CMake 条件判断

CMake简介

- CMake 是做什么的?
  CMake是一套类似于automake的跨平台辅助项目编译的工具。 我觉得语法更加简单易用。
- CMake的工作流程
  CMake处理顶级目录的 `CMakeLists.txt` （CMake的配置文件，配置了子目录，编译目标，编译依赖等等），最后根据配置生成相应的MakeFile。

  使用make命令既可以进行编译。

CMake 基本语法

CMake定义了一套领域编程语言或者说脚本，称为CMake语言，支持变量定义、流程控制、函数、预制函数。

文件组织

CMake能够处理cmake语言源码。

在一个项目中，cmake语言源码存在的位置分为以下两种。

```
文件夹 (CMakeLists.txt),
script脚本 (<script>.cmake，后缀不重要)
```

- Directories

  在项目中，项目根目录的CMakeLists.txt是CMake的入口点，也就是说CMake命令先找CMakeLists.txt，并执行内部的命令，生成构建系统。CMakeLists.txt应该定义了所有的编译控制。
  并用add_subdirectory()指定要处理的子文件夹（子项目），子文件夹内部也要有CMakeLists.txt文件，在CMake执行到add_subdirectory()时，CMake会进入到指定的子文件夹，然后在子文件夹内部寻找CMakeLists.txt执行，生成子文件夹的构建系统。子文件夹的源码的构建工作目录就是在子文件夹内。
- Scripts
  stripts脚本如果要单独执行，需要cmake -P xxx.cmake。stripts脚本不会生成构建系统，因为在stripts脚本中，不允许指定构建目标。

command

CMake代码由一系列command的调用组成。包括if else 都属于command。
类似于下面这个命令

```
\# 添加可执行目标hello， 参数为world.c

add_executable(hello world.c)
```

command调用语法为

```
identifier(以空格隔开的参数表)
参数可以用()括起来，表示这个单个参数。
如if(TRUE OR (TRUE AND FALSE))
```

注意：command名大小写不敏感

参数类型有

方括号形式

```
[={len}[

内部随便写点文本，cmake不会内部的变量引用或者换行进行处理。可以保持文本原始样子。${variable}

\-escape

]=]
```

方括号不允许嵌套

`={len}` 的意思： `len` 表示结束符的=个数。当 `[=2]` 时，]==] 才是结束符。

例子：

```
message([=3[
This is the first line in a bracket argument with bracket length 1.
No \-escape sequences or ${variable} references are evaluated.
This is always one argument even though it contains a ; character.
The text does not end on a closing bracket of length 0 like ]].
It does end in a closing bracket of length 1.
]===])
```

引号形式

引号形式的就是放在""内部的参数，""会被当成一个参数传进函数。""内部的变量引用或者转义会被解析。可以用\表示字符串还没有结束。

```
message("This is a quoted argument containing multiple lines.
This is always one argument even though it contains a ; character.
```

```
Both \\-escape sequences and ${variable} references are evaluated.
The text does not end on an escaped double-quote like \".
It does end in an unescaped double quote.
")
```

无引号形式

CMake支持参数不带任何引号，因为所有值都会转换成String。所有的参数会被封装成List。
List的分隔符为;，所以参数列表内如果一个字符串用;分割，;两边会被当成两个参数。

```
\#这里有四个参数
commandName(arg arg2 arg3;arg4)
```

```
foreach(arg
    NoSpace
    Escaped\ Space
    This;Divides;Into;Five;Arguments
    Escaped\;Semicolon
    )
  message("${arg}")
endforeach()
```

输出

```
NoSpace
Escaped Space
This
Divides
Into
Five
Arguments
Escaped;Semicolon
```

注释

注释分为行注释和块注释

- 行注释

  # 行注释，只能写一行内容

- 块注释
  块注释有结尾有开头，可以写多行注释。用[[]]括起来，注意[要紧跟#。

  #[[这是多
  行注释]]

流程控制

条件控制

if(condition)

elseif(condition)

else()

endif()

如

```
if(VAR1 MATCHES "Hello")
    message("this is hello")
    message("this is hello2")
elseif(VAR1 MATCHES "world")
    message("this is world")
    message("this is world2")
endif()
```

循环

for循环

语法为

```
foreach(loop_var arg1 arg2 ...)
  COMMAND1(ARGS ...)
  COMMAND2(ARGS ...)
  ...
endforeach(loop_var)
```

示例

```
set(mylist "a" "b" c "d")
foreach(_var ${mylist})
     message("当前变量是: ${_var}")
endforeach()
```

上面是最简单的用法，还有一个foreach(loop_var RANGE start stop [step]) 的用法。

```
set(result 0)
foreach(_var RANGE 0 100)
     math(EXPR result "${result}+${_var}")
endforeach()
message("from 0 plus to 100 is:${result}")
```

在foreach循环中，支持break()和continue()。

while循环

```
while(condition)
  COMMAND1(ARGS ...)
  COMMAND2(ARGS ...)
  ...
endwhile(condition)
```

自定义command

CMake系统内置了一批command，https://cmake.org/cmake/help/v3.7/manual/cmake-commands.7.html
但是开发者仍然能够自定义command。

function

```
function(<name> [arg1 [arg2 [arg3 ...]]])
  COMMAND1(ARGS ...)
  COMMAND2(ARGS ...)
  ...
endfunction(<name>)
```

在function内可以使用一些变量取得传入的参数信息。

变量名	意义
ARGC	参数个数
ARGV	参数列表
ARGV0	参数0
ARGV1	参数1
ARGV2	参数2
ARGN	超出最后一个预期参数的参数列表

函数原型声明时，只接受一个参数，那么调用函数时传递给函数的参数列表中，
从第二个参数（如果有的话）开始就会保存到ARGN。

例如

```
function (argument_tester arg)
    message(STATUS "ARGN: ${ARGN}")
```

```
    message(STATUS "ARGC: ${ARGC}")
    message(STATUS "ARGV: ${ARGV}")
    message(STATUS "ARGV0: ${ARGV0}")
    message(STATUS "ARGV0: ${arg}")

    list(LENGTH ARGV  argv_len)
    message(STATUS "length of ARGV: ${argv_len}")
    set(i 0)
    while( i LESS ${argv_len})
         list(GET ARGV ${i} argv_value)
         message(STATUS "argv${i}: ${argv_value}")

         math(EXPR i "${i} + 1")
    endwhile()


endfunction ()
argument_tester(arg0 arg1 arg2 arg3)
```

macro

宏和function的作用是一样的，但是宏只是对字符串的简单替换。和define类似。

macro([arg1 [arg2 [arg3 ...]]])
COMMAND1(ARGS ...)
COMMAND2(ARGS ...)
...
endmacro()

下面简单的用一个实例区分两者的区别

```
set(var "ABC")

macro(Moo arg)
  message("arg = ${arg}")
  set(arg "abc")
  message("# After change the value of arg.")
  message("arg = ${arg}")
endmacro()
message("=== Call macro ===")
Moo(${var})

function(Foo arg)
  message("arg = ${arg}")
  set(arg "abc")
  message("# After change the value of arg.")
  message("arg = ${arg}")
```

```
endfunction()
message("=== Call function ===")
Foo(${var})
```

结果为：

```
=== Call macro ===
arg = ABC
\# After change the value of arg.
arg = ABC
=== Call function ===
arg = ABC
\# After change the value of arg.
arg = abc
```

变量定义和引用

CMake中，变量的值要么是String要么是String组成的List。

CMake没有用=赋值的操作，只有通过set,option来定义变量。
option只能定义OFF,ON的变量。

变量定义

set

set分为两种

- set普通变量

```
set(<variable> <value>... [PARENT_SCOPE])
```

例如

```
//VA=a;b, VA是一个字符串list
set(VA a b)
//VA=a,VA是一个字符串
set(VB a)
```

- set CACHE变量
  CACHE变量会自动保存到CMakeCache.txt中，上次的结果下次继续用。

```
set(<variable> <value>... CACHE <type> <docstring> [FORCE])
```

示例

```
set(ICD_LIBRARY "${PROJECT_BINARY_DIR}/lib" CACHE INTERNAL "ICD Library location"
)
```

option

```
option(<option_variable> "help string describing option"
       [initial value])
```

变量引用

可以使用${variable_name} 。如果变量没有定义，返回空. 变量引用可以嵌套，变量引用的值从内往外计算。
如

```
${outer_${inner_variable}_variable}.
```

CMake系统内置了一堆的变量，可以查阅

https://cmake.org/cmake/help/v3.7/manual/cmake-variables.7.html

环境变量的访问

$ENV{VAR}

变量只有string类型。变量名字大小写敏感，并且可以包含任意字符。
采用set()/unset()定义和取消定义

变量作用域存在于set的当前作用域

变量作用域：

- Function Scope
  在函数内部set的变量，作用域作用于当前函数及其调用的函数内。return 后就没了。
- Directory Scope
  再CMakeLists.txt定义的变量(非function内部)，作用域在当前Directory及其子Directory中。
- Persistent Cache
  持久缓存。变量值会缓存到CMakeCache.txt中，下次运行，会使用CMakeCache中的值。

采用 `set(variable value CACHE <type> "")` 方式设置。
如

```
set(ICD_LIBRARY "${PROJECT_BINARY_DIR}/lib" CACHE INTERNAL "ICD Library location"
)
```

CMake将会自动把find_path和option的值放到CMakeCache中。

Lists

在CMake中，所有的值都会被当成string来存储，但是在某些情况下， 多个string可以组成**list**。

例如在无""参数，多个字符串中间加了一个;。可以使用循环来遍历**List**

```
set(srcs a.c b.c c.c) #sets "srcs" to "a.c;b.c;c.c"
```

CMake专门提供了一个内置command来处理**list**

```
list(LENGTH <list> <output variable>) //获得list长度
list(GET <list> <element index> [<element index> ...]
     <output variable>) //获得list的某个位置元素
list(APPEND <list> [<element> ...])//add
list(FILTER <list> <INCLUDE|EXCLUDE> REGEX <regular_expression>)//清理
list(FIND <list> <value> <output variable>) //查找
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>)
```