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

## 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.

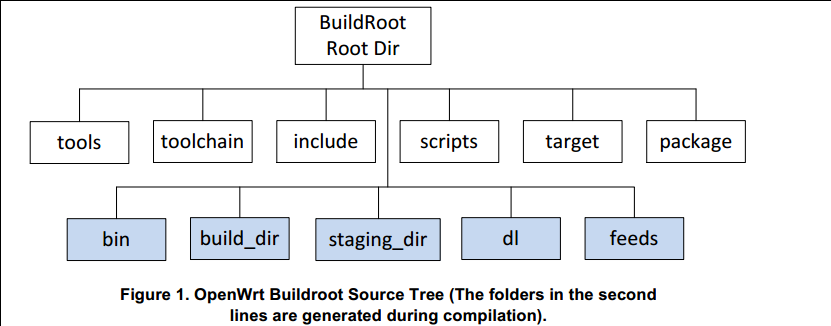
PermitRootLogin yes

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

./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

## make prepare

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

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

# 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。

# Makefile调试方法

## gnu remake

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

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

# 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>