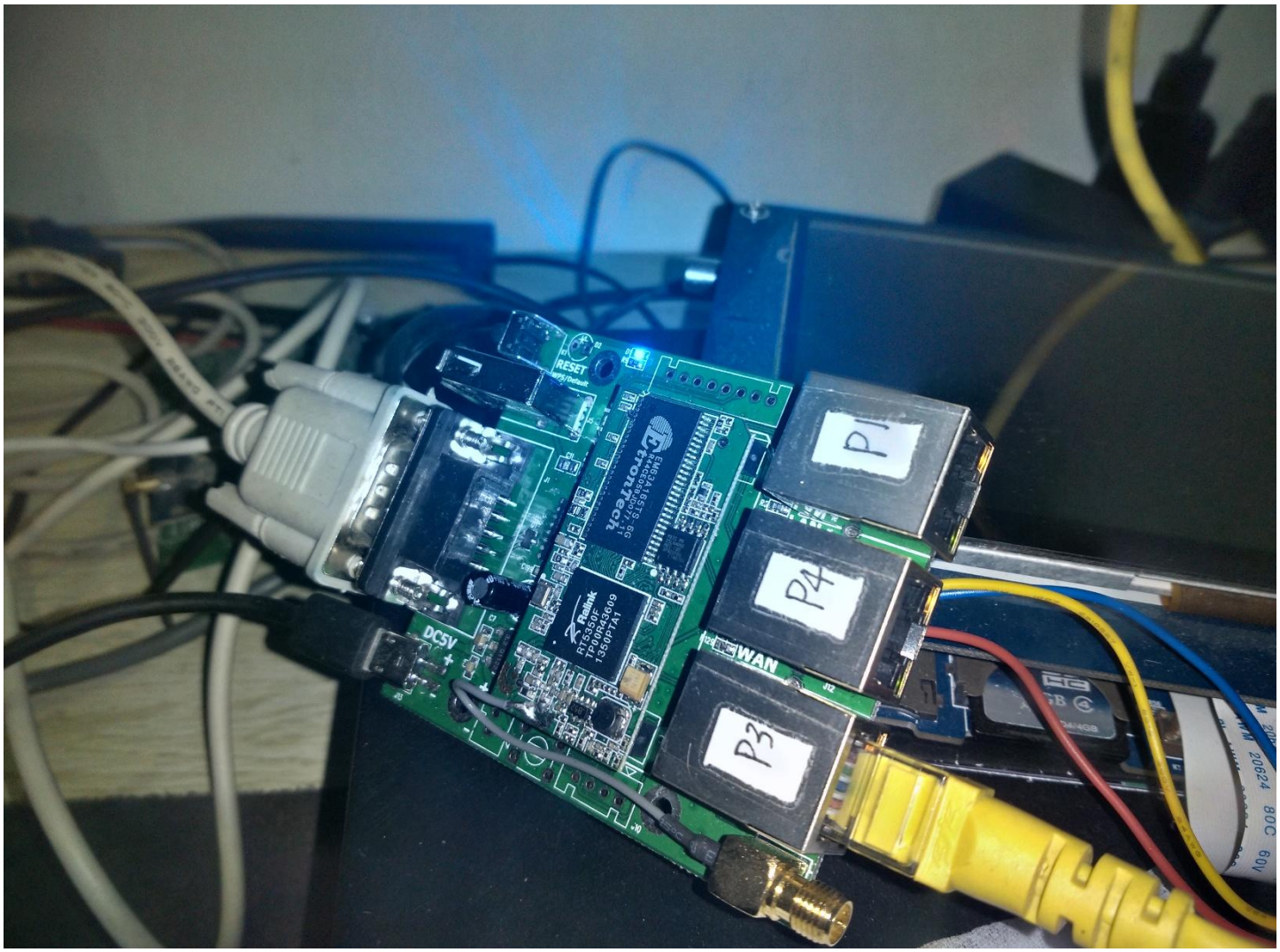
目录

1 基础

参考资料：嵌入式 Linux 学习笔记 http://pan.baidu.com/s/1fEfG6
参考书籍：
《TCPIP 协议族(第 4 版)》
《图解 TCP_IP_第 5 版》
《深入理解 LINUX 网络技术内幕》
《追踪 Linux.TCP / IP 代码运行：基于 2.6 内核.秦健》
淘宝店铺：http://yytec2008.taobao.com
个人 QQ：809205580
技术交流群：153530783



1.1 安装编译环境

root@zjh-vm:/home/work# apt-get install subversion build-essential libncurses5-dev zlib1g-dev gawk git ccache gettext libssl-dev xsltproc

1.2 下载源码

在 Openwrt 的官方网站 https://openwrt.org 上可以看到目前的最新稳定版为 BarrierBreaker 14.07，在自己的虚拟机 Ubuntu 系统中创建一个工作目录

root@zjh-vm:/home/work# mkdir openwrt

然后使用 svn 工具下载最新稳定版的源代码，下载完成后，进入 barrier_breaker 目录

```
root@zjh-vm:/home/work# cd openwrt
root@zjh-vm:/home/work/openwrt# svn co svn://svn.openwrt.org/openwrt/branches/barrier_breaker
root@zjh-vm:/home/work/openwrt# cd barrier_breaker/
```

首先可以执行 make help 查看一些帮助

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# make help
This is the buildsystem for the OpenWrt Linux distribution.

Please use "make menuconfig" to configure your appreciated
configuration for the toolchain and firmware.

You need to have installed gcc, binutils, bzip2, flex, python, perl, make,
find, grep, diff, unzip, gawk, getopt, subversion, libz-dev and libc headers.

Run "./scripts/feeds update -a" to get all the latest package definitions
defined in feeds.conf / feeds.conf.default respectively
and "./scripts/feeds install -a" to install symlinks of all of them into
package/feeds/.

Use "make menuconfig" to configure your image.

Simply running "make" will build your firmware.
It will download all sources, build the cross-compile toolchain,
the kernel and all choosen applications.

You can use "scripts/flashing/flash.sh" for remotely updating your embedded
system via tftp.

The OpenWrt system is documented in docs/. You will need a LaTeX distribution
and the tex4ht package to build the documentation. Type "make -C docs/" to build it.

To build your own firmware you need to have access to a Linux, BSD or MacOSX system
(case-sensitive filesystem required). Cygwin will not be supported because of
the lack of case sensitiveness in the file system.



Sunshine!
      Your OpenWrt Project
      http://openwrt.org
```

执行 svn info 查看当前下载的源码的修订版本

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# svn info
Path: .
Working Copy Root Path: /home/work/openwrt/barrier_breaker
URL: svn://svn.openwrt.org/openwrt/branches/barrier_breaker
Relative URL: ^/branches/barrier_breaker
Repository Root: svn://svn.openwrt.org/openwrt
Repository UUID: 3c298f89-4303-0410-b956-a3cf2f4a3e73
Revision: 43660
Node Kind: directory
Schedule: normal
Last Changed Author: nbd
Last Changed Rev: 43618
Last Changed Date: 2014-12-11 22:39:40 +0800 (四, 11 12 月  2014)
```

Openwrt 会经常更新源码，可以执行 svn update 更新已下载的源码

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# svn update
Updating '.':
Restored '.gitignore'
U    target/linux/generic/patches-3.10/645-bridge_multicast_to_unicast.patch
U    target/linux/generic/patches-3.10/644-bridge_optimize_netfilter_hooks.patch
……
U    tools/firmware-utils/src/mktplinkfw.c
U    toolchain/Makefile
Updated to revision 44071.
```

现在已经更新到 44071 版本了，现在再次执行 svn info 查看修订版本

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# svn info
Path: .
Working Copy Root Path: /home/work/openwrt/barrier_breaker
URL: svn://svn.openwrt.org/openwrt/branches/barrier_breaker
Relative URL: ^/branches/barrier_breaker
Repository Root: svn://svn.openwrt.org/openwrt
Repository UUID: 3c298f89-4303-0410-b956-a3cf2f4a3e73
Revision: 44071
Node Kind: directory
Schedule: normal
Last Changed Author: nbd
Last Changed Rev: 44065
Last Changed Date: 2015-01-21 00:41:46 +0800 (三, 21  1 月  2015)
```

更新 Feeds

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# ./scripts/feeds update -a
Updating feed 'packages' from 'https://github.com/openwrt/packages.git;for-14.07' ...
remote: Counting objects: 123, done.
remote: Compressing objects: 100% (76/76), done.
remote: Total 123 (delta 70), reused 92 (delta 39)
Receiving objects: 100% (123/123), 13.89 KiB | 0 bytes/s, done.
```

```
Resolving deltas: 100% (70/70), completed with 19 local objects.
From https://github.com/openwrt/packages
    ad7c25a..71719eb    for-14.07    -> origin/for-14.07
Updating ad7c25a..71719eb
```

使下载的软件包可以出现在 make menuconfig 配置菜单中

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# ./scripts/feeds install <PACKAGENAME>
```

或者

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# ./scripts/feeds install –a
```

注意：./scripts/feeds 这个脚本只是使软件包可以出现在 make menuconfig 配置菜单中，而并不是真正的安装或者编译软件。

1.3 配置

Openwrt 默认不允许使用 root 用户操作，若要使用 root 用户操作可以修改 include/prereq-build.mk 这个文件

root@zjh-vm:/home/work/openwrt/barrier_breaker# vi include/prereq-build.mk

define Require/non-root

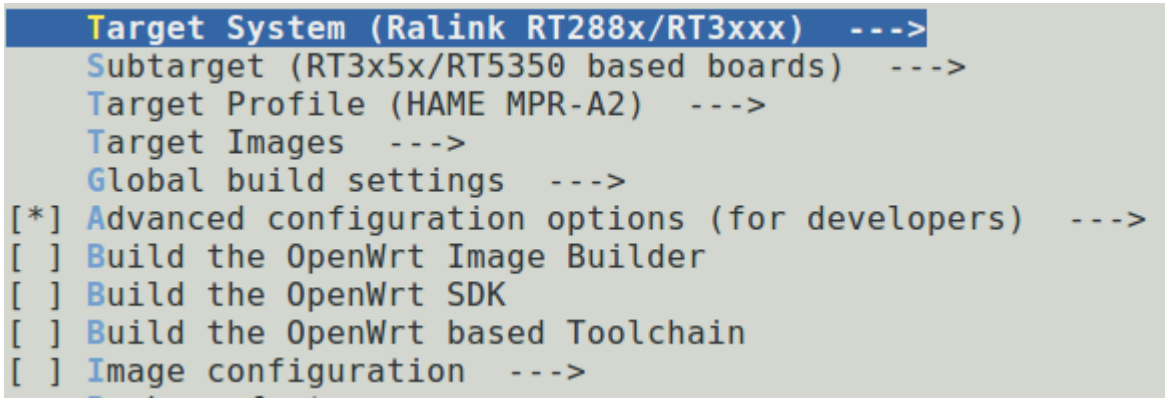
["$$(shell whoami)" != "root"]

endef

把红色那行注释掉即可使用 root 用户操作了。

执行 make menuconfig 打开配置菜单

root@zjh-vm:/home/work/openwrt/barrier_breaker# make menuconfig



每个选项前面都有一个[ ]标签，我们可以选择 y，m 和 n

- 选择 y 设置为<*>，表示将软件包编译进固件 image 文件；
- 选择 m 设置为<M>，表示软件包会被编译，但不会编译进固件 image 文件。Openwrt 会把设置为 M 选项的软件包编译后再制作成一个以 ipk 为后缀（.ipk）的文件，然后在设备上通过 opkg 来安装到设备上；
- 选择 n 设置为<>，表示不编译该软件包。

配置主要包括 4 个部分

- Target system（目标系统）
- Package selection（软件包选择）
- Build system settings（编译系统设置）
- Kernel modules（内核模块配置）

我们这里先简单配置一下，具体的以后再讲。

首先配置目标系统，根据自己的设备选择 SOC 类型，我使用的设备使用是 RT5350，是 Ralink 公司的 SOC，8MB Flash，32MB SDRAM，从 http://wiki.openwrt.org/toh/start 这个地址得知 openwrt 支持的设备

HAME

Model	Version	Status	Target(s)	Platform	CPU Speed (MHz)	Flash (MB)	RAM (MB)	Wireless NIC	Wireless Standard	Wired Ports	VLAN Config	USB
mpr-a1	-	trunk r38348	rt305x	RT5350	360 MHz	4	16	SoC	11b/g/n	1xEth	None	1 x USB2
mpr-a2	-	trunk r38348	rt305x	RT5350	360 MHz	8	32	SoC	11b/g/n	1xEth	None	1 x USB2

因此 Target System 配置如下

```
Target System (Ralink RT288x/RT3xxx)  --->
Subtarget (RT3x5x/RT5350 based boards)  --->
Target Profile (HAME MPR-A2)  --->
```

Target Profile 对应上面列表的 mpr-a2

当配置完成并保存配置后，openwrt 将根据你的配置创建一个配置文件.config

root@zjh-vm:/home/work/openwrt/barrier_breaker# vi .config

```
#
# Automatically generated file; DO NOT EDIT.
# OpenWrt Configuration
#
CONFIG_MODULES=y
CONFIG_HAVE_DOT_CONFIG=y
# CONFIG_TARGET_ppc40x is not set
# CONFIG_TARGET_realview is not set
# CONFIG_TARGET_atheros is not set
# CONFIG_TARGET_ar71xx is not set
# CONFIG_TARGET_at91 is not set
# CONFIG_TARGET_avr32 is not set
```

如果要配置 busybox，可以进行如下操作

root@zjh-vm:/home/work/openwrt/barrier_breaker# make menuconfig

Base system --->

<*> busybox..... Core utilities for embedded Linux --->

从 busybox 这个菜单按回车就可以进入 busybox 的配置界面了。

如果要配置内核选项可以执行 make kernel_menuconfig

root@zjh-vm:/home/work/openwrt/barrier_breaker# make kernel_menuconfig

```
    Machine selection  --->
[ ] Enable FPU emulation
[*] OpenWrt specific image command line hack
    Endianness selection (Little endian)  --->
    CPU selection  --->
```

1.4 编译

可以直接执行 make 进行编译
root@zjh-vm:/home/work/openwrt/barrier_breaker# make

添加 make 编译选项
root@zjh-vm:/home/work/openwrt/barrier_breaker# make -j 3 V=s
-j：多进程编译，这样编译快些。-j 指定的参数一般为你的 CPU 核数+1，比如双核 CPU 就指定为 3
V=s：输出编译的详细信息，这样编译出错时，我们才知道错在哪里

编译成功后，会在 bin 目录下生成固件文件

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# ls bin/ramips/
md5sums                                          openwrt-ramips-rt305x-vmlinux.bin
openwrt-ramips-rt305x-mpr-a2-squashfs-sysupgrade.bin  openwrt-ramips-rt305x-vmlinux.elf
openwrt-ramips-rt305x-root.squashfs              packages
openwrt-ramips-rt305x-uImage.bin
```

其中的 openwrt-ramips-rt305x-mpr-a2-squashfs-sysupgrade.bin 就是用来烧写到设备的固件。

1.5 清理工程

root@zjh-vm:/home/work/openwrt/barrier_breaker# make clean
删除/home/work/openwrt/barrier_breaker/bin 和/ home/work/openwrt/barrier_breaker/ build_dir 这 2 个目录

root@zjh-vm:/home/work/openwrt/barrier_breaker# make dirclean
删除/home/work/openwrt/barrier_breaker/bin、/ home/work/openwrt/barrier_breaker/ build_dir、
/ home/work/openwrt/barrier_breaker/staging_dir、/ home/work/openwrt/barrier_breaker/staging_di r/toolchain 和
/ home/work/openwrt/barrier_breaker/staging_dir

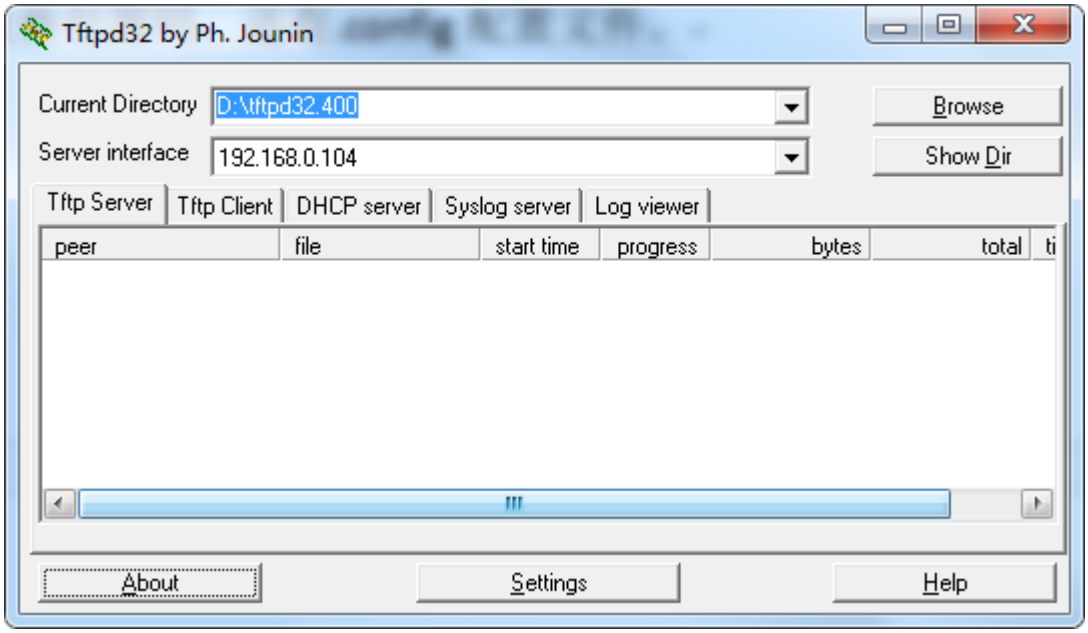
root@zjh-vm:/home/work/openwrt/barrier_breaker# make distclean
删除所有编译的或者配置的和下载的 feeds 内容以及下载的软件包源码，还有.config 配置文件。

1.6 编译/清理单个软件包

以 uci 这个软件包为例
root@zjh-vm:/home/work/openwrt/barrier_breaker# make package/system/uci/compile V=s　　编译
root@zjh-vm:/home/work/openwrt/barrier_breaker# make package/system/uci/clean V=s　　清理

1.7 烧写固件

打开 tftpd32，把刚才编译生成的固件 openwrt-ramips-rt305x-mpr-a2-squashfs-sysupgrade.bin 拷贝到 tftpd32 软件目录



下面的操作根据自己的设备的 u-boot 操作，不同的 u-boot 操作步骤和设置不同。

用网线连接设备和电脑，用串口线连接设备和电脑，打开 SecureCRT，波特率设置为 57600。给设备上电启动设备，

```
 ##### The CPU freq = 360 MHZ ####
 estimate memory size =32 Mbytes

Please choose the operation:
   1: Load system code to SDRAM via TFTP.
   2: Load system code then write to Flash via TFTP.
   3: Boot system code via Flash (default).
   4: Entr boot command line interface.
   7: Load Boot Loader code then write to Flash via Serial.
   9: Load Boot Loader code then write to Flash via TFTP.
 4
You choosed 4

 0
raspi_read: from:40028 len:6
.

4: System Enter Boot Command Line Interface.

U-Boot 1.1.3 (Nov 26 2013 - 22:36:55)
RT5350 #
```

在倒计时时按 4 进入命令行界面

设置 tftpboot 相关参数

RT5350 # set ipaddr 192.168.0.100　　设置设备 IP 地址

RT5350 # set serverip 192.168.0.104　　设置 tftp 服务器 IP 地址，即与你的设备相连的电脑的 IP 地址

RT5350 # set bootfile openwrt-ramips-rt305x-mpr-a2-squashfs-sysupgrade.bin 设置要下载的固件名称

RT5350 # save　　将设置写入 Flash

执行 reset 重启设备，在倒计时时按 2 进入 tftp 烧写固件步骤

```
 ##### The CPU freq = 360 MHZ ####
 estimate memory size =32 Mbytes

Please choose the operation:
   1: Load system code to SDRAM via TFTP.
   2: Load system code then write to Flash via TFTP.
   3: Boot system code via Flash (default).
   4: Entr boot command line interface.
   7: Load Boot Loader code then write to Flash via Serial.
   9: Load Boot Loader code then write to Flash via TFTP.

You choosed 2

 0
raspi_read: from:40028 len:6
.

2: System Load Linux Kernel then write to Flash via TFTP.
 Warning!! Erase Linux in Flash then burn new one. Are you sure?(Y/N)
```

选择 y 进入设备 IP 设置步骤，之前已经设置好了，这里就不用再设置了，直接一路回车

```
Loading: checksum bad
Got ARP REQUEST, return our IP

ArpTimeoutCheck
T T T T Got ARP REPLY, set server/gtwy eth addr (44:8a:5b:ec:49:27)
Got it
#############################################################checksum bad
############
         ####################checksum bad
##############################################
         ##########################################################################
         ##########################################################################
         ##########################################################################
```

这就表示开始下载固件了，等从电脑下载固件完成，u-boot 就会将下载的固件烧写到 Flash，烧写完成后，u-boot 自动启动系统。

```
BusyBox v1.22.1 (2015-01-18 18:36:51 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.


  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 BARRIER BREAKER (Barrier Breaker, r43618)
 -----------------------------------------------------
  * 1/2 oz Galliano         Pour all ingredients into
  * 4 oz cold Coffee        an irish coffee mug filled
  * 1 1/2 oz Dark Rum       with crushed ice. Stir.
  * 2 tsp. Creme de Cacao
 -----------------------------------------------------
root@OpenWrt:/#
```

2 UCI（统一配置接口）

2.1 概述

UCI 是 Openwrt 中为实现所有系统配置的一个统一接口，英文名 Unified Configuration Interface，即统一配置接口。每一个程序的配置文件都保存在/etc/config 目录，可以通过文本编辑器、uci（一个可执行程序）以及各种 API（Shell、Lua 和 C）来修改这些配置文件。

UCI 配置文件由一个或多个 config 语句组成，每一个 config 语句伴随着一个或多个 option 语句。这样的由一个 config 语句以及伴随的几个 option 语句组成的段落就叫做一个 section。

在/etc/config 目录下创建一个配置文件 example，其内容如下
root@OpenWrt:~# vi /etc/config/example

```
config example 'test'
        option name 'value'
        list collection 'first item'
        list collection 'second item'
```

config example 'test'中的 example 表示这个 section 的类型，'test'表示这个 section 的名称。
option name 'value'定义了一个“名称-值”对 name=value
list 定义了一个列表，表示一个名称有多个值，比如这里的 collection 就有 2 个值'first item'和'second item'

下面的例子都是正确的 UCI 语法
option example value
option 'example' value
option example "value"
option "example" 'value'
option 'example' "value"

下面的例子都是错误的 UCI 语法
option 'example" "value'　(引号不对称)
option example some value with space　(缺少引号，引起歧义，不知道哪几个单词相结合)

UCI 提供了一个命令行工具 uci，它可以对配置文件中的内容进行修改、添加、删除和读取。

```
root@OpenWrt:~# uci
Usage: uci [<options>] <command> [<arguments>]

Commands:
        batch
        export      [<config>]
        import      [<config>]
        changes     [<config>]
        commit      [<config>]
        add         <config> <section-type>
        add_list    <config>.<section>.<option>=<string>
        del_list    <config>.<section>.<option>=<string>
        show        [<config>[.<section>[.<option>]]]
        get         <config>.<section>[.<option>]
        set         <config>.<section>[.<option>]=<value>
        delete      <config>[.<section>[[.<option>][=<id>]]]
        rename      <config>.<section>[.<option>]=<name>
        revert      <config>[.<section>[.<option>]]
        reorder     <config>.<section>=<position>

Options:
        -c <path>   set the search path for config files (default: /etc/config)
        -d <str>    set the delimiter for list values in uci show
        -f <file>   use <file> as input instead of stdin
        -m          when importing, merge data into an existing package
        -n          name unnamed sections on export (default)
        -N          don't name unnamed sections
        -p <path>   add a search path for config change files
        -P <path>   add a search path for config change files and use as default
        -q          quiet mode (don't print error messages)
        -s          force strict mode (stop on parser errors, default)
        -S          disable strict mode
        -X          do not use extended syntax on 'show'
```

通过 uci 写配置文件后，整个配置会被重新写入配置文件，这意味着配置文件中任何无关行都会被删除，包括注释（# 开头的内容）。因此如果你要保留自己写的注释，就只能通过文本编辑器修改配置文件。

下面列出详细用法

Command	Target	Description
commit	[<config>]	Writes changes of the given configuration file, or if none is given, all configuration files, to the filesystem. All "uci set", "uci add", "uci rename" and "uci delete" commands are staged into a temporary location and written to flash at once with "uci commit". This is not needed after editing configuration files with a text editor, but for scripts, GUIs and other programs working directly with UCI files.
batch	-	Executes a multi-line UCI script which is typically wrapped into a *here* document syntax.
export	[<config>]	Exports the configuration in a machine readable format. It is used internally to evaluate configuration files as shell scripts.
import	[<config>]	Imports configuration files in UCI syntax.

Command	Target	Description
changes	[<config>]	List staged changes to the given configuration file or if none given, all configuration files.
add	<config> <section-type>	Add an anonymous section of type section-type to the given configuration.
add_list	<config>.<section>.<option>=<string>	Add the given string to an existing list option.
show	[<config>[.<section>[.<option>]]]	Show the given option, section or configuration in compressed notation.
get	<config>.<section>[.<option>]	Get the value of the given option or the type of the given section.
set	<config>.<section>[.<option>]=<value>	Set the value of the given option, or add a new section with the type set to the given value.
delete	<config>[.<section[.<option>]]	Delete the given section or option.
rename	<config>.<section>[.<option>]=<name>	Rename the given option or section to the given name.
revert	<config>[.<section>[.<option>]]	Revert the given option, section or configuration file.

2.2 实例操作

导出一个配置文件的内容，如果不指定配置文件则表示导出所有配置文件的内容。

```
root@OpenWrt:~# uci export example
package example

config example 'test'
        option name 'value'
        list collection 'first item'
        list collection 'second item'
```

显示一个配置文件的配置信息，若未指定配置文件，则表示显示所有配置文件的配置信息

```
root@OpenWrt:~# uci show example
example.test=example
example.test.name=value
example.test.collection=first item second item
```

这里表示 example 这个配置只有一个名为 test 的 section，其类型为 example，该 section 下有一个 option 选项，其名称为 name，值为 value，test 这个 section 下还有一个 list，其名称为 collection，该 list 有 2 个值 first item 和 second item

添加或修改一个 option

```
root@OpenWrt:~# uci set example.test.name=xxx
root@OpenWrt:~# uci set example.test.y=1
root@OpenWrt:~# uci commit
root@OpenWrt:~# uci show example
example.test=example
example.test.collection=first item second item
example.test.name=xxx
example.test.y=1
```

这里将已经存在的 example.test.name 的值修改为了 xxx，并且添加了一个新的 option：example.test.y=1

注意：当使用 set、add、rename、delete 命令操作后，再使用 show 命令查看的是修改后的未写入文件的配置信息，只有通过 commit 命令写入文件后，show 命令查看配置信息才与文件一致。

修改一个命名的 section 的类型或添加一个命名的 section

```
root@OpenWrt:~# uci set example.test=interface
root@OpenWrt:~# uci set example.lan=interface
root@OpenWrt:~# uci commit
root@OpenWrt:~# uci show example
example.test=interface
example.test.collection=first item second item
example.test.name=xxx
example.test.y=1
example.lan=interface
```

这里将已经存在的名为 test 的这个 section 的类型修改为了 interfere，同时添加了一个名为 lan，类型为 interfere 的 section

添加一个 list

```
root@OpenWrt:~# uci add_list example.test.collection='third item'
root@OpenWrt:~# uci commit
root@OpenWrt:~# uci show example
example.test=interface
example.test.collection=first item second item third item
example.test.name=xxx
example.test.y=1
example.lan=interface
```

如果要添加的 list 已经存在，则在已存在的 list 上添加成员，如果不存在在新添加一个 list。

删除一个 option 或者 section

```
root@OpenWrt:~# uci delete example.test.y
root@OpenWrt:~# uci delete example.lan
root@OpenWrt:~# uci commit
root@OpenWrt:~# uci show example
example.test=interface
example.test.collection=first item second item third item
example.test.name=xxx
```

这里删除了名为 test 的 section 的名为 y 的 option，同时删除了名为 lan 的 section

获取 section 的类型或者 option 的值

```
root@OpenWrt:~# uci get example.test
interface
root@OpenWrt:~# uci get example.test.name
xxx
```

这里获取到了配置 example 中名为 test 的 section 的类型，接着获取到了配置 example 中名为 test 的 section 的名为 name 的 option 的值。

重命名一个 section 或者 option

```
root@OpenWrt:~# uci rename example.test=exam
root@OpenWrt:~# uci rename example.exam.name=age
root@OpenWrt:~# uci commit
root@OpenWrt:~# uci show example
example.exam=interface
example.exam.collection=first item second item third item
example.exam.age=xxx
```

这里将配置 example 中的名为 test 的 section 的名称重命名为了 exam，接着以新的名称引用将名为 name 的 option 的名称修改为 age。

撤销之前的操作（在未执行 commit 命令之前）

```
root@OpenWrt:~# uci set example.exam.age=10
root@OpenWrt:~# uci revert example.exam.age
root@OpenWrt:~# uci show example
example.exam=interface
example.exam.collection=first item second item third item
example.exam.age=xxx
```

这里首先将 age 修改为 10，然后执行 revert 撤销对 age 的操作，然后执行 show 可以看到 age 的值恢复到以前的 xxx 了

匿名 section，就是没有名称只有类型的 section。比如要启动多个不同配置的 httpd，就有多个不同的针对 httpd 的 section

```
root@OpenWrt:~# uci export httpd
package httpd

config httpd
        option port '80'

config httpd
        option port '8080'
```

比如这个配置，有两个匿名 section，其类型都为 httpd，首先通过 show 显示其配置信息如下

```
root@OpenWrt:~# uci show httpd
httpd.@httpd[0]=httpd
httpd.@httpd[0].port=80
httpd.@httpd[1]=httpd
httpd.@httpd[1].port=8080
root@OpenWrt:~#
```

没有名称的 section 都是以 config.@section-type[index]这样的格式表示的具有相同类型的匿名 section，按照 index 从 0 开始编号。

获取匿名 section 的配置信息

```
root@OpenWrt:~# uci get httpd.@httpd[0].port
80
root@OpenWrt:~# uci get httpd.@httpd[-1].port
8080
```

这里的索引可以取负值，-1 表示同类型的匿名 section 中的倒数第一个 section，-2 则表示倒数第 2 个，依次类推。

添加一个匿名 section

```
root@OpenWrt:~# uci add httpd httpd
cfg068cc9
root@OpenWrt:~# uci commit
root@OpenWrt:~# uci show httpd
httpd.@httpd[0]=httpd
httpd.@httpd[0].port=80
httpd.@httpd[1]=httpd
httpd.@httpd[1].port=8080
httpd.@httpd[2]=httpd
```

为 list 选项设置分隔符

```
root@OpenWrt:~# uci -d ',' get example.exam.collection
first item,second item,third item
root@OpenWrt:~# uci -d ',' show example.exam.collection
example.exam.collection=first item,second item,third item
```

这里以','作为显示 example.exam.collection 的分隔符。

上面所有的操作，命令行程序 uci 都是从默认路径/etc/config 搜索 config，可以通过-c 指定搜索路径。

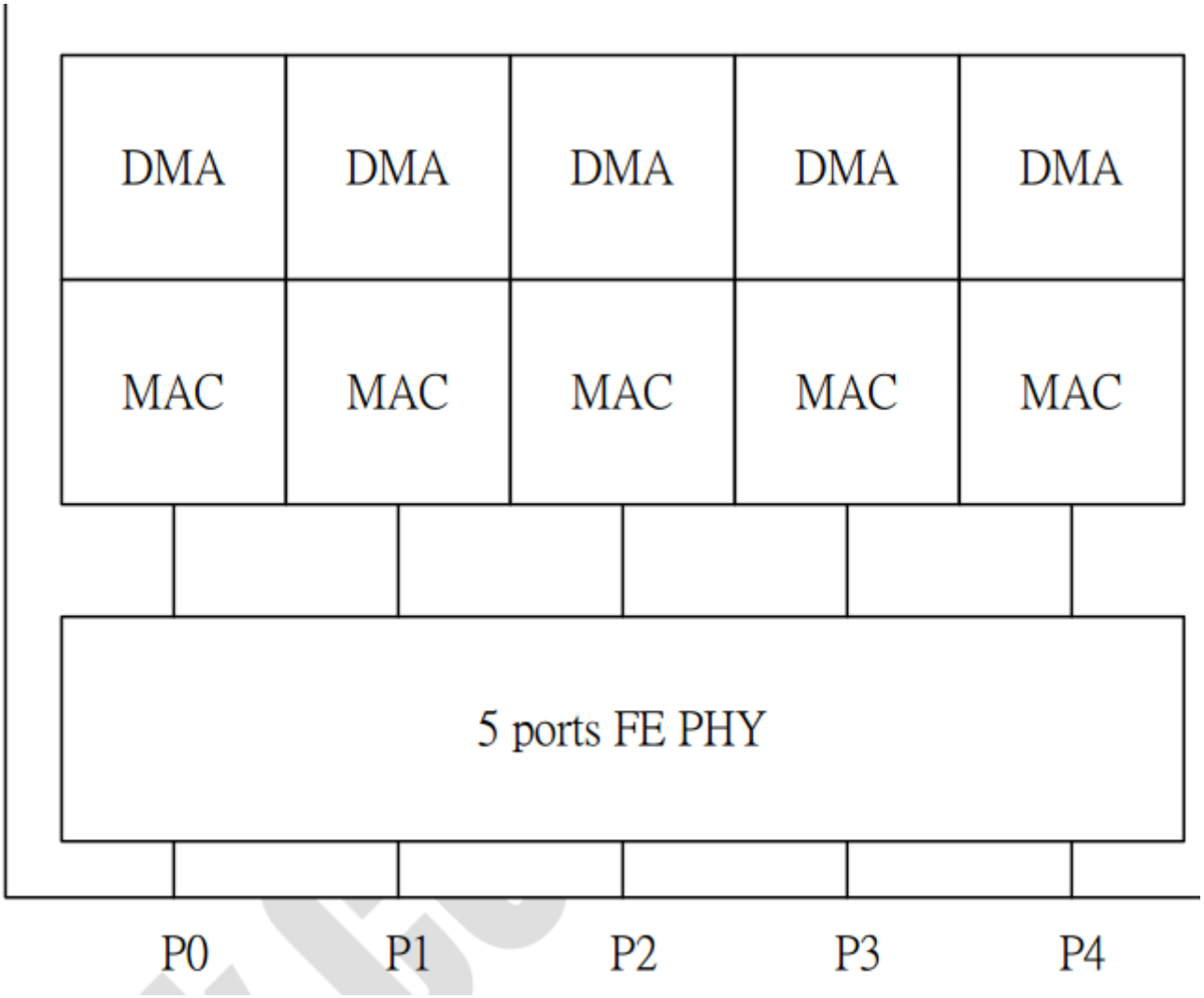
```
root@OpenWrt:~# mv /etc/config/example .
root@OpenWrt:~# uci -c . show example
example.test=interface
example.test.collection=first item second item third item
example.test.age=xxx
```

这里将默认路径/etc/config 中的配置 example 移动到当前目录，然后以-c 指定从当前目录搜索 example。

3 网络配置

3.1 概述

本章所需知识：除了需要掌握基本的 TCP/IP 协议栈，还需要掌握 VLAN、桥接。RT5350 芯片内部自带以太网交换功能和 VLAN 功能。带有 5 个 10/100Mbps 端口，支持 1K 端口地址转发表。下面是 RT5350 的部分框图



关于网络接口的配置文件为/etc/config/network

针对每个网络接口的配置都有一个类型为 interface 这样的 section，每个 interface 要么直接指向一个以太网/WIFI 接口（eth0、wlan0）或者包括多个接口的桥接。

例如：

```
config interface 'lan'
        option ifname 'eth0'
        option proto 'static'
        option ipaddr '192.168.10.1'
        option netmask '255.255.255.0'
        option gateway '192.168.1.254'
        option dns     '192.168.1.254'
```

ifname 指明了 Linux 网络接口名称。如果想桥接一个或多个接口，可以将 ifname 设置成一个 list，并且添加一个 option type 'bridge'，如下所示：

```
config interface 'lan'
        option type 'bridge'
        list ifname 'eth0.1'
        list ifname 'eth0.2'
        option proto 'static'
        option ipaddr '192.168.10.1'
        option netmask '255.255.255.0'
        option gateway '192.168.1.254'
        option dns     '192.168.1.254'
```

设置完成后，执行如下命令重启网络

root@OpenWrt:/# /etc/init.d/network restart

按照这样的配置，系统会创建一个名称为 br-lan，包含 eth0.1、eth0.2 两个桥接端口的桥接接口，使用 ifconfig 命令查看桥接接口 br-lan 如下所示：

```
root@OpenWrt:/# ifconfig br-lan
br-lan    Link encap:Ethernet  HWaddr 00:0C:43:30:50:D8
          inet addr:192.168.10.1  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:43ff:fe30:50d8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:266 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:13328 (13.0 KiB)  TX bytes:1678 (1.6 KiB)
```

使用 brctl 命令查看桥接接口 br-lan 的成员端口如下所示：

```
root@OpenWrt:/# brctl show
bridge name     bridge id               STP enabled     interfaces
br-lan          7fff.000c433050d8       no              eth0.1
                                                        eth0.2
```

proto 选项指明了 interfere 所使用的协议，包括 static、dhcp 和 pppoe。

static：ipaddr 和 netmask 这两个 option 是必须的，gateway 和 dns 是可选的。可以设置多个 dns，以空格分隔。

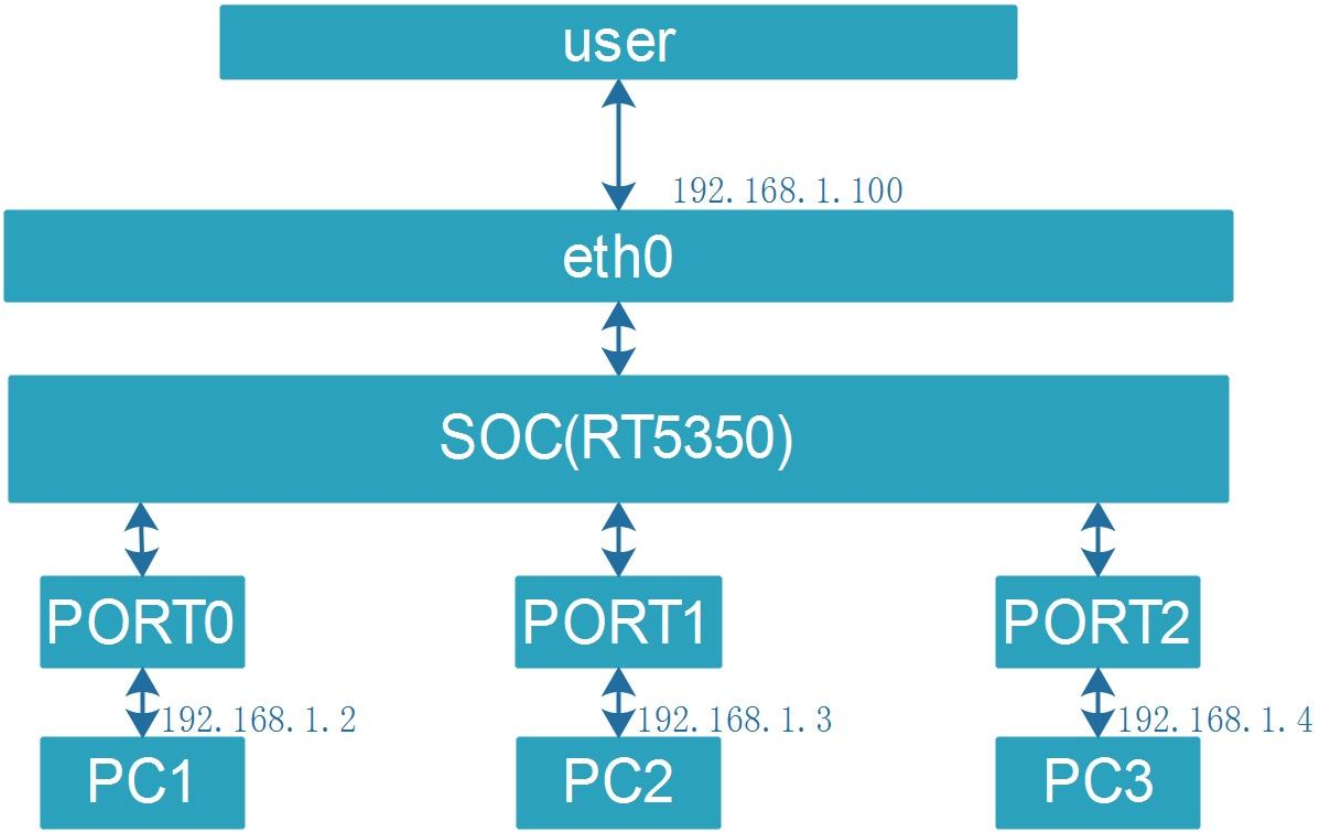
dhcp：只接受两个 option，分别是 ipaddr（想从 DHCP 服务器申请的 IP 地址）和 hostname（用于标识客户端 hostname），并且这两个 option 都是可选的。

pppoe：接受如下 option：

- username：用于 PPP 验证的用户名
- password：用于 PPP 验证的密码
- keepalive：使用 LCP 协议 ping PPP 服务器。这个 option 的值指明 ping 失败多少次后重新连接。ping 的间隔默认为 5，但是可以在 keepalive 选项中增加",<interval>"来改变间隔值。
- demand：按需拨号。该选项的值指明多长时间没使用流量就断开连接。

更多说明，参考 openwrt 官方文档：http://wiki.openwrt.org/doc/uci/network

3.2 配置成交换机



未开启 VLAN 的 RT5350 就是一个普通的交换机。相当于 eth0 把 3 个端口桥接在一起。PC1、PC2 和 PC3 处在同一链路，他们之间相互发送的数据不会进入协议栈（除了广播），即不会传递到 eth0，在芯片内部即可完成数据交换。

具体配置如下：

```
root@OpenWrt:~# uci export network
package network

config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

config interface 'lan'
        option ifname 'eth0'
        option proto 'static'
        option ipaddr '192.168.0.200'
        option netmask '255.255.255.0'
        option macaddr '02:0c:43:30:50:d8'

config switch
        option name 'rt305x'
        option reset '1'
        option enable_vlan '0'
```

把无关的配置全部删除了，只留下环回接口、lan 和 switch。option enable_vlan '0'表示禁止 VLAN。并不是所有的 SOC 都

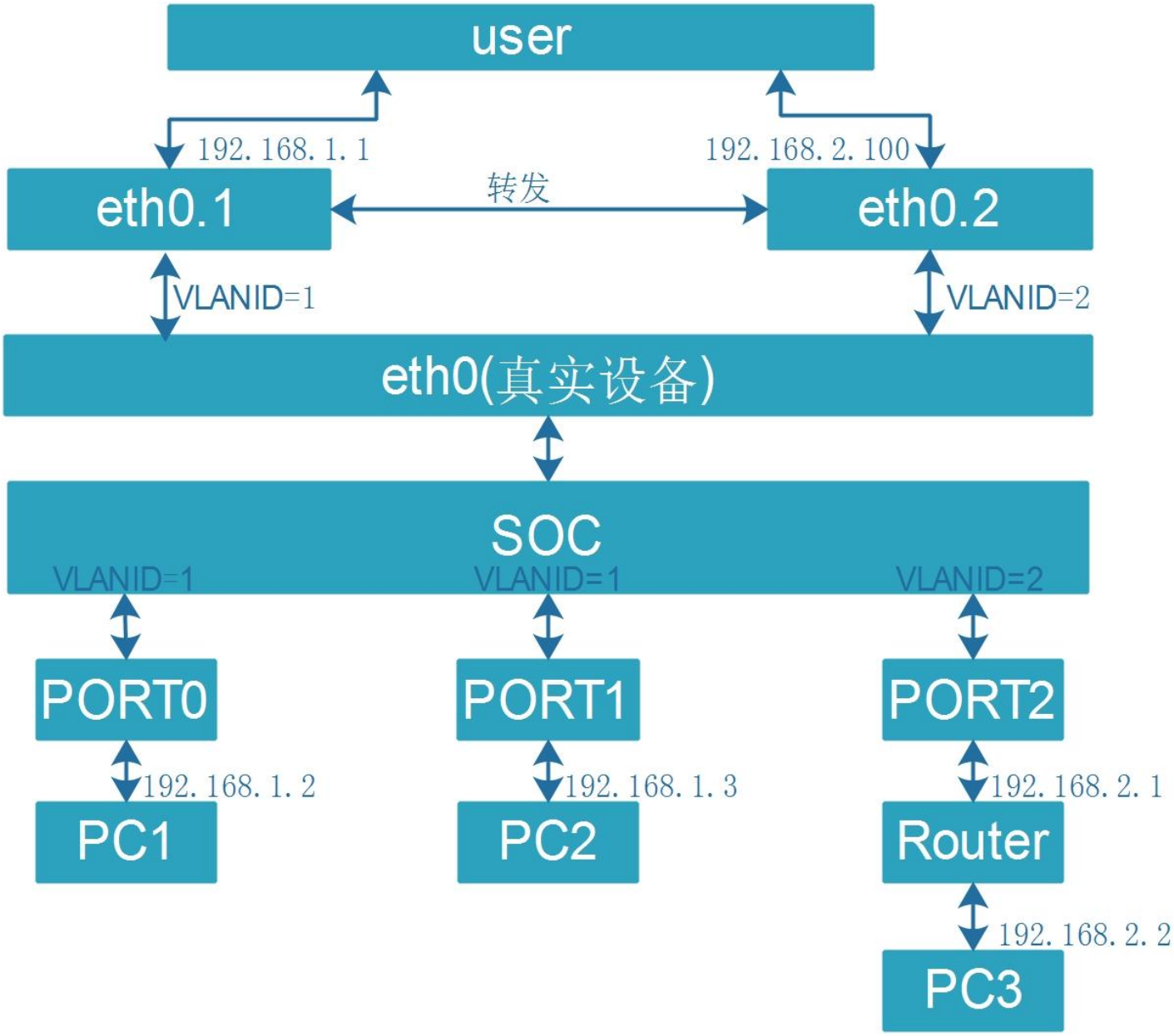
支持可编程的 switch。
然后重启网络
root@OpenWrt:~# /etc/init.d/network restart

使用 ifconfig 查看接口

```
root@OpenWrt:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:0C:43:30:50:D8
          inet addr:192.168.0.200  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::c:43ff:fe30:50d8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:11467 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4697 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1751072 (1.6 MiB)  TX bytes:853237 (833.2 KiB)
          Interrupt:5

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:45886 errors:0 dropped:0 overruns:0 frame:0
          TX packets:45886 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3120256 (2.9 MiB)  TX bytes:3120256 (2.9 MiB)
```

现在，设备就成了一台交换机了。

3.3 配置成路由器



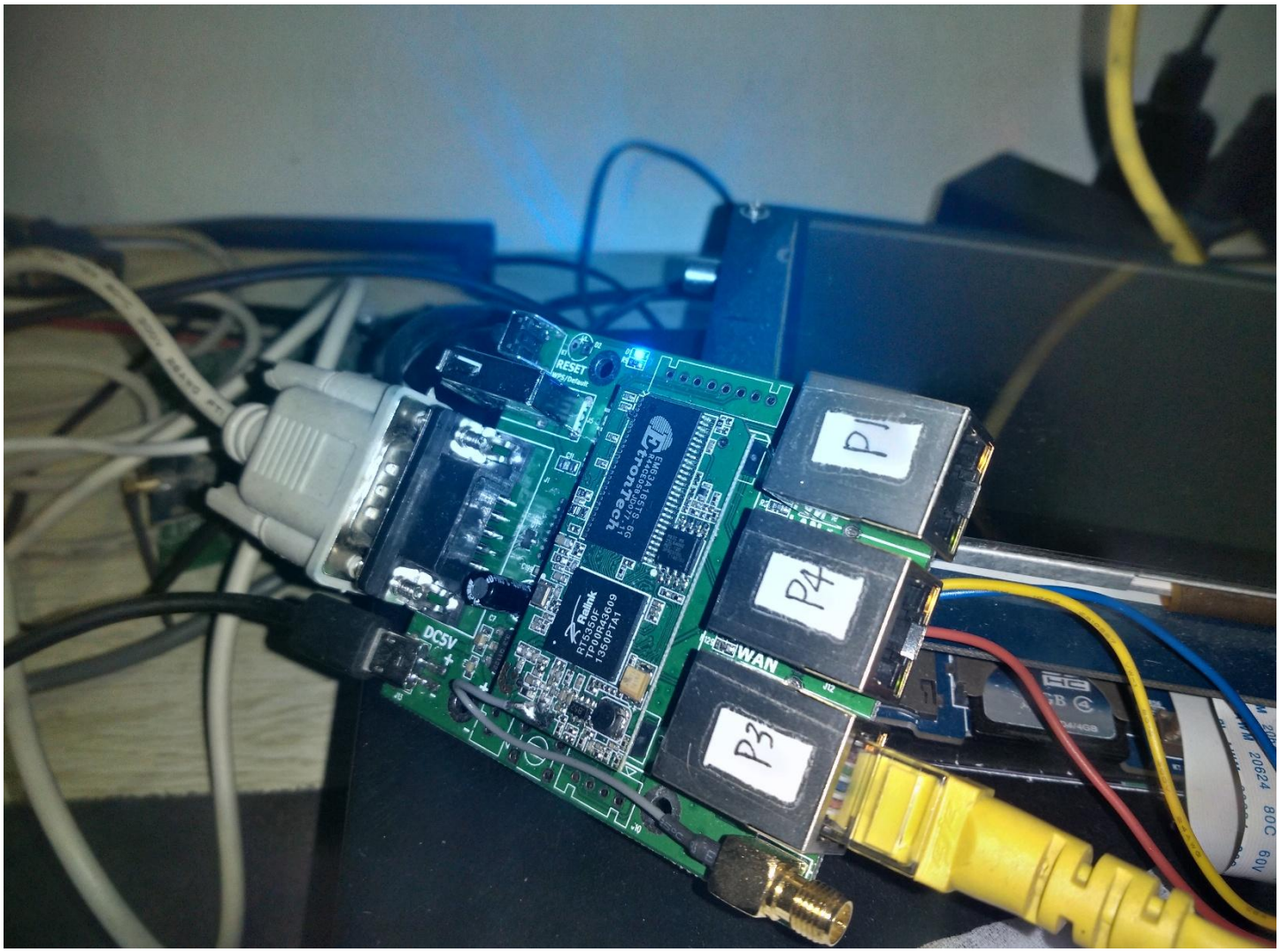
从 SOC 的每个端口进来的数据帧都会被插入 VLAN 标签。从哪个端口进来的数据插入哪个 VLANID，在系统启动时已经由应用程序设置好。比如示意图中从 PORT0 和 PORT1 进入的数据帧会被 SOC 插入 VLANID=1 的 VLAN 标签，从 PORT2 进入的数据帧会被 SOC 插入 VLANID=2 的 VLAN 标签。

eth0 从数据帧中取出 VLAN 协议标签，根据 VLANID 选择将数据帧交给对应的 VLAN 虚拟网络设备(eth0.x)处理。

示意图中 PC1 和 PC2 的 IP 在同一个网段，并且连接的是同一个 VLAN，他们之间互相单播通信将直接在 SOC 内部完成，数据帧不会进入 eth0 以上的 Linux TCP/IP 协议栈。

PC1 或者 PC2 要与 PC3 通信，他们处在不同的网段，不同的 VLAN，他们之间的数据帧会进入 eth0，eth0 根据 VLANID 将数据递交给对应的 VLAN 虚拟设备（eth0.x）处理，eth0.x 通过查找路由表决定递交到本机、转发还是丢弃。

在配置之前，需要查看原理图或设备说明书，弄清楚设备上的每个网口分别对应 RT5350 的哪个物理端口。



上图是我使用的设备，我通过查看原理图，已经将对应的物理端口在 RJ45 网口上标出。我们可以随便选择一个端口作为 WAN 口，然后其它所有端口作为 LAN 口。一般选择最边上的端口作为 WAN 口，我这里选择 P1 端口作为 WAN 口，其余的端口作为 LAN 口。这样总共分两组 VLAN，一组 VLAN 用于 LAN 口，包含 P0、P2、P3、P4 总共 4 个物理端口，另外一组 VALN 用于 WAN 口，包含一个端口 P1。

LAN——VLAN1——eth0.1：P0、P2、P3、P4

WAN——VLAN2——eth0.2：P1

具体配置如下：

```
root@OpenWrt:~# uci export network
package network

config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

config interface 'lan'
        option ifname 'eth0.1'
        option proto 'static'
        option ipaddr '192.168.10.1'
        option netmask '255.255.255.0'
        option macaddr '02:0c:43:30:50:d8'

config interface 'wan'
        option ifname 'eth0.2'
        option proto 'dhcp'
        option macaddr '02:0c:43:30:50:d9'

config switch
        option name 'rt305x'
        option reset '1'
        option enable_vlan '1'

config switch_vlan
        option device 'rt305x'
        option vlan '1'
        option ports '0 2 3 4 6t'

config switch_vlan
        option device 'rt305x'
        option vlan '2'
        option ports '1  6t'
```

首先通过 option enable_vlan '1'使能 VLAN，然后配置了两组 VLAN，一个 switch_vlan 类型的 section 表示一组 VLAN。option vlan 指定 VLANID，option ports 指定该组 VLAN 包括的端口，伴随字母 t 的端口表示从该端口离开的帧将被打上 VLAN 标记，从其它的端口离开的帧将被解除 VLAN 标签。

option ports '0 2 3 4 6t'表示从端口 0，2，3，4 离开的帧将被解除 VLAN 标签，而从端口 6 离开的帧将被打上 VLAN 标签。端口 6 是 SOC 内部的端口（查看 RT5350 的芯片手册得知 RT5350 总共有 7 个端口 P0~P6）。

按照这样的配置会创建两个 VLAN 虚拟设备，eth0.1 和 eth0.2，分别对应 VLAN1 和 VLAN2。

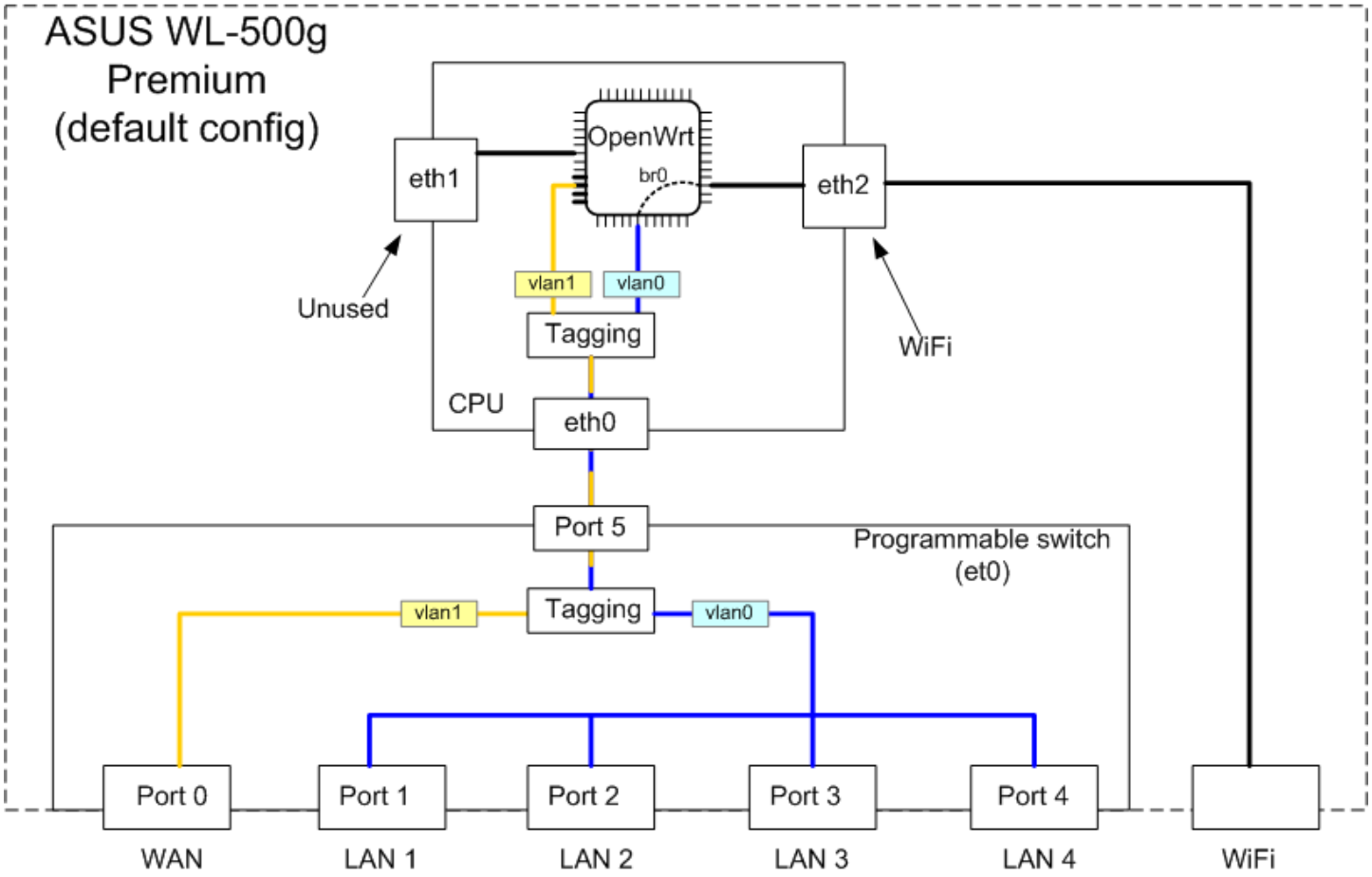
然后配置 LAN 的网络接口为 eth0.1，使用静态 IP，配置 WAN 的网络接口为 eth0.2，动态获取 IP。

然后将设备的 WAN 口连接到家里能上网的交换机或者路由器的 LAN 口，将自己的电脑连接到设备的 LAN 口，重启网络

root@OpenWrt:/# /etc/init.d/network restart

```
root@OpenWrt:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:43:30:50:D8
          inet6 addr: fe80::20c:43ff:fe30:50d8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:43719 errors:0 dropped:0 overruns:0 frame:0
          TX packets:40433 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:24247522 (23.1 MiB)  TX bytes:23618564 (22.5 MiB)
          Interrupt:5

eth0.1    Link encap:Ethernet  HWaddr 02:0C:43:30:50:D8
          inet addr:192.168.10.1  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::c:43ff:fe30:50d8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:19808 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16356 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:19979418 (19.0 MiB)  TX bytes:2262016 (2.1 MiB)

eth0.2    Link encap:Ethernet  HWaddr 02:0C:43:30:50:D9
          inet addr:192.168.0.113  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::c:43ff:fe30:50d9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:17427 errors:0 dropped:0 overruns:0 frame:0
          TX packets:21061 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2398779 (2.2 MiB)  TX bytes:20304768 (19.3 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:21 errors:0 dropped:0 overruns:0 frame:0
          TX packets:21 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2040 (1.9 KiB)  TX bytes:2040 (1.9 KiB)
```

更多关于 switch 的说明参考 openwrt 官方文档：http://wiki.openwrt.org/doc/uci/network/switch



3.4 配置 Wireless

Wireless 的配置文件为/etc/config/wireless

我们可以将 Wireless 网络接口单独作为一个 LAN 口，也可以和 eth0.1 桥接在一起，使用同一个 LAN，我们就按第 2 种方式，这样配置少一些。

首先需要将 LAN 口改为桥接，将 wireless 网络接口和之前的 LAN 口的 eth0.1 桥接在一起。

```
root@OpenWrt:/etc/config# uci set network.lan.type=bridge
root@OpenWrt:/etc/config# uci commit
root@OpenWrt:/etc/config# uci show network.lan
network.lan=interface
network.lan.ifname=eth0.1
network.lan.proto=static
network.lan.ipaddr=192.168.10.1
network.lan.netmask=255.255.255.0
network.lan.macaddr=02:0c:43:30:50:d8
network.lan.type=bridge
```

然后配置 wireless，我的配置如下：

```
root@OpenWrt:~# uci export wireless
package wireless

config wifi-device 'radio0'
        option type 'mac80211'
        option channel '11'
        option hwmode '11g'
        option path '10180000.wmac'
        option htmode 'HT20'

config wifi-iface
        option device 'radio0'
        option network 'lan'
        option mode 'ap'
        option ssid 'OpenWrt'
        option encryption 'psk2'
        option key '12345678'
```

关于配置中的各个 section 和 option 代表什么意思，在 openwrt 官方文档 http://wiki.openwrt.org/doc/uci/wireless 中有详细说明。

重启网络后使用 brctl 查看桥接信息如下

```
root@OpenWrt:~# brctl show
bridge name     bridge id               STP enabled     interfaces
br-lan          7fff.020c433050d8       no              eth0.1
                                                        wlan0
```

系统创建了一个新的虚拟网络接口 br-lan，它将 eth0.1 和 wlan0 桥接在一起。

3.5 查询接口状态信息（IP 地址、子网、网关、DNS 等）

Openwrt 为我们封装了一些 shell 函数，它们定义在/lib/functions/network.sh 中。这些函数可以方便的查询给定逻辑接口的相关信息。

首先创建一个 shell 脚本 test.sh，其内容如下所示：

```
#!/bin/sh
. /lib/functions/network.sh

if [ "$#" -ne 1 ];then
        echo "Syntax:$0 interface"
        exit 1
fi

network_get_ipaddr ip $1
echo $ip
```

如下所示：

然后给 test.sh 添加可执行权限，并执行

root@OpenWrt:/# chmod a+x test.sh

root@OpenWrt:/# ./test.sh lan

192.168.10.1

上面的 network_get_ipaddr 即是/lib/functions/network.sh 中的 shell 函数。

3.5.1 查询逻辑 interfere 的第一个 IPv4 地址：network_get_ipaddr

上面的 test.sh 就是一个查询给定逻辑 interfere 的第一个 IPv4 地址的 shell 脚本

```
root@OpenWrt:/# ./test.sh lan
192.168.10.1
root@OpenWrt:/# ./test.sh wan
192.168.0.113
```

这里分别查看接口 lan 和接口 wan 的第一个 IPv4 地址。另一个号上 network_get_ipaddrs 则是查询所有的 IPv4 地址

3.5.2 查询逻辑 interfere 所对应的 L3 层 Linux 网络设备：network_get_device

```
#!/bin/sh
. /lib/functions/network.sh

if [ "$#" -ne 1 ];then
        echo "Syntax:$0 interface"
        exit 1
fi

network_get_device ifname $1
echo $ifname
```

```
root@OpenWrt:/# ./test.sh lan
br-lan
root@OpenWrt:/# ./test.sh wan
eth0.2
```

另外 network_get_physdev 这个函数用于查询给定逻辑 interfere 所对应的 L2 层 Linux 网络设备。

3.5.3 查询逻辑接口的第一个 IPv4 子网：network_get_subnet

```
#!/bin/sh
. /lib/functions/network.sh

if [ "$#" -ne 1 ];then
        echo "Syntax:$0 interface"
        exit 1
fi

network_get_subnet subnet $1
echo $subnet
```

```
root@OpenWrt:/# vi test.sh
root@OpenWrt:/# ./test.sh lan
192.168.10.1/24
```

```
root@OpenWrt:/# ./test.sh wan
192.168.0.113/24
```

另一个号上 network_get_subnet 则是查询所有 IPv4 子网

3.5.4 查询逻辑接口（interfere）的 IPv4 网关：network_get_gateway

```
#!/bin/sh
. /lib/functions/network.sh

if [ "$#" -ne 1 ];then
        echo "Syntax:$0 interface"
        exit 1
fi

network_get_gateway gateway $1
echo $gateway
```

```
root@OpenWrt:/# ./test.sh lan

root@OpenWrt:/# ./test.sh wan
192.168.0.1
```

lan 口没有配置网关

3.5.5 查询逻辑 interfere 的 DNS 服务器：network_get_dnsserver

```
#!/bin/sh
. /lib/functions/network.sh

if [ "$#" -ne 1 ];then
        echo "Syntax:$0 interface"
        exit 1
fi

network_get_dnsserver dnsserver $1
echo $dnsserver
```

```
root@OpenWrt:/# ./test.sh lan

root@OpenWrt:/# ./test.sh wan
192.168.1.1 192.168.0.1
```

lan 口没有配置 DNS 服务器

3.5.6 查询逻辑 interfere 所使用的协议：network_get_protocol

```
#!/bin/sh
. /lib/functions/network.sh
```

```
if [ "$#" -ne 1 ];then
        echo "Syntax:$0 interface"
        exit 1
fi

network_get_protocol proto $1
echo $proto
```

```
root@OpenWrt:/# ./test.sh lan
static
root@OpenWrt:/# ./test.sh wan
dhcp
```

3.5.7 查询逻辑 interfere 的状态（UP/DOWN）：network_is_up

```
#!/bin/sh
. /lib/functions/network.sh

if [ "$#" -ne 1 ];then
        echo "Syntax:$0 interface"
        exit 1
fi

network_is_up $1

if [ $? = 0 ];then
        echo "up"
else
        echo "down"
fi
```

```
root@OpenWrt:/# ./test.sh wan
up
root@OpenWrt:/# ifconfig eth0.2 down
root@OpenWrt:/# ./test.sh wan
down
```

第一次执行 test.sh，输出 up，然后使用 ifconfig 关闭 wan 口所对应的 Linux 网络设备，再次执行 test.sh，输出 down

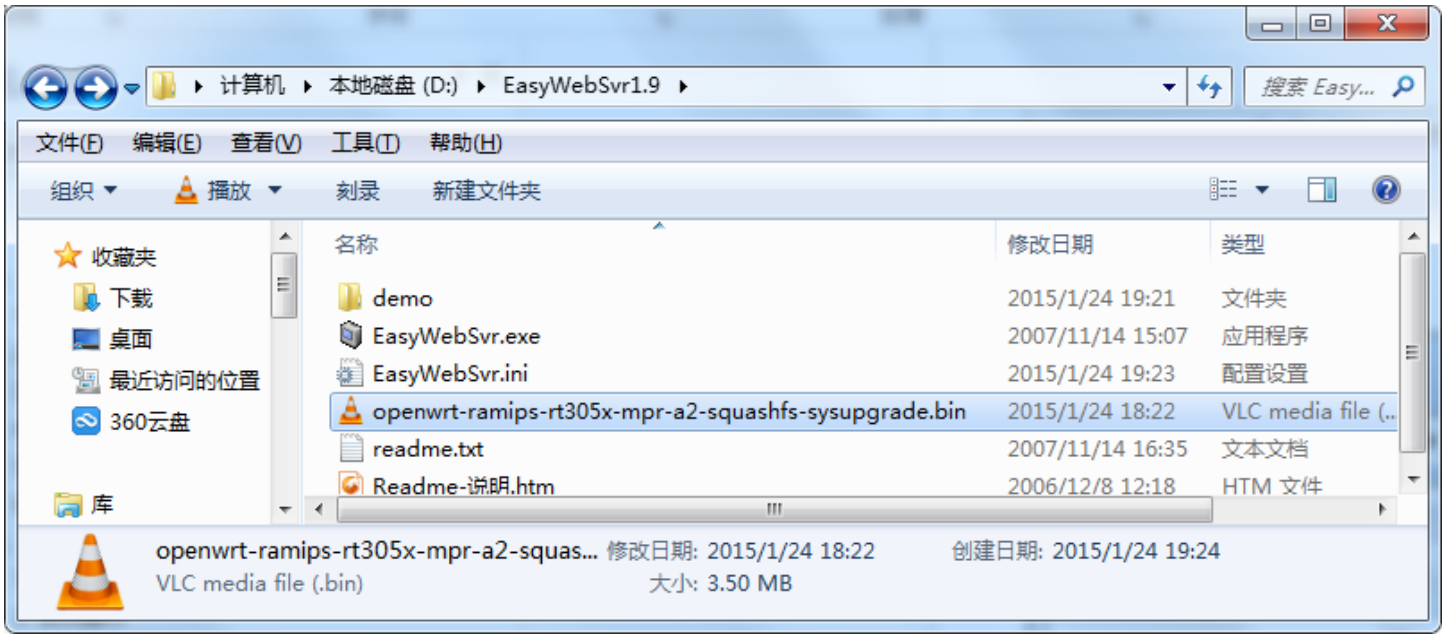
4 升级固件

如果直接从 u-boot 更新固件，会把之前的配置都重置，openwrt 提供了一个更新固件的工具 sysupgrade。下面列出其用法

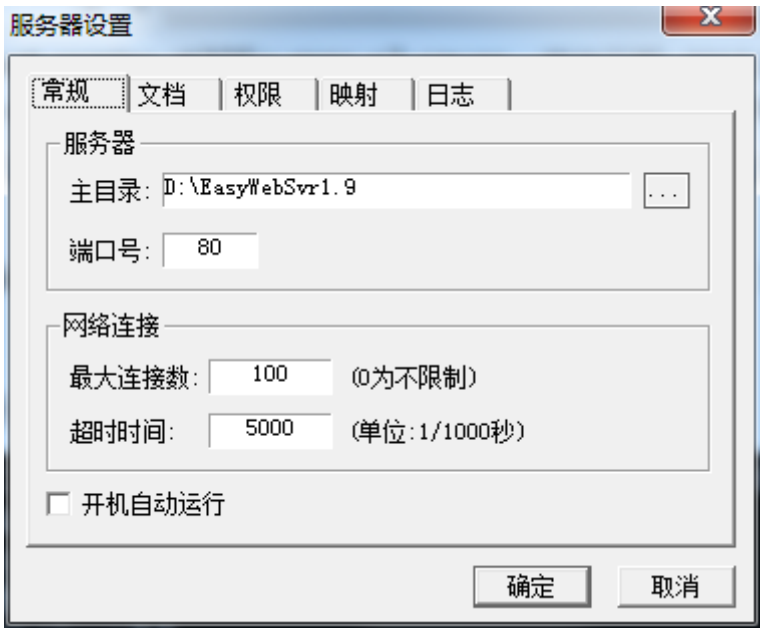
```
root@OpenWrt:/# sysupgrade
Usage: /sbin/sysupgrade [<upgrade-option>...] <image file or URL>
```

```
        /sbin/sysupgrade [-q] [-i] <backup-command> <file>

upgrade-option:
        -d <delay>      add a delay before rebooting
        -f <config>     restore configuration from .tar.gz (file or url)
        -i              interactive mode
        -c              attempt to preserve all changed files in /etc/
        -n              do not save configuration over reflash
        -T | --test
                        Verify image and config .tar.gz but do not actually flash.
        -F | --force
                        Flash image even if image checks fail, this is dangerous!
        -q              less verbose
        -v              more verbose
        -h | --help     display this help

backup-command:
        -b | --create-backup <file>
                        create .tar.gz of files specified in sysupgrade.conf
                        then exit. Does not flash an image. If file is '-',
                        i.e. stdout, verbosity is set to 0 (i.e. quiet).
        -r | --restore-backup <file>
                        restore a .tar.gz created with sysupgrade -b
                        then exit. Does not flash an image. If file is '-',
                        the archive is read from stdin.
        -l | --list-backup
                        list the files that would be backed up when calling
                        sysupgrade -b. Does not create a backup file.
```

首先下载一个运行与 Windows 下的 Web 服务器 http://www.onlinedown.net/soft/47720.htm
下载后将其解压，并将编译生成的固件拷贝到该目录



运行里面的 EasyWebSvr.exe，然后在右下角找到其图标  ，右击选“设置”，设置服务器主目录为当前目录



单击“确定，然后在右下角找到其图标  ，右击选“启动服务器”

在设备里执行更新命令

```
root@OpenWrt:/# sysupgrade -v http://192.168.0.104/openwrt-ramips-rt305x-mpr-a2-
squashfs-sysupgrade.bin
Saving config files...
etc/config/dhcp
etc/config/firewall
etc/config/luci
etc/config/network
etc/config/qos
etc/config/system
etc/config/ucitrack
etc/config/uhttpd
etc/config/wireless
etc/group
etc/hosts
etc/inittab
etc/passwd
etc/profile
etc/rc.local
etc/shadow
etc/shells
etc/sysctl.conf
killall: watchdog: no process killed
Sending TERM to remaining processes ... uhttpd ntpd ubusd logd netifd dnsmasq
Sending KILL to remaining processes ...
Switching to ramdisk...
Performing system upgrade...
Unlocking firmware ...

Writing from <stdin> to firmware ...
Appending jffs2 data from /tmp/sysupgrade.tgz to firmware...
Writing from <stdin> to firmware ...
Upgrade completed
Rebooting [  829.090000] Restarting system.
 U-Boot 1.1.3 (Nov 26 2013 - 22:36:55)
```

sysupgrade 会从指定的 URL 地址去下载固件，然后将其烧写到 Flash，并且保留所有配置。-v 表示输出详细信息。如果不想保留配置可以添加-n 选项。

5 配置 DHCP 服务器和 DNS 服务器

openwrt 使用同一个程序 dnsmasq 来实现 DHCP 服务器和 DNS 服务器。dhcpd 是一个 DHCP 服务器，openwrt 未使用。dhcpd 和 dnsmasq 在软件包配置中默认都是选中的，既然 dhcpd 没有使用，就可以不编译 dhcpd，配置 Openwrt 如下：

root@zjh-vm:/home/work/openwrt/barrier_breaker# make menuconfig

Base system ---->

<*> dnsmasq

< > dnsmasq-dhcpv6

< > dnsmasq-full

Network ---->

< > odhcpd

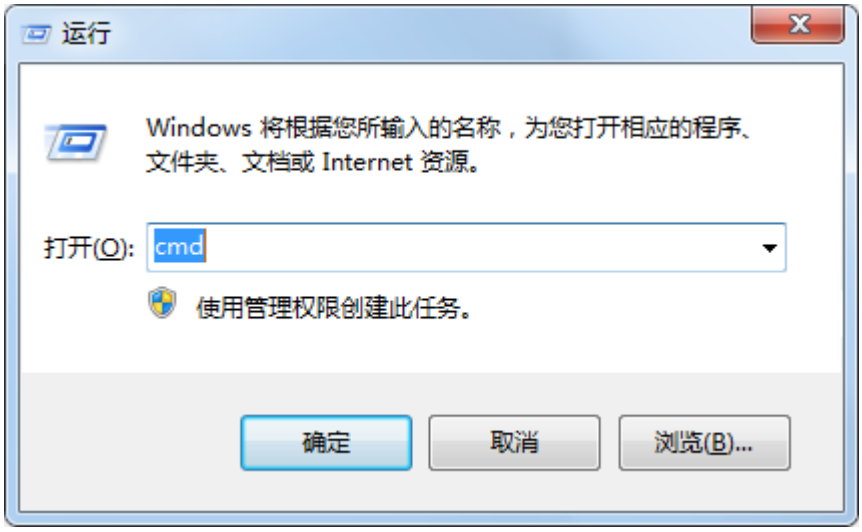
只选中 dnsmasq 这个软件包，其他几个都取消。重新编译固件，然后使用 sysupgrade 更新固件。

DHCP 服务器和 DNS 服务器的配置文件为/etc/config/dhcp

默认配置文件中包括一个公用的 section 来配置 DNS 和程序相关的一些 option，以及一个或多个为每个网络接口定义 DHCP 地址池的 section。

5.1 公用选项配置

首先在电脑上按快捷键 Win+R 打开运行对话框，输入 cmd，运行命令行窗口程序，在命令行窗口程序中执行 nslookup.exe www.baidu.com



nslook.exe 是一个域名查询工具，这里使用该工具查询域名 www.baidu.com，查询到域名 www.baidu.com 所对应的 IP 为 180.97.33.107、180.97.33.108，别名为 www.baidu.com

并且输出了查询所使用的域名服务器的域名为 OpenWrt.lan，其地址为 192.168.10.1 ，如果设备安装了 Web 服务器，就可以直接在浏览器的地址栏输入 OpenWrt.lan 来访问设备了。这里的 OpenWrt 实际上是设备的主机名，可在系统配置文件/etc/config/system 中修改 option hostname，比如修改为 zjh-openwrt

```
root@OpenWrtX:~# uci show system.@[0].hostname
system.cfg02e48a.hostname=OpenWrt
root@OpenWrtX:~# uci set system.@[0].hostname=zjh-openwrt
root@OpenWrtX:~# uci commit system
root@OpenWrtX:~# uci show system.@[0].hostname
system.cfg02e48a.hostname=zjh-openwrt
```

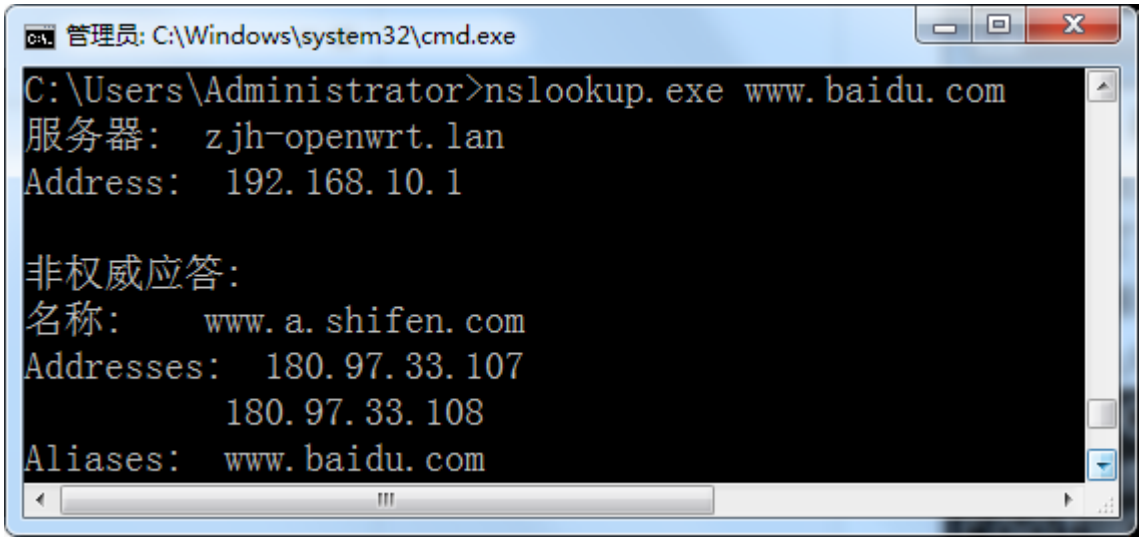
这里首先使用 uci show 命令查看之前的主机名，然后使用 uci set 命令进行修改，然后使用 uci commit 提交修改，然后再次使用 uci show 查看。

然后重启 system 程序和 dnsmasq 程序

```
root@openwrt:~# /etc/init.d/system restart
root@zjh-openwrt:~# /etc/init.d/dnsmasq restart
```

可以看到重启 system 程序后，主机名已经变成了 zjh-openwrt

在电脑上再次查询域名



可以看到域名服务器的域名已经变为 zjh-openwrt.lan

再说一下 zjh-openwrt.lan 中的 lan，这个字段在配置文件/etc/config/dhcp 中配置

option domain 'lan'

公用选项默认配置如下

```
config dnsmasq
        option domainneeded '1'
        option boguspriv '1'
        option filterwin2k '0'
        option localise_queries '1'
        option rebind_protection '1'
        option rebind_localhost '1'
        option local '/lan/'
        option domain 'lan'
        option expandhosts '1'
        option nonegcache '0'
        option authoritative '1'
        option readethers '1'
        option leasefile '/tmp/dhcp.leases'
        option resolvfile '/tmp/resolv.conf.auto'
```

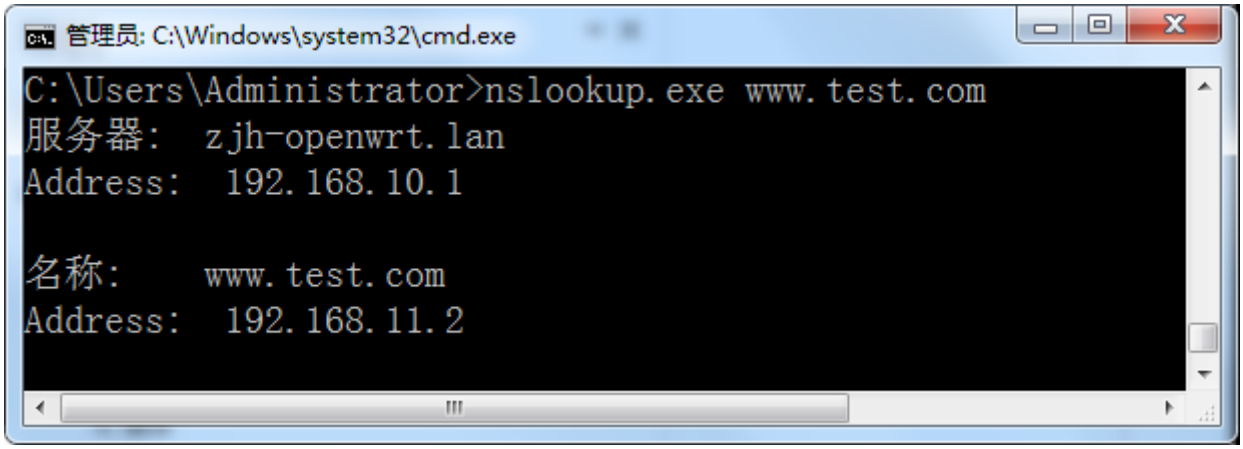
这里的每一项代表什么意思在 openwrt 官方文档 http://wiki.openwrt.org/doc/uci/dhcp 中有详细说明。

如果添加本地域名解析，可以在设备的/etc/hosts 中添加，例如添加：192.168.11.2 www.test.com

root@zjh-openwrt:~# vi /etc/hosts

```
127.0.0.1 localhost
192.168.11.2 www.test.com
```

然后重启 dnsmasq，然后在电脑上使用 nslookup.exe 查询域名 www.test.com



5.2 配置 DHCP 地址池

```
config dhcp 'lan'
        option interface 'lan'
        option start '100'
        option limit '150'
        option leasetime '12h'
```

对每一个接口都有一个类型为 dhcp 的 section
start 用来指定租给客户的最小地址。其值表示从网络接口的网络地址的偏移。比如 lan 接口的网络地址为 192.168.10.0，这里指定 start 为 100，表示租给客户的最小 IP 地址为 192.168.10.100
limit 指定租给客户的最大 IP 个数，这里指定 limit 为 150，表示最多可以租给客户 150 个 IP 地址。
leasetime 表示租期时间。

MAC——IP 绑定：指定给一个主机分配一个固定的 IP。

```
config host
        option ip '192.168.10.2'
        option mac '44:8A:5B:EC:49:27'
```

dnsmasq 首先会匹配 host 的规则，然后才匹配 dhcp 的规则，因此虽然 dhcp 指定最小 IP 为 192.168.10.100，但是先匹配 host，所以可以成功给'44:8A:5B:EC:49:27'这个 MAC 的主机分配 192.168.10.2 这个 IP 地址。

在测试 DHCP 分配地址时，可以在 Windows 上执行 ipconfig.exe /release 来释放自己已有的 IP，然后执行 ipconfig.exe /renew 来重新请求 IP 地址。

更多说明参考 openwrt 官方文档：http://wiki.openwrt.org/doc/uci/dhcp

6 添加软件包

6.1 概述

参考 openwrt 官方文档 http://wiki.openwrt.org/doc/devel/packages

下面用<BUILDROOT> 表示 Openwrt 源码树顶层目录。Openwrt 所有的软件包都都保存在<BUILDROOT>/package 目录下，一个软件包占一个子目录。一个典型的 Openwrt 用户空间软件包，例如 helloworld，如下所示

<BUILDROOT>/package/helloworld/Makefile

<BUILDROOT>/package/helloworld/files/

<BUILDROOT>/package/helloworld/patches/

其中 files 目录是可选的，一般存放配置文件和启动脚本。patches 目录也是可选的，一般存放 bug 修复或者程序优化的补丁。

其中 Makefile 是必须的也是最重要的，它具有和我们所熟悉的 GNU Makefile 完全不同的语法，它定义了如何构建一个 Openwrt 软件包，包括从哪里下载源码，怎样编译和安装。

首先需要包含几个文件进来，它们定义了一些变量、规则、函数

include $(TOPDIR)/rules.mk

include $(INCLUDE_DIR)/package.mk

include $(INCLUDE_DIR)/kernel.mk 定义内核模块时才需要

include $(INCLUDE_DIR)/cmake.mk 使用 cmake 构建软件时才需要

一个典型的 Openwrt 软件包目录下没有源码目录，构建 Openwrt 软件包的第一步是从指定的 URL 下载指定的源码包，这个 URL 在 Makefile 中定义，如下所示：

PKG_SOURCE:=helloworld.tar.gz

PKG_SOURCE_URL:=http://www.example.com/

PKG_MD5SUM:=e06c222e186f7cc013fd272d023710cb

这 3 行的意思是从 http://www.example.com/helloworld.tar.gz 下载源码包， PKG_MD5SUM 用于验证下载的软件包的完整性。下载的 helloworld.tar.gz 一般存放在<BUILDROOT>/dl/目录，然后解压到<BUILDROOT>/build_dir/target-xxx/$(PKG_BUILD_DIR)

另一种构建 Openwrt 软件包的方式是从自己写的代码构建，这种方式需要在<BUILDROOT>/package/helloworld/目录下创建一个子目录 src，用来保存自己写的源码文件，如下所示：

<BUILDROOT>/helloworld/Makefile

<BUILDROOT>/package/helloworld/files/

<BUILDROOT>/helloworld/patches/

<BUILDROOT>/helloworld/src/

<BUILDROOT>/helloworld/src/helloworld.c

<BUILDROOT>/helloworld/src/Makefile

很显然，这种方式没有任何源码包需要从 Web 下载，这里的<BUILDROOT>/helloworld/src/Makefile 和通常的 GNU Makefile 语法一样。

OpenWRT BuildPackage Makefile（例如上面的<BUILDROOT>/helloworld/Makefile）定义了一些用于指示 OpenWRT Buildroot 或者 SDK 编译软件包的变量。

PKG_*定义了和软件包相关的一些信息，如下所示：

- PKG_NAME -软件包的名称
- PKG_VERSION -下载的或者自己写的软件包的版本（必须）
- PKG_RELEASE -当前的编译版本，可能根据不同需求，编译方式不同。
- PKG_BUILD_DIR -在哪个目录下编译该软件包，默认为 $(BUILD_DIR) /$(PKG_NAME)-$(PKG_VERSION)
- PKG_SOURCE -要从网上下载的的源码包的文件名
- PKG_SOURCE_URL -从哪里下载源码包

- PKG_MD5SUM -用于验证下载的源码包的完整性
- PKG_CAT -怎样解压源码包 (zcat, bzcat, unzip)
- PKG_BUILD_DEPENDS -定义需要在该软件包之前编译的软件包或者一些版本的依赖。和下面的 DEPENDS 语法相同。

Package/xxx 描述软件包在 menuconfig 和 ipkg 中的条目（其中的 xxx 即出现在 menuconfig 中的标签以及.config 中的 CONFIG_PACKAGE_xxx=y）

- SECTION - 软件包的类型（当前未使用）
- CATEGORY - 该软件包出现在 menuconfig 中的哪个菜单下，如果是第一次使用，则 menuconfig 中会新创建一个菜单
- TITLE - 对该软件包的一个简短描述
- URL - 在哪里可以找到该软件包的源码包
- MAINTAINER -定义该软件包的维护者的联系方式
- DEPENDS - 定义需要在该软件包之前编译/安装的软件包，或者其他一些依赖条件，比如版本依赖等，多个依赖之间使用空格分隔，下面列出其具体语法：
  - DEPENDS:=+foo 当前软件包和 foo 都会出现在 menuconfig 中，当前软件包被选中时，其依赖的软件包 foo 也被自动选中；如果依赖额软件包 foo 先前没有选中而当前软件包取消选中时，其依赖分软件包 foo 也会被取消选中。
  - DEPENDS:=foo 只有当其依赖的软件包 foo 被选中时，当前软件包才会出现在 menuconfig 中。
  - DEPENDS:=@FOO 软件包依赖符号 CONFIG_FOO，只有当符号 CONFIG_FOO 被设置时，该软件包才会出现在 menuconfig 中。通常用于依赖于某个特定的 Linux 版本或目标（比如 CONFIG_LINUX_3_10、CONFIG_TARGET_ramips）。
  - DEPENDS:=+FOO:bar 当前软件包和软件包 bar 都会出现在 menuconfig 中。如果符号 CONFIG_FOO 被设置，则当前软件包依赖于软件包 bar，否则不依赖
  - DEPENDS:=@FOO:bar 如果符号 CONFIG_FOO 被设置，则当前软件包依赖于软件包 bar，否则不依赖，并且只有其依赖包 bar 被选中时，当前软件包才会出现在 menuconfig 中。

KernelPackage/xxx 描述内核模块在 menuconfig 和 ipk 中的条目（在 menuconfig 中看到的标签为 kmod-xxx）

- SUBMENU – 该模块出现在 menuconfig 中的 Kernel modules 菜单下的哪个子菜单
- TITLE – 对该模块的一个简短描述
- FILES – 模块名称（全路径），多个模块文件之间用空格分开，例如：

```
FILES:= \
    $(PKG_BUILD_DIR)/Embedded/src/1588/timesync.ko \
    $(PKG_BUILD_DIR)/Embedded/src/CAN/can.ko
```

- AUTOLOAD – 自动加载，例如：AUTOLOAD:=$(call AutoLoad,40,timesync can)表示自动加载 timesync.ko 和 can.ko，其加载的优先级为 40

Build/Prepare 定义一组指令，比如用于给源码打补丁，创建编译目录，拷贝自己的源码到编译目录。对于从网上下载源码包一般不需要定义，对于从自己写的代码构建软件包，一般按如下方式定义：

```
define Build/Prepare
    mkdir -p $(PKG_BUILD_DIR)
    $(CP) ./src/* $(PKG_BUILD_DIR)/
endef
```

Build/Configure 对于需要通过执行./ configure 的软件包可以在这里自定义。可参考 package/libs/openssl/Makefile

Build/Compile 如何编译源码，一般不用定义。可参考 package/libs/openssl/Makefile。但对于内核模块需要这样定义：

```
define Build/Compile
	$(MAKE) -C "$(LINUX_DIR)" \
		ARCH="$(LINUX_KARCH)" \
		CROSS_COMPILE="$(TARGET_CROSS)" \
		SUBDIRS="$(PKG_BUILD_DIR)" \
		EXTRA_CFLAGS="$(EXTRA_CFLAGS)" \
		$(EXTRA_KCONFIG) \
		modules
endef
```

Package/xxx/install 如何安装软件包。定义了一组命令，用来在嵌入式文件系统中创建目录或者拷贝相关文件到嵌入式文件系统或者 ipk。例如：

```
define Package/ helloworld /install
	$(INSTALL_DIR) $(1)/bin
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/helloworld $(1)/bin/
endef
```

这里的$(1)表示嵌入式根文件系统 rootfs 的根目录，比如 staging_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/root-ramips/

$(INSTALL_DIR) $(1)/bin 表示在嵌入式根文件系统中创建目录 bin

$(INSTALL_BIN) $(PKG_BUILD_DIR)/helloworld $(1)/bin/表示将可执行程序$(PKG_BUILD_DIR)/helloworld 拷贝到嵌入式根文件系统的 bin 目录下

INSTALL_XXX 在<BUILDROOT>/rules.mk 中定义

```
204 INSTALL_BIN:=install -m0755
205 INSTALL_DIR:=install -d -m0755
206 INSTALL_DATA:=install -m0644
207 INSTALL_CONF:=install -m0600
```

INSTALL_DIR 用来安装目录，安装的目录权限为所属用户可读/写/执行，其他用户可读/执行。

INSTALL_DATA 用来安装数据文件，其权限为所属用户可读/写，其他用户可读

INSTALL_CONF 用来安装配置文件，其权限为所属用户可读/写，其他用户不能读/写/执行。

Package/xxx/preinst 定义一段需要在软件包安装前执行的脚本。例如

```
define Package/helloworld/preinst
	#!/bin/sh
	echo 'before install $(PKG_NAME)'
endef
```

Package/xxx/postinst 定义一段需要在软件包安装后执行的脚本

Package/xxx/prerm 定义一段需要在软件包删除前执行的脚本

Package/xxx/postrm 定义一段需要在软件包删除后执行的脚本

注意：这些脚本必须可以同时在嵌入式设备和主机上都能执行。

最后需要调用$(eval $(call BuildPackage,xxx))或者$(eval $(call KernelPackage,xxx))，前者用于应用程序，后者用于内核模块，如果有多个软件包需要针对每个软件包调用。

6.2 实例：添加应用程序软件包

首先在<BUILDROOT>/package/目录下创建一个目录，其名称一般为你要添加的软件包的名称。比如 helloworld

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# mkdir package/helloworld
```

然后在<BUILDROOT>/package/helloworld 中创建一个目录 src，用来保存自己写的代码。
root@zjh-vm:/home/work/openwrt/barrier_breaker# mkdir package/ helloworld /src

然后将自己写的代码拷贝到<BUILDROOT>/package/helloworld/src，总共两个文件 helloworld.c 和 Makefile，其中 Makefile 内容如下：

```
all:
	$(CC) helloworld.c -o helloworldhelloworld.c
```

helloworld.c 的内容如下：

```
/* helloworld.c */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main (int argc, char **argv)
{
	unsigned int i = 0;
	while (1)
	{
		printf("hello world:%u\n", i++);
		sleep(1);
	}
	return 0;
}
```

然后在<BUILDROOT>/package/helloworld 目录下编写一个比较重要的 Makefile，其内容如下

```
include $(TOPDIR)/rules.mkinclude $(INCLUDE_DIR)/package.mk

PKG_NAME:=helloworld
PKG_VERSION:=1.0

define Package/helloworld
  CATEGORY:=My Package
  #DEPENDS:= 如果需要依赖其他软件包，在这里添加
  TITLE:=helloworld for test
  MAINTAINER:=zjh <809205580@qq.com>
endef

define Build/Prepare
	mkdir -p $(PKG_BUILD_DIR)
	$(CP) ./src/* $(PKG_BUILD_DIR)/
endef

define Package/helloworld/install
	$(INSTALL_DIR) $(1)/bin $(INSTALL_BIN) $(PKG_BUILD_DIR)/helloworld $(1)/bin/
endef

define Package/helloworld/preinst
```

```
	#!/bin/sh
	echo 'before install $(PKG_NAME)'
endef

$(eval $(call BuildPackage,helloworld))
```

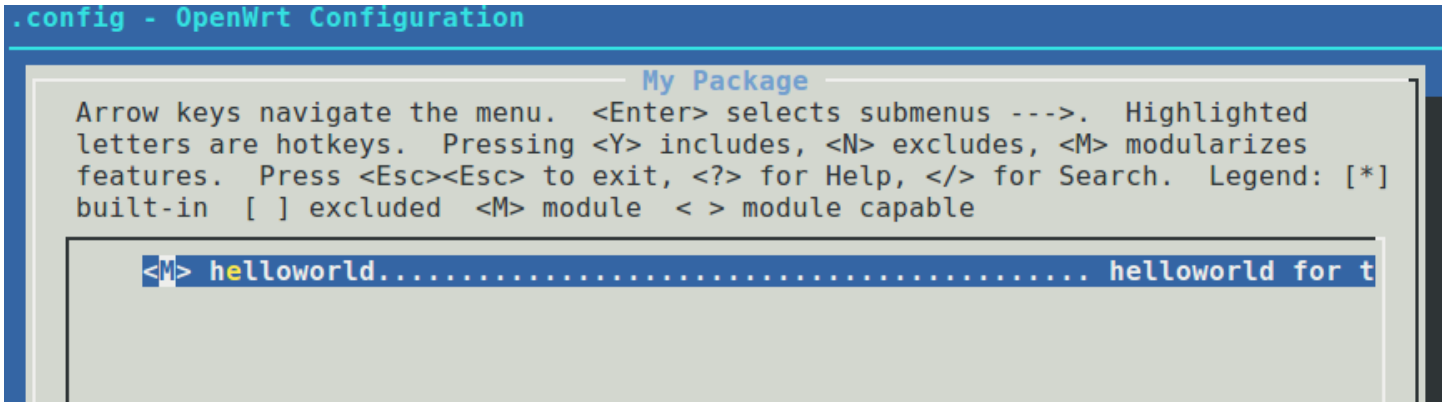
整个 helloworld 软件包的文件组织结构如下图所示：

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# tree package/helloworld/
package/helloworld/
├── Makefile
└── src
    ├── helloworld.c
    └── Makefile

1 directory, 3 files
```

下面开始配置 Openwrt，将添加的软件包 helloworld 选中

root@zjh-vm:/home/work/openwrt/barrier_breaker# make menuconfig

```
.config - OpenWrt Configuration
                    OpenWrt Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted
  letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
  built-in  [ ] excluded  <M> module  < > module capable
      ^(-)
      [*] Advanced configuration options (for developers)  --->
      [ ] Build the OpenWrt Image Builder
      [ ] Build the OpenWrt SDK
      [ ] Build the OpenWrt based Toolchain
      [ ] Image configuration  --->
          Package features  --->
          Base system  --->
          Administration  --->
          Boot Loaders  --->
          Development  --->
          Emulators  --->
          Firmware  --->
          Kernel modules  --->
          Languages  --->
          Libraries  --->
          LuCI  --->
          Mail  --->
          Multimedia  --->
          My Package  --->
          Network  --->
          Sound  --->
      v(+)
             <Select>    < Exit >    < Help >    < Save >    < Load >
```



这里选择为 M，表示安装成 ipk 模块，也可以选择 y 将其编译进固件。

现在可以单独编译该软件包，执行如下命令

root@zjh-vm:/home/work/openwrt/barrier_breaker# make package/helloworld/compile V=s

编译完成后，将在<BUILDROOT>/bin/ramips/packages/base/下生成 helloworld_1.0_ramips_24kec.ipk

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# ls bin/ramips/packages/base/myuci_1.0_ramips_24kec.ipk
bin/ramips/packages/base/myuci_1.0_ramips_24kec.ipk
```

将这个 ipk 文件拷贝到之前更新固件时用到的 EasyWebSvr 目录下，启动 EasyWebSvr.exe，然后在设备上执行如下命令下载这个文件

```
root@OpenWrt:~# wget http://192.168.10.104/myuci_1.0_ramips_24kec.ipk
Connecting to 192.168.10.104 (192.168.10.104:80)
myuci_1.0_ramips_24k 100% |*********************************|  1990   0:00:00 E
TA
```

然后执行如下命令安装 helloworld

```
root@OpenWrt:~# opkg install helloworld_1.0_ramips_24kec.ipk
Installing helloworld (1.0) to root...
before install helloworld
Configuring helloworld.
```

安装完成后可以将 helloworld_1.0_ramips_24kec.ipk 删除了。这里执行了在 Makefile 中定义的那段脚本代码

```
24 define Package/helloworld/preinst
25     #!/bin/sh
26     echo 'before install $(PKG_NAME)'
27 endef
```

通过 opkg 已经把 helloworld 安装到 bin 目录了，现在可以执行 helloworld 了

```
root@OpenWrt:~# helloworld
hello world:0
hello world:1
hello world:2
```

按 Ctrl+C 结束程序运行，执行 opkg remove helloworld 删除软件包 helloworld

```
root@OpenWrt:~# opkg remove helloworld
Removing package helloworld from root...
```

6.3 实例：添加内核模块

和“添加应用程序软件包”类似。
首先在<BUILDROOT>/package/kernel/下创建一个目录 helloworld-drv
root@zjh-vm:/home/work/openwrt/barrier_breaker# mkdir package/kernel/helloworld-drv

在<BUILDROOT>/package/kernel/helloworld-drv 下创建目录 src
root@zjh-vm:/home/work/openwrt/barrier_breaker# mkdir package/kernel/helloworld-drv/src

在<BUILDROOT>/package/kernel/helloworld-drv/src 下创建一个驱动源文件 helloworld.c 和一个 Makefile，其中 Makefile 内容如下

```
obj-m += helloworld.o
```

helloworld.c 内容如下

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

static int helloworld_init(void)
{
	printk("helloworld installed\n");
	return 0;
}

static void helloworld_exit(void)
{
	printk("helloworld uninstalled\n");
}

module_init(helloworld_init);
module_exit(helloworld_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("ZHAOJIANHUI");
```

在<BUILDROOT>/package/kernel/helloworld-drv 下创建一个 Makefile，其内容如下：

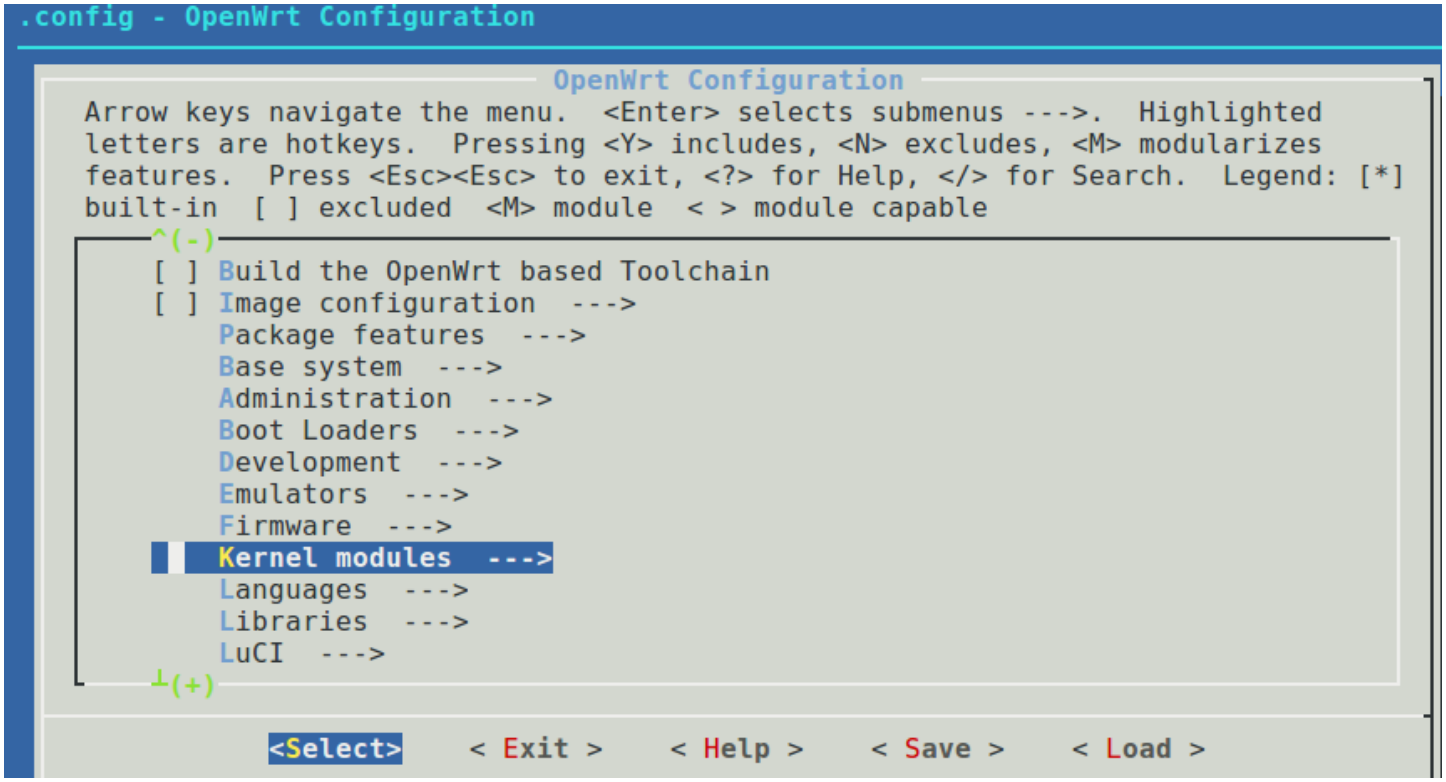
```
include $(TOPDIR)/rules.mk
include $(INCLUDE_DIR)/kernel.mk
include $(INCLUDE_DIR)/package.mk

PKG_NAME:=helloworld-drv
PKG_VERSION:=1.0
```

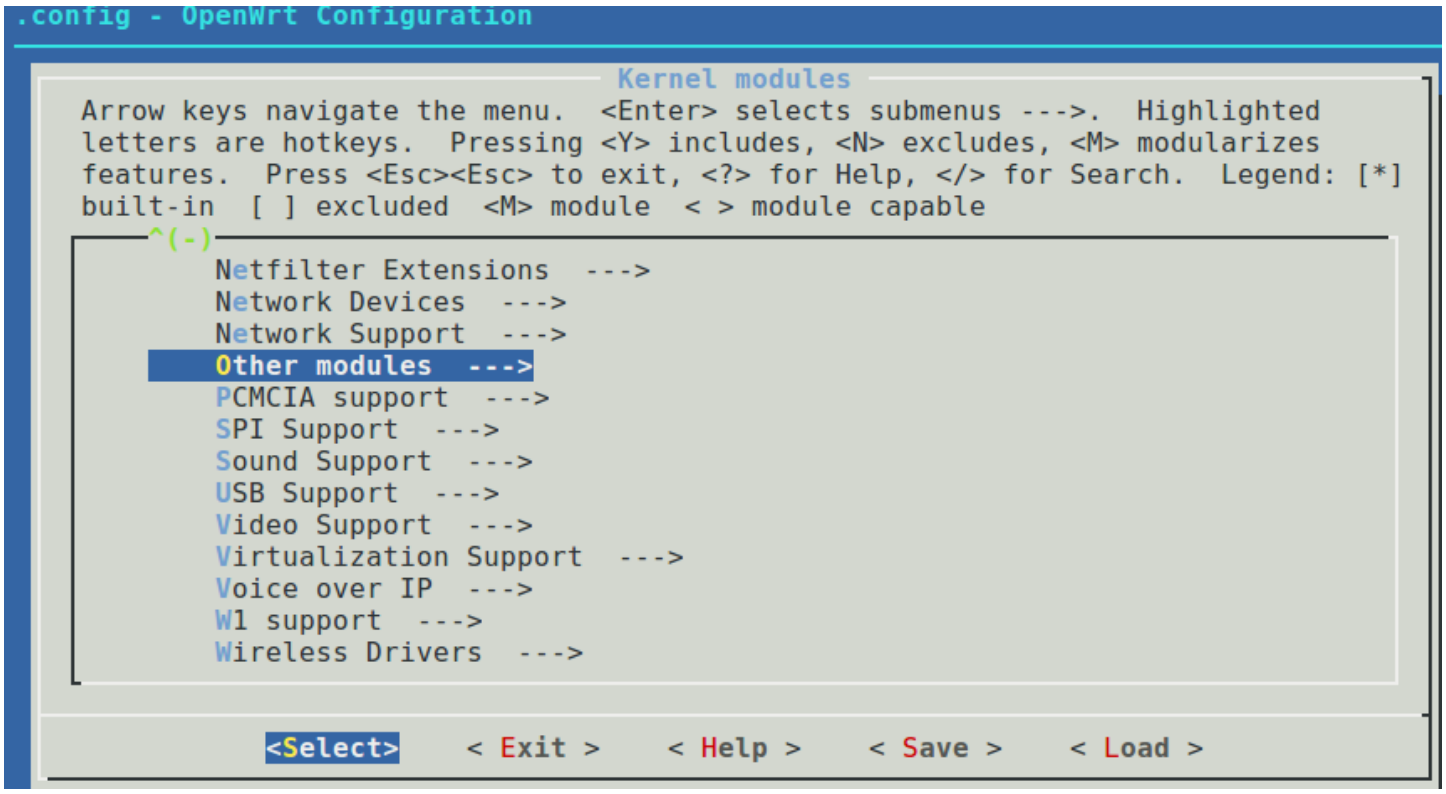
```
define KernelPackage/helloworld-drv
  SUBMENU:=Other modules
  TITLE:=helloworld-drv for test
  FILES:=$(PKG_BUILD_DIR)/helloworld.ko
  #自动加载，30 表示优先级（1 最优先），后面的 helloworld 表示模块名称
  #AUTOLOAD:=$(call AutoLoad,30,helloworld)
endef

define Build/Prepare
	mkdir -p $(PKG_BUILD_DIR)
	$(CP) ./src/* $(PKG_BUILD_DIR)/
endef

define Build/Compile
	$(MAKE) -C "$(LINUX_DIR)" \
		ARCH="$(LINUX_KARCH)" \
		CROSS_COMPILE="$(TARGET_CROSS)" \
		SUBDIRS="$(PKG_BUILD_DIR)" \
		EXTRA_CFLAGS="$(EXTRA_CFLAGS)" \
		$(EXTRA_KCONFIG) \
		modules
endef

$(eval $(call KernelPackage,helloworld-drv))
```

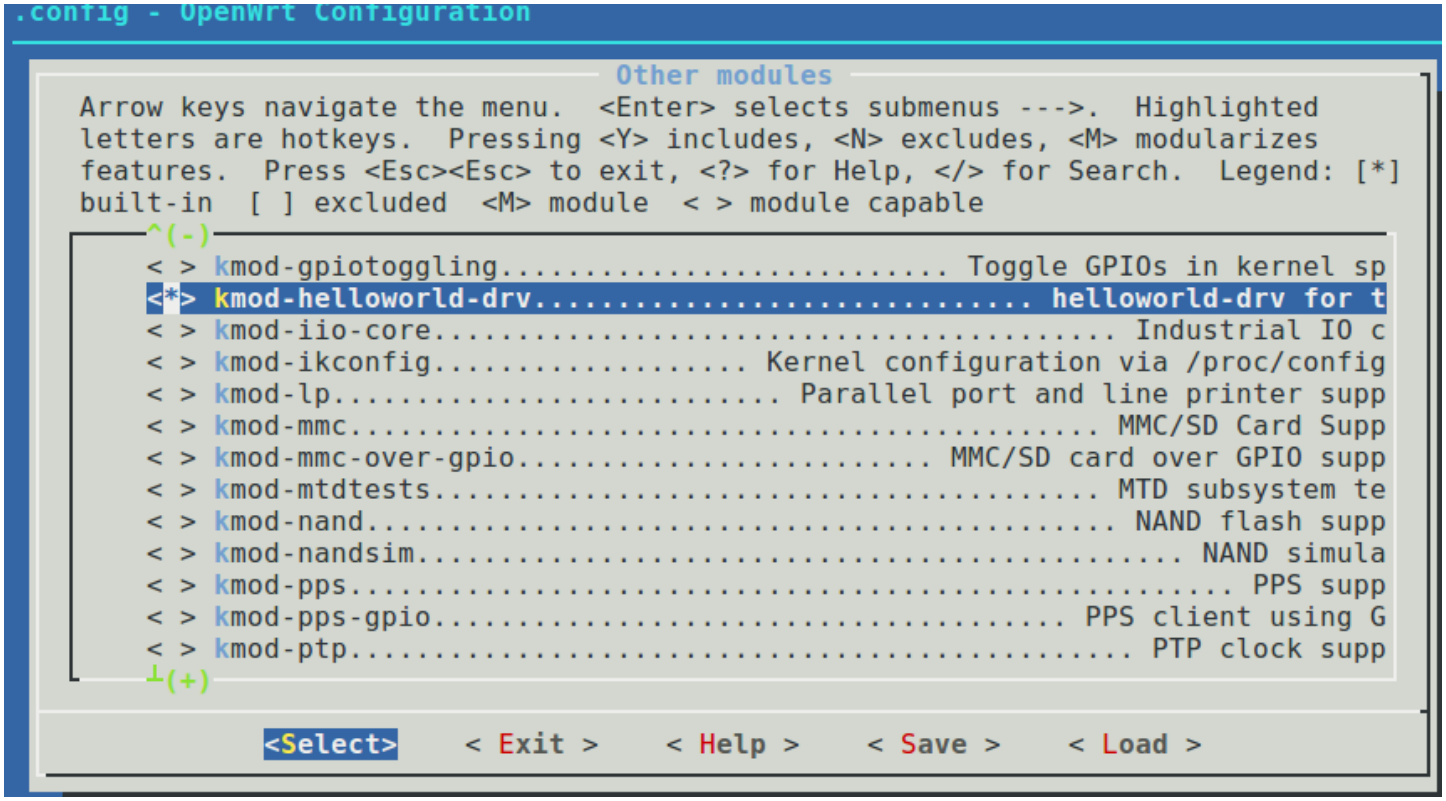
整个 helloworld-drv 软件包的文件组织结构如下所示：

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# tree package/kernel/helloworld-drv/
package/kernel/helloworld-drv/
├── Makefile
└── src
    ├── helloworld.c
    └── Makefile

1 directory, 3 files
```

配置 Openwrt，选中 helloworld-drv


.config - OpenWrt Configuration
OpenWrt Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes
features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*]
built-in [ ] excluded <M> module < > module capable
^(-)
[ ] Build the OpenWrt based Toolchain
[ ] Image configuration --->
Package features --->
Base system --->
Administration --->
Boot Loaders --->
Development --->
Emulators --->
Firmware --->
Kernel modules --->
Languages --->
Libraries --->
LuCI --->
v(+)
<Select> < Exit > < Help > < Save > < Load >


.config - OpenWrt Configuration
Kernel modules
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted
letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes
features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*]
built-in [ ] excluded <M> module < > module capable
^(-)
Netfilter Extensions --->
Network Devices --->
Network Support --->
Other modules --->
PCMCIA support --->
SPI Support --->
Sound Support --->
USB Support --->
Video Support --->
Virtualization Support --->
Voice over IP --->
W1 support --->
Wireless Drivers --->
<Select> < Exit > < Help > < Save > < Load >



单独编译 helloworld-drv

root@zjh-vm:/home/work/openwrt/barrier_breaker# make package/kernel/helloworld-drv/compile V=s

编译成功后，将会在<BUILDROOT>/bin/ramips/packages/base/下生成文件

kmod-helloworld-drv_3.10.49+1.0-1_ramips_24kec.ipk

将这个 ipk 文件下载到设备，并通过 opkg 安装，然后使用 modprobe 加载模块

```
root@OpenWrt:~# wget http://192.168.10.104/kmod-helloworld-drv_3.10.49+1.0-1_ram
ips_24kec.ipk
Connecting to 192.168.10.104 (192.168.10.104:80)
kmod-helloworld-drv_ 100% |*********************************|  1213   0:00:00 ETA
root@OpenWrt:~# opkg install kmod-helloworld-drv_3.10.49\+1.0-1_ramips_24kec.ipk

Installing kmod-helloworld-drv (3.10.49+1.0-1) to root...
Configuring kmod-helloworld-drv.
root@OpenWrt:~# modprobe helloworld
[  236.430000] helloworld installed
```

执行 opkg remove kmod-helloworld-drv 卸载软件包 kmod-helloworld-drv

```
root@OpenWrt:~# opkg remove kmod-helloworld-drv
Removing package kmod-helloworld-drv from root...
```

如果将#AUTOLOAD:=$(call AutoLoad,30,helloworld)前面的#去掉，然后再次编译该软件包，下载到设备并安装

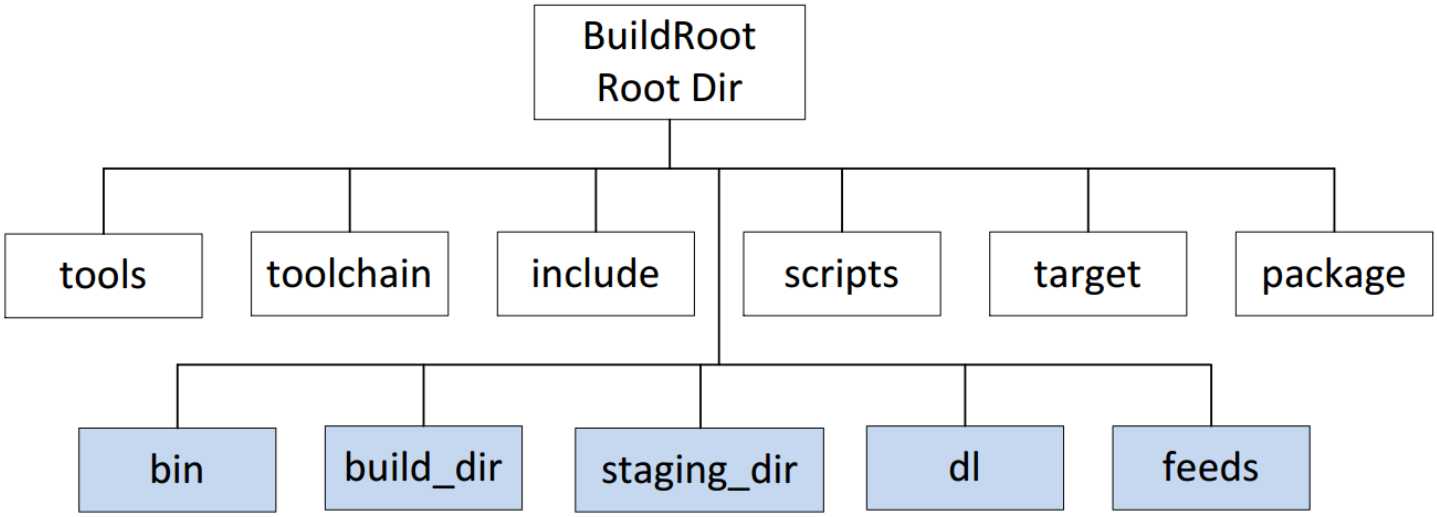
```
root@OpenWrt:~# wget http://192.168.10.104/kmod-helloworld-drv_3.10.49+1.0-1_ram
ips_24kec.ipk
Connecting to 192.168.10.104 (192.168.10.104:80)
kmod-helloworld-drv_ 100% |*********************************|  1379   0:00:00 ETA
root@OpenWrt:~# opkg install kmod-helloworld-drv_3.10.49+1.0-1_ramips_24kec.ipk
Installing kmod-helloworld-drv (3.10.49+1.0-1) to root...
Configuring kmod-helloworld-drv.
[  218.030000] helloworld installed
```

可以看到，安装后，模块自动加载了，重启系统如下图所示，可以看到模块随系统自动加载了。

```
- config restore -
procd: - early -
procd: - watchdog -
procd: - ubus -
procd: - init -
Please press Enter to activate this console.
[   52.810000] NET: Registered protocol family 10
[   52.820000] helloworld installed
[   52.860000] nf_conntrack version 0.5.0 (460 buckets, 1840
[   52.890000] ip6_tables: (C) 2000-2006 Netfilter Core Team
[   52.960000] u32 classifier
[   52.970000]     input device check on
[   52.980000]     Actions configured
[   53.000000] Mirror/redirect action on
[   53.050000] Loading modules backported from Linux version
2-0-gf2032ea
```

7 Openwrt 源码树目录组织结构

Openwrt Buildroot 是一个集合了 Makefile，patches 和 scripts 的一个系统构建环境。它能制作交叉编译工具链，下载 Linux 内核，制作根文件系统，管理第三方软件包。开发人员可以使用它来编译出自定义的固件 image 来支持自己的硬件设备。在 Openwrt Buildroot 源码树中没有任何 Linux 内核和第三方源码包。这些 Makefile 决定下载哪个版本的 Linux 内核以及哪个版本的源码包来编译固件。下图是表示了 Openwrt Buildroot 源码树组织结构



第一行目录是 Openwrt Buildroot 源码树原始的目录，第二行是编译过程中产生的。

- tools：包含了一些 Makefile。编译固件需要一些命令，比如 mkimage、lzma 等，而这些 Makefile 则定义了如何获得这些命令的源码包，以及如何编译/安装这些命令。
- toolchain：包含了一些 Makefile。这些 Makefile 定义了如何获得 kernel headers, C library, bin-utils, compiler, debugger 以及如何编译/安装它们。这些源码是用来制作交叉编译工具链的。如果你要添加一个全新的架构，你需要在这里添加一个配置。
- target：各个硬件平台在这里面定义了编译固件和内核的步骤、方法。
- package：包含了一些针对各个软件包的 Makefile 以及补丁。这些 Makefile 定义了如何获取这些软件包以及如何编译/安装。
- include：存放了一些公用的 Makefile，这些 Makefile 会被其它 Makefile 包含。

- scripts：一些用于 Openwrt 软件包管理的 perl 脚本。
- dl：存放从网上下载的用户空间软件包的源码包
- build_dir：所有软件包都解压到该目录，并在该目录下编译。
- staging_dir：最终安装目录，包括运行在主机的工具、交叉编译工具链、嵌入式根文件系统 rootfs
- bin：编译完成后，最终生成的固件 image 以及软件包的 ipk 文件都会放在这里。

8 Openwrt Buildroot 工作过程概述

一旦 Openwrt Buildroot 经过了正确的配置，比如目标平台和架构都已经指定，用户空间软件包已选择，Openwrt Buildroot 将会通过下面 6 个步骤来产生固件 image：

1. 下载交叉编译工具，比如内核头文件；
2. 创建阶段性目录（staging_dir/）。交叉编译工具链将会被安装到这里（比如 toolchain-mipsel_24kec+dsp_gcc-4.8-linaro_uClibc-0.9.33.2/）。如果你要使用这个交叉编译工具链编译特定的程序，你可以在这个目录找到交叉编译工具。嵌入式根文件系统也会安装到 staging_dir/目录，比如 target-mipsel_24kec+dsp_uClibc-0.9.33.2/root-ramips；
3. 创建下载目录（默认为 dl/）。所有源码包会被下载到这个目录；
4. 创建编译目录（build_dir/）。所有用户空间工具会在这里编译；
5. 创建目标根文件系统目录（比如 build_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/root-ramips/）以及根文件系统框架。
6. 安装用户空间软件包到根文件系统以及压缩整个文件系统为适当的格式。

9 启动脚本（Init Scripts）

9.1 概述

参考 Openwrt 官方文档：http://wiki.openwrt.org/doc/techref/initscripts

启动脚本存放在/etc/init.d 目录，首先创建一个启动脚本/etc/init.d/example，如下所示：

```
#!/bin/sh /etc/rc.common

START=10
STOP=15

start() {
        echo 'start example'
}

stop() {
        echo 'stop example'
}
```

然后给/etc/init.d/example 添加可执行权限

root@OpenWrt:~# chmod a+x /etc/init.d/example

执行 example

```
root@OpenWrt:~# /etc/init.d/example
Syntax: /etc/init.d/example [command]

Available commands:
        start   Start the service
        stop    Stop the service
        restart Restart the service
        reload  Reload configuration files (or restart if that fails)
        enable  Enable service autostart
        disable Disable service autostart
```

不跟参数执行 example 时，程序提示需要一个参数，并列出了所有可用的参数。

下面跟上参数再次执行 example

```
root@OpenWrt:~# /etc/init.d/example start
start example
root@OpenWrt:~# /etc/init.d/example stop
stop example
root@OpenWrt:~# /etc/init.d/example restart
stop example
start example
```

可以看到，跟参数 start 时，程序执行了 start 函数，跟参数 stop 时，程序执行了 stop 函数，跟参数 restart 时，程序先执行 stop 函数，再执行 start 函数。

其实，在/etc/rc.common 中已经定义了所有默认的函数，当我们没有重新实现这些函数时，程序会执行/etc/rc.common 中的默认函数，比如在这个 example 中，我们没有定义 restart 函数，我们跟参数 restart 时，程序执行/etc/rc.common 中的 restart 函数，该函数首先调用 stop 函数，再调用 start 函数。而一旦我们重新实现了这些函数，在执行时，程序就会调用我们自己实现的函数。

脚本中的 START= 和 STOP= 分别决定了该脚本会在系统启动过程和系统关机过程的哪个点执行，数字越小越先执行。当我们以 enable 参数调用启动脚本时，系统会创建一个该脚本文件的符号链接（例如 S10example、K15example），放在/etc/rc.d/目录下，而当以 disable 参数调用脚本时，系统会删除该符号链接。

- START=10：意味着该脚本文件会被/etc/rc.d/S10example 链接。也就是说，该脚本会在 START=10 及之下这样的脚本后面执行，而在 START=11 及之上这样的脚本前面执行。
- STOP=15：意味着该脚本文件会被/etc/rc.d/K15example 链接。也就是说，该脚本会在 STOP=14 及之下这样的脚本后面执行，而在 STOP=16 及之上这样的脚本前面执行。

```
root@OpenWrt:/# /etc/init.d/example enable
root@OpenWrt:/# ls -l /etc/rc.d/*example
lrwxrwxrwx    1 root     root            17 Jan 27 13:38 /etc/rc.d/K15example ->
../init.d/example
lrwxrwxrwx    1 root     root            17 Jan 27 13:38 /etc/rc.d/S10example ->
../init.d/example
```

添加自己的命令

```
#!/bin/sh /etc/rc.common

EXTRA_COMMANDS="custom"
EXTRA_HELP="            custom  Help for the custom command"
```

```
custom() {
        echo 'custom command'
}
```

```
root@OpenWrt:/# ./etc/init.d/example
Syntax: ./etc/init.d/example [command]

Available commands:
        start   Start the service
        stop    Stop the service
        restart Restart the service
        reload  Reload configuration files (or restart if that fails)
        enable  Enable service autostart
        disable Disable service autostart
        custom  Help for the custom command
root@OpenWrt:/# ./etc/init.d/example custom
custom command
```

9.2 实例：实现 6.2 节的 helloworld 开机自启动

完善 6.2 节的 helloworld 软件包，使其支持开机自启动。

首先在 6.2 节的基础上，在<BUILDROOT>/package/helloworld/目录下创建一个目录 files/，用来存放启动脚本

root@zjh-vm:/home/work/openwrt/barrier_breaker# mkdir package/helloworld/files

然后在<BUILDROOT>/package/helloworld/files/目录下创建启动脚本文件 helloworld，其内容如下：

```
#!/bin/sh /etc/rc.common

START=100
STOP=10

start() {
        /bin/helloworld &
}

stop() {
        killall helloworld
}
```

然后修改<BUILDROOT>/package/helloworld/Makefile，修改后的 Makefile 如下所示：

```
include $(TOPDIR)/rules.mk
include $(INCLUDE_DIR)/package.mk

PKG_NAME:=helloworld
PKG_VERSION:=1.0

define Package/$(PKG_NAME)
  CATEGORY:=My Package
  #DEPENDS:= 如果需要依赖其他软件包，在这里添加
  TITLE:=helloworld for test
  MAINTAINER:=zjh <809205580@qq.com>
```

```
endef

define Build/Prepare
	mkdir -p $(PKG_BUILD_DIR)
	$(CP) ./src/* $(PKG_BUILD_DIR)/
endef

define Package/$(PKG_NAME)/install
	$(INSTALL_DIR) $(1)/bin $(1)/etc/init.d/
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/helloworld $(1)/bin/
	$(INSTALL_BIN) ./files/helloworld $(1)/etc/init.d/
endef

define Package/$(PKG_NAME)/postinst
	#!/bin/sh
	# check if we are on real system
	if [ -z "$${IPKG_INSTROOT}" ]; then
		echo "Enabling rc.d symlink for helloworld"
		/etc/init.d/helloworld enable
	fi
	exit 0
endef

define Package/$(PKG_NAME)/prerm
	#!/bin/sh
	# check if we are on real system
	if [ -z "$${IPKG_INSTROOT}" ]; then
		echo "Removing rc.d symlink for helloworld"
		/etc/init.d/helloworld disable
	fi
	exit 0
endef

$(eval $(call BuildPackage,helloworld))
```

主要的改动，已经用红色标注。

$IPKG_INSTROOT 这个变量如果为空，则表示在目标设备运行，否则在主机运行。

还是将 helloworld 软件包配置成 ipk 模块

```
<M> helloworld..... helloworld for test
```

单独编译它

root@zjh-vm:/home/work/openwrt/barrier_breaker# make package/helloworld/compile V=s -j 3

将生成的 helloworld_1.0_ramips_24kec.ipk 下载到目标设备并通过 opkg 安装

```
root@OpenWrt:/# wget http://192.168.10.104/helloworld_1.0_ramips_24kec.ipk
Connecting to 192.168.10.104 (192.168.10.104:80)
helloworld_1.0_ramip 100% |*********************************|  2134   0:00:00 ETA
root@OpenWrt:/# opkg install helloworld_1.0_ramips_24kec.ipk
Installing helloworld (1.0) to root...
Configuring helloworld.
Enabling rc.d symlink for helloworld
root@OpenWrt:/# ls /etc/init.d/helloworld
/etc/init.d/helloworld
root@OpenWrt:/# ls /etc/rc.d/*helloworld*
/etc/rc.d/K10helloworld   /etc/rc.d/S100helloworld
```

通过 opkg 安装完成后，已成功创建启动脚本。现在执行 reboot 重启系统，通过 ps 命令查看系统进程

```
root@OpenWrt:/# ps | grep "helloworld"
  432 root       768 S    /bin/helloworld
 1485 root      1480 S    grep helloworld
```

helloworld 已经自动启动了。

通过 opkg 卸载 helloworld

```
root@OpenWrt:/# opkg remove helloworld
Removing package helloworld from root...
Removing rc.d symlink for helloworld
```

下面将 helloworld 这个软件包直接编译进固件

```
<*> helloworld..................................... helloworld for test
```

然后编译整个 Openwrt 固件

root@zjh-vm:/home/work/openwrt/barrier_breaker# make V=s -j 3

更新固件重启系统

```
BusyBox v1.22.1 (2015-01-18 18:36:51 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 BARRIER BREAKER (Barrier Breaker, r44065)
 -----------------------------------------------------
  * 1/2 oz Galliano             Pour all ingredients into
  * 4 oz cold Coffee            an irish coffee mug filled
  * 1 1/2 oz Dark Rum           with crushed ice. Stir.
  * 2 tsp. Creme de Cacao
 -----------------------------------------------------
root@OpenWrt:/# ps | grep "helloworld"
  447 root       768 S    /bin/helloworld
 1550 root      1480 S    grep helloworld
```

同样成功启动 helloworld。

10 通过 shell 脚本操作 UCI 配置

Openwrt 提供了一些 shell 脚本函数，这些函数使得操作 UCI 配置文件变得非常的高效。要使用这些 shell 函数，首先需要加载/lib/functions.sh，然后实现 config_cb(), option_cb(), 和 list_cb()这些 shell 函数，当我们调用 Openwrt 提供的 shell 函数 config_load 来解析配置文件时，程序就会回调我们自己定义的 config_cb(), option_cb(), 和 list_cb()这些 shell 函数。下面创建一个 shell 脚本 test.sh，其内容如下：

```
#!/bin/sh
. /lib/functions.sh

reset_cb

config_cb() {
        local type="$1"
        local name="$2"
        # commands to be run for every section
        if [ -n "$type" ];then
                echo "$name=$type"
        fi
}

option_cb() {
        local name="$1"
        local value="$2"
        # commands to be run for every option
        echo $name=$value
}

list_cb() {
        local name="$1"
        local value="$2"
        # commands to be run for every list item
        echo $name=$value
}

config_load $1
```

reset_cb 表示将 xxx_cb 这 3 个函数初始化为没有任何操作。config_load 首先从绝对路径文件名加载配置，如果失败再从/etc/config 路径加载配置文件。xxx_cb 那 3 个函数必须在包含/lib/functions.sh 和调用 config_load 之间定义。

给 test.sh 添加可执行权限

```
root@OpenWrt:/# chmod a+x test.sh
```

执行 test.sh

```
root@OpenWrt:/# ./test.sh dhcp
cfg02411c=dnsmasq
domainneeded=1
```

```
boguspriv=1
filterwin2k=0
localise_queries=1
rebind_protection=1
rebind_localhost=1
local=/lan/
domain=lan
expandhosts=1
nonegcache=0
authoritative=1
readethers=1
leasefile=/tmp/dhcp.leases
resolvfile=/tmp/resolv.conf.auto
lan=dhcp
interface=lan
start=100
limit=150
leasetime=12h
wan=dhcp
interface=wan
ignore=1
cfg06fe63=host
ip=192.168.10.104
mac=44:8A:5B:EC:49:27
```

可以看到，当 config_load 解析配置文件时，解析到 section 时，就会回调 config_cb，解析 option 时，就会回调 option_cb，解析 list 时，就会回调 list_cb。我们可以在对应的函数里面添加自己的相关命令。

另一种方式是使用 config_foreach 针对每一个 section 调用一个指定的自定义的函数，看下面这个 shell 脚本 test.sh

```
#!/bin/sh
. /lib/functions.sh

reset_cb

handle_interface() {
        local config="$1"
        local custom="$2"
        # run commands for every interface section
        if [ "$config" = "$custom" ];then
                echo $custom
        fi
}

config_load $1
config_foreach handle_interface interface $2
```

这里定义了一个函数 handle_interface，然后以 handle_interface、interface 和$2 这 3 个参数调用 config_foreach。config_foreach 会遍历以$1 指定的配置文件中的所有的 section，一旦其类型与参数 interface 相等时，则回调 handle_interface，在 config_foreach 函数内部，会以当前正在解析的 section 的 ID 和调用 config_foreach 时的第 3 个参数作为调用 handle_interface 的参数。

```
root@OpenWrt:/# ./test.sh network wan
wan
```

这里以参数 network 和 wan 调用 test.sh，然后程序首先调用 config_load 加载配置文件 network，然后 config_foreach 在配置文件 network 中搜寻类型为 interface 的 section，一旦找到就以当前 section 的 ID 和 wan 作为参数调用 handle_interface，在 handle_interface 函数中，得到 config=wan，custom=wan，最终打印出 wan。

使用 config_get 读取配置选项的值，该函数需要至少 3 个参数：

- 一个存储返回值的变量
- 要读取的 section 的 ID（如果是有名称的 section 就是其名称）
- 要读取的 option 的名称

在 handle_interface 中使用 config_get 和 config_set 来读取和设置当前的 section。

```
#!/bin/sh
. /lib/functions.sh

reset_cb

handle_interface() {
	local config="$1"
	local custom="$2"
	# run commands for every interface section
	if [ "$config" = "$custom" ];then
		config_get prot $config proto
		echo $prot
	fi
}

config_load $1
config_foreach handle_interface interface $2
```

```
root@OpenWrt:/# ./test.sh network lan
static
root@OpenWrt:/# ./test.sh network wan
dhcp
```

如果明确知道 section 的名称，可以直接调用 config_get 读取配置选项的值。

config_set 用来设置配置选项的值，它需要 3 个参数：

- section 的 ID（如果有名称就是其名称）
- option 的名称
- 要设置的值

更多操作参考 Openwrt 官方文档：http://wiki.openwrt.org/doc/devel/config-scripting

11 Openwrt 启动流程

11.1 Openwrt 固件生成过程（基于 MPR-A2 硬件平台）

（1） 编译 Linux 内核生成 vmlinux，并将其拷贝为 vmlinux-mpr-a2

build_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/linux-ramips_rt305x/vmlinux-mpr-a2

（2） 使用内核自带的工具 dtc 将 target/linux/ramips/dts/MPRA2.dts 编译成 dtb 格式的 MPRA2.dtb

build_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/linux-ramips_rt305x/MPRA2.dtb

（3） 使用 Openwrt 提供的工具 patch-dtb 将上面生成的 MPRA2.dtb 填充到 vmlinux-mpr-a2

（4） 使用 lzma 压缩 vmlinux-mpr-a2，生成 vmlinux-mpr-a2.bin.lzma

build_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/linux-ramips_rt305x/vmlinux-mpr-a2.bin.lzma

（5） 使用 u-boot 提供的工具 mkimage 将 vmlinux-mpr-a2.bin.lzma 制作成 vmlinux-mpr-a2.uImage

build_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/linux-ramips_rt305x/vmlinux-mpr-a2.uImage

（6） 使用 mksquashfs4 将根文件系统 build_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/root-ramips/制作成 root.squashfs

build_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/linux-ramips_rt305x/root.squashfs

（7） 使用 dd 命令将 root.squashfs 写入 openwrt-ramips-rt305x-root.squashfs

bin/openwrt-ramips-rt305x-root.squashfs

（8） 使用 cat 命令读取 vmlinux-mpr-a2.uImage 和 root.squashfs，将它们依次写入文件 openwrt-ramips-rt305x-mpr-a2-squashfs-sysupgrade.bin

11.2 Openwrt 启动流程：procd

首先 u-boot 从 Flash 中读取 Linux 内核到内存，然后跳转到内存执行 Linux 内核。Linux 内核进行一些列验证，注册相关驱动，根据分区表（target/linux/ramips/dts/MPRA2.dts）创建分区。然后挂载根文件系统。然后启动第一个用户空间进程。原生的 Linux 内核默认启动的第一个用户空间进程是/sbin/init,Openwrt 将其修改为默认启动的第一个用户空间进程为/etc/preinit。参考内核代码 linux-3.10.49/init/main.c

```
static int __ref kernel_init(void *unused)
{
	kernel_init_freeable();
	/* need to finish all async __init code before freeing the memory */
	async_synchronize_full();
	free_initmem();
	mark_rodata_ro();
	system_state = SYSTEM_RUNNING;
	numa_default_policy();
```

```
    flush_delayed_fput();

    if (ramdisk_execute_command) {
        if (!run_init_process(ramdisk_execute_command))
            return 0;
        pr_err("Failed to execute %s\n", ramdisk_execute_command);
    }

    /*
     * We try each of these until one succeeds.
     *
     * The Bourne shell can be used instead of init if we are
     * trying to recover a really broken machine.
     */
    if (execute_command) {
        if (!run_init_process(execute_command))
            return 0;
        pr_err("Failed to execute %s.    Attempting defaults...\n",
            execute_command);
    }

    if (!run_init_process("/etc/preinit") ||
        !run_init_process("/sbin/init") ||
        !run_init_process("/etc/init") ||
        !run_init_process("/bin/init") ||
        !run_init_process("/bin/sh"))
        return 0;

    panic("No init found.    Try passing init= option to kernel. "
          "See Linux Documentation/init.txt for guidance.");
}
```

/etc/preinit 就是一个 shell 脚本，其第一行内容如下：

```
[ -z "$PREINIT" ] && exec /sbin/init
```

一开始$PREINIT 为空，因此首先执行/sbin/init，该程序来自于 Openwrt 软件包 package/system/procd 中的 init.c，其 main 函数如下所示：

```
int
main(int argc, char **argv)
{
    pid_t pid;
    ……
    early(); //调用 early_mounts 挂载 proc、sysfs、tmpfs，调用 early_console 打开 3 个文件描述符（0、1、2）
    cmdline(); //解析内核参数（init_debug=([0-9]），设置提示级别
    watchdog_init(1); //初始化看门狗
    ……
    //创建 2 个子进程：分别执行/sbin/procd -h /etc/hotplug-preinit.json 和/bin/sh /etc/preinit
```

```
    //当子进程/bin/sh /etc/preinit 退出时，调用函数 spawn_procd，该函数通过 execvp 执行/sbin/procd 替换自己
    //最终/sbin/procd 成为用户空间的第一个进程（PID=1）
    preinit();
    ……
    return 0;
}
```

Openwrt 软件包 package/system/procd 中的 prod.c 中的 main 函数如下：

```
int main(int argc, char **argv)
{
    ……
    //未带参数执行/sbin/procd 直接执行到这里
    procd_state_next(); //  该函数调用 state_enter
    ……
}
```

state_enter 函数根据不同的状态，执行相应的操作

```
static void state_enter(void)
{
    char ubus_cmd[] = "/sbin/ubusd";

    switch (state) {
    case STATE_EARLY:
        LOG("- early -\n");
        watchdog_init(0);
        hotplug("/etc/hotplug.json");
        procd_coldplug();
        break;

    case STATE_INIT:
        // try to reopen incase the wdt was not available before coldplug
        watchdog_init(0);
        LOG("- ubus -\n");
        procd_connect_ubus();

        LOG("- init -\n");
        service_init();
        service_start_early("ubus", ubus_cmd);

        procd_inittab(); //解析/etc/inittab
        procd_inittab_run("respawn");
        procd_inittab_run("askconsole");
        procd_inittab_run("askfirst");
        procd_inittab_run("sysinit"); //  以参数"boot"执行/etc/rc.d/下的所有 S 开头的脚本
        break;
```

```
        case STATE_RUNNING:
                LOG("- init complete -\n");
                break;

        case STATE_SHUTDOWN:
                LOG("- shutdown -\n");
                procd_inittab_run("shutdown"); //  以参数"shutdown"执行/etc/rc.d/下的所有 K 开头的脚本
                sync();
                break;

        case STATE_HALT:
                LOG("- reboot -\n");
                reboot(reboot_event);
                break;

        default:
                ERROR("Unhandled state %d\n", state);
                return;
        };
}
```

11.3 Openwrt 启动流程：/etc/preinit

/etc/preinit 是一系列脚本的入口，这一系列脚本保存在/lib/preinit/目录下。

```
02_default_set_state
03_preinit_do_ramips.sh
04_handle_checksumming
07_set_preinit_iface_ramips
10_indicate_failsafe
10_indicate_preinit
30_failsafe_wait
40_run_failsafe_hook
50_indicate_regular_preinit
70_initramfs_test
80_mount_root
99_10_failsafe_login
99_10_run_init
```

/etc/preinit 将/lib/preinit/目录下的这些列脚本分为如下 5 类。

preinit_essential

preinit_main

failsafe

initramfs

preinit_mount_root

目前/lib/preinit/下的所有脚本只实现了 preinit_main 和 failsafe 这两类，如下所示：

preinit_main：　define_default_set_state（/lib/preinit/02_default_set_state）
do_ramips（/lib/preinit/03_preinit_do_ramips.sh）
do_checksumming_disable（04_handle_checksumming）
ramips_set_preinit_iface（/lib/preinit/07_set_preinit_iface_ramips）
preinit_ip（/lib/preinit/10_indicate_preinit）
pi_indicate_preinit（/lib/preinit/10_indicate_preinit）
failsafe_wait（/lib/preinit/30_failsafe_wait）
run_failsafe_hook（/lib/preinit/40_run_failsafe_hook）
indicate_regular_preinit（/lib/preinit/50_indicate_regular_preinit）
initramfs_test（/lib/preinit/70_initramfs_test）
do_mount_root（/lib/preinit/80_mount_root）
run_init（/lib/preinit/99_10_run_init）
failsafe：　indicate_failsafe（/lib/preinit/10_indicate_failsafe）
failsafe_netlogin（/lib/preinit/99_10_failsafe_login）
failsafe_shell（/lib/preinit/99_10_failsafe_login）

在/etc/preinit 中调用 boot_run_hook preinit_essential 和 boot_run_hook preinit_main 分别执行 preinit_essential 和 preinit_main 这两类函数。

- define_default_set_state（/lib/preinit/02_default_set_state）

  加载/etc/diag.sh 中的两个状态操作函数：get_status_led 和 set_state

- do_ramips do_ramips（/lib/preinit/03_preinit_do_ramips.sh）

  调用/lib/ramips.sh 中的 ramips_board_detect 函数探测板子名称，将其保存在

  /tmp/sysinfo/board_name 和/tmp/sysinfo/model 这 2 个文件中，我们可以通过 cat 查看

  root@OpenWrt:/lib/preinit# cat /tmp/sysinfo/board_name
  mpr-a2
  root@OpenWrt:/lib/preinit# cat /tmp/sysinfo/model
  HAME MPR-A2

- ramips_set_preinit_iface（/lib/preinit/07_set_preinit_iface_ramips）

  初始化网络接口，设置 switch。针对 RT5350 的设置，为了避免进入 Failsafe 模式 TCP 连接超时。

- preinit_ip（/lib/preinit/10_indicate_preinit）

  预初始化 IP 地址

- pi_indicate_preinit（/lib/preinit/10_indicate_preinit）

  设置 LED，指示进入 preinit 过程

- failsafe_wait（/lib/preinit/30_failsafe_wait）

  根据内核参数/proc/cmdline 或者用户是否按下相应按键，决定是否进入 failsafe 模式（故障恢复模式）。如果内核参数含有 FAILSAFE=true，则表示直接进入 failsafe 模式，否则调用 fs_wait_for_key 等待用户终端按键（默认为'F'+Enter），等待时间为 fs_failsafe_wait_timeout（在/etc/preinit 中设置，默认为 2s），如果在等待时间内用户按下指定的按键，则表示要进入 failsafe 模式。接着再检测文件 /tmp/failsafe_button 是否存在，如果存在则表示要进入 failsafe 模式（该文件可通过设备上的实体按

键来创建）。如果要进入 failsafe 模式，则设置全局变量 FAILSAFE=true

- run_failsafe_hook（/lib/preinit/40_run_failsafe_hook）

  如果全局变量 FAILSAFE=true，则执行 failsafe 这类函数。

- indicate_regular_preinit（/lib/preinit/50_indicate_regular_preinit）

  设置 LED，指示进入正常模式

- do_mount_root（/lib/preinit/80_mount_root）

  执行/sbin/mount_root 挂载根文件系统，该命令来自于 Openwrt 软件包 package/system/fstools。

- indicate_failsafe（/lib/preinit/10_indicate_failsafe）

  设置 LED,指示进入 failsafe 模式

- failsafe_netlogin（/lib/preinit/99_10_failsafe_login）

  启动 Telnet 服务器（telnetd）

- failsafe_shell（/lib/preinit/99_10_failsafe_login）

  启动 shel

11.4 Openwrt 启动流程：/etc/rc.d/S*

在 11.2 节，提到过，在 procd 执行/etc/rc.d/S*时，其参数为"boot"（例如：/etc/rc.d/S00sysfixtime boot），这样就会执行每个脚本里面的 boot 函数，也肯能是间接执行 stat 函数。/etc/rc.d/下的所有脚本都是链接到/etc/init.d/下的脚本。下面分析几个重要的脚本。

- S10boot

  调用 uci_apply_defaults 执行第 1 次开机时的 UCI 配置初始化工作。该函数执行/etc/uci-defaults/下的所有脚本，执行成功后就删除。因此该目录下的脚本只有第一次开机才会执行。

- S10system

  根据 UCI 配置文件/etc/config/system 配置系统，具体可参考该配置文件。

- S11sysctl

  根据/etc/sysctl.conf 设置系统配置（[-f /etc/sysctl.conf] && sysctl -p -e >&-）。

- S19firewall

  启动防火墙 fw3。该工具来自 Openwrt 软件包 package/network/config/firewall

- S20network

  根据 UCI 配置文件/etc/config/network，使用守护进程/sbin/netifd 来配置网络。

12 Failsafe 模式（故障恢复模式）

当系统出现故障时，比如忘记密码，忘记 IP 等，就可以通过进入 Failsafe 模式来修复系统。在 failsafe 模式下，系统会启动 Telnet 服务器，我们可以无需密码通过 Telnet 登录到路由器。在 failsafe 模式下，系统一般会关闭 VLAN，给 eth0 设置默认 IP 为 192.168.1.1（在/etc/preinit 中默认设置）。RT5350 例外，该 SOC 有一个 bug，需要打开 VLAN 才能使用 TCP 协议。

在 11.3 节已经分析了如何进入 Failsafe 模式。下面是系统启动时输出的日志：

```
Press the [f] key and hit [enter] to enter failsafe mode
Press the [1], [2], [3] or [4] key and hit [enter] to select the debug level
```

提示我们按下 F 键和 Enter 键进入 failsafe 模式。

针对 RT5350，系统启动时，首先打开 VLAN，并设置 VLAN 1 的端口为 0，如下所示：

/etc/preinit/07_set_preinit_iface_ramips

```
ramips_set_preinit_iface() {
	RT3X5X=`cat /proc/cpuinfo | egrep "(RT3.5|RT5350)"`
	if [ -n "${RT3X5X}" ]; then
		swconfig dev rt305x set reset 1
	fi

	if echo $RT3X5X | grep -q RT5350; then
		# This is a dirty hack to get by while the switch
		# problem is investigated. When VLAN is disabled, ICMP
		# pings work as expected, but TCP connections time
		# out, so telnetting in failsafe is impossible. The
		# likely reason is TCP checksumming hardware getting
		# disabled:
		# https://www.mail-archive.com/openwrt-devel@lists.openwrt.org/msg19870.html
		swconfig dev rt305x set enable_vlan 1
		swconfig dev rt305x vlan 1 set ports "0 6"
		swconfig dev rt305x port 6 set untag 0
		swconfig dev rt305x set apply 1
		vconfig add eth0 1
		ifconfig eth0 up
		ifname=eth0.1
	else
		ifname=eth0
	fi
}
```

这里需要将 VLAN 1 的端口修改为 0,1,2,3,4，在 Openwrt 源码树中修改，修改文件为：

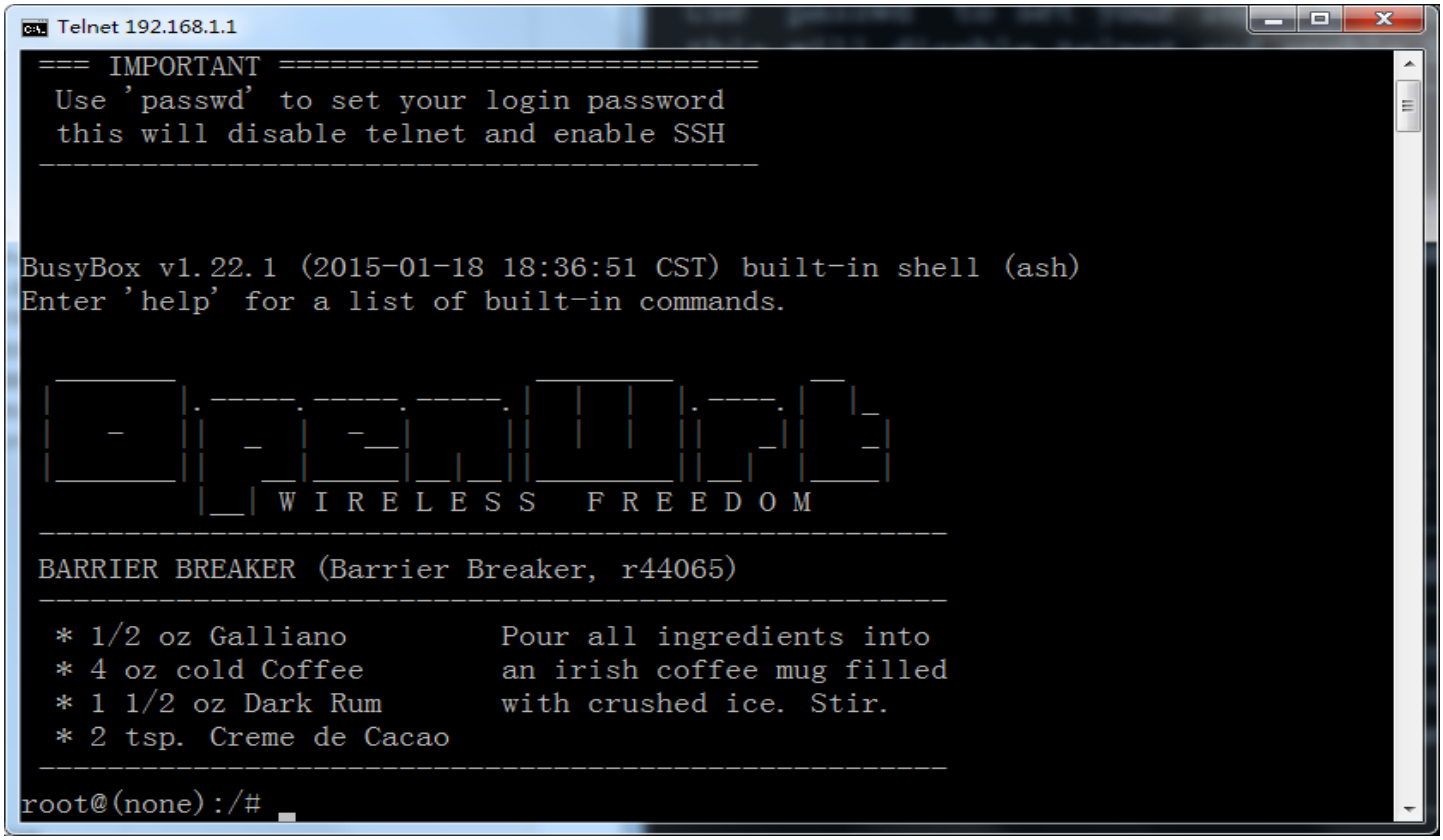
target/linux/ramips/base-files/lib/preinit/07_set_preinit_iface_ramips

swconfig dev rt305x vlan 1 set ports "0 1 2 3 4 6"

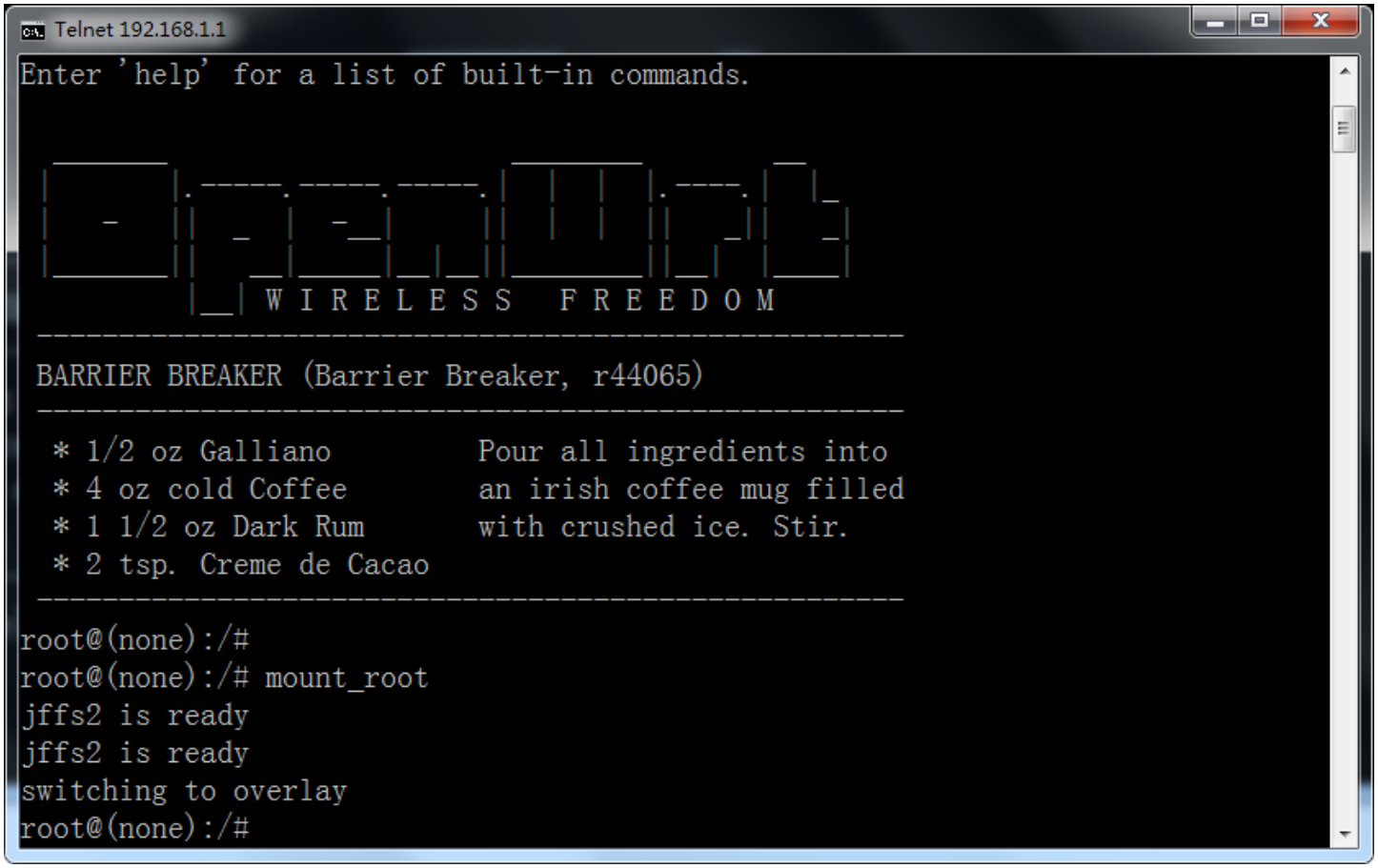
然后重新编译固件，更新固件，重启系统，在日志中出现“Press the [f] key and hit [enter] to enter failsafe mode”这样的提示时，按下 F 和 Enter 键，就进入 failsafe 模式了。

```
procd: - preinit -
[    7.950000] rt305x-esw 10110000.esw: link changed 0x00
Press the [f] key and hit [enter] to enter failsafe mode
Press the [1], [2], [3] or [4] key and hit [enter] to select the debug
f
- failsafe -


BusyBox v1.22.1 (2015-01-18 18:36:51 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

ash: can't access tty; job control turned off
 _______                     ________          __
|       |.-----.-----.-----.|  |  |  |.----.|  |_
|   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
|_______||   __|_____|__|__||________||__|  |____|
         |__| W I R E L E S S   F R E E D O M
 -----------------------------------------------------
 BARRIER BREAKER (Barrier Breaker, r44065)
 -----------------------------------------------------
  * 1/2 oz Galliano         Pour all ingredients into
  * 4 oz cold Coffee        an irish coffee mug filled
  * 1 1/2 oz Dark Rum       with crushed ice. Stir.
  * 2 tsp. Creme de Cacao
 -----------------------------------------------------
root@(none):/# [    9.540000] rt305x-esw 10110000.esw: link changed 0x
```

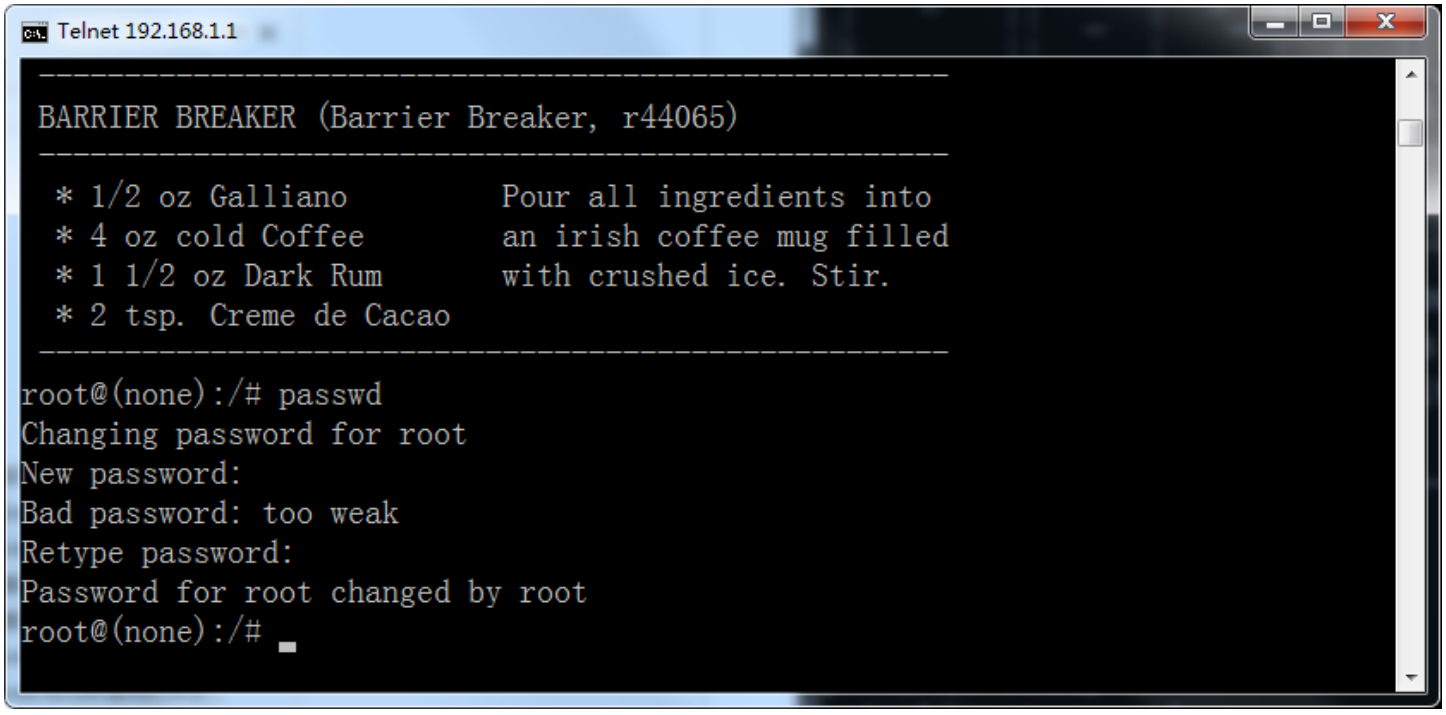
现在需要将我们的电脑的 IP 设置为 192.168.1.2，然后使用 Telnet 工具登录到我们的路由器。



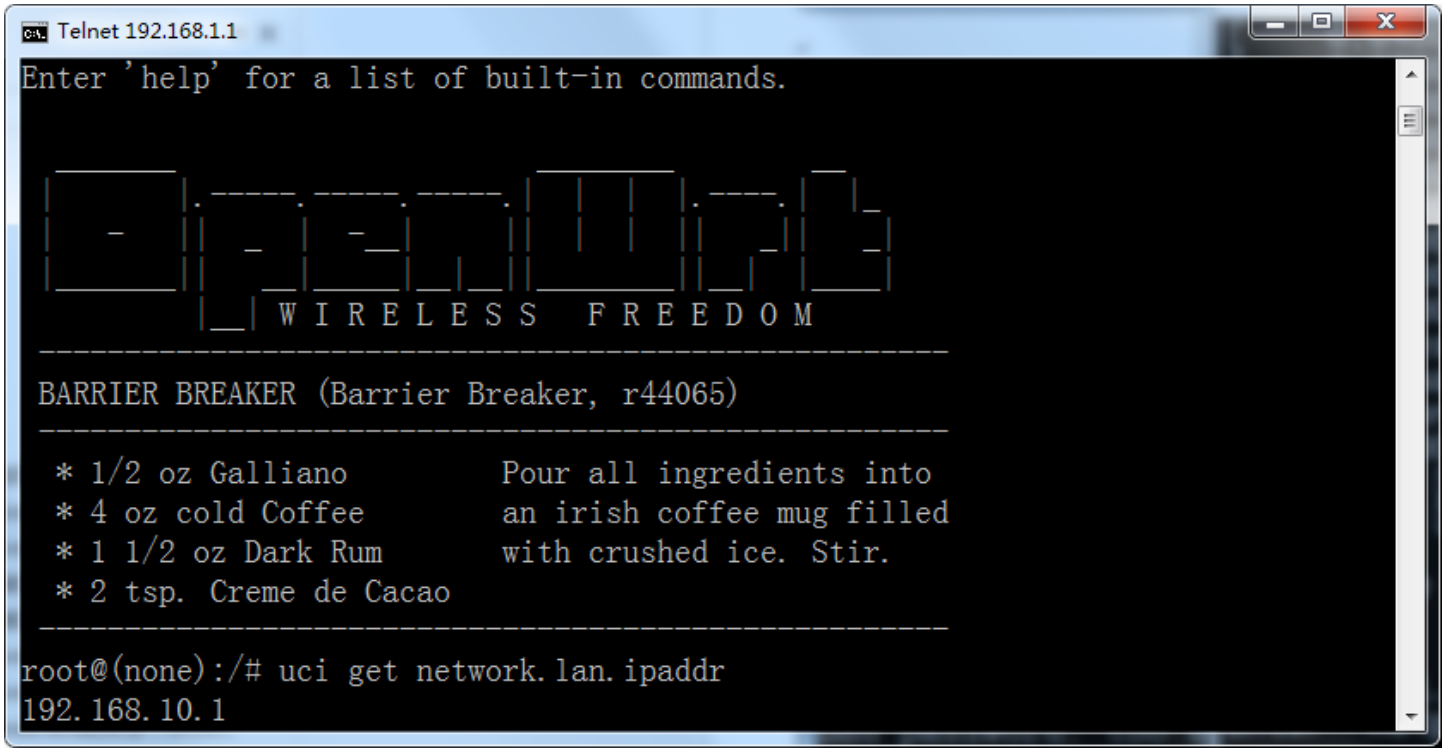
在进入 failsafe 后，系统只挂载了只读的 SquashFS 分区，我们可以通过命令 mount_root 来挂载 jffs2 分区



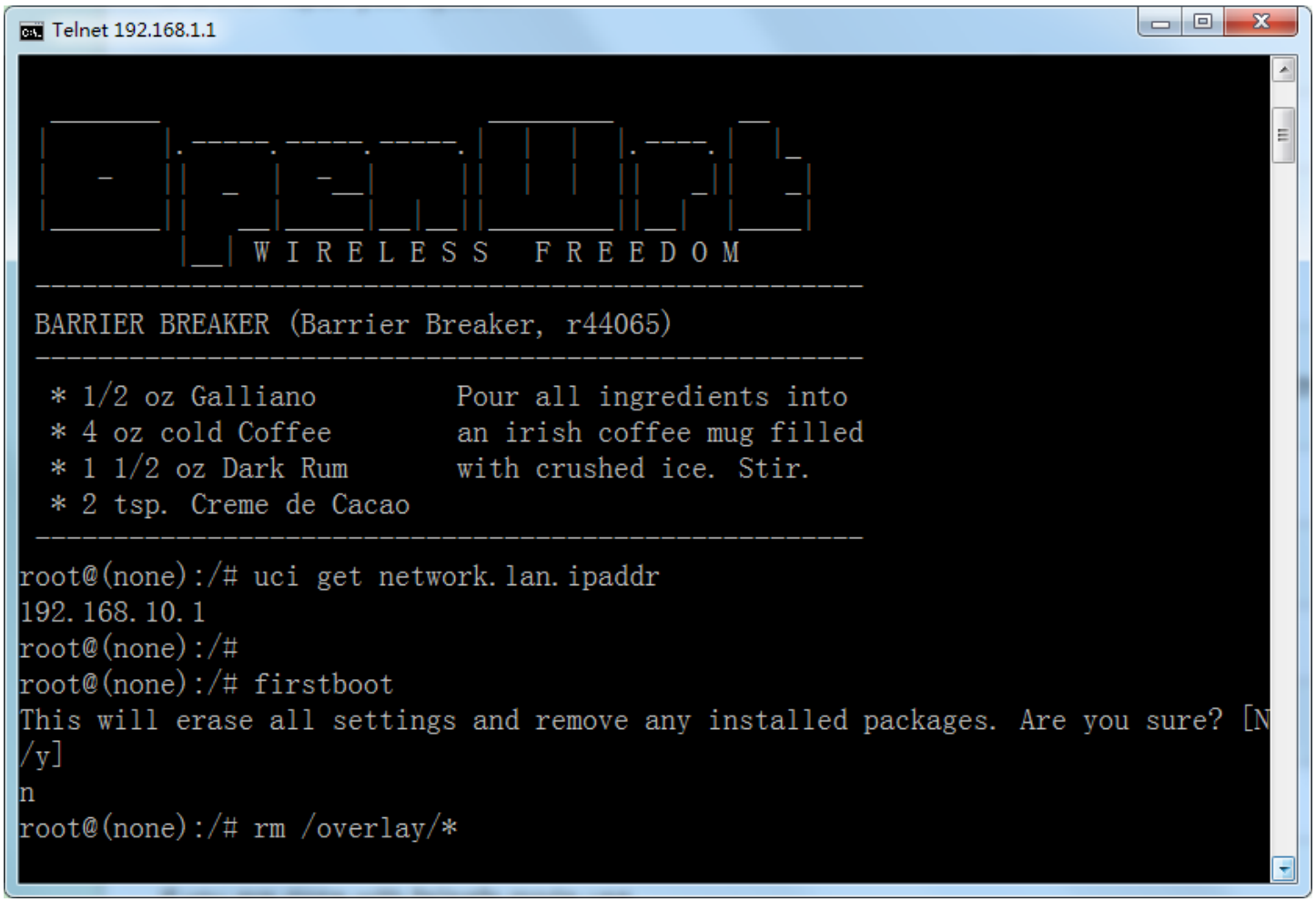
在 failsafe 模式，我们可以修改密码。



如果忘记 IP，可以在这里查看 IP



可以执行 firstboot 命令或者删除/overlay/*来恢复出厂设置。



13 防火墙

13.1 理论知识

参考 Openwrt 官方文档：http://wiki.openwrt.org/doc/uci/firewall
防火墙的配置文件为/etc/config/firewall

Openwrt 依赖于用于包过滤、NAT 和管理的 Netfilter。UCI 抽象出 iptables 系统为防火墙配置提供了一个配置接口，旨在提供一个简单的配置模型。

UCI 防火墙将一个或多个接口映射在一个 Zones 中，Zones 是用来描述一个给定接口的默认规则，接口之间的转发规则，不会被前两个覆盖的扩展规则。在配置文件（/etc/config/firewall）中，默认规则放在最开始位置，但却是最后起作用。数据包进入协议栈，会依次经过 Netfilter 子系统的规则处理，直到最后一条或者封包被丢弃。第一条被匹配的规则经常加载另一条规则，直到封包被 ACCEPT 或者 DROP/REJECT。最终结果，默认规则最后起作用，大多数特殊规则先起作用。Zones 也可以用来配置伪装，也即是我们所熟知的 NAT（网络地址转换），以及端口转发，也就是常说的重定向。

Zones 必须被一个或多个接口所映射，最终映射到一个物理网络设备。

UCI 防火墙配置包括的 section：defaults、zone、forwarding、rule、redirect 和 include。

对于一个路由器，一个最小的防火墙配置通常包括一个 defaults，至少两个 zone（lan 和 wan），一个 forwarding（从 lan 到 wan 转发流量）

defaults：声明一些全局的防火墙设置，它不属于任何 zone。

zone：一个 zone 组织一个或多个接口，以及作为源或目的来为 forwarding、rule 和 redirect 服务。

- 对一个 zone 的 INPUT rule 描述了尝试到达路由器自己的流量将发生什么；
- 对一个 zone 的 OUTPUT rule 描述了来源于路由器自己要出去的流量将发生什么；
- 对一个 zone 的 FORWARD rule 描述了从一个 zone 到另一个 zone 的流量将发生什么。

forwarding：用于控制游走于 zone 之间的流量。

redirect：用于控制 DNAT。

rule：定义基本的的接受或拒绝来允许或限制某个指定的端口或主机。

include：包含一个自定义的防火墙脚本，该脚本使用 iptables 命令定义规则。

下面是一个最简单的 UCI 防火墙配置

```
root@OpenWrt:/# uci export firewall
package firewall

config defaults
        option input 'ACCEPT'
        option output 'ACCEPT'
        option forward 'REJECT'

config zone
        option name 'lan'
        list network 'lan'
        option input 'ACCEPT'
        option output 'ACCEPT'
```

```
        option forward 'ACCEPT'

config zone
        option name 'wan'
        list network 'wan'
        option input 'REJECT'
        option output 'ACCEPT'
        option masq '1'
        option forward 'REJECT'

config forwarding
        option src 'lan'
        option dest 'wan'

config include
        option path '/etc/firewall.user'
```

defaults 中设置 filter 表中的 3 个链的默认策略；
第一个 zone 表示经过 LAN 口的流量，不论是出去、进来还是转发，都放行；
第二个 zone 表示经过 WAN 口的流量，拒绝进入本地和转发的流量，但允许出口流量，并且对出口流量进行地址伪装。
forwarding 表示允许从 lan 转发到 wan；
include 包含一个脚本，我们可以在该脚本中通过 iptables 命令定义更复杂的规则。

有了这个最简单的防火墙配置，就可以在此基础上添加更多的规则。比如 MAC 过滤、端口过滤、IP 过滤、协议过滤、NAT、重定向等。

13.2 UCI 防火墙配置实例

13.2.1 MAC 地址过滤

```
config rule
        option src          'lan'
        option dest         'wan'
        option src_mac      '44:8A:5B:EC:49:27'
        option proto        'all'
        option target       'REJECT'
```

- 从 lan 转发到 wan 并且源 MAC 为'44:8A:5B:EC:49:27'的所有流量都被拒绝。
- proto 选项的默认值为 tcpudp。因此，如果未指定该选项，则表示只匹配 tcp 和 udp 协议。
- target 选项可以指定为 ACCEPT、REJECT 和 DROP，ACCEPT 表示放行，REJECT 表示拒绝，并且路由器会向源主机发送 ICMP 目的不可达消息，DROP 表示悄悄的将数据包丢弃。
- 如果源 zone 和目的 zone 都指定，则 rule 只匹配转发流量；如果只指定源 zone，则 rule 只匹配入站流量（发往本机）；如果只指定目的 zone，则 rule 只匹配出站流量。
- rule 还有很多选项，比如 src_ip，src_port，dest_ip，dest_port，另外还有很多时间选项，可以匹配指

定的时间段，比如规定什么时间段可以上网。

13.2.2 端口转发

```
config redirect
        option src          'wan'
        option dest         'lan'
        option src_dport 8080
        option proto        'tcp'
        option dest_ip    '192.168.10.104'
        option dest_port 80
```

假设 wan 口 IP 为 192.168.0.105，我们在连接到 wan 的机器上访问 192.168.0.105:8080，路由器将会将我们的数据转发到连接到 lan 的 IP 为 192.168.10.104 的机器的 80 端口。

14 配置 PPPOE Server

首先配置 Openwrt 软件包

```
Network    --->
    dial-in/up    --->
        <*> rp-pppoe-server
```

rp-pppoe-server 在运行时，还需要一个程序 pppoe，该程序默认没有被安装，因此需要修改一个 Makefile
在 feeds/oldpackages/net/rp-pppoe/Makefile 中添加一行，如下所示

```
define Package/rp-pppoe-client/install
    $(INSTALL_DIR) $(1)/etc/init.d
    $(INSTALL_BIN) ./files/pppoe-client.init $(1)/etc/init.d/pppoe-client
    $(INSTALL_DIR) $(1)/etc/ppp
    $(CP) $(PKG_INSTALL_DIR)/etc/ppp/pppoe.conf $(1)/etc/ppp/
    $(INSTALL_DIR) $(1)/usr/sbin
    $(CP) $(PKG_INSTALL_DIR)/usr/sbin/pppoe $(1)/usr/sbin/
    $(INSTALL_BIN) $(PKG_BUILD_DIR)/scripts/pppoe-connect $(1)/usr/sbin/
    $(INSTALL_BIN) $(PKG_BUILD_DIR)/scripts/pppoe-start $(1)/usr/sbin/
    $(INSTALL_BIN) $(PKG_BUILD_DIR)/scripts/pppoe-stop $(1)/usr/sbin/
    $(SED) 's,modprobe,insmod,g' $(1)/usr/sbin/pppoe-connect
endef
```

然后执行 make 编译，然后更新固件
在路由器上修改/etc/ppp/pppoe-server-options 认证方式
require-chap

在/etc/ppp/chap-secrets 中添加用户名和密码
#USERNAME PROVIDER PASSWORD IPADDRESS
root * 123 *

设置 br-lan 的 IP 为 0.0.0.0
root@OpenWrt:/# ifconfig br-lan 0.0.0.0

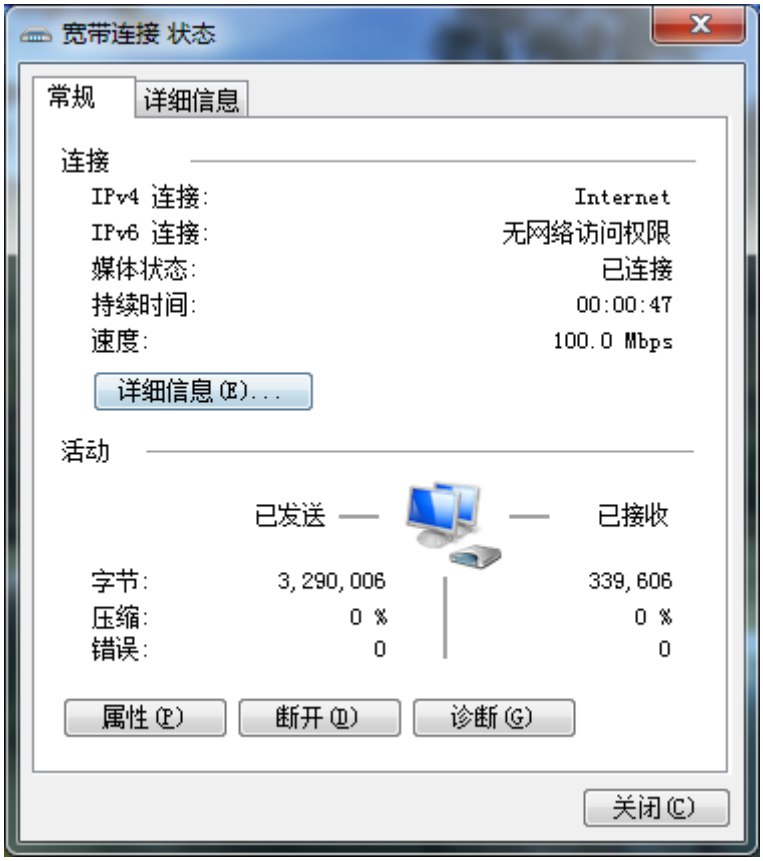
修改/etc/ppp/options 添加 dns
ms-dns 192.168.10.1

启动 pppoe-serve
pppoe-server -I br-lan -L 192.168.10.1 -R 192.168.10.5 -N 10

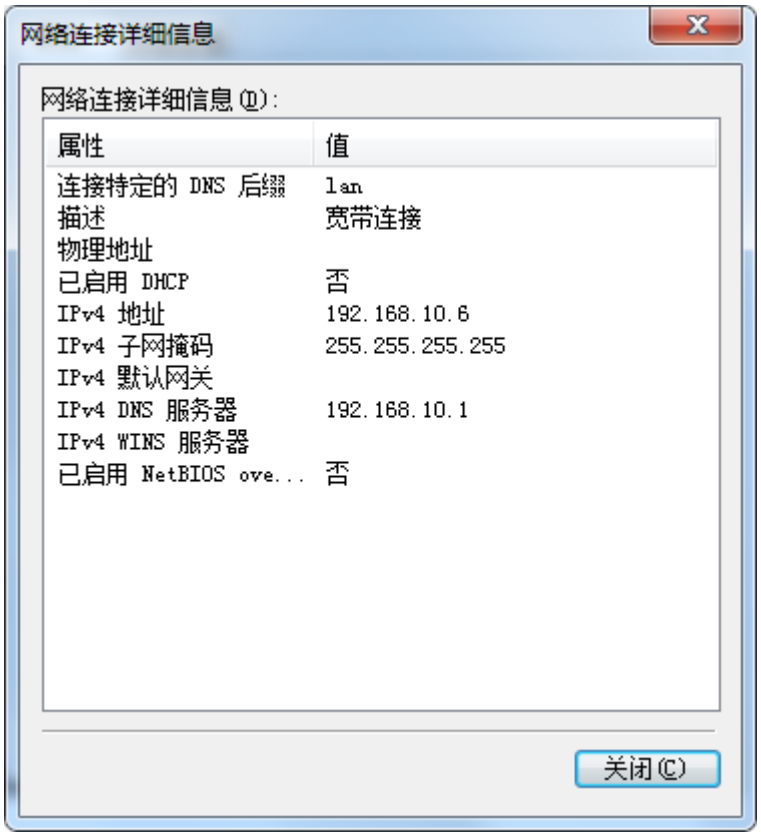
设置防火墙
root@OpenWrt:/# iptables -t filter -I FORWARD 1 -o eth0.2 -i ppp0 -j ACCEPT

这样客户端通过 PPPOE 宽带拨号连接，就可以上网了，如果浏览器提示“代理服务器拒绝”，则需设置 Internet 选项连接->宽带连接设置取消代理服务器

然后双击桌面的宽带连接



单击详细信息



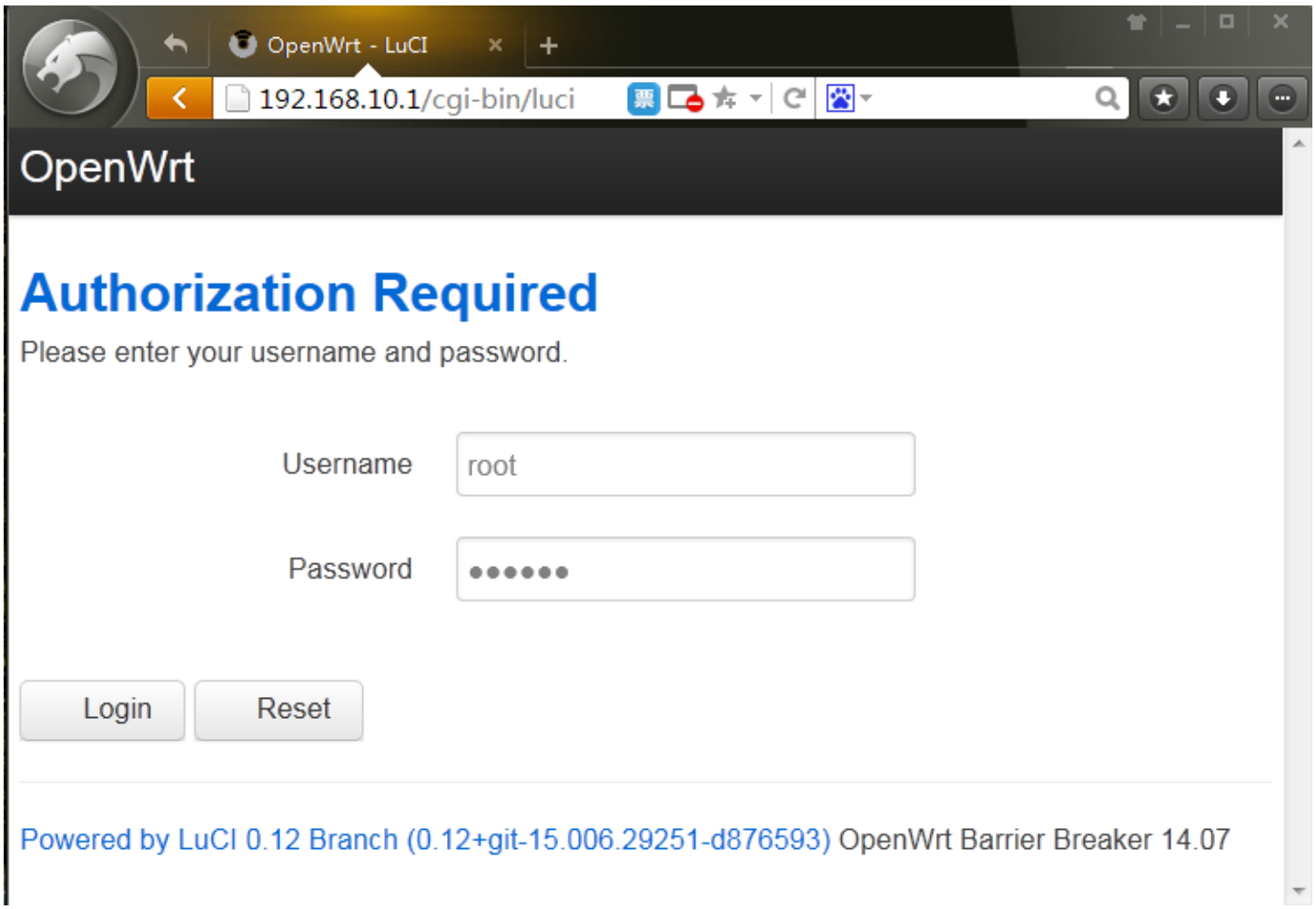
现在已经可以上网了。

15 LuCI

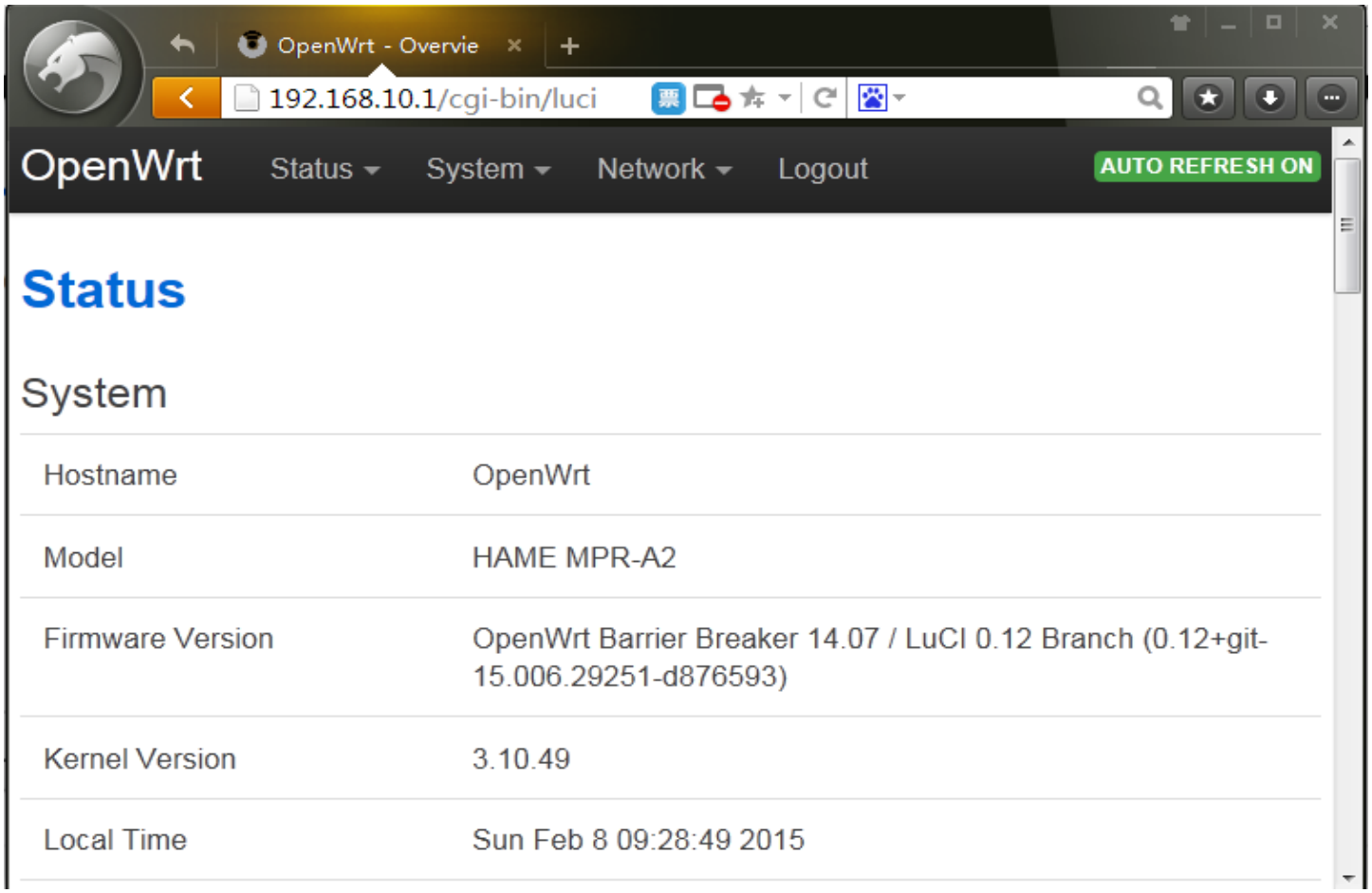
15.1 配置 Openwrt 支持 LuCI

LuCI ---->
1. Collections ---->
-*- luci
<*> luci-ssl
5. Translations ---->
<*> luci-i18n-chinese
<*> luci-i18n-english

重新编译 Openwrt 并更新固件。然后通过浏览器访问路由器开发板的 LAN 口 IP 地址



然后点击 Login 按钮



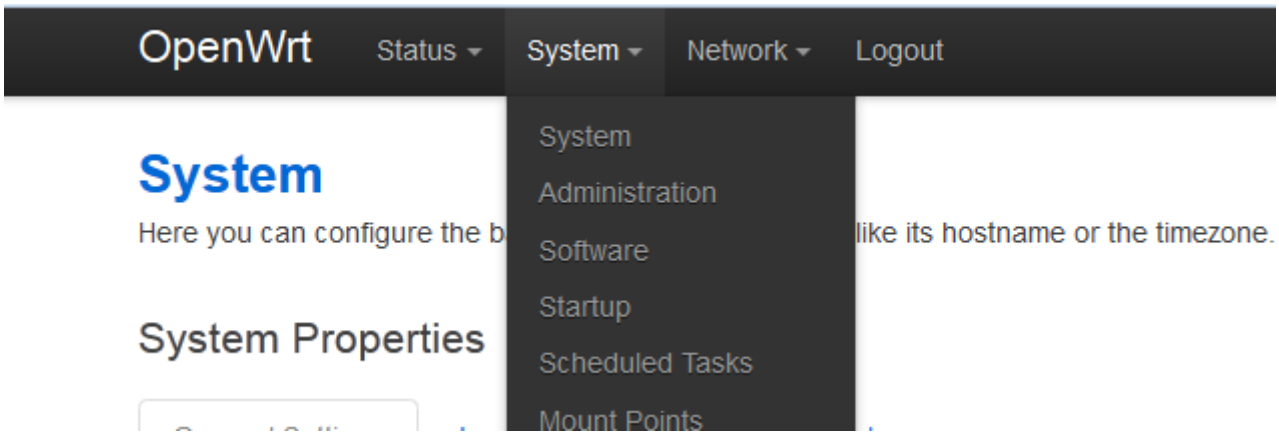
15.2 LuCI

轻量级 LUA 语言的官方版本只包括一个精简的核心和最基本的库。这使得 LUA 体积小、启动速度快，从而适合嵌入在别的程序里。UCI 是 Openwrt 中为实现所有系统配置的一个统一接口，英文名 Unified Configuration Interface，即统一配置接口。LuCI 即是这两个项目的合体，可以实现路由的网页配置界面。

最好先学习 LUA 脚本编程，网上关于 LUA 的资料非常多。

LuCI 使用 MVC（模型/视图/控制）模型，在/usr/lib/lua/luci/下有三个目录 model、view、controller，它们分别对应 M、V、C。我们要做的主要工作就是基于 LuCI 框架编写 LUA 脚本、在 html 页面中嵌入 LUA 脚本。

LuCI 将网页中的每一个菜单视作一个节点，如下所示：



这里有一级节点 Status、System、Network、Logout，二级节点 System、Administration、Software、Startup 等。在我们使

用浏览器向路由器发起请求时，LuCI 会从 controller 目录下的 index.lua 开始组织这些节点。index.lua 中定义了根节点 root，其他所有的节点都挂在这个根节点上。
我们可以将 controller 目录下的这些.lua 文件叫做模块，这样的模块文件的第一行必须是如下格式：

```
module("luci. controller.xx.xx.xx", package.seeall)
```

上面的 luci.controller.xx.xx.xx 表示该文件的路径，luci. controller 表示/usr/lib/lua/luci/controller/，比如上面的 index.lua 其第一行为：module("luci.controller.admin.index", package.seeall)，表示其路径为：/usr/lib/lua/luci/controller/admin/index.lua

接着是定义一个 index 方法，其主要工作是调用 entry 方法创建节点，可以多次调用创建多个节点。其调用方式如下：
entry (path, target, title, order)

- path 指定该节点的位置（例如 node1.node2.node3）
- target 指定当该节点被调度（即用户点击）时的行为，主要有三种：call、template 和 cbi，后面有 3 个实例。
- title：标题，即我们在网页中看到的菜单
- order：同一级节点之间的顺序，越小越靠前，反之越靠后（可选）

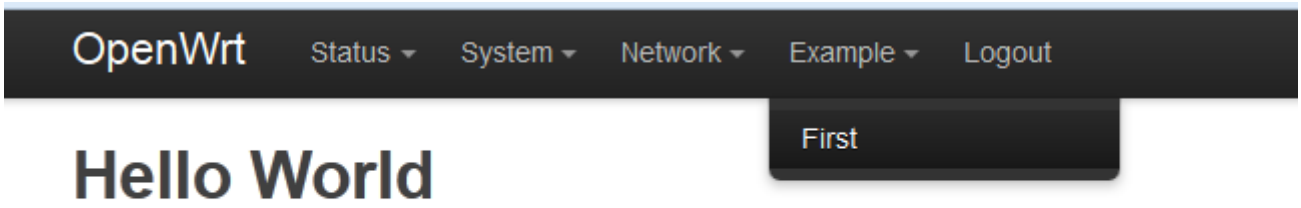
LuCI 默认开启了缓存机制，这样当我们修改了代码后，不会立即生效，除非删除缓存，操作如下：

```
root@OpenWrt:/# rm /tmp/luci-* -
```

为了便于调试，我们可以直接关闭缓存，修改配置文件/etc/config/luci，操作如下：

```
root@OpenWrt:/# uci set luci.ccache.enable=0
root@OpenWrt:/# uci commit
```

15.3 实例一：call



创建 lua 脚本文件：/usr/lib/lua/luci/controller# ls example.lua，其内容如下：

```
--第一行声明模块路径
module("luci.controller.example", package.seeall)

function index()
    --[[
        创建一级菜单 example，firstchild()表示链接到其第一个子节点，即
        当我们单击菜单 Example 时，LuCI 将调度其第一个子节点。"Example"即
        在网页中显示的菜单。60 表示其顺序，LuCI 自带的模块的顺序为：
        Administration(10),Status(20),System(30),Network(50),Logout(90)。
        call("first_action")表示当子节点被调度时将执行下面定义的方法 first_action()
    --]]
    entry({"admin", "example"}, firstchild(), "Example", 60)
```

```
    entry({"admin", "example", "first"}, call("first_action"), "First")
end

function first_action()
    --加载/usr/lib/lua/luci/view/header.htm
    luci.template.render("header")
    --输出 html 内容
    luci.http.write("<h1>Hello World</h1>")
end
```

在 lua 脚本中，使用--表示单行注释，--[[--]]表示多行注释。

15.4 实例二：template

效果和 15.3 节一样，不同的是上节使用 call 调度执行一个函数，本节直接调用一个 html 页面。

创建 html 文件：/usr/lib/lua/luci/view/example# ls example.htm，其内容如下

```
<%+header%>
<h1>Hello World</h1>
```

这里的<% %>用来在 html 代码中嵌入 LUA 脚本，这里的<%+header%>表示首先加载/usr/lib/lua/luci/view/header.htm

修改后的/usr/lib/lua/luci/controller/example.lua 内容如下：

```
--第一行声明模块路径
module("luci.controller.example", package.seeall)

function index()
    --[[
        创建一级菜单 example，firstchild()表示链接到其第一个子节点，即
        当我们单击菜单 Example 时，LuCI 将调度其第一个子节点。"Example"即
        在网页中显示的菜单。60 表示其顺序，LuCI 自带的模块的顺序为：
        Administration(10),Status(20),System(30),Network(50),Logout(90)
        template ("example/example")表示当子节点被调度时 LuCI 将使用我们指定的
        html 页面/usr/lib/lua/luci/view/example/example.htm 来生成页面
    --]]
    entry({"admin", "example"}, firstchild(), "Example", 60)
    entry({"admin", "example", "first"}, template("example/example"), "First")
end
```

15.5 实例三：cbi

该方法与 UCI 配置息息相关。主要用来修改 UCI 配置文件以及使配置生效。

首先修改/usr/lib/lua/luci/controller/example.lua，内容如下：

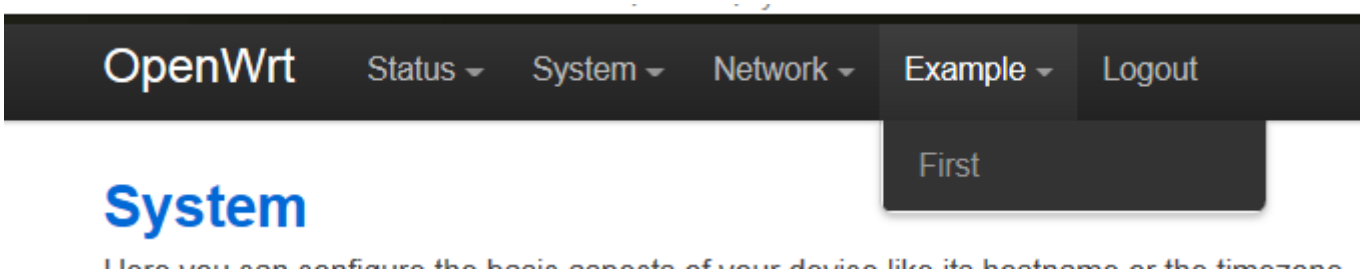
```
--第一行声明模块路径
module("luci.controller.example", package.seeall)

function index()
    --[[
        创建一级菜单 example，firstchild()表示链接到其第一个子节点，即
        当我们单击菜单 Example 时，LuCI 将调度其第一个子节点。"Example"即
```

```
        在网页中显示的菜单。60 表示其顺序，LuCI 自带的模块的顺序为：
        Administration(10),Status(20),System(30),Network(50),Logout(90)
        call("first_action")表示当子节点被调度时将执行下面定义的方法 first_action()
    --]]
    entry({"admin", "example"}, firstchild(), "Example", 60)
    entry({"admin", "example", "first"}, template("example/example"), "First", 1)

    --[[
        如果配置文件/etc/config/example 存在，则创建 Example 的子节点 Second。
        cbi("example/example")表示当节点 Second 被调度时，LuCI 会将
        /usr/lib/lua/luci/model/cbi/example/example.lua 这个脚本转换成 html 页面
        发给客户端。
    --]]
    if nixio.fs.access("/etc/config/example") then
        entry({"admin", "example", "second"}, cbi("example/example"), "Second", 2)
    end
end
```

新建 lua 脚本文件：/usr/lib/lua/luci/model/cbi/example/example.lua，内容如下：

```
--[[
    有关 Map 相关的用法参考 LuCI 官方文档：http://luci.subsignal.org/trac/wiki/Documentation/CBI
    Map (config, title, description)
--]]
m = Map("example", "Example for cbi", "This is very simple example for cbi")
return m
```

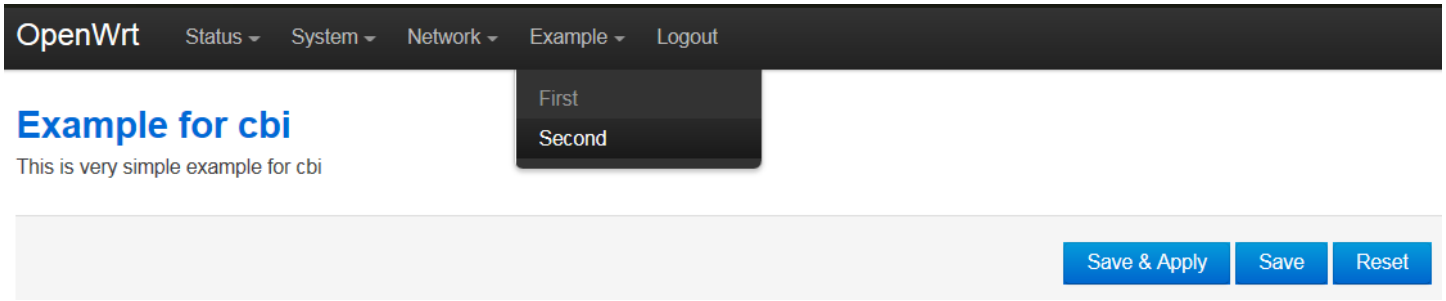
现在刷新网页，看下效果



没有出现 Second 这个节点，是因为还没创建配置文件/etc/config/example，下面使用 touch 命令创建这个文件：

```
root@OpenWrt:/# touch /etc/config/example
```

再次刷新网页，看下效果：

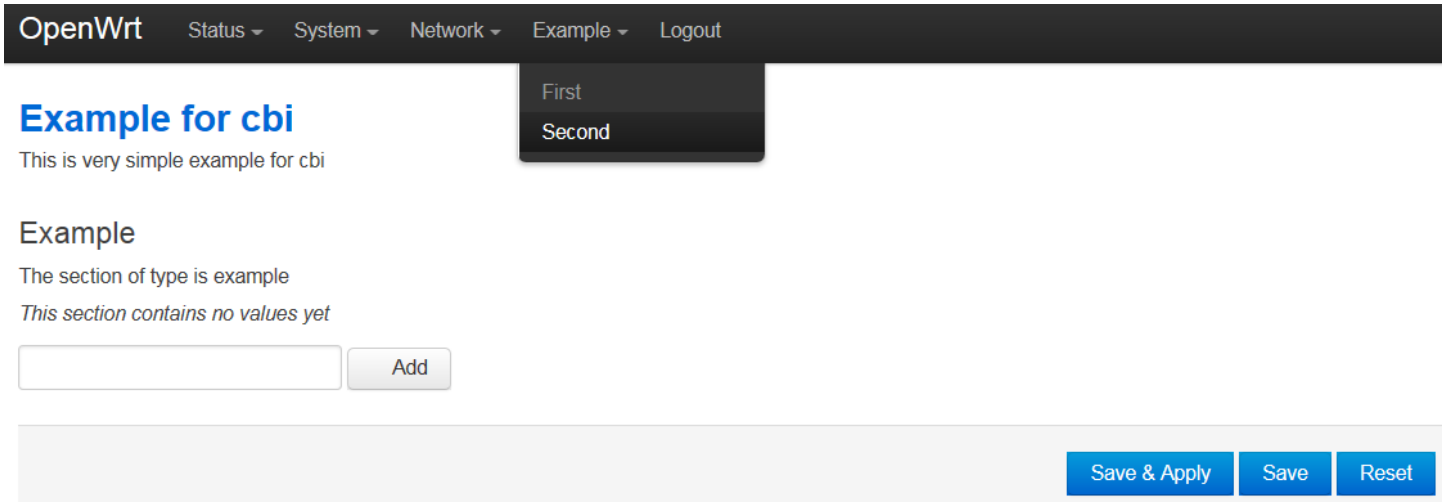


现在还没添加具体的业务功能，只是一个模型。下面继续修改/usr/lib/lua/luci/model/cbi/example/example.lua，内容如下：

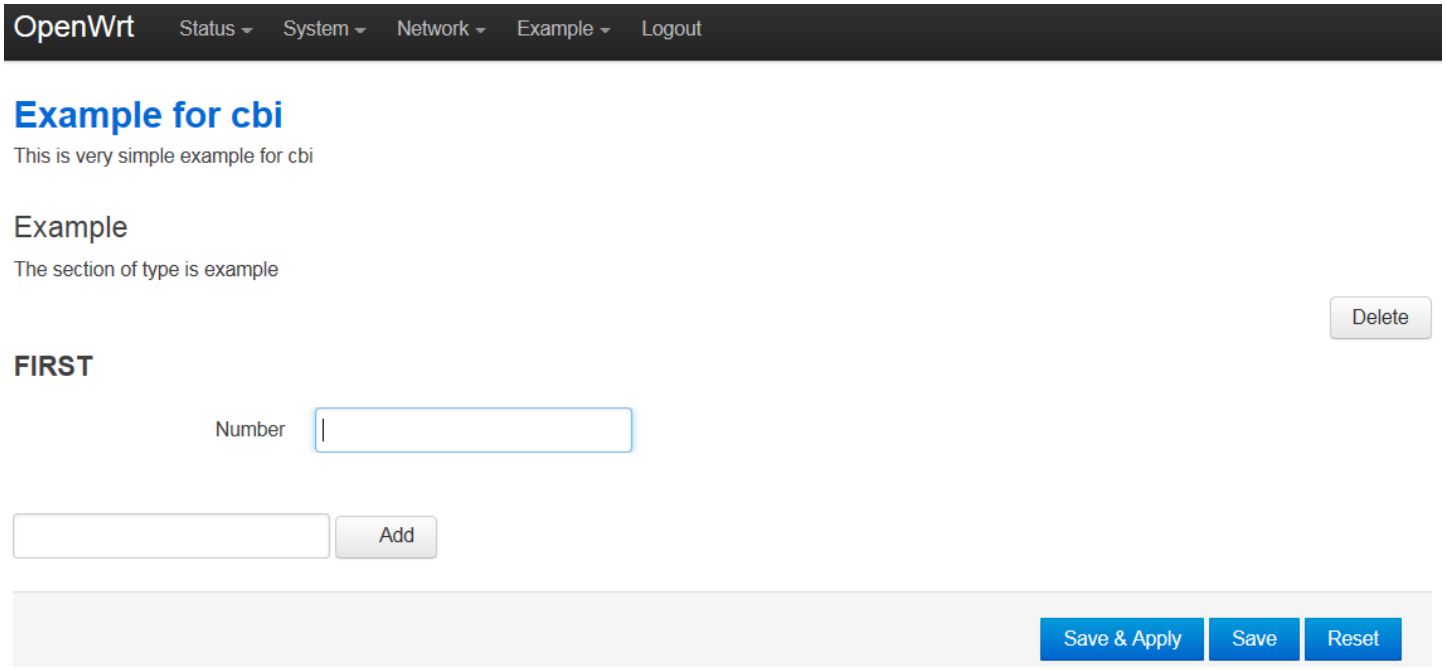
```
--[[
```

```
	有关 Map 相关的用法参考 LuCI 官方文档：http://luci.subsignal.org/trac/wiki/Documentation/CBI
	Map (config, title, description)
--]]
m = Map("example", "Example for cbi", "This is very simple example for cbi")

--描述一个类型为 example 的 section
s = m:section(TypedSection, "example", "Example", "The section of type is example")

--允许用户添加和删除这个 section，默认为 false
s.addremove = true

--是否为匿名 section，默认为 false
s.anonymous = false

--该 section 的 num 选项
n = s:option(Value, "num", "Number")

--如果为空值则不设置，默认为 true
n.rmempty = true

return m
```

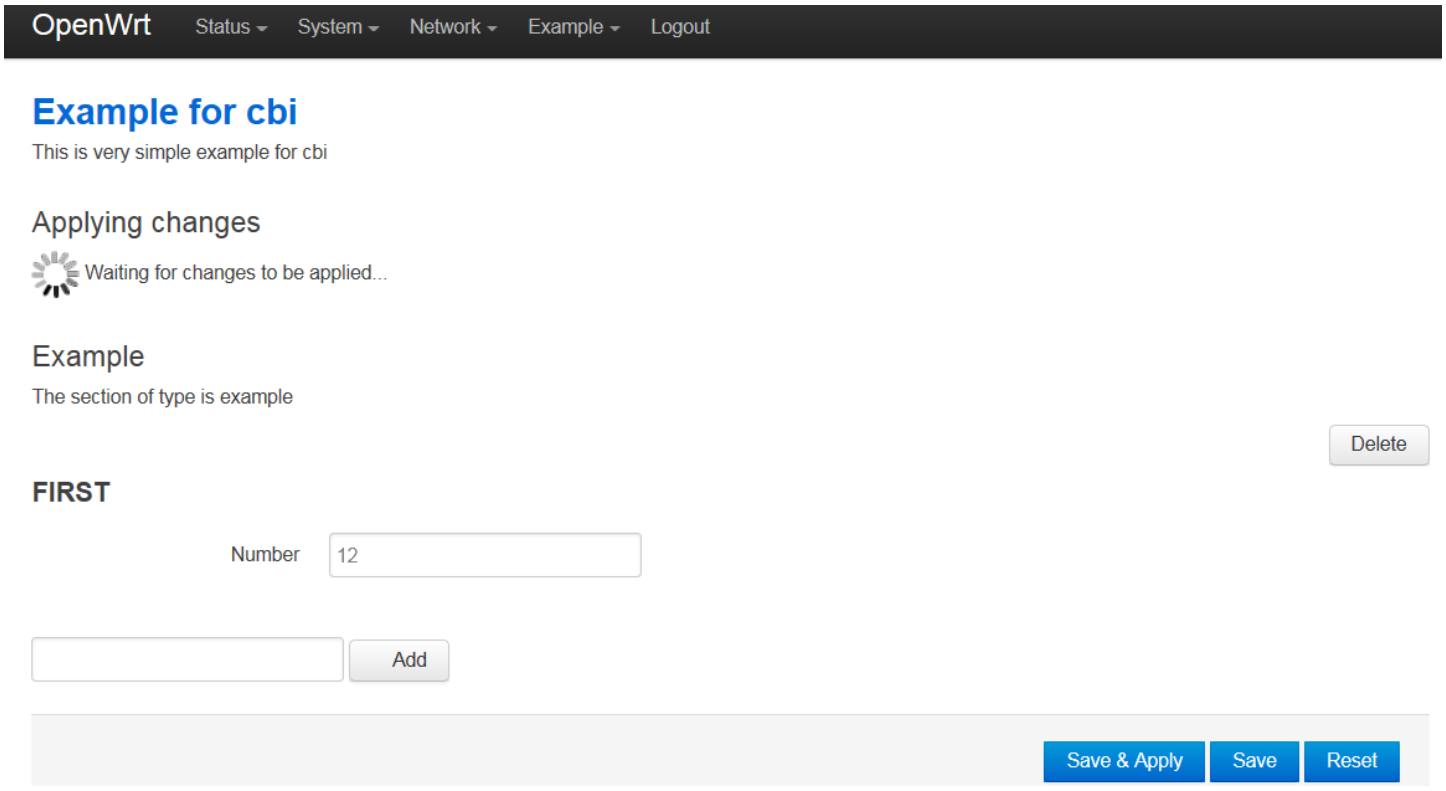
刷新网页，查看效果：



现在可以向配置文件/etc/config/example 中添加 section 了，首先在文本框中输入 first，然后单击 Add，如下所示：



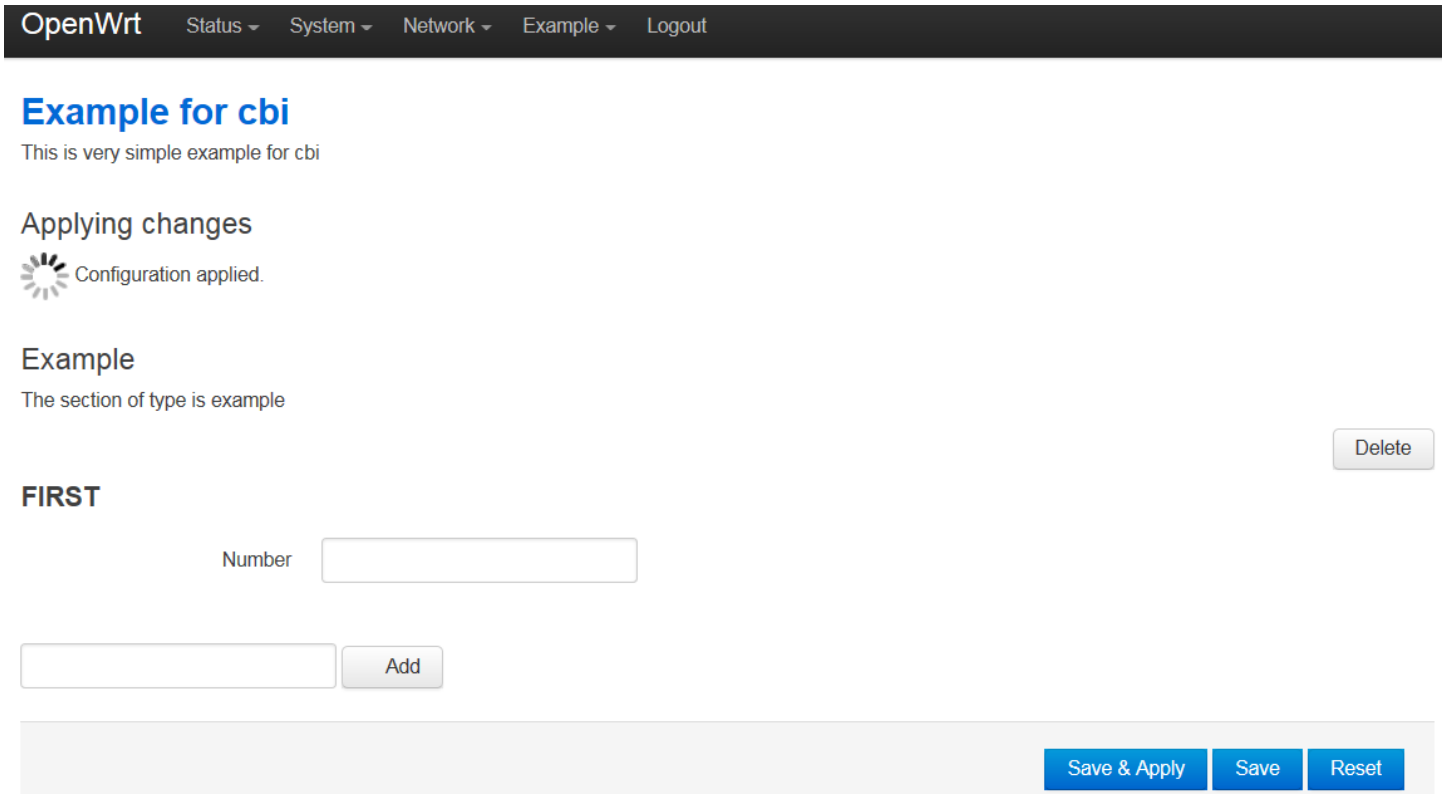
然后在 Number 后面随便输入一个数字，比如 12，然后单击 Save & Apply，如下所示：



现在在路由器开发板上查看一些配置文件

```
root@OpenWrt:/# uci export example
package example

config example 'first'
        option num '12'
```

我们在代码中将 n.rmempty 设置为 true，表示当用户对该选项的输入值为空值时，LuCI 会将该选项从配置文件中移除。现在我们将 Number 的值删除，再单击 Save & Apply，如下所示：



现在再来查看配置文件：

```
root@OpenWrt:/# uci export example
package example

config example 'first'
```

一般情况下，一个配置文件与一个启动脚本对应，比如/etc/config/network 和/etc/init.d/network，当我们在页面上单击 Save & Apply 时，LuCI 首先保存配置文件，然后就以 reload 为参数调用配置文件对应的启动脚本。

为配置文件 example 创建一个启动脚本/etc/init.d/example，同时为其添加可执行权限。其内容如下：

```
#!/bin/sh /etc/rc.common
# Copyright (C) 2006 OpenWrt.org

START=50

start() {
        echo "start example" > /dev/ttyS0
}

reload() {
        echo "reload example" > /dev/ttyS0
}
```

另外，LuCI 通过以配置文件名作为参数调用/sbin/luci-reload 来使配置生效，而 luci-reload 会解析另一个配置文件 /etc/config/ucitrack，我们需要将我们的 example 添加进去。用 vi 打开/etc/config/ucitrack，在最后添加如下内容：

```
config example
        option init example
```

当我们单击网页中的 Save & Apply 后，LuCI 将会调用/sbin/luci-reload example，最终调用到/etc/rc.d/example 中的 reload 函数，因此还需要执行如下操作：

```
root@OpenWrt:/# /etc/init.d/example enable
```

现在单击网页中的 Save & Apply，可以看到开发板中输出了如下内容：
reload example
说明确实执行了/etc/init.d/example 中的 reload 函数。

15.6 CBI 参考手册

15.6.1 Map

Map (config, title, description)
映射一个配置文件，返回一个 Map 对象。其中 description 可以省略。

m = Map("example", "Example for cbi"), "This is very simple example for cbi")

15.6.2 section

s = m:section(TypedSection, type, title, description)　　根据 section 类型解析 section
s = m:section(NamedSection, name, type, title, description)　　根据 section 类型和名称解析 section
其中 m 为 Map 返回的对象，s 为 Map 类的成员函数 section 返回的 section 对象

section 对象有一些属性如下：
template：html 模板，默认为"cbi/tsection"
addremove：是否可以增加和删除，默认为 false
anonymous：是否为匿名 section，默认为 false

15.6.3 option

o = s:option(type, name, title, description)
调用 section 对象的成员函数 option，返回一个 option 对象。其中 type 有多个取值：Value、DynamicList、Flag、ListValue。
option 对象的一些属性如下：

- rmempty：如果为空值则删除该选项，默认为 true
- default：默认值，如果为空值，则设置为该默认值
- datatype：限制输入类型。例如"and(uinteger,min(1))"限制输入无符号整形而且大于 0，"ip4addr"限制输入 IPv4 地址，"port"限制输入类型为端口，更多参考/usr/lib/lua/luci/cbi/datatypes.lua
- placeholder：占位符（html5 才支持）

15.6.4 Tab

s:tab(tabname, title)

调用 section 的 tab 函数创建一个名称为 tabname，标题为 title 的 Tab 标签。

对应的 option 则使用 taboption
s:taboption(tabname, type, name, title)
在指定的 tabname 下创建一个 option。

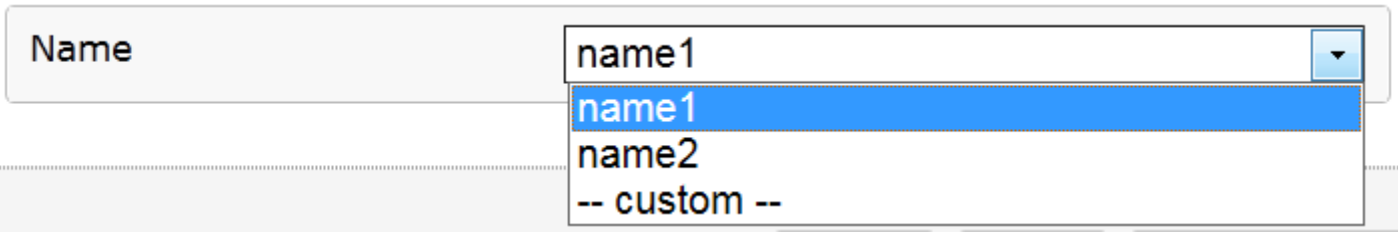
参考下节的例 6。

15.6.5 实例

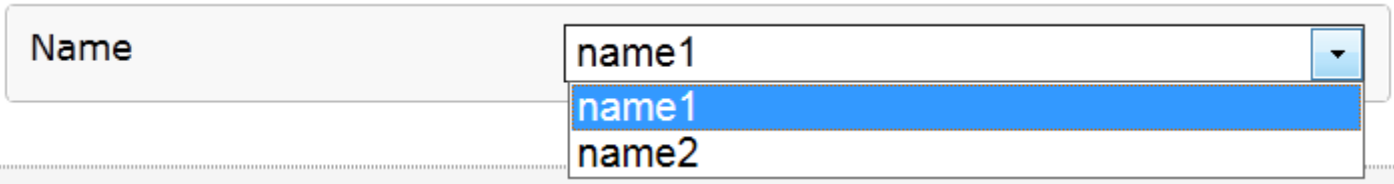
例 1：
o = s:option(Value, "xx")
o.datatype = "port"

这里将 option 类型设置为 port，当非法输入时，LuCI 以红色提示输入错误。

例 2：
o = s:option(Value, "name", "Name")
o:value("value1")
o:value("value2")
o.default = " value1"

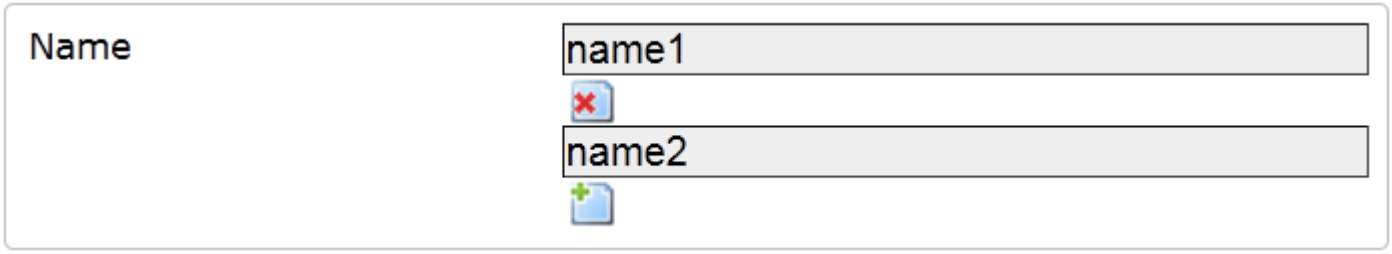


例 3：
o = s:option(ListValue, "name", "Name")
o:value("name1")
o:value("name2")
o.default = "name1"



例 3 和例 2 类似，区别是 option 的类型不同，例 2 中类型为 Value，例 3 中类型为 ListValue，前者用户可以选择 custom 自定义，而后者不能。

例 4：
o = s:option(DynamicList, "name", "Name")

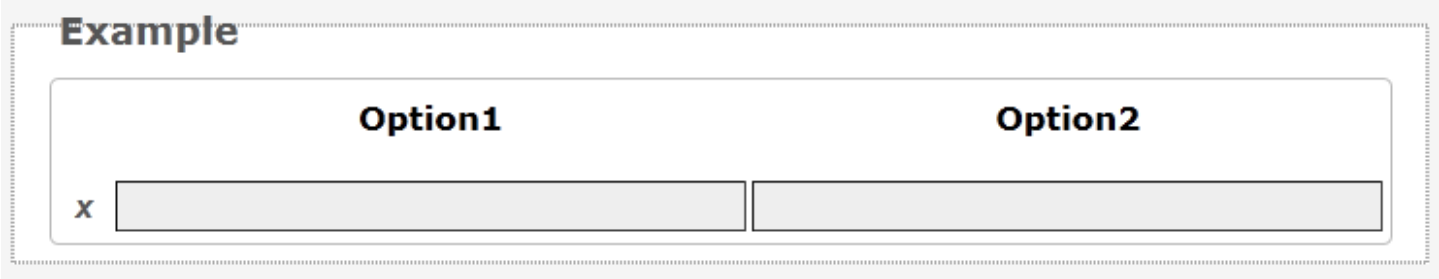


DynamicList 对应 UCI 配置文件中的 list

```
config example 'x'
        list name 'name1'
        list name 'name2'
```

例 5：

```
s = m:section(TypedSection, "example", translate("Example"))
s.template = "cbi/tblsection"
s:option(Value, "option1", "Option1")
s:option(Value, "option2", "Option2")
```



该实例通过 m 的 template 属性改变了 html 的模板。

例 6：

```
s = m:section(TypedSection, "example", translate("Example"))
s:tab("tab1", "Tab1")
s:tab("tab2", "Tab2")
s:taboption("tab1", Value, "option1", "Option1")
s:taboption("tab2", Value, "option2", "Option2")
```

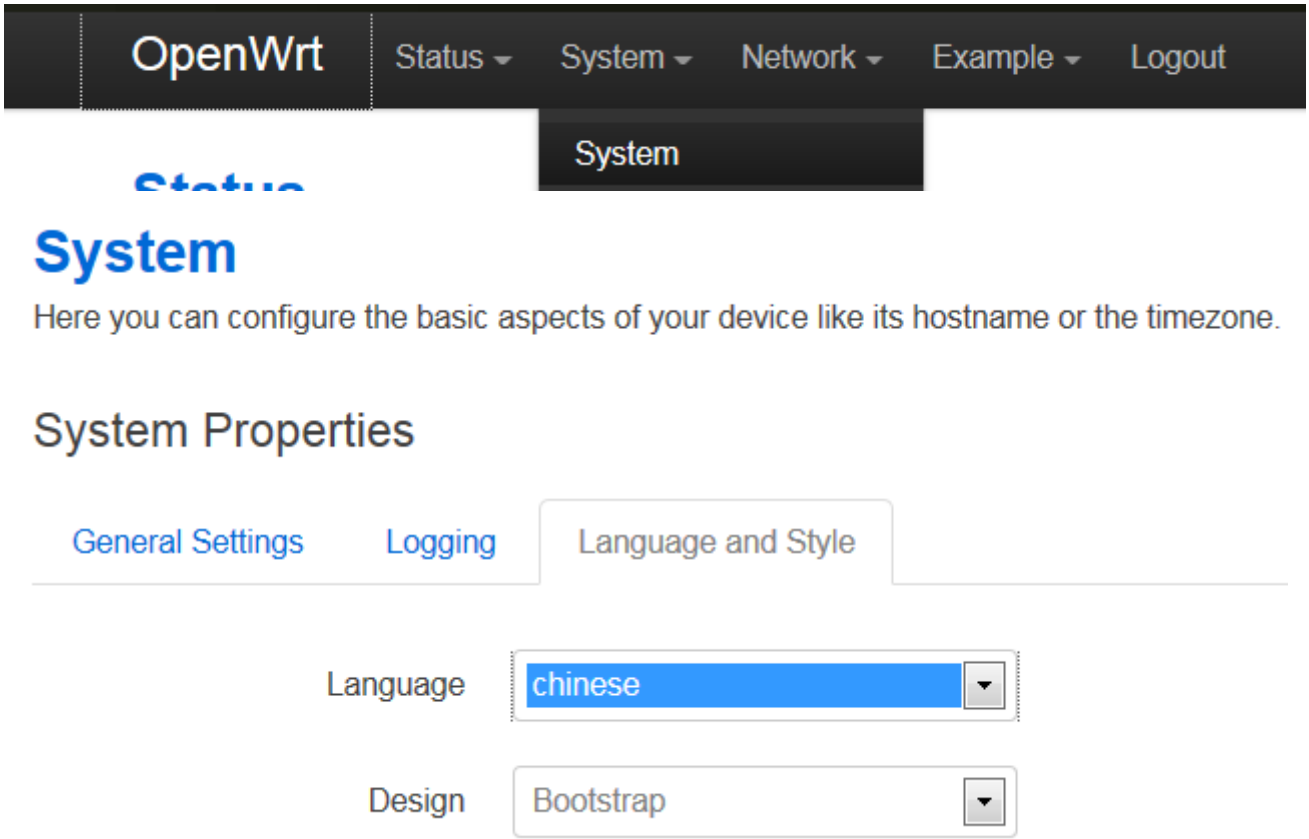
15.7 国际化

LucI 支持多种语言，我们可以在 menuconfig 中进行配置

```
LuCI    --->
        5. Translations    --->
                < > luci-i18n-catalan..... Catalan (by Eduard Duran)
                <*> luci-i18n-chinese..... Chinese (by Chinese Translators)
                <*> luci-i18n-english..... English
                < > luci-i18n-french..... French (by Florian Fainelli)
```

< > luci-i18n-german..... German
< > luci-i18n-greek..... Greek (by Vasilis Tsiligiannis)
< > luci-i18n-hebrew..... Hebrew
< > luci-i18n-hungarian..... Hungarian
< > luci-i18n-italian..... Italian (by Matteo Croce)
< > luci-i18n-japanese..... Japanese (by Tsukasa Hamano)
< > luci-i18n-malay..... Malay (by Teow Wai Chet)
< > luci-i18n-norwegian..... Norwegian (by Lars Hardy)
< > luci-i18n-polish..... Polish
< > luci-i18n-portuguese..... Portuguese (by Jose Monteiro)
< > luci-i18n-portuguese-brazilian
< > luci-i18n-romanian..... Romanian
< > luci-i18n-russian..... Russian (by Skryabin Dmitry)
< > luci-i18n-spanish..... Spanish (by Guillermo Javier Nardoni)
< > luci-i18n-ukrainian..... Ukrainian
< > luci-i18n-vietnamese..... Vietnamese (by Hong Phuc Dang)

在网页中选择语言，单击 System 下的 System 菜单，然后选择 Language and Style



这里选择中文，然后单击 Save & Apply 使配置生效，然后刷新。

下面修改我们自己写的模块，使其支持多语言。对于/usr/lib/lua/luci/controller/目录下的模块，我们可以使用_()来翻译标题，修改/usr/lib/lua/luci/controller/example.lua，修改后分内容如下：

```
--第一行声明模块路径
module("luci.controller.example", package.seeall)

require "luci.fs"

function index()
    --[[
        创建一级菜单 example，firstchild()表示链接到其第一个子节点，即
        当我们单击菜单 Example 时，LuCI 将调度其第一个子节点。"Example"即
        在网页中显示的菜单。60 表示其顺序，LuCI 自带的模块的顺序为：
        Administration(10),Status(20),System(30),Network(50),Logout(90)
        call("first_action")表示当子节点被调度时将执行下面定义的方法 first_action()
    --]]
    entry({"admin", "example"}, firstchild(), _("Example"), 60)
    entry({"admin", "example", "first"}, template("example/example"), _("First"), 1)

    --[[
        如果配置文件/etc/config/example 存在，则创建 Example 的子节点 Second。
        cbi("example/example")表示当节点 Second 被调度时，LuCI 会将
        /usr/lib/lua/luci/model/cbi/example/example.lua 这个脚本转换成 html 页面
        发给客户端。
    --]]
    if nixio.fs.access("/etc/config/example") then
        entry({"admin", "example", "second"}, cbi("example/example"), _("Second"), 2)
    end
end
```

对于/usr/lib/lua/luci/model/cbi/下的 lua 脚本，则使用 translate()来翻译，修改/usr/lib/lua/luci/model/cbi/example/example.lua，修改后分内容如下：

```
--[[
    有关 Map 相关的用法参考 LuCI 官方文档：http://luci.subsignal.org/trac/wiki/Documentation/CBI
    Map (config, title, description)
```

```
--]]
m = Map("example", translate("Example for cbi"),
	translate("This is very simple example for cbi"))

--描述一个类型为 example 的 section
s = m:section(TypedSection, "example", translate("Example"), translate("The section of type is example"))

--允许用户添加和删除这个 section
s.addremove = true

--显示 section 的名称
s.anonymous = false

--该 section 的 num 选项
e = s:option(Value, "num", translate("Number"))

--当用户输入一个空值时，从配置文件中移除该选项
e.rmempty = true

return m
```

在虚拟机 Ubuntu 中创建一个文件 example.po，其内容如下：

```
msgid "Example"
msgstr "例子"

msgid "First"
msgstr "第一"

msgid "Second"
msgstr "第二"

msgid "Example for cbi"
msgstr "有关 cbi 的例子"

msgid "This is very simple example for cbi"
msgstr "这是有关 cbi 的一个非常简单的例子"

msgid "The section of type is example"
msgstr "类型为 example 的 section"

msgid "Number"
msgstr "数字"
```

使用 Luci 提供的工具 po2lmo 将这个 example.po 转换成.lmo 格式，该工具所在路径为：feeds/luci/build/po2lmo,操作如下所示：

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# ./feeds/luci/build/po2lmo example.po example.zh-cn.lmo
```

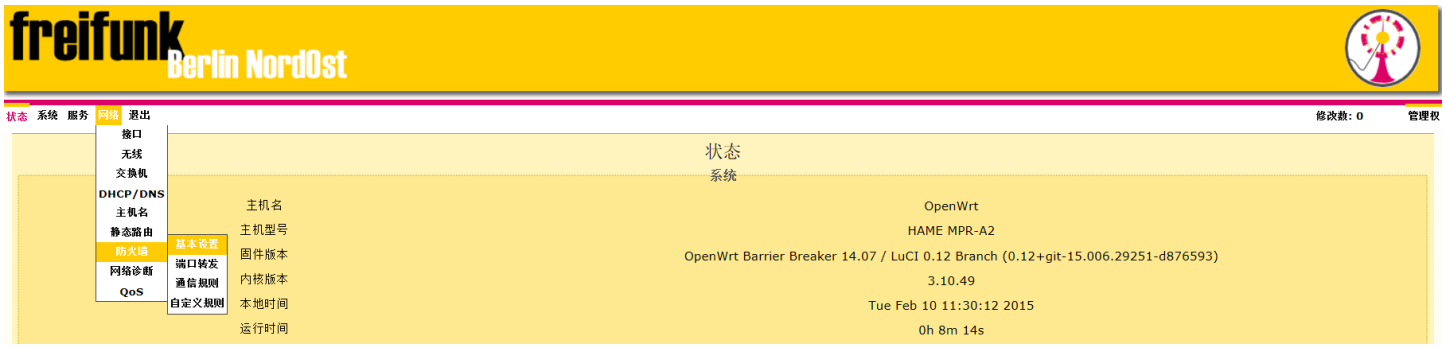
将生成的 example.zh-cn.lmo 拷贝到开发板的/usr/lib/lua/luci/i18n/目录，现在刷新页面，效果如下：

15.8 主题

Openwrt 自带了几个主题，可以在 menuconfig 中配置：

```
LuCI    --->
        4. Themes    --->
                -*- luci-theme-bootstrap
                <*> luci-theme-freifunk-bno
                <*> luci-theme-freifunk-generic
                <*> luci-theme-openwrt
```

在网页配置中，选择“系统->系统”，然后选择“语言和界面”，选择一个主题，比如选择“Freifunk_BNO”，效果如下：



与主题相关的代码存放在/usr/lib/lua/luci/view/themes/目录以及/www/luci-static/目录，一个主题占一个目录，比如 bootstrap 这个主题有/usr/lib/lua/luci/view/themes/bootstrap/和/www/luci-static/bootstrap/这两个目录。

/usr/lib/lua/luci/view/themes/bootstrap/目录下有两个 html 文件，分别是 header.htm 和 footer.htm，分别用来控制网页的顶部和底部的显示方式。

上面的显示方式由 header.htm 控制

Powered by LuCI 0.12 Branch (0.12+git-15.006.29251-d876593) OpenWrt Barrier Breaker 14.07

上面的显示方式由 footer.htm 控制

/www/luci-static/bootstrap/下包含一些 css 文件和 js 文件，用来控制整体界面以及各个控件的显示样式。
添加主题，需要对 html、js、css 这些知识非常熟悉。这些一般由专门的工程师做。

15.9 在 Openwrt 源码中添加 LuCI 模块

仿照 LuCI 自带的模块添加，创建目录 feeds/luci/applications/luci-example/并添加代码，代码组织结构如下所示：

```
root@zjh-vm:/home/work/openwrt/barrier_breaker/feeds/luci/applications# tree luci-example/
luci-example/
├── luasrc
│   ├── controller
│   │   └── example.lua
│   ├── model
│   │   └── cbi
│   │       └── example
│   │           └── example.lua
│   └── view
│       └── example
│           └── example.htm
└── Makefile

7 directories, 4 files
```

其中的 Makefile 内容如下所示：

```
PO = example

include ../../build/config.mk
include ../../build/module.mk
```

两个 example.lua 对应 15.5 节的代码, example.htm 对应 15.4 节的代码。

然后修改 feeds/luci/contrib/package/luci/Makefile，添加一行代码，如下所示：

```
$(eval $(call application,firewall,Firewall and Portforwarding application,\
	+PACKAGE_luci-app-firewall:firewall))

$(eval $(call application,qos,Quality of Service configuration module,\
	+PACKAGE_luci-app-qos:qos-scripts))

$(eval $(call application,commands,LuCI Shell Command Module))
$(eval $(call application,example,Example for LuCI))
```

创建翻译文件 feeds/luci/po/zh_CN/example.po，其内容对应 15.7 节的 example.po

现在配置 menuconfig

```
LuCI    --->
        3. Applications    --->
                <*> luci-app-example
```

首先删除 luci 的编译目录，否则翻译文件不能自动安装。

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# rm build_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/luci* -r
```

重新编译 Openwrt，更新固件。使用浏览器访问路由器：

只有第一个子节点，没有创建第二个子节点，是因为还没创建对应的配置文件 example，下面创建一个

```
root@OpenWrt:/# touch /etc/config/example
```

然后刷新网页，效果如下：

有关配置文件/etc/config/example 和启动脚本/etc/init.d/example 的创建，可以参照 6.2 节添加软件包实例。

15.10 开启 LuCI 缓存

LuCI 模块添加完成，并且调试通过后，现在开启 LuCI 缓存，提升速度。

```
root@OpenWrt:/# uci set luci.ccache.enable=1
root@OpenWrt:/# uci commit
```

16 支持 U 盘

```
Base system    --->
        <*> block-mount
Kernel modules    --->
        Filesystems    --->
                <*> kmod-fs-ext4
```

<*> kmod-fs-msdos
<*> kmod-fs-ntfs
<*> kmod-fs-vfat
Native Language Support --->
<*> kmod-nls-cp437
<*> kmod-nls-iso8859-1
USB Support --->
<*> kmod-usb-ohci
<*> kmod-usb-storage
<*> kmod-usb-storage-extras
<*> kmod-usb-uhci
<*> kmod-usb2
<*> kmod-usb3

重新编译 Openwrt，更新固件。

U 盘挂载的配置文件为/etc/config/fstab，向该配置文件添加一个挂载点

```
config mount
        option enabled '1'
        option device '/dev/sda1'
        option target '/udisk'
        option fstype 'vfat'
```

有关该配置文件的详细说明，参考 Openwrt 官方文档：http://wiki.openwrt.org/doc/uci/fstab

使配置生效

```
root@OpenWrt:/# /etc/init.d/fstab reload
```

现在插入 U 盘，然后执行 df 命令查看系统挂载情况

```
root@OpenWrt:/# df
Filesystem           1K-blocks      Used Available Use% Mounted on
rootfs                    3968       292      3676   7% /
/dev/root                 3072      3072         0 100% /rom
tmpfs                    14748       960     13788   7% /tmp
/dev/mtdblock5            3968       292      3676   7% /overlay
overlayfs:/overlay        3968       292      3676   7% /
tmpfs                      512         0       512   0% /dev
/dev/sda1              7133312   3195656   3937656  45% /udisk
```

可以看到，Openwrt 已经自动将 U 盘设备/dev/sda1 挂载到/udisk 目录。

也可以使用网页配置

17 opkg

opkg 是一个轻量快速的套件管理系统，目前已成为 Opensource 界嵌入式系统标准。常用于路由器、交换机等嵌入式设备中，用来管理软件包的安装升级与下载。

```
root@OpenWrt:/# opkg
opkg must have one sub-command argument
usage: opkg [options...] sub-command [arguments...]
where sub-command is one of:

Package Manipulation:
        update                  Update list of available packages
        upgrade <pkgs>          Upgrade packages
        install <pkgs>          Install package(s)
        configure <pkgs>        Configure unpacked package(s)
        remove <pkgs|regexp>    Remove package(s)
        flag <flag> <pkgs>      Flag package(s)
         <flag>=hold|noprune|user|ok|installed|unpacked (one per invocation)

Informational Commands:
        list                    List available packages
        list-installed          List installed packages
        list-upgradable         List installed and upgradable packages
        list-changed-conffiles  List user modified configuration files
        files <pkg>             List files belonging to <pkg>
        search <file|regexp>    List package providing <file>
        find <regexp>           List packages whose name or description matches <regexp>
        info [pkg|regexp]       Display all info for <pkg>
        status [pkg|regexp]     Display all status for <pkg>
```

```
        download <pkg>                  Download <pkg> to current directory
        compare-versions <v1> <op> <v2>
                                        compare versions using <= < > >= = << >>
        print-architecture              List installable package architectures
        depends [-A] [pkgname|pat]+
        whatdepends [-A] [pkgname|pat]+
        whatdependsrec [-A] [pkgname|pat]+
        whatrecommends[-A] [pkgname|pat]+
        whatsuggests[-A] [pkgname|pat]+
        whatprovides [-A] [pkgname|pat]+
        whatconflicts [-A] [pkgname|pat]+
        whatreplaces [-A] [pkgname|pat]+

Options:
        -A                              Query all packages not just those installed
        -V[<level>]                     Set verbosity level to <level>.
        --verbosity[=<level>]           Verbosity levels:
                                                0 errors only
                                                1 normal messages (default)
                                                2 informative messages
                                                3 debug
                                                4 debug level 2
        -f <conf_file>                  Use <conf_file> as the opkg configuration file
        --conf <conf_file>
        --cache <directory>             Use a package cache
        -d <dest_name>                  Use <dest_name> as the the root directory for
        --dest <dest_name>              package installation, removal, upgrading.
                                          <dest_name> should be a defined dest name from
                                          the configuration file, (but can also be a
                                          directory name in a pinch).
        -o <dir>                        Use <dir> as the root directory for
        --offline-root <dir>            offline installation of packages.
        --add-arch <arch>:<prio>        Register architecture with given priority
        --add-dest <name>:<path>        Register destination with given path

Force Options:
        --force-depends                 Install/remove despite failed dependencies
        --force-maintainer              Overwrite preexisting config files
        --force-reinstall               Reinstall package(s)
        --force-overwrite               Overwrite files from other package(s)
        --force-downgrade               Allow opkg to downgrade packages
        --force-space                   Disable free space checks
        --force-postinstall             Run postinstall scripts even in offline mode
        --force-remove      Remove package even if prerm script fails
        --force-checksum                Don't fail on checksum mismatches
        --noaction                      No action -- test only
        --download-only No action -- download only
        --nodeps                        Do not follow dependencies
```

```
        --nocase                    Perform case insensitive pattern matching
        --force-removal-of-dependent-packages
                                      Remove package and all dependencies
        --autoremove                Remove packages that were installed
                                      automatically to satisfy dependencies
        -t                          Specify tmp-dir.
        --tmp-dir                   Specify tmp-dir.

regexp could be something like 'pkgname*' '*file*' or similar
e.g. opkg info 'libstd*' or opkg search '*libop*' or opkg remove 'libncur*'
```

17.1 安装软件包

```
opkg update
opkg install <package>
```

opkg update：更新可用软件包列表，这个列表保存在临时目录/tmp/中，重启就会被丢失，因此在安装软件之前应确保已经更新了可用软件包列表（安装本地软件包除外）。

opkg install <package> ：安装一个软件包，也可用同时安装多个软件包（多个软件包之间以空格分隔）。

17.2 删除软件包

```
root@OpenWrt:/# opkg remove helloworld
Removing package helloworld from root...
```

17.3 查询已安装软件包

```
root@OpenWrt:/# opkg list
base-files - 156-r44162
block-mount - 2014-06-22-e0430f5c62f367e5a8e02755412977b02c3fc45e
busybox - 1.22.1-3
dnsmasq - 2.71-4
dropbear - 2014.63-2
firewall - 2014-09-19
fstools - 2014-06-22-e0430f5c62f367e5a8e02755412977b02c3fc45e
helloworld - 1.0
……
```

opkg list 仅显示软件包名称和版本。

```
root@OpenWrt:/# opkg info
Package: kmod-usb-storage
Version: 3.10.49-1
Depends: kernel (= 3.10.49-1-611ed7481859235022353b946a131359), kmod-scsi-core, kmod-usb-core
Status: install hold installed
Architecture: ramips_24kec
Installed-Time: 1423584550
```

```
Package: kmod-sched-connmark
Version: 3.10.49-1
Depends: kernel (= 3.10.49-1-611ed7481859235022353b946a131359), kmod-sched-core, kmod-ipt-core, kmod-ipt-conntrack-extra
Status: install hold installed
Architecture: ramips_24kec
Installed-Time: 1423584550
……
```

opkg info 将显示所有信息。

opkg list 和 opkg info 还可以只显示当软件包的信息。例如：

```
root@OpenWrt:/# opkg list helloworld
helloworld - 1.0
root@OpenWrt:/# opkg info helloworld
Package: helloworld
Version: 1.0
Depends: libc, libubox
Status: install user installed
Architecture: ramips_24kec
Installed-Time: 1423891709
```

17.4 更新软件包

```
root@OpenWrt:/# opkg upgrade helloworld
```

17.5 安装目的地

比如在路由器上连接了一个 U 盘或者移动硬盘，挂载目录为/udisk/。我们可以把软件包安装到 U 盘或者移动硬盘。

首先修改 opkg 配置文件/etc/opkg.conf，添加一行 dest usb /udisk

```
dest root /
dest usb /udisk
dest ram /tmp
lists_dir ext /var/opkg-lists
```

/udisk 表示 U 盘或者移动硬盘的挂载目录。

然后安装软件，例如安装 helloworld

```
root@OpenWrt:/# opkg -d usb install helloworld_1.0_ramips_24kec.ipk
Installing helloworld (1.0) to usb...
Configuring helloworld.
```

程序已经安装到/udisk/bin/helloworld

修改/etc/profile，配置环境变量，添加两行

```
export PATH=/udisk/bin:/udisk/sbin:/udisk/usr/bin:/udisk/usr/sbin:/udisk/usr/local/bin:/usr/local/sbin:$PATH
export LD_LIBRARY_PATH=/udisk/lib:/udisk/usr/lib:/udisk/usr/local/lib:$LD_LIBRARY_PATH
```

然后使配置生效

```
root@OpenWrt:/# . /etc/profile
  _______                     ________        __
 |       |.-----.-----.-----.|  |  |  |.----.|  |_
 |   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
 |_______||   __|_____|__|__||________||__|  |____|
          |__| W I R E L E S S   F R E E D O M
-----------------------------------------------------
 BARRIER BREAKER (Barrier Breaker, r44162)
-----------------------------------------------------
  * 1/2 oz Galliano              Pour all ingredients into
  * 4 oz cold Coffee             an irish coffee mug filled
  * 1 1/2 oz Dark Rum             with crushed ice. Stir.
  * 2 tsp. Creme de Cacao
-----------------------------------------------------
```

18 LED

LED 驱动基于 GPIO 驱动。在 target/linux/ramips/dts/rt5350.dtsi 中配置了 GPIO，如下所示：

```
gpio0: gpio@600 {
                compatible = "ralink,rt5350-gpio", "ralink,rt2880-gpio";
                reg = <0x600 0x34>;

                resets = <&rstctrl 13>;
                reset-names = "pio";

                interrupt-parent = <&intc>;
                interrupts = <6>;

                gpio-controller;
                #gpio-cells = <2>;

                ralink,gpio-base = <0>;
                ralink,num-gpios = <22>;
                ralink,register-map = [ 00 04 08 0c
                                        20 24 28 2c
                                        30 34 ];
        };
```

在 target/linux/ramips/dts/MPRA2.dts 中配置了 LED，如下所示：

```
gpio-leds {
        compatible = "gpio-leds";
        system {
                label = "hame:blue:system";
                gpios = <&gpio0 20 1>;
        };
        power {
```

```
            label = "hame:red:power";
            gpios = <&gpio0 17 1>;
        };
    };
```

这里配置了 2 个基于 GPIO 的 LED。

另外，在驱动 rt2x00leds.c 中的 rt2x00leds_register 函数中注册了 3 个系统 LED。在开发板上可以通过 sysfs 操作，如下所示：

```
root@OpenWrt:/sys/class/leds# ls
hame:blue:system            rt2800soc-phy0::assoc       rt2800soc-phy0::radio
hame:red:power              rt2800soc-phy0::quality
```

其中 rt2800soc-phy0::radio 对应系统的 WLAN LED。我们进入这个文件夹

```
root@OpenWrt:/sys/class/leds# cd rt2800soc-phy0\:\:radio/
root@OpenWrt:/sys/devices/10180000.wmac/leds/rt2800soc-phy0::radio#
```

然后执行如下操作可以点亮和熄灭 LED

```
root@OpenWrt:/sys/devices/10180000.wmac/leds/rt2800soc-phy0::radio# echo 1 > brightness
root@OpenWrt:/sys/devices/10180000.wmac/leds/rt2800soc-phy0::radio# echo 0 > brightness
```

我们可以配置内核驱动，使这些 LED 支持特定的触发方式：比如心跳、PWM、定时器、netdev、phy0rx 等。

```
Kernel modules    --->
    LED modules    --->
        <*> kmod-leds-gpio
        <*> kmod-ledtrig-heartbeat
        <*> kmod-ledtrig-netdev
```

在开发板上执行如下操作，查看 LED 支持哪些触发方式

```
root@OpenWrt:/sys/devices/10180000.wmac/leds/rt2800soc-phy0::radio# cat trigger
[none] pwmtimer timer default-on netdev heartbeat phy0rx phy0tx phy0assoc phy0radio
```

[]表示当前触发方式。

设置为心跳触发方法：

```
root@OpenWrt:/sys/devices/10180000.wmac/leds/rt2800soc-phy0::radio# echo heartbeat > trigger
```

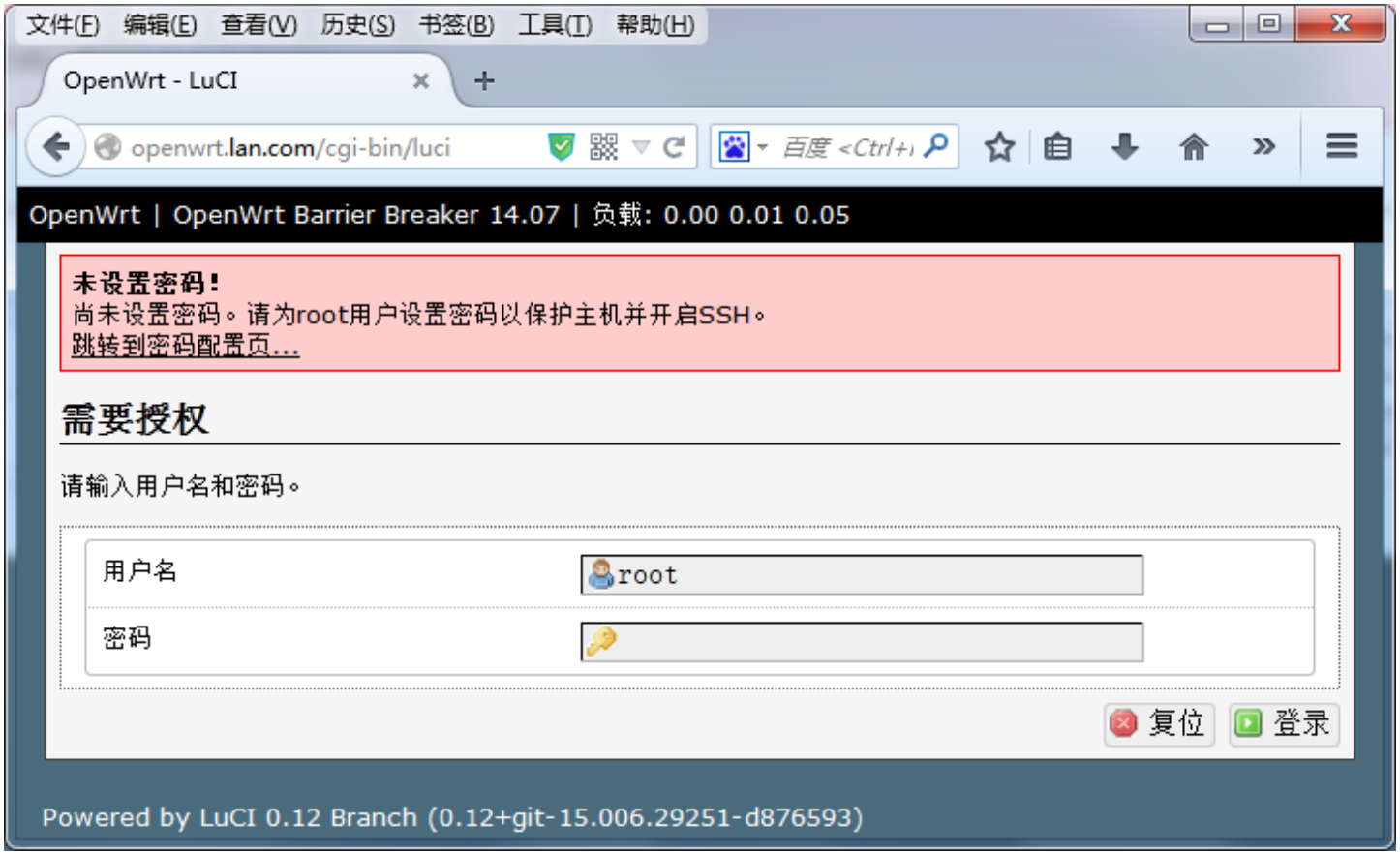
可以看到 LED 开始按照固定的频率闪烁。心跳触发一般用来指示系统是否在工作

另外，Openwrt 提供了一个 UCI 接口，用来配置 LED，其对应的配置文件为：/etc/config/system，例如：

```
config led 'led_wifi_led'
        option name 'wifi'
        option trigger 'none'
        option sysfs 'rt2800soc-phy0::assoc'
        option default '0'
```

该配置详细说明参考 Openwrt 官方文档：http://wiki.openwrt.org/doc/uci/system

下面直接通过 Web 来配置 WLAN LED，在浏览器中输入：http://openwrt.lan.com 来方法开发板（其中 openwrt 对应开发板的主机名，lan.com 在/etc/config/dhcp 中设置，具体参考 5.1 节）。



进入“系统->LED 配置”界面，最终设置如下所示：

然后保存和应用。

现在只要有设备连接开发板的 WIFI 或者有数据经过网络接口 WLAN0，这个 WAN LED 就会闪烁。

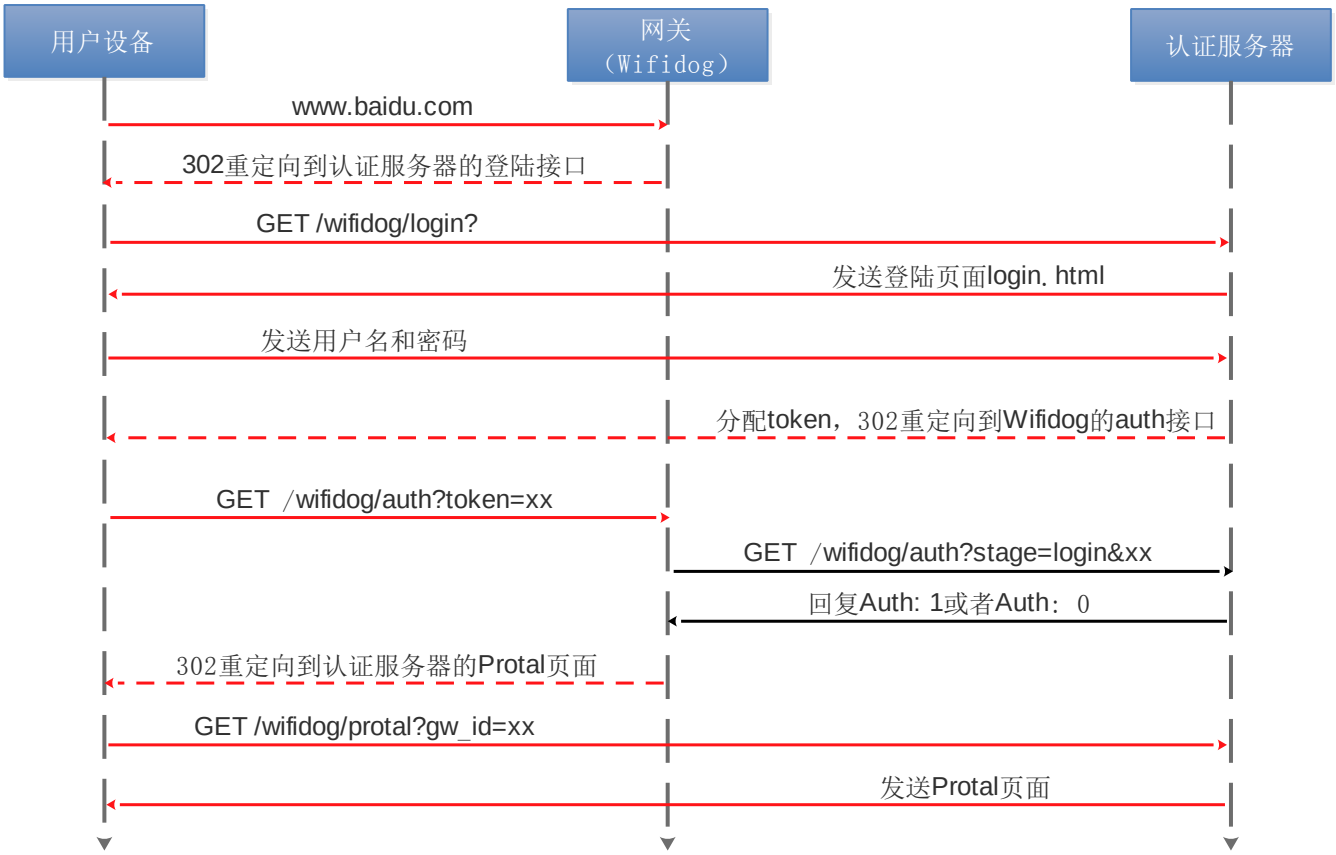
19 上网认证

19.1 概述

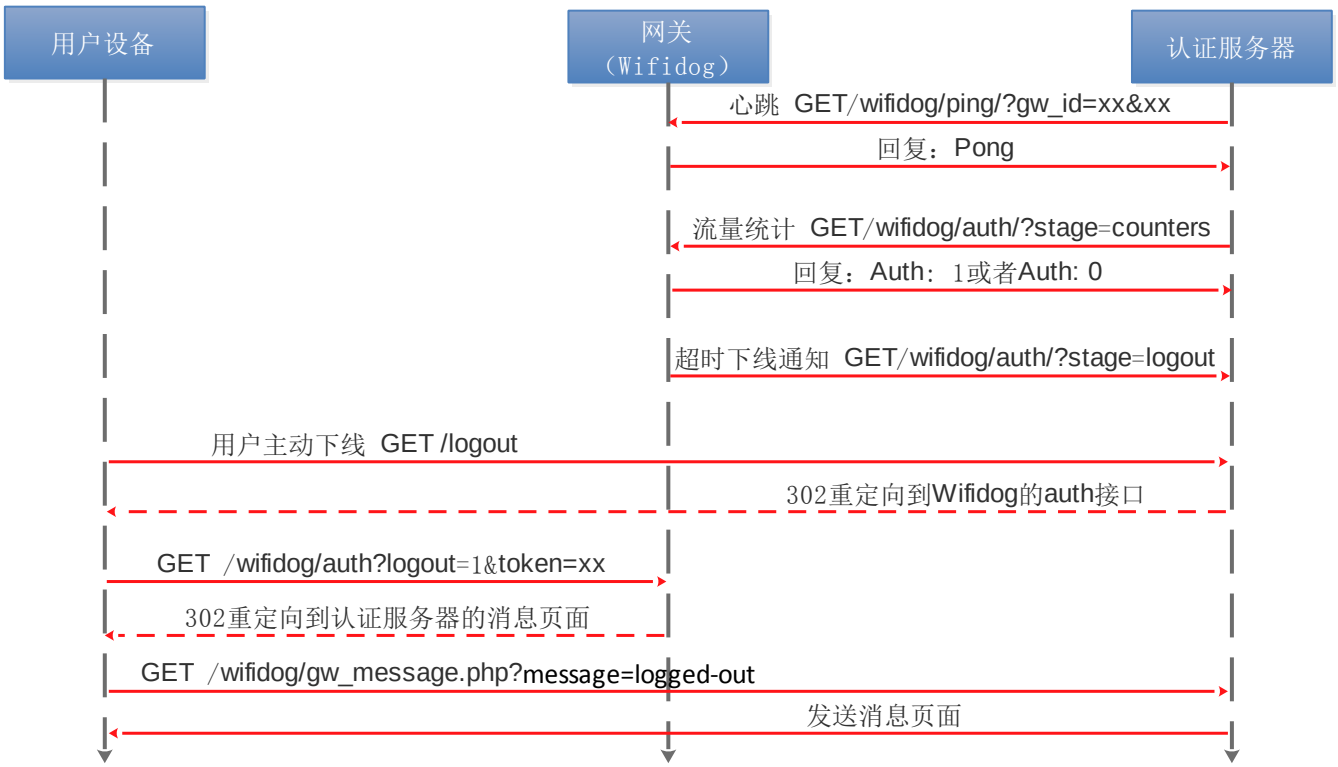
当我们在机场、咖啡厅等有公共 WIFI 的地方用手机连接 WIFI 上网时，首先会弹出一个登陆认证页面，这里的认证方式有多种，比如短信验证、QQ 验证等。上网认证需要一个认证客户端和一个认证服务器。认证客户端运行在路由器中，认证服务器一般运行在 PC 机上，在认证服务器中可以实现计费等功能。本文使用 wifidog 作为认证客户端。Wifidog 不仅可以认证 Wifi 连接，而且可以认证有线连接。

在 wifidog 进程启动后，它首先建立了一系列的防火墙规则：

- 接受来自 LAN 口的所有目标 IP 为内网的流量（包括目标 IP 为路由器 LAN 口 IP 的流量）
- 转发所有目标 IP 为认证服务器的 IP 的流量
- 将其它所有到外网的 TCP 协议 80 端口的流量重定向到路由器的 2060 端口（Wifidog 内部创建了一个端口为 2060 的 HTTP 服务器）
- 开放 DNS 端口（UDP 53、TCP 53）和 DHCP 端口（UDP 67、TCP 67）
- 转发其它所有被标记 0x01 和 0x02 的流量（通过认证的流量都被 Wifidog 使用 iptables 工具标记为 0x01 或者 0x02，具体参考 Wifidog 源码）
- 拒绝转发其它所有流量



Wifidog 工作流程图：登陆和认证



Wifidog 工作流程图：心跳、流量统计、下线

19.2 Wifidog 接口协议

19.2.1 网关心跳

网关向认证服务器请求的格式：

方向：网关->认证服务器

```
GET /wifidog/ping/?gw_id=xx&sys_uptime=xx&sys_memfree=xx&sys_load=xx&wifidog_uptime=xx HTTP/1.0
```

gw_id：网关的唯一标识，可以在 Wifidog 的配置文件配置

sys_uptime：网关运行时间

sys_memfree：网关可用内存

sys_load：网关负载百分比

wifidog_uptime：Wifidog 进程运行时间

心跳间隔可在 Wifidog 配置文件中配置，默认为 60s

认证服务器回复格式：

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 4

Pong
```

19.2.2 设备登陆及认证

当用户访问 www 服务时，网关将用户请求重定向到认证服务器的 login 接口，其请求格式如下：
方向：用户设备->认证服务器

```
GET /wifidog/login/?gw_address=x.x.x.x&gw_port=xx&gw_id=xx&mac=x:x:x:x:x:x&url=http%3A//www.xx.com/ HTTP/1.1
```

gw_address：网关 LAN 口 IP 地址
gw_port：Wifidog 监听端口
gw_id：网关唯一标识
mac：网关 LAN 口 MAC 地址
url：用户正在访问的域名

认证服务器收到该请求后，首先解析 url 中所带的参数，然后向用户发送登陆页面，用户在登陆页面输入验证信息后，点击提交将验证信息发回认证服务器，然后认证服务器对用户的输入的验证信息进行验证，如果验证通过，则为该用户分配一个唯一的标识 token，然后以分配的 token 作为 url 参数将用户的请求重定向到网关的 auth 接口，格式如下：
方向：用户设备->网关

```
GET /wifidog/auth?token=xx HTTP/1.1
```

当网关收到用户的请求后，首先解析 url 中的参数 token 以及用户的 IP 和 MAC 地址，创建一个 t_client 结构体实例，并将其添加到全局链表 firstclient 中。t_client 结构体保存了客户的 IP、MAC、token、连接状态等信息。然后网关向认证服务器请求认证结果，格式如下：
方向：网关->认证服务器

```
GET /wifidog/auth/?stage=login&ip=x.x.x.x&mac=x:x:x:x:x:x&token=xx&incoming=0&outgoing=0&gw_id=xx HTTP/1.0
```

stage：认证阶段。这里为 login 表示第一次登陆
ip：用户设备的 IP 地址
mac：用户设备的 MAC 地址
token：由认证服务器分配的唯一标识
incoming：入口流量，一开始为 0
outgoing：出口流量，一开始为 0
gw_id：网关唯一标识

当认证服务器收到用户的认证请求后，根据认证阶段以及其它信息决定是否通过认证。其回复格式如下：

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 7

Auth: x
```

Auth: 1 表示通过认证，Auth: 0 表示拒绝。（注意：Auth:后面有一个空格，具体参考 Wifidog 源码）

当网关收到认证服务器的认证结果后，如果通过认证，则将之前用户的请求重定向到认证服务器的门户页面，并且通过 iptables 添加防火墙规使用户能够上网，否则将其重定向到认证服务器的认证失败消息页面。格式如下：
方向：用户->认证服务器

```
GET /wifidog/portal/?gw_id=xx HTTP/1.1
```

```
GET /wifidog/gw_message.php?message=denied HTTP/1.1
```

19.2.3 流量统计

网关会周期性的向认证服务器发送用户的流量消息，其请求格式如下：
方向：网关->认证服务器

```
GET /wifidog/auth/?stage=counters&ip=x.x.x.x&mac=x:x:x:x:x:x&token=xx&incoming=xx&outgoing=xx&gw_id=xx HTTP/1.0
```

stage：认证阶段，这里为 counters 表示统计流量
ip：用户设备的 IP 地址
mac：用户设备的 MAC 地址
token：由认证服务器分配的唯一标识
incoming：入口流量
outgoing：出口流量

当认证服务器收到流量统计信息后，决定是否继续允许用户上网，其回复格式如下：
认证服务器回复格式：

```
HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 7

Auth: x
```

19.2.4 设备下线（主动）

网关提供的接口格式如下：
方向：用户设备->网关（实际可以为用户访问认证服务器的主动下线接口，认证服务器将其重定向到网关的下线接口）

```
GET /wifidog/auth?logout=1&token=xx HTTP/1.0
```

当网关收到该请求后，将与指定 token 相关联的客户踢下线，并将请求重定向到认证服务器的设备下线信息页面，格式如下：
方向：用户设备->认证服务器

```
GET /wifidog/gw_message.php?message=logged-out HTTP/1.0
```

19.2.5 设备下线（超时）

当用户设备在一定时间（该时间可在 Wifidog 配置文件中配置，默认为 300 秒）内未连接网关后，Wifidog 将主动修改防火墙规则，将用户踢下线，并通知认证服务器，其请求格式如下：
方向：网关->认证服务器

```
GET /wifidog/auth/?stage=logout&ip=x.x.x.x&mac=x:x:x:x:x:x&token=xx&incoming=xx&outgoing=xx&gw_id=xx HTTP/1.0
```

19.3 Wifidog 配置

在 Wifidog 的配置文件为/etc/wifidog.conf

19.3.1 网关 ID（可选）

用于区分使用同一个认证服务器的多个网关。如果未指定，则默认使用 GatewayInterface 网络接口的 MAC 地址作为其 ID。
格式：
GatewayID xxx
其中 xxx 为网关 ID，可以是除空格外的任意可读字符的组合。

19.3.2 外部网络接口（可选）

定义网关到 Inernet 或更大的局域网的网络接口，俗称外网接口。通常由 Wifidog 自动探测。
格式：
ExternalInterface eth0.2

19.3.3 网关接口（必须）

定义网关内部局域网接口。
格式：
GatewayInterface br-lan

19.3.4 网关内部局域网 IP 地址（可选）

通常由 Wifidog 自动探测。
格式：
GatewayAddress 192.168.1.1

19.3.5 Wifidog 消息页面（可选）

指定一个 html 文件，用于 Wifidog 输入系统错误、客户连接状态等消息。该 html 文件中任何$ $title、 $message 和$node 变量都会被 Wifidog 替换。默认的 html 消息文件为:/etc/wifidog-msg.html
格式：
HtmlMessageFile /opt/wifidog/etc/wifidog-.html

19.3.6 认证服务器（必须，可重复）

该选项定义认证服务器的 IP 地址或者域名、端口（SSL、HTTP）、cgi 路径、cgi 接口。当定义多个认证服务器时，Wifidog 将依次尝试每一个认证服务器，直到收到响应为止。
格式：

```
AuthServer {
    Hostname                www.auth.com
    SSLAvailable            yes
    SSLPort                 433
    HTTPPort                 80
    Path                     /wifidog/
```

```
    LoginScriptPathFragment     login/?
    PortalScriptPathFragment    portal/?
    MsgScriptPathFragment       gw_message.php?
    PingScriptPathFragment      ping/?
    AuthScriptPathFragment      auth/?
}
```

其中只有 Hostname 为必须，其它都为可选。

Hostname：可以指定为认证服务器的域名或者 IP 地址。

SSLAvailable：认证服务器是否有 ssl，可取值为：yes、no，默认为 no

SSLPort：当 SSLAvailable 为 yes 时，该选项可以指定认证服务器的 ssl 端口，默认为 443

HTTPPort：指定认证服务器的 HTTP 端口，默认为 80

Path：指定认证服务器的认证接口 cgi 根路径，默认为：/wifidog/

LoginScriptPathFragment：指定认证服务器的登陆接口，默认为：login/?

PortalScriptPathFragment：指定认证服务器的门户页面接口，默认为：portal/?

MsgScriptPathFragment：指定认证服务器的消息接口（下线、认证拒绝），默认为：gw_message.php?

PingScriptPathFragment：指定认证服务器的心跳接口，默认为：ping/?

AuthScriptPathFragment：指定认证服务器的认证接口，默认为：auth/?

19.3.7 是否后台运行（可选）

是否后台运行，即是否以守护进程运行，默认以守护进程运行。

格式：

Daemon 1

或者

Daemon 0

19.3.8 Wifidog 监听端口（可选）

指定 Wifidog 内置的 http 服务器的监听端口，默认为 2060.

格式：

GatewayPort 2060

19.3.9 超时检测间隔、心跳间隔、流量统计间隔（可选）

这 3 个间隔使用同一配置选项，单位为妙。超时检测间隔即多长时间检测一次用户设备是否接入网关。

格式：

CheckInterval 60

19.3.10 超时时间（可选）

该选项决定用户设备多长时间未接入网关后，Wifidog 就将该用户设备踢下线。默认为 5。

格式：

ClientTimeout 5

实际时间为：CheckInterval * ClientTimeout

19.3.11 白名单（可选）

该选项指定不需要认证的 MAC 地址。多个 MAC 地址之间以逗号分隔。
格式：
TrustedMACList 00:00:DE:AD:BE:AF,00:00:C0:1D:F0:0D

19.3.12 防火墙规则（必须）

格式：

```
FirewallRuleSet xx {
    FirewallRule xx
    ……
}
```

防火墙规则由多个 FirewallRuleSet 选项组成，每一个 FirewallRuleSet 选项又由多个 FirewallRule 选项组成，FirewallRule 指定具体的规则（允许（阻止）某协议（端口））。
总共有 5 个 FirewallRuleSet 选项：locked-users、global、validating-users、known-users、unknown-users。
Wifidog 将这些规则按照一定的顺序添加到 Linux 防火墙 Filter 表，最终 Linux 防火墙处理的顺序为：
locked-users、global、validating-users、known-users、unknown-users。

locked-users 匹配那些被 Wifidog 标记了 0x254 的流量
global 匹配所有流量
validating-users 匹配被 Wifidog 标记了 0x01 的流量（认证服务器返回 Auth: 5）
known-users 匹配被 Wifidog 标记了 0x02 的流量（认证服务器返回 Auth: 1）
unknown-users 匹配其它流量

FirewallRule 格式：
FirewallRule (block|drop|allow|log|ulog) [(tcp|udp|icmp) [port X]] [to IP/CIDR]

例 1：丢弃通过认证的流量中的目的 IP 为 192.168.0.1 的 ICMP 包

```
FirewallRuleSet known-users {
    FirewallRule drop icmp to 192.168.0.1
    FirewallRule allow to 0.0.0.0/0
}
```

例 2：开放 DNS 和 DHCP 端口

```
FirewallRuleSet unknown-users {
    FirewallRule allow udp port 53
    FirewallRule allow tcp port 53
    FirewallRule allow udp port 67
    FirewallRule allow tcp port 67
}
```

19.4 实例（认证服务器）

首先配置 Openwrt 使能编译 wifidog 软件包

```
Network  --->
    Captive Portals  --->
        <*> wifidog
```

然后单独编译 Wifidog

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# make package/feeds/oldpackages/wifidog/compile V=s
```

然后将生成的
bin/ramips/packages/oldpackages/wifidog_20130917-440445db60b0c3aff528ea703a828b0567293387_ramips_24kec.ipk
拷贝到路由器，使用 opkg 安装。

参考 19.3 节配置 Wifidog，我的配置如下：

```
# wifidog config file:/etc/wifidog.conf
GatewayInterface br-lan

AuthServer {
    Hostname 192.168.10.105
    MsgScriptPathFragment gw_message/?
}

FirewallRuleSet known-users {
    FirewallRule allow to 0.0.0.0/0
}

FirewallRuleSet unknown-users {
    FirewallRule allow udp port 53
    FirewallRule allow tcp port 53
    FirewallRule allow udp port 67
    FirewallRule allow tcp port 67
}
```

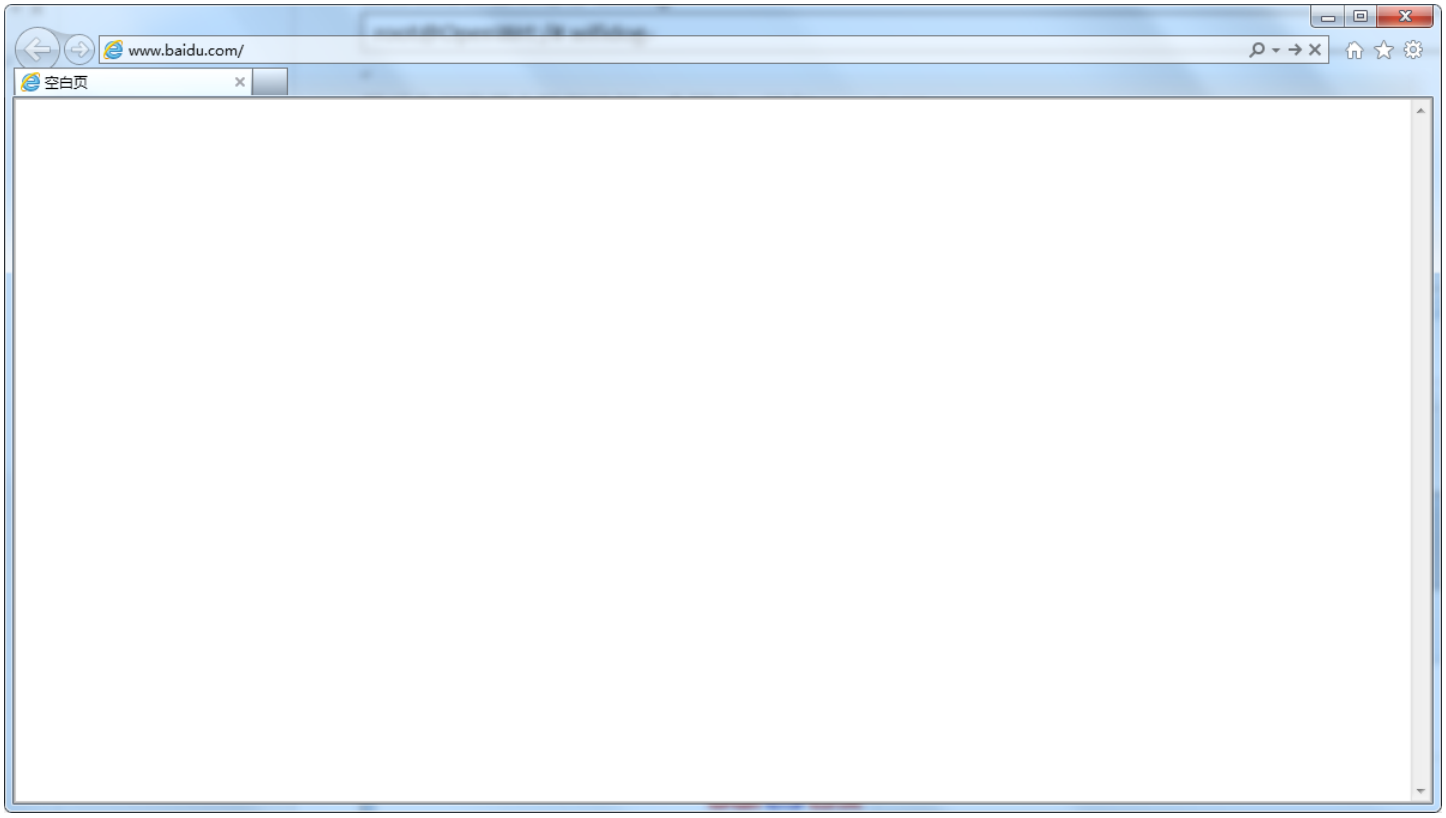
然后在路由器上运行 wifidog

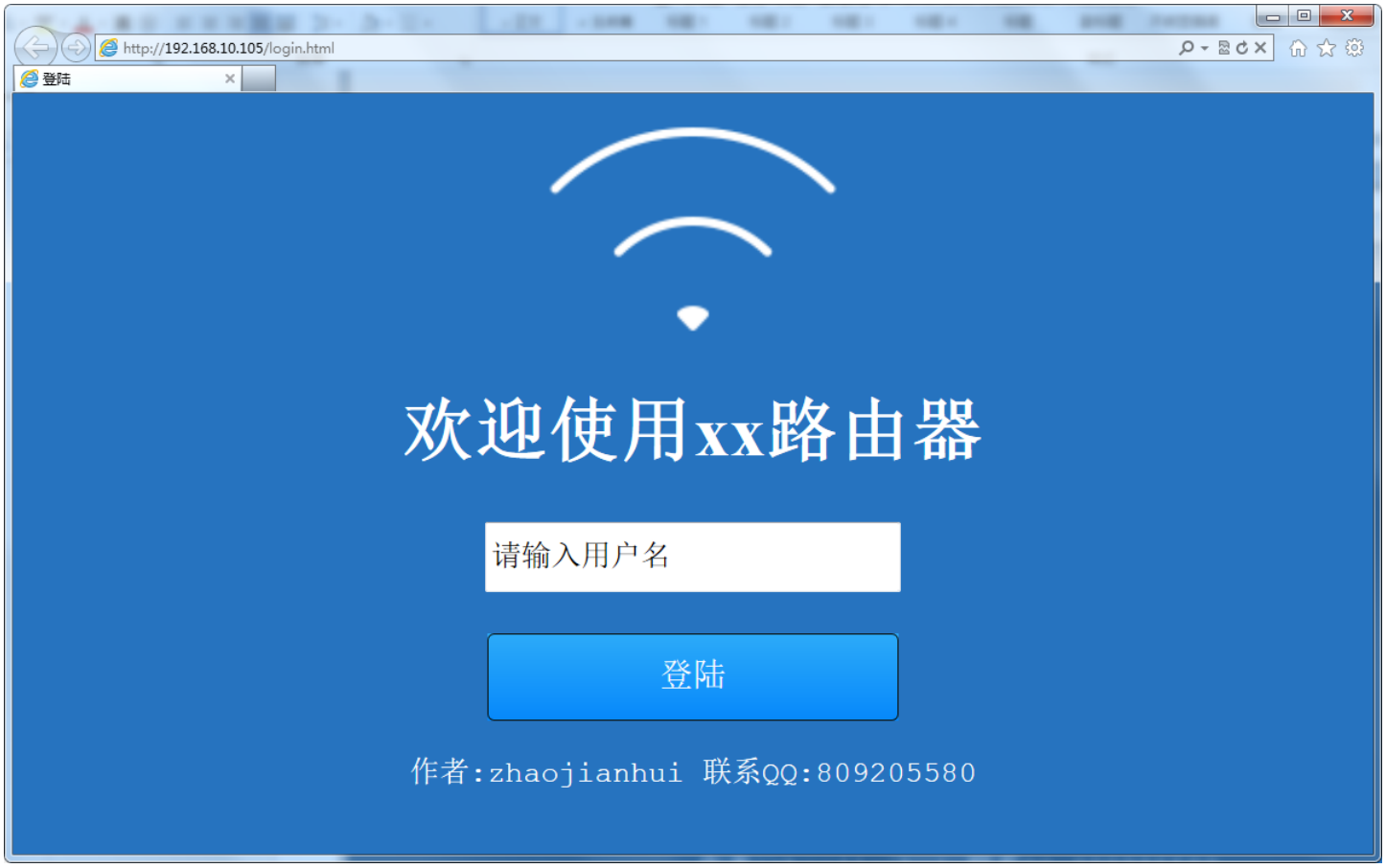
```
root@OpenWrt:/# wifidog
```

根据上面的配置，我自己写了一个简单的认证服务器（wifidog_authserver.tar.bz2）
源码下载地址：http://pan.baidu.com/s/1hqH1JBQ
下载源码后，将其拷贝到虚拟机中并解压，然后执行 make 编译，接着运行里面的 server，如下所示：

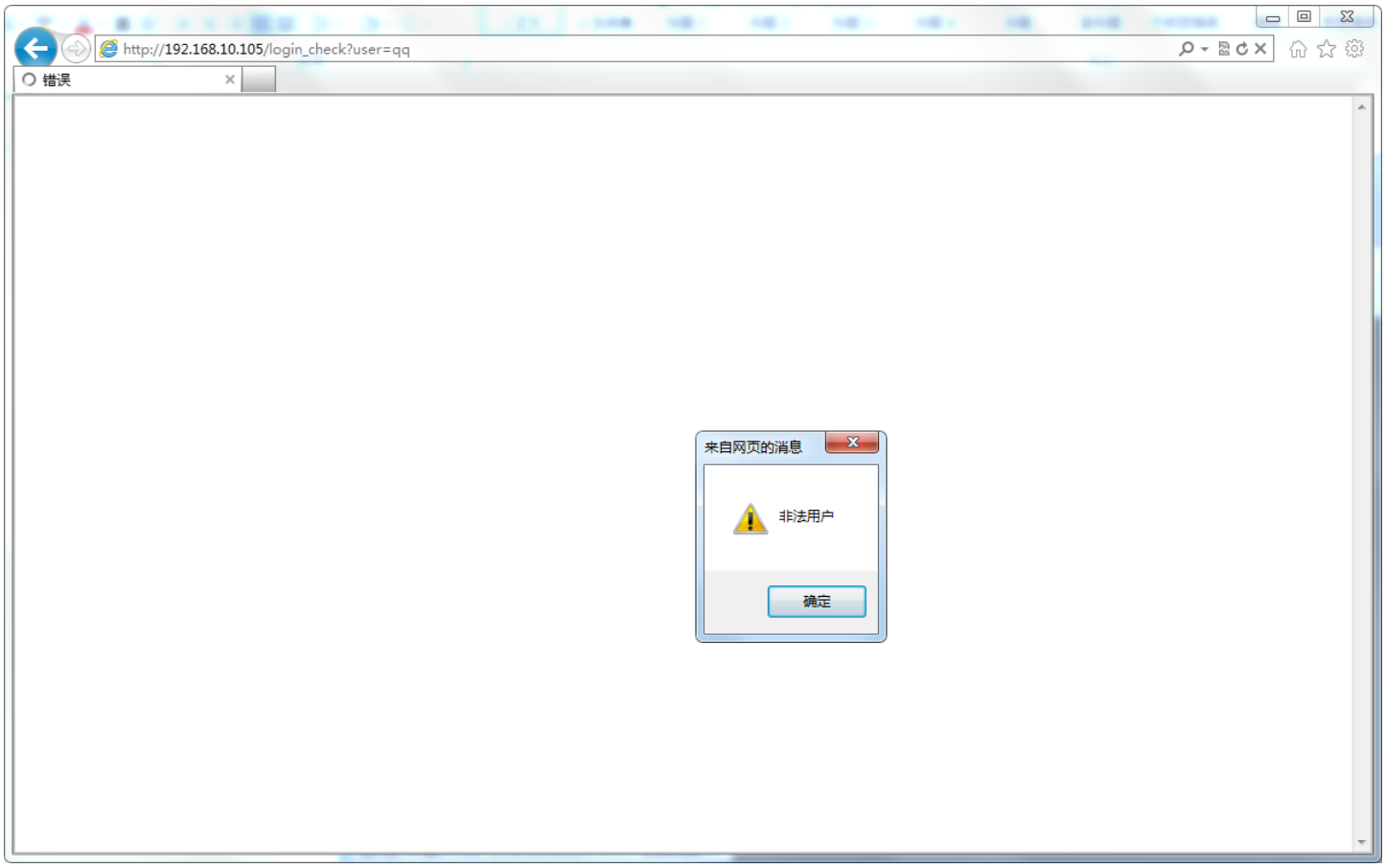
```
root@zjh-vm:/home/work# tar jxvf wifidog_authserver.tar.bz2
root@zjh-vm:/home/work# cd wifidog_authserver
root@zjh-vm:/home/work/wifidog_authserver# make
root@zjh-vm:/home/work/wifidog_authserver# ./server
debug[http_server_start|1539]    libhttp 1.6 , authored by zjh
```

然后在浏览器中随便访问一个网站，比如 www.baidu.com

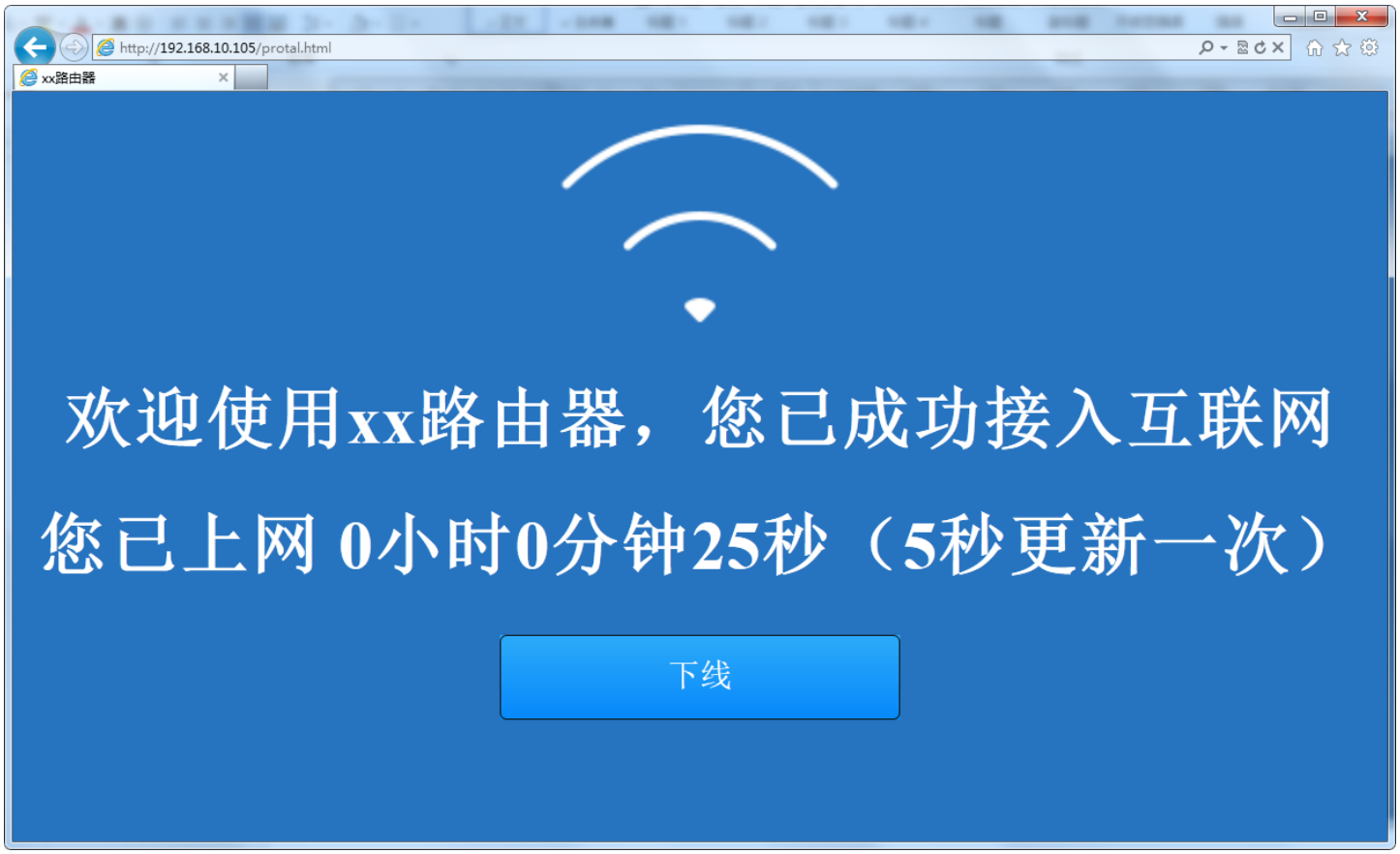




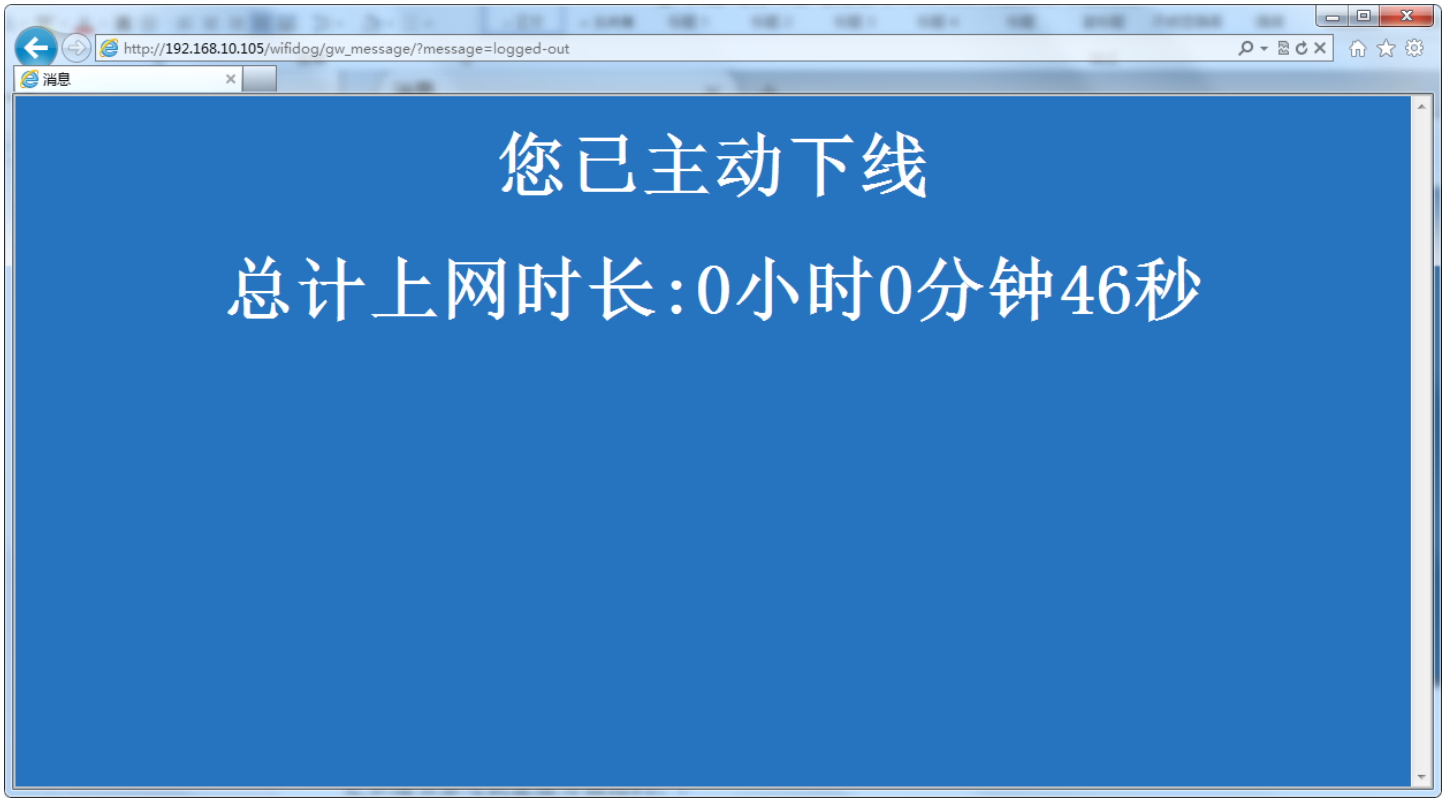
Wifidog 将我们的请求重定向到认证服务器的 login 页面了。我在认证服务器程序里只开通了一个用户名：zjh，当我们输入非法用户名时，效果如下：



单击“确定”将返回到登录界面，然后输入正确的用户名 zjh 并点击“登陆”按钮，效果如下：



网关将我们的请求重定向到认证服务器的门户页面了,现在可以打开一个新的页面上网了，可以单击“下线”按钮主动下线。



Wifidog 将我们的请求重定向到认证服务器的消息页面了。

使用手机连接 WiFi，效果如下：

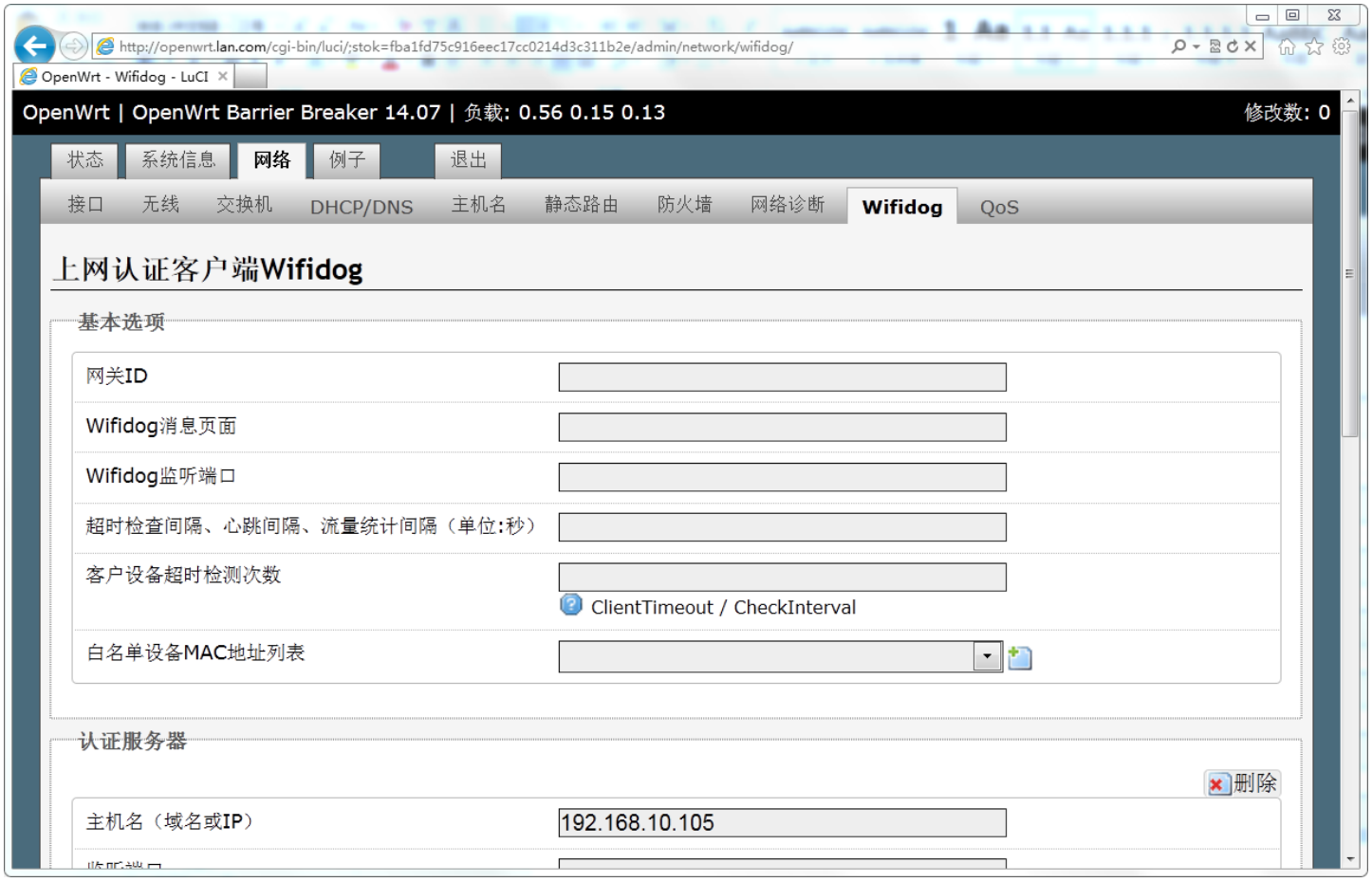
更多细节参考认证服务器源码和 Wifidog 源码。

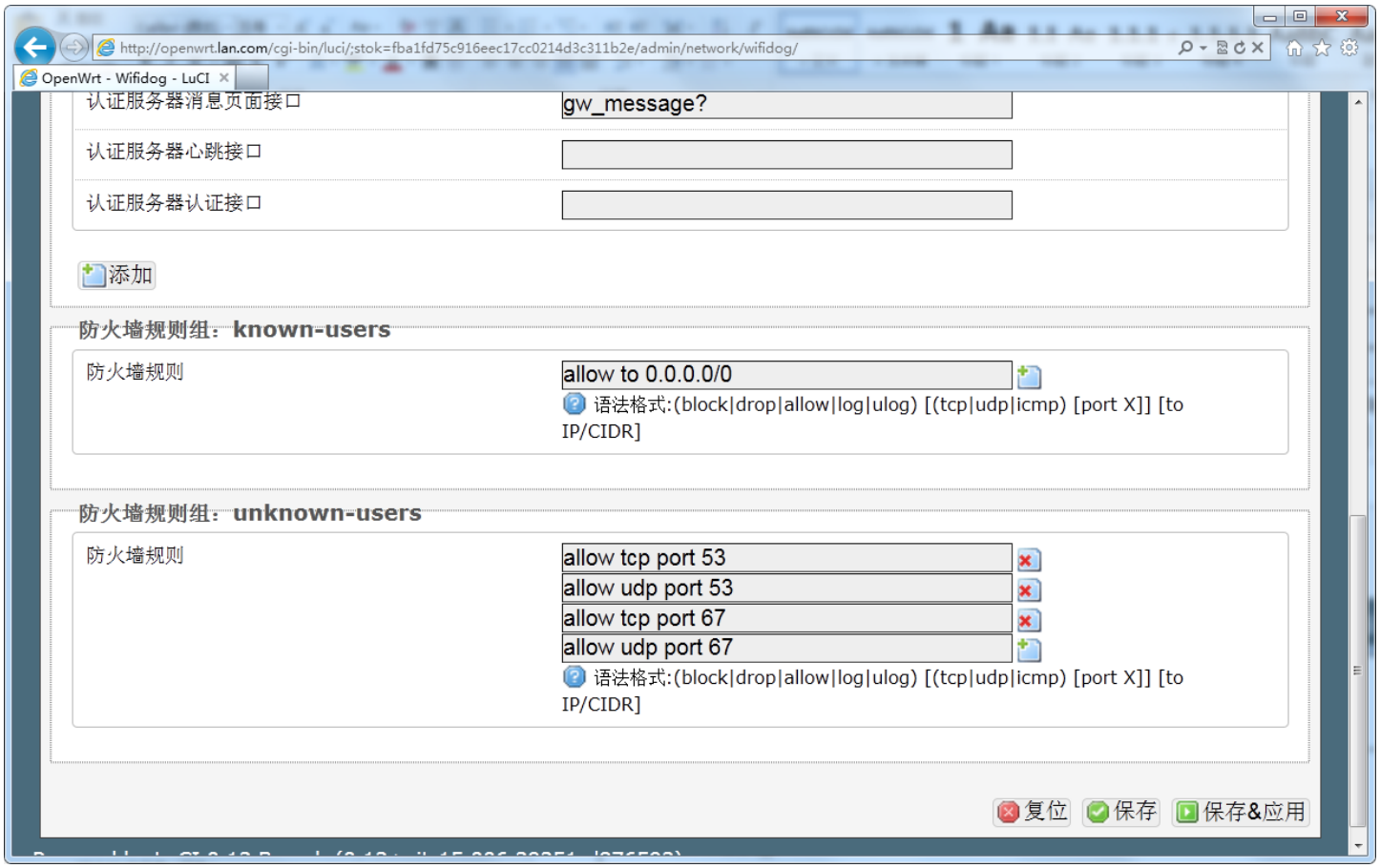
19.5 实例：使用 LuCI 配置 wifidog

本实例将创建 1 个 LUCI 界面，用来配置 Wifidog。
参考第 15 章 LuCI。

19.5.1 编写代码测试

首先在开发板上实现，下一节向 Openwrt 源码添加软件包。先看下效果





首先创建 lua 文件/usr/lib/lua/luci/controller/wifidog.lua，内容如下：

```
module("luci.controller.wifidog", package.seeall)

function index()
	if not nixio.fs.access("/etc/config/wifidog") then
		return
	end

	local page

	page = entry({"admin", "network", "wifidog"}, cbi("wifidog/wifidog"), _("Wifidog"))
	page.dependent = true
end
```

然后创建/usr/lib/lua/luci/model/cbi/wifidog/wifidog.lua，内容如下：

```
local sys = require "luci.sys"
m = Map("wifidog", translate("Wifidog for auth"))

--通用选项
s = m:section(TypedSection, "common", translate("Common"))
s.anonymous = true

--网关 ID
s:option(Value, "gateway_id", translate("GatewayID"))

--Wifidog 消息页面
s:option(Value, "html_messageFile", translate("HtmlMessageFile"))

--Wifidog 监听端口
gp = s:option(Value, "gateway_port", translate("GatewayPort"))
gp.datatype = "and(uinteger,min(1))"

--超时检查间隔、心跳间隔、流量统计间隔
ci = s:option(Value, "check_interval", translate("CheckInterval(s)"))
ci.datatype = "and(uinteger,min(1))"

--客户超时检测次数
ct = s:option(Value, "client_timeout", translate("ClientTimeout"), translate("ClientTimeout / CheckInterval"))
ct.datatype = "and(uinteger,min(1))"

--白名单 MAC 列表
tmc = s:option(Value, "trusted_mac_list", translate("TrustedMACList"))
tmc.datatype = "list(macaddr)"
sys.net.arptable(function(entry)
	tmc:value(
		entry["HW address"],
		entry["HW address"] .. " (" .. entry["IP address"] .. ")"
	)
```

```
end)

--认证服务器
s = m:section(TypedSection, "auth_server", translate("AuthServer"))
s.anonymous = true
s.addremove = true

--认证服务器域名或 IP
hn = s:option(Value, "hostname", translate("Hostname"))
hn.rmempty = false

--认证服务器监听端口
hp = s:option(Value, "http_port", translate("HTTPPort"))
hp.datatype = "port"

--认证服务器认证接口 cgi 根路径
s:option(Value, "path", translate("Path"))

--认证服务器登陆接口
s:option(Value, "login_script_path_fragment", translate("LoginScriptPathFragment"))

--认证服务器门户页面接口
s:option(Value, "portal_script_path_fragment", translate("PortalScriptPathFragment"))

--认证服务器消息页面接口
s:option(Value, "msg_script_path_fragment", translate("MsgScriptPathFragment"))

--认证服务器心跳接口
s:option(Value, "ping_script_path_fragment", translate("PingScriptPathFragment"))

--认证服务器认证接口
s:option(Value, "auth_script_path_fragment", translate("AuthScriptPathFragment"))

--防火墙规则 known-users
s = m:section(NamedSection, "known_users", "firewall_rule_set", translate("Known Users"))
s:option(DynamicList, "firewall_rule", translate("Firewall rule"), translate("syntax:(block|drop|allow|log|ulog)
[(tcp|udp|icmp) [port X]] [to IP/CIDR]"))

--防火墙规则 unknown-users
s = m:section(NamedSection, "unknown_users", "firewall_rule_set", translate("Unknown Users"))
s:option(DynamicList, "firewall_rule", translate("Firewall rule"), translate("syntax:(block|drop|allow|log|ulog)
[(tcp|udp|icmp) [port X]] [to IP/CIDR]"))

return m
```

参考 15.7 制作翻译文件 wifidog.po，内容如下：

```
msgid "Wifidog"
msgstr "Wifidog"
```

```
msgid "Wifidog for auth"
msgstr "上网认证客户端 Wifidog"

msgid "Common"
msgstr "基本选项"

msgid "GatewayID"
msgstr "网关 ID"

msgid "HtmlMessageFile"
msgstr "Wifidog 消息页面"

msgid "GatewayPort"
msgstr "Wifidog 监听端口"

msgid "CheckInterval(s)"
msgstr "超时检查间隔、心跳间隔、流量统计间隔（单位:秒）"

msgid "ClientTimeout"
msgstr "客户设备超时检测次数"

msgid "TrustedMACList"
msgstr "白名单设备 MAC 地址列表"

msgid "AuthServer"
msgstr "认证服务器"

msgid "Hostname"
msgstr "主机名（域名或 IP）"

msgid "HTTPPort"
msgstr "监听端口"

msgid "Path"
msgstr "认证接口 cgi 根路径"

msgid "LoginScriptPathFragment"
msgstr "认证服务器登录接口"

msgid "PortalScriptPathFragment"
msgstr "认证服务器门户页面接口"

msgid "MsgScriptPathFragment"
msgstr "认证服务器消息页面接口"

msgid "PingScriptPathFragment"
msgstr "认证服务器心跳接口"
```

```
msgid "AuthScriptPathFragment"
msgstr "认证服务器认证接口"

msgid "Known Users"
msgstr "防火墙规则组：known-users"

msgid "Unknown Users"
msgstr "防火墙规则组：unknown-users"

msgid "Firewall rule"
msgstr "防火墙规则"

msgid "syntax:(block|drop|allow|log|ulog) [(tcp|udp|icmp) [port X]] [to IP/CIDR]"
msgstr "语法格式:(block|drop|allow|log|ulog) [(tcp|udp|icmp) [port X]] [to IP/CIDR]"
```

创建 Wifidog 的 UCI 格式的配置文件/etc/config/wifidog，内容如下：

```
config common
        option gateway_interface 'br-lan'

config auth_server
        option hostname '192.168.10.105'
        option msg_script_path_fragment 'gw_message?'

config firewall_rule_set 'known_users'
        list firewall_rule 'allow to 0.0.0.0/0'

config firewall_rule_set 'unknown_users'
        list firewall_rule 'allow tcp port 53'
        list firewall_rule 'allow udp port 53'
        list firewall_rule 'allow tcp port 67'
        list firewall_rule 'allow udp port 67'
```

创建 Wifidog 配置文件初始化脚本/usr/bin/wifidog-init-conf，该脚本用来将 UCI 格式的配置文件转换成 Wifidog 能够识别的配置文件（/etc/wifidog.conf），该脚本的内容如下：

```
#!/bin/sh
. /lib/functions.sh

conf=/etc/wifidog.conf
reset_cb
rm $conf

handle_common() {
    local config="$1"

    config_get tmp $config gateway_id
    if [ -n "$tmp" ];then
        echo "GatewayID $tmp" > $conf
```

```
	fi

	config_get tmp $config external_interface
	if [ -n "$tmp" ];then
		echo "ExternalInterface $tmp" >> $conf
	fi

	config_get tmp $config gateway_interface
	if [ -n "$tmp" ];then
		echo "GatewayInterface $tmp" >> $conf
	fi

	config_get tmp $config gateway_address
	if [ -n "$tmp" ];then
		echo "GatewayAddress $tmp" >> $conf
	fi

	config_get tmp $config html_messageFile
	if [ -n "$tmp" ];then
		echo "HtmlMessageFile $tmp" >> $conf
	fi

	config_get tmp $config gateway_port
	if [ -n "$tmp" ];then
		echo "GatewayPort $tmp" >> $conf
	fi

	config_get tmp $config check_interval
	if [ -n "$tmp" ];then
		echo "CheckInterval $tmp" >> $conf
	fi

	config_get tmp $config client_timeout
	if [ -n "$tmp" ];then
		echo "ClientTimeout $tmp" >> $conf
	fi
}

handle_auth_server() {
	local config="$1"

	echo "AuthServer { " >> $conf

	config_get tmp $config hostname
	if [ -n "$tmp" ];then
		echo "	Hostname $tmp" >> $conf
	fi
```

```
	config_get tmp $config http_port
	if [ -n "$tmp" ];then
		echo "	HTTPPort $tmp" >> $conf
	fi

	config_get tmp $config path
	if [ -n "$tmp" ];then
		echo "	Path $tmp" >> $conf
	fi

	config_get tmp $config login_script_path_fragment
	if [ -n "$tmp" ];then
		echo "	LoginScriptPathFragment $tmp" >> $conf
	fi

	config_get tmp $config portal_script_path_fragment
	if [ -n "$tmp" ];then
		echo "	PortalScriptPathFragment $tmp" >> $conf
	fi

	config_get tmp $config msg_script_path_fragment
	if [ -n "$tmp" ];then
		echo "	MsgScriptPathFragment $tmp" >> $conf
	fi

	config_get tmp $config ping_script_path_fragment
	if [ -n "$tmp" ];then
		echo "	PingScriptPathFragment $tmp" >> $conf
	fi

	config_get tmp $config auth_script_path_fragment
	if [ -n "$tmp" ];then
		echo "	AuthScriptPathFragment $tmp" >> $conf
	fi

	echo "}" >> $conf
}

handle_tmc() {
	echo -n "$1," >> $conf
}

handle_common_tmc() {
	local config="$1"
	echo -n "TrustedMACList " >> $conf
	config_list_foreach "$config" trusted_mac_list handle_tmc
	echo "" >> $conf
}
```

```
handle_firewall_rule() {
	echo "	FirewallRule $1" >> $conf
}

handle_firewall_rule_set() {
	local config="$1"
	if [ "$1" = "known_users" ];then
		echo "FirewallRuleSet known-users {" >> $conf
		config_list_foreach "$config" firewall_rule handle_firewall_rule
		echo "}" >> $conf
	fi

	if [ "$1" = "unknown_users" ];then
		echo "FirewallRuleSet unknown-users {" >> $conf
		config_list_foreach "$config" firewall_rule handle_firewall_rule
		echo "}" >> $conf
	fi
}

config_load wifidog
config_foreach handle_common common
config_foreach handle_common_tmc common
config_foreach handle_auth_server	auth_server
config_foreach handle_firewall_rule_set	firewall_rule_set
```

然后改改脚本添加可执行权限。

修改/etc/init.d/wifidog 中的 start 函数，在里面添加一行执行我们自己编写的脚本

```
start() {
		/usr/bin/wifidog-init-conf
		/usr/bin/wifidog-init start
}
```

修改/etc/config/ucitrack，在最后添加一个配置

```
config wifidog
		option init 'wifidog'
```

执行这个操作后，通过 LuCI 修改配置文件才能使配置生效。

然后执行/etc/init.d/wifidog enable，使能 Wifidog 自启动。

现在可以通过浏览器访问路由器来修改 Wifidog 的配置文件了，修改后的配置文件内容如下（/etc/wifidog.conf）：

```
GatewayInterface br-lan
TrustedMACList
AuthServer {
		Hostname 192.168.10.105
		MsgScriptPathFragment gw_message?
}
FirewallRuleSet known-users {
```

```
        FirewallRule allow to 0.0.0.0/0
}
FirewallRuleSet unknown-users {
        FirewallRule allow tcp port 53
        FirewallRule allow udp port 53
        FirewallRule allow tcp port 67
        FirewallRule allow udp port 67
}
```

到这里，已经完成在路由器上的操作了，下节将通过添加软件包的形式来操作。

19.5.2 添加软件包

创建目录 feeds/luci/applications/luci-wifidog/并添加代码，代码组织结构如下所示：

```
root@zjh-vm:/home/work/openwrt/barrier_breaker/feeds/luci/applications# tree luci-wifidog/
luci-wifidog/
├── luasrc
│   ├── controller
│   │   └── wifidog.lua
│   └── model
│       └── cbi
│           └── wifidog
│               └── wifidog.lua
└── Makefile

5 directories, 3 files
```

其中 Makefile 的内容如下：

```
PO = wifidog

include ../../build/config.mk
include ../../build/module.mk
```

两个 wifidog.lua 的代码分别对应 19.5.1 节的代码。
然后修改 feeds/luci/contrib/package/luci/Makefile，添加一行代码，如下所示：

```
$(eval $(call application,firewall,Firewall and Portforwarding application,\
    +PACKAGE_luci-app-firewall:firewall))

$(eval $(call application,qos,Quality of Service configuration module,\
    +PACKAGE_luci-app-qos:qos-scripts))

$(eval $(call application,commands,LuCI Shell Command Module))
$(eval $(call application,example,Example for LuCI))
$(eval $(call application,wifidog,Wifidog for LuCI))
```

然后创建翻译文件 feeds/luci/po/zh_CN/wifidog.po，其内容对应 19.5.1 节的 wifidog.po

然后删除 luci 的编译目录，否则翻译文件不能自动安装。

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# rm build_dir/target-mipsel_24kec+dsp_uClibc-0.9.33.2/luci* -r
```

修改 wifidog 软件包代码 feeds/oldpackages/net/wifidog/files/wifidog.init 中的 start 函数

```
start() {
        /usr/bin/wifidog-init-conf
        /usr/bin/wifidog-init start
}
```

在 wifidog 软件包中创建脚本文件 feeds/oldpackages/net/wifidog/files/wifidog-init-conf，其内容对应 19.5.1 节的代码。

在 wifidog 软件包中创建 UCI 配置文件 feeds/oldpackages/net/wifidog/files/wifidog.uci，其内容如下：

```
config common
        option gateway_interface 'br-lan'

config auth_server
        option hostname '192.168.10.105'
        option msg_script_path_fragment 'gw_message?'

config firewall_rule_set 'known_users'
        list firewall_rule 'allow to 0.0.0.0/0'

config firewall_rule_set 'unknown_users'
        list firewall_rule 'allow tcp port 53'
        list firewall_rule 'allow udp port 53'
        list firewall_rule 'allow tcp port 67'
        list firewall_rule 'allow udp port 67'
```

在 wifidog 软件包中创建 ucitrack 初始化脚本 feeds/oldpackages/net/wifidog/files/ucitrack_init，其内容如下：

```
#!/bin/sh
uci add ucitrack wifidog
uci set ucitrack.@wifidog[0].init=wifidog
uci commit ucitrack
```

修改 wifidog 软件包代码 feeds/oldpackages/net/wifidog/Makefile，如下所示：

```
#define Package/wifidog/conffiles
#/etc/wifidog.conf
#endef

define Package/wifidog/install
        $(INSTALL_DIR) $(1)/usr/bin
        $(INSTALL_BIN) $(PKG_BUILD_DIR)/scripts/init.d/wifidog $(1)/usr/bin/wifidog-init
        $(INSTALL_BIN) ./files/wifidog-init-conf $(1)/usr/bin/wifidog-init-conf
        $(INSTALL_BIN) $(PKG_INSTALL_DIR)/usr/bin/wifidog $(1)/usr/bin/
        $(INSTALL_BIN) $(PKG_INSTALL_DIR)/usr/bin/wdctl $(1)/usr/bin/
        $(INSTALL_DIR) $(1)/usr/lib
        $(CP) $(PKG_INSTALL_DIR)/usr/lib/libhttpd.so* $(1)/usr/lib/
```

```
	$(INSTALL_DIR) $(1)/etc
	#$(INSTALL_DATA) $(PKG_BUILD_DIR)/wifidog.conf $(1)/etc/
	$(INSTALL_DIR) $(1)/etc/config $(1)/etc/uci-defaults
	$(INSTALL_DATA) ./files/wifidog.uci $(1)/etc/config/wifidog
	$(INSTALL_DATA) ./files/ucitrack_init $(1)/etc/uci-defaults/wifidog
	$(INSTALL_DATA) $(PKG_BUILD_DIR)/wifidog-msg.html $(1)/etc/
	$(INSTALL_DIR) $(1)/etc/init.d
	$(INSTALL_BIN) ./files/wifidog.init $(1)/etc/init.d/wifidog
endef
```

然后配置 Openwrt，使能编译 luci-app-wifidog

```
LuCI  --->
	3. Applications  --->
	<*> luci-app-wifidog
```

然后重新编译整个 Openwrt

```
root@zjh-vm:/home/work/openwrt/barrier_breaker# make V=s
```

然后更新固件

到这里，一切就绪。

19.6 实例：使用 LuCI 显示 wifidog 状态

19.6.1 编写代码测试

通过命令行程序可以查看 wifidog 的状态，操作如下：

```
root@OpenWrt:/# /etc/init.d/wifidog status
WiFiDog status

Version: 20130917
Uptime: 0d 0h 3m 14s
Has been restarted: no
Internet Connectivity: yes
Auth server reachable: yes
Clients served this session: 2

2 clients connected.

Client 0
  IP: 192.168.10.104 MAC: 44:8a:5b:ec:49:27
  Token: 1424756989.666
  Downloaded: 45001
  Uploaded: 22322

Client 1
  IP: 192.168.10.232 MAC: 0c:1d:af:c4:db:fc
  Token: 1424757036.471
```

```
    Downloaded: 0
    Uploaded: 0

Authentication servers:
    Host: 192.168.10.105 (192.168.10.105)
```

状态信息包括 wifidog 版本、运行时间、是否重启、是否接入互联网、是否和认证服务器保持通畅、客户设备连接信息等。下面将通过 Luci 界面来实时显示这些状态信息。

首先看下效果如下

该页面会自动刷新，默认每隔 5 秒刷新一次，参见后面创建的/usr/lib/lua/luci/view/wifidog/wifidog.htm 中的 XHR.poll

首先修改/usr/lib/lua/luci/controller/wifidog.lua

```
module("luci.controller.wifidog", package.seeall)

function index()
    if not nixio.fs.access("/etc/config/wifidog") then
        return
    end

    local page

    page = entry({"admin", "network", "wifidog"}, cbi("wifidog/wifidog"), _("Wifidog"))
    page.dependent = true
```

```
    entry({"admin", "status", "wifidog"}, template("wifidog/wifidog"), _("Wifidog"))
end
```

在原来的基础上增加了一个节点

创建 html 文件/usr/lib/lua/luci/view/wifidog/wifidog.htm，其内容如下：

```
<%
    if luci.http.formvalue("status") == "1" then
        local rv = {
            clients = {},
        }
        local f = io.popen("/etc/init.d/wifidog status 2>/dev/null")

        local num = 1

        while true do
            local line = f:read("*line")

            if not line then break end

            local id, i, j = nil
            i, j = string.find(line, "Version")
            if i then rv.version = string.sub(line, j+2) end

            i, j = string.find(line, "Uptime")
            if i then rv.uptime = string.sub(line, j+2) end

            i, j = string.find(line, "Has been restarted")
            if i then rv.restarted = string.sub(line, j+2) end

            i, j = string.find(line, "Internet Connectivity")
            if i then rv.connectivity_internet = string.sub(line, j+2) end

            i, j = string.find(line, "Auth server reachable")
            if i then rv.auth_server_reachable = string.sub(line, j+2) end

            i, j = string.find(line, "Client ")
            if i then
                rv.clients[num] = {}

                line = f:read("*line")
                i, j = string.find(line, "MAC:")

                rv.clients[num].ip = string.sub(line, 7, i - 2)
                rv.clients[num].mac = string.sub(line, j+1)

                line = f:read("*line")
                rv.clients[num].token = string.sub(line, 10)
```

```
                        line = f:read("*line")
                        rv.clients[num].downloaded = string.sub(line, 14)

                        line = f:read("*line")
                        rv.clients[num].uploaded = string.sub(line, 12)

                        num = num + 1
                    end

                end
                luci.http.prepare_content("application/json")
                luci.http.write_json(rv)

                return
            end
-%>

<%+header%>

<script type="text/javascript" src="<%=resource%>/cbi.js"></script>
<script type="text/javascript">//<![CDATA[
        XHR.poll(5, '<%=REQUEST_URI%>', { status: 1 },
                function(x, info)
                {
                        if (e = document.getElementById('version'))
                                e.innerHTML = info.version;

                        if (e = document.getElementById('uptime'))
                                e.innerHTML = info.uptime;

                        if (e = document.getElementById('restarted'))
                                e.innerHTML = info.restarted;

                        if (e = document.getElementById('connectivity_internet'))
                                e.innerHTML = info.connectivity_internet;

                        if (e = document.getElementById('auth_server_reachable'))
                                e.innerHTML = info.auth_server_reachable;

                        var cli = document.getElementById('clients');
                        if (cli)
                        {
                                /* clear all rows */
                                while(cli.rows.length > 2)
                                        cli.deleteRow(1);

                                for (i = 0; i < info.clients.length; i++)
                                {
```

```
                        var tr = cli.insertRow(-1);
                        tr.className = 'cbi-section-table-row cbi-rowstyle-' + ((i % 2) + 1);

                        tr.insertCell(-1).innerHTML = info.clients[i].mac
                        tr.insertCell(-1).innerHTML = info.clients[i].ip
                        tr.insertCell(-1).innerHTML = info.clients[i].token
                        tr.insertCell(-1).innerHTML = info.clients[i].downloaded
                        tr.insertCell(-1).innerHTML = info.clients[i].uploaded
                    }
                }
            }
        );
//]]></script>

<fieldset class="cbi-section">
    <legend><%:System%></legend>

    <table width="100%" cellspacing="10">
        <tr><td width="33%"><%:Version%></td><td id="version">-</td></tr>
        <tr><td width="33%"><%:Uptime%></td><td id="uptime">-</td></tr>
        <tr><td width="33%"><%:Has been restarted%></td><td id="restarted">-</td></tr>
        <tr><td width="33%"><%:Internet Connectivity%></td><td id="connectivity_internet">-</td></tr>
        <tr><td width="33%"><%:Auth server reachable%></td><td id="auth_server_reachable">-</td></tr>
    </table>
</fieldset>

<fieldset class="cbi-section">
    <legend><%:Authenticated Clients%></legend>

    <table class="cbi-section-table" id="clients">
        <tr class="cbi-section-table-titles">
            <th class="cbi-section-table-cell"><%:MAC-Address%></th>
            <th class="cbi-section-table-cell"><%:IP-Address%></th>
            <th class="cbi-section-table-cell"><%:Token%></th>
            <th class="cbi-section-table-cell"><%:Downloaded%></th>
            <th class="cbi-section-table-cell"><%:Uploaded%></th>
        </tr>
        <tr></tr>
    </table>
</fieldset>

<%+footer%>
```

该 html 文件使用嵌入 Lua 脚本读取前面介绍的/etc/init.d/wifidog status 操作的结果，然后将它们分别写入对应的位置。

修改翻译文件 wifidog.po，新增的内容如下

```
msgid "System"
msgstr "系统信息"
```

```
msgid "Version"
msgstr "版本"

msgid "Uptime"
msgstr "运行时间"

msgid "Has been restarted"
msgstr "是否重启过"

msgid "Internet Connectivity"
msgstr "是否接入互联网"

msgid "Auth server reachable"
msgstr "是否接通认证服务器"

msgid "Authenticated Clients"
msgstr "已认证的客户设备"

msgid "IP-Address"
msgstr "IP 地址"

msgid "Token"
msgstr "设备标识(Token)"

msgid "Downloaded"
msgstr "下载流量"

msgid "Uploaded"
msgstr "上传流量"
```

参考前面的操作方法，转换翻译文件，并拷贝到开发板对应位置。

19.6.2 添加软件包

参考 19.5.2 节。