# Cmake命令之add_subdirectory介绍

Domibaba
2020.09.18 06:09:00 字数 1,014 阅读 25,082

- **命令格式**

> `add_subdirectory` (source_dir [binary_dir] [ `EXCLUDE_FROM_ALL` ])
> **添加一个子目录并构建该子目录。**

- **命令解析**

  - `source_dir`

    **必选参数。**该参数指定一个子目录，子目录下应该包含 `CMakeLists.txt` 文件和代码文件。子目录可以是相对路径也可以是绝对路径，如果是相对路径，则是相对当前目录的一个相对路径。

  - `binary_dir`

    **可选参数。**该参数指定一个目录，用于存放输出文件。可以是相对路径也可以是绝对路径，如果是相对路径，则是相对当前输出目录的一个相对路径。如果该参数没有指定，则默认的输出目录使用 `source_dir` 。

  - `EXCLUDE_FROM_ALL`

    **可选参数。**当指定了该参数，则子目录下的目标不会被父目录下的目标文件包含进去，父目录的 `CMakeLists.txt` 不会构建子目录的目标文件，必须在子目录下显式去构建。 `例外情况：当父目录的目标依赖于子目录的目标，则子目录的目标仍然会被构建出来以满足依赖关系（例如使用了target_link_libraries）` 。

- **举例说明**

目录结构及说明如下：

```
├── CMakeLists.txt          #父目录的CMakeList.txt
├── main.cpp            #源文件，包含main函数
├── sub             #子目录
    └── CMakeLists.txt          #子目录的CMakeLists.txt
    └── test.h          #子目录头文件
    └── test.cpp            #子目录源文件
```

子目录 `sub` 下的 `test.cpp` 定义了一个函数 `test()` ，将输入参数打印出来，相应的头文件 `test.h` 则对 `test()` 进行声明， `CMakelists.txt` 则将 `sub` 下的源文件编译成库文件。

```cpp
//  sub/test.cpp
#include "test.h"
#include <iostream>

void test(std::string str)
{
    std::cout << str << std::endl;
}
```

```
//  sub/test.h
#include <string>

void test(std::string str);
```

```
# sub/CMakeLists.txt
cmake_minimum_required(VERSION 3.10.2)
project(sub)
add_library(sub test.cpp)
```

- **场景1**：父目录`CMakeLists.txt`的`add_subdirectory`只指定了`source_dir`。

```
# 父目录下的CMakeLists.txt
cmake_minimum_required(VERSION 3.10.2)
project(test)

add_subdirectory(sub)
```

在父目录下调用`cmake .`构建之后，在`sub`目录下会出现`libsub.a`库，说明当不指定`binary_dir`，输出目标文件就会放到`source_dir`目录下。

- **场景2**：父目录`CMakeLists.txt`的`add_subdirectory`指定了`source_dir`和`binary_dir`。

```
# 父目录下的CMakeLists.txt
cmake_minimum_required(VERSION 3.10.2)
project(test)

add_subdirectory(sub output)
```

在父目录下调用`cmake .`构建之后，在`output`目录下会出现`libsub.a`库，`sub`目录下则没有`libsub.a`。说明当指定`binary_dir`，输出目标文件就会放到`binary_dir`目录下。

- **场景3**：父目录`CMakeLists.txt`的`add_subdirectory`指定了`EXCLUDE_FROM_ALL`选项。

```
# 父目录下的CMakeLists.txt
cmake_minimum_required(VERSION 3.10.2)
project(test)

add_subdirectory(sub output EXCLUDE_FROM_ALL)
add_executable(test main.cpp)
```

在父目录下调用`cmake .`构建之后，在`output`目录或`sub`目录下`不会`出现`libsub.a`库，说明当指定`EXCLUDE_FROM_ALL`选项，子目录的目标文件不会生成。

- **场景4**：父目录`CMakeLists.txt`的`add_subdirectory`指定了`EXCLUDE_FROM_ALL`选项，且父目录的目标文件依赖子目录的目标文件。

```
# 父目录下的CMakeLists.txt
cmake_minimum_required(VERSION 3.10.2)
project(test)

add_subdirectory(sub output EXCLUDE_FROM_ALL)
add_executable(test main.cpp)
target_link_libraries(test sub)
```

在父目录下调用`cmake .`构建之后，在`output`目录`会`出现`libsub.a`库，说明即使指定`EXCLUDE_FROM_ALL`选项，当父目录目标文件对子目录目标文件存在依赖关系时，子目录的目标

文件仍然会生成以满足依赖关系。

最后，以一个完整的例子来结束本文（`sub`目录下的`CMakeList.txt`、`test.h`、`test.cpp`等文件内容如上文所示，没有变化），父目录下的`main.cpp`和`CMakeList.txt`如下：

```
# 父目录下的CMakeLists.txt
cmake_minimum_required(VERSION 3.10.2)
project(test)

include_directories(sub)
add_subdirectory(sub output)

add_executable(test main.cpp)
target_link_libraries(test sub)
```

```
# 父目录下的main.cpp
#include "test.h"
#include <iostream>

int main(int argc, char** argv)
{
    std::cout << "In main..." << std::endl;
    test("hello, world!");
    return 0;
}
```

```
# 输出
> cmake --build .
Scanning dependencies of target sub
[ 25%] Building CXX object output/CMakeFiles/sub.dir/test.cpp.o
[ 50%] Linking CXX static library libsub.a
[ 50%] Built target sub
Scanning dependencies of target test
[ 75%] Building CXX object CMakeFiles/test.dir/main.cpp.o
[100%] Linking CXX executable test
[100%] Built target test
>./test
In main...
hello, world!
```

## 附录：参考资料


1. https://cmake.org/cmake/help/latest/command/add_subdirectory.html