[Android 中makefile功能解析](http://blog.chinaunix.net/uid-20729583-id-1884674.html)

一个Android.mk file用来向编译系统描述你的源代码。具体来说：该文件是GNU Makefile的一小部分，会被编译系统解析一次或多次。你可以在每一个Android.mk file中定义一个或多个模块，你也可以在几个模块中使用同一个源代码文件。编译系统为你处理许多细节问题。例如，你不需要在你的Android.mk中列出头文件和依赖文件。NDK编译系统将会为你自动处理这些问题。这也意味着，在升级NDK后，你应该得到新的toolchain/platform支持，而且不需要改变你的Android.mk文件。  
      先看一个简单的例子：一个简单的"hello world"，比如下面的文件：  
sources/helloworld/helloworld.c   
sources/helloworld/Android.mk  
相应的Android.mk文件会象下面这样：  
---------- cut here ------------------  
LOCAL\_PATH := $(call my-dir)  
include $(CLEAR\_VARS)  
LOCAL\_MODULE  
:= helloworld  
LOCAL\_SRC\_FILES := helloworld.c  
include $(BUILD\_SHARED\_LIBRARY)  
---------- cut here ------------------  
      我们来解释一下这几行代码：  
LOCAL\_PATH := $(call my-dir)   
一个Android.mk file首先必须定义好LOCAL\_PATH变量。它用于在开发树中查找源文件。在这个例子中，宏函数’my-dir’, 由编译系统提供，用于返回当前路径（即包含Android.mk file文件的目录）。  
include $( CLEAR\_VARS)  
CLEAR\_VARS由编译系统提供，指定让GNU MAKEFILE为你清除许多LOCAL\_XXX变量（例如 LOCAL\_MODULE, LOCAL\_SRC\_FILES, LOCAL\_STATIC\_LIBRARIES, 等等...),除LOCAL\_PATH 。这是必要的，因为所有的编译控制文件都在同一个GNU MAKE执行环境中，所有的变量都是全局的。  
LOCAL\_MODULE := helloworld  
LOCAL\_MODULE变量必须定义，以标识你在Android.mk文件中描述的每个模块。名称必须是唯一的，而且不包含任何空格。注意编译系统会自动产生合适的前缀和后缀，换句话说，一个被命名为'foo'的共享库模块，将会生成'libfoo.so'文件。  
LOCAL\_SRC\_FILES := helloworld.c   
LOCAL\_SRC\_FILES变量必须包含将要编译打包进模块中的C或C++源代码文件。注意，你不用在这里列出头文件和包含文件，因为编译系统将会自动为你找出依赖型的文件；仅仅列出直接传递给编译器的源代码文件就好。

      在Android中增加本地程序或者库，这些程序和库与其所载路径没有任何关系，只和它们的Android.mk文件有关系。Android.mk和普通的Makefile有所不同，它具有统一的写法，主要包含一些系统公共的宏。  
     在一个Android.mk中可以生成多个可执行程序、动态库和静态库。  
**1，编译应用程序的模板：**     #Test Exe  
     LOCAL\_PATH := $(call my-dir)  
     #include $(CLEAR\_VARS)  
     LOCAL\_SRC\_FILES:= main.c  
     LOCAL\_MODULE:= test\_exe  
     #LOCAL\_C\_INCLUDES :=  
     #LOCAL\_STATIC\_LIBRARIES :=  
     #LOCAL\_SHARED\_LIBRARIES :=  
     include $(BUILD\_EXECUTABLE)  
（菜鸟级别解释：:=是赋值的意思，$是引用某变量的值）LOCAL\_SRC\_FILES中加入源文件路径，LOCAL\_C\_INCLUDES 中加入所需要包含的头文件路径，LOCAL\_STATIC\_LIBRARIES加入所需要链接的静态库（\*.a）的名称，LOCAL\_SHARED\_LIBRARIES中加入所需要链接的动态库（\*.so）的名称，LOCAL\_MODULE表示模块最终的名称，BUILD\_EXECUTABLE表示以一个可执行程序的方式进行编译。  
**2，编译静态库的模板：**  
     #Test Static Lib  
     LOCAL\_PATH := $(call my-dir)  
     include $(CLEAR\_VARS)  
     LOCAL\_SRC\_FILES:= \  
               helloworld.c  
     LOCAL\_MODULE:= libtest\_static  
     #LOCAL\_C\_INCLUDES :=  
     #LOCAL\_STATIC\_LIBRARIES :=  
     #LOCAL\_SHARED\_LIBRARIES :=  
     include $(BUILD\_STATIC\_LIBRARY)  
一般的和上面相似，BUILD\_STATIC\_LIBRARY表示编译一个静态库。  
**3，编译动态库的模板：**  
     #Test Shared Lib  
     LOCAL\_PATH := $(call my-dir)  
     include $(CLEAR\_VARS)  
     LOCAL\_SRC\_FILES:= \  
               helloworld.c  
     LOCAL\_MODULE:= libtest\_shared  
     TARGET\_PRELINK\_MODULES := false  
     #LOCAL\_C\_INCLUDES :=  
     #LOCAL\_STATIC\_LIBRARIES :=  
     #LOCAL\_SHARED\_LIBRARIES :=  
      include $(BUILD\_SHARED\_LIBRARY)  
一般的和上面相似，BUILD\_SHARED\_LIBRARY表示编译一个静态库。  
      以上三者的生成结果分别在如下，generic依具体target会变：  
out/target/product/generic/obj/EXECUTABLE  
out/target/product/generic/obj/STATIC\_LIBRARY  
out/target/product/generic/obj/SHARED\_LIBRARY  
      每个模块的目标文件夹分别为：  
可执行程序：XXX\_intermediates  
静态库：      XXX\_static\_intermediates  
动态库：      XXX\_shared\_intermediates  
      另外，在Android.mk文件中，还可以指定最后的目标安装路径，用LOCAL\_MODULE\_PATH和LOCAL\_UNSTRIPPED\_PATH来指定。不同的文件系统路径用以下的宏进行选择：  
TARGET\_ROOT\_OUT：表示根文件系统。  
TARGET\_OUT：表示system文件系统。  
TARGET\_OUT\_DATA：表示data文件系统。  
用法如：  
 CAL\_MODULE\_PATH:=$(TARGET\_ROOT\_OUT)