# chapter2.1 cmake子工程使用

## 1. 文件结构：

```unix
.
├── CMakeLists.txt
├── subbinary
│   ├── CMakeLists.txt
│   └── main.cpp
├── subbinary1
│   ├── CMakeLists.txt
│   ├── include
│   │   └── sublib1
│   │       └── sublib1.h
│   └── src
│       └── sublib1.cpp
└── subbinary2
    ├── CMakeLists.txt
    └── include
        └── sublib2
            └── sublib2.h

```

## 2. 文件内容填充：

### 2.1 添加子目录

CMakeLists.txt文件可以包含和调用包含CMakeLists.txt文件的子目录。

```
add_subdirectory(sublibrary1)
add_subdirectory(sublibrary2)
add_subdirectory(subbinary)
```

### 2.2 引用子项目目录

使用project（）命令创建项目时，CMake将自动创建许多变量，这些变量可用于引用有关该项目的详细信息。 这些变量然后可以由其他子项目或主项目使用。 例如，要引用您可以使用的其他项目的源目录。

```
    ${sublibrary1_SOURCE_DIR}
    ${sublibrary2_SOURCE_DIR}
```

ps: CMake中有一些变量会自动创建:

| Variable | Info |
| --- | --- |
| PROJECT_NAME | 当前project（）设置的项目的名称。 |
| CMAKE_PROJECT_NAME | 由project（）命令设置的第一个项目的名称，即顶层项目。 |
| PROJECT_SOURCE_DIR | 当前项目的源文件目录。 |
| PROJECT_BINARY_DIR | 当前项目的构建目录。 |
| name_SOURCE_DIR | 在此示例中，创建的源目录为 `sublibrary1_SOURCE_DIR`，`sublibrary2_SOURCE_DIR`，and `subbinary_SOURCE_DIR` |
| name_BINARY_DIR | 本工程的二进制目录是 `sublibrary1_BINARY_DIR`，`sublibrary2_BINARY_DIR`，和 `subbinary_BINARY_DIR` |

### 2.3 Header only Libraries

如果您有一个库被创建为仅头文件的库，则cmake支持INTERFACE目标，以允许创建没有任何构建输出的目标:

```
add_library(${PROJECT_NAME} INTERFACE)
```

创建目标时，您还可以使用INTERFACE范围包含该目标的目录。 INTERFACE范围用于制定在链接此目标的任何库中使用的目标需求，但在目标本身的编译中不使用。

```
target_include_directories(${PROJECT_NAME}
    INTERFACE
        ${PROJECT_SOURCE_DIR}/include
)
```

### 2.4 引用子项目中的库

如果子项目创建了一个库，则其他项目可以通过在target_link_libraries（）命令中调用该项目的名称来引用该库。 这意味着您不必引用新库的完整路径，而是将其添加为依赖项。

```
target_link_libraries(subbinary
    PUBLIC
        sublibrary1
)
```

或者，您可以创建一个别名目标，该目标允许您在上下文（其实就是某个目标的绰号）中引用该目标。

```
add_library(sublibrary2)
add_library(sub::lib2 ALIAS sublibrary2)
```

To reference the alias, just it as follows:

```
target_link_libraries(subbinary
    sub::lib2
)
```

### 2.5 包含子项目中的目录

从cmake v3开始从子项目添加库时，无需将项目include目录添加到二进制文件的include目录中。

创建库时，这由target_include_directories（）命令中的作用域控制。 在此示例中，因为子二进制可执行文件链接了sublibrary1和sublibrary2库，所以当它们与库的PUBLIC和INTERFACE范围一起导出时，它将自动包含$ {sublibrary1_SOURCE_DIR} / inc和$ {sublibrary2_SOURCE_DIR} / inc文件夹。（这个地方设及到了PUBLIC和INTERFACE的使用）

## 3. 文件解析：

```
.
├── CMakeLists.txt--------------- called cmakelist_top
├── subbinary
│   ├── CMakeLists.txt----------- called cmakelist_0
│   └── main.cpp
├── sublibrary1
│   ├── CMakeLists.txt----------- called cmakelist_1
│   ├── include
│   │   └── sublib1
```

```
│   │       └── sublib1.h
│   └── src
│       └── sublib1.cpp
└── sublibrary2
    ├── CMakeLists.txt----------- called cmakelist_2
    └── include
        └── sublib2
            └── sublib2.h
```

**cmakelist_top**

```
cmake_minimum_required (VERSION 3.5)

project(subprojects)           创建总项目名称叫做subprojects

# Add sub directories
add_subdirectory(sublibrary1)  添加子目录1——sublibrary1
add_subdirectory(sublibrary2)  添加子目录2——sublibrary2
add_subdirectory(subbinary)    添加子目录3——subbinary
```

**cmakelist_0 ( in subbinary )**

```
 (1) 创建子项目3——subbinary
project(subbinary)

# Create the executable
 (2) 通过 程序main.cpp 建立 可执行文件${PROJECT_NAME}_此指subbinary
add_executable(${PROJECT_NAME} main.cpp)

# Link the static library from subproject1 using it's alias sub::lib1
# Link the header only library from subproject2 using it's alias sub::lib2
# This will cause the include directories for that target to be added to this project
#使用别名sub :: lib1从subproject1链接静态库
#使用别名sub :: lib2从subproject2链接仅标头的库
#这将导致该目标的包含目录添加到该项目中

 (3) 通过链接库sub::lib1&sub::lib2 为 可执行文件${PROJECT_NAME} 引入 头文件sub::lib1&sub::lib2
target_link_libraries(${PROJECT_NAME}
    sub::lib1
    sub::lib2
)
```

**cmakelist_1 ( in sublibrary1 )**

```
 (1) 创建子项目3——sublibrary1
# Set the project name
project (sublibrary1)

 (2) 通过 程序sublib1.cpp 建立 可执行文件${PROJECT_NAME}_此指sublibrary1
# Add a library with the above sources
#此处${PROJECT_NAME}是当前project的名字, sublibrary1
add_library(${PROJECT_NAME} src/sublib1.cpp)

 (2.5) 将可执行文件${PROJECT_NAME}改名: from sublibrary1 to sub::lib1
add_library(sub::lib1 ALIAS ${PROJECT_NAME})

 (3) 通过链接库sublib1.h 为 可执行文件${PROJECT_NAME} 引入 头文件sublib1.h
target_include_directories( ${PROJECT_NAME}
```

```
    PUBLIC ${PROJECT_SOURCE_DIR}/include
)
```

**cmakelist_2 ( in sublibrary2 )**

```
（1）创建子项目3——sublibrary1
# Set the project name
project (sublibrary2)

（2）通过 程序sublib2.cpp 建立 可执行文件${PROJECT_NAME}_此指sublibrary2
add_library(${PROJECT_NAME} INTERFACE)

// 对比：上面cmakelist_1 为什么这里是INTERFACE而不是 .../sublibrary2
// review: 2.3 如果您有一个库被创建为仅头文件的库，则cmake支持INTERFACE目标，以允许创建没有任何构建输出的目标

（2.5）将可执行文件${PROJECT_NAME}改名：from sublibrary2 to sub::lib2
add_library(sub::lib2 ALIAS ${PROJECT_NAME})

（3）通过链接库sublib2.h 为 可执行文件${PROJECT_NAME} 引入 头文件sublib2.h
target_include_directories(${PROJECT_NAME}
    INTERFACE
        ${PROJECT_SOURCE_DIR}/include
)
```

## 4. 总览：

```
huluobo@huluobodeMacBook-Pro ▶ ~/cmake-examples/myCmake ▶ ᚠ main ± ▶ cd chapter2.1
 huluobo@huluobodeMacBook-Pro ▶ ~/cmake-examples/myCmake/chapter2.1 ▶ ᚠ main ± ▶ cd build
 huluobo@huluobodeMacBook-Pro ▶ ~/cmake-examples/myCmake/chapter2.1/build ▶ ᚠ main ± ▶ cmake ..
-- The C compiler identification is AppleClang 15.0.0.15000040
-- The CXX compiler identification is AppleClang 15.0.0.15000040
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /Library/Developer/CommandLineTools/usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /Library/Developer/CommandLineTools/usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.3s)
-- Generating done (0.0s)
-- Build files have been written to: /Users/huluobo/cmake-examples/myCmake/chapter2.1/build
 huluobo@huluobodeMacBook-Pro ▶ ~/cmake-examples/myCmake/chapter2.1/build ▶ ᚠ main ± ▶ make
[ 25%] Building CXX object sublibrary1/CMakeFiles/sublibrary1.dir/src/sublib1.cpp.o
[ 50%] Linking CXX static library libsublibrary1.a
[ 50%] Built target sublibrary1
[ 75%] Building CXX object subbinary/CMakeFiles/subbinary.dir/main.cpp.o
[100%] Linking CXX executable subbinary
[100%] Built target subbinary
 huluobo@huluobodeMacBook-Pro ▶ ~/cmake-examples/myCmake/chapter2.1/build ▶ ᚠ main ± ▶
```