Android 编译大全

Android 编译大全（一）
1. 编译前的准备
1.1. 编译环境准备

v 先构建一个 Ubuntu 9.10虚拟机

v 在虚拟机中确认下面的包是否已经安装：

n sudo apt-get install build-essential

n sudo apt-get install make

n sudo apt-get install gcc

n sudo apt-get install g++

n sudo apt-get install libc6-dev

n sudo apt-get install flex

n sudo apt-get install bison

n sudo apt-get install patch

n sudo apt-get install texinfo

n sudo apt-get install libncurses-dev

n sudo apt-get install git-core gnupg //(gnupg 系统可能已自带)

n sudo apt-get install flex bison gperf libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential zip curl

n sudo apt-get install ncurses-dev

n sudo apt-get install zlib1g-dev

n sudo apt-get install valgrind

n sudo apt-get install python2.5 (Ubuntu 可能已经自带)

n sudo apt-get install sun-java5-jdk （Ubuntu9.10里面已经用1.6替换了1.5，需要重新下载一个1.5）

v 下载 repo 脚本，加上可执行权。

n $ curl http://android.git.kernel.org/repo > repo

n $ sudo chmod a+x repo

注：本文档以 Froyo 版本为基础进行编译。
1.2. Source code 的获取

v 在源码存放目录 android 中执行

$ repo init -u git://codeaurora.org/platform/manifest.git -b carrot.cupcake

-m M7201JSDCBALYA6380.xml

就可以获得代码了

关于-b 和 -m 参数的说明参见：

https://www.codeaurora.org/wiki/QAEP

v 执行 repo sync 就可以开始下载源码了

$ repo sync

（漫长的过程，视速度而言需要半天以上）
Android 编译大全（二）
2. 编译源代码

v 执行 ls -la /bin/sh 命令，如果输出如下：

rwxrwxrwx 1 root root 4 2010-02-10 17:14 /bin/sh -> dash

请执行$sudo dpkg-reconfigure dash 命令修改 sh 版本，并选择“否”；

此处如果不改好的话，编译时会出现错误。

v 执行 source build/envsetup.sh 命令

v 执行 choosecombo 命令，出现选择对话框

u Build for the simulator or the device?

u 1. Device

u 2. Simulator

u

u Which would you like? [1]

u

u Build type choices are:

u 1. release

u 2. debug

u

u Which would you like? [2]

u

u Product choices are:

u 1. core

u 2. full_dream

u 3. full

u 4. full_passion

u 5. full_sapphire

u 6. generic_dream

u 7. generic

u 8. generic_passion

u 9. generic_sapphire

u 10. msm7625_qrd

u 11. msm7627_ffa

u 12. msm7627_surf

u 13. msm7630_surf

u 14. qsd8250_ffa

u 15. qsd8250_surf

u 16. sample_addon

u 17. sdk

u 18. sim

u You can also type the name of a product if you know it.

u Which product would you like? [generic] 3

u

u Variant choices are:

u 1. user

u 2. userdebug

u 3. eng

u Which would you like? [eng]

如果执行这个命令的时候，报错：/bin/sh: Syntax error: "(" unexpected

#请执行$sudo dpkg-reconfigure dash 命令，并选择“否”；

v 配置环境变量

export JAVA_HOME=/usr/lib/jvm/java-5-sun

export CLASSPATH=$JAVA_HOME/lib

export JRE_HOME=$JAVA_HOME/jre

export JAVA_PATH=$JAVA_HOME/bin:$JRE_HOME/bin

export CLASSPATH=.:$JAVA_HOME/lib:$JRE_HOME/lib:

$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar

export ANDROID_JAVA_HOME=$JAVA_HOME

export PATH=$JAVA_PATH:$PATH

v 执行 make 命令

如果安装的是 java1.6，将报错，如下：

Your version is: java version "1.6.0_15".

The correct version is: 1.5.

解决 java 编译错误，如下：

1) 下载 jdk1.5（ftp://202.112.80.252/java/jdk-1_5_0_21-linux-i586.bin）；

2) 将 jdk-1_5_0_21-linux-i586.bin 变为可执行权限

$sudo chmod a+x jdk-1_5_0_21-linux-i586.bin

3)　在命令行下执行./jdk-1_5_0_21-linux-i586.bin 安装 sdk

4)　建立一个软连接到 jdk 目录

$sudo ln jdk1.5.0_21/ java-5-sun -s

5)　然后配置环境变量：sudo gedit/etc/enviroment 在其中添加两行：

CLASSPATH=/usr/lib/jvm/java-5-sun/lib

JAVA_HOME=/usr/lib/jvm/java-5-sun

如果在出现类似使用了旧版 api 的错误，请先按照提示执行 make update-api 命令。该命令执行结束之后，再继续执行 make 命令就可以编译成功了。

在配置好 shell 命令类型之后，也可以新建一个 shell 脚本如下，进行自动编译：

```
export JAVA_HOME=/usr/lib/jvm/java-5-sun

export CLASSPATH=$JAVA_HOME/lib

export JRE_HOME=$JAVA_HOME/jre

export JAVA_PATH=$JAVA_HOME/bin:$JRE_HOME/bin

export
CLASSPATH=.:$JAVA_HOME/lib:$JRE_HOME/lib:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib
/tools.jar

export ANDROID_JAVA_HOME=$JAVA_HOME

export PATH=$JAVA_PATH:$PATH

source build/envsetup.sh

choosecombo 1 1 7 3

make
```

build 之后的 log 如下：

```
… …

creating boot.img...

creating recovery.img...

creating system.img...
```

creating userdata.img...

cleaning up...

Done.

Android 编译大全（三）

3. 验证编译之后的模块

$export ANDROID_PRODUCT_OUT=<SrcDir>/out/target/product/generic

$cd ./out/host/linux-x86/bin

$./emulator

4. 编译完成之后的代码结构

Android 编译完成后，将在根目录中生成一个 out 文件夹，所有生成的内容均放置在这个文件夹中。out 文件夹如下所示：

```
out/
|-- CaseCheck.txt
|-- casecheck.txt
|-- host
|    |-- common
|    `-- linux-x86
`-- target
     |-- common
     `-- product
```

主要的两个目录为 host 和 target，前者表示在主机（x86）生成的工具，后者表示目标机（模认为 ARMv5）运行的内容。

host 目录的结构如下所示：

```
out/host/
|-- common
|    `-- obj                （JAVA 库）
`-- linux-x86
     |-- bin                （二进制程序）
     |-- framework          （JAVA 库,*.jar 文件）
     |-- lib                （共享库*.so）
     `-- obj                （中间生成的目标文件）
```

host 目录是一些在主机上用的工具，有一些是二进制程序,有一些是 JAVA 的程序。

target 目录的结构如下所示：

```
out/target/
|-- common
|    |-- R                （资源文件）
|    |-- docs
|    `-- obj              （目标文件）
`-- product
     `-- generic
```

其中 common 目录表示通用的内容，product 中则是针对产品的内容。

在 common 目录的 obj 中，包含两个重要的目录：

APPS　中包含了 JAVA 应用程序生成的目标，每个应用程序对应其中一个子目录，将结合每个应用程序的原始文件生成 Android 应用程序的 APK 包。

JAVA_LIBRARIES　中包含了 JAVA 的库，每个库对应其中一个子目录。

在默认的情况下，Android 编译将生成 generic 目录，如果选定产品还可以生成其他的目录。generic 包含了以下内容：

```
out/target/product/generic/
|-- android-info.txt
|-- clean_steps.mk
|-- data
|-- obj
|-- ramdisk.img
|-- root
|-- symbols
|-- system
|-- system.img
|-- userdata-qemu.img
`-- userdata.img
```

在 generic/obj/APPS 目录中包含了各种 JAVA 应用，与 common/APPS 相对应，但是已经打成了 APK 包。

system 目录是主要的文件系统，data 目录是存放数据的文件系统。

obj/SHARED_LIBRARIES 中存放所有动态库。

obj/STATIC_LIBRARIES 中存放所有静态库。

几个以 img 为结尾的文件是几个目标映像文件，其中 ramdisk 是作为内存盘的根文件系统映像，system.img 是主要文件系统的映像，这是一个比较大的文件，data.img 是数据内容映像。这几个 image 文件是运行时真正需要的文件。

5. make SDK

5.1. sdk 编译

在编译完整个系统之后，再运行 make sdk，就可以进行 sdk 的编译了。make sdk 将各种工具和 image 打包，供开发和调试使用。

```
export JAVA_HOME=/usr/lib/jvm/java-5-sun

export CLASSPATH=$JAVA_HOME/lib

export JRE_HOME=$JAVA_HOME/jre

export JAVA_PATH=$JAVA_HOME/bin:$JRE_HOME/bin

export
CLASSPATH=.:$JAVA_HOME/lib:$JRE_HOME/lib:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib
/tools.jar

export ANDROID_JAVA_HOME=$JAVA_HOME

export PATH=$JAVA_PATH:$PATH



source build/envsetup.sh
```

make sdk

……

Package SDK Stubs:
out/target/common/obj/PACKAGING/android_jar_intermediates/android.jar

Package SDK: out/host/linux-x86/sdk/android-sdk_eng.huangjx_linux-x86.zip

build 之后的 log 如下：

5.2. 验证编译之后的模块

将 out/host/linux-x86/sdk/android-sdk_eng.huangjx_linux-x86.zip 解压到本地目录。进入 tools 目录，输入下面的命令创建 AVD：

$./android create avd -t 1 -c 128M -n froyo -s HVGA

Android 2.2 is a basic Android platform.

Do you wish to create a custom hardware profile [no]

Created AVD 'froyo' based on Android 2.2, with the following hardware config:

hw.lcd.density=160

输入下面的命令，启动模拟器：

$ ./emulator -avd froyo -partition-size 160 &
Android 编译大全（四）
6. 编译详细分解
6.1. build 系统简介
6.1.1.build 系统文件结构

./build

|-- CleanSpec.mk

|-- buildspec.mk.default

|-- core

| |-- Makefile

| |-- apicheck_msg_current.txt

| |-- apicheck_msg_last.txt

```
|   |-- armelf.x
|   |-- armelf.xsc
|   |-- armelflib.x
|   |-- base_rules.mk
|   |-- binary.mk
|   |-- build-system.html
|   |-- build_id.mk
|   |-- checktree
|   |-- cleanbuild.mk
|   |-- cleanspec.mk
|   |-- clear_vars.mk
|   |-- combo
|   |   |-- HOST_darwin-x86.mk
|   |   |-- HOST_linux-x86.mk
|   |   |-- HOST_windows-x86.mk
|   |   |-- TARGET_linux-arm.mk
|   |   |-- TARGET_linux-sh.mk
|   |   |-- TARGET_linux-x86.mk
|   |   |-- arch
|   |   |   `-- arm
|   |   |       |-- armv4t.mk
|   |   |       |-- armv5te-vfp.mk
|   |   |       |-- armv5te.mk
|   |   |       |-- armv7-a-neon.mk
|   |   |       `-- armv7-a.mk
|   |   |-- javac.mk
|   |   `-- select.mk
|   |-- config.mk
```

```
|   |-- copy_headers.mk
|   |-- definitions.mk
|   |-- device.mk
|   |-- distdir.mk
|   |-- droiddoc.mk
|   |-- dynamic_binary.mk
|   |-- envsetup.mk
|   |-- executable.mk
|   |-- filter_symbols.sh
|   |-- find-jdk-tools-jar.sh
|   |-- host_executable.mk
|   |-- host_java_library.mk
|   |-- host_prebuilt.mk
|   |-- host_shared_library.mk
|   |-- host_static_library.mk
|   |-- java.mk
|   |-- java_library.mk
|   |-- key_char_map.mk
|   |-- main.mk
|   |-- multi_prebuilt.mk
|   |-- node_fns.mk
|   |-- notice_files.mk
|   |-- package.mk
|   |-- pathmap.mk
|   |-- prebuilt.mk
|   |-- prelink-linux-arm-2G.map
|   |-- prelink-linux-arm.map
|   |-- process_wrapper.sh
```

```
|    |-- process_wrapper_gdb.cmds
|    |-- process_wrapper_gdb.sh
|    |-- product.mk
|    |-- product_config.mk
|    |-- proguard.flags
|    |-- proguard_tests.flags
|    |-- raw_executable.mk
|    |-- raw_static_library.mk
|    |-- root.mk
|    |-- shared_library.mk
|    |-- static_java_library.mk
|    |-- static_library.mk
|    |-- tasks
|    |    |-- apicheck.mk
|    |    |-- cts.mk
|    |    |-- product-graph.mk
|    |    `-- sdk-addon.mk
|    `-- version_defaults.mk
|-- envsetup.sh
|-- libs
|    `-- host
|         |-- Android.mk
|         |-- CopyFile.c
|         |-- include
|         |    `-- host
|         |         |-- CopyFile.h
|         |         |-- Directories.h
|         |         `-- pseudolocalize.h
```

```
|           |-- list.java
|           `-- pseudolocalize.cpp
|-- target
|    |-- board
|    |    |-- Android.mk
|    |    |-- emulator
|    |    |    |-- AndroidBoard.mk
|    |    |    |-- BoardConfig.mk
|    |    |    |-- README.txt
|    |    |    |-- tuttle2.kcm
|    |    |    `-- tuttle2.kl
|    |    |-- generic
|    |    |    |-- AndroidBoard.mk
|    |    |    |-- BoardConfig.mk
|    |    |    |-- README.txt
|    |    |    |-- system.prop
|    |    |    |-- tuttle2.kcm
|    |    |    `-- tuttle2.kl
|    |    `-- sim
|    |         |-- AndroidBoard.mk
|    |         `-- BoardConfig.mk
|    `-- product
|         |-- AndroidProducts.mk
|         |-- core.mk
|         |-- full.mk
|         |-- generic.mk
|         |-- languages_full.mk
|         |-- languages_small.mk
```

```
|       |-- sdk.mk
|       |-- security
|       |   |-- README
|       |   |-- media.pk8
|       |   |-- media.x509.pem
|       |   |-- platform.pk8
|       |   |-- platform.x509.pem
|       |   |-- shared.pk8
|       |   |-- shared.x509.pem
|       |   |-- testkey.pk8
|       |   `-- testkey.x509.pem
|       `-- sim.mk
```

Android 编译大全（五）

6.1.2.make 文件分类

² 配置类

主要用来配置 product、board，以及根据你的 Host 和 Target 选择相应的工具以及设定相应的通用编译选项：

config 文件

说明

build/core/config.mk

Config 文件的概括性配置

build/core/envsetup.mk

generate 目录构成等配置

build/target/product

产品相关的配置

build/target/board

硬件相关的配置

build/core/combo

编译选项配置

这里解释下这里的 board 和 product。board 主要是设计到硬件芯片的配置，比如是否提供硬件的某些功能，比如说 GPU 等等，或者芯片支持浮点运算等等。product 是指针对当前的芯片配置定义你将要生产产品的个性配置，主要是指 APK 方面的配置，哪些 APK 会包含在哪个 product 中，哪些 APK 在当前 product 中是不提供的。

config.mk 是一个总括性的东西，它里面定义了各种 module 编译所需要使用的 HOST 工具以及如何来编译各种模块，比如说 BUILT_PREBUILT 就定义了如何来编译预编译模块。envsetup.mk 主要会读取由 envsetup.sh 写入环境变量中的一些变量来配置编译过程中的输出目录，combo 里面主要定义了各种 Host 和 Target 结合的编译器和编译选项。

² 模块组织类

这类文件主要定义了如何来处理 Module 的 Android.mk，以及采用何种方式来生成目标模块，这些模块生成规则都定义在 config.mk 里面。我们可以看看：

CLEAR_VARS:= $(BUILD_SYSTEM)/clear_vars.mk

BUILD_HOST_STATIC_LIBRARY:=$(BUILD_SYSTEM)/host_static_library.mk

BUILD_HOST_SHARED_LIBRARY:=$(BUILD_SYSTEM)/host_shared_library.mk

BUILD_STATIC_LIBRARY:=$(BUILD_SYSTEM)/static_library.mk

BUILD_RAW_STATIC_LIBRARY :=$(BUILD_SYSTEM)/raw_static_library.mk

BUILD_SHARED_LIBRARY:=$(BUILD_SYSTEM)/shared_library.mk

BUILD_EXECUTABLE:= $(BUILD_SYSTEM)/executable.mk

BUILD_RAW_EXECUTABLE:=$(BUILD_SYSTEM)/raw_executable.mk

BUILD_HOST_EXECUTABLE:=$(BUILD_SYSTEM)/host_executable.mk

BUILD_PACKAGE:= $(BUILD_SYSTEM)/package.mk

BUILD_HOST_PREBUILT:=$(BUILD_SYSTEM)/host_prebuilt.mk

BUILD_PREBUILT:= $(BUILD_SYSTEM)/prebuilt.mk

BUILD_MULTI_PREBUILT:=$(BUILD_SYSTEM)/multi_prebuilt.mk

BUILD_JAVA_LIBRARY:= $(BUILD_SYSTEM)/java_library.mk

BUILD_STATIC_JAVA_LIBRARY:=$(BUILD_SYSTEM)/static_java_library.mk

BUILD_HOST_JAVA_LIBRARY:=$(BUILD_SYSTEM)/host_java_library.mk

BUILD_DROIDDOC:= $(BUILD_SYSTEM)/droiddoc.mk

BUILD_COPY_HEADERS := $(BUILD_SYSTEM)/copy_headers.mk

BUILD_KEY_CHAR_MAP :=$(BUILD_SYSTEM)/key_char_map.mk

除了 CLEAR_VARS 是清楚本地变量之外，其他所有的都对应了一种模块的生成规则，每一个本地模块最后都会 include 其中的一种来生成目标模块。大部分上面的.mk 都会包含 base_rules.mk，这是对模块进行处理的基础文件，建议要写本地模块的都去看看，看明白了为什么 Android.mk 要这么写就会大致明白了。

² 单个模块编译类

本地模块的 Makefile 文件就是我们在 Android 里面几乎上随处可见的 Android.mk。Android 进行编译的时候会通过下面的函数来遍历所有子目录中的 Android.mk，一旦找到就不会再往层子目录继续寻找(所有你的模块定义的顶层 Android.mk 必须包含自己定义的子目录中的 Android.mk)。

subdir_makefiles += \

$(shellbuild/tools/findleaves.sh --prune="./out" $(subdirs) Android.mk)

不同类型的本地模块具有不同的语法，但基本上是相通的，只有个别变量的不同，如何添加模块在前面的帖子已经说过了，大家可以参考。

Android 通过 LOCAL_MODULE_TAGS 来决定哪些本地模块会不会编译进系统，通过 PRODUCT 和 LOCAL_MODULE_TAGS 来决定哪些应用包会编译进系统，如果用户不指定 LOCAL_MODULE_TAGS，默认它的值是 user。此外用户可以通过 buildspec.mk 来指定你需要编译进系统的模块。用户也可以通过 mm 来编译指定模块，或者通过 make clean-module_name 来删除指定模块。

² 系统生成类

这主要指的是 build/core/Makefile 这个文件，它定义了生成各种 img 的方式，包括 ramdisk.img userdata.img system.img update.zip recover.img 等。我们可以看看这些 img 都是如何生成的，对应着我们常用的几个 make goals。

在实际的过程中，我们也可以自己编辑 out 目录下的生成文件，然后手工打包相应生成

相应的 img，最常用的是加入一些需要集成进的 prebuilt file。所有的 Makefile 都通过 build/core/main.mk 这个文件组织在一起，它定义了一个默认 goals：droid，当我们在 TOP 目录下敲 Make 实际上就等同于我们执行 make droid。当 Make include 所有的文件，完成对所有 make 文件的解析以后就会寻找生成 droid 的规则，依次生成它的依赖，直到所有满足的模块被编译好，然后使用相应的工具打包成相应的 img。

6.2. makefile 文件

控制整个 android 系统编译的 make 文件。其内容如下：

```
### DO NOT EDIT THIS FILE ###

include build/core/main.mk

### DO NOT EDIT THIS FILE ###
```

可以看出，实际上控制编译的文件是：build/core/main.mk

6.3. Make 命令

- make droid：等同于 make 命令。droid 是默认的目标名称。
- make all： make all 将 make 所有 make droid 会编译的项目。同时，将编译 LOCAL_MODULE_TAGS 定义的不包括 android tag 的模块。这将确保所有的在代码树里面同时有 Android.mk 文件的模块。
- clean-$(LOCAL_MODULE)和 clean-$(LOCAL_PACKAGE_NAME)：

删除某个模块的目标文件。例如：clean-libutils 将删除所有的 libutils.so 以及和它相关的中间文件；clean-Home 将删除 Home 应用。

- make clean：删除本次配置所编译输出的结果文件。类似于：rm –rf ./out/ <configuration>
- make clobber：删除所有配置所编译输出的结果文件。类似于：rm –rf ./out/
- make dataclean：make dataclean deletes contents of the data directory inside the current combo directory. This is especially useful on the simulator and emulator, where the persistent data remains present between builds.
- make showcommands：在编译的时候显示脚本的命令，而不是显示编译的简报。用于调试脚本。
- make LOCAL_MODULE：编译一个单独得模块（需要有 Android.mk 文件存在）。
- make targets：将输出所有拟可以编译的模块名称列表。

注：还有一些命令，从 make 文件里面应该可以找到。本文不做探讨。

6.4. build/core/config.mk

config.mk 文件的主要内容如下：

Ø 头文件的定义；（各种 include 文件夹的设定）

在定义头文件的部分，还 include 了 pathmap.mk，如下：

```
include $(BUILD_SYSTEM)/pathmap.mk
```

该文件设置 include 目录和 frameworks/base 下子目录等的信息。

Ø 编译系统内部 mk 文件的定义； <Build system internal files>

Ø 设定通用的名称；<Set common values>

Ø Include 必要的子配置文件；<Include sub-configuration files>

n buildspec.mk

n envsetup.mk

n BoardConfig.mk

n /combo/select.mk

n /combo/javac.mk

Ø 检查 BUILD_ENV_SEQUENCE_NUMBER 版本号；

In order to make easier for people when the build system changes, when it is necessary to make changes to buildspec.mk or to rerun the environment setup scripts, they contain a version number in the variable BUILD_ENV_SEQUENCE_NUMBER. If this variable does not match what the build system expects, it fails printing an error message explaining what happened. If you make a change that requires an update, you need to update two places so this message will be printed.

· In config/envsetup.make, increment the CORRECT_BUILD_ENV_SEQUENCE_NUMBER definition.

· In buildspec.mk.default, update the BUILD_ENV_SEQUENCE_DUMBER definition to match the one in config/envsetup.make

The scripts automatically get the value from the build system, so they will trigger the warning as well.

Ø 设置常用工具的常量；< Generic tools.>

Ø 设置目标选项；< Set up final options.>

Ø 遍历并设置 SDK 版本；

6.5. buildspec.mk

默认情况下，buildspec.mk 文件是不存在的，表示使用的多少默认选项。Android 只提供了 buildspec.mk 文件的模板文件 build/buildspec.mk.default。如果需要使用 buildspec.mk 文件，请将该文件拷贝到<srcDir>根目录下面，并命名为 buildspec.mk。同时，需要将模板文件里面的一些必要的配置项启用或者修改为你所需要的目标选项。

buildspec.mk 文件主要配置下面的选项：

Ø TARGET_PRODUCT：设置编译之后的目标（产品）类型；

可以设置的值在：build/target/product/中定义。比如，product 目录下有下面几个 mk 文件：

- AndroidProducts.mk
- core.mk
- full.mk
- generic.mk
- languages_full.mk
- languages_small.mk
- sdk.mk
- sim.mk

那么，在这里可以设置的值就为上面几个 mk 文件的前缀名称（generic 等）。

Ø TARGET_BUILD_VARIANT：设置 image 的类型；

包括三个选项：user、userdebug、eng。

usr： 出厂时候面向用户的 image；

userdebug： 打开了一些 debug 选项的 image；

eng： 为了开发而包含了很多工具的 image

Ø CUSTOM_MODULES：设置额外的总是会被安装到系统的模块；

这里设置的模块名称采用的是简单目标名，比如：Browser 或者 MyApp 等。这些名字在 LOCAL_MODULE 或者在 LOCAL_PACKAGE_NAME 里面定义的。

LOCAL_MODULE is the name of what's supposed to be generated from your Android.mk. For exmample, for libkjs, the LOCAL_MODULE is "libkjs" (the build system adds the appropriate suffix -- .so .dylib .dll). For app modules, use LOCAL_PACKAGE_NAME instead of LOCAL_MODULE. We're planning on switching to ant for the apps, so this might become moot.

Ø TARGET_SIMULATOR：设置是否要编译成 simulator <true or false>；

Ø TARGET_BUILD_TYPE：设置是 debug 还是 release 版本 <release or debug>；

Set this to debug or release if you care. Otherwise, it defaults to release for arm and debug for the simulator.

Ø HOST_BUILD_TYPE：设置 Host 目标是 debug 版还是 release 版；

<release or debug, default is debug>

Ø DEBUG_MODULE_ModuleName：配置单个模块的版本是 debug 还是 release；<ture or false>

Ø TARGET_TOOLS_PREFIX：工具名前缀，默认为 NULL

Ø HOST_CUSTOM_DEBUG_CFLAGS/ TARGET_CUSTOM_DEBUG_CFLAGS：增加额外的编译选项 LOCAL_CFLAGS。

LOCAL_CFLAGS：If you have additional flags to pass into the C or C++ compiler, add them here. For example: LOCAL_CFLAGS += -DLIBUTILS_NATIVE=1

Ø CUSTOM_LOCALES：增加额外的 LOCALES 到最总的 image；

Any locales that appear in CUSTOM_LOCALES but not in the locale list for the selected product will be added to the end of PRODUCT_LOCALES.

Ø OUT_DIR：编译之后文件保存路径。默认为<build-root>/out 目录；

Ø ADDITIONAL_BUILD_PROPERTIES：指定（增加）额外的属性文件；

Ø NO_FALLBACK_FONT：设置是否只支持英文（这将减少 image 的大小）。<true, false>

Ø WEBCORE_INSTRUMENTATION：webcore 支持；

Ø ENABLE_SVG：SVG 支持；

Ø BUILD_ENV_SEQUENCE_NUMBER：编译系列号；