CMake Hacking

sammietocat

2017-10-08

Contents

Foreword

Chapter 1

Why CMake

Chapter 2

Say Hello

Chapter 3

Add a Header File

For better organization, a C++ project tends to put its interfaces into separate files. And these separate files usually take form of

- **header files** with .hpp suffix, is where interfaces are declared
- **source files** with .cpp suffix, to specify the detailed implementation of interfaces in the corresponding header files

In such structure, the compiler needs information of how to find those header files required by the project. One conventional way to do so is specifying paths (either relative or absolute) by means of compilation options.

In the world of `cmake`, these options can be defined in the `CMakeLists.txt`. As usual, we're going to explain how it's done by an example. For brevity, our demo will go by adding a header-only interface to `Hello` project from previous chapter.

About header-only interfaces

For a header-only interface, the declaration and implementation of it is put together in one header file, no corresponding source file

3.1 Make up the Hello Project

The file structure of the project is organized as follows

```
|-CMakeLists.txt
|-include
| |-greeter
| | |-about_time.hpp
|-src
| |-hello.cpp
```

3.1.1 C++ source codes

The source codes of the C++ part are respectively shown by listings 3.1 and 3.2.

Listing 3.1: Codes for `include/greeter/about_time.hpp`

```
#ifndef HELLO_ABOUT_TIME_HPP
#define HELLO_ABOUT_TIME_HPP

#include <iostream>

void sayGoodMorningTo(const std::string &who) {
    std::cout << "Good morning, " << who << std::endl;
}

#endif //HELLO_ABOUT_TIME_HPP
```

Listing 3.2: Codes for `src/hello.cpp`

```
#include "greeter/about_time.hpp"

int main(int argc, char *argv[]) {
    sayGoodMorningTo("CMake");

    return 0;
}
```

As indicated, the program invoke the the `sayGoodMorningTo()` function in `include/greeter/about_time.hpp` to print a `"Good morning, CMake"` to the standard output.

3.1.2 Especially CMakeLists Script

Our `CMakeLists` file goes as listings 3.3.

Listing 3.3: Codes for `CMakeLists.txt`

```
cmake_minimum_required(VERSION 3.9.4)
project(Hello CXX)

#[[ add the "include" directory under the source tree to the search path
  so that the '#include "greeter/about_time.hpp"' directive can be
  resolved ]]
include_directories("${PROJECT_SOURCE_DIR}/include")

# specify source files needed by the executable 'hello' to build
add_executable(hello src/hello.cpp)
```

Apart form the `cmake_minimum_required`, `project` and `add_executable` commands, we introduce 3 new features here

- Comments
  - **Bracket Comment**: start with `#[[` and end with `]]`, which can span across mutiple lines
  - **Line Comment**: start with `#` and run until the end of the line
- `include_directories` command: add the given directories to paths which compilers use to search for the include files. If the argument is specified as relative paths, it will be interpreted with respect to the current source directory.
- **Variable References**
  - format: `${variable_name}`
  - A variable reference will be dereferenced as the value of variable if the value is set, and an empty string otherwise.
  - Here, the variable in use is a built-in variable `CMAKE_SOURCE_DIR` which is predefined by the `cmake`. It refers to full path to the top level of the source tree. And its counterpart is the `CMAKE_BINARY_DIR` variable assuming the path to the top level of binary tree.

3.2 Build the Project

So after horrible jargons, let's build the project to check if it's ok. Suppose we're in the source tree of the project now. Just execute following commands as listing 3.4 one by one, we will see it actually works!

Listing 3.4: Command to build the project

```
mkdir build
cd build
cmake ..
make
```

Which make a directory `build` as the binary tree and compile the project to generate the executable in it.

If nothing wrong, the output should be something similar to listing 3.5.

Listing 3.5: Successful build

```
[ 50%] Building CXX object CMakeFiles/hello.dir/src/hello.cpp.o
[100%] Linking CXX executable hello
[100%] Built target hello
```

Finally, find the generated `hello` executable, run it, and you should see "`Good morning, CMake`" in the standard output. Congratulations!

That's all for this example