<https://blog.csdn.net/wzzfeitian/article/details/40963457>

#### 本文代码托管于github  [cmake\_demo](https://github.com/carl-wang-cn/demo/tree/master/cmake" \t "_blank)

#### cmake中一些预定义变量

* PROJECT\_SOURCE\_DIR 工程的根目录
* PROJECT\_BINARY\_DIR 运行cmake命令的目录,通常是${PROJECT\_SOURCE\_DIR}/build
* CMAKE\_INCLUDE\_PATH 环境变量,非cmake变量
* CMAKE\_LIBRARY\_PATH 环境变量
* CMAKE\_CURRENT\_SOURCE\_DIR 当前处理的CMakeLists.txt所在的路径
* CMAKE\_CURRENT\_BINARY\_DIR target编译目录  
  使用ADD\_SURDIRECTORY(src bin)可以更改此变量的值  
  SET(EXECUTABLE\_OUTPUT\_PATH <新路径>)并不会对此变量有影响,只是改变了最终目标文件的存储路径
* CMAKE\_CURRENT\_LIST\_FILE 输出调用这个变量的CMakeLists.txt的完整路径
* CMAKE\_CURRENT\_LIST\_LINE 输出这个变量所在的行
* CMAKE\_MODULE\_PATH 定义自己的cmake模块所在的路径  
  SET(CMAKE\_MODULE\_PATH ${PROJECT\_SOURCE\_DIR}/cmake),然后可以用INCLUDE命令来调用自己的模块
* EXECUTABLE\_OUTPUT\_PATH 重新定义目标二进制可执行文件的存放位置
* LIBRARY\_OUTPUT\_PATH 重新定义目标链接库文件的存放位置
* PROJECT\_NAME 返回通过PROJECT指令定义的项目名称
* CMAKE\_ALLOW\_LOOSE\_LOOP\_CONSTRUCTS 用来控制IF ELSE语句的书写方式

系统信息

* CMAKE\_MAJOR\_VERSION cmake主版本号,如2.8.6中的2
* CMAKE\_MINOR\_VERSION cmake次版本号,如2.8.6中的8
* CMAKE\_PATCH\_VERSION cmake补丁等级,如2.8.6中的6
* CMAKE\_SYSTEM 系统名称,例如Linux-2.6.22
* CAMKE\_SYSTEM\_NAME 不包含版本的系统名,如Linux
* CMAKE\_SYSTEM\_VERSION 系统版本,如2.6.22
* CMAKE\_SYSTEM\_PROCESSOR 处理器名称,如i686
* UNIX 在所有的类UNIX平台为TRUE,包括OS X和cygwin
* WIN32 在所有的win32平台为TRUE,包括cygwin

开关选项

* BUILD\_SHARED\_LIBS 控制默认的库编译方式。如果未进行设置,使用ADD\_LIBRARY时又没有指定库类型,默认编译生成的库都是静态库 （可在t3中稍加修改进行验证）
* CMAKE\_C\_FLAGS 设置C编译选项
* CMAKE\_CXX\_FLAGS 设置C++编译选项

#### cmake常用命令

##### 基本语法规则：

* cmake变量使用${}方式取值,但是在IF控制语句中是直接使用变量名
* 环境变量使用$ENV{}方式取值,使用SET(ENV{VAR} VALUE)赋值
* 指令(参数1 参数2…)  
  参数使用括弧括起,参数之间使用空格或分号分开。
* 以ADD\_EXECUTABLE指令为例：
* ADD\_EXECUTABLE(hello main.c func.c)或者
* ADD\_EXECUTABLE(hello main.c;func.c)
* 指令是大小写无关的,参数和变量是大小写相关的。推荐你全部使用**大写**指令。

##### 部分常用命令列表：

* PROJECT  
  PROJECT(projectname [CXX] [C] [Java])  
  指定工程名称,并可指定工程支持的语言。支持语言列表可忽略,默认支持所有语言
* SET  
  SET(VAR [VALUE] [CACHE TYPE DOCSTRING [FORCE]])  
  定义变量(可以定义多个VALUE,如SET(SRC\_LIST main.c util.c reactor.c))
* MESSAGE  
  MESSAGE([SEND\_ERROR | STATUS | FATAL\_ERROR] “message to display” …)  
  向终端输出用户定义的信息或变量的值  
  SEND\_ERROR, 产生错误,生成过程被跳过  
  STATUS, 输出前缀为—的信息  
  FATAL\_ERROR, 立即终止所有cmake过程
* ADD\_EXECUTABLE  
  ADD\_EXECUTABLE(bin\_file\_name ${SRC\_LIST})  
  生成可执行文件
* ADD\_LIBRARY  
  ADD\_LIBRARY(libname [SHARED | STATIC | MODULE] [EXCLUDE\_FROM\_ALL] SRC\_LIST)  
  生成动态库或静态库  
  SHARED 动态库  
  STATIC 静态库  
  MODULE 在使用dyld的系统有效,若不支持dyld,等同于SHARED  
  EXCLUDE\_FROM\_ALL 表示该库不会被默认构建
* SET\_TARGET\_PROPERTIES  
  设置输出的名称,设置动态库的版本和API版本
* CMAKE\_MINIMUM\_REQUIRED  
  CMAKE\_MINIMUM\_REQUIRED(VERSION version\_number [FATAL\_ERROR])  
  声明CMake的版本要求
* ADD\_SUBDIRECTORY  
  ADD\_SUBDIRECTORY(src\_dir [binary\_dir] [EXCLUDE\_FROM\_ALL])  
  向当前工程添加存放源文件的子目录,并可以指定中间二进制和目标二进制的存放位置  
  EXCLUDE\_FROM\_ALL含义：将这个目录从编译过程中排除
* SUBDIRS  
  deprecated,不再推荐使用  
  (hello sample)相当于分别写ADD\_SUBDIRECTORY(hello),ADD\_SUBDIRECTORY(sample)
* INCLUDE\_DIRECTORIES  
  INCLUDE\_DIRECTORIES([AFTER | BEFORE] [SYSTEM] dir1 dir2 … )  
  向工程添加多个特定的头文件搜索路径,路径之间用空格分隔,如果路径包含空格,可以使用双引号将它括起来,默认的行为为追加到当前头文件搜索路径的后面。有如下两种方式可以控制搜索路径添加的位置：
  + CMAKE\_INCLUDE\_DIRECTORIES\_BEFORE,通过SET这个cmake变量为on,可以将添加的头文件搜索路径放在已有路径的前面
  + 通过AFTER或BEFORE参数,也可以控制是追加还是置前
* LINK\_DIRECTORIES  
  LINK\_DIRECTORIES(dir1 dir2 …)  
  添加非标准的共享库搜索路径
* TARGET\_LINK\_LIBRARIES  
  TARGET\_LINK\_LIBRARIES(target lib1 lib2 …)  
  为target添加需要链接的共享库
* ADD\_DEFINITIONS  
  向C/C++编译器添加-D定义  
  ADD\_DEFINITIONS(-DENABLE\_DEBUG -DABC),参数之间用空格分隔
* ADD\_DEPENDENCIES  
  ADD\_DEPENDENCIES(target-name depend-target1 depend-target2 …)  
  定义target依赖的其他target,确保target在构建之前,其依赖的target已经构建完毕
* AUX\_SOURCE\_DIRECTORY  
  AUX\_SOURCE\_DIRECTORY(dir VAR)  
  发现一个目录下所有的源代码文件并将列表存储在一个变量中  
  把当前目录下的所有源码文件名赋给变量DIR\_HELLO\_SRCS
* EXEC\_PROGRAM  
  EXEC\_PROGRAM(Executable [dir where to run] [ARGS <args>][OUTPUT\_VARIABLE <var>] [RETURN\_VALUE <value>])  
  用于在指定目录运行某个程序（默认为当前CMakeLists.txt所在目录）,通过ARGS添加参数,通过OUTPUT\_VARIABLE和RETURN\_VALUE获取输出和返回值,如下示例
* # 在src中运行ls命令,在src/CMakeLists.txt添加
* EXEC\_PROGRAM(ls ARGS "\*.c" OUTPUT\_VARIABLE LS\_OUTPUT RETURN\_VALUE LS\_RVALUE)
* IF (not LS\_RVALUE)
* MESSAGE(STATUS "ls result: " ${LS\_OUTPUT}) # 缩进仅为美观,语法无要求
* ENDIF(not LS\_RVALUE)
* INCLUDE  
  INCLUDE(file [OPTIONAL]) 用来载入CMakeLists.txt文件  
  INCLUDE(module [OPTIONAL])用来载入预定义的cmake模块  
  OPTIONAL参数的左右是文件不存在也不会产生错误  
  可以载入一个文件,也可以载入预定义模块（模块会在CMAKE\_MODULE\_PATH指定的路径进行搜索）  
  载入的内容将在处理到INCLUDE语句时直接执行
* FIND\_
  + FIND\_FILE(<VAR> name path1 path2 …)  
    VAR变量代表找到的文件全路径,包含文件名
  + FIND\_LIBRARY(<VAR> name path1 path2 …)  
    VAR变量代表找到的库全路径,包含库文件名
  + FIND\_LIBRARY(libX X11 /usr/lib)
  + IF (NOT libx)
  + MESSAGE(FATAL\_ERROR "libX not found")
  + ENDIF(NOT libX)
  + FIND\_PATH(<VAR> name path1 path2 …)  
    VAR变量代表包含这个文件的路径
  + FIND\_PROGRAM(<VAR> name path1 path2 …)  
    VAR变量代表包含这个程序的全路径
  + FIND\_PACKAGE(<name> [major.minor] [QUIET] [NO\_MODULE] [[REQUIRED | COMPONENTS] [componets …]])  
    用来调用预定义在CMAKE\_MODULE\_PATH下的Find<name>.cmake模块,你也可以自己定义Find<name>  
    模块,通过SET(CMAKE\_MODULE\_PATH dir)将其放入工程的某个目录供工程使用
* IF  
  语法：
* IF (expression)
* COMMAND1(ARGS ...)
* COMMAND2(ARGS ...)
* ...
* ELSE (expression)
* COMMAND1(ARGS ...)
* COMMAND2(ARGS ...)
* ...

ENDIF (expression) # 一定要有ENDIF与IF对应

IF (expression), expression不为：空,0,N,NO,OFF,FALSE,NOTFOUND或<var>\_NOTFOUND,为真  
IF (not exp), 与上面相反  
IF (var1 AND var2)  
IF (var1 OR var2)  
IF (COMMAND cmd) 如果cmd确实是命令并可调用,为真  
IF (EXISTS dir) IF (EXISTS file) 如果目录或文件存在,为真  
IF (file1 IS\_NEWER\_THAN file2),当file1比file2新,或file1/file2中有一个不存在时为真,文件名需使用全路径  
IF (IS\_DIRECTORY dir) 当dir是目录时,为真  
IF (DEFINED var) 如果变量被定义,为真  
IF (var MATCHES regex) 此处var可以用var名,也可以用${var}  
IF (string MATCHES regex)

当给定的变量或者字符串能够匹配正则表达式regex时为真。比如：

IF ("hello" MATCHES "ell")

MESSAGE("true")

ENDIF ("hello" MATCHES "ell")

**数字比较表达式**  
IF (variable LESS number)  
IF (string LESS number)  
IF (variable GREATER number)  
IF (string GREATER number)  
IF (variable EQUAL number)  
IF (string EQUAL number)

**按照字母表顺序进行比较**  
IF (variable STRLESS string)  
IF (string STRLESS string)  
IF (variable STRGREATER string)  
IF (string STRGREATER string)  
IF (variable STREQUAL string)  
IF (string STREQUAL string)

一个小例子,用来判断平台差异：

IF (WIN32)

MESSAGE(STATUS “This is windows.”)

ELSE (WIN32)

MESSAGE(STATUS “This is not windows”)

ENDIF (WIN32)

上述代码用来控制在不同的平台进行不同的控制,但是,阅读起来却并不是那么舒服,ELSE(WIN32)之类的语句很容易引起歧义。

可以SET(CMAKE\_ALLOW\_LOOSE\_LOOP\_CONSTRUCTS ON)

这时候就可以写成:

IF (WIN32)

ELSE ()

ENDIF ()

配合ELSEIF使用,可能的写法是这样:

IF (WIN32)

#do something related to WIN32

ELSEIF (UNIX)

#do something related to UNIX

ELSEIF(APPLE)

#do something related to APPLE

ENDIF (WIN32)

* WHILE  
  语法：
* WHILE(condition)
* COMMAND1(ARGS ...)
* COMMAND2(ARGS ...)
* ...
* ENDWHILE(condition)

其真假判断条件可以参考IF指令

* FOREACH  
  FOREACH指令的使用方法有三种形式：
  + 列表  
    语法：
  + FOREACH(loop\_var arg1 arg2 ...)
  + COMMAND1(ARGS ...)
  + COMMAND2(ARGS ...)
  + ...
  + ENDFOREACH(loop\_var)

示例：

AUX\_SOURCE\_DIRECTORY(. SRC\_LIST)

FOREACH(F ${SRC\_LIST})

MESSAGE(${F})

ENDFOREACH(F)

* + 范围  
    语法：
  + FOREACH(loop\_var RANGE total)
  + COMMAND1(ARGS ...)
  + COMMAND2(ARGS ...)
  + ...
  + ENDFOREACH(loop\_var)

示例：

从0到total以１为步进

FOREACH(VAR RANGE 10)

MESSAGE(${VAR})

ENDFOREACH(VAR)

输出：

012345678910

* + 范围和步进  
    语法：
  + FOREACH(loop\_var RANGE start stop [step])
  + COMMAND1(ARGS ...)
  + COMMAND2(ARGS ...)
  + ...
  + ENDFOREACH(loop\_var)

从start开始到stop结束,以step为步进,  
**注意：**直到遇到ENDFOREACH指令,整个语句块才会得到真正的执行。

FOREACH(A RANGE 5 15 3)

MESSAGE(${A})

ENDFOREACH(A)

输出：

581114

#### cmake中如何生成动态库和静态库

参考ADD\_LIBRARY和SET\_TARGET\_PROPERTIES用法  
t3示例

#### cmake中如何使用动态库和静态库（查找库的路径）

参考INCLUDE\_DIRECTORIES, LINK\_DIRECTORIES, TARGET\_LINK\_LIBRARIES用法  
t4示例使用动态库或静态库  
t5示例如何使用cmake预定义的cmake模块(以FindCURL.cmake为例演示)  
t6示例如何使用自定义的cmake模块(编写了自定义的FindHELLO.cmake)  
**注意**读t5和t6的CMakeLists.txt和FindHELLO.cmake中的注释部分

#### cmake中如何指定生成文件的输出路径

* 如上ADD\_SUBDIRECTORY的时候指定目标二进制文件输出路径（推荐使用下面这种）
* 使用SET命令重新定义EXECUTABLE\_OUTPUT\_PATH和LIBRARY\_OUTPUT\_PATH变量来指定最终的二进制文件的位置
* SET(EXECUTABLE\_OUTPUT\_PATH ${PROJECT\_BINARY\_DIR}/bin)
* SET(LIBRARY\_OUTPUT\_PATH ${PROJECT\_BINARY\_DIR}/lib)

上面的两条命令通常紧跟ADD\_EXECUTABLE和ADD\_LIBRARY,与其写在同一个CMakeLists.txt即可

#### cmake中如何增加编译选项

使用变量CMAKE\_C\_FLAGS添加C编译选项  
使用变量CMAKE\_CXX\_FLAGS添加C++编译选项  
使用ADD\_DEFINITION添加

#### cmake中如何增加头文件路径

参考INCLUDE\_DIRECTORIES命令用法

#### cmake中如何在屏幕上打印信息

参考MESSAGE用法

#### cmake中如何给变量赋值

参考SET和AUX\_SOURCE\_DIRECTORY用法

**建议**：在Project根目录先建立build,然后在build文件夹内运行cmake ..，这样就不会污染源代码, 如果不想要这些自动生成的文件了，只要简单的删除build文件夹就可以

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/wzzfeitian/article/details/40963457