Makefile Introduction

Android Build System

罗流毅

xluoly@gmail.com

2018-7-26

Agenda

手工编译

有这样一个 C 语言项目

```
calc
|-- calc.h
|-- getch.c
|-- getop.c
`-- stack.c
```

编译命令为

```
$ gcc -o calc main.c getch.c getop.c stack.c
```

缺点：

- 手工输入一长串命令
- 其中一个文件有修改就要编译所有文件，文件多的时候会很耗时

Makefile 一

最直接的 Makefile

```
calc: main.c getch.c getop.c stack.c
    gcc -o calc main.c getch.c getop.c stack.c
```

Makefile 二

定义一些变量方便管理

```
CC   = gcc
PROM = calc
SRC  = main.c getch.c getop.c stack.c

$(PROM): $(SRC)
    $(CC) -o $(PROM) $(SRC)
```

缺点:

- 其中一个文件有修改就要编译所有文件，文件多的时候会很耗时
- 如果改动 calc.h 并不会重新编译

Makefile 三

建立目标依赖关系

```
CC   = gcc
PROM = calc
DEPS = calc.h
OBJS = main.o getch.o getop.o stack.o

$(PROM): $(OBJS)
    $(CC) -o $(PROM) $(OBJS)

main.o: main.c $(DEPS)
    $(CC) -c main.c

getch.o: getch.c $(DEPS)
    $(CC) -c getch.c

getop.o: getop.c $(DEPS)
    $(CC) -c getop.c

stack.o: stack.c $(DEPS)
    $(CC) -c stack.c
```

Makefile 四

使用模式匹配，是 Makefile 更简洁，更容易维护

```
CC   = gcc
PROM = calc
DEPS = calc.h
OBJS = main.o getch.o getop.o stack.o

$(PROM): $(OBJS)
    $(CC) -o $(PROM) $(OBJS)

%.o: %.c $(DEPS)
    $(CC) -c $< -o $@
```

Makefile 五

加入 clean 目标，帮助清理编译产出物

```
CC   = gcc
PROM = calc
DEPS = $(wildcard *.h)
SRC  = $(wildcard *.c)
OBJS = $(SRC: %.c=%.o)

$(PROM): $(OBJS)
    $(CC) -o $(PROM) $(OBJS)

%.o: %.c $(DEPS)
    $(CC) -c $< -o $@

.PHONY: clean
clean:
    rm -rf $(OBJS) $(PROM)
```

Agenda

Makefile 简介

- make 是一个软件构建工具，也可以认为是软件项目管理工具
- make 通过读取 Makefile 来工作
- makefile 不是顺序执行的脚本，它是用于描述项目资源之间的组织关系，制定构建策略
- GUN make 搜索 makefile 文件名的顺序为 GNUmakefile -> makefile -> Makefile
- 可以用 `-f` 或 `--file` 参数来指定 makefile 文件

```
$ make -f rules.mk
```

或者

```
$ make --file=rules.mk
```

Makefile 规则

- targets: 第一行冒号之前的部分，可以是多个，以空格分隔，可以使用通配符
- prerequisites: 第一行冒号之后的部分，依赖的文件或目标列表，如果依赖有更新，targets 就认为是过时的，需要重新生成
- command: 第二行，行首必须是 Tab 键，不能是空格，每一行在单独的 shell 中执行

```
targets : prerequisites
    command
    ...
```

文件搜索路径

- VPATH 变量

```
VPATH = src:src/submodule1:src/submodule2:include
```

- vpath 关键字

```
vpath %.h include
vpath %.c src
vpath %.c src/submodule1
vpath %.c src/submodule2
```

GCC 头文件和库文件搜索

- -I 头文件路径
- -L 库文件路径

```
CFLAGS  += -Iinclude #头文件位于include目录中
LDFLAGS += -Llibs    #库文件位于libs目录中

$(PROM): $(OBJS)
    $(CC) -o $(LDFLAGS) $(PROM) $(OBJS) $(LIBS)

%.o: %.c $(DEPS)
    $(CC) -c $(CFLAGS) $< -o $@
```

伪目标

- .PHONY
- 伪目标不是真实文件，只是一个标签

```
.PHONY : clean
```

生成多个产物

- Makefile 中第一个目标为默认目标，通常用 `all` 表示
- 需要生成的产物对应的目标作为 `all` 的依赖

```
.PHONY: all
all: $(BIN) $(HEX)

$(BIN): $(OBJS)
    <command>

$(HEX): $(BIN)
    <command>
```

自动生成依赖

- 由 gcc 为每个源文件生成一个依赖规则文件 (.d)
- 将所有.d 文件 include 到 Makefile 文件中

```
%.d: %.c
    @set -e; rm -f $@; \
    $(CC) -M $(CPPFLAGS) $< > $@.$$$$; \
    sed 's,\($*\)\.o[ :]*,\1.o $@ : ,g' < $@.$$$$ > $@; \
    rm -f $@.$$$$

include $(SRC:.c=.d)
```

模式匹配

- 符号%
- 用于匹配文件文件名

```
%.o : %.c
```

变量赋值

- =：在执行时扩展，允许递归扩展

```
A = $(B)
B = foo
```

- :=：在定义时扩展

```
B = foo
A := $(B)
```

- ?=：在变量未赋值时才进行赋值

```
BUILE_TYPE ?= release #设置BUILD_TYPE默认值为release
```

- +=：追加赋值

```
CFLAGS += -D__DEBUG #编译参数中添加宏定义 __DEBUG
```

自动变量

- `$@`: 目标的完整名称
- `$<`: 依赖列表中的第一个名称，通常指用来构建目标所使用的源文件
- `$?`: 相比目标有更新的依赖列表
- `$*`: 指目标模式匹配符`%`匹配的部分

命令前缀

- -: 忽略该行命令返回状态，继续往下执行
- @: 禁止命令回显

```
all:
    @echo "Beginning build at:"
    @date
    @echo "--------"
```

如果没有加 @，会显示成下面这样:

```
echo "Beginning build at:"
Beginning build at:
date
Sun Jun 18 01:13:21 CDT 2006
echo "--------"
--------
```

引用其他 Makefile

- `include`: 被引用的 Makefile 找不到会出现致命错误而终止执行
- `-include`: 即使被引用的 Makefile 找不到也继续执行
- `sinclude`: 兼容性更好，作用同 `-include`

make 的工作方式

1. 读入所有的 Makefile
2. 读入被 include 的其它 Makefile
3. 初始化和计算文件中的变量
4. 推导隐晦规则，并分析所有规则
5. 为所有的目标文件创建依赖关系链
6. 根据依赖关系，决定哪些目标要重新生成
7. 执行命令

调试 Makefile

- `$(warning)` 或 `$(error)` 函数输出信息
- `make -n`：只是打印出命令，但是并不真正执行命令
- 增加目标规则调试变量如下，然后执行 make 时将调试的变量作为目标，如：`make OBJS`

```
%:
    @echo '$*=$($*)'

d-%:
    @echo '$*=$($*)'
    @echo '  origin = $(origin $*)'
    @echo '   value = $(value  $*)'
    @echo '  flavor = $(flavor $*)'
```

自动生成 Makefile

- autoconf/automake：GNU 构建系统工具，通常称为 Autotools，是软件项目有很好的移植性，通过简单的./configure、make 和 make install 三步骤完成程序的配置、编译和安装
- cmake：开源的跨平台自动化建构系统，配置文件为 CMakeLists.txt，生成标准的建构档（如 Unix 的 Makefile，Windows Visual Studio 项目文件和 xcode 项目文件）
- qmake：跨平台项目开发的构建工具，Qt 附带的工具之一。能够自动生成 Makefile、Microsoft Visual Studio 项目文件和 xcode 项目文件。不管源代码是否是用 Qt 写，都能使用

大型项目的管理 -- 递归 make

- 各层次各模块的 Makefile 都是独立完整的 Makefile
- 在一个项目 Makefile 的命令中进入子项目执行 make

```
subsystem:
    cd subdir && $(MAKE)
```

或者，等效于

```
subsystem:
    $(MAKE) -C subdir
```

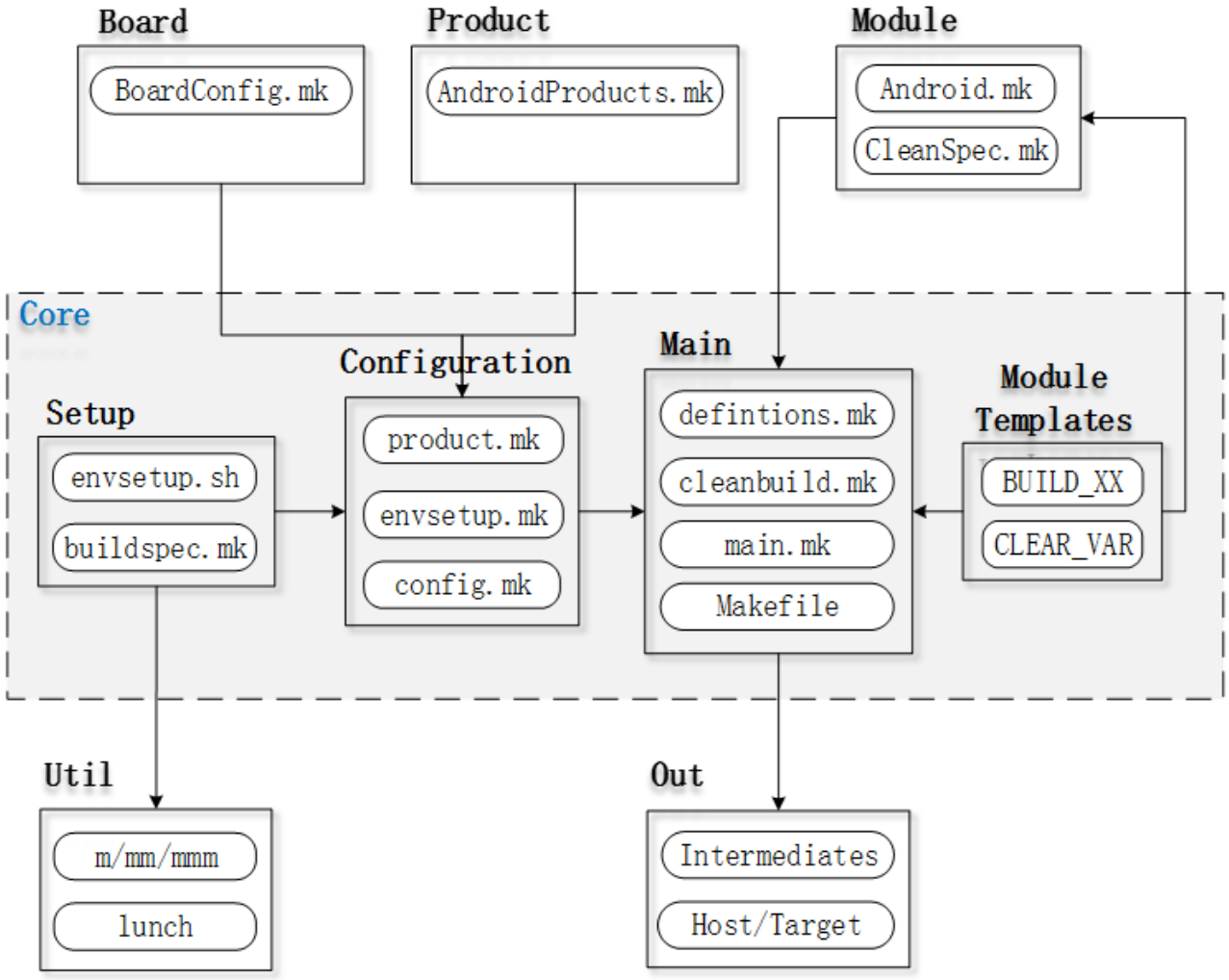
大型项目的管理 -- 非递归 make

- 将一份 Makefile 实现模块化设计
- 各模块最终汇集组合成顶层的单个 Makefile
- 根据目标依赖关系管理整个项目的各个模块

Agenda

Android Makefile Overview

- 非递归 make



构建系统的参数配置

- 芯片级 (Architecture)：CPU 体系结构和指令集的配置，如 arm, x86, mips
- 平台级 (Board)：定义在 BoardConfig.mk，内核、驱动、CPU 等与硬件紧密相关的配置
- 产品级 (Product)：定义在 AndroidProducts.mk，产品名称、需要包含哪些模块和文件等
- 模块级 (Module)：由各模块的 Android.mk 定义，模块具体配置，模块名称、模块类型、对其他模块的依赖等

构建系统的初始化

- 导入 envsetup.sh，执行 lunch 进行配置

```
$ source build/envsetup.sh   # 将envsetup.sh添加到shell执行环境中
$ lunch                      # 通过lunch来交互式的完成参数配置
```

- 配置 buildspec.mk，将文件置于 Android 源码的根目录，模板位于 build/buildspec.mk.default

添加新产品

- BoardConfig.mk：BSP 的配置，一般可以继承自现有的产品
- AndroidProducts.mk：定义变量 PRODUCT_MAKEFILES 包含新产品的 makefile 配置
- <ProductName>.mk：文件名必须与产品名称一致，新产品相关的定义，可以继承自现有产品，再定制差异部分
- verndorsetup.sh：通过 add_lunch_combo 函数在 lunch 函数中添加一个菜单选项

示例 -- CDR6012 项目

```
device/qcom/msm8952_64
|-- AndroidProducts.mk
|-- BoardConfig.mk
|-- msm8952_64.mk
|-- cdr6012_common.mk
|-- cdr6012_ak.mk
|-- cdr6012_ds.mk
|-- cdr6012_it.mk
|-- cdr6012_ta.mk
|-- cdr6012_tec.mk
`-- vendorsetup.sh
```

示例 -- CDR6012 项目

文件 AndroidProducts.mk

```
PRODUCT_MAKEFILES := \
    $(LOCAL_DIR)/msm8952_64.mk \
    $(LOCAL_DIR)/cdr6012_ak.mk \
    $(LOCAL_DIR)/cdr6012_ta.mk \
    $(LOCAL_DIR)/cdr6012_tec.mk \
    $(LOCAL_DIR)/cdr6012_it.mk \
    $(LOCAL_DIR)/cdr6013_ds.mk
```

示例 -- CDR6012 项目

文件 vendorsetup.sh

```
add_lunch_combo cdr6012_ak-userdebug
add_lunch_combo cdr6012_ta-userdebug
add_lunch_combo cdr6012_tec-userdebug
add_lunch_combo cdr6012_it-userdebug
add_lunch_combo cdr6013_ds-userdebug
```

示例 -- CDR6012 项目

文件 cdr6012_common.mk 继承自 msm8952_64.mk

```
$(call inherit-product, device/qcom/msm8952_64/msm8952_64.mk)

PRODUCT_NAME := cdr6012
PRODUCT_BRAND := Askey
PRODUCT_SUB_MODEL ?= AK
PRODUCT_MANUFACTURER := Askey
...
```

文件 cdr6012_ak.mk 继承自 cdr6012_common.mk

```
PRODUCT_BASE := msm8952_64
PRODUCT_SUB_MODEL := AK

$(call inherit-product, device/qcom/msm8952_64/cdr6012_common.mk)

PRODUCT_NAME := cdr6012_ak
PRODUCT_MODEL := CDR6012
```

示例 -- CDR6012 项目

```
$ lunch

You're building on Linux

Lunch menu... pick a combo:
     1. aosp_arm-eng
     2. aosp_arm64-eng
     ...
     32. msm8952_64-userdebug
     33. cdr6012_ak-userdebug
     34. cdr6012_ta-userdebug
     35. cdr6012_tec-userdebug
     36. cdr6012_it-userdebug
     37. cdr6013_ds-userdebug
```

常用模块配置

CLEAR_VARS	clear_vars.mk	清除 LOCAL 变量
BUILD_PACKAGE	package.mk	编译 APK
BUILD_JAVA_LIBRARY	java_library.mk	编译 jar 包
BUILD_PREBUILT	prebuilt.mk	一个编译好的资源
PREBUILT_SHARED_LIBRARY	shared_library.mk	编译共享库
PREBUILT_STATIC_LIBRARY	static_library.mk	编译静态库
BUILD_EXECUTABLE	executable.mk	编译平台可执行程序

以上定义在 build/core/config.mk

编译静态库

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := foo
LOCAL_SRC_FILES := src/foo.c
LOCAL_EXPORT_LDLIBS := -llog
include $(BUILD_STATIC_LIBRARY)
```

编译动态库

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := bar
LOCAL_SRC_FILES := src/bar.c
LOCAL_STATIC_LIBRARIES := foo
include $(BUILD_SHARED_LIBRARY)
```

安装预编译的动态库

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := foo-prebuilt
LOCAL_SRC_FILES := libfoo.so
include $(PREBUILT_SHARED_LIBRARY)
```

编译可执行程序

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := foo
LOCAL_SRC_FILES := foo.c
include $(BUILD_EXECUTABLE)
```

安装预编译的 APK

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := DvrIov
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE_CLASS := APPS
LOCAL_SRC_FILES := DvrIov.apk
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_PRIVILEGED_MODULE := true
LOCAL_CERTIFICATE := platform
include $(BUILD_PREBUILT)
```

安装预编译的 APK（需安装 JNI 库）

- LOCAL_PREBUILT_JNI_LIBS 库文件前加 @，自行从 APK 中分离出 so 库
- 在 CDR6012 项目中得不到期望的结果

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := CyberonTTS
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE_CLASS := APPS
LOCAL_SRC_FILES := CyberonTTS.apk
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_PRIVILEGED_MODULE := true
LOCAL_CERTIFICATE := platform
LOCAL_MULTILIB := 32
LOCAL_PREBUILT_JNI_LIBS := \
    @lib/armeabi/libCReader.so \
    @lib/armeabi/libLexMgr.0404.so \
    @lib/armeabi/libLexMgr.0409.so
include $(BUILD_PREBUILT)
```

安装预编译的 APK（需安装 JNI 库）

- LOCAL_PREBUILT_JNI_LIBS 库文件前无 @，安装文件系统中已存在的 so 库, $(BUILD_SYSTEM)/install_jni_libs_internal.mk 中有相关处理

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := CyberonTTS
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE_CLASS := APPS
LOCAL_SRC_FILES := CyberonTTS.apk
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_PRIVILEGED_MODULE := true
LOCAL_CERTIFICATE := platform
LOCAL_MULTILIB := 32
LOCAL_PREBUILT_JNI_LIBS := \
    lib/armeabi/libCReader.so \
    lib/armeabi/libLexMgr.0404.so \
    lib/armeabi/libLexMgr.0409.so
include $(BUILD_PREBUILT)
```

安装预编译的 APK（需安装 JNI 库）

- 为了避免维护两份 so 库文件（APK 中和本地文件系统各一份）造成版本不一致，CDR6012 项目中采取一下折中的办法

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := CyberonTTS
...
LOCAL_PREBUILT_JNI_LIBS := \
    lib/armeabi/libCReader.so \
    lib/armeabi/libLexMgr.0404.so \
    lib/armeabi/libLexMgr.0409.so
MY_SRC_WRAP := $(LOCAL_PATH)/$(LOCAL_SRC_FILES)
MY_JNI_WRAP:= $(addprefix $(LOCAL_PATH)/, $(LOCAL_PREBUILT_JNI_LIBS))

.INTERMEDIATE: $(MY_JNI_WRAP)
$(MY_JNI_WRAP): $(MY_SRC_WRAP)
    unzip -o $< lib/*/$(notdir $@) -d $(dir $<)
include $(BUILD_PREBUILT)
```

编译 APK

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

LOCAL_MODULE_TAGS := optional

LOCAL_SRC_FILES := $(call all-java-files-under, src) \
    src/com/android/music/IMediaPlaybackService.aidl

LOCAL_STATIC_JAVA_LIBRARIES = android-support-v4
LOCAL_STATIC_JAVA_LIBRARIES += android-support-v7-appcompat
LOCAL_PACKAGE_NAME := Music

LOCAL_CERTIFICATE := platform

LOCAL_PROGUARD_FLAG_FILES := proguard.flags

include $(BUILD_PACKAGE)
```

编译 java library

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES := $(call all-java-files-under,src)
LOCAL_MODULE_TAGS := optional
LOCAL_MODULE:= com.android.future.usb.accessory

include $(BUILD_JAVA_LIBRARY)
```

一些构建命令

make update-api	framework API 改动之后，更新 API
make help	帮助信息，显示主要的 make 目标
make snod	从已经编译出的包快速重建系统镜像
make libandroid_runtime	编译所有 JNI framework 内容
make framework	编译所有 Java framework 内容
make services	编译系统服务和相关内容
make dump-products	显示所有产品的编译配置信息
make bootimage	生成 boot.img
make recoveryimage	生成 recovery.img
make userdataimage	生成 userdata.img
make cacheimage	生成 cache.img

Agenda

使用 cmake

- 在 app build.gradle 中指定 CMakeLists.txt

```
externalNativeBuild {
    cmake {
        path "src/main/jni/CMakeLists.txt"
    }
}
```

编写 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.4.1)
set( CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mfpu=neon" )
include_directories(src/main/cpp/include/)
add_library( # Specifies the name of the library.
             native-lib
             # Sets the library as a shared library.
             SHARED
             # Provides a relative path to your source file(s).
             src/main/cpp/native-lib.cpp )
find_library( # Sets the name of the path variable.
              log-lib
              # Specifies the name of the NDK library that
              # you want CMake to locate.
              log )
target_link_libraries( # Specifies the target library.
                       native-lib
                       # Links the target library to the log library
                       # included in the NDK.
                       ${log-lib} )
```

编译生成多个 JNI 库

- 建立单独的目录为 JNI 库子项目
- 使用 ADD_SUBDIRECTORY 管理子项目

```
# 目录结构
app/src/main/jni
|-- CMakeLists.txt
|-- foo
|   |--CMakeLists.txt
|   `-- foo.c
`-- bar
    |--CMakeLists.txt
    `-- bar.c
```

顶层 CMakeLists.txt

```
# app/src/main/jni/CMakeLists.txt
cmake_minimum_required(VERSION 3.4.1)

ADD_SUBDIRECTORY(foo)
ADD_SUBDIRECTORY(bar)
```

Agenda

Q & A