cmake

安装

cmake：https://cmake.org/download/

MinGW：Releases · niXman/mingw-builds-binaries (github.com)

构建和运行

- 新建一个构建目录

```
mkdir build
```

- 进入该目录并配置项目

```
cd build
cmake ../src
```

如果不是使用默认的Generator，应当添加`-G`选项：

```
cmake ../src -G "MinGW Makefiles"  -Bc:/Users/29848/Desktop/demo/build  #-B选项用于指定生成的构建系统文件的目录
```

- 构建

```
cmake --build . #在build目录下
```

- 运行

说明

- cmake命令不区分大小写，但是参数、变量区分大小写
- 参数用空格或分号隔开
- 使用`${VAR}`引用变量
- 引号可加可不加，但如果字符串中有空格必须加

概念

- 目标文件（`target`）：可执行文件（`add_executable`）、库文件（`add_library`）

命令

cmake_minimum_required

设置最低cmake版本。

```
cmake_minimum_required(VERSION <min>)
```

```
cmake_minimum_required(VERSION 3.10)
```

project

设置项目名。

```
project(<PROJECT-NAME> [<language-name>...])
project(<PROJECT-NAME>
        [VERSION <major>[.<minor>]]
        [LANGUAGES <language-name>...])

# 项目名会被存储在变量 PROJECT_NAME 和 CMAKE_PROJECT_NAME 中
# PROJECT_SOURCE_DIR 等价于 <PROJECT-NAME>_SOURCE_DIR
# PROJECT_BINARY_DIR 等价于 <PROJECT-NAME>_BINARY_DIR

# 如果定义了版本号
# 版本号被保存在 PROJECT_VERSION 和 <PROJECT-NAME>_VERSION 中
# 主版本号被保存在 PROJECT_VERSION_MAJOR 和 <PROJECT-NAME>_VERSION_MAJOR 中
# 次版本号被保存在 PROJECT_VERSION_MINOR 和 <PROJECT-NAME>_VERSION_MINOR 中
```

```
project(Tutorial)
project(Tutorial C CXX)
project(Tutorial VERSION 2.3 LANGUAGES CXX)
```

add_executable

用指定的源文件为项目添加可执行文件。

```
add_executable(<name> <source1...> )

# <name>即生成可执行文件的名字（与项目名没有关系），在一个项目中必须唯一
# 如windows系统会生成<name>.exe文件
```

```
add_executable(Tutorial tutorial.cxx)
```

message

打印信息。

```
message([<mode>] "message text" ...)

# STATUS 前缀为--的信息
# FATAL_ERROR 产生错误，终止运行
```

set

将变量设置为指定值。

```
set(<variable> <value>)
```

设置C++标准

```
set(CMAKE_CXX_STANDARD 11)
```

设置输出文件位置

```
# 设置运行时目标文件（exe、dll）的输出位置
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)

# 设置存档目标文件（lib、a）的输出位置
set(CMAKE_ARCHIVE_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/lib)
```

option

定义一个开关。

```
option(<variable> "<help_text>" [value])

# value的值为 ON 或 OFF ，默认为 OFF
# 命令行 -D<variable>=ON/OFF
```

configure_file

将输入文件进行替换并生成输出文件。

```
configure_file(<input> <output>)

# 输入文件中形如 @VAR@ 或 ${VAR} 的字符串会被替换为这些变量的当前值，如果未定义则被替换为空字符串
# 其他规则见下
```

```
#cmakedefine VAR ...
// 会被替换为以下两行之一，取决于VAR是否被设置位非零值
#define VAR ...
/* #undef VAR */
```

target_include_directories

指定目标的头文件路径。

```
target_include_directories(<target>
                           <INTERFACE|PUBLIC|PRIVATE> <items1...>)

# 目标文件有 INCLUDE_DIRECTORIES 和 INTERFACE_INCLUDE_DIRECTORIES 两个属性
# INCLUDE_DIRECTORIES 对内头文件目录,只有自己目标的源文件才能访问
# INTERFACE_INCLUDE_DIRECTORIES 对外头文件目录，提供给上级目录的源文件访问，自己目标的源文件不能访问
```

	INCLUDE_DIRECTORIES	INTERFACE_INCLUDE_DIRECTORIES
PRIVATE	√	

	INCLUDE_DIRECTORIES	INTERFACE_INCLUDE_DIRECTORIES
INTERFACE		√
PUBLIC	√	√

参考：cmake：target_** 中的 PUBLIC，PRIVATE，INTERFACE - 知乎 (zhihu.com)

add_subdirectory

添加源文件目录。

```
add_subdirectory(source_dir)

# 添加之后，会进入子目录，执行子目录的CMakeLists.txt文件
```

add_library

用指定的源文件生成库。

```
add_library(<name> <STATIC | SHARED | MODULE> <source...>)

# STATIC 静态库
# SHARED 动态库
# 生成的库文件名为 lib<name>.xxx
```

target_link_libraries

为目标链接库。

```
target_link_libraries(<target> <PRIVATE|PUBLIC|INTERFACE> <item...>)

# item 可以是库的目标名、库的绝对路径（必须保证文件存在），PRIVATE：只链接自己的目标，PUBLIC 还提供给其他目标链接自己链接的库
```

参考：CMake的链接选项：PRIVATE，INTERFACE，PUBLIC - 知乎 (zhihu.com)