服务时，通过 flag 标志位，改变当前状态；
- 关闭 HTTP 服务后，需要释放相应资源；
- 所有的配置处理接口，在 HandleRequest 回调函数中；

(PS：我这电脑网络有问题，GitHub 一直打不开，手机热点也不行，暂时还没别的好办法，只能麻烦您自己去网页上看啦。)

6.4 注意事项

在使用过程中，我这遇到一个问题。

您看下边这块代码，是这样的


```
1 void hs_init_session(http_request_t* session) {
2     session->flags = HTTP_AUTOMATIC;
3     session->parser = (http_parser_t){};
4     session->stream = (hs_stream_t){};
5     if (session->tokens.buf) {
6         free(session->tokens.buf);
7         session->tokens.buf = NULL;
8     }
9     http_token_dyn_init(&session->tokens, 32);
10 }
11
```

在第 3、4 行末尾，直接就是一个大括号，里边什么都没写。

然后我的程序在这个地方就一直编译不过去。后来在公司前辈指点下，按照下图所示，添加了 0 之后，就能编译通过了。

推测原因是当前编译器使用的 C 标准，不支持这么高级的用法。

```
1047 void hs_init_session(http_request_t* session) {
1048     session->flags = HTTP_AUTOMATIC;
1049     session->parser = (http_parser_t){0 };
1050     session->stream = (hs_stream_t){ 0 };
1051     if (session->tokens.buf) {
1052         free(session->tokens.buf);
1053         session->tokens.buf = NULL;
1054     }
1055     http_token_dyn_init(&session->tokens, 32);
1056 }
```



注意：需要修改的不止这一处，其他地方如有编译报错，也需做类似修改。

6.5 小结

针对本次的功能需求，最困难的地方，在于寻找一个合适的 HTTP Serve 库来使用。

过程虽然艰难，但也锻炼了自己找东西的能力。

7 联网优先级

之前的文章，讲了如何移植 WiFi 驱动，如何使用 WiFi 功能；没看过的小伙伴点这里[\[我对 WiFi 驱动移植过程，做了一次总结复盘\]](#)

本篇文章分享一个实际功能需求，简要描述为：**当设备通过多种方式联网时，如何控制网络连接方式的优先级？**

7.1 需求场景

设备可以通过三种方式连接网络，分别是：

- 4G
- WiFi
- 有线网络

功能需求：就是要能控制这三种联网方式的优先级，实现不同场景的切换。

举个例子，如果设备当前既可以连接 WiFi（只有局域网），又可以连接 4G 模块，此时设备需要同时连接 **互联网 + 局域网**。如果不做优先级控制，设备全部通过 WiFi 联网，则 **只能连接局域网，无法连接互联网**。

此时，如果我们可以控制优先级，当连接互联网时，优先通过 4G 连接；当连接局域网时，通过 WiFi 连接，这不就实现了我们想要的功能吗？

7.2 方案分析

如何实现我们想要的功能呢？

首先把所有功能组合列出来，然后根据需求，看是否可以做一些简化。如下图

序号	4G	WiFi	有线	联网方式	备注	简化思路
1	×	×	×	不联网	1、不考虑此种情况；	
2	×	×	√	纯有线联网	1、可连接外网； 2、可连接局域网；	
3	×	√	×	纯 WiFi 联网	1、可连接外网； 2、可连接局域网；	
4	×	√	√	WiFi + 有线	1、两种方式都可连接外网，都可连接局域网； 2、个人认为两种方式同时存在时，通过有线联网更稳定一些；	1、有线优先，简化为纯有线联网；
5	√	×	×	纯 4G 联网	1、无 WiFi 环境，无法通过有线联网； 3、只能连接外网，无法连接局域网；	
6	√	×	√	4G + 有线	1、外网连接优先通过 4G； 2、局域网连接通过有线方式；	
7	√	√	×	4G + WiFi	1、外网连接优先通过 4G； 2、局域网连接通过 WiFi 方式；	
8	√	√	√	4G + WiFi + 有线	1、理论上此情况可简化；	1、有线优先，简化为 4G + 有线；

可以看到，图片中相同颜色的，我把他归纳为同一类。

纯有线联网 == WiFi + 有线连接

由于这两种方式都可以连接外网 + 局域网，但是由于 **有线网络更加稳定**，因此简化为使用有线连接网络。

4G + 有线联网 == 4G + WiFi + 有线联网

这两种方案，基本与上述方案处理情况一致，即 **WiFi 和有线网络同时有效时，优先使用有线网络**。

7.3 场景举例

通过查找资料，发现可以通过控制 metric，来控制对应网卡优先级。

参考资料释义：

metric Metric 为路由指定一个整数成本值标（从 1 至 9999），当在路由表（与转发的数据包目标地址最匹配）的多个路由中进行选择时可以使用。

参考网址：[linux服务器两块网卡路由优先级冲突 Metric值 \(https://blog.csdn.net/memory6364/article/details/84826150\)](https://blog.csdn.net/memory6364/article/details/84826150)

简单理解为，从目的地 A 到目的地 B，需要经过的路由器跳数（中转数）。

此数值越小，则对应的网卡优先级越高，因为 **路程近了**，网络肯定会优先选择近的路走。

当 **数值相同** 时，则由内核随机选择一个网卡，进行数据交互。

由下图可知，绿色线路，metric 跳数为 0，也是最近的“道路”；红色线路，metric 跳数为 3，很直观的看绕路了，因此走这条路的概率就会大大降低。



(PS: metric 跳数仅做举例演示用，与网络中实际跳数可能不一致。)

基于此，我想到了几种场景，分别如下：

1、首先规定：4G 模块相关操作，必须断电进行；

(1) 设备只在上电时，判断一次 4G 模块状态；

(2) 在设备工作过程中，如果拔出 4G 模块，设备不会自动切换到其他连接方式，局域网连接功能不受影响；

2、4G + 有线网络同时开启

(1) 设计方案：4G 连接外网，有线连接局域网；

(2) 其他情况说明：

- 如果 4G 模块正常，则需要降低有线网卡优先级，以达到设计方案目的；
- 如果 4G 模块异常（模块不存在、网卡不存在、没插入 SIM 卡），则应使用有线连接互联网、局域网；
- 4G 模块仅在设备上电后，判断一次；

3、4G + WiFi 同时开启

(1) 设计方案：4G 连接外网，WiFi 连接局域网；

(2) 其他情况说明：

- 如果 4G 模块正常，则需要降低 WiFi 网卡优先级，以达到设计方案目的；
- 如果 4G 模块异常（模块不存在、网卡不存在、没插入 SIM 卡），则应使用 WiFi 连接互联网、局域网；
- 4G 模块仅在设备上电后，判断一次；

4、4G + 有线网络 + WiFi 同时开启

(1) 设计方案：简化为 4G + 有线网络方案；

(2) 其他情况说明：

- 如果拔出网线，则切换为 4G + WiFi 方案，对应上文第 3 点；
- 如果插入网线，则切换为 4G + 有线网络方案，对应上文第 2 点；

5、有线网络开启，WiFi 关闭

(1) 设计方案：有线网络连接互联网、局域网；

(2) 其他情况说明：

- 如果拔出网线，设备不能正常联网；
- 如果插入网线，设备可正常联网；

6、有线网络关闭，WiFi 开启或者 WiFi 热点开启

(1) 设计方案：WiFi 连接互联网、局域网；或者 WiFi 热点正常工作；

(2) 其他情况说明：

- 如果拔出网线，设备应正常使用 WiFi 功能，AP / Station；
- 如果插入网线，设备应正常使用 WiFi 功能，AP / Station；

7.4 方案实现

7.4.1 方案原理

有了上述理论知识 + 设计方案，就能来实现我们想要的功能了。

简单的说，就是控制网卡优先级，或者说 **控制网关优先级**。

当只有有线网络时，查看当前默认网关，结果如下

```
1 [root]#ip route
2 default via 192.168.3.1 dev eth0
3 192.168.3.0/24 dev eth0 proto kernel scope link src 192.168.3.10
```

此种情况，设备 **只能通过有线网关连接网络**。

如果设备同时开启有线网络 + WiFi 时，查看当前默认网关，结果如下

```
1 [root]#ip route
2 default via 192.168.3.1 dev wlan0
3 default via 192.168.3.1 dev eth0
4 192.168.3.0/24 dev eth0 proto kernel scope link src 192.168.3.10
5 192.168.3.0/24 dev wlan0 proto kernel scope link src 192.168.3.6
```

此时会发现，有两个一模一样的 **默认网关**，这个时候设备就会出现混乱，他不知道该用哪个网关进行连接。

7.4.2 基础命令

上一小节对默认网关有了基础认识。这一小节，我们就要通过命令行，来实际操作网关。

参考资料：[linux 路由表设置之 route 指令详解](https://cloud.tencent.com/developer/article/1441501)
(<https://cloud.tencent.com/developer/article/1441501>)

- 添加默认网关；

```
1 route add default gw 192.168.3.1
```

- 删除默认网关；

```
1 route del default gw 192.168.3.1
```

- 添加某一网卡默认网关;

```
1 # 添加 wlan0 网卡默认网关
2 route add default gw 192.168.3.1 wlan0
3
4 # 添加 eth0 网卡默认网关
5 route add default gw 192.168.3.1 eth0
```

- 删除某一网卡默认网关;

```
1 # 删除 wlan0 网卡默认网关
2 route del default gw 192.168.3.1 wlan0
3
4 # 删除 eth0 网卡默认网关
5 route del default gw 192.168.3.1 eth0
```

- 添加路由域; 猜想应该是代表网关所在的地址范围;

```
1 route add -net 192.168.3.0/24 eth0
```

- 删除路由域;

```
1 route del -net 192.168.3.0/24 eth0
```

- 添加默认网关, 并设置网关优先级;
- 注意: **metric 数值越低优先级越高**;

```
1 route add default gw 192.168.3.1 metric 200
2
3 # 添加 wlan0 网卡的默认网关, 并设置网关优先级
4 route add default gw 192.168.3.1 metric 100 wlan0
5
6 # 添加 eth0 网卡的默认网关, 并设置网关优先级
7 route add default gw 192.168.3.1 metric 200 eth0
```

好了, 基础命令就这么多, 剩下的就是组合利用这些命令, 实现我们想要的功能。

7.4.3 实际验证

以 **4G + 有线网络同时开启** 场景为例。

由于 4G 模块, 使用的也是命令行启动方式, 因此在启动后, 会自动设置默认的网关。

那我们就只能赶在 4G 命令启动之前, 先配置好有线网络的网关。

操作步骤简述如下:

- 删除目前有线网络默认网关;
- 添加有线网络默认网关, 并调整有线网关优先级, 类似 metric 100 ;
- 启动 4G 模块, 使其自动分配默认网关;

配置后的路由表应该类似这样:


```
1 [root]#ip route
2 default via 10.168.100.6 dev eth1
3 default via 192.168.3.1 dev eth0 metric 100
4 10.168.100.0/24 dev eth1 proto kernel scope link src 10.168.100.6
5 192.168.3.0/24 dev eth0 proto kernel scope link src 192.168.3.10
```

其中 eth1 为 4G 网卡。

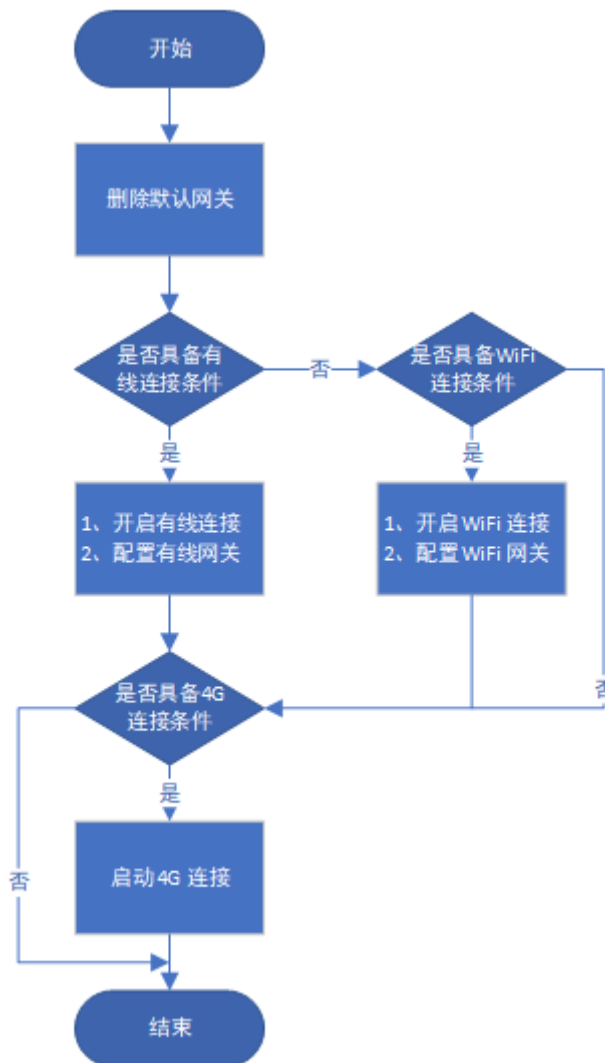
其他方案，操作步骤基本相似，只是根据各自的功能不同，略有偏差。

说到这里突然想起来，**4G + 有线网络 + WiFi 同时开启** 这个场景稍微有点复杂，也在这里分析总结一下吧。

其实在之前的方案总结中，已经拆分的很清楚了。有几个要点：

- 三种方式都能联网；
- 当有线网络与 WiFi 同时有效时，由于有线网络更稳定，因此优先级更高；
- 当有线网络失效（网线拔出、网卡down）时，可以自动连接 WiFi；

为了便于您理解，画了一个流程图，如下



图中有几个概念，在这里备注一下

1、是否具备有线连接条件？

- 网线是否插入？
- 有线网络是否已使能？

- 有线网卡是否已开启？

备注：

- 网线拔出，内核自动检测网卡 down 掉；
- **程序 down 掉有线网卡，内核会认为网线已经拔出；**

2、是否具备 WiFi 连接条件？

- WiFi 是否已使能？
- WiFi 连接路由参数是否正常？

3、是否具备 4G 连接条件？

- 4G 模块是否存在？
- 4G 网卡是否存在？
- SIM 卡是否正常？

好了，通过上述步骤，应该就可以设置我们自己的网络优先级了。

7.5 问题说明

说了这么多，这里还有几个问题，需要跟您指出来 😊

7.5.1 天线问题

4G 模块和 WiFi 模块，一定、一定、一定需要连接匹配的天线。

7.5.2 局域网连接

如何确保设备只连接到局域网？

通过 **控制 DNS** 即可间接实现。

当不想让设备连接外网时，删除默认配置的 DNS 即可。

7.6 小结

本篇文章，从需求、到实际应用场景、再到具体实现方案，详细记录了一个功能点是如何开发的，他又有哪些应用场景。

结合实际应用，分析需求，这样才更容易理解。

8 总结

本篇文章，从添加驱动文件、修改内核源码，再到命令交叉编译，再到基础业务开发等，一步一步的完成 WiFi 驱动、业务的相关开发；

当然，在开发过程中，有很多不熟练的地方，直到现在，对部分命令、文件的修改还不甚了解，但先仿照添加出来，后续深入理解了，再不断完善嘛。

PDF 文档获取

扫描下方二维码，关注公众号「编码小二」，在公众号后台回复关键字「WiFi驱动移植」即可免费获取。



也可以扫描下方二维码，添加小二微信，回复关键字「WiFi驱动移植」即可免费获取。人工处理，可能稍慢，请多多包涵💕💕💕

