Android.mk的组织形式

Android 以模块的形式来组织各个系统中的部件， Eng 专业点的词汇就是 Module ，就是各位在几乎每个目录下都能看到的 Android.mk 。可以简单地把 Android 所有的 Make 文件分为 4 种：

## 1 、 For config

这类文件主要来配置 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：

build/core/config.mk summary ofconfig

build/core/envsetup.mk generate dir config and so on

build/target/product productconfig

build/target/board board config

build/core/combo build flags config

这里解释下这里的 board 和 product 。 borad 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 、 Module Compile

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。

## 3 、 Local Module

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子目录继续寻找 ( 所有你的模块定义的顶层的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。

## 4 、 Package

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 我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 。

基本上 Android building system 就是以这样一种方式组织在一起的了，下面说一点闲散的东西。首先是如何来加快 Android 的编译过程，因为每次 Android 都要遍历所有的 Android.mk ，不管是编译整个工程还是只编译某个模块。所以可以将遍历的结果保存下来，下次直接从文件读就好了，但是这里容易出错，所以一定要确认是否正确包含了所有的 .mk ，当新加入文件的时候确认将原来保存的文件删除。下面是我写的加快编译的一个 makefile ，将下面的语句替换掉 main.mk 中的相应部分就可以了：

FROM ：

subdir\_makefiles += \

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

TO ：

ifneq ($(ONE\_SHOT\_MAKEFILE),)

else

ifneq ($(CASH\_MK),true)

subdir\_makefiles += \

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

else

subdir-makefiles-cash := $(shell catbuild/subdir\_mk\_cash)

ifeq ($(subdir-makefiles-cash),)

$(warning No .mk cash ,create now !)

subdir\_makefiles += \

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

mk-to-file := $(shell echo $(subdir\_makefiles) >build/subdir\_mk\_cash)

else

$(warning Using cash mk !)

subdir\_makefiles := $(shell cat build/subdir\_mk\_cash)

endif

endif

endif

通过CASH\_MK＝true来打开快速编译的功能，因为没有对错误进行检测的操作，所以使用的时候一定要特别小心。

每个应用模块都会有个Android.mk文件,在这个文件里都会定义LOCAL\_MODULE\_TAGS和LOCAL\_PACKAGE\_NAME,如果想把这个应用编译进去的话,就需要在generic.mk里面的PRODUCT这个变量后面加上这个应用的 LOCAL\_PACKAGE\_NAME,LOCAL\_MODULE\_TAGS一般是user ,debug,eng这几个选择一个举个例子:PRODUCT+=Music // LOCAL\_PACKAGE\_NAME