

第六章 CMake install部署项目

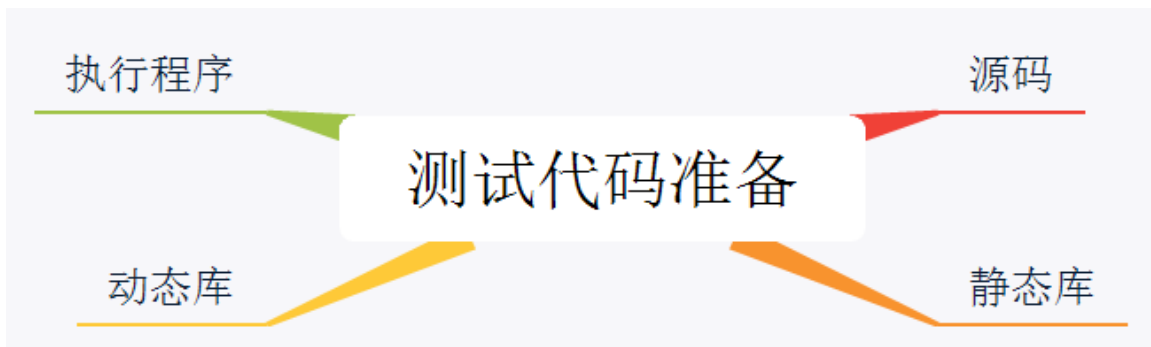
第六章 CMake install部署项目	1
1. 测试代码准备	4
1.1. 源码	4
1.1.1. include/slib.h	5
1.1.2. include/slib_pri.h	5
1.1.3. src/slib.cpp	5
1.1.4. src/dlib.cpp	6
1.1.5. main.cpp	6
1.2. 静态库	7
1.2.1. add_library(slib STATIC src/slib.cpp)	7
1.2.2. set_target_properties(slib PROPERTIES PUBLIC_HEADER include/slib.h) set_target_properties(slib PROPERTIES PRIVATE_HEADER include/slib_pri.h)	8
1.3. 动态库	8
1.3.1. add_library(dlib SHARED src/dlib.cpp)	8
1.4. 执行程序	8
1.4.1. add_executable(\${PROJECT_NAME} main.cpp)	8
2. 安装命令	8
2.1. 指定安装路径	8
2.1.1. cmake build -D CMAKE_INSTALL_PREFIX=./	9
2.2. cmake --install build	9
3. 安装目标	9
3.1. 语法	9
3.1.1. install(TARGETS targets... [EXPORT <export-name>] [RUNTIME_DEPENDENCIES args... RUNTIME_DEPENDENCY_SET <set-name>] [[ARCHIVE LIBRARY RUNTIME OBJECTS FRAMEWORK BUNDLE PRIVATE_HEADER PUBLIC_HEADER RESOURCE FILE_SET <set-name>] [DESTINATION <dir>] [PERMISSIONS permissions...] [CONFIGURATIONS [Debug Release ...]] [COMPONENT <component>] [NAMELINK_COMPONENT <component>] [OPTIONAL] [EXCLUDE_FROM_ALL] [NAMELINK_ONLY NAMELINK_SKIP]] [...] [INCLUDES DESTINATION [<dir> ...]])	9
3.2. DESTINATION 安装路径	10
3.2.1. 指定安装的目录,可以是相对路径或绝对路径	10
3.2.2. 相对路径则这相对于CMAKE_INSTALL_PREFIX	10
3.3. PERMISSIONS 权限	10
3.3.1. 指定文件权限 OWNER_READ, OWNER_WRITE, OWNER_EXECUTE, GROUP_READ, GROUP_WRITE, GROUP_EXECUTE, WORLD_READ, WORLD_WRITE, WORLD_EXECUTE, SETUID, 和SETGID. 在某些平台上无意义的权限会被忽略。 .11	

3.4.	CONFIGURATIONS (Debug Release)	11
3.4.1.	指定安装规则适用的构建配置列表 (Debug, Release)	11
3.4.2.	install(TARGETS target CONFIGURATIONS Debug RUNTIME DESTINATION Debug/bin) install(TARGETS target CONFIGURATIONS Release RUNTIME DESTINATION Release/bin)	11
3.4.3.	需要设置在RUNTIME DESTINATION之前	11
3.5.	OPTIONAL	11
3.5.1.	可选的, 如果目标不存在, 不失败	11
3.6.	目标分类	11
3.6.1.	RUNTIME	12
3.6.2.	ARCHIVE	12
3.6.3.	LIBRARY	13
3.6.4.	PUBLIC_HEADER、PRIVATE_HEADER	15
3.6.5.	代码演示	15
4.	cmake install 安装文件	15
4.1.	语法	16
4.1.1.	install(<FILES PROGRAMS> files... TYPE <type> DESTINATION <dir> [PERMISSIONS permissions...] [CONFIGURATIONS [Debug Release ...]] [COMPONENT <component>] [RENAME <name>] [OPTIONAL [EXCLUDE_FROM_ALL]])	16
4.2.	文件权限	16
4.2.1.	安装的文件默认权限OWNER_WRITE, OWNER_READ, GROUP_READ, WORLD_READ	16
4.3.	TYPE	16
4.3.2.	include(GNUInstallDirs) install(FILES t.h TYPE doc)	18
4.3.3.	DATAROOT	18
5.	cmake install 目录	18
5.1.	语法	18
5.1.1.	install(DIRECTORY dirs... TYPE <type> DESTINATION <dir> [FILE_PERMISSIONS permissions...] [DIRECTORY_PERMISSIONS permissions...] [USE_SOURCE_PERMISSIONS] [OPTIONAL] [MESSAGE_NEVER] [CONFIGURATIONS [Debug Release ...]] [COMPONENT <component>] [EXCLUDE_FROM_ALL] [FILES_MATCHING] [[PATTERN <pattern> REGEX <regex>] [EXCLUDE] [PERMISSIONS permissions...] [...]])	18
5.2.	测试内容准备	19
5.2.1.	file(WRITE doc/index.html " ") file(WRITE doc/index.cc " ") file(WRITE doc/index.c " ") file(WRITE doc/.svn/tmp.cc " ") file(WRITE doc/.svn/tmp.html " ") file(WRITE doc/.git/tmp.cc " ") file(WRITE doc/d1/tmp.cc " ")	19
5.3.	只匹配指定类型文件, 所有目录都复制	19
5.3.1.	install(DIRECTORY doc DESTINATION doc1 FILES_MATCHING PATTERN "*.html")	19

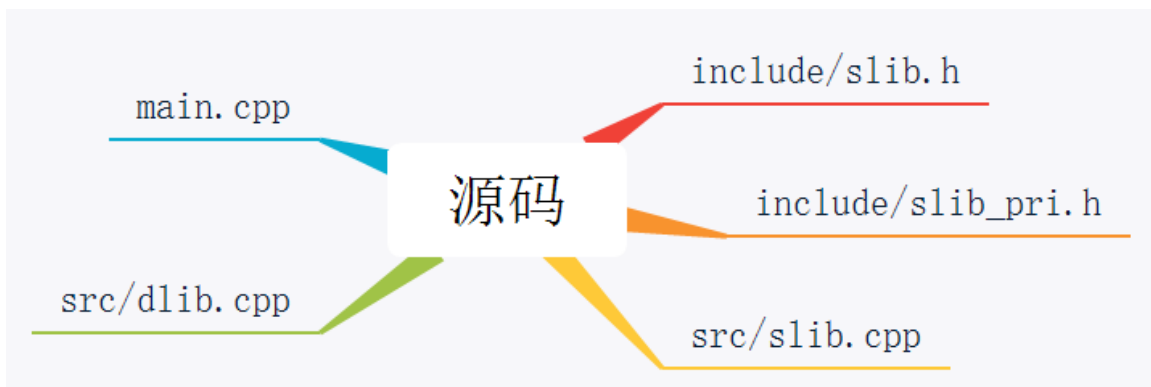
5.4.	去除所有EXCLUDE指定的目录，并匹配指定条件的文件	20
5.4.1.	install(DIRECTORY doc DESTINATION doc2 FILES_MATCHING PATTERN "*.cc" PATTERN ".git" EXCLUDE #PATTERN "d1" EXCLUDE)	20
5.5.	仅排除指定目录 加上 FILES_MATCHING 如果没有指定匹配文件内容，则不匹配任何文件	20
5.5.1.	install(DIRECTORY doc DESTINATION doc3 PATTERN ".git" EXCLUDE PATTERN ".svn" EXCLUDE #PATTERN "d1" EXCLUDE)	20
6.	安装时执行程序	20
6.1.	# %Y-%m-%dT%H:%M:%S install(CODE "MESSAGE(\"Sample install message.\")") install(CODE [= [string(TIMESTAMP now "%Y-%m-%d %H:%M:%S") message(\${now}) FILE(APPEND install_log.txt "\${now}\n")]=)	21
7.	安装指定的模块	21
7.1.	cmake .. -DCMAKE_INSTALL_PREFIX= ./	21
7.2.	cmake -DCOMPONENT=Runtime -P cmake_install.cmake	21
7.3.	install(TARGETS \${PROJECT_NAME} dlib slib RUNTIME DESTINATION bin2 COMPONENT Runtime #test_install LIBRARY DESTINATION lib2 COMPONENT Runtime # libdlib.so ARCHIVE DESTINATION lib2/myproject COMPONENT Development PUBLIC_HEADER DESTINATION pub_include COMPONENT Development PRIVATE_HEADER DESTINATION pri_include) #libslib.a	21
7.4.	cmake -DBUILD_TYPE=Debug -P cmake_install.cmake	21
8.	自定义find_package可导入库	21
8.1.	find_package	22
8.1.1.	find_package(<PackageName> [version] [EXACT] [QUIET] [MODULE] [REQUIRED] [[COMPONENTS] [components...]] [OPTIONAL_COMPONENTS components...] [NO_POLICY_SCOPE])	22
8.1.2.	<PackageName>_FOUND	22
8.1.3.	Module mode	22
8.1.4.	Config mode	22
8.1.5.	使用示例	25
8.2.	install export	25
8.2.1.	install(TARGETS slib EXPORT slib RUNTIME DESTINATION bin LIBRARY DESTINATION lib PUBLIC_HEADER DESTINATION include)	26
8.2.2.	install (EXPORT slib NAMESPACE xcpp:: FILE slibConfig.cmake DESTINATION slib)	26
8.3.	code	26
8.3.1.	code1	26
8.3.2.	code2	30



1. 测试代码准备



1.1. 源码



1.1.1. include/slib.h

include/slib.h

```
file(WRITE include/slib.h [=[  
void SLib();  
]=])
```

```
file(WRITE include/slib.h [=[  
void SLib();  
]=])
```

1.1.2. include/slib_pri.h

include/slib_pri.h

```
file(WRITE include/slib_pri.h [=[  
#define PRI  
]=])
```

```
file(WRITE include/slib_pri.h [=[  
#define PRI  
]=])
```

1.1.3. src/slib.cpp

src/slib.cpp

```
file(WRITE src/slib.cpp [=[  
#include <iostream>  
#include "slib.h"  
void SLib()  
{  
    std::cout<<"In Slib\n";  
}  
]=])
```

```
file(WRITE src/slib.cpp [=[  
#include <iostream>
```

```

#include "slib.h"
void SLib()
{
    std::cout<<"In Slib\n";
}
]=])

```

1.1.4. src/dlib.cpp

src/dlib.cpp

```

file(WRITE src/dlib.cpp [=[
#include <iostream>
#ifdef _WIN32
__declspec(dllexport)
#endif
void DLib()
{
    std::cout<<"In Dlib\n";
}
]=])

```

```

file(WRITE src/dlib.cpp [=[
#include <iostream>
#ifdef _WIN32
__declspec(dllexport)
#endif
void DLib()
{
    std::cout<<"In Dlib\n";
}
]=])

```

1.1.5. main.cpp

main.cpp

```
file(WRITE main.cpp [=[  
#include <iostream>  
#include "slib.h"  
int main()  
{  
    void DLib();  
    DLib();  
    SLib();  
    std::cout<<"In main\n";  
    return 0;  
}  
]=])
```

```
file(WRITE main.cpp [=[  
#include <iostream>  
#include "slib.h"  
int main()  
{  
    void DLib();  
    DLib();  
    SLib();  
    std::cout<<"In main\n";  
    return 0;  
}  
]=])
```

1.2. 静态库

静态库

```
add_library(slib STATIC src/slib.cpp)
```

```
set_target_properties(slib PROPERTIES PUBLIC_HEADER include/slib.h)  
set_target_properties(slib PROPERTIES PRIVATE_HEADER include/slib_pri.h)
```

1.2.1. add_library(slib STATIC src/slib.cpp)

1.2.2. `set_target_properties(slib PROPERTIES PUBLIC_HEADER include/slib.h)`
`set_target_properties(slib PROPERTIES PRIVATE_HEADER include/slib_pri.h)`

1.3. 动态库

```
add_library(dlib SHARED src/dlib.cpp)
```

动态库

1.3.1. `add_library(dlib SHARED src/dlib.cpp)`

1.4. 执行程序

```
add_executable(${PROJECT_NAME} main.cpp)
```

执行程序

1.4.1. `add_executable(${PROJECT_NAME} main.cpp)`

2. 安装命令

指定安装路径

安装命令

```
cmake --install build
```

2.1. 指定安装路径


```
cmake build -D CMAKE_INSTALL_PREFIX=.
```

指定安装路径

2.1.1. cmake build -D CMAKE_INSTALL_PREFIX=.

2.2. cmake --install build

3. 安装目标



3.1. 语法

语法

```
install(TARGETS targets... [EXPORT <export-name>]
  [RUNTIME_DEPENDENCIES args... | RUNTIME_DEPENDENCY_SET <set-name>]
  [[ARCHIVE | LIBRARY | RUNTIME | OBJECTS | FRAMEWORK | BUNDLE |
    PRIVATE_HEADER | PUBLIC_HEADER | RESOURCE | FILE_SET <set-name>]
  [DESTINATION <dir>]
  [PERMISSIONS permissions... ]
  [CONFIGURATIONS [Debug | Release | ...]]
  [COMPONENT <component>]
  [NAMELINK_COMPONENT <component>]
  [OPTIONAL] [EXCLUDE_FROM_ALL]
  [NAMELINK_ONLY | NAMELINK_SKIP]
  ] [...]
  [INCLUDES DESTINATION [<dir> ...]]
)
```

3.1.1. install(TARGETS targets... [EXPORT <export-name>]

[RUNTIME_DEPENDENCIES args... | RUNTIME_DEPENDENCY_SET <set-name>]

[[ARCHIVE | LIBRARY | RUNTIME | OBJECTS | FRAMEWORK | BUNDLE |

PRIVATE_HEADER | PUBLIC_HEADER | RESOURCE | FILE_SET <set-name>]

[DESTINATION <dir>]

[PERMISSIONS permissions...]

[CONFIGURATIONS [Debug | Release | ...]]

```

[COMPONENT <component>]
[NAMELINK_COMPONENT <component>]
[OPTIONAL] [EXCLUDE_FROM_ALL]
[NAMELINK_ONLY|NAMELINK_SKIP]
] [...]
[INCLUDES DESTINATION [<dir> ...]]
)

```

```

install(TARGETS targets... [EXPORT <export-name>]
[RUNTIME_DEPENDENCIES args...|RUNTIME_DEPENDENCY_SET <set-name>]
[[ARCHIVE|LIBRARY|RUNTIME|OBJECTS|FRAMEWORK|BUNDLE|
PRIVATE_HEADER|PUBLIC_HEADER|RESOURCE|FILE_SET <set-name>]
[DESTINATION <dir>]
[PERMISSIONS permissions...]]
[CONFIGURATIONS [Debug|Release|...]]
[COMPONENT <component>]
[NAMELINK_COMPONENT <component>]
[OPTIONAL] [EXCLUDE_FROM_ALL]
[NAMELINK_ONLY|NAMELINK_SKIP]
] [...]
[INCLUDES DESTINATION [<dir> ...]]
)

```

3.2. DESTINATION 安装路径

指定安装的目录, 可以是相对路径或绝对路径

DESTINATION 安装路径

相对路径则这相对于CMAKE_INSTALL_PREFIX

3.2.1. 指定安装的目录, 可以是相对路径或绝对路径

3.2.2. 相对路径则这相对于CMAKE_INSTALL_PREFIX

3.3. PERMISSIONS 权限

指定文件权限 OWNER_READ, OWNER_WRITE, OWNER_EXECUTE, GROUP_READ, GROUP_WRITE, GROUP_EXECUTE, WORLD_READ, WORLD_WRITE, WORLD_EXECUTE, SETUID, 和SETGID. 在某些平台上无意义的权限会被忽略。

PERMISSIONS 权限

3.3.1. 指定文件权限 OWNER_READ, OWNER_WRITE, OWNER_EXECUTE, GROUP_READ, GROUP_WRITE, GROUP_EXECUTE, WORLD_READ, WORLD_WRITE, WORLD_EXECUTE, SETUID, 和SETGID. 在某些平台上无意义的权限会被忽略。

3.4. CONFIGURATIONS (Debug Release)

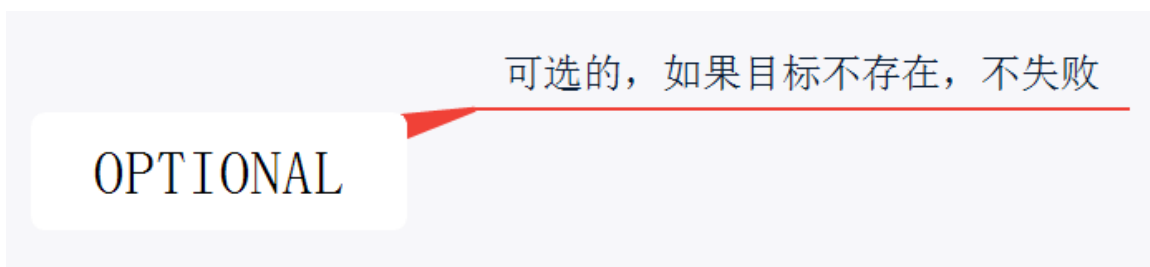


3.4.1. 指定安装规则适用的构建配置列表 (Debug, Release)

3.4.2. install(TARGETS target
CONFIGURATIONS Debug
RUNTIME DESTINATION Debug/bin)
install(TARGETS target
CONFIGURATIONS Release
RUNTIME DESTINATION Release/bin)

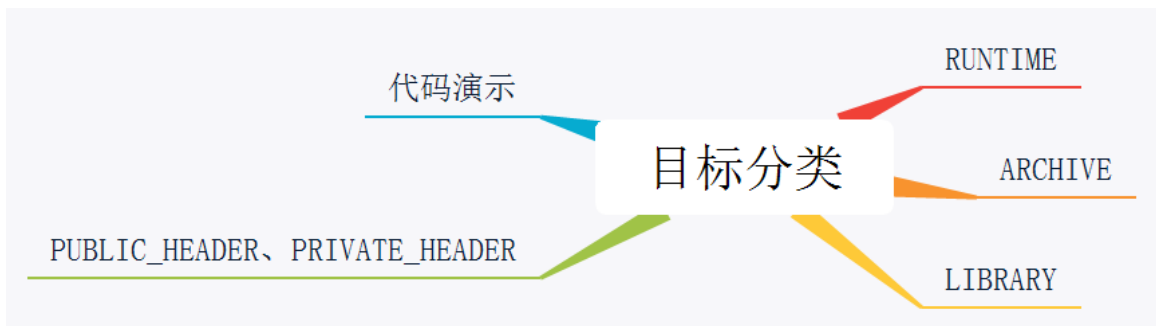
3.4.3. 需要设置在RUNTIME DESTINATION之前

3.5. OPTIONAL



3.5.1. 可选的，如果目标不存在，不失败

3.6. 目标分类



3.6.1. RUNTIME



执行程序



由add_executable创建

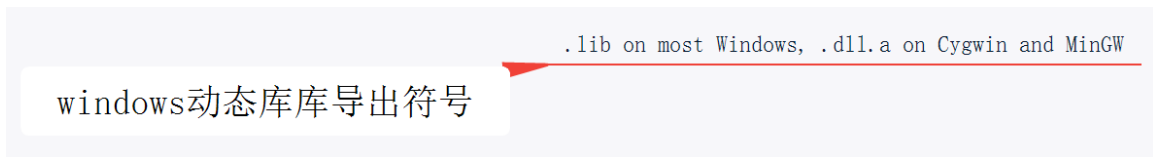
windows动态链接库dll文件

设置bin

3.6.2. ARCHIVE

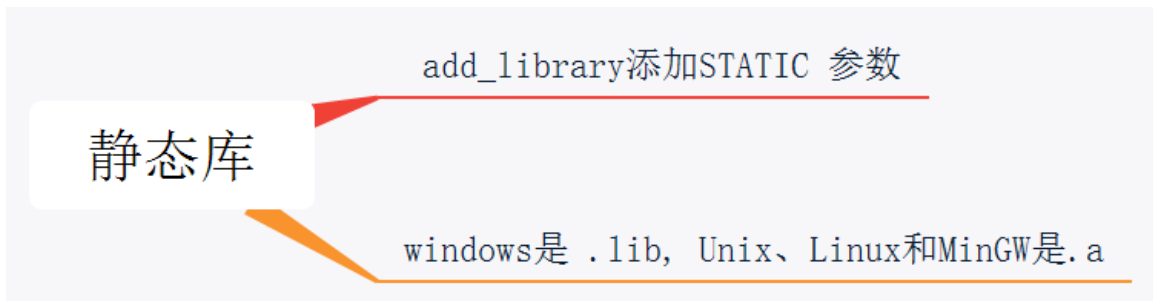


windows动态库库导出符号



.lib on most Windows, .dll.a on Cygwin and MinGW

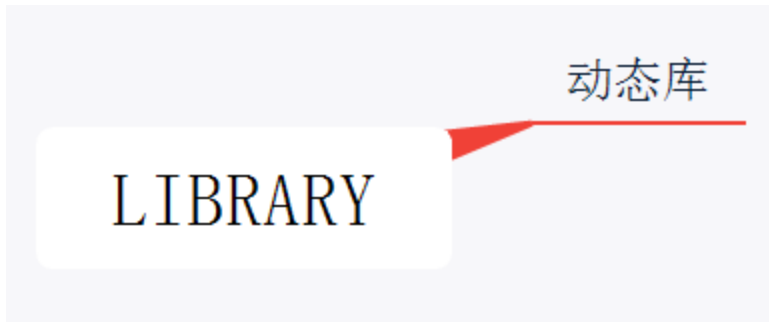
静态库



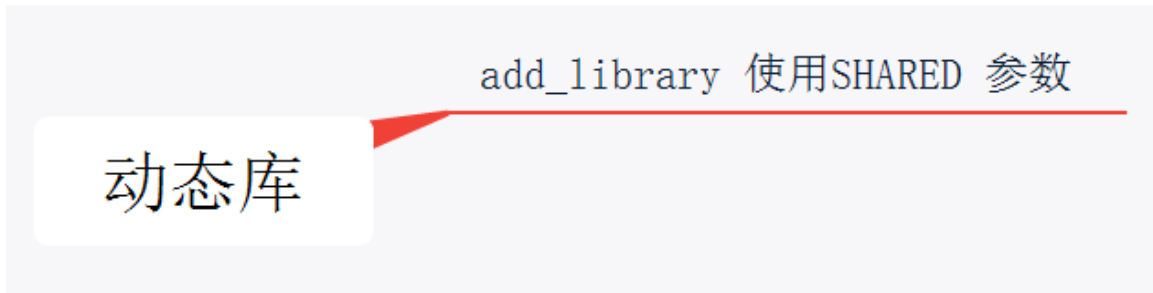
add_library添加STATIC 参数

windows是 .lib, Unix、Linux和MinGW是.a

3.6.3. LIBRARY



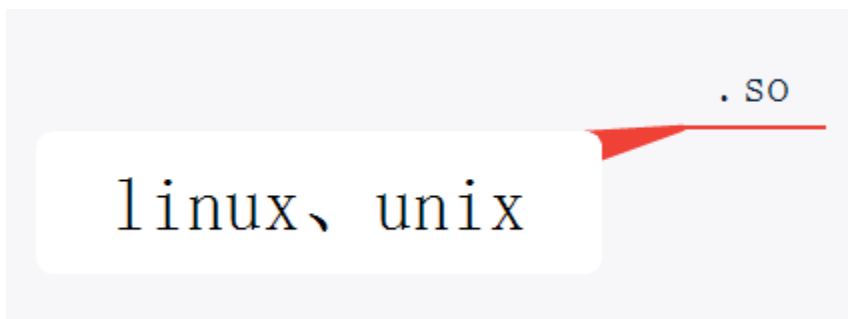
动态库



add_library 使用SHARED 参数

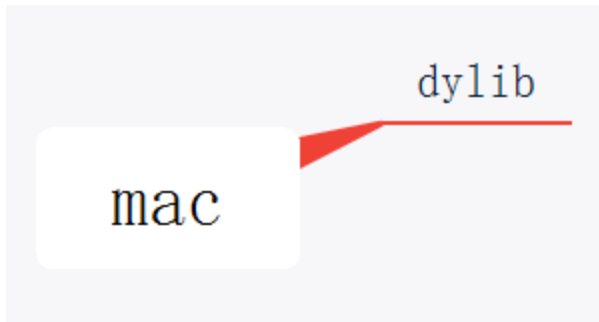


linux、unix



.so

mac



dylib

3.6.4. PUBLIC_HEADER、PRIVATE_HEADER

PUBLIC_HEADER、PRIVATE_HEADER

```
set_target_properties(slib PROPERTIES PUBLIC_HEADER include/slib.h)
set_target_properties(slib PROPERTIES PRIVATE_HEADER include/slib_pri.h)
```

set_target_properties(slib PROPERTIES PUBLIC_HEADER include/slib.h)

set_target_properties(slib PROPERTIES PRIVATE_HEADER include/slib_pri.h)

3.6.5. 代码演示

代码演示

```
install(TARGETS mylib
        RUNTIME DESTINATION bin
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib/myproject)
```

install(TARGETS mylib

RUNTIME DESTINATION bin

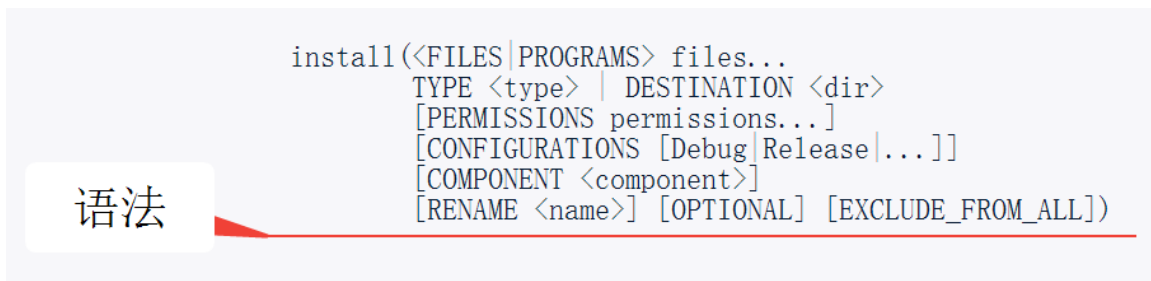
LIBRARY DESTINATION lib

ARCHIVE DESTINATION lib/myproject)

4. cmake install 安装文件



4.1. 语法



4.1.1. install(<FILES| PROGRAMS> files...

TYPE <type> | DESTINATION <dir>

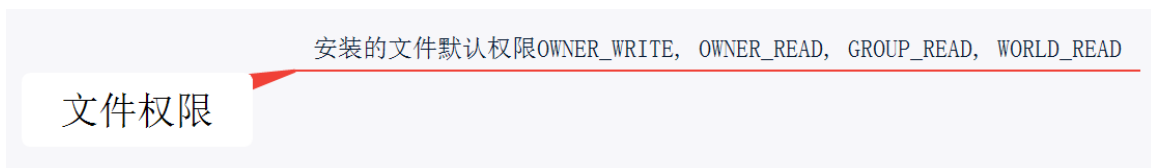
[PERMISSIONS permissions...]

[CONFIGURATIONS [Debug|Release|...]]

[COMPONENT <component>]

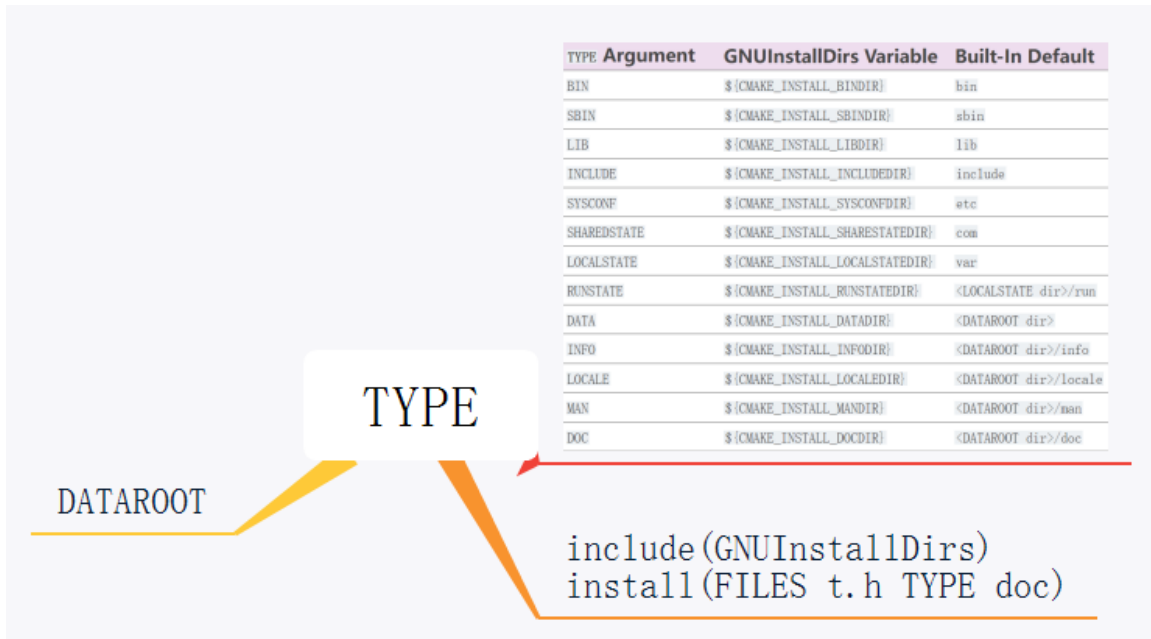
[RENAME <name>] [OPTIONAL] [EXCLUDE_FROM_ALL])

4.2. 文件权限



4.2.1. 安装的文件默认权限OWNER_WRITE, OWNER_READ, GROUP_READ, WORLD_READ

4.3. TYPE



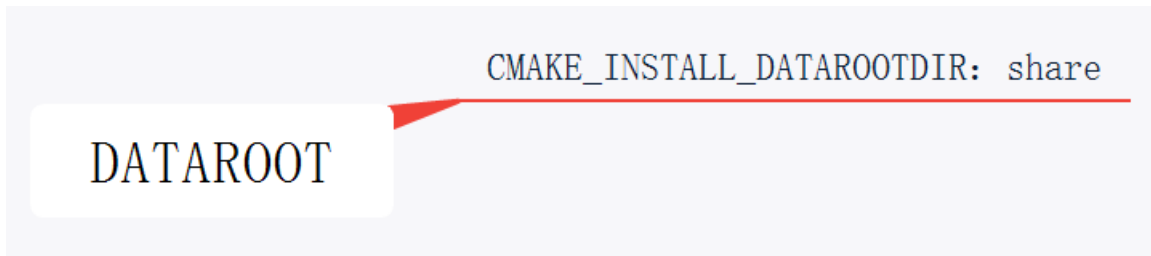
4.3.1.

TYPE	Argument	GNUInstallDirs Variable	Built-In Default
BIN		<code>\${CMAKE_INSTALL_BINDIR}</code>	<code>bin</code>
SBIN		<code>\${CMAKE_INSTALL_SBINDIR}</code>	<code>sbin</code>
LIB		<code>\${CMAKE_INSTALL_LIBDIR}</code>	<code>lib</code>
INCLUDE		<code>\${CMAKE_INSTALL_INCLUDEDIR}</code>	<code>include</code>
SYSCONF		<code>\${CMAKE_INSTALL_SYSCONFDIR}</code>	<code>etc</code>
SHAREDSTATE		<code>\${CMAKE_INSTALL_SHARESTATEDIR}</code>	<code>com</code>
LOCALSTATE		<code>\${CMAKE_INSTALL_LOCALSTATEDIR}</code>	<code>var</code>
RUNSTATE		<code>\${CMAKE_INSTALL_RUNSTATEDIR}</code>	<code><LOCALSTATE dir>/run</code>
DATA		<code>\${CMAKE_INSTALL_DATADIR}</code>	<code><DATAROOT dir></code>
INFO		<code>\${CMAKE_INSTALL_INFODIR}</code>	<code><DATAROOT dir>/info</code>
LOCALE		<code>\${CMAKE_INSTALL_LOCALEDIR}</code>	<code><DATAROOT dir>/locale</code>
MAN		<code>\${CMAKE_INSTALL_MANDIR}</code>	<code><DATAROOT dir>/man</code>
DOC		<code>\${CMAKE_INSTALL_DOCDIR}</code>	<code><DATAROOT dir>/doc</code>

4.3.2. include(GNUInstallDirs)

install(FILES t.h TYPE doc)

4.3.3. DATAROOT

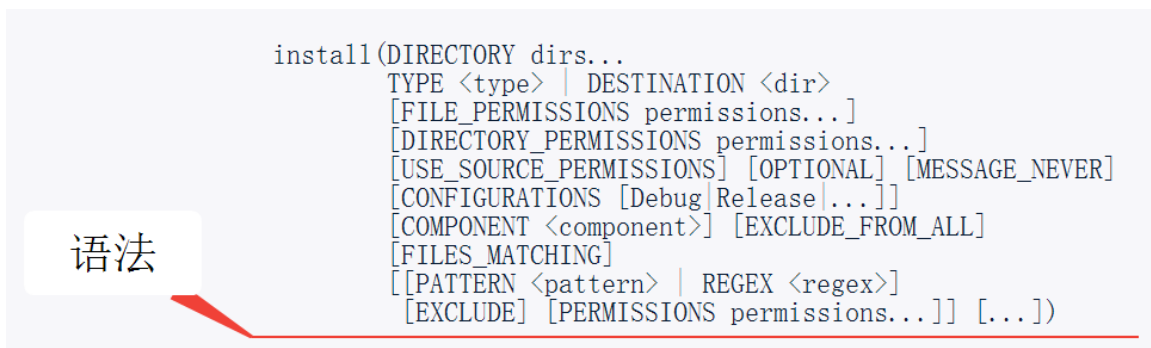


CMAKE_INSTALL_DATAROOTDIR: share

5. cmake install 目录



5.1. 语法



5.1.1. install(DIRECTORY dirs...

TYPE <type> | DESTINATION <dir>

[FILE_PERMISSIONS permissions...]

[DIRECTORY_PERMISSIONS permissions...]

[USE_SOURCE_PERMISSIONS] [OPTIONAL] [MESSAGE_NEVER]

[CONFIGURATIONS [Debug|Release|...]]

[COMPONENT <component>] [EXCLUDE_FROM_ALL]

[FILES_MATCHING]

[[PATTERN <pattern> | REGEX <regex>]

[EXCLUDE] [PERMISSIONS permissions...] [...]]

5.2. 测试内容准备

测试内容准备

```
file(WRITE doc/index.html " ")
file(WRITE doc/index.cc " ")
file(WRITE doc/index.c " ")
file(WRITE doc/.svn/tmp.cc " ")
file(WRITE doc/.svn/tmp.html " ")
file(WRITE doc/.git/tmp.cc " ")
file(WRITE doc/dl/tmp.cc " ")
```

5.2.1. file(WRITE doc/index.html " ")

file(WRITE doc/index.cc " ")

file(WRITE doc/index.c " ")

file(WRITE doc/.svn/tmp.cc " ")

file(WRITE doc/.svn/tmp.html " ")

file(WRITE doc/.git/tmp.cc " ")

file(WRITE doc/dl/tmp.cc " ")

5.3. 只匹配指定类型文件，所有目录都复制

只匹配指定类型文件，所有目录都复制

```
install(DIRECTORY doc DESTINATION doc1
FILES_MATCHING
PATTERN
"*.html"
)
```

5.3.1. install(DIRECTORY doc DESTINATION doc1

FILES_MATCHING

PATTERN

"*.html"

)

5.4. 去除所有EXCLUDE指定的目录，并匹配指定条件的文件

去除所有EXCLUDE指定的目录，并匹配指定条件的文件

```
install(DIRECTORY doc DESTINATION doc2
FILES_MATCHING
PATTERN
"*.*cc"
PATTERN ".git" EXCLUDE
#PATTERN "dl" EXCLUDE
)
```

5.4.1. install(DIRECTORY doc DESTINATION doc2

FILES_MATCHING

PATTERN

"*.*cc"

PATTERN ".git" EXCLUDE

#PATTERN "dl" EXCLUDE

)

5.5. 仅排除指定目录 加上 FILES_MATCHING

如果没有指定匹配文件内容，则不匹配任何文件

仅排除指定目录 加上 FILES_MATCHING 如果没有指定匹配文件内容，则不匹配任何文件

```
install(DIRECTORY doc DESTINATION doc3
PATTERN ".git" EXCLUDE
PATTERN ".svn" EXCLUDE
#PATTERN "dl" EXCLUDE
)
```

5.5.1. install(DIRECTORY doc DESTINATION doc3

PATTERN ".git" EXCLUDE

PATTERN ".svn" EXCLUDE

#PATTERN "dl" EXCLUDE

)

6. 安装时执行程序

安装时执行程序

```
# %Y-%m-%dT%H:%M:%S
install(CODE "MESSAGE(\"Sample install message.\")")
install(CODE [=
string(TIMESTAMP now "%Y-%m-%d %H:%M:%S")
message(${now})
FILE(APPEND install_log.txt "${now}\n")
]=])
```

6.1. # %Y-%m-%dT%H:%M:%S

```
install(CODE "MESSAGE(\"Sample install message.\")")
install(CODE [=]
string(TIMESTAMP now "%Y-%m-%d %H:%M:%S")
message(${now})
FILE(APPEND install_log.txt "${now}\n")
]=])
```

7. 安装指定的模块



7.1. cmake .. -DCMAKE_INSTALL_PREFIX=.

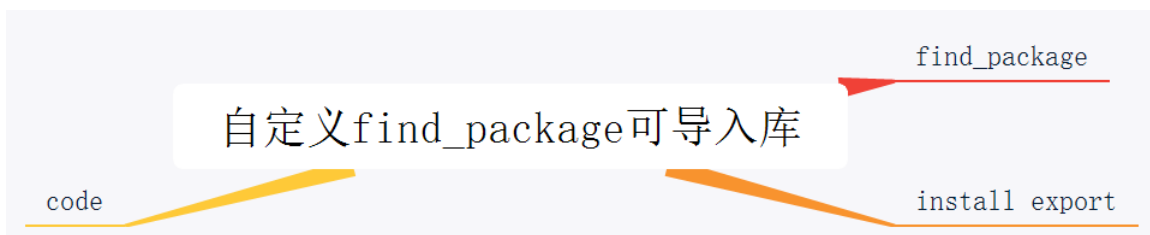
7.2. cmake -DCOMPONENT=Runtime -P cmake_install.cmake

7.3. install(TARGETS \${PROJECT_NAME} dlib slib

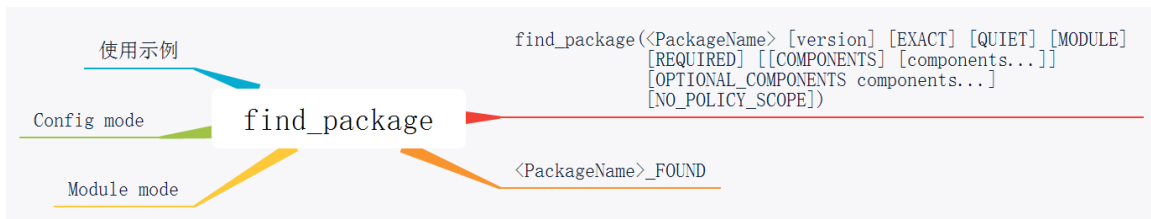
```
RUNTIME DESTINATION bin2          COMPONENT Runtime #test_install
LIBRARY DESTINATION lib2          COMPONENT Runtime # libdlib.so
ARCHIVE DESTINATION lib2/myproject COMPONENT Development
PUBLIC_HEADER DESTINATION pub_include COMPONENT Development
PRIVATE_HEADER DESTINATION pri_include
) #libslib.a
```

7.4. cmake -DBUILD_TYPE=Debug -P cmake_install.cmake

8. 自定义find_package可导入库



8.1. find_package



8.1.1. find_package(<PackageName> [version] [EXACT] [QUIET] [MODULE] [REQUIRED] [[COMPONENTS] [components...]] [OPTIONAL_COMPONENTS components...] [NO_POLICY_SCOPE])

8.1.2. <PackageName>_FOUND

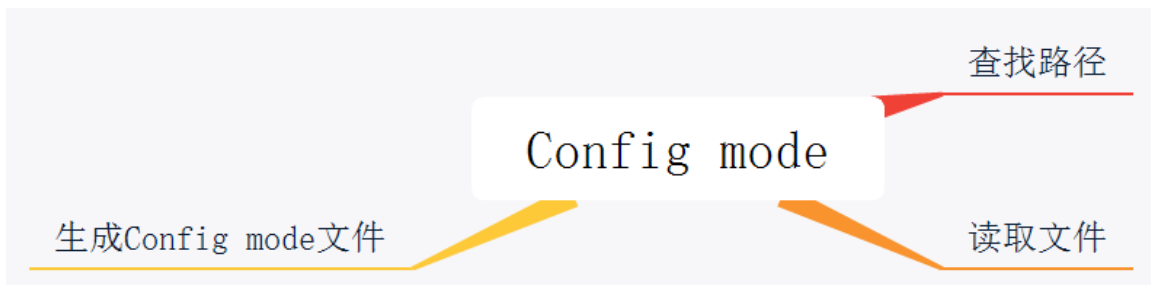
8.1.3. Module mode



Find<PackageName>.cmake

CMAKE_MODULE_PATH

8.1.4. Config mode



查找路径

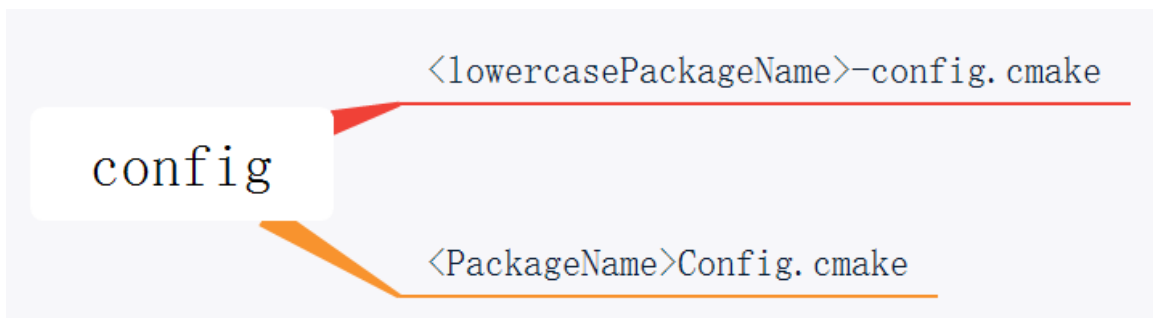


CMAKE_PREFIX_PATH

读取文件



config



<lowercasePackageName>-config.cmake

<PackageName>Config.cmake

version

version

`<lowercasePackageName>-config-version.cmake`

`<PackageName>ConfigVersion.cmake`

`<lowercasePackageName>-config-version.cmake`

`<PackageName>ConfigVersion.cmake`

生成Config mode文件

生成Config mode文件

config

version

config

config

```
install(TARGETS slib
EXPORT slib
RUNTIME DESTINATION bin
LIBRARY DESTINATION ${CMAKE_SOURCE_DIR} lib
PUBLIC_HEADER DESTINATION include
)
```

```
install (EXPORT slib
NAMESPACE xcpp::
FILE slibConfig.cmake DESTINATION mod/slib/)
```

install(TARGETS slib

EXPORT slib

RUNTIME DESTINATION bin

LIBRARY DESTINATION \${CMAKE_SOURCE_DIR} lib


```
PUBLIC_HEADER DESTINATION include
)
```

```
install (EXPORT slib
```

```
NAMESPACE xcpp::
```

```
FILE slibConfig.cmake DESTINATION mod/slib/)
```

version

version

```
include(CMakePackageConfigHelpers)
write_basic_package_version_file(${CMAKE_SOURCE_DIR}/out/mod/slib-${version}/slibConfigVersion.cmake
                                VERSION ${version}
                                COMPATIBILITY SameMajorVersion)
```

```
include(CMakePackageConfigHelpers)
```

```
write_basic_package_version_file(${CMAKE_SOURCE_DIR}/out/mod/slib-
${version}/slibConfigVersion.cmake
```

```
VERSION ${version}
```

```
COMPATIBILITY SameMajorVersion)
```

8.1.5. 使用示例

使用示例

```
find_package(slib)
add_executable(main main.cpp)
target_link_libraries(main slib)
```

```
find_package(slib)
```

```
add_executable(main main.cpp)
```

```
target_link_libraries(main slib)
```

8.2. install export

install export

```
install(TARGETS slib
EXPORT slib
RUNTIME DESTINATION bin
LIBRARY DESTINATION lib
PUBLIC_HEADER DESTINATION include
)
```

```
install (EXPORT slib
NAMESPACE xc++::
FILE slibConfig.cmake DESTINATION slib)
```

**8.2.1. install(TARGETS slib
EXPORT slib
RUNTIME DESTINATION bin
LIBRARY DESTINATION lib
PUBLIC_HEADER DESTINATION include
)**

**8.2.2. install (EXPORT slib
NAMESPACE xc++::
FILE slibConfig.cmake DESTINATION slib)**

8.3. code

code

code1

code2

8.3.1. code1

code1

```
cmake_minimum_required(VERSION 3.22)

project(slib2)
if(NOT version)
set(version 1.1)
endif()
file(WRITE include/slib.h [=[
void SLib();
]=])

file(WRITE include/slib_pri.h [=[
void SLib2();
]=])

file(WRITE slib.cpp.in [=[
#include <iostream>
void SLib()
{
    std::cout<<"test slib ${version} \n";
}
]=])

configure_file("slib.cpp.in" "${CMAKE_SOURCE_DIR}/slib.cpp" )

file(WRITE slib2.cpp [=[
#include <iostream>
void SLib()
{
    std::cout<<"test slib 1.1 \n";
}
]=])

add_library(slib SHARED slib.cpp)
set_target_properties(slib PROPERTIES VERSION ${version})

target_include_directories(slib PUBLIC /home/xcj/test_mode/out/include )
set_target_properties(slib PROPERTIES PUBLIC_HEADER include/slib.h)
set_target_properties(slib PROPERTIES PRIVATE_HEADER include/slib_pri.h)

install(TARGETS slib
EXPORT slib
RUNTIME DESTINATION bin
LIBRARY DESTINATION ${CMAKE_SOURCE_DIR}/out/mod/slib-${version}/lib
PUBLIC_HEADER DESTINATION include
PRIVATE_HEADER DESTINATION include/in
)
include(CMakePackageConfigHelpers)
write_basic_package_version_file("${CMAKE_SOURCE_DIR}/out/mod/slib-${version}/slibConfigVersion.cmake
                                VERSION ${version}
                                COMPATIBILITY SameMajorVersion)

#write_basic_package_version_file("${CMAKE_SOURCE_DIR}/out/mod1/slibConfigVersion.cmake
#                                VERSION 1.1
#                                COMPATIBILITY SameMajorVersion)
install (EXPORT slib
NAMESPACE xcpp::
        FILE slibConfig.cmake DESTINATION mod/slib-${version}/)
#install(EXPORT slib_mod NAMESPACE mp_
#install(EXPORT slib_mod
#        DESTINATION mod)

#export(PACKAGE slib_mod)
message("path is ${CMAKE_SOURCE_DIR}/out/mod/")
#set(CMAKE_MODULE_PATH "${CMAKE_SOURCE_DIR}/out/mod/")

set(CMAKE_PREFIX_PATH "${CMAKE_SOURCE_DIR}/out/mod/")
#find_package(slib ${version})

message("slib_DIR = ${slib_DIR}")
message("slib_FOUND = ${slib_FOUND}")
message("slib_INCLUDES = ${slib_INCLUDES}")
message("slib_INCLUDE_DIR = ${slib_INCLUDE_DIR}")
message("slib_LIBRARY = ${slib_LIBRARY}")
message("slib_LIBRARIES = ${slib_LIBRARIES}")

message("slib_CONSIDERED_CONFIGS = ${slib_CONSIDERED_CONFIGS}")
message("slib_CONSIDERED_VERSIONS = ${slib_CONSIDERED_VERSIONS}")
message("slib_CONFIG = ${slib_CONFIG}")

#FIND_PACKAGE(curl)
#message("CURL_DIR = ${curl_DIR}")
```

cmake_minimum_required(VERSION 3.22)

project(slib2)

```

if(NOT version)
set(version 1.1)
endif()
file(WRITE include/slib.h [=[
void SLib();
]=])

```

```

file(WRITE include/slib_pri.h [=[
void SLib2();
]=])

```

```

file(WRITE slib.cpp.in [=[
#include <iostream>
void SLib()
{
    std::cout<<"test slib ${version} \n";
}
]=])

```

```

configure_file("slib.cpp.in" "${CMAKE_SOURCE_DIR}/slib.cpp" )

```

```

file(WRITE slib2.cpp [=[
#include <iostream>
void SLib()
{
    std::cout<<"test slib 1.1 \n";
}
]=])

```

```
}  
]=])
```

```
add_library(slib SHARED slib.cpp)  
set_target_properties(slib PROPERTIES VERSION ${version})
```

```
target_include_directories(slib PUBLIC /home/xcj/test_mode/out/include )  
set_target_properties(slib PROPERTIES PUBLIC_HEADER include/slib.h)  
set_target_properties(slib PROPERTIES PRIVATE_HEADER include/slib_pri.h)
```

```
install(TARGETS slib  
EXPORT slib  
RUNTIME DESTINATION bin  
LIBRARY DESTINATION ${CMAKE_SOURCE_DIR}/out/mod/slib-${version}/lib  
PUBLIC_HEADER DESTINATION include  
PRIVATE_HEADER DESTINATION include/in  
)  
include(CMakePackageConfigHelpers)  
write_basic_package_version_file(${CMAKE_SOURCE_DIR}/out/mod/slib-  
${version}/slibConfigVersion.cmake  
VERSION ${version}  
COMPATIBILITY SameMajorVersion)
```

```
#write_basic_package_version_file(${CMAKE_SOURCE_DIR}/out/mod1/slibConf  
igVersion.cmake  
# VERSION 1.1  
# COMPATIBILITY SameMajorVersion)  
install (EXPORT slib
```

NAMESPACE xcpp::

```
    FILE slibConfig.cmake DESTINATION mod/slib-${version}/)
#install(EXPORT slib_mod NAMESPACE mp_
#install(EXPORT slib_mod
#    DESTINATION mod)

#export(PACKAGE slib_mod)
message("path is ${CMAKE_SOURCE_DIR}/out/mod/")
#set(CMAKE_MODULE_PATH "${CMAKE_SOURCE_DIR}/out/mod/")

set(CMAKE_PREFIX_PATH "${CMAKE_SOURCE_DIR}/out/mod/")
#find_package(slib ${version})

message("slib_DIR = ${slib_DIR}")
message("slib_FOUND = ${slib_FOUND}")
message("slib_INCLUDES = ${slib_INCLUDES}")
message("slib_INCLUDE_DIR = ${slib_INCLUDE_DIR}")
message("slib_LIBRARY = ${slib_LIBRARY}")
message("slib_LIBRARIES = ${slib_LIBRARIES}")

message("slib_CONSIDERED_CONFIGS = ${slib_CONSIDERED_CONFIGS}")
message("slib_CONSIDERED_VERSIONS = ${slib_CONSIDERED_VERSIONS}")
message("slib_CONFIG = ${slib_CONFIG}")

#FIND_PACKAGE(curl)
#message("CURL_DIR = ${curl_DIR}")
```

8.3.2. code2

code2

```
cmake_minimum_required(VERSION 3.22)

project(findpkg)

file(WRITE main.cpp [=[
#include <iostream>
int main()
{
    std::cout<<"test main\n";
    void SLib();
    SLib();
    return 0;
}
]=])

set(CMAKE_PREFIX_PATH "/home/xcj/test_mode/out/mod/;/home/xcj/test_mode/out/mod1/")
find_package(slib 1.2)
add_executable(main main.cpp)
target_link_libraries(main xcpp::slib)

get_target_property(pa xcpp::slib INCLUDE_DIRECTORIES)
include(CMakePrintHelpers)
cmake_print_properties(TARGETS xcpp::slib PROPERTIES
INCLUDE_DIRECTORIES
INTERFACE_INCLUDE_DIRECTORIES
)

message("xcpp::slib INCLUDE_DIRECTORIES = ${pa}")
message("slib_DIR = ${slib_DIR}")
message("slib_FOUND = ${slib_FOUND}")
message("slib_INCLUDES = ${slib_INCLUDES}")
message("slib_INCLUDE_DIR = ${slib_INCLUDE_DIR}")
message("slib_LIBRARY = ${slib_LIBRARY}")
message("slib_LIBRARIES = ${slib_LIBRARIES}")

message("slib_CONSIDERED_CONFIGS = ${slib_CONSIDERED_CONFIGS}")
message("slib_CONSIDERED_VERSIONS = ${slib_CONSIDERED_VERSIONS}")
message("slib_CONFIG = ${slib_CONFIG}")
```

cmake_minimum_required(VERSION 3.22)

project(findpkg)

file(WRITE main.cpp [=[

#include <iostream>

int main()

{

std::cout<<"test main\n";

void SLib();

SLib();

return 0;

```
}  
]=])
```

```
set(CMAKE_PREFIX_PATH  
"/home/xcj/test_mode/out/mod/;/home/xcj/test_mode/out/mod1/")  
find_package(slib 1.2)  
add_executable(main main.cpp)  
target_link_libraries(main xcpp::slib)
```

```
get_target_property(pa xcpp::slib INCLUDE_DIRECTORIES)  
include(CMakePrintHelpers)  
cmake_print_properties(TARGETS xcpp::slib PROPERTIES  
INCLUDE_DIRECTORIES  
INTERFACE_INCLUDE_DIRECTORIES  
)
```

```
message("xcpp::slib INCLUDE_DIRECTORIES = ${pa}")  
message("slib_DIR = ${slib_DIR}")  
message("slib_FOUND = ${slib_FOUND}")  
message("slib_INCLUDES = ${slib_INCLUDES}")  
message("slib_INCLUDE_DIR = ${slib_INCLUDE_DIR}")  
message("slib_LIBRARY = ${slib_LIBRARY}")  
message("slib_LIBRARIES = ${slib_LIBRARIES}")
```

```
message("slib_CONSIDERED_CONFIGS = ${slib_CONSIDERED_CONFIGS}")  
message("slib_CONSIDERED_VERSIONS = ${slib_CONSIDERED_VERSIONS}")  
message("slib_CONFIG = ${slib_CONFIG}")
```