chapter1.7 cmake引入第三方库

1. 文件构成：

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

2. 文件填充：

2.1 main.cpp

```
#include <iostream>
#include <boost/shared_ptr.hpp>
#include <boost/filesystem.hpp>
/*
（1）Boost库是为C++语言标准库提供扩展的一些C++程序库的总称，由Boost社区组织开发、
维护。

（2）Boost库可以与C++标准库完美共同工作，并且为其提供扩展功能。
*/

int main(int argc, char *argv[])
{
    std::cout << "Hello Third Party Include!" << std::endl;

    // use a shared ptr
    boost::shared_ptr<int> isp(new int(4));

    // trivial use of boost filesystem
    boost::filesystem::path path = "/usr/share/cmake/modules";
    if(path.is_relative())
    {
        std::cout << "Path is relative" << std::endl;
    }
    else
    {
        std::cout << "Path is not relative" << std::endl;
    }

   return 0;
}
```

2.2 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
# Set the project name
project (third_party_include)

# find a boost install with the libraries filesystem and system
#  (1) 使用库文件系统和系统查找boost install
find_package(Boost 1.46.1 REQUIRED COMPONENTS filesystem system)

# 这是第三方库，而不是自己生成的静态动态库
#  (2) 检查是否能找到这个库文件（check if boost was found）
```

```
if(Boost_FOUND)
    message ("boost found")
else()
    message (FATAL_ERROR "Cannot find Boost")
endif()

# （3）Add an executable 制作main.cpp成可执行文件，called：third_party_include
add_executable(third_party_include main.cpp)

# （4）link against the boost libraries 链接：库函数Boost::filesystem 和 可执行文件third_party_include
target_link_libraries( third_party_include
    PRIVATE
        Boost::filesystem
)
```

3. 文件解析：

几乎所有不平凡的项目都将要求包含第三方库，头文件或程序。
CMake支持使用find_package（）函数查找这些工具的路径。
这将从CMAKE_MODULE_PATH中的文件夹列表中搜索格式为“ FindXXX.cmake”的CMake模块。
在linux上，默认搜索路径将是/ usr / share / cmake / Modules。 在我的系统上，这包括对大约142个通用第三方库的支持。

此示例要求将Boost库安装在默认系统位置。

3.1 Finding a Package

如上所述，find_package（）函数将从CMAKE_MODULE_PATH中的文件夹列表中搜索“ FindXXX.cmake”中的CMake模块。

find_package参数的确切格式取决于要查找的模块：

- 这通常记录在FindXXX.cmake文件的顶部

```
find_package(Boost 1.46.1 REQUIRED COMPONENTS filesystem system)
```

- 参数：

> [1] Boost-库名称。 这是用于查找模块文件FindBoost.cmake的一部分
> [2] 1.46.1 - 需要的boost库最低版本
> [3] REQUIRED - 告诉模块这是必需的，如果找不到会报错
> [4] COMPONENTS - 要查找的库列表。从后面的参数代表的库里找boost
> [5] 可以使用更多参数，也可以使用其他变量。 在后面的示例中提供了更复杂的设置。

3.2 Checking if the package is found

大多数被包含的包将设置变量XXX_FOUND，该变量可用于检查软件包在系统上是否可用。

在此示例中，变量为Boost_FOUND：

```
if(Boost_FOUND)
    message ("boost found")
    include_directories(${Boost_INCLUDE_DIRS})
else()
    message (FATAL_ERROR "Cannot find Boost")
endif()
```

3.3 Exported Variables

找到包后，它会自动导出变量，这些变量可以通知用户在哪里可以找到库，头文件或可执行文件。 与XXX_FOUND变量类似，它们与包绑定在一起，通常记录在FindXXX.cmake文件的顶部。

本例中的变量：

```
Boost_INCLUDE_DIRS – boost头文件的路径
```

在某些情况下，您还可以通过使用ccmake或cmake-gui检查缓存来检查这些变量。

3.4 Alias / Imported targets别名/导入目标

大多数modern CMake库在其模块文件中导出别名目标。
导入目标的好处是它们也可以填充包含目录和链接的库。

例如，从CMake v3.5开始，Boost模块支持此功能。 与使用自己的别名目标相似，模块中的别名可以使引用找到的目标变得更加容易。 对于Boost，所有目标均使用Boost ::标识符，然后使用子系统名称导出。

例如，您可以使用：

- `Boost::boost` for header only libraries
- `Boost::system` for the boost system library.
- `Boost::filesystem` for filesystem library. 与您自己的目标一样，这些目标包括它们的依赖关系，因此与Boost :: filesystem链接将自动添加Boost :: boost和Boost :: system依赖关系。 要链接到导入的目标，可以使用以下命令：

```
target_link_libraries( third_party_include
      PRIVATE
          Boost::filesystem
  )
```

> 将Boost::filesystem制作成库函数，该库called：third_party_include

3.5 Non-alias targets

尽管大多数现代库都使用导入的目标，但并非所有模块都已更新。 如果未更新库，则通常会发现以下可用变量：

- xxx_INCLUDE_DIRS - 指向库的包含目录的变量。
- xxx_LIBRARY - 指向库路径的变量。
  然后可以将它们添加到您的target_include_directories和target_link_libraries中，如下所示：

```
# Include the boost headers
target_include_directories( third_party_include
    PRIVATE ${Boost_INCLUDE_DIRS}
)

# link against the boost libraries
target_link_libraries( third_party_include
    PRIVATE
    ${Boost_SYSTEM_LIBRARY}
    ${Boost_FILESYSTEM_LIBRARY}
)
```

4. 总览：

略，参数太多，见原网站