# Cmake简介

cmake 是一个跨平台、开源的构建系统。它是一个集软件构建、测试、打包于一身的软件。它使用与平台和编译器独立的配置文件来对软件编译过程进行控制。

# 常用指令

## 注释

* 单行注释

# 内容

* 多行注释

#[[ 内容 ]]

## 打印日志

message([STATUS|WARNING|AUTHOR\_WARNING|FATAL\_ERROR|SEND\_ERROR] "message to display" ... )

消息的级别

* (无) ：重要消息
* STATUS ：非重要消息
* WARNING：CMake 警告，会继续执行
* AUTHOR\_WARNING：CMake 警告 (dev), 会继续执行
* SEND\_ERROR：CMake 错误，继续执行，但是会跳过生成的步骤
* FATAL\_ERROR：CMake 错误，终止所有处理过程

## 设置cmake最小版本

* 命令

cmake\_minimum\_required(VERSION 3.0)；

* 使用详解

可选;最好加上（CMakeLists.txt 文件中使用了一些高版本cmake特有的一些命令的时候，就需要加上这样一行，提醒用户升级到该版本之后再执行 cmake。）

## 设置cmake策略

隐式设置策略

* 命令

cmake\_policy( VERSION <min>[ ...<max> ] ) //通过版本来设置策略

* 使用详解

1. 调用cmake\_minimum\_required 会隐式调用cmake\_policy( VERSION ) , 指定版本引入的所有策略都将设置为NEW行为 ;
2. 如果没有特别的策略要指定，可以指定通过调用cmake\_mininum\_required即可，无需调用cmake\_policy ;
3. 示例

CMake最低版本为3.0 ：

cmake\_minimum\_required( VERSION 3.0 ) == cmake\_policy( VERSION 3.0 )

CMake最低版本为3.0，最高版本为3.7 ：

cmake\_minimum\_required( VERSION 3.0...3.7 ) == cmake\_policy( VERSION 3.0...3.7 )

如果需要运行 的 CMake 版本 高于 3.0，但是引入的策略为 CMake 版本 在 3.0 和 3.5 之间 ：

cmake\_minimum\_required( VERSION 3.0 )

cmake\_policy( VERSION 3.0...3.5 )

## 设置项目名称

* 命令

project( <project\_name>

[VERSION <major>[.<minor>[.<patch>[.<tweak>] ] ] ]

[DESCRIPTION <project-description-string>]

[HOMEPAGE\_URL <url-string>]

[LANGUAGES <language-name>...] )

* 使用详解

指定cmake工程的名称（VERSION ： 版本号 ， DESCRIPTION ： 项目描述 ，

HOMEPAGE\_URL : …… , LANGUAGES : 编译工程使用额语言）；

可选;最好加上（它会引入两个变量：demo\_BINARY\_DIR和demo\_SOURCE\_DIR ; 同时Cmake自动定义了两个等价的变量： PROJECT\_BINARY\_DIR 和 PROJECT\_SOURCE\_DIR ）

## 设置编译类型

* 命令

add\_executable(demo demo.cpp) # 生成可执行文件 （demo : 可执行文件名称 , 源文件名称 ）

add\_library(common STATIC util.cpp) # 生成静态库

add\_library(common SHARED util.cpp) # 生成动态库或共享库

* 使用详解

add\_library 默认生成是静态库，通过以上命令生成文件名字:

在 Linux 下是：

demo

libcommon.a

libcommon.so

在 Windows 下是：

demo.exe

common.lib

common.dll

## 开启项目测试功能

开启项目测试功能：

* 命令1

enable\_testing()

* 使用详解

一定要在根目录下的CmakeLists.txt中开启，不然执行make test时会报错；

* 命令2

add\_test( NAME <name>

[CONFIGURATIONS [Debug|Release|...]]

[WORKING\_DIRECTORY dir ]

COMMAND <command> [arg1 [arg2 ...]] )

* 使用详解

NAME 指定本测试的名称;

CONFIGURATIONS 指定不同的编译版本下是否进行测试;

WORKING\_DIRECTORY 指定工作路径

COMMAND 指定可运行程序

## 指定编译包含的源文件

明确指定包含哪些源文件

add\_library(demo demo.cpp test.cpp util.cpp)

搜索所有的cpp文件

* 命令

aux\_source\_directory( . SRC\_LIST ) # 搜索当前目录下的所有.cpp文件

add\_library(demo ${SRC\_LIST})

* 使用详解

aux\_source\_directory(dir VAR) ： 发现一个目录dir下所有的源代码文件，并将列表存储在一个变量VAR中。

自定义搜索规则

* 命令

file(GLOB SRC\_LIST "\*.cpp" "protocol/\*.cpp")

add\_library(demo ${SRC\_LIST})

# 或者

file(GLOB SRC\_LIST "\*.cpp")

file(GLOB SRC\_PROTOCOL\_LIST "protocol/\*.cpp")

add\_library(demo ${SRC\_LIST} ${SRC\_PROTOCOL\_LIST})

# 或者

aux\_source\_directory(. SRC\_LIST)

aux\_source\_directory(protocol SRC\_PROTOCOL\_LIST)

add\_library(demo ${SRC\_LIST} ${SRC\_PROTOCOL\_LIST})

* 使用详解

file(GLOB files LIST\_DIRECTORIES false \*) //收集文件指令

GLOB命令将所有匹配的文件挑选出来，默认以字典顺序排序；

## 查找指定的库文件

* 命令

find\_library( log-lib log )

* 使用详解

find\_library(var name 预编译库名) 查找到指定的预编译库，并将它的路径存储在变量中；默认的搜索路径为cmake包含的系统库，因此如果是NDK的公共库只需要指定库的name即可；

类似的还有：find\_file()/find\_path()/find\_program()/find\_package() ；

## 设置包含目录

* 命令

include\_directories

(

${CMAKE\_CURRENT\_SOURCE\_DIR}

${CMAKE\_CURRENT\_BINARY\_DIR}

${CMAKE\_CURRENT\_SOURCE\_DIR}/include

)

* 使用详解

Linux 下还可以通过如下方式设置包含的目录:

set( CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -I${CMAKE\_CURRENT\_SOURCE\_DIR}" )

## 设置链接库搜索目录

* 命令

link\_directories(${CMAKE\_CURRENT\_SOURCE\_DIR}/libs)

* 使用详解

Linux 下还可以通过如下方式设置链接库搜索的目录:

set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -L${CMAKE\_CURRENT\_SOURCE\_DIR}/libs")

## 执行Cmake命令

* 命令

cmake

* 使用详解

cmake CmakeLists.txt文件路径

# Cmake之file

读：

file( READ <filename> <out-var> […] )

file( STRINGS <filename> <out-var> […] )

file( <HASH> <filename> <out-var> )

file( TIMESTAMP <filename> <out-var> […] )

file( GET\_RUNTIME\_DEPENDENCIES […] )

写：

file({ WRITE | APPEND } <filename> <content>... )

file({ TOUCH | TOUCH\_NOCREATE } [<file>...] )

file( GENERATE OUTPUT <output-file> [...] )

file( CONFIGURE OUTPUT <output-file> CONTENT <content> [...] )

文件系统:

file({GLOB | GLOB\_RECURSE} <out-var> [...] [<globbing-expr>...])

file(MAKE\_DIRECTORY [<dir>...])

file({REMOVE | REMOVE\_RECURSE } [<files>...])

file(RENAME <oldname> <newname> [...])

file(COPY\_FILE <oldname> <newname> [...])

file({COPY | INSTALL} <file>... DESTINATION <dir> [...])

file(SIZE <filename> <out-var>)

file(READ\_SYMLINK <linkname> <out-var>)

file(CREATE\_LINK <original> <linkname> [...])

file(CHMOD <files>... <directories>... PERMISSIONS <permissions>... [...])

file(CHMOD\_RECURSE <files>... <directories>... PERMISSIONS <permissions>... [...])

路径转换:

file(REAL\_PATH <path> <out-var> [BASE\_DIRECTORY <dir>] [EXPAND\_TILDE])

file(RELATIVE\_PATH <out-var> <directory> <file>)

file({TO\_CMAKE\_PATH | TO\_NATIVE\_PATH} <path> <out-var>)

传输：

file(DOWNLOAD <url> [<file>] [...]) file(UPLOAD <file> <url> [...])

锁定：

file(LOCK <path> [...])

归档：

file( ARCHIVE\_CREATE OUTPUT <archive> PATHS <paths>... [...] )

file( ARCHIVE\_EXTRACT INPUT <archive> [...] )

## 读

### READ

file( READ <filename> <variable> [ OFFSET <offset> ] [ LIMIT <max-in> ] [ HEX ] )

* 描述

读取文件（filename）的内容保存到变量（variable）中；

* 参数

[]内的参数代表可选，从offset作为开始位置，最多读取max-in个字节的数据；

HEX：将数据转为十六进制标识（对二进制数据很有用）；

如果指定了HEX选项，输出的字母(a到f)都是小写。

* 示例

假如有config.cmake文件，内容如下：

set(TEST, "this is test")

则file的READ用法：

file( READ config.cmake contents )

string(REGEX REPLACE "set\\(([A-Za-z\_0-9]+)\\) \"([^\"]\*)\"" "\\1 \\2" contents "${contents}")

message("TEST: ${TEST}")

### STRINGS

file( STRINGS <filename> <variable> [<options>...] )

* 描述

解析<filename>中的ASCII字符串列表，并将其存储在<variable>中。忽略文件中的二进制数据。忽略回车(\r, CR)字符（遇到换行符才重新开始读行）；

* 参数

<filename> ： 文件名  
<variable> ： 变量名

<options> ：可选项如下

1. LENGTH\_MAXIMUM <max-len> ：只考虑长度不超过给定值（max-len）的字符串
2. LENGTH\_MINIMUM <min-len> ：只考虑不少于给定长度的字符串
3. LIMIT\_COUNT <max-num> ：限制要提取的不同字符串的数量
4. LIMIT\_INPUT <max-in> ：限制从文件中读取的输入字节数
5. LIMIT\_OUTPUT <max-out> ：限制（变量）中存储的总字节数。
6. NEWLINE\_CONSUME ：将换行符( \n , LF )视为字符串内容的一部分，而不是以换行符结束。
7. NO\_HEX\_CONVERSION ：除非提供此选项，否则英特尔十六进制和摩托罗拉S-record文件在读取时将自动转换为二进制。
8. REGEX <regex> ：只考虑匹配给定正则表达式的字符串
9. ENCODING <encoding-type> ：3.1版本新功能，考虑给定编码的字符串。目前支持的编码有:UTF-8、UTF-16LE、UTF-16BE、UTF-32LE、UTF-32BE。如果没有提供编码选项，并且文件有字节顺序标记，则编码选项将默认遵守字节顺序标记。

### 其他用到在补充

# Cmake之string

搜索和替换

string( FIND <string> <substring> <out-var> [...] )

string( REPLACE <match-string> <replace-string> <out-var> <input>... )

string( REGEX MATCH <match-regex> <out-var> <input>... )

string( REGEX MATCHALL <match-regex> <out-var> <input>... )

string( REGEX REPLACE <match-regex> <replace-expr> <out-var> <input>... )

操作

string( APPEND <string-var> [<input>...] )

string( PREPEND <string-var> [<input>...] )

string( CONCAT <out-var> [<input>...] )

string( JOIN <glue> <out-var> [<input>...] )

string( TOLOWER <string> <out-var> )

string( TOUPPER <string> <out-var> )

string( LENGTH <string> <out-var> )

string( SUBSTRING <string> <begin> <length> <out-var> )

string( STRIP <string> <out-var> )

string( GENEX\_STRIP <string> <out-var> )

string( REPEAT <string> <count> <out-var> )

比较

string( COMPARE <op> <string1> <string2> <out-var> )

哈希

string( <HASH> <out-var> <input> )

生成

string( ASCII <number>... <out-var> )

string( HEX <string> <out-var> )

string( CONFIGURE <string> <out-var> [...] )

string( MAKE\_C\_IDENTIFIER <string> <out-var> )

string( RANDOM [<option>...] <out-var> )

string( TIMESTAMP <out-var> [<format string>] [UTC] )

string( UUID <out-var> ... )

JSON

string( JSON <out-var> [ERROR\_VARIABLE <error-var>] {GET | TYPE | LENGTH | REMOVE} <json-string> <member|index> [<member|index> ...] )

string( JSON <out-var> [ERROR\_VARIABLE <error-var>] MEMBER <json-string> [<member|index> ...] <index> )

string( JSON <out-var> [ERROR\_VARIABLE <error-var>] SET <json-string> <member|index> [<member|index> ...] <value> )

string( JSON <out-var> [ERROR\_VARIABLE <error-var>] EQUAL <json-string1> <json-string2> )

## 查找

* 命令

string( FIND <string> <substring> <output\_variable> [REVERSE] )

* 使用详解

在<string>中查找<substring>，返回值存放于<output\_variable>,

存在：返回<string>中的下标 ;

未存在：返回 -1 ;

默认首次出现的匹配，若使用了“REVERSE”则为最后一次匹配；

注：下标从0开始，以字节为单位；因此遇到中文时，下标表示字符编码第一字节的位置；

## 替换

* 命令

string( REPLACE <match\_string> <replace\_string> <output\_variable> <input> [ <input>... ] )

* 使用详解

从所有<input> ...中查找<match\_string>并使用<replace\_string>替换，替换后的字符串存放于<output\_variable>。

多个输入时，先将所有输入连接后，再做查找替换。

## 正则表达式

### 查找

* 命令1

string( REGEX MATCH <regular\_expression> <output\_variable> <input> [ <input> ... ] )

* 使用详解

从所有<input> ...中查找<regular\_expression>匹配到的字符串，并存放于<output\_variable>，查不到输出为空字符串。多个输入时先连接再做操作。只匹配第一次。

* 命令2

string( REGEX MATCHALL <regular\_expression> <output\_variable> <input> [<input>...] )

* 使用详解

从所有<input> ...中查找<regular\_expression>匹配到的字符串，并存放于<output\_variable>，查不到输出为空字符串。多个输入时先连接再做操作。匹配所有符合的字符串，并连接起来输出。

### 替换

* 命令

string( REGEX REPLACE <regular\_expression> <replacement\_expression> <output\_variable> <input> [<input>...] )

* 使用详解

根据正则表达式查找，并替换。找不到则输出为输入字符串。

<replacement\_expression>可以使用\\1 \\2 ..,\\9来表示()元字符匹配到的字符;这里使用两个\是因为 : “\”要先在字符串中转义，然后在正则匹配中进行转义。

\1 : 子表达式1

\2 : 子表达式2

……

\9 : 子表达式9

注意：子表达式以“()”分隔。

### 元字符

|  |  |
| --- | --- |
| 元字符 | 意义 |
| **^** | 匹配字符串开头 |
| $ | 匹配字符串结尾 |
| . | 匹配任意单个字符 |
| \<char> | 转义元字符。在字符串中要使用\\<char> |
| [] | 匹配括号内任何字符 |
| [^] | 匹配任何不在括号内的字符 |
| [a-e] | 表示范围，意义同[abcde] |
| \* | 匹配0次或多次 |
| + | 匹配1次或多次 |
| ? | 匹配0次或1次 |
| | | 或运算符 |
| () | 保存匹配的子表达式 |

## 字符串操作

尾部追加

头部追加

连接

大小写转换

长度

取子串

删除收尾空白字符

删除子串中的生成器表达式

复制字符串

比较

字符串加密

转换

数字/字符互转

变量替换

\_替换

UUID生成

# Cmake之list

读：

list(LENGTH <list><output variable>)

list(GET <list> <elementindex> [<element index> ...]<output variable>)

list(APPEND <list><element> [<element> ...])

list(FIND <list> <value><output variable>)

list(INSERT <list><element\_index> <element> [<element> ...])

list(REMOVE\_ITEM <list> <value>[<value> ...])

list(REMOVE\_AT <list><index> [<index> ...])

list(REMOVE\_DUPLICATES <list>)

list(REVERSE <list>)

list(SORT <list>)

list格式如下：

list (subcommand <list> [args...])

subcommand子命令：

LENGTH ：返回list的长度；

GET ：返回list中index的element到value中；

APPEND ：添加新的element到list中；

FIND ：返回list中element的index , 没有找到返回-1 ；

INSERT ：将新element 插入到list中index的位置；

REMOVE\_ITEM ：从list中删除某个element；

REMOVE\_AT ：从list中删除指定index的element；

REMOVE\_DUPLICATES ：从list中删除重复的element；

REVERSE ：将list的内容反转；

SORT ：将list按字母顺序排序；

## 列表创建

set (list\_test a b c d) # 创建列表变量"a;b;c;d"

## 列表读取

* 命令1

list( LENGTH <list> <output variable> )

* 使用详解

读取列表长度；

* 命令2

list ( GET <list> <element\_index> [<element\_index> ...] <output variable>)

* 使用详解

读取列表中指定索引的元素，可以指定多个索引；

<element\_index>：为列表元素的索引，从0开始编号；索引也可以为负数，-1：最后一个，-2：倒数第二个，以此类推；

<output variable>：为新创建的变量，存储指定索引元素的返回结果，也是一个列表；

## 列表查找

* 命令1

list (FIND <list> <value> <output variable>)

* 使用详解

如果列表<list>中存在<value>，那么返回<value>在列表中的索引，如果未找到则返回-1

# Cmake之条件语句

if( <condition> )

<commands>

elseif( <condition> )

<commands>

else()

<commands>

endif()

if语句括号内的condition的写法，有三种：

1. if( <constant> )

cmake 定义了一些变量 ;如：

* 1 , ON , YES , TRUE , Y ,非0数 表示true ;
* 0 ，OFF , NO , FALSE , N , IGNORE , NOTFOUND , 空字符串或以-NOTFOUND结尾的字符串，表示flase ;

1. if( <variable|string> )

if语句内可以是变量或者字符串，注：if语句内变量直接引用即可,不需要${};

1. 运用cmake的一些特殊命令

**逻辑操作**

if( NOT <condition> )

如果<condition>不为真，则NOT <condition>为真 ；

if( <cond1> AND <cond2> )

一假则假；

if( <cond1> OR <cond2> )

一真则真；

**存在性检查**

if( COMMAND command-name )

若给定名称是可以调用的命令、宏或函数，则为True ;

if( POLICY policy-id )

若给定的名称是一个已存在的策略（形式为CMP< NNNN >）,则为True ;

if( TARGET target-name )

若给定名称是通过调用已调用（在任何目录中）的add\_executable() 、 add\_library() 或 add\_custom\_target() 命令创建的现有逻辑目标名称，则为True

if( TEST test-name )

若给定名称是由add\_test()命令创建的现有测试名称，则为True ;

if( DEFINED <name> | CACHE {<NAME>} | ENV{ name } )

若定义了具有给定的变量、缓存变量或环境变量，则为True。变量的值无关紧要。请注意，宏参数不是变量;

if( <variable|string> IN\_LIST < variable> )

若给定元素包含在命名列表变量中，则为True；

**文件操作**

if( EXISTS path-to-file-or-dir )

* + - * 如果指定的文件或目录存在，则为True ;
      * 只对完整路径进行了定义，不能使用相对路径；
      * 解析符号链接，即如果指定的文件或目录是符号链接，如果符号链接的目标存在，则返回为True ;

if( file1 IS\_NEWER\_THAN file2 )

* + - * + 若file1比file2新，或者两个文件中有一个不存在，则为True ;
        + 行为只对完整路径进行定义;
        + 若文件时间戳完全相同，IS\_NEWER\_THAN比较将返回true;

if( IS\_DIRECTORY path-to-directory )

* + 若给定的名称是目录，则为True ;
  + 行为仅对完整路径定义良好 ;

if( IS\_SYMLINK file-name )

* + 若给定名称是符号链接，则为True ;
  + 行为仅对完整路径定义良好 ;

if( IS\_ABSOLUTE path )

* + 若给定路径是绝对路径，则为True ;
  + 空路径，则为false ;
  + 在Windows主机上，任何以驱动器号和冒号( C: )、正斜杠或反斜杠开头的路径都为True ;(C:NO\BASE\DIR 计算为true )
  + 在非Windows主机上，任何以波浪线（~）开头的路径，则为True

**比较**

if( <variable|string> MATCHES regex )

* 若给定的字符串或变量的值与给定的正则条件匹配，则为True。
* 关于正则表达式格式，请参见正则表达式规范。

if( <variable|string> LESS <variable|string> )

* 若给定字符串或变量的值是有效的数字，且小于右边的值，则为True。

if( <variable|string> GREATER <variable|string> )

* 若给定字符串或变量的值是有效的数字，且大于右边的值，则为True。

if( <variable|string> EQUAL <variable|string> )

* 若给定字符串或变量的值是有效的数字，且等于右边的值，则为True。

if( <variable|string> LESS\_EQUAL <variable|string> )

* 若给定字符串或变量的值是有效的数字，且小于等于右边的值，则为True。

if( <variable|string> GREATER\_EQUAL <variable|string> )

* 若给定字符串或变量的值是有效的数字，且大于等于右边的值，则为True。

if( <variable|string> STRLESS <variable|string> )

* 若给定字符串或变量的值按字典顺序小于右侧的字符串或变量，则为True。

if（ <variable|string> STREQUAL <variable|string> ）

* 若给定字符串或变量的值在词典中等于右侧的字符串或变量，则为True。

if（ <variable|string> STRLESS|EQUAL <variable|string> ）

* 版本3.7中的新功能：若给定字符串或变量的值按字典顺序小于或等于右侧的字符串或变量，则为True。

if（<variable | string>STRGREATER|u EQUAL<variable | string>）

* 版本3.7中的新功能：若给定字符串或变量的值按字典顺序大于或等于右侧的字符串或变量，则为True。

# Cmake之find\_package

find\_package( <PackageName> [version] [EXACT] [QUIET] [MODULE]

   [REQUIRED] [[COMPONENTS] [components...]]

  [OPTIONAL\_COMPONENTS components...]

   [NO\_POLICY\_SCOPE] )

几个重要的参数介绍：

* **PackageName**：待查找包的名称。此外它还决定两种搜索模下的.cmake文件名称：例如模块模式下的名称为Find<PackageName>.cmake，而配置模式下为<lowercasePackageName>-config.cmake/<lowercasePackageName>-config-version.cmake。
* **MODULE**：该选项指定find\_package命令只使用模块模式搜索方式查找。未指定该选项时，find\_package会优先使用模块模式搜索，仍未找到包时，会切换成配置模式搜索。
* **version**：待查找包的版本号要求，版本号为点分格式，由四个部分组成，每个部分都是一个数字，均为可选：major[.minor[.patch[.tweak]]]，例如1.1.1.1、1.0、等。同样也可以指定版本范围（CMake 3.19及之后才支持），格式为：versionMin...[<]versionMax，versionMin和versionMax均是major[.minor[.patch[.tweak]]]形式的版本号，默认情况下会包含这个指定区间两端的版本号，但如果指定了<，那么会排除掉versionMax，例如1.1.1.1...1.1.2.0、1.1.1.1...<1.1.2.0等。
* **EXACT**：该选项要求待查找包的版本必须与指定的版本精确匹配，因此如果指定的是一个版本范围，不能使用该参数。
* **QUIET**：禁止输出信息，正常情况当找到包时，CMake会打印一些信息，指定该选项时会禁止掉这些打印。例外是当同时指定QUIET时，如果找不到包，仍然会输出错误信息并终止执行过程。
* **REQUIRED**：当未找到满足条件的包（例如版本号不匹配，或指定组件未找到等），会终止CMake的执行过程，并输出一条错误信息。如果未指定该选项，即使未找到满足条件的包，CMake的执行过程也会继续。
* **COMPONENTS**：指定要查找的组件。通常一个包可能包含多个组件（可以理解为多个库，例如把C++的std看成一个包的概念，那么vector就是std下的其中一个组件），我们的工程可能会依赖包下的具体某个组件，因此可以通过这个选项来检测这些组件是否存在。**通常的约定是，该选项后的组件应该都找到时才认为包找到，否则认为未找到满足条件的包。这个约束会依赖包的.cmake来实现**，通过find\_package命令传入的COMPONENTS可以通过<PackName>\_FIND\_COMPONENTS这个变量来获得。

## Cmake支持哪些包

* 官方预定义的包

通常包含在…/share/cmake-3.17/Modules，也可以通过CMAKE\_MODULE\_PATH来指定；

* 非官方但支持Cmake的包

这一类包虽然不在cmake的预支持列表中，但自行下载使用cmake编译安装后也能被cmake识别，使用方法与官方预定义包一致，比如glog包

* 默认不支持cmake的包

最典型的是自定义编译的库文件，这类文件既不再cmake官方列表中，安装时默认也不会被cmake搜索到，此时就需要我们自行编写相应的配置文件（.cmake文件）并放到对应的路径中去

## Cmake如何查找包

* Cmake并不是直接去查找包本身，而是通过包对应的配置文件查找包。该文件说明了包一些相关信息，比如版本，目录等。在不同模式下，配置文件可能不同，但是只要目标包有对应的.cmake文件且存在于cmake的查找路径中，就可以被find\_package直接使用。
* 搜索路径顺序：
  + - 1. 从变量 CMAKE\_MODULE\_PATH 指定的路径中进行查找；
      2. 从Cmake安装路径中查找。Cmake会在其安装路径下提供很多.cmake文件，例如：/XXX/cmake/Modules目录下。
* 两种搜索模式：

模块模式（ Module mode ）

在该模式下，Cmake会搜索一个名为Find<PackageName>.cmake的文件，其中<PackageName>为待搜索包的名称；

若找到文件Find<PackageName>.cmake，Cmake会读取并处理该文件，简而言之，它负责检查一些条件（如版本号是否满足等）是否满足，并在找到包后，返回给调用者一些变量，用以获取包的详细信息。

一般来说，Find<PackageName>.cmake文件不是随包本身一起提供的，更多的是外部针对已有包的重新包装，例如操作系统、Cmake程序、甚至是调用find\_package命令的工程针对已有的包提供针对该包的.cmake文件。

配置模式（ Config Mode ）

该模式下，Cmake会搜索<lowercasePackageName>-config.cmake文件或<PackageName>Config.cmake文件。如果find\_package命令中指定了具体的版本，也会搜索<lowercasePackageName>-config-version.cmake或<PackageName>ConfigVersion.cmake文件，因此配置模式下通常会提供配置文件和版本文件，并且作为包的一部分一起提供给使用者；

# 常用变量

|  |  |
| --- | --- |
| 变量名称 | 含义 |
| PROJECT\_SOURCE\_DIR | 工程根目录 |
| 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 |
| CMAKE\_LIBRARY\_PATH | 环境变量 |
| CMAKE\_CURRENT\_SOURCE\_DIR | 当前处理的CmakeList.txt所在路径 |
| CMAKE\_CURRENT\_BINARY\_DIR | Target编译目录   1. 使用ADD\_SURDIRECTORY(src bin) 可以更改此变量的值； 2. 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 | 输出该变量所在行 |
| CMAKE\_MODULE\_PATH | 定义自己的cmake模块所在路径   1. 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 | 通过PROJECT指令定义的项目名称 |
| CMAKE\_PROJECT\_NAME | 第一个被创建的project name |
| CMAKE\_ALLOW\_LOOSE\_LOOP\_CONSTRUCTS | 用来控制IF ELSE语句的书写方式 |

系统信息

|  |  |
| --- | --- |
| 变量名称 | 含义 |
| 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，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

|  |  |
| --- | --- |
| 变量名称 | 含义 |
| BUILD\_SHARED\_LIBS | 控制默认的库编译方式；  如果未进行设置，使用ADD\_LIBRARY时又没指定库类型，默认编译生成的库都是静态库。 |
| CMAKE\_C\_FLAGS | 设置C编译选项 |
| CMAKE\_CXX\_FLAGS | 设置C++编译选项 |