SDK内化文档

# 总体说明

## 修订历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 日期 | 作者/修订者 | 修改内容 | 评审者 |
| 1.0 | 20200909 | 倪友田 | 初稿 |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 主机环境搭建

## 编译docker 镜像



sudo docker build -t sdk:v3 .

## 运行docker 镜像

sudo docker run -it -p 2222:22 -v /home/steven/study2/work\_sdk:/home/share:rw sdk:v4

## 将运行中的容器保存为docker镜像

$ sudo docker commit <当前运行的container id> <仓库名称>:<tag>

$ sudo docker save -o <仓库名称>-<tag>.img <仓库名称>:<tag>

示例如下:

$ sudo docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

111111111111 222222222222 "/bin/bash" 5 minutes ago Up 5 minutes jello

$ sudo docker commit 111111111111 bash:1.0

$ sudo docker save -o bash-1.0.img bash:1.0

## 安装ssh

**apt-get install -y openssh-server**

给root设置一个密码 root

passwd root

vim /etc/ssh/sshd\_config

注释第一行，添加第二行

# PermitRootLogin prohibit-password.

PermitRootLogin yes

保存退出后，重启ssh服务

/etc/init.d/ssh restart

# SDK作用

## SDK介绍

网关SDK用来编译网关插件：

云客户端扩展插件（extcc） ---- 薛豪俊、钟美燕

组网插件（inter\_connd） ---- 章辉

网关质量插件（opmaintain） ---- 江俊晔、秦旭日

统一消息插件（u01） ---- 钟美燕、薛豪俊

网络质量插件（xrobot）

DPI插件（b01）

游戏加速插件（gamespeeder）

路由器SDK用来编译路由器插件：

路由器采集插件（apmaintain）

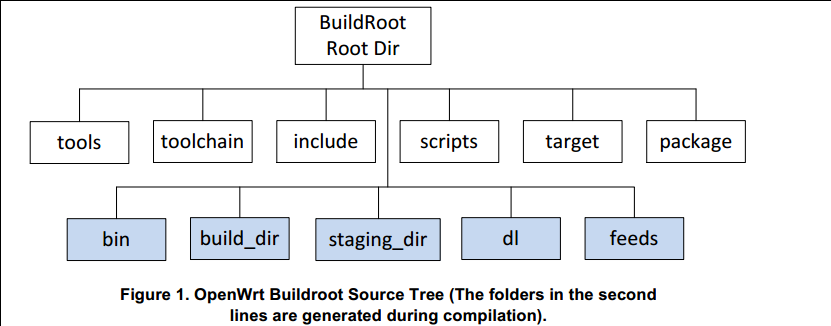
## 最新SDK下载

最新网关SDK framework\_v3.0\_R4001.sh

网盘地址：https://cloud.189.cn/t/umIRFfNjQ3ai（访问码：xr2t）

# SDK整体

## openwrt目录结构



上图是openwrt目录结构，其中第一行是原始目录，第二行是编译过程中生成的目录。各目录的作用是：

tools - 编译时需要一些工具， tools里包含了获取和编译这些工具的命令。里面是一些Makefile，有的可能还有patch。每个Makefile里都有一句 $(eval $(call HostBuild))，表示编译这个工具是为了在主机上使用的。

toolchain - 包含一些命令去获取kernel headers, C library, bin-utils, compiler, debugger

target - 各平台在这个目录里定义了firmware和kernel的编译过程。

package - 包含针对各个软件包的Makefile。openwrt定义了一套Makefile模板，各软件参照这个模板定义了自己的信息，如软件包的版本、下载地址、编译方式、安装地址等。

include - openwrt的Makefile都存放在这里。文件名为 \*.mk 。这里的文件上是在Makefile里被include的,类似于库文件.这些文件定义了编译过程。

scripts - 存放了一些脚本,使用了bash,Python,perl等多种脚本语言.编译过程中,用于第三方软件包管理的feeds文件也是在这个目录当中.在编译过程中,使用到的脚本也统一放在这个目录中。

dl - 软件包下载后都放到这个目录里

build\_dir - 软件包都解压到build\_dir/里，然后在此编译

staging\_dir - 最终安装目录。tools, toolchain被安装到这里，rootfs也会放到这里。

feeds -

bin - 编译完成之后，firmware和各ipk会放到此目录下。

## 主Makefile工作过程

OpenWrt的主Makefile文件只有100行，可以简单分为三部分，1~17行为前导部分，19~31为首次执行部分，33~101为再次执行部分。

### 前导部分

CURDIR为make默认变量，默认值为当前目录。

前导部分主要把变量TOPDIR赋值为当前目录，把变量LC\_ALL、LANG赋值为C，并使用变量延伸指示符export，把上述三个变量延伸到下层Makefile。

使用文件使用指示符include引入$(TOPDIR)/include/host.mk。在OpenWrt的主Makefile文件使用了多次include指示符，说明主Makefile文件被拆分成多个文件，被拆分的文件放在不同的目录。拆分的目的是明确各部分的功能，而且增加其灵活性。

### 首次执行部分

OPENWRT\_BUILD是区分首次执行与再次执行的变量。在首次执行时使用强制赋值指示符override把OPENWRT\_BUILD赋值为1，并使用变量延伸指示符export把OPENWRT\_BUILD延伸。在OPENWRT\_BUILD使用强制赋值指示符override意味着make命令行可能引入OPENWRT\_BUILD参数。

引入$(TOPDIR)/include/debug.mk、$(TOPDIR)/include/depends.mk、$(TOPDIR)/include/toplevel.mk三个文件，由于TOPDIR是固定的，所以三个文件也是固定的。其中$(TOPDIR)/include/toplevel.mk的135行%::有效解释首次执行时world目标的规则。

### 再次执行部分

引入rules.mk、$(INCLUDE\_DIR)/depends.mk、$(INCLUDE\_DIR)/subdir.mk、target/Makefile、package/Makefile、tools/Makefile、toolchain/Makefile七个文件，rules.mk没有目录名，即引入与主Makefile文件目录相同的rules.mk。在rules.mk定义了INCLUDE\_DIR为$(TOPDIR)/include，所以$(INCLUDE\_DIR)/depends.mk实际上与首次执行时引入的$(TOPDIR)/include/depends.mk是同一个文件。

四个子目录下的Makefile实际上是不能独立执行。主要利用$(INCLUDE\_DIR)/subdir.mk动态建立规则，诸如$(toolchain/stamp-install)目标是靠$(INCLUDE\_DIR)/subdir.mk的stampfile函数动态建立。在package/Makefile动态建立了$(package/ stamp-prereq)、$(package/stamp-cleanup)、$(package/ stamp-compile)、$(package/stamp-install)、$(package/ stamp-rootfs-prepare)目标。

定义一些使用变量命名的目标，其变量的赋值位置在$(INCLUDE\_DIR)/subdir.mk的stampfile函数中。目标只有依赖关系，可能说明其工作顺序，在$(INCLUDE\_DIR)/subdir.mk的stampfile函数中有进一步说明其目标执行的命令，并为目标建立一个空文件，即使用变量命名的目标为真实的文件。

定义一些使用固定的目标规则。

其中：clean是清除编译结果的目标，清除$(BUILD\_DIR) $(BIN\_DIR)$(BUILD\_LOG\_DIR)三个目录的用意是十分明确。暂时不知道为什么执行maketarget/linux/clean。

dirclean是删除所有编译过程产生的目录和文件的目标，执行dirclean目标依赖于clean，因此将执行clean目标所执行的命令，然后删除$(STAGING\_DIR) $(STAGING\_DIR\_HOST) $(STAGING\_DIR\_TOOLCHAIN)$(TOOLCHAIN\_DIR) $(BUILD\_DIR\_HOST)$(BUILD\_DIR\_TOOLCHAIN)目录，以及删除$(TMP\_DIR)目录。上述目录的变量均在rules.mk定义。好像删除staging\_dir目录就意味着删除staging\_dir目录下的所有子目录，不知道为什么要强调删除$(STAGING\_DIR\_HOST)$(STAGING\_DIR\_TOOLCHAIN)$(TOOLCHAIN\_DIR)目录。同样删除builde\_dir目录就意味着删除builde\_dir目录下的所有子目录，不知道为什么要强调删除$(BUILD\_DIR\_TOOLCHAIN)目录。

tmp/.prereq\_packages目标是对所需软件包的预处理。目标依赖于.config，即执行make menuconfig后将会进行一次所需软件包的预处理。不知什么原因在编译前删除tmp目录，执行时无法建立tmp/.prereq\_packages文件。

prereq应该是预请求目标，在OpenWrt执行Makefile时好像都要先执行prereq目标。

prepare应该是准备目标，是world依赖的一个伪目标。依赖于文件.config和$(tools/stamp-install)$(toolchain/stamp-install)目标。

world就是编译的目标。依赖于prepare为目标和前面提到的变量命名目标。采用取消隐含规则方式执行package/index目标。package/index目标在package/Makefile的92行定义。

package/symlinks和package/symlinks-install是更新或安装软件包来源的目标，使用$(SCRIPT\_DIR)/feeds脚本文件完成。

package/symlinks-clean是清除软件包来源的目标，也是使用$(SCRIPT\_DIR)/feeds脚本文件完成。

最后使用伪目标.PHONY说明clean dirclean prereq prepareworld package/symlinks package/symlinks-installpackage/symlinks-clean属于伪目标。通过伪目标说明可以知道可以执行的目标。

## 生成.config

### menuconfig简介

menuconfig是一套图像化配置工具，由ncurses库提供软件支持。ncurses库提供了一系列的函数以便使用者调用它们去生成基于文本的用户界面。

menuconfig本身的软件只负责提供menuconfig工作的这一套逻辑，比如说通过上下左右调整光标，Enter选中等，并不负责提供内容。menuconfig运行之后会读取Kconfig、读取/写入.config文件，Kconfig提供菜单项的内容，.config用来记录菜单项的选择值。

### menuconfig语法

choice

prompt "Star Memleak Test Tool"

default STAR\_MEMLEAK\_OFF

config STAR\_MEMLEAK\_ON

bool "ON"

config STAR\_MEMLEAK\_OFF

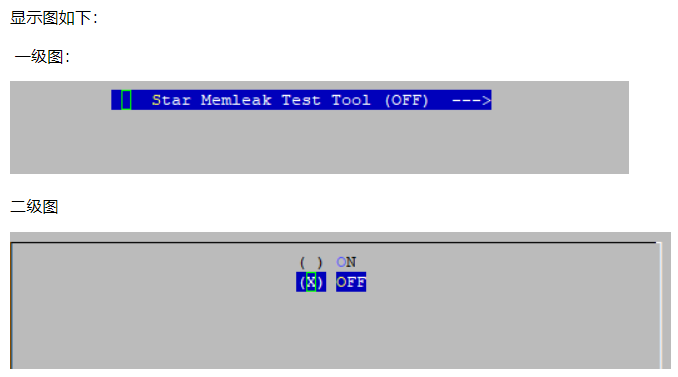
bool "OFF"

endchoice

prompt ：描述了config的名字

choice：定义选项

default： 此处默认STAR\_MEMLEAK\_OFF，是关闭的显示OFF



config :定义了配置项名字

menu ：定义了菜单名字，必须以endmenu结束，menu后跟的名字会显示在菜单中。

在其控制范围内再添加config 或者menu

bool ：描述了配置项显示在菜单中的名字

default ：y 选中，菜单中有\* 表示，n 未选中，空白表示

help： 帮助，描述文档

depends on：依赖，如上菜单SL\_CONFIG\_MQTT 依赖于config MQTT\_COMM\_SUPPORT，选中MQTT\_COMM\_SUPPORT才会出现SL\_CONFIG\_MQTT菜单

### 生成配置文件

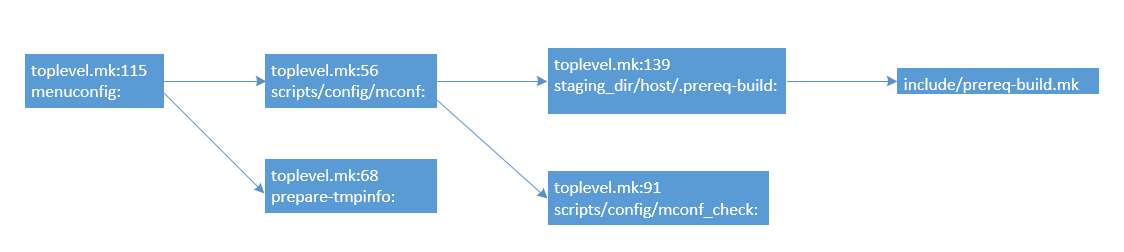
./profileselect.sh 选择对应的厂商配置文件

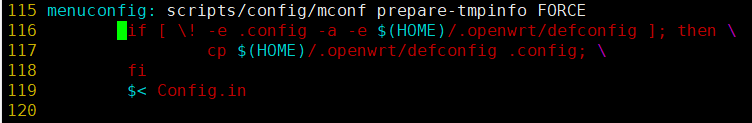
最终生成软连接 .config -> confs/zx279128.config

make menuconfig后会修改.config文件如下：

.config

.config.old -> confs/zx279128.config





对119行展开 scripts/config/mconf Config.in

$@ 表示目标文件

$^ 表示所有的依赖文件

$< 表示第一个依赖文件

$? 表示比目标还要新的依赖文件列表

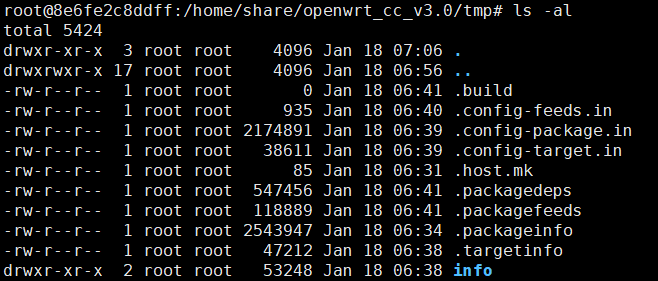
<https://www.cnblogs.com/gamesun/p/3323155.html>

最终的.config 文件 以zx279128为例



|  |  |
| --- | --- |
| 根Config.in下的子文件 | 功能描述 |
| target/Config.in | 设置厂家，芯片型号，linux版本，cpu型号，默认库 |
| config/Config-images.in | 定义文件系统 |
| config/Config-build.in | 内核编译选项，通用编译选项，package编译选项 |
| config/Config-devel.in | 设置二进制，下载目录，编译优化选项 |
| toolchain/Config.in | 设置 C库 |
| target/imagebuilder/Config.in |  |
| target/sdk/Config.in |  |
| target/toolchain/Config.in |  |
| tmp/.config-package.in |  |

执行make menuconfig后



## make prepare

### make prepare 流程



1. 第一目标规则include/host.mk:29

$(TMP\_DIR)/.host.mk: $(TOPDIR)/include/host.mk

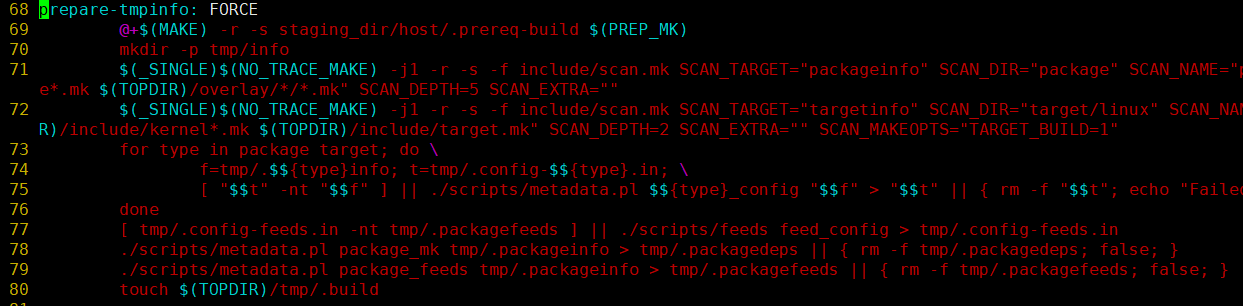
1. 第二目标规则include/toplevel.mk:181 %::



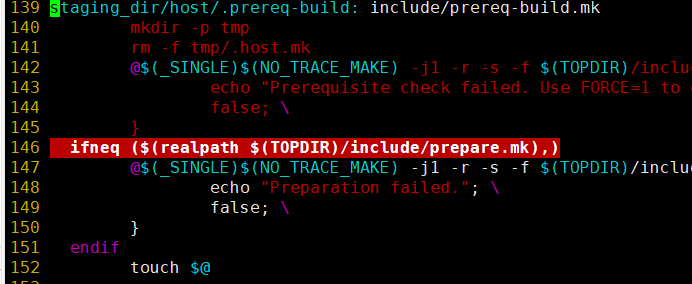
1. include/toplevel.mk:166 prereq: prepare-tmpinfo .config



1. include/toplevel.mk:68 prepare-tmpinfo: FORCE



1. include/toplevel.mk:139



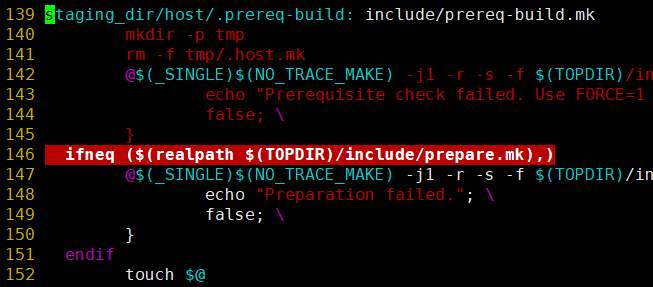
检查主机编译命令GCC等是否可以工作

1. include/toplevel.mk:56

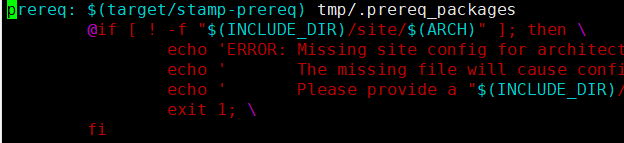
.config: scripts/config/conf prepare-tmpinfo staging\_dir/host/.prereq-build



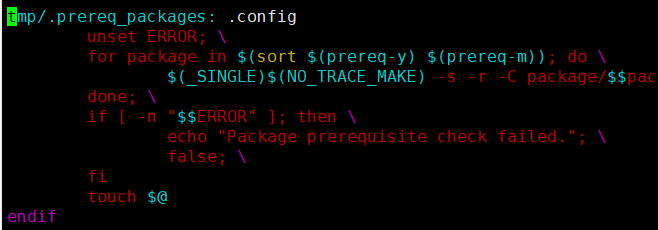
1. include/toplevel.mk:139



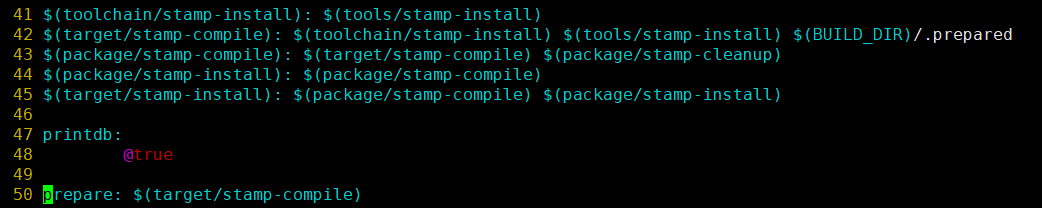
1. Makefile:77



1. Makefile:64 tmp/.prereq\_packages: .config



1. Makefile:50



1. tools/Makefile:121

prepare: .config /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/stamp/.tools\_install\_yynyynynynyyyyynnyynyyyyyynnynyyyyynnyyynnyynnnyy /home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-arm-27912x-linux-uclibc/stamp/.toolchain\_install

$(eval $(call stampfile,$(curdir),tools,install,,\_$(subst $(space),,$(tools\_enabled))))

变量$(tools/stamp-install) 是通过tools/Makefile如下展开得到



### make prepare 规则入口

1. %::
2. @+$(PREP\_MK) $(NO\_TRACE\_MAKE) -r -s prereq  /\*OPENWRT\_BUILD= QUIET=0 make V=s -r -s prereq\*/
3. @( \
4. cp .config tmp/.config; \
5. ./scripts/config/conf --defconfig=tmp/.config -w tmp/.config Config.in > /dev/null 2>&1; \
6. **if** ./scripts/kconfig.pl '>' .config tmp/.config | grep -q CONFIG; then \
7. printf "$(\_R)WARNING: your configuration is out of sync. Please run make menuconfig, oldconfig or defconfig!$(\_N)\n" >&2; \
8. fi \
9. )
10. @+$(ULIMIT\_FIX) $(SUBMAKE) -r $@ $(**if** $(WARN\_PARALLEL\_ERROR), || { \
11. printf "$(\_R)Build failed - please re-run with -j1 to see the real error message$(\_N)\n" >&2; \
12. **false**; \
13. } )

PREP\_MK = OPENWRT\_BUILD= QUIET=0

NO\_TRACE\_MAKE = make V=s

10行替换成 @+$(SUBMAKE) -r $@ ----umask 022; remake -w -r prepare

prereq目标依赖于prepare-tmpinfo .config，并且会进入到第二逻辑中Makefile:77

1. prereq:: prepare-tmpinfo .config
2. @+$(NO\_TRACE\_MAKE) -r -s $@

prepare目标会进入到第二逻辑中Makefile:85

1. 76 # check prerequisites before starting to build
2. 77 prereq: $(target/stamp-prereq) tmp/.prereq\_packages
3. 78         @**if** [ ! -f "$(INCLUDE\_DIR)/site/$(ARCH)" ]; then \
4. 79                 echo 'ERROR: Missing site config for architecture "$(ARCH)" !'; \
5. 80                 echo '       The missing file will cause configure scripts to fail during compilation.'; \
6. 81                 echo '       Please provide a "$(INCLUDE\_DIR)/site/$(ARCH)" file and restart the build.'; \
7. 82                 exit 1; \
8. 83         fi
9. 84
10. 85 prepare: .config $(tools/stamp-install) $(toolchain/stamp-install)

tools/stamp-install 是在include tools/Makefile里定义

include/toplevel.mk 68行

1. prepare-tmpinfo: FORCE
2. @+$(MAKE) -r -s staging\_dir/host/.prereq-build $(PREP\_MK)  /\*remake -r -s staging\_dir/host/.prereq-build OPENWRT\_BUILD= QUIET=0\*/
3. mkdir -p tmp/info
4. $(\_SINGLE)$(NO\_TRACE\_MAKE) -j1 -r -s -f include/scan.mk SCAN\_TARGET="packageinfo" SCAN\_DIR="package" SCAN\_NAME="package" SCAN\_DEPS="$(TOPDIR)/include/package\*.mk $(TOPDIR)/overlay/\*/\*.mk" SCAN\_DEPTH=5 SCAN\_EXTRA=""
5. $(\_SINGLE)$(NO\_TRACE\_MAKE) -j1 -r -s -f include/scan.mk SCAN\_TARGET="targetinfo" SCAN\_DIR="target/linux" SCAN\_NAME="target" SCAN\_DEPS="profiles/\*.mk $(TOPDIR)/include/kernel\*.mk $(TOPDIR)/include/target.mk" SCAN\_DEPTH=2 SCAN\_EXTRA="" SCAN\_MAKEOPTS="TARGET\_BUILD=1"
6. **for** type in package target; **do** \
7. f=tmp/.$${type}info; t=tmp/.config-$${type}.in; \
8. [ "$$t" -nt "$$f" ] || ./scripts/metadata.pl $${type}\_config "$$f" > "$$t" || { rm -f "$$t"; echo "Failed to build $$t"; **false**; **break**; }; \
9. done
10. [ tmp/.config-feeds.in -nt tmp/.packagefeeds ] || ./scripts/feeds feed\_config > tmp/.config-feeds.in
11. ./scripts/metadata.pl package\_mk tmp/.packageinfo > tmp/.packagedeps || { rm -f tmp/.packagedeps; **false**; }
12. ./scripts/metadata.pl package\_feeds tmp/.packageinfo > tmp/.packagefeeds || { rm -f tmp/.packagefeeds; **false**; }
13. touch $(TOPDIR)/tmp/.build
15. .config: ./scripts/config/conf $(**if** $(CONFIG\_HAVE\_DOT\_CONFIG),,prepare-tmpinfo)
16. @+**if** [ \! -e .config ] || ! grep CONFIG\_HAVE\_DOT\_CONFIG .config >/dev/null; then \
17. [ -e $(HOME)/.openwrt/defconfig ] && cp $(HOME)/.openwrt/defconfig .config; \
18. $(\_SINGLE)$(NO\_TRACE\_MAKE) menuconfig $(PREP\_MK); \
19. fi

staging\_dir/host/.prereq-build: include/prereq-build.mk toplevel.mk:140

检查编译依赖



1. staging\_dir/host/.prereq-build: include/prereq-build.mk
2. mkdir -p tmp
3. rm -f tmp/.host.mk
4. @$(\_SINGLE)$(NO\_TRACE\_MAKE) -j1 -r -s -f $(TOPDIR)/include/prereq-build.mk prereq 2>/dev/null || { \
5. echo "Prerequisite check failed. Use FORCE=1 to override."; \
6. **false**; \
7. }
8. ifneq ($(realpath $(TOPDIR)/include/prepare.mk),)
9. @$(\_SINGLE)$(NO\_TRACE\_MAKE) -j1 -r -s -f $(TOPDIR)/include/prepare.mk prepare 2>/dev/null || { \
10. echo "Preparation failed."; \
11. **false**; \
12. }
13. endif
14. touch $@

export MAKEFLAGS= ;make V=ss -j1 -r -s -f /home/share/openwrt\_cc\_v3.0/include/prereq-build.mk prereq 2>/dev/null || { \

echo "Prerequisite check failed. Use FORCE=1 to override."; \

false; \

}

make 命令

返回值：

make命令的返回值

make命令执行完成后，可以依据make的返回值查询make运行的结果。可以通过shell命令“echo $?”查看make的返回值。make返回值有以下3种：

1）0：表示成功执行。

2）1：如果make运行时出现任何错误，返回1。

3）2：如果使用了make的-q选项，并且make使得部分目标不需要更新，则返回2。

### tools/Makefile分析

该Makefile决定了哪些工具是需要编译的，通过tools-y来指示编译

CONFIG\_EXTERNAL\_TOOLCHAIN 定义在.config里 CONFIG\_EXTERNAL\_TOOLCHAIN=y

代表使用厂商提供的外部工具链，否则自定义工具链

CONFIG\_GCC\_USE\_GRAPHITE 未定义

tools-=gmp mpfr mpc libelf wrt350nv2-builder upslug2 upx qemu elftosb mtools dosfstools lzma-old squashfs b43-tools ppl cloog sparse

tools-y= m4 libtool autoconf automake flex bison pkg-config sed mklibs sstrip make-ext4fs e2fsprogs mtd-utils mkimage firmware-utils patch-image patch quilt yaffs2 flock padjffs2 mm-macros missing-macros xz cmake scons bc findutils gengetopt patchelf lzma squashfs4

root@8e6fe2c8ddff:/home/share/openwrt\_cc\_v3.0# echo "m4 libtool autoconf automake flex bison pkg-config sed mklibs sstrip make-ext4fs e2fsprogs mtd-utils mkimage firmware-utils patch-image patch quilt yaffs2 flock padjffs2 mm-macros missing-macros xz cmake scons bc findutils gengetopt patchelf lzma squashfs4"|sed -s "s/ /\n/g"|wc -l

32

对tools/Makefile $(call subdir,$(curdir))进行展开得到目标tools/install



tools\_enabled = $(foreach tool,$(sort $(tools-y) $(tools-)),$(if $(filter $(tool),$(tools-y)),y,n))

1. tools\_enabled=y y n y y n y n y n y y y y y n n y y n y y y y y y n n y n y y y y y n n y y y n n y y n n n y y

$(call stampfile,$(curdir),tools,install,,\_$(subst $(space),,$(tools\_enabled)))

1. tools/stamp-install:=/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/stamp/.tools\_install\_yynyynynynyyyyynnyynyyyyyynnynyyyyynnyyynnyynnnyy
2. $(tools/stamp-install): /home/share/openwrt\_cc\_v3.0/tmp/.build
3. @+/home/share/openwrt\_cc\_v3.0/scripts/timestamp.pl -n $(tools/stamp-install) tools  || make --no-print-directory $(tools/flags-install) tools/install
4. @mkdir -p $$(dirname $(tools/stamp-install))
5. @touch $(tools/stamp-install)
7. $(**if** ,,.SILENT: $(tools/stamp-install))
9. .PRECIOUS: $(tools/stamp-install) # work around a make bug
11. tools//clean:=tools/stamp-install/clean
12. tools/stamp-install/clean: FORCE
13. @rm -f $(tools/stamp-install)

&&是左边返回0（执行成功）再执行右边，和编程语言的逻辑且比较像。

||是左边返回非0（执行失败）再执行右边，和编程语言的逻辑或比较像。

但这些是shell命令的特色不是makefile的。

最终执行 make tools/install

实验：

make /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/stamp/.tools\_install\_yynyynynynyyyyynnyynyyyyyynnynyyyyynnyyynnyynnnyy -j1 V=s

遍历tools目录下所有的文件，找到一个最新的文件（通过stat命令得到time），和/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/stamp/.tools\_install\_yynyynynynyyyyynnyynyyyyyynnynyyyyynnyyynnyynnnyy比较，如果/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/stamp/.tools\_install\_yynyynynynyyyyynnyynyyyyyynnynyyyyynnyyynnyynnnyy更新返回0

timestamp.pl -n AAA BBB 比较两者哪个更新，AAA更新返回0，否则返回1

### tools/patch 规则生成

对HostBuild进行展开



主要功能为：软件包自动化下载，打补丁，配置，编译，安装

tools/Makefile

1. tools/patch/compile:  /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/.prepared /home/share/openwrt\_cc\_v3.0/staging\_dir/host/.prepared

4. @+ $(SUBMAKE) -r -C tools/patch compile BUILD\_VARIANT=""

7. # aliases



12. tools/patch/install:  tools/patch/compile

15. @+ $(SUBMAKE) -r -C tools/patch install BUILD\_VARIANT=""

### toolchain/wrapper 规则生成

对HostBuild进行展开



### make prepare 日志

make prepare -j1 V=s > log.txt 2>&1 日志如下



1. make[1]: Entering directory '/home/share/openwrt\_cc\_v3.0'
2. make[2]: Entering directory '/home/share/openwrt\_cc\_v3.0'
3. + mkdir -p /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc
4. + cd /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc
5. + mkdir -p bin lib include stamp
6. mkdir -p /home/share/openwrt\_cc\_v3.0/build\_dir/target-arm-27912x-linux-uclibc/stamp
7. touch /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/.prepared
8. + mkdir -p /home/share/openwrt\_cc\_v3.0/staging\_dir/host
9. + cd /home/share/openwrt\_cc\_v3.0/staging\_dir/host
10. + mkdir -p bin lib include stamp
11. mkdir -p /home/share/openwrt\_cc\_v3.0/build\_dir/host/stamp /home/share/openwrt\_cc\_v3.0/staging\_dir/host/include/sys
12. install -m0644 /home/share/openwrt\_cc\_v3.0/tools/include/\*.h /home/share/openwrt\_cc\_v3.0/staging\_dir/host/include/
13. install -m0644 /home/share/openwrt\_cc\_v3.0/tools/include/sys/\*.h /home/share/openwrt\_cc\_v3.0/staging\_dir/host/include/sys/
14. ln -sf lib /home/share/openwrt\_cc\_v3.0/staging\_dir/host/lib64
15. touch /home/share/openwrt\_cc\_v3.0/staging\_dir/host/.prepared
16. make[3]: Entering directory '/home/share/openwrt\_cc\_v3.0/tools/patch'
17. make -j1 -C /home/share/openwrt\_cc\_v3.0/build\_dir/host/patch-2.7.5

第19392行是编译toolchain目标

19387 mkdir -p /home/share/openwrt\_cc\_v3.0/staging\_dir/host/stamp

19388 touch /home/share/openwrt\_cc\_v3.0/build\_dir/host/squashfs4.2/.built

19389 touch /home/share/openwrt\_cc\_v3.0/staging\_dir/host/stamp/.squashfs4\_installed

19390 make[3]: Leaving directory '/home/share/openwrt\_cc\_v3.0/tools/squashfs4'

19391 make[2]: Leaving directory '/home/share/openwrt\_cc\_v3.0'

19392 make[2]: Entering directory '/home/share/openwrt\_cc\_v3.0'

19393 + mkdir -p /home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-arm-27912x-linux-uclibc

19394 + cd /home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-arm-27912x-linux-uclibc

19395 + ln -nsf lib lib64

19396 + ln -nsf lib lib32

19397 + mkdir -p stamp lib usr/include usr/lib

19398 make[3]: Entering directory '/home/share/openwrt\_cc\_v3.0/toolchain/wrapper'

19399 Testing external toolchain **for** softfloat support ... ok

## package编译（testapp)

### package makefile

对package/Makefile进行展开

$(call stampfile,$(curdir),package,prereq,.config)

1. package/stamp-prereq:=/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/stamp/.package\_prereq
2. $(package/stamp-prereq): /home/share/openwrt\_cc\_v3.0/tmp/.build .config
3. @+/home/share/openwrt\_cc\_v3.0/scripts/timestamp.pl -n $(package/stamp-prereq) package .config || make --no-print-directory $(package/flags-prereq) package/prereq
4. @mkdir -p $$(dirname $(package/stamp-prereq))
5. @touch $(package/stamp-prereq)
7. $(**if** ,,.SILENT: $(package/stamp-prereq))
9. .PRECIOUS: $(package/stamp-prereq) # work around a make bug
11. package//clean:=package/stamp-prereq/clean
12. package/stamp-prereq/clean: FORCE
13. @rm -f $(package/stamp-prereq)

$(call stampfile,$(curdir),package,cleanup,$(TMP\_DIR)/.build)

1. package/stamp-cleanup:=/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/stamp/.package\_cleanup
2. $(package/stamp-cleanup): /home/share/openwrt\_cc\_v3.0/tmp/.build /home/share/openwrt\_cc\_v3.0/tmp/.build
3. @+/home/share/openwrt\_cc\_v3.0/scripts/timestamp.pl -n $(package/stamp-cleanup) package /home/share/openwrt\_cc\_v3.0/tmp/.build || make --no-print-directory $(package/flags-cleanup) package/cleanup
4. @mkdir -p $$(dirname $(package/stamp-cleanup))
5. @touch $(package/stamp-cleanup)
7. $(**if** ,,.SILENT: $(package/stamp-cleanup))
9. .PRECIOUS: $(package/stamp-cleanup) # work around a make bug
11. package//clean:=package/stamp-cleanup/clean
12. package/stamp-cleanup/clean: FORCE
13. @rm -f $(package/stamp-cleanup)

$(call stampfile,$(curdir),package,compile,$(TMP\_DIR)/.build)

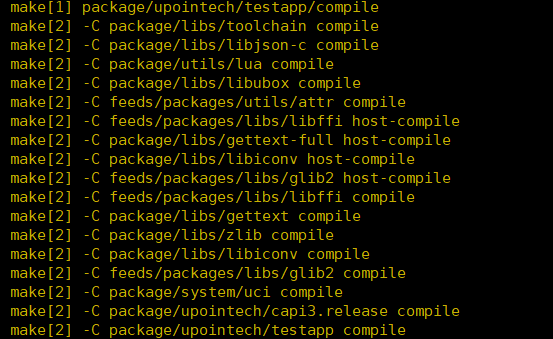
1. package/stamp-compile:=/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/stamp/.package\_compile
2. $(package/stamp-compile): /home/share/openwrt\_cc\_v3.0/tmp/.build /home/share/openwrt\_cc\_v3.0/tmp/.build
3. @+/home/share/openwrt\_cc\_v3.0/scripts/timestamp.pl -n $(package/stamp-compile) package /home/share/openwrt\_cc\_v3.0/tmp/.build || make --no-print-directory $(package/flags-compile) package/compile
4. @mkdir -p $$(dirname $(package/stamp-compile))
5. @touch $(package/stamp-compile)
7. $(**if** ,,.SILENT: $(package/stamp-compile))
9. .PRECIOUS: $(package/stamp-compile) # work around a make bug
11. package//clean:=package/stamp-compile/clean
12. package/stamp-compile/clean: FORCE
13. @rm -f $(package/stamp-compile)

$(call stampfile,$(curdir),package,install,$(TMP\_DIR)/.build)

1. package/stamp-install:=/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/stamp/.package\_install
2. $(package/stamp-install): /home/share/openwrt\_cc\_v3.0/tmp/.build /home/share/openwrt\_cc\_v3.0/tmp/.build
3. @+/home/share/openwrt\_cc\_v3.0/scripts/timestamp.pl -n $(package/stamp-install) package /home/share/openwrt\_cc\_v3.0/tmp/.build || make --no-print-directory $(package/flags-install) package/install
4. @mkdir -p $$(dirname $(package/stamp-install))
5. @touch $(package/stamp-install)
7. $(**if** ,,.SILENT: $(package/stamp-install))
9. .PRECIOUS: $(package/stamp-install) # work around a make bug
11. package//clean:=package/stamp-install/clean
12. package/stamp-install/clean: FORCE
13. @rm -f $(package/stamp-install)

### testapp编译流程

make -f Makefile package/upointech/testapp/compile -j1 V=s



package/upointech/testapp/compile是由主Makefile通过 include package/Makefile 再对package/Makefile展开得到



依赖关系图

### testapp编译日志



Prepared:创建编译testappv3-1.0.0的目录，复制package下源代码到指定编译目录。

1. mkdir -p /home/share/openwrt\_cc\_v3.0/build\_dir/target-arm-27912x-linux-uclibc/testappv3-1.0.0
2. cp -fpR ./src/\* /home/share/openwrt\_cc\_v3.0/build\_dir/target-arm-27912x-linux-uclibc/testappv3-1.0.0
3. touch /home/share/openwrt\_cc\_v3.0/build\_dir/target-arm-27912x-linux-uclibc/testappv3-1.0.0/.prepared\_3d7d44711cbd509ef1ac96034fa00c2e

### testapp编译规则

1. compile: prepare-package-install /home/share/openwrt\_cc\_v3.0/bin/zx279128/packages/base/testappv3\_1.0.0\_zx279128.ipk /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/testappv3.provides /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/root-zx279128/stamp/.testappv3\_installed /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/testapp.**default**.install.testappv3 /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/stamp/.testappv3\_installed

include/package.mk, line 332

1. prepare-package-install:
2. @mkdir -p $(PKG\_INFO\_DIR)
3. @touch $(PKG\_INSTALL\_STAMP).clean
4. @echo "$(filter-out essential,$(PKG\_FLAGS))" > $(PKG\_INSTALL\_STAMP).flags
5. ###################################################
6. rules.mk:117  PKG\_INFO\_DIR = /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo

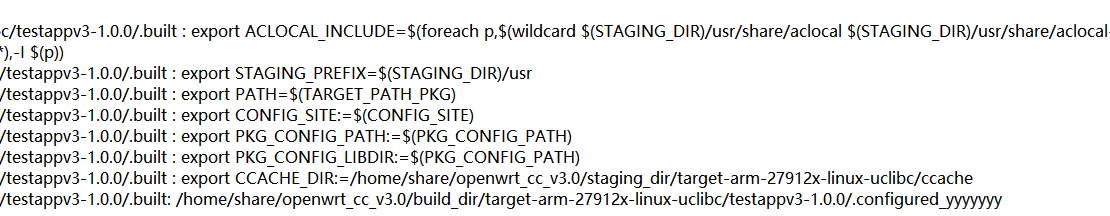
package.mk:130 PKG\_INSTALL\_STAMP = /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/testapp.**default**.install

echo "" > /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/testapp.default.install.flags

1. $(IPKG\_testappv3) : export CONTROL=$(Package/testappv3/CONTROL)
2. $(IPKG\_testappv3) : export DESCRIPTION=$(Package/testappv3/description)
3. $(IPKG\_testappv3): /home/share/openwrt\_cc\_v3.0/build\_dir/target-arm-27912x-linux-uclibc/testappv3-1.0.0/.built /home/share/openwrt\_cc\_v3.0/include/package-ipkg.mk
4. @rm -rf $(PDIR\_testappv3)/testappv3\_\* $(IDIR\_testappv3)
5. mkdir -p /home/share/openwrt\_cc\_v3.0/bin/zx279128/packages $(IDIR\_testappv3)/CONTROL /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo
6. install -d -m0755 $(IDIR\_testappv3)/opt/apps/testappv3/files
7. install -m0755 /home/share/openwrt\_cc\_v3.0/build\_dir/target-arm-27912x-linux-uclibc/testappv3-1.0.0/bin/testappv3 $(IDIR\_testappv3)/opt/apps/testappv3/files/testappv3
8. -find $(IDIR\_testappv3) -name 'CVS' -o -name '.svn' -o -name '.#\*' -o -name '\*~'| xargs -r rm -rf
9. @( find $(IDIR\_testappv3) -name lib\\*.so\\* -or -name \\*.ko | awk -F/ '{ print $$NF }'; **for** file in $(patsubst %,/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/%.provides,$(IDEPEND\_testappv3)); **do** **if** [ -f "$$file" ]; then cat $$file; fi; done;  ) | sort -u > /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/testappv3.provides
11. @( rm -f /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/testappv3.missing; ( export READELF=arm-buildroot-linux-uclibcgnueabi-readelf OBJCOPY=arm-buildroot-linux-uclibcgnueabi-objcopy XARGS="xargs -r"; /home/share/openwrt\_cc\_v3.0/scripts/gen-dependencies.sh "$(IDIR\_testappv3)"; ) | **while** read **FILE**; **do** grep -qxF "$$FILE" /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/testappv3.provides || echo "$$FILE" >> /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/testappv3.missing; done; **if** [ -f "/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/testappv3.missing" ]; then echo "Package testappv3 is missing dependencies for the following libraries:" >&2; cat "/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/testappv3.missing" >&2; **false**; fi; )
13. export CROSS="arm-buildroot-linux-uclibcgnueabi-" NO\_RENAME=1 ; NM="arm-buildroot-linux-uclibcgnueabi-nm" STRIP="arm-buildroot-linux-uclibcgnueabi-strip --strip-all" STRIP\_KMOD="/home/share/openwrt\_cc\_v3.0/scripts/strip-kmod.sh" PATCHELF="/home/share/openwrt\_cc\_v3.0/staging\_dir/host/bin/patchelf" /home/share/openwrt\_cc\_v3.0/scripts/rstrip.sh $(IDIR\_testappv3)
14. (cd $(IDIR\_testappv3)/CONTROL; ( echo "$$CONTROL"; printf "Description: "; echo "$$DESCRIPTION" | sed -e 's,^[[:space:]]\*, ,g'; ) > control; chmod 644 control; ( echo "#!/bin/sh"; echo "[ \"\$${IPKG\_NO\_SCRIPT}\" = \"1\" ] && exit 0"; echo ". \$${IPKG\_INSTROOT}/lib/functions.sh"; echo "default\_postinst \$$0 \$$@"; ) > postinst; ( echo "#!/bin/sh"; echo ". \$${IPKG\_INSTROOT}/lib/functions.sh"; echo "default\_prerm \$$0 \$$@"; ) > prerm; chmod 0755 postinst prerm;  )
16. ifneq ($(KEEP\_testappv3),)
17. @( keepfiles=""; **for** x in $(KEEP\_testappv3); **do** [ -f "$(IDIR\_testappv3)/$$x" ] || keepfiles="$${keepfiles:+$$keepfiles }$$x"; done; [ -z "$$keepfiles" ] || { mkdir -p $(IDIR\_testappv3)/lib/upgrade/keep.d; **for** x in $$keepfiles; **do** echo $$x >> $(IDIR\_testappv3)/lib/upgrade/keep.d/testappv3; done; }; )
18. endif
20. install -d -m0755 $(PDIR\_testappv3)
21. /home/share/openwrt\_cc\_v3.0/scripts/ipkg-build -c -o 0 -g 0 $(IDIR\_testappv3) $(PDIR\_testappv3)
22. @[ -f $(IPKG\_testappv3) ]

对第21行展开：/home/share/openwrt\_cc\_v3.0/scripts/ipkg-build -c -o 0 -g 0 /home/share/openwrt\_cc\_v3.0/build\_dir/target-arm-27912x-linux-uclibc/testappv3-1.0.0/ipkg-zx279128/testappv3 /home/share/openwrt\_cc\_v3.0/bin/zx279128/packages/base

生成bin/zx279128/packages/base/testappv3\_1.0.0\_zx279128.ipk



TARGET\_PATH\_PKG

rules.mk:170 (origin: makefile) TARGET\_PATH\_PKG = /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/host/bin:/home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-arm-27912x-linux-uclibc/bin:/opt/zte/cross\_compiler/arm-linux-uclibcgnueabi/bin:/home/share/openwrt\_cc\_v3.0/staging\_dir/host/bin:/home/share/openwrt\_cc\_v3.0/staging\_dir/host/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

### package/libs/toolchain

对package/libs/toolchain/Makefile进行展开

1. compile: prepare-package-install /home/share/openwrt\_cc\_v3.0/bin/zx279128/packages/base/libc\_-1\_zx279128.ipk /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/libc.provides /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/root-zx279128/stamp/.libc\_installed /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/toolchain.**default**.install.libc /home/share/openwrt\_cc\_v3.0/bin/zx279128/packages/base/libgcc\_-1\_zx279128.ipk /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/libgcc.provides /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/root-zx279128/stamp/.libgcc\_installed /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/toolchain.**default**.install.libgcc /home/share/openwrt\_cc\_v3.0/bin/zx279128/packages/base/libpthread\_-1\_zx279128.ipk /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/libpthread.provides /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/root-zx279128/stamp/.libpthread\_installed /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/toolchain.**default**.install.libpthread /home/share/openwrt\_cc\_v3.0/bin/zx279128/packages/base/librt\_-1\_zx279128.ipk /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/librt.provides /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/root-zx279128/stamp/.librt\_installed /home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo/toolchain.**default**.install.librt

include/package.mk

330 .PHONY: prepare-package-install

331 prepare-package-install:

332         @mkdir -p $(PKG\_INFO\_DIR)   #/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/pkginfo

333         @touch $(PKG\_INSTALL\_STAMP).clean

334         @echo "$(filter-out essential,$(PKG\_FLAGS))" > $(PKG\_INSTALL\_STAMP).flags

335

336 $(PACKAGE\_DIR):

337         mkdir -p $@

338

339 dumpinfo:

340 download:

341 prepare:

342 configure:

343 compile: prepare-package-install

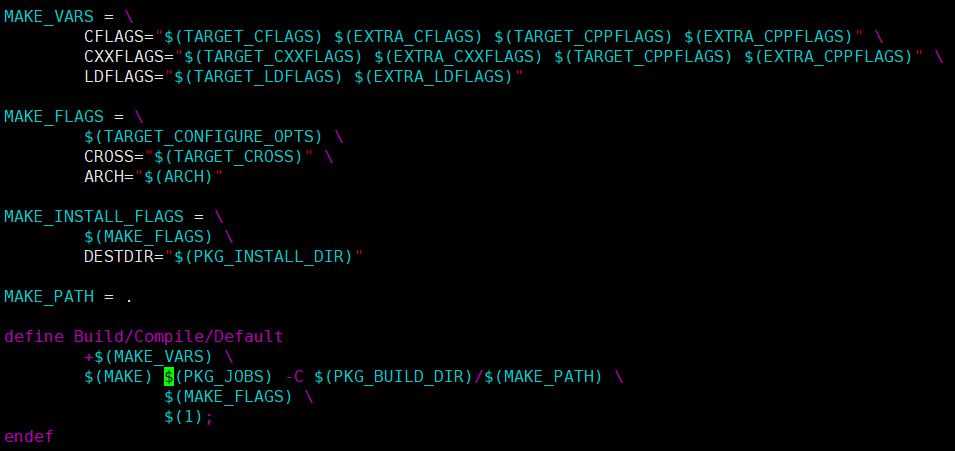
344 install: compile

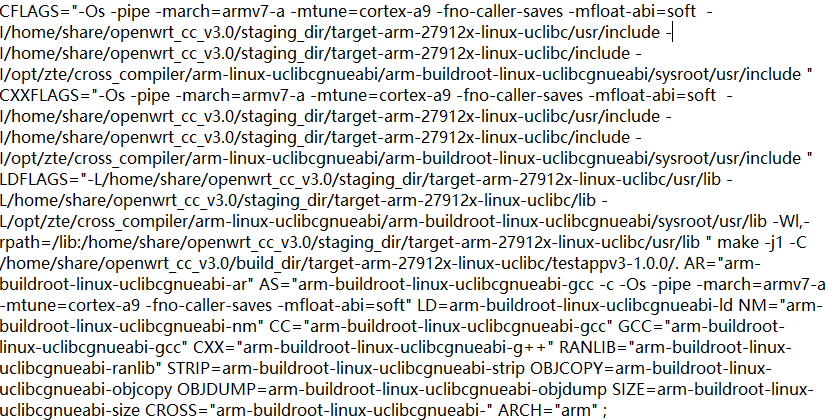
### testapp调试信息添加

在testapp/Makefile文件里添加

$(warning "\*\*steven1 MAKE\_VARS=$(MAKE\_VARS)")

MAKE\_VARS定义在include/package-defaults.mk





DESTDIR:

<https://blog.csdn.net/AlxRose/article/details/7756222>

## package编译（opmaintain)

### opmaintain编译步骤

make menuconfig 选择

Upointech ---> opmaintainv3

### Package Makefile修改

DEPENDS:=+glib2 +libcapi3 +libubox +jansson +libuci +c-ares +mosquitto +libpcap

package/Makefile的日志



Mosquitto 展开Makefile



Opmaintain 展开Makefile



1. compile: prepare-package-install /home/share/openwrt\_cc\_v3.0/bin/bcm6838/packages/base/opmaintainv3\_3.4.21\_bcm6838.ipk /home/share/openwrt\_cc\_v3.0/staging\_dir/target-mips-bcm6848-linux-uclibc/pkginfo/opmaintainv3.provides /home/share/openwrt\_cc\_v3.0/staging\_dir/target-mips-bcm6848-linux-uclibc/root-bcm6838/stamp/.opmaintainv3\_installed /home/share/openwrt\_cc\_v3.0/staging\_dir/target-mips-bcm6848-linux-uclibc/pkginfo/opmaintain.**default**.install.opmaintainv3 /home/share/openwrt\_cc\_v3.0/staging\_dir/target-mips-bcm6848-linux-uclibc/stamp/.opmaintainv3\_installed

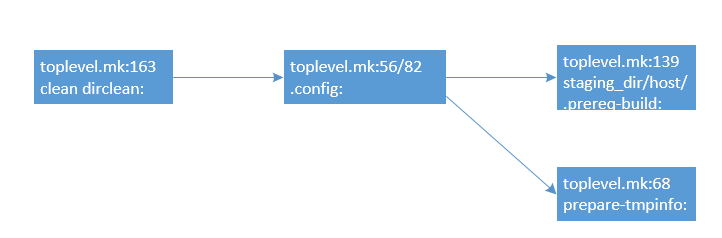
prepare-package-install package.mk:331

### opmaintain编译流程

$(warning "\*\*steven2\*\*\*subdir=$(call subdir,$(curdir))")



## make clean流程



.config合并后：

.config: scripts/config/conf prepare-tmpinfo staging\_dir/host/.prereq-build

## 研究BuildPackage



BuildPackage的第一行就是：$(Build/IncludeOverlay)，其定义：

define Build/IncludeOverlay

$(eval -include $(wildcard $(TOPDIR)/overlay/\*/$(PKG\_NAME).mk))

define Build/IncludeOverlay

endef

endef

Build/IncludeOverlay作用就是

inlcude (TOPDIR)/overlay/\*/(PKG\_NAME).mk 文件。

比如我要更改lua这个包里面的某个变量，如果我们直接修改 package/xx/lua/Makefile 文件，总有不妥之处。因为这个文件是从网上源码上同步下来的。我们最好不要去直修改它们。若需要用户自定义某些内容，不防使用overlay。就是在 trunk/overlay/ 路径下的任意子目录下建立与包名对应的 .mk 文件，比如lua，那么就建 lua.mk。

我们在 trunk/overlay/xxx/lua.mk 文件里重新定义我们的变量或操作。

$(eval $(Package/Default))

导入默认的配置项，Package/Default 定义在 include/package-defaults.mk 中，其中对大量的变量进行默认赋值

$(eval $(Package/$(1)))

导入库中Makefile定义的变量。如果是lua库，那么 Package/$(1)，即是 Package/lua 变量。在这个变量里，我们通常要定义 TITLE, CATEGORY, SECTION, VERSION 等变量。

$(STAMP\_PREPARED): $$(if $(QUILT)$(DUMP),,$(call find\_library\_dependencies,$(DEPENDS)))

如果 (QUILT)(QUILT)(DUMP) 都没有定义，那么以$(DEPENDS)为参引用 find\_library\_dependencies 变量。

其中 find\_library\_dependencies 定义如下：

find\_library\_dependencies = $(wildcard $(patsubst %,$(STAGING\_DIR)/pkginfo/%.version, \

$(filter-out $(BUILD\_PACKAGES),$(foreach dep, \

$(filter-out @%, $(patsubst +%,%,$(1))), \

$(if $(findstring :,$(dep)), \

$(word 2,$(subst :,$(space),$(dep))), \

$(dep) \

) \

))))

暂不解释，有点复杂，略过。其功能应该是找出依赖的库。

生成：

XXXX/.prepared : aa bb cc dd

$(foreach FIELD, TITLE CATEGORY SECTION VERSION,

ifeq ($($(FIELD)),)

$$(error Package/$(1) is missing the $(FIELD) field)

endif

)

这几句是判断 FIELD, TITLE, CATEGORY, SECTION, VERSION 这几个变量有没有定义，如果没有定义那就就报错。

$(if $(DUMP), \

$(Dumpinfo/Package), \

$(foreach target, \

$(if $(Package/$(1)/targets),$(Package/$(1)/targets), \

$(if $(PKG\_TARGETS),$(PKG\_TARGETS), ipkg) \

), $(BuildTarget/$(target)) \

) \

)

$(DUMP)为空，$(Dumpinfo/Package) 也为空

$(Package/$(1)/targets) 为空，$(PKG\_TARGETS) 也为空

$(foreach target, ipkg, $(BuildTarget/$(target))

调用了package-ipkg.mk:91的BuildTarget/ipkg

虽然写得有点复杂，但意思是：

将 Package/(1)/targets 或 PKG\_TARGETS 中的每一项或 ipkg 作为(1)/targets或PKG TARGETS中的每一项或ipkg作为(target)，引用 $(BuildTarget/$(target))。其中：

BuildTarget/bin 定义在 include/package-bin.mk

BuildTarget/ipkg 定义在 include /package-ipkg.mk

具体这两个变量的内容暂时不去研究。其内容莫非是描述如何生成bin与ipk目标。这里留个引子，下次再研究。

如果没有定义 Package/$(1)/targets 与 PKG\_TARGETS，那么上次就默认将ipk作为target，引用 BuildTarget/ipk 变量了。

$(if $(PKG\_HOST\_ONLY)$(DUMP),,$(call Build/DefaultTargets,$(1)))

如果没有定义 PKG\_HOST\_ONLY 与 DUMP，那么这句可以简化为：

$(call Build/DefaultTargets,$(1))

即默认的目标构建。

如果在自己的Makefile里没有定义Build/Prepare，会使用package.mk里的

Build/Prepare=$(call Build/Prepare/Default,) 入参为空

## 编译依赖过程

package/upointech/opmaintain/Makefile

DEPENDS:=+glib2 +libcapi3 +libubox +jansson +libuci +c-ares +mosquitto +libpcap

将依赖的库，加入到DEPENDS

1. package/upointech/opmaintain/compile: package/feeds/packages/c-ares/compile package/feeds/packages/glib2/compile package/feeds/packages/jansson/compile package/feeds/packages/mosquitto/compile package/libs/libpcap/compile package/libs/libubox/compile package/libs/toolchain/compile package/system/uci/compile package/upointech/capi3.release/compile

## ./configure常用参数

–prefix=/home/carl：指定安装位置，如果安装位置不指定，默认路径为/usr/local下的bin、lib等文件夹

–build：就是你正在使用的机器，查看本机使用的可以在configure目录下执行./config.guess

–host：就是你编译好的程序可以运行的机器

–target：就是你编译好的程序可以处理的其他平台

比如你build的机器是x86，想运行的host机器是arm，程序可以处理的程序target是mips

CC：指定编译工具链的名字

CFLAGS：指定头文件（.h文件）的路径，如：CFLAGS=-I/usr/include -I/path/include

LDFLAGS：gcc 等编译器会用到的一些优化参数，也可以在里面指定库文件的位置。用法：LDFLAGS=-L/usr/lib -L/path/to/your/lib

LIBS：告诉链接器要链接哪些库文件，如LIBS = -lpthread -liconv

--prefix=PREFIX

把所有文件安装在PREFIX中而不是默认的路径（如 /usr/local/xxx 中）。默认是"/"或者是"/usr/local"等等。。

以下还有部分参数细化 PREFIX 的其它目录

--exec-prefix=EXEC-PREFIX

制定体系相关的文件的安装位置，默认是PREFIX

--bindir=DIRECTORY

--sbindir=SBINDIR

可执行文件的安装位置。默认是PREFIX/bin和PREFIX/sbin

--datadir=DIRECTORY

指定所安装的只读文件的目录，默认是PREFIX/share

--sysconfdir=SYS-CONFDIR

用于各种配置文件，默认是PREFIX/etc，也可能是PREFIX/var

--libdir=LIBDIR

库文件目录，默认EXEC-PREFIX/lib

--includedir=INCLUDEDIR

C/C++头文件的目录，默认是PREFIX/include

--docdir=DOCDIR

文档文件（除了man）被安装的路径，默认PREFIX/doc

--mandir=MANDIR

Manual为之，默认是PREFIX/man

--with-includes=WITH-INC

一系列冒号分割的目录，这些会被加入-I选项中

--with-libraries=WITH-LIB

一系列冒号分割的目录，指定编译时搜索的库的位置

--enable-locale

打开区域/地区支持。一般不用

--enable-multibyte

允许多字节编码，用于支持东亚字体

--with-pgport=NUMBER

有些项目会有绑定自定义的端口，这里就需要指定。

--with-xxx

--without-xxx

--enable-xxx

--disable-xxx

启用/禁用库里面的某一些feature

--program-prefix=PROG\_PREFIX

--program-suffix=PROG\_SUFFIX

制定目标可执行文件的前缀和后缀，默认为空。比如我们编译tar这个命令，指定了前缀为“gnu-”，后缀为“.bin”，那么最终生成的可执行命令为“gnu-tar.bin”

--program-transform-name=PROG-TRANS

这里指定了一个sed参数，生成程序时经过sed -e PROG-TRANS来重命名目标

CC=...

指定编译器，默认为gcc、cc或者HOST-gcc

## ipk的打包过程

scripts/ipkg-build

1. tmp\_dir=$dest\_dir/IPKG\_BUILD.$$
2. mkdir $tmp\_dir
4. echo $CONTROL > $tmp\_dir/tarX
5. # Preserve permissions (-p) when creating data.tar.gz as non-root user
6. ( cd $pkg\_dir && $TAR $ogargs -X $tmp\_dir/tarX --format=gnu -czpf $tmp\_dir/data.tar.gz . )
7. echo "pkg\_dir=$pkg\_dir"
8. echo "TAR=$TAR"
9. echo "ogargs=$ogargs"
11. installed\_size=`stat -c "%s" $tmp\_dir/data.tar.gz`
12. sed -i -e "s/^Installed-Size: .\*/Installed-Size: $installed\_size/" \
13. $pkg\_dir/$CONTROL/control
15. ( cd $pkg\_dir/$CONTROL && $TAR $ogargs --format=gnu -czf $tmp\_dir/control.tar.gz . )
16. rm $tmp\_dir/tarX
18. echo "2.0" > $tmp\_dir/debian-binary
20. pkg\_file=$dest\_dir/${pkg}\_${version}\_${arch}.ipk
21. rm -f $pkg\_file
22. ( cd $tmp\_dir && $TAR --format=gnu -zcf $pkg\_file ./debian-binary ./data.tar.gz ./control.tar.gz )
24. rm $tmp\_dir/debian-binary $tmp\_dir/data.tar.gz $tmp\_dir/control.tar.gz
25. rmdir $tmp\_dir
27. echo "Packaged contents of $pkg\_dir into $pkg\_file"

tmp\_dir=/home/share/openwrt\_cc\_v3.0/bin/zx279128/packages/base/IPKG\_BUILD.2892

pkg\_dir=/home/share/openwrt\_cc\_v3.0/build\_dir/target-arm-27912x-linux-uclibc/testappv3-1.0.0/ipkg-zx279128/testappv3

TAR=/bin/tar

ogargs=--owner=0 --group=0

可以对ipk文件进行tar解压操作

## Makefile里用到的linux命令

xzcat tar gawk

if命令

## 设置Makefile编译选项，并传递到CMakeList

.config 里

CONFIG\_VERSION\_NUMBER="2.1" 2.1为SDK 3.0 1.1为SDK2.0

U01 Makefile里

ifeq ($(CONFIG\_VERSION\_NUMBER),"1.1")

CMAKE\_OPTIONS += -DFRAMEWORK2=ON

Endif

CMakeList.txt里

if(FRAMEWORK2)

ADD\_DEFINITIONS(-DFRAMEWORK2)

SET(APP\_NAME u01)

SET(CAPI\_LIB capi)

SET(LIBCAPI libcapi)

else()

SET(APP\_NAME u01v3)

SET(CAPI\_LIB capi3)

SET(LIBCAPI libcapi3)

endif()

## CPPFLAGS LDFLAGS CFLAGS选项详解

CC 与 CXX

这是 C 与 C++ 编译器命令。默认值一般是 “gcc” 与 “g++”。这个变量本来与优化没有关系，但是有些人因为担心软件包不遵守那些约定俗成的规范，害怕自己苦心设置的 CFLAGS/CXXFLAGS/LDFLAGS 之类的变量被忽略了，而索性将原本应当放置在其它变量中的选项一股老儿塞到 CC 或 CXX 中，比如：CC=”gcc -march=k8 -O2 -s”。这是一种怪异的用法，本文不提倡这种做法，而是提倡按照变量本来的含义使用变量。

CPPFLAGS

这是用于预处理阶段的选项。不过能够用于此变量的选项，看不出有哪个与优化相关。如果你实在想设一个，那就使用下面这两个吧：

-DNDEBUG

“NDEBUG”是一个标准的 ANSI 宏，表示不进行调试编译。

-D\_FILE\_OFFSET\_BITS=64

大多数包使用这个来提供大文件(>2G)支持。

CFLAGS 与 CXXFLAGS

CFLAGS 表示用于 C 编译器的选项，CXXFLAGS 表示用于 C++ 编译器的选项。这两个变量实际上涵盖了编译和汇编两个步骤。大多数程序和库在编译时默认的优化级别是”2″(使用”-O2″选项)并且带有调试符号来编 译，也就是 CFLAGS=”-O2 -g”, CXXFLAGS=$CFLAGS 。事实上，”-O2″已经启用绝大多数安全的优化选项了。另一方面，由于大部分选项可以同时用于这两个变量，所以仅在最后讲述只能用于其中一个变量的选 项。[提醒]下面所列选项皆为非默认选项，你只要按需添加即可。

rules.mk:175

TARGET\_CFLAGS = -Os -pipe -march=armv7-a -mtune=cortex-a9 -fno-caller-saves -mfloat-abi=soft

rules.mk:164

TARGET\_LDFLAGS= -L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib -L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/lib -L/opt/zte/cross\_compiler/arm-linux-uclibcgnueabi/arm-buildroot-linux-uclibcgnueabi/sysroot/usr/lib

https://blog.csdn.net/shenhuxi\_yu/article/details/79788760

# 定制OpenWrt系统及添加自开发软件

## 简介

OpenWrt 是一个比较完善的嵌入式Linux开发平台，在无线路由器应用上已有4000多个软件包。用户可以按照OpenWrt的约定增加开源软件或自行开发的软件。加入软件包需要在package目录下创建一个目录，以包含该软件包的各种信息和与OpenWrt建立联系的文件。然后创建一个Makefile与OpenWrt建立联系，Makefile需要遵循OpenWrt的约定。另外可以创建一个patchs目录保存patch文件，对下载的源代码进行适量修改。

## OpenWrt使用三个Makefile的子文件

OpenWrt使用三个Makefile的子文件，分别为：

include $(TOPDIR)/rules.mk

include $(INCLUDE\_DIR)/kernel.mk

include $(INCLUDE\_DIR)/package.mk

由这些Makefile子文件确定软件包加入OpenWrt的方式和方法。$(TOPDIR)/rules.mk一般在Makefile的开头，$(INCLUDE\_DIR)/kernel.mk文件对于软件包为内核时是不可缺少的，$(INCLUDE\_DIR)/package.mk一般在软件包的基本信息完成后再引入。

## 软件包的基本信息

软件包的信息均以PKG\_开头，其意思和作用如下：

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

PKG\_VERSION 表示软件包版本号。

PKG\_RELEASE 表示Makefile的版本号。

PKG\_SOURCE 表示源代码的文件名。

PKG\_SOURCE\_URL 表示源代码的下载网站位置。@SF表示在sourceforge网站，@GNU表示在GNU网站，还有@GONE、@KERNEL。

PKG\_MD5SUM 表示源代码文件的校验码。用于核对软件包是否下载正确。

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

PKG\_BUILD\_DIR 表示软件包编译目录。它的父目录为$(BUILD\_DIR)。如果不指定，默认为$(BUILD\_DIR)/$(PKG\_NAME)/$(PKG\_VERSION)。

## 应用程序编译包定义

应用程序的编译包以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。下面使用$(PKG\_NAME)只是做一个标志，并非真正使用$(PKG\_NAME)，如Package/$(PKG\_NAME)。

SECTION 表示包的类型，预留。

CATRGORY 表示分类，在make menuconfig的菜单下将可以找到。

TITLE 用于软件包的简短描述。

DESCRIPTION 用于软件包的详细描述，已放弃使用。如果使用DESCRIPTION将会提示“error DESCRIPTION:= is obsolete, use Package/PKG\_NAME/description”。

URL 表示软件包的下载位置。

MAINTAIER 表示维护者，选项。

DEPENDS 表示与其他软件的依赖。即编译或安装其他软件时需要说明。如果存在多个依赖，则每个依赖需要用空格分开。依赖前使用+号表示默认为显示，即对象没有选中时也会显示，使用@则默认为不显示，即当依赖对象选中后才显示。

在用户空间的应用程序软件包中没有内核驱动模块的AUTOLOAD参数。如果应用软件需要在boot时自动运行，则需要在/etc/init.d中增加相应的脚本文件。脚本文件需要START参数，说明在boot时的优先级，如果在boot过程启动后再关闭，则需要进一步设置STOP参数。如果STOP参数存在，其值必须大于START。脚本文件需要start()和stop()两个函数，start()是执行程序，stop()是关闭程序。关闭程序一般需要执行killall命令。由/etc/rc.d/S10boot知道，装载内核驱动模块的优先级为10，需要使用自己设计的内核驱动模块的程序其START的值必须大于10。同样由/etc/rc.d/S40network知道，使用网络通信的程序其START的值必须大于40。

Package/$(PKG\_NAME)/conffiles 本包安装的配置文件，一行一个。如果文件结尾使用/，则表示为目录。用于备份配置文件说明，在sysupgrade命令执行时将会用到。

Package/$(PKG\_NAME)/description 软件包的详细描述，取代前面提到的DESCRIPTION详细描述。

Build/Prepare 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。一般的准备方法为：

define Build/Prepare

mkdir -p $(PKG\_BUILD\_DIR)

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

Endef

按OpenWrt的习惯，一般把自己设计的程序全部在src目录下。

Build/Compile 编译方法，没有特别说明的可以不予以定义。如果不定义将默认使用编译方法Build/Compile/Default。

自行开发的软件包可以考虑使用下面的定义。

define Build/Compile

$(MAKE) -C $(PKG\_BUILD\_DIR) \

$(TARGET\_CONFIGURE\_OPTS) CFLAGS="$(TARGET\_CFLAGS) -I $(LINUX\_DIR)/include"

endef

Package/$(PKG\_NAME)/install 软件包的安装方法，包括一系列拷贝编译好的文件到指定位置。调用时会带一个参数，就是嵌入式系统的镜像文件系统目录，因此$(1)表示嵌入式系统的镜像目录。一般可以采用下面的方法：

define Package/$(PKG\_NAME)/install

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

$(INSTALL\_DIR) $(PKG\_BUILD\_DIR)/ $(PKG\_NAME) $(1)/usr/bin/

endef

INSTALL\_DIR、INSTALL\_BIN在$(TOPDIR)/rules.mk文件定义，所以本Makefile必须引入$(TOPDIR)/rules.mk文件。

INSTALL\_DIR:=install -d -m0755 意思是创建所属用户可读写和执行，其他用户可读可执行的目录。

INSTALL\_BIN:=install -m0755 意思是编译好的文件存放到镜像文件目录。

如果用户空间的应用软件在boot时要自动运行，则需要在安装方法说明中增加自动运行的脚本文件安装和配置文件安装方法。

例如：

define Package/mountd/install

$(INSTALL\_DIR) $(1)/sbin/ $(1)/etc/config/ $(1)/etc/init.d/

$(INSTALL\_BIN) $(PKG\_BUILD\_DIR)/mountd $(1)/sbin/

$(INSTALL\_DATA) ./files/mountd.config $(1)/etc/config/mountd

$(INSTALL\_BIN) ./files/mountd.init $(1)/etc/init.d/mountd

endef

安装文件放在files子目录下，不要与源代码文件目录src混在一起，以提高可读性。使用清晰的文件扩展名，更方便安装识别文件。

Package/$(PKG\_NAME)/preinst 软件包安装前处理方法，使用脚本语言，因此定义的第一行需要下面的格式

#!/bin/sh

调用时带入的参数为嵌入式系统的镜像目录。

Package/$(PKG\_NAME)/postinst 软件包安装后处理方法，使用脚本语言。

Package/$(PKG\_NAME)/prerm 软件包删除前处理方法，使用脚本语言。

Package/$(PKG\_NAME)/postrm 软件包删除后处理方法，使用脚本语言。

## 内核驱动模块包定义

Linux分为内核空间和用户空间。开发者开发的内核部分可以直接加入Linux的Kernel程序，也可以生成内核模块以便需要时装入内核。OpenWrt一般希望开发者生成内核模块，在Linux启动后自动装载或手工使用insmod命令装载。内核模块使用KernelPackage开头，其他与一般应用软件包基本相同。

在内核驱动模块定义中增加了：

SUBMENU 表示子菜单位置，在$(INCLUDE)/kernel.mk对内核模块定义了CATEGORY为kernel modules，所以内核模块在menuconfig中的主菜单为kernel modules，然后有下一级子菜单$(SUBMENU)。在子菜单下可以看到以kmod-$(PKG\_NAME)项目。

DEFAULT 表示直接编入内核或产生内核模块，y表示直接编入内核，m表示产生内核模块。

AUTOLOAD 表示自动装入内核，一般表示方法为：

AUTOLOAD:=$(call AutoLoad, $(PRIORITY),$(AUTOLOAD\_MODS))

AutoLoad的第一个参数$(PRIORITY)为优先级，01为最优先，99为最后装载。有关自动装载可以在/etc/modules.d目录下看到，第二个参数$(AUTOLOAD\_MODS)模块名，每个模块名以空格符分隔。即可同时装载多个内核模块。

在开发过程最好不要使用自动装载，经过严格调试后再使用，可以减轻调试的工作量。

使用定义

完成前面定义后，必须使用eval函数实现各种定义。其格式为：

对于一般应用软件包

$(eval $(call Package, $(PKG\_NAME)))

或对于内核驱动模块

$(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驱动模块，上面使用PKG\_NAME需要灵活变通。eval函数可以设计多个。也可以当成多个软件包处理。

## 编译变量定义

INSTALL\_xxx变量在$(TOPDIR)/rules.mk中定义

INSTALL\_BIN:=install -m0755

INSTALL\_DIR:=install -d -m0755

INSTALL\_DATA:=install -m0644

INSTALL\_CONF:=install -m0600

<https://blog.csdn.net/weixin_43025071/article/details/85251123>

# SDK打patch分析

在framework\_v3.0\_R3910.sh 基础上打framework\_patch\_r4001.tgz，就和framework\_v3.0\_R4001.sh是一样的

Linux patch命令

<https://www.runoob.com/linux/linux-comm-patch.html>

diff test2/openwrt\_cc\_v3.0/toolchain/Config.in openwrt\_cc\_v3.0/toolchain/Config.in > Config\_R4320.patch

patch -p0 testfile1 testfile.patch

patch -p0 package/upointech/testapp/src/testapp2.c testapp\_R4320.patch

patch -p0 confs/zx279131.config zx279131\_R4320.patch

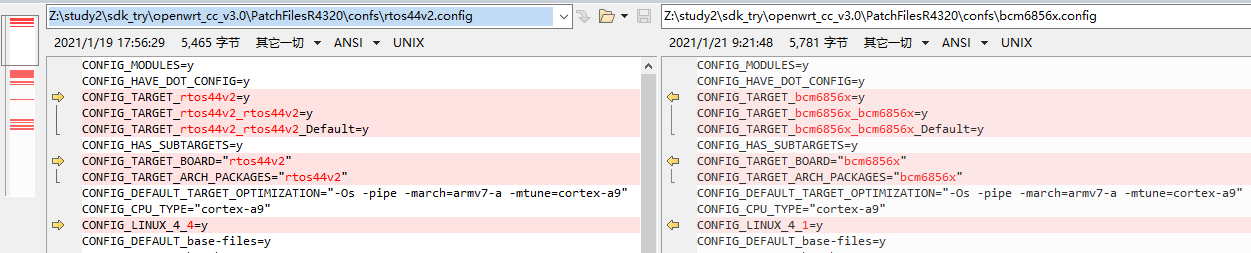
patch -p0 toolchain/Config.in Config\_R4320.patch

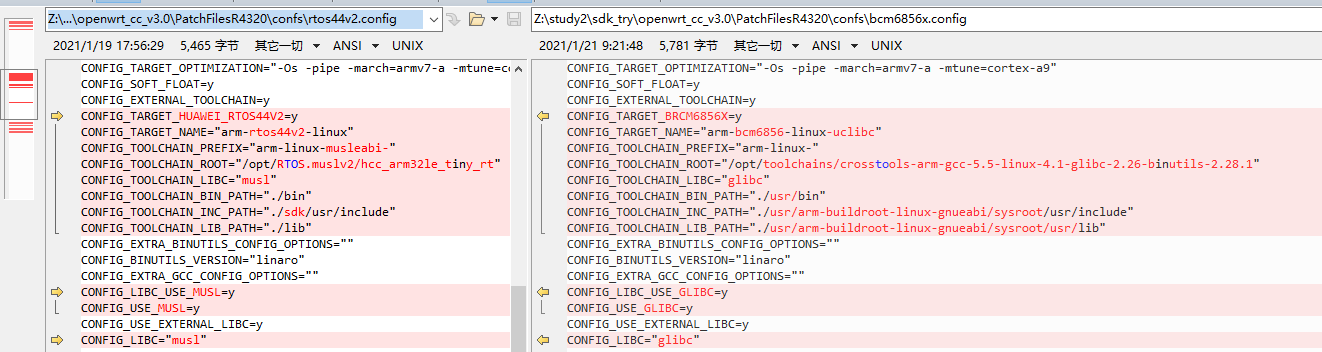
# SDK升级分析

通过厂家原始工具链RTOS4.4\_207.6.tar.gz 与有点适配好的比较分析。

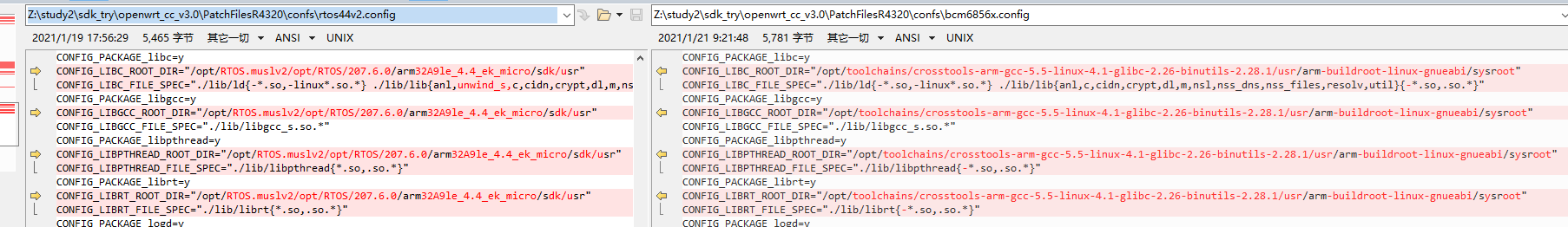
## 配置文件修改

### 配置文件修改点









### 内核版本设置

rtos44v2.config CONFIG\_LINUX\_4\_4=y

opt/RTOS/207.6.0/etc/RTOS\_BASE-Revision 有一行linux-4.4-build

grep "linux" /opt/zte/cross\_compiler/arm-linux-uclibcgnueabi/version

linux kernel-4.1.25

这个配置不影响，中间件和插件都不需要编内核

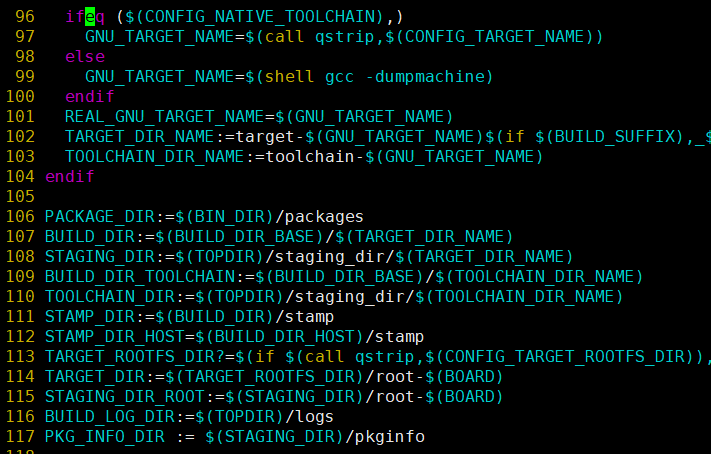
### libc设置

目前嵌入式里有3种libc可以选择：uClibc 、 Glibc 、Musl-libc

根据工具链判断

### CONFIG\_TARGET\_NAME

rules.mk



build\_dir/target-arm-27912x-linux-uclibc/opmaintainv3-1

对应.config CONFIG\_TARGET\_NAME="arm-27912x-linux-uclibc"

# cmake交叉编译

现在移动开发越老越火，我们免不了会将一些项目porting到Android／iOS平台，这个时候就要用到交叉编译：即在你host宿主机上(例如你用的Ubuntu电脑)要生成target目标机(例如Android手机)的程序。在编译的过程中会涉及到相关头文件的切换和编译器的选择以及环境变量的改变等，今天就来看看CMake是如何来做交叉编译的。

首先需要明确以下几点：

CMake不能自动判断出目标机系统，需要我们指定。

一般情况下Build出来的可执行文件是不能直接运行在宿主机上。

编译过程中不能用宿主机上的原声头文件和库，而是需要用到一套不同的头文件和库。

## 设置目标系统以及Toolchain

* CMAKE\_SYSTEM\_NAME:

在toolchain脚本中必须要设置的变量，只有当CMAKE\_SYSTEM\_NAME这个变量被设置了，CMake才认为此时正在交叉编译，它会额外设置一个变量CMAKE\_CROSSCOMPILING为TRUE。

CMAKE\_SYSTEM\_NAME即目标机target所在的操作系统名称，比如ARM或者Linux你就需要写”Linux”,如果Android平台你就写”Android”,如果你的嵌入式平台没有相关OS你即需要写成”Generic”.

* CMAKE\_SYSTEM\_PROCESSOR:

这个是可选项，但是在移动开发中很重要，代表目标系统的硬件或者CPU型号，例如ARM，X86 etc。

* CMAKE\_C\_COMPILER:

即C语言编译器，这里可以将变量设置成完整路径或者文件名

* CMAKE\_CXX\_COMPILER:

C++编译器。

1. (cd /home/share/openwrt\_cc\_v3.0/build\_dir/target-arm-27912x-linux-uclibc/testappv3-1.0.0; CFLAGS="-Os -pipe -march=armv7-a -mtune=cortex-a9 -fno-caller-saves -mfloat-abi=soft " CXXFLAGS="-Os -pipe -march=armv7-a -mtune=cortex-a9 -fno-caller-saves -mfloat-abi=soft " LDFLAGS="-L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib -L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/lib -L/opt/zte/cross\_compiler/arm-linux-uclibcgnueabi/arm-buildroot-linux-uclibcgnueabi/sysroot/usr/lib -Wl,-rpath=/lib:/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib " cmake **-DCMAKE\_SYSTEM\_NAME=Linux -DCMAKE\_SYSTEM\_VERSION=1 -DCMAKE\_SYSTEM\_PROCESSOR=arm** -DCMAKE\_BUILD\_TYPE=Release -DCMAKE\_C\_FLAGS\_RELEASE="-DNDEBUG" -DCMAKE\_CXX\_FLAGS\_RELEASE="-DNDEBUG" **-DCMAKE\_C\_COMPILER="/home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-arm-27912x-linux-uclibc/bin/arm-buildroot-linux-uclibcgnueabi-gcc"** -DCMAKE\_C\_COMPILER\_ARG1="" **-DCMAKE\_CXX\_COMPILER="/home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-arm-27912x-linux-uclibc/bin/arm-buildroot-linux-uclibcgnueabi-g++"** -DCMAKE\_CXX\_COMPILER\_ARG1="" -DCMAKE\_ASM\_COMPILER="/home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-arm-27912x-linux-uclibc/bin/arm-buildroot-linux-uclibcgnueabi-gcc" -DCMAKE\_ASM\_COMPILER\_ARG1="" -DCMAKE\_EXE\_LINKER\_FLAGS:STRING="-L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib -L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/lib -L/opt/zte/cross\_compiler/arm-linux-uclibcgnueabi/arm-buildroot-linux-uclibcgnueabi/sysroot/usr/lib -Wl,-rpath=/lib:/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib" -DCMAKE\_MODULE\_LINKER\_FLAGS:STRING="-L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib -L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/lib -L/opt/zte/cross\_compiler/arm-linux-uclibcgnueabi/arm-buildroot-linux-uclibcgnueabi/sysroot/usr/lib -Wl,-rpath=/lib:/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib -Wl,-Bsymbolic-functions" -DCMAKE\_SHARED\_LINKER\_FLAGS:STRING="-L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib -L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/lib -L/opt/zte/cross\_compiler/arm-linux-uclibcgnueabi/arm-buildroot-linux-uclibcgnueabi/sysroot/usr/lib -Wl,-rpath=/lib:/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib -Wl,-Bsymbolic-functions" -DCMAKE\_AR="/home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-arm-27912x-linux-uclibc/bin/arm-buildroot-linux-uclibcgnueabi-ar" -DCMAKE\_NM="/home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-arm-27912x-linux-uclibc/bin/arm-buildroot-linux-uclibcgnueabi-nm" -DCMAKE\_RANLIB="/home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-arm-27912x-linux-uclibc/bin/arm-buildroot-linux-uclibcgnueabi-ranlib" -DCMAKE\_FIND\_ROOT\_PATH="/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc;/home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-arm-27912x-linux-uclibc;"/opt/zte/cross\_compiler/arm-linux-uclibcgnueabi"" -DCMAKE\_FIND\_ROOT\_PATH\_MODE\_PROGRAM=BOTH -DCMAKE\_FIND\_ROOT\_PATH\_MODE\_LIBRARY=ONLY -DCMAKE\_FIND\_ROOT\_PATH\_MODE\_INCLUDE=ONLY -DCMAKE\_STRIP=: -DCMAKE\_INSTALL\_PREFIX=/usr -DDL\_LIBRARY=/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc -DCMAKE\_PREFIX\_PATH=/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc -DCMAKE\_SKIP\_RPATH=TRUE  . )
2. SET(CMAKE\_SYSTEM\_NAME Linux)
3. SET(CMAKE\_SYSTEM\_PROCESSOR arm)
4. SET(TOOLCHAIN\_DIR "/opt/mv\_pro\_4.0/montavista/pro/devkit/arm/v5t\_le")
5. SET(CMAKE\_FIND\_ROOT\_PATH ${TOOLCHAIN\_DIR})
6. SET(CMAKE\_FIND\_ROOT\_PATH\_MODE\_PROGRAM NEVER)
7. SET(CMAKE\_FIND\_ROOT\_PATH\_MODE\_LIBRARY ONLY)
8. SET(CMAKE\_FIND\_ROOT\_PATH\_MODE\_INCLUDE ONLY)
10. SET(CMAKE\_C\_COMPILER ${TOOLCHAIN\_DIR}/bin/arm\_v5t\_le-gcc)
11. SET(CMAKE\_CXX\_COMPILER ${TOOLCHAIN\_DIR}/bin/arm\_v5t\_le-g++)
12. #告知当前使用的是交叉编译方式，必须配置
13. SET(CMAKE\_SYSTEM\_NAME Linux)
14. #指定编译工具，一定要设置
15. #或交叉编译器使用绝对地址
16. SET(CMAKE\_C\_COMPILER "arm-linux-gcc")
17. #指定C++交叉编译器
18. SET(CMAKE\_CXX\_COMPILER "arm-linux-g++")
20. #不一定需要设置
21. #指定交叉编译环境安装目录...
22. SET(CMAKE\_FIND\_ROOT\_PATH "...")
23. #从来不在指定目录下查找工具程序
24. SET(CMAKE\_FIND\_ROOT\_PATH\_MODE\_PROGRAM NEVER)
25. #只在指定目录下查找库文件
26. SET(CMAKE\_FIND\_ROOT\_PATH\_MODE\_LIBRARY ONLY)
27. #只在指定目录下查找头文件
28. SET(CMAKE\_FIND\_ROOT\_PATH\_MODE\_INCLUDE ONLY)

<https://blog.csdn.net/chenxiuli0810/article/details/88733084>

<https://blog.csdn.net/samssm/article/details/51479741>

## cmake参考资料

<https://cmake.org/cmake/help/v3.0/manual/cmake-variables.7.html>

<https://cmake.org/Wiki/CMake_Useful_Variables>

# 芯片与工具链对应关系

芯片型号，厂商，工具链名称等对应关系



# 自动编译脚本

./build.pl --target testapp

# 编译SDK碰到的问题

## 找不到json.h

1. [  2%] Building C object CMakeFiles/blobmsg\_json-**static**.dir/blobmsg\_json.c.o
2. /home/share/eos\_sdk/build\_dir/target-arm-hi5651-linux\_musl/libubox-2018-07-25-c83a84af/blobmsg\_json.c:21:19: fatal error: json.h: No such file or directory
3. #include <json.h>
4. ^
5. compilation terminated.
6. CMakeFiles/blobmsg\_json-**static**.dir/build.make:62: recipe **for** target 'CMakeFiles/blobmsg\_json-static.dir/blobmsg\_json.c.o' failed

原因：PKG\_SEARCH\_MODULE(JSONC json-c) 返回的JSONC\_INCLUDE\_DIRS 出错

修改：cp -rf target-arm-hi5651-linux\_musl/usr/include/json/\*

/opt/AP/armv8-hsan-linux-musleabi1.1.22/armv8-hsan-linux-musleabi/sysroot/usr/include/

## 帮助解决问题

2021-01-15 解决U01 2.0，3.0 Makefile合并编译问题

解决OP插件 libcurl库升级问题

## 找不到libreadline.so.4（en7526 en7528）

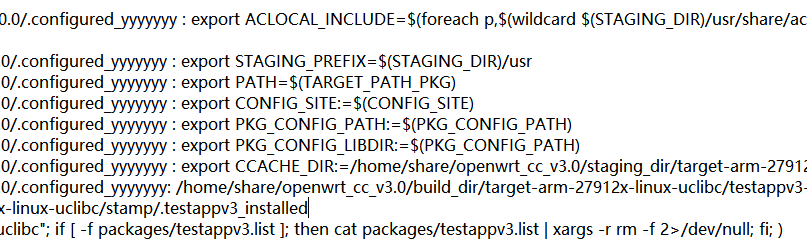
en7526 en7528

1. config.status: creating Makefile
2. gawk: error while loading shared libraries: libreadline.so.4: cannot open shared object file: No such file or directory
3. config.status: error: could not create Makefile
4. Makefile:59: recipe for target '/home/share/openwrt\_cc\_v3.0/build\_dir/target-mips-en7526-linux/json-c-0.12.1/.configured\_yyyyyyy' failed
5. make[2]: \*\*\* [/home/share/openwrt\_cc\_v3.0/build\_dir/target-mips-en7526-linux/json-c-0.12.1/.configured\_yyyyyyy] Error 1
6. make[2]: Leaving directory '/home/share/openwrt\_cc\_v3.0/package/libs/libjson-c'
7. package/Makefile:209: recipe for target 'package/libs/libjson-c/compile' failed
8. make[1]: \*\*\* [package/libs/libjson-c/compile] Error 2
9. make[1]: Leaving directory '/home/share/openwrt\_cc\_v3.0'
10. /home/share/openwrt\_cc\_v3.0/include/toplevel.mk:181: recipe for target 'package/upointech/testapp/compile' failed
11. make: \*\*\* [package/upointech/testapp/compile] Error 2

root@8e6fe2c8ddff:/home/share/openwrt\_cc\_v3.0# which gawk

/usr/bin/gawk

gawk需要libreadline.so的动态库，系统里找不到



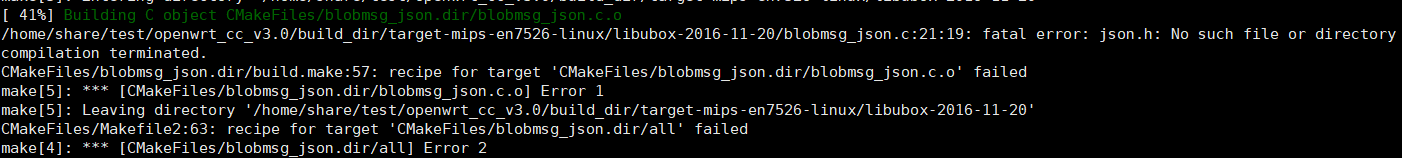
/home/share/openwrt\_cc\_v3.0/rules.mk:170 (origin: makefile) TARGET\_PATH\_PKG = /home/share/openwrt\_cc\_v3.0/staging\_dir/target-mips-en7526-linux/host/bin:/home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-mips-en7526-linux/bin:/opt/trendchip/mips-linux-glibc2.22-4.6.3/usr/bin:/home/share/openwrt\_cc\_v3.0/staging\_dir/toolchain-mips-en7526-linux/bin:/opt/trendchip/mips-linux-glibc2.22-4.6.3/usr/bin:/home/share/openwrt\_cc\_v3.0/staging\_dir/host/bin:/home/share/openwrt\_cc\_v3.0/staging\_dir/host/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

在/opt/trendchip/mips-linux-glibc2.22-4.6.3/usr/bin目录下也有一个gawk，并且排在/usr/bin/gawk之前

解决方法，删除/opt/trendchip/mips-linux-glibc2.22-4.6.3/usr/bin目录下的gawk

工具链需要重新打包

## 找不到json.h（en7526 en7528 mt7580)



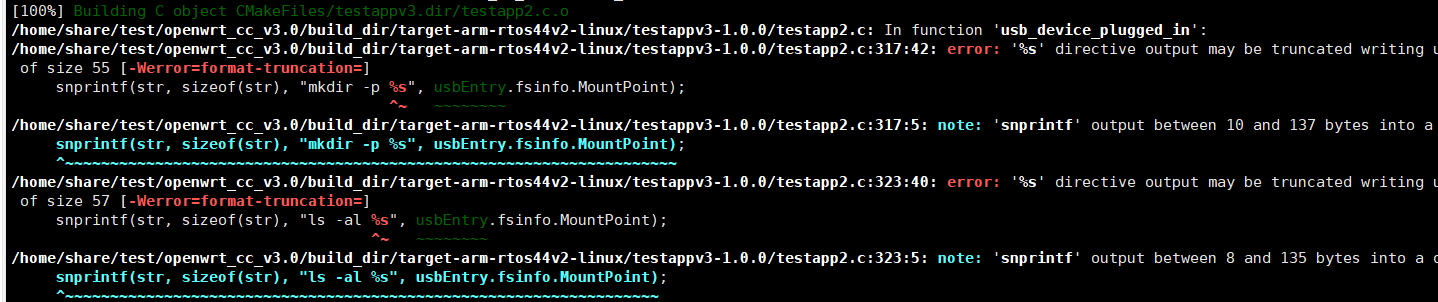
编译过程日志如下：

1. -- The C compiler identification is GNU 4.6.3
2. -- Check **for** working C compiler: /opt/trendchip/mips-linux-glibc2.22-4.6.3/usr/bin/mips-linux-gcc
3. -- Check **for** working C compiler: /opt/trendchip/mips-linux-glibc2.22-4.6.3/usr/bin/mips-linux-gcc -- works
4. -- Detecting C compiler ABI info
5. -- Detecting C compiler ABI info - done
6. -- Found PkgConfig: /opt/trendchip/mips-linux-glibc2.22-4.6.3/usr/bin/pkg-config (found version "0.27")
7. -- checking **for** one of the modules 'json-c'

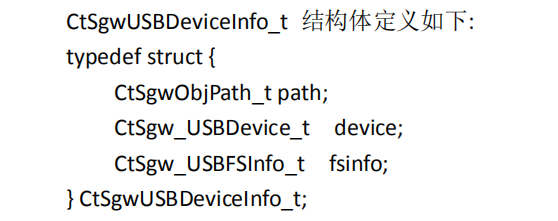
原因是使用了/opt/trendchip/mips-linux-glibc2.22-4.6.3/usr/bin/pkg-config 和 pkgconf

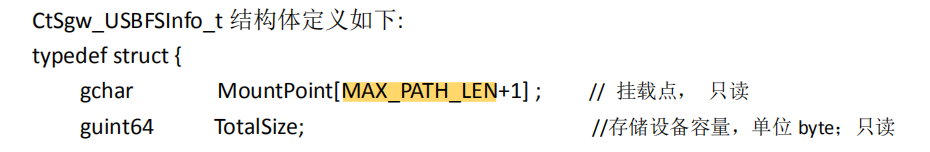
修改方法，删除

## 编译testapp失败（rtos44 rtos44v1 rtos44v2)



原因





MAX\_PATH\_LEN = 127

修改：testapp/src/testapp2.c

303 char str[64] = {0}; ---> char str[160] = {0};

## 找不到libpthread.so.0库（sd5117x） \*\*\*

1. CFLAGS="-Os -pipe -mno-branch-likely -fno-caller-saves -msoft-float  -I/home/share/test/openwrt\_cc\_v3.0/staging\_dir/target-mips-sd5117x-linux/usr/include -I/home/share/test/openwrt\_cc\_v3.0/staging\_dir/target-mips-sd5117x-linux/include -I/opt/huawei\_5117x/mips-linux-gnu/usr/include " CXXFLAGS="-Os -pipe -mno-branch-likely -fno-caller-saves -msoft-float  -I/home/share/test/openwrt\_cc\_v3.0/staging\_dir/target-mips-sd5117x-linux/usr/include -I/home/share/test/openwrt\_cc\_v3.0/staging\_dir/target-mips-sd5117x-linux/include -I/opt/huawei\_5117x/mips-linux-gnu/usr/include " LDFLAGS="-L/home/share/test/openwrt\_cc\_v3.0/staging\_dir/target-mips-sd5117x-linux/usr/lib -L/home/share/test/openwrt\_cc\_v3.0/staging\_dir/target-mips-sd5117x-linux/lib -L/opt/huawei\_5117x/lib -Wl,-rpath=/lib:/home/share/test/openwrt\_cc\_v3.0/staging\_dir/target-mips-sd5117x-linux/usr/lib " make -j1 -C /home/share/test/openwrt\_cc\_v3.0/build\_dir/target-mips-sd5117x-linux/testappv3-1.0.0/. AR="rlx-linux-gnu-ar" AS="rlx-linux-gnu-gcc -c -Os -pipe -mno-branch-likely -fno-caller-saves -msoft-float" LD=rlx-linux-gnu-ld NM="rlx-linux-gnu-nm" CC="rlx-linux-gnu-gcc" GCC="rlx-linux-gnu-gcc" CXX="rlx-linux-gnu-g++" RANLIB="rlx-linux-gnu-ranlib" STRIP=rlx-linux-gnu-strip OBJCOPY=rlx-linux-gnu-objcopy OBJDUMP=rlx-linux-gnu-objdump SIZE=rlx-linux-gnu-size CROSS="rlx-linux-gnu-" ARCH="mips" ;
2. Linking C executable bin/testappv3
3. /opt/huawei\_5117x/bin/../lib/gcc/mips-linux-gnu/8.4.0/../../../../mips-linux-gnu/bin/ld: warning: libpthread.so.0, needed by /home/share/test/openwrt\_cc\_v3.0/staging\_dir/target-mips-sd5117x-linux/usr/lib/libgio-2.0.so, not found (try using -rpath or -rpath-link)

原因是少了如下红色部分

TARGET\_LDFLAGS += -Wl,-rpath=/lib:$(STAGING\_DIR)/usr/lib

参考11.2.4

将/opt/huawei\_5117x/lib目录下对应的库复制到/home/share/test/openwrt\_cc\_v3.0/staging\_dir/target-mips-sd5117x-linux/usr/lib

原因是工具链有问题，LDFLAGS= 第一个-L启作用，而这些库在第三个-L路径

## C编译器不工作(zx279131)



原因是C编译器不支持CONFIG\_SOFT\_FLOAT 即选项-msoft-float

修改点：

在confs/zx279131.config增加一条CONFIG\_SOFT\_FLOAT=n

cat <<-EOT | /opt/arm-linux-uclibcgnueabi/bin/arm-linux-uclibcgnueabi-gcc -msoft-float -o /dev/null -x c -

int main(int argc, char \*\*argv)

{

double a = 0.1;

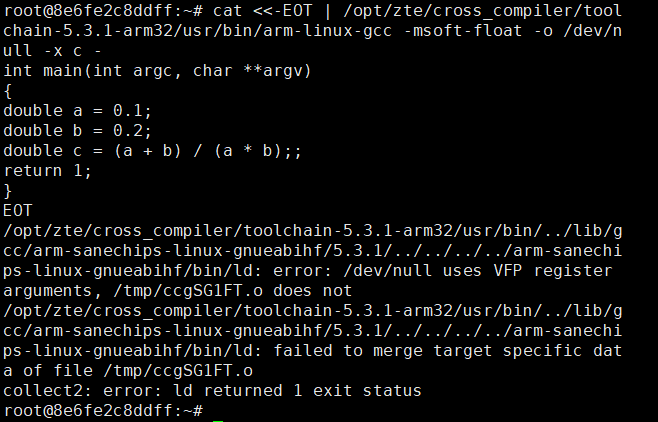
double b = 0.2;

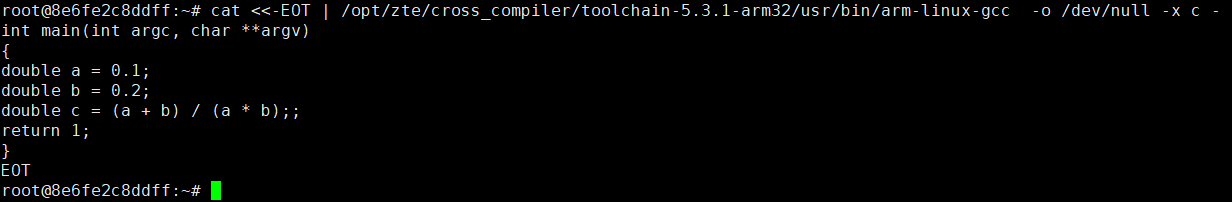
double c = (a + b) / (a \* b);;

return 1;

}

EOT





## Zlib问题（zx279127 zx279128） \*\*\*

Linking C executable bin/testappv3

/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib/libglib-2.0.so: warning: the use of OBSOLESCENT `utime' is discouraged, use `utimes'

/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib/libuci.so: warning: the use of `mktemp' is dangerous, better use `mkstemp'

/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib/libgio-2.0.so: undefined reference to `deflateSetHeader@ZLIB\_1.2.2'

/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib/libgio-2.0.so: undefined reference to `inflateGetHeader@ZLIB\_1.2.2'

collect2: error: ld returned 1 exit status

CMakeFiles/testappv3.dir/build.make:92: recipe for target 'bin/testappv3' failed

用工具链下的zlib库替换staging\_dir/target-arm-27912x-linux-uclibc/usr/lib下的zlib库

# Makefile知识点

## Makefile之变量

### CURDIR

CURDIR是Makefile的内嵌变量，自动设置为当前目录

## Makefile之规则

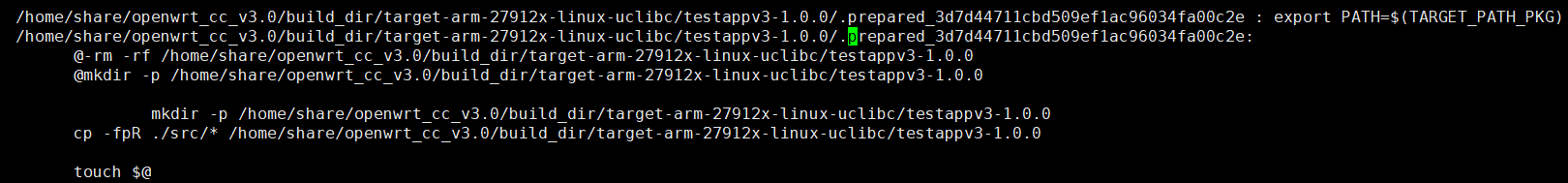
### .PRECIOUS

<https://www.cnblogs.com/gaojian/archive/2012/09/21/2696772.html>

### makefile同名目标处理

<https://blog.csdn.net/lixiangminghate/article/details/50448664>

### makefile中命令前加一个@



通常，make会把其要执行的命令行在命令执行前输出到屏幕上。当我们用“@”字符在命令

行前，那么，这个命令将不被make显示出来，最具代表性的例子是，我们用这个功能来像

屏幕显示一些信息。如：

@echo 正在编译XXX模块......

当make执行时，会输出“正在编译XXX模块......”字串，但不会输出命令，如果没有“@

”，那么，make将输出：

echo 正在编译XXX模块......

正在编译XXX模块......

如果make执行时，带入make参数“-n”或“--just-print”，那么其只是显示命令，但

不会执行命令，这个功能很有利于我们调试我们的Makefile，看看我们书写的命令是执行

起来是什么样子的或是什么顺序的。

而make参数“-s”或“--slient”则是全面禁止命令的显示。

makefile中在命令前加上-，则会忽略该命令的出错，继续执行下一条命令

### -Wl,-rpath的使用

1. LDFLAGS="-L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib -L/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/lib -L/opt/zte/cross\_compiler/arm-linux-uclibcgnueabi/arm-buildroot-linux-uclibcgnueabi/sysroot/usr/lib -Wl,-rpath=/lib:/home/share/openwrt\_cc\_v3.0/staging\_dir/target-arm-27912x-linux-uclibc/usr/lib "

<https://blog.csdn.net/zsy986212/article/details/89513976>

<https://www.cnblogs.com/homejim/p/8004883.html>

-L -Wl,-rpath-link -Wl,-rpath区别精讲

<http://www.manongjc.com/article/94831.html>

### LD时库混用

动态库，静态库混用

https://blog.csdn.net/nodeathphoenix/article/details/9058531?utm\_medium=distribute.pc\_relevant\_t0.none-task-blog-BlogCommendFromBaidu-1.not\_use\_machine\_learn\_pai&depth\_1-utm\_source=distribute.pc\_relevant\_t0.none-task-blog-BlogCommendFromBaidu-1.not\_use\_machine\_learn\_pai

## Makefile之函数

### findstring

$(findstring FIND,IN)

函数名称：查找字符串函数—findstring。

函数功能：搜索字串“IN”，查找“FIND”字串。

返回值：如果在“IN”之中存在“FIND” ，则返回“FIND”，否则返回空。

函数说明：字串“IN”之中可以包含空格、[Tab]。搜索需要是严格的文本匹配。

示例：

$(findstring a,a b c)

$(findstring a,b c)

第一个函数结果是字“a”；第二个值为空字符。

### if

$(if CONDITION,THEN-PART[,ELSE-PART])

¾ 函数功能：第一个参数“CONDITION” ，在函数执行时忽略其前导和结尾空字

符，如果包含对其他变量或者函数的引用则进行展开。如果“CONDITION”的

展开结果非空，则条件为真，就将第二个参数“THEN\_PATR”作为函数的计算

表达式；“CONDITION”的展开结果为空，将第三个参数“ELSE-PART”作为

函数的表达式，函数的返回结果为有效表达式的计算结果。

¾ 返回值：根据条件决定函数的返回值是第一个或者第二个参数表达式的计算结

果。当不存在第三个参数“ELSE-PART” ，并且“CONDITION”展开为空，函

数返回空。

¾ 函数说明：函数的条件表达式“CONDITION”决定了函数的返回值只能是

“THEN-PART”或者“ELSE-PART”两个之一的计算结果。

¾ 函数示例：

SUBDIR += $(if $(SRC\_DIR) $(SRC\_DIR),/home/src)

函数的结果是：如果“SRC\_DIR”变量值不为空，则将变量“SRC\_DIR”指定

的目录作为一个子目录；否则将目录“/home/src”作为一个子目录。

**rules.mk:78**

DL\_DIR:=$(if $(call qstrip,$(CONFIG\_DOWNLOAD\_FOLDER)),$(call qstrip,$(CONFIG\_DOWNLOAD\_FOLDER)),$(TOPDIR)/dl)

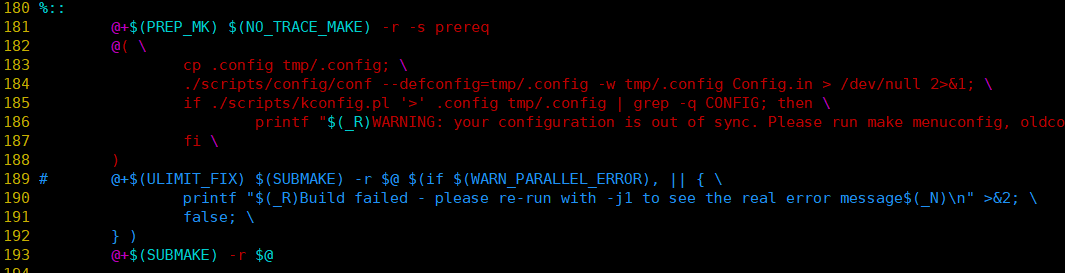
函数的结果是：如果$(call qstrip,$(CONFIG\_DOWNLOAD\_FOLDER))变量值不为空，返回$(call qstrip,$(CONFIG\_DOWNLOAD\_FOLDER))，否则返回$(TOPDIR)/dl

# Makefile调试方法

## gnu remake

**<http://bashdb.sourceforge.net/remake/>**

**remake -f Makefile prepare --debugger**



remake -f Makefile package/upointech/testapp/compile -j1 V=s --debugger

## Makefile中的$(error)和$(warning)

$(warning "\*\*steven2\*\*\*subdir=$(call subdir,$(curdir))")



$(error TEXT…)

$(warning TEXT…)

Cmake里的调试方法为message

message("STAGING\_DIR=${STAGING\_DIR}")

make prepare -j1 V=s > log.txt 2>&1

# 常用方法

## vi 中复制多行内容

第一步： ：set num

第二步 ： 查看要复制的内容从num1 到num2 之间

第三步：:num1 ,num2 w! >> /你要保存的地址/文件名

## find命令

find . -maxdepth 1 -type f |wc -l

## sed

sed -i '/#/d' .config.2

查找文件.config.2 ,匹配带#的行，并删除该行

sed -i '/^\s\*$/d' .config.2

## xargs

cat packages/testappv3.list | xargs -r rm -f 2>/dev/null

-r no-run-if-empty 当xargs的输入为空的时候则停止xargs，不用再去执行了。

# 参考资料

openwrt目录结构



<https://blog.csdn.net/u011554317/article/details/50614482>

<https://blog.csdn.net/weixin_43025071/article/details/85265049?utm_medium=distribute.pc_relevant.none-task-blog-searchFromBaidu-2.not_use_machine_learn_pai&depth_1-utm_source=distribute.pc_relevant.none-task-blog-searchFromBaidu-2.not_use_machine_learn_pai>

menuconfig流程

<https://blog.csdn.net/peixiuhui/article/details/90702979?utm_medium=distribute.pc_relevant.none-task-blog-title-10&spm=1001.2101.3001.4242>

主Makefile

<http://blog.chinaunix.net/uid-26675482-id-4704952.html>

开源openwrt编译过程

<https://www.cnblogs.com/elewei/p/9353908.html>

menuconfig

<https://blog.csdn.net/eastonwoo/article/details/8995363>

cmake编译选项

<https://www.cnblogs.com/lidabo/p/13878829.html>