##### **交叉编译**

在OPENWRT下执行make menuconfig，然后选择Utilities -> JMtry 为<M>，保存退出就可以了

在OPENWRT下执行make package/JMtry/compile编译就会得到该ipk了

想要清理的话使用以下命令：make package/JMtry/clean

###### **toolchain(工具链)**

openWRT下载

在编译openWRT之前，需要安装依赖，否则会自动停止进入make menuconfig。

一次性安装所需依赖组件，输入：

sudo apt-get install gcc g++ binutils patch bzip2 flex bison make autoconf gettext texinfo unzip sharutils subversion libncurses5-dev ncurses-term zlib1g-dev subversion git-core gawk asciidoc libz-dev zlib1g-dev libssl-dev openssl

分步安装如下：

sudo apt-get install g++

sudo apt-get install libncurses5-dev

sudo apt-get install zlib1g-dev

sudo apt-get install bison

sudo apt-get install flex

sudo apt-get install unzip

sudo apt-get install autoconf

sudo apt-get install gawk

sudo apt-get install make

sudo apt-get install gettext

sudo apt-get install gcc

sudo apt-get install binutils

sudo apt-get install patch

sudo apt-get install bzip2

sudo apt-get install libz-dev

sudo apt-get install asciidoc

sudo apt-get install subversion

sudo apt-get install sphinxsearch

sudo apt-get install libtool

sudo apt-get install sphinx-common

sudo apt-get install openssl

配置 menuconfig

cd openwrt

make menuconfig

进入下图所示的menuconfig配置界面，进行如下配置：

1、Target System

我手上的路由器CPU芯片是MT7620A，所以配置为 RT288x

2、Subtarget

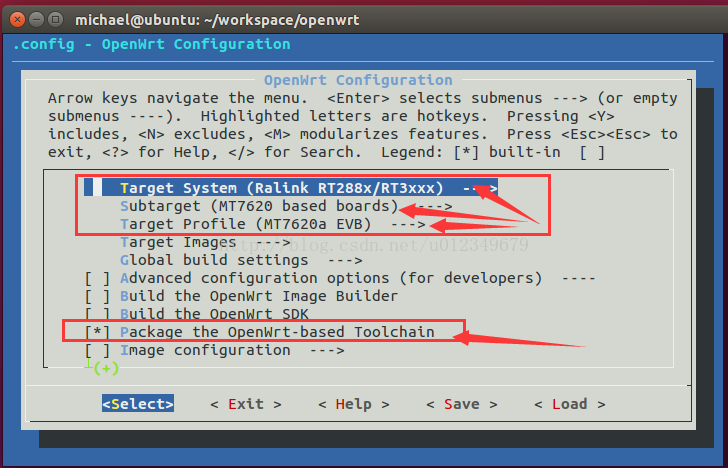
配置为MT7620

3、Target Profile

openwrt-18.06.2-ramips-mt7620-zbt-wr8305rt-squashfs-sysupgrade

4、Package the OpenWrt-based Toolchain

因为我们要制作toolchain交叉编译工具，所以这里通过click 空格键来切换到“\*”为选中状态



编译 Toolchain

配置完成后，保存并退出，然后输入命令编译：

make V=s

一切无错的情况下,会在openwrt源码目录/bin/ramips/目录下找到编译好的toolchain

###### **设置环境变量**

这里只有两个变量需要设置

1. 将工具链中的bin文件夹加入PATH
2. 添加一个STAGING\_DIR 变量，将其指向放toolchain的文件夹

比如我把toolchain解压到了/home/yj/openwrt/mips-linux-gcc/下,就把STAGING\_DIR 的值赋为/home/yj/openwrt/mips-linux-gcc/

设置好环境变量后，可以检验一下,在shell中输入编译器的前几部分，看是否可以自动补全

###### **进行编译**

1.编译单个文件,就像普通的编译一样，只不过把gcc换成了交叉编译的编译器 执行 mips-openwrt-linux-uclibc-gcc test.c -o test 得到的test就是路由器上可以执行的文件

2.编译软件，在./configure 时加上–host=mips-openwrt-linux-uclibc

###### **利用SDK来生成ipk文件**

SDK中其实是包含toolchain的，SDK就是将编译好的文件在打包到一个ipk中，便于安装。

SDK生成ipk包时需要的只是一个makefile文件，makefile里写了所需下载的文件、生成规则、软件版本、类型等。

用SDK进行编译错误比较多，最好使用openwrt编译环境进行编译

###### **openwrt的结构.**

代码上来看有几个重要目录package, target, build\_root, bin, dl....

---build\_dir/host目录是建立工具链时的临时目录

---build\_dir/toolchain-<arch>\*是对应硬件的工具链的目录

---staging\_dir/toolchain-<arch>\* 则是工具链的安装位置

---target/linux/<platform>目录里面是各个平台(arch)的相关代码

---target/linux/<platform>/config-3.10文件就是配置文件了

---dl目录是'download'的缩写, 在编译前期，需要从网络下载的数据包都会放在这个目录下，这些软件包的一个特点就是，会自动安装在所编译的固件中，也就是我们make menuconfig的时候，为固件配置的一些软件包。如果我们需要更改这些源码包，只需要将更改好的源码包打包成相同的名字放在这个目录下，然后开始编译即可。编译时，会将软件包解压到build\_dir目录下。

---而在build\_dir/目录下进行解压，编译和打补丁等。

---package目录里面包含了我们在配置文件里设定的所有编译好的软件包。默认情况下，会有默认选择的软件包。在openwrt中ipk就是一切, 我们可以使用

$ ./scripts/feeds update来对软件包进行更新.

$ ./scripts/feeds search nmap 查找软件包'nmap'

Search results in feed ’packages’:

nmap Network exploration and/or security auditing utility

$ ./scripts/feeds install nmap 安装'nmap'这个软件

$ make package/symlinks //估计意思是更新软件源之类的

---bin目录下生成了很多bin文件，根据不同的平台来区分。另外bin/<platform>/package目录，里面有很多ipk后缀的文件，都是package目录下的源码在build\_dir目录下编译后的生成的结果。

##### Makefile工程创建

###### 创建helloworld项目

新建helloworld.c文件和对应的Makefile文件（注意Makefile文件的书写格式）

$mkdir -p ~/temp/hellworld/src

$cd ~/temp/helloworld/src

$touch helloworld.c Makefile

如下为helloworld.c的内容:

#include <stdio.h>

int main()

{

printf("This is my helloworld!\n");

return 0;

}

如下为Makefile文件的内容:

helloworld : helloworld.o

$(CC) $(LDFLAGS) helloworld.o -o helloworld

helloworld.o : helloworld.c

$(CC) $(CFLAGS) -c helloworld.c

clean :

rm \*.o helloworld

$(CC)

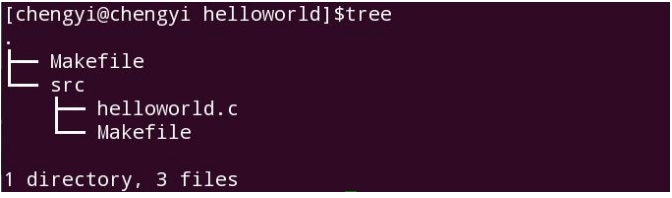
$(CC): 这个值由其他Makefile文件规定,表示我们使用编译器.

$(LDFLAGS)\$(CFLAGS) : 这个表示编译器的一些选项,这里是可选的,去掉也没有问题.

可以输入$make看看有没有问题, 最后,输入$make clean来清理掉生成的二进制文件.因为上一步make所使用的编译器并不是我们的交叉编译链,生成的二进制文件并不能在开发板中运行，只是验证我们的src中的内容正确与否.

###### **创建helloworld包**

为openwrt创建一个新的Makefile文件,在这个文件中要描述helloworld包的信息,比如:如何配置,如何编译,如何打包,安装位置等.这个Makefile必须放在第一层目录里，能够被openwrt编译环境识别。



package

|-----Makefile(SDK自带)

|-----XXX(文件夹,随便起名字)

|-----Makefile(需要自己写,根据模板)

|-----src(文件夹, 必须是src这个名字)

|----src里就是需要编译的源码, 里面就自己随便写了,就跟平时做linux开发一样的

$cd ~/temp/helloworld

$touch Makefile

如下为Makefile内容:

include $(TOPDIR)/rules.mk

PKG\_NAME:=helloworld

PKG\_RELEASE:=1

PKG\_BUILD\_DIR:=$(BUILD\_DIR)/$(PKG\_NAME)

include $(INCLUDE\_DIR)/package.mk

define Package/helloworld

SECTION:=utils

CATEGORY:=Utilities

TITLE:=Helloworld -- prints a snarky message

endef

define Package/helloworld/description

It's my first package demo.

endef

define Build/Prepare

echo "Here is Package/Prepare"

mkdir -p $(PKG\_BUILD\_DIR)

$(CP) ./src/\* $(PKG\_BUILD\_DIR)/

endef

define Package/helloworld/install

echo "Here is Package/install"

$(INSTALL\_DIR) $(1)/bin

$(INSTALL\_BIN) $(PKG\_BUILD\_DIR)/helloworld $(1)/bin/

endef

$(eval $(call BuildPackage,helloworld))

第1行include $(TOPDIR)/rules.mk:

一般在Makefile的开头,包含了包的基本信息,比如Makefile中的$(BUILD\_DIR),$(INCLUDE\_DIR),$(CP),$(INSTALL\_DIR),$(INSTALL\_BIN)都是这里定义的.具体内容可以到源码主目录下,查看rules.mk文件.

3~5行,软件包的信息均以“PKG\_”开头，其意思和作用如下:

PKG\_NAME：软件包名称，将在menuconfig和ipkg可以看到。

PKG\_VERSION：待编译软件本身的版本号。

PKG\_RELEASE：本Makefile文件的版本号

PKG\_SOURCE：源代码的文件名，如果从git clone，则下载之后按该文件名打包

PKG\_SOURCE\_PROTO：源代码仓库的管理协议(git, svn)

PKG\_SOURCE\_URL：源代码的下载网站位置。

PKG\_SOURCE\_VERSION：使用git时必须指明，定义检出的分支

PKG\_SOURCE\_SUBDIR：使用"svn" or "git"时必须指明，把代码检出下载到的目录名称，缺省为$(PKG\_NAME)-$(PKG\_VERSION)

PKG\_MD5SUM：源代码文件的效验码。用于核对软件包是否下载正确。

PKG\_CAT：源代码文件的解压方法。包括zcat, bzcat, unzip等。

PKG\_BUILD\_DIR：软件包编译目录，把代码拷贝到那里进行编译。它的父目录为$(BUILD\_DIR)

PKG\_INSTALL：Setting it to "1" will call the package's original "make install" with prefix set to PKG\_INSTALL\_DIR

PKG\_INSTALL\_DIR：Where "make install" copies the compiled files

PKG\_BUILD\_PARALLEL在include/host-build.mk中找到了相关的定义，就是类似make的时候的参数 -j,就是多线程编译。

第7行include $(INCLUDE\_DIR)/package.mk:

一般在软件包的基本信息完成后再引入，他定义了用户态软件包的规则。编译包分为用户态和内核模块，用户态软件包使用Package，内核模块使用KernelPackage.$(INCLUDE\_DIR)/Kernel.mk文件对于软件包为内核时不可缺少，$(INCLUDE\_DIR)/package.mk应用在用户态。接下来讲述用户态软件包。用户程序的编译包以Package/开头，然后接着软件名，在Package定义中的软件名可以与软件包名不一样，而且可以多个定义。

9~13行:

定义包的名称为helloworld

SECTION : 包的类型为utils

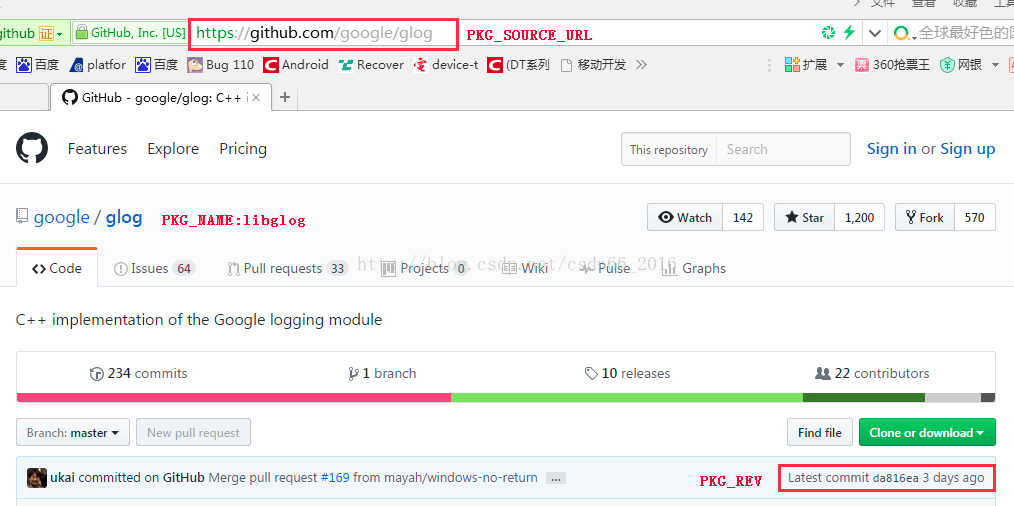
CATEGORY : 目录为Utilitis,即文件在menuconfig中的位置;有时还会有SUBMENU项,即子目录.

TITLE : 用于软件包的简短描述,将显示在menuconfig中.

URL ： 软件包的下载位置。

MAINTAINER ： 维护者选项。

DEPENDS ： 与其他软件的依赖。即如编译或安装需要其他软件时需要说明。如果存在多个依赖，则每个依赖需用空格分开。依赖前使用+号表示默认显示，即对象沒有选中时也会显示，，即当依赖对象选中后才显示。





通过页面上面的信息，可以定义这些变量：

PKG\_NAME:=libglog

PKG\_VERSION:=0.3.3

PKG\_REV:=da816ea70645e463aa04f9564544939fa327d5a7

PKG\_SOURCE\_URL:=git://github.com/google/glog.git

PKG\_SOURCE\_PROTO:=git

扩展变量定义：

PKG\_SOURCE:=$(PKG\_NAME)-$(PKG\_VERSION).tar.gz

PKG\_SOURCE\_VERSION:=$(PKG\_REV)

PKG\_SOURCE\_SUBDIR:=$(PKG\_NAME)-$(PKG\_VERSION)

其他相关变量定义：

PKG\_INSTALL:=1 需要make install

PKG\_BUILD\_PARALLEL:=1 允许并行编译

PKG\_LICENSE:=LGPL-2.1 license

15~17行:

软件包的详细描述,将显示在make menuconfig中

19~23行:

编译准备方法，对于网上下载的软件包不需要再描述。对于非网上下载或自行开发的软件包必须说明编译准备方法。本文所用的准备方法就是首先创建软件包目录，然后将源码拷贝到刚刚创建的目录中。按OpenWrt的习惯，一般把自己设计的程序全部放在src目录下。

25~29行:

软件包的安装方法，包括一系列拷贝编译好的文件到指定位置。调用时会带一个参数，就是嵌入系统的镜像文件系统目录，因此$(1)表示嵌入系统的镜像目录。

INSTALL\_DIR:=install -d -m0755 : 创建所属用戶可读写、执行，其他用戶可读可执行的目录

INSTALL\_BIN:=install -m0755 : 编译好的文件到镜像文件目录

31行 $(eval $(call BuildPackage,helloworld)):

完成前面定义后，必须使用eval函数实现各种定义。其格式为：

对于一般软件包：$(eval $(call Package,$(PKG\_NAME)))

或对于内核模块：$(eval $(call KernelPackage,$(PKG\_NAME)))

如果一个软件包有多个程序，例如：一个应用程序有自己的内核模块，上面使用的PKG\_NAME需要灵活变通。eval函数可能设计多个。也可以当成多个软件包处理。

###### **编译软件**

至此我们的软件已经基本完成,下面进行编译

首先将文件文件夹拷贝到openwrt目录中的package文件中,这里我的源码目录为~/openwrt,你需要把openwrt目录替换为你的openwrt源码目录.

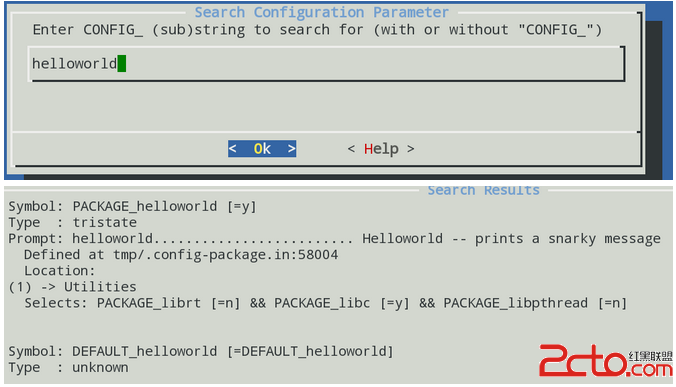
$mv ~/temp/helloworld ~/openwrt/package

然后回到项目主目录运行make menuconfig

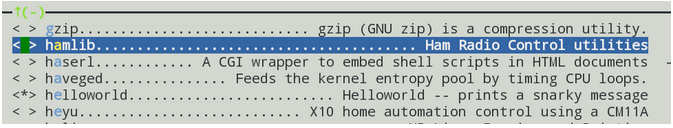
$cd ~/openwrt

$make menuconfig

按"/"后,输入helloworld,搜索对应的路径



接着到Utilities目录下,找到helloworld并按空格打开;



保存后退出;

$cd ~/openwrt

$make package/helloworld/compile V=s

编译完成后,ipk应该已经生成

$find bin/ -name "helloworld\*.ipk"

###### **安装执行软件**

上传helloworld\_1\_bcm47xx.ipk至路由器

执行# opkg install helloworld\_1\_bcm47xx.ipk

然后输入hello然后按Tab键，发现openwrt中已经有helloworld可执行命令。

执行 helloworld命令来查看程序的效果。

Hell! O' world, why won't my code compile?

#### 其他参考文档：

1、OpenWRT开发之——研究包的Makefile

https://blog.csdn.net/xiaopang1122/article/details/50617346