makefile(mk)文件，一旦写好，只需一个make命令，整个工程完全自动编译。

注意MK文件每一行结束之后不要有多余的空格字符，比如说换行符后面多了空字符会有“\*\* commands commence before first target”

1、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）。

2、include $( CLEAR\_VARS)，CLEAR\_VARS由编译系统提供（(可以在 android 安装目录下的/build/core/config.mk 文件看到其定义，为 CLEAR\_VARS := $(BUILD\_SYSTEM)/clear\_vars.mk)），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。

3、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。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（也可以直接以libxxx命名好）。

4、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。

5、LOCAL\_C\_INCLUDES：可选变量，表示头文件的搜索路径。默认的头文件的搜索路径是LOCAL\_PATH目录。示例：LOCAL\_C\_INCLUDES := sources/foo或LOCAL\_C\_INCLUDES := $(LOCAL\_PATH)/../foo 。

6、TARGET\_ARCH：目标 CPU平台的名字；TARGET\_PLATFORM：Android.mk 解析的时候，目标 Android 平台的名字；TARGET\_ARCH\_ABI：暂时只支持两个 value，armeabi 和 armeabi-v7a 。

7、LOCAL\_STATIC\_LIBRARIES: 表示该模块需要使用哪些静态库，以便在编译时进行链接。

8、LOCAL\_SHARED\_LIBRARIES: 表示模块在运行时要依赖的共享库（动态库），在链接时就需要，以便在生成文件时嵌入其相应的信息。

9、LOCAL\_LDLIBS: 编译模块时要使用的附加的链接器选项。

10、LOCAL\_ARM\_MODE: 默认情况下， arm目标二进制会以 thumb 的形式生成(16 位)，你可以通过设置这个变量为 arm如果你希望你的 module 是以 32 位指令的形式 。

11、LOCAL\_CFLAGS: 可选的编译器选项，在编译 C 代码文件的时候使用 。

12，include $(call all-subdir-makefiles)：返回一个位于当前 ‘my-dir’路径的子目录中的所有Android.mk的列表。

**1、编译应用程序的模板：**

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)

**注释** :=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：**

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)

**3、编译动态库的模板：**

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)

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)