CMake写一个HelloWord

cpp文件

```cpp
#include <iostream>

int main(){
    std::cout << "hello word" << std::endl;
    return 0;
}
```

Cmakelists文件

```
cmake_minimum_required(VERSION 3.12) # 指定最低版本要求
project (Hello) # 项目名
add_executable(Hello test.cpp) # 添加可执行文件
```

编译、运行

生成makefile文件

```
mkdir build // 建立build文件夹
cd build // 进入Build
make .. -G "MinGW Makefiles" //生成Makefile文件
```

编译、运行

```
make
cd ..
./Hello.exe
```

关键字

project

```
project(Hello) # 指定了工程的名字，并且支持所有语言—建议
project(Hello CXX) # 指定了工程的名字，并且支持语言是C++
```

```
project(Hello C CXX) # 指定了工程的名字，并且支持语言是C和C++
```

set

用于显示的指定变量， `${param_name}` : 取出指定的变量

```
set(SRC_LIST main.cpp) # SRC_LIST变量就包含了main.cpp
set(SRC_LIST "main main.cpp") # 源文件名有空格必须加双引号
set(SRC_LIST main.cpp t1.cpp t2.cpp) # SRC_LIST变量就包含了main.cpp t1.cpp t2.cpp
add_executable(Hello ${SRC_LIST}) # ${}表示取出SRC_LIST的内容，等价于add_executable(Hello main.cpp t1.cpp t2.cpp)
```

message

```
set(SRC_LIST main.cpp t1.cpp t2.cpp)
message(SRC_LIST) # 输出SRC_LIST包含的信息
```

cmake_install_prefix

```
默认安装路径为 /usr/local/ 或者使用 set(CMAKE_INSTALL_PREFIX /usr/local)
INSTALL(PROGRAMS runtest.bat DESTINATION bin) // 则安装在 /usr/local/bin目录下
set(CMAKE_INSTALL_PREFIX ${CMAKE_SOURCE_DIR}/build/install) // 修改安装路径
INSTALL(FILES  COPYRIGHT README DESTINATION share/doc/cmake/) // 安装在当前目录的build/install文件夹下
```

cmake_source_dir与cmake_current_source_dir与

```
cmake_source_dir // cmake根目录
cmake_current_source_dir // CMakeLists当前目录
```

add_executable

```
add_executable(test main.cpp t1.cpp) # 项目名加所有的需要编译的文件，hpp也需要编译
add_executable(test main.cpp; t1.cpp) # 可以用空格分隔，也可以用；分隔
add_executable(test main t1) # 可以不写.c或.cpp，但最好写上，容易引起冲突
```

使Hello World更像一个工程

添加src文件夹存放工程源代码

目录结构

```
.
├── build
├── CMakeLists.txt
└── src
    ├── CMakeLists.txt
    └── main.cpp
```

外层CMakelists文件

```
cmake_minimum_required(VERSION 3.12) # 指定最低版本要求
project(Hello)
add_subdirectory(src) # 添加源码子路径
```

内层CMakelists文件

```
add_executable(Hello test.cpp)
```

添加doc文件夹存放工程文档

```
.
├── build
├── CMakeLists.txt
└── src
    ├── CMakeLists.txt
    └── main.cpp
└── doc
    └── Hello.txt
```

添加bin文件夹存放中间和目标二进制文件

目录结构

```
.
```

```
├── build
 ├── bin
 ├── lib
 └── executable
├── CMakeLists.txt
└── src
 ├── CMakeLists.txt
 └── main.cpp
└── doc
 └── Hello.txt
```

外层CMakelists文件

```
cmake_minimum_required(VERSION 3.12) # 指定最低版本要求
project(Hello)
add_subdirectory(src bin) # src文件夹用于存放源码，bin文件夹用于存放二进制文件
```

更改二进制文件路径

此时修改内层CMakelists文件

```
set(src_dir test.cpp) # 显示的指定变量
message(${src_dir})
SET(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR}/executable) # 修改可执行文件路径
SET(LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib) # 修改库文件路径
add_executable(Hello ${src_dir}) # 添加可执行文件
```

在工程目录添加文本文件 COPYRIGHT, README

目录结构

```
.
├── build
 ├── bin
 ├── lib
 └── executable
├── CMakeLists.txt
└── src
 ├── CMakeLists.txt
```

```
    └── main.cpp
└── doc
    └── Hello.txt
└── COPYRIGHT
└── README
└── runtest.sh/bat
```

添加runtest.sh/bat，用来调用二进制test可执行文件

runtest.sh/bat内容

```
.\build\executable\Hello.exe # windows使用向右的斜线
../../executable/Hello # Linux使用向左的斜线
```

安装文件

使用cmake指令 `make install` ，安装的内容可以包含二进制文件、动态库、静态库、文件、目录以及脚本等，使用cmake变量 `cmake_install_prefix`

目录结构

```
.
├── build
│   ├── bin
│   ├── lib
│   └── executable
├── CMakeLists.txt
└── src
    ├── CMakeLists.txt
    └── main.cpp
└── doc
    └── Hello.txt
└── COPYRIGHT
└── README
└── runtest.sh/bat
```

安装文件

路径设置

```
默认安装路径为 /usr/local/ 或者使用 set(CMAKE_INSTALL_PREFIX /usr/local)
INSTALL(PROGRAMS runtest.bat DESTINATION bin) // 则安装在 /usr/local/bin目录下
set(CMAKE_INSTALL_PREFIX ${CMAKE_SOURCE_DIR}/build/install) // 修改安装路径
INSTALL(FILES  COPYRIGHT README DESTINATION share/doc/cmake/) // 安装在当前目录的build/install文件夹下
```

外层 CMakelists文件

```
cmake_minimum_required(VERSION 3.12)
project (Hello) # 项目名
add_subdirectory(src bin) # 添加源码子路径
// 安装文件，如COPYRIGHT README
set(CMAKE_INSTALL_PREFIX ${CMAKE_SOURCE_DIR}/install/build) // 修改安装路径
INSTALL(FILES  COPYRIGHT README DESTINATION share/doc/cmake/) // 安装文件，使用FILES
INSTALL(PROGRAMS runtest.bat DESTINATION bin) // 安装程序
INSTALL(DIRECTORY doc/ DESTINATION share/doc/cmake) // 安装目录，使用doc/表示安装目录下文件，使用doc则为安装整个目录
```

安装后目录结构

```
.
├── build
 ├── bin
 ├── lib
 ├── install
 └── executable
├── CMakeLists.txt
└── src
 ├── CMakeLists.txt
 └── main.cpp
└── doc
 └── Hello.txt
└── COPYRIGHT
└── README
└── runtest.sh/bat
```

静态库和动态库的构建

目录结构

```
.
├── build
├── CMakeLists.txt
└── lib
 ├── CMakeLists.txt
 ├── Hello.h
 └── Hello.cpp
└── doc
 └── Hello.txt
└── COPYRIGHT
└── README
└── runtest.sh/bat
```

外层CMakelists文件

```
cmake_minimum_required(VERSION 3.21)
project (Hello) # 项目名
add_subdirectory(lib bin) # 添加源码子路径
set(CMAKE_INSTALL_PREFIX ${CMAKE_SOURCE_DIR}/build/install) // 安装文件
INSTALL(FILES  COPYRIGHT README DESTINATION share/doc/cmake/)
INSTALL(PROGRAMS runtest.bat DESTINATION bin)
INSTALL(DIRECTORY doc/ DESTINATION share/doc/cmake)
```

内层CMakelists文件

```
cmake_minimum_required(VERSION 3.21)
set(src_dir Hello.cpp) # 显示的指定变量
include_directories(${CMAKE_CURRENT_SOURCE_DIR}) // 头文件路径，设置头文件路径则可以使用#include <Hello.h>,不设置则需要将.h文件与.cpp文件放在一起，且使用#include "Hello.h"
message("${CMAKE_CURRENT_SOURCE_DIR}")
SET(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR}/executable) # 修改可执行文件路径
SET(LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib) # 修改库文件路径
add_library(Hello SHARED ${src_dir})
```

生成动态库与静态库

注意应将动态库和静态库放在不同的文件夹，动态库生成时会生成.dll.a文件，将其删掉才可以调用静态库。

```
add_library(hello SHARED ${src_dir}) // 生态动态库
add_library(hello_static STATIC ${src_dir}) // 生成静态库，两个库名字相同会引起冲突
------------------------------修改库名称---------------------------------
add_library(hello SHARED ${src_dir}) # 生成动态库
SET_TARGET_PROPERTIES(hello PROPERTIES OUTPUT_NAME "hello") # 对动态库重命名
SET_TARGET_PROPERTIES(hello PROPERTIES CLEAN_DIRECT_OUTPUT 1)
add_library(hello_static STATIC ${src_dir}) # 生成静态库
SET_TARGET_PROPERTIES(hello_static PROPERTIES OUTPUT_NAME "hello") # 对静态库重命名
SET_TARGET_PROPERTIES(hello_static PROPERTIES CLEAN_DIRECT_OUTPUT 1)
```

设置版本号

```
SET_TARGET_PROPERTIES(hello PROPERTIES VERSION 1.2 SOVERSION 1) // windows下没有成功，Linux没有问题
```

安装共享库和头文件

注意在windows安装有bug，安装在同一个文件夹会只存在静态库，需分开安装。

```
INSTALL(FILES Hello.h DESTINATION include/hello) # 安装头文件
INSTALL(TARGETS hello LIBRARY DESTINATION bin) # LIBRARY指动态库
INSTALL(TARGETS hello_static ARCHIVE DESTINATION lib) # ARCHIVE指静态库
```

使用外部库

使用include和link命令

外层CMakeLlists

```
cmake_minimum_required(VERSION 3.10)
project (Hello) # 项目名
add_subdirectory(src bin) # 添加源码子路径
set(CMAKE_INSTALL_PREFIX ${CMAKE_SOURCE_DIR}/build/install)
INSTALL(FILES  COPYRIGHT README DESTINATION share/doc/cmake/)
INSTALL(PROGRAMS runtest.bat DESTINATION bin)
```

```
INSTALL(DIRECTORY doc/ DESTINATION share/doc/cmake)
```

内层CMakeLlists

Windows下使用静态库前应先将.dll.a文件删掉，使用动态编译后应将动态库移动到可执行文件目录下；Linux下正常运行。

```
cmake_minimum_required(VERSION 3.1)
project(Hello)
set(src_dir test.cpp) # 显示的指定变量
message(${src_dir})
SET(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR}/executable)
SET(LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib)
INCLUDE_DIRECTORIES(C:\\Users\\Administrator\\Desktop\\C++\\Cmake_Lib\\build\\install\\include\\hello) // 添加头文件路径
link_directories(C:\\Users\\Administrator\\Desktop\\C++\\Cmake_Lib\\build\\install\\lib) // 添加库文件路径
add_executable(Hello ${src_dir}) # 添加可执行文件
target_link_libraries(Hello hello) // 链接库
```

使用find_package

windows需要将opencv编译时的build文件夹放入环境变量。

外层CMakeLlists

```
cmake_minimum_required(VERSION 3.10)
project (Hello) # 项目名
add_subdirectory(src bin) # 添加源码子路径
set(CMAKE_INSTALL_PREFIX ${CMAKE_SOURCE_DIR}/build/install)
INSTALL(FILES  COPYRIGHT README DESTINATION share/doc/cmake/)
INSTALL(PROGRAMS runtest.bat DESTINATION bin)
INSTALL(DIRECTORY doc/ DESTINATION share/doc/cmake)
```

内层CMakeLlists

```
cmake_minimum_required(VERSION 3.1)
project(Hello)
set(src_dir test.cpp) # 显示的指定变量
message(${src_dir})
```

```
SET(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR}/executable)
SET(LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib)
INCLUDE_DIRECTORIES(C:\\Users\\Administrator\\Desktop\\C++\\Cmake_Lib\\build\\install\\include\\hello) // 添加头文件路径
link_directories(C:\\Users\\Administrator\\Desktop\\C++\\Cmake_Lib\\build\\install\\lib) // 添加库文件路径
find_package(OpenCV REQUIRED)
add_executable(Hello ${src_dir}) # 添加可执行文件
target_link_libraries(Hello hello ${OpenCV_LIBS}}) // 链接库
```

使用命令行

```
g++ -c Hello.cpp -fPIC -shared -I ./ -std=c++11 -pthread -fopenmp -o hello.o // 生成.o文件
ar crv libhello.a *.o // 生成库文件
g++ test.cpp -o test -L ./ -I ./ libhello.a // 使用库名
```

find_package