CMakeLists

> **CmakeLists 是 cmake 编写cmake语法的配置文件，**
>
> 使用**cmake直接执行CMakeLists** 生成不同平台的编译配置文件

一、语法特性

> **CMake要求要求工程主目录和所有存放源代码子目录下都要编写CMakeLists.txt文件**

1、基本语法格式

> **指令（参数1，参数2，...）**

- 参数使用**括号**括起
- 参数直接使用**空格**和**分号**分开

2、参数|指令书写

> **指令是大小写无关的，参数和变量是大小写相关的**

```
set(HELLO hello.cpp)
add_executable(hello main.cpp hello.cpp)
ADD_EXECUTABLE(hello main.cpp ${HELLO})
```

3、变量

> **变量取值**

- 变量使用**${}**方式取值
- 在**if**语句中**直接使用变量名**
  - if(TARGET)
  - ~~if(${TARGET})~~

二、重要指令

提示

[...] >>>>>可选参数

1、指定cmake最小版本

> - **cmake_minimum_required**(VERSION 3.4.1 [提示语法])

```
#最低版本要求3.4.1
cmake_minimum_required(VERSION 3.4.1)
```

2、设置项目名称

> 定义工程名称，并且可以指定支持的语言
>
> - **project(projectname [CXX] [C] [java])**

```
#指定工程名为 HELLO
project(HELLO)
```

3、设置编译类型

> 设置编译类型
>
> - **add_executable(exename source1 source2 ...)# 生成可执行文件**
> - **add_library(libname STATIC source1 source2 ...)# 生成静态库**
> - **add_library(libname SHARED source1 source2 ...) # 生成动态库或共享库**

1. 可执行文件

```
# 编译main.cpp生成可执行文件main
add_executable(main main.cpp)
```

2. 静态库

```
# 通过变量 SRC 生成 libhello.a | lib共享库
add_library(hello STATIC ${SRC})
```

3. 动态库

```
# 通过变量 SRC 生成 libhello.so | dll 共享库
add_library(hello SHARED ${SRC})
```

4、定义变量

> 定义变量
>
> - **set(VAR [VALUE] [CACHE TYPE DOCSTRING [FORCE]])**

```
# 定义SRC变量，其值为sayhello.cpp hello.cpp
set(SRC sayhello.cpp hello.cpp)
```

4.1、set 直接设置变量的值

```
set(SRC_LIST main.cpp test.cpp)
add_executable(demo ${SRC_LIST})
```

4.2、set 追加设置变量的值

```
set(SRC_LIST main.cpp)
set(SRC_LIST ${SRC_LIST} test.cpp)
add_executable(demo ${SRC_LIST})
```

4.3、list 追加或者删除变量的值

```
set(SRC_LIST main.cpp)
list(APPEND SRC_LIST test.cpp)
list(REMOVE_ITEM SRC_LIST main.cpp)
add_executable(demo ${SRC_LIST})
```

5、工程添加头文件搜索目录

> 向工程添加多个特定的头文件搜索路径
>
> **相当于指定g++编译器的-I参数**
>
> - **include_directories([AFTER|BEFORE] [SYSTEM] dir1 dir2 ...)**

```
include_directories(
    ${CMAKE_CURRENT_SOURCE_DIR}
    ${CMAKE_CURRENT_BINARY_DIR}
    ${CMAKE_CURRENT_SOURCE_DIR}/include
)
```

6、设置库文件的搜索目录

> 向工程添加多个特定的库文件搜索路径
>
> **相当于指定g++编译器的-L参数**
>
> - **link_directories(dir1 dir2 ...)**

```
link_directories(
    ${CMAKE_CURRENT_SOURCE_DIR}/libs
)
```

7、添加编译参数

> 添加编译参数
>
> - **add_compile_options(**
>   ...)

```
# 添加编译参数 -Wall -std=c++11 -O2
add_compile_options(-Wall -std=c++11 -O2)
```

8、设置 target 需要链接的共享库

> 为 target 添加需要链接的共享库
>
> **相同于指定g++编译器-l参数**
>
> - **target_link_libraries(target library1<debug | optimized> library2...)**

```
# 将hello动态库文件链接到可执行文件main
target_link_libraries(main hello)
```

9、添加子目录

> **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 存放的位置
>
> - **add_subdirectory(source_dir [binary_dir] [EXCLUDE_FROM_ALL])**

```
# 添加src子目录，src中需有一个CMakeLists.txt
add_subdirectory(src)
```

10、指定编译包含的源文件

> **指定编译包含的源文件**
>
> - **add_library(demo source1 source2 ...)** #明确包含哪些源文件
> - **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

1. 明确指定包含哪些源文件

```
add_library(demo demo.cpp test.cpp util.cpp)
```

2. 搜索所有的 cpp 文件

```
aux_source_directory(. SRC_LIST) # 搜索当前目录下的所有.cpp文件
add_library(demo ${SRC_LIST})
```

3. 自定义搜索规则

```
file(GLOB SRC_LIST "*.cpp" "protocol/*.cpp")
add_library(demo ${SRC_LIST})
# or
file(GLOB SRC_LIST "*.cpp")
file(GLOB SRC_PROTOCOL_LIST "protocol/*.cpp")
add_library(demo ${SRC_LIST} ${SRC_PROTOCOL_LIST})
# or
file(GLOB_RECURSE SRC_LIST "*.cpp") #递归搜索
FILE(GLOB SRC_PROTOCOL RELATIVE "protocol" "*.cpp") # 相对protocol搜索
add_library(demo ${SRC_LIST} ${SRC_PROTOCOL_LIST})
# or
aux_source_directory(. SRC_LIST)
aux_source_directory(protocol SRC_PROTOCOL_LIST)
add_library(demo ${SRC_LIST} ${SRC_PROTOCOL_LIST})
```

11、查找指定的库文件

> 查找指定的库文件
>
> 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。
>
> - **find_library(VAR name path)**

```
find_library( # Sets the name of the path variable.
              log-lib

              # Specifies the name of the NDK library that
              # you want CMake to locate.
              log )
```

12、设置编译类型

> 设置编译类型 Debug | Release

```
# 设定编译类型为debug，调试时需要选择debug
set(CMAKE_BUILD_TYPE Debug)
# 设定编译类型为release，发布时需要选择release
set(CMAKE_BUILD_TYPE Release)
```

三、常用语法

> 条件控制

四、常用变量

- 预定义变量
- 环境变量
- 系统信息
- 编译选项

1、预定义变量

变量	含义
PROJECT_SOURCE_DIR	工程的根目录
PROJECT_BINARY_DIR	运行cmake的目录
PROJECT_NAME	工程名称
CMAKE_BUILD_TYPE	编译类型
CMAKE_SOURCE_DIR	工程根目录
CMAKE_CURRENT_SOURCE_DIR	当前处理的**CMakeLists**所在的路径
CMAKE_CURRENT_BINARY_DIR	**target**编译目录
CMAKE_CURRENT_LIST_DIR	**CMakeLists.txt** 的完整路径
CMAKE_CURRENT_LIST_LINE	当前所在的行
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

变量	含义
CMAKE_C_COMPILER	指定C编译器
CMAKE_CXX_COMPILER	指定C++编译器

2、环境变量

- 使用环境变量

```
$ENV{Name}
```

- 写入环境变量

```
set(ENV{Name} value) # 这里没有“$”符号
```

3、系统信息

变量	含义
CMAKE_SYSTEM	系统名称
CMAKE_SYSTEM_NAME	不包含版本的系统名称
CMAKE_SYSTEM_VERSION	系统版本
CMAKE_SYSTEM_PROCESSOR	处理器名称
UNIX	在所有的类 UNIX 平台下为TRUE，包括 OS X 和 cygwin
WIN32	在所有的 win32 平台下该值为 TRUE，包括 cygwin

4、编译选项

变量	含义
BUILD_SHARED_LIBS	控制默认（静态库）的库编译方式
CMAKE_C_FLAGS	C 编译选项
CMAKE_CXX_FLAGS	C++ 编译选项

> **BUILD_SHARED_LIBS**：这个开关用来控制默认的库编译方式，如果不进行设置，使用 add_library 又没有指定库类型的情况下，**默认编译生成的库都是静态库**。如果 **set(BUILD_SHARED_LIBS ON)** 后，**默认生成的为动态库**