SHUN-ReRound

CMake-002 构建自己xxxConfig.cmake

合集 - CMake(2)

1. CMake-001 入门 01-23

2. CMake-002 构建自己xxxConfig.cmake 01-25

收起

原文 创建自己的xxxConfig.cmake，用于第三方使用

添了一点注释

1. 构建自己xxxConfig.cmake

构建自己的xxxConfig.cmake，可以让第三方人员通过find_package找到并进行使用。

1.1.文件架构

整个文件的架构如下所示：

```
├── CMakeLists.txt  # cmake构建脚本
├── include # 库的头文件
│   └── plus.h
├── PLUSConfig.cmake.in # 用于生成 xxxConfig.cmake
└── src # 库的源文件
    └── plus.cpp
```

源文件文件内容

其中plus.cpp的作用是将两个整数进行求和，其内容如下;

```h
// plus.h
#ifndef PLUS_H
#define PLUS_H

__declspec(dllexport) int add(int a, int b);
#endif // PLUS_H
```

```cpp
// plus.cpp
#include "plus.h"

int add(int a, int b){
    return a+b;
}
```

构建一个库，该库会提供上述的函数plus供第三方使用

1.2.configure_package_config_file

公告

Reround
希望。。。。

昵称： ReRound
园龄： 5个月
粉丝： 0
关注： 0
+加关注

2025年7月
日 一 二 三 四 五 六
29 30 1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31 1 2
3 4 5 6 7 8 9

搜索

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签
visual studio(3)
Qt(3)
python(3)
CPP(3)
CMake(3)
mysql(1)
Matlab(1)
Linux(1)
数据库(1)
设置(1)
更多

合集
QsciScintilla(1)
VTK(4)
CMake(2)
CPP(6)
python(1)

该命令用于生成xxxCionfig.cmake文件的，其使用方式如下：

```
configure_package_config_file(
  <input>
  <output>
  INSTALL_DESTINATION <path>
  [PATH_VARS <var1> <var2> ... <varN>]
  [NO_SET_AND_CHECK_MACRO]
  [NO_CHECK_REQUIRED_COMPONENTS_MACRO]
  [INSTALL_PREFIX <path>]
  )
```

`input` ：文件名，一般为 `xxxConfig.cmake.in` 文件，需要自己提供
`output` ：文件名，一般为 `xxConfig.cmake` 文件。其会通过 `input` 中的文件进行生成
`INSTALL_DESTINATION` ：改参数后跟绝对或相对路径，表示 `output` 中的文件在 `install` 的时候会被装载到那个位置。如果使用相对路径，则其相对于 `INSTALL_PREFIX` 所表示的路径
`PATH_VARS` ：其后跟这变量的名字，这些变量需要在 `xxxConfig.cmake.in` 文件中出现。例如变量名 `A` ，则在 `xxxConfig.cmake.in` 中要以 `@PACKAGE_A@` 的形式出现。这些变量的作用一般是在 `xxxConfig.cmake.in` 生成 `xxxConfig.cmake` 的时候进行对应的变量替换
`INSTALL_PREFIX` ： `install` 时候的 `prefix path`

1.3.CMakeLists.txt编写

主要参考cmake的官方文档：Generating a Package Configuration File

`CMakeLists.txt` 的编写如下：

```
# ----------------------------------生成库----------------------------------
cmake_minimum_required(VERSION 3.4)
project(PLUS)

# 我自定义了install时候的路径，也可以定义为自己的install路径
SET(CMAKE_INSTALL_PREFIX ${PROJECT_SOURCE_DIR}/install)

add_library(plus SHARED ${CMAKE_CURRENT_SOURCE_DIR}/src/plus.cpp)
target_include_directories(plus PUBLIC
        $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
        $<INSTALL_INTERFACE:include>)

set_target_properties(plus PROPERTIES PUBLIC_HEADER "include/plus.h")

#==================生成目标文件的xxxTarget.cmake=====================
# 会将生成的库libplus.so安装到${CMAKE_INSTALL_PREFIX}/lib下
# ARCHIVE DESTINATION <path>：指定静态库（如.a文件）的安装路径。
# LIBRARY DESTINATION <path>：指定动态库（如.so文件）的安装路径。
# RUNTIME DESTINATION <path>：指定可执行文件或动态库的运行时路径（如Windows下的.dll
# PUBLIC_HEADER DESTINATION <path>：指定公共头文件的安装路径。
install(
    TARGETS plus
    EXPORT ${PROJECT_NAME}Targets
    PUBLIC_HEADER DESTINATION include
    ARCHIVE DESTINATION lib
    LIBRARY DESTINATION lib
    RUNTIME DESTINATION bin
)
# 生成 xxxTargets.cmake文件
# 指定到处的文件名和文件路径
install(
    EXPORT ${PROJECT_NAME}Targets
    FILE ${PROJECT_NAME}Targets.cmake
    DESTINATION lib/cmake/mylib
)



#====================生成 xxxConfig.cmake=============================
# 该变量会通过xxxConfig.cmake.in
# 用于在生成的xxxConfig.cmake中

```


随笔档案

2025年5月(1)
2025年2月(5)
2025年1月(14)

文章分类

Pyqt(1)

阅读排行榜

1. 【vcpkg】安装与使用(688)
2. 【python】uv 教程(534)
3. 【VS】 Visual Studio 安装离线插件(332)
4. 【工具】Gitea 的安装(229)
5. 【CPP】调用父类的重写方法(129)

```
# INCLUDE_DIRS：定义库的头文件路径。
# LIBRARIES：定义库的名称。
# LIB_DIR：定义库文件的安装路径。
set(INCLUDE_DIRS include)
set(LIBRARIES plus)
set(LIB_DIR lib)

# 包含CMake模块
include(CMakePackageConfigHelpers)

# 生成 xxxConfigVersion.cmake文件
# write_basic_package_version_file：生成一个 xxxConfigVersion.cmake 文件，该文件
# VERSION 1.1.1：指定库的版本号。
# COMPATIBILITY SameMajorVersion：指定版本兼容性规则，表示只有主版本号相同的情况下才
write_basic_package_version_file(
    ${PROJECT_BINARY_DIR}/${PROJECT_NAME}ConfigVersion.cmake
    VERSION 1.1.1
    COMPATIBILITY SameMajorVersion
)

# 用于生成 xxxConfig.cmake文件
# PATH_VARS INCLUDE_DIRS LIBRARIES LIB_DIR：指定在模板文件中需要替换的变量。
configure_package_config_file(
    ${PROJECT_SOURCE_DIR}/${PROJECT_NAME}Config.cmake.in
    ${PROJECT_BINARY_DIR}/${PROJECT_NAME}Config.cmake
    INSTALL_DESTINATION lib/cmake/mylib
    PATH_VARS INCLUDE_DIRS LIBRARIES LIB_DIR
    INSTALL_PREFIX ${CMAKE_INSTALL_PREFIX}
)

install(
    FILES ${PROJECT_BINARY_DIR}/${PROJECT_NAME}Config.cmake ${PROJECT_BINARY_
    DESTINATION lib/cmake/mylib
)
```

1.4.cmake.in文件编写

其中 `xxxConfig.cmake.in` 文件中的内容如下：

```
@PACKAGE_INIT@

set( @PROJECT_NAME@_LIBRARIES  plus)
set( @PROJECT_NAME@_INCLUDE_DIRS  @PACKAGE_INCLUDE_DIRS@)
set( @PROJECT_NAME@_LIBRARY_DIRS @PACKAGE_LIB_DIR@)

check_required_components(${PROJECT_NAME})

```

其中：

`@PACKAGE_INIT@` ：必须有，用于命令 `configure_package_config_file` 进行识别。该语句之后的命令才会被用于构建 `xxxConfig.cmake`

在 `CMakeLists.txt` 中的变量 `INCLUDE_DIRS` ，在 `xxxConfig.cmake.in` 中以 `@PACKAGE_INCLUDE_DIRS@` 形式出现，在生成xxxConfig.cmake的时候，会被替换成变量 `INCLUDE_DIRS` 表示的内容

1.5.构建

构建命令如下：

```
# Linux 下

cd <project_path>
mkdir build && cd build
cmake ..
make
make install

```

```
# windows 下

mkdir build
cd build
cmake ..
cmake --build .
cmake --install .
```

执行命令后，得到文件分布如下：

```
├── CMakeLists.txt
├── include
│   └── plus.h
├── install
│   ├── include
│   │   └── plus.h
│   └── lib
│       ├── cmake
│       │   └── mylib
│       │       ├── PLUSConfig.cmake
│       │       ├── PLUSConfigVersion.cmake
│       │       ├── PLUSTargets.cmake
│       │       └── PLUSTargets-noconfig.cmake
│       └── libplus.so
├── PLUSConfig.cmake.in
└── src
    └── plus.cpp
```

其中 xxxConfig.cmake文件的内容：

```
####### Expanded from @PACKAGE_INIT@ by configure_package_config_file() #####
####### Any changes to this file will be overwritten by the next CMake run ##
####### The input file was PLUSConfig.cmake.in                            ###

get_filename_component(PACKAGE_PREFIX_DIR "${CMAKE_CURRENT_LIST_DIR}/../../..

macro(set_and_check _var _file)
  set(${_var} "${_file}")
  if(NOT EXISTS "${_file}")
    message(FATAL_ERROR "File or directory ${_file} referenced by variable ${
  endif()
endmacro()

macro(check_required_components _NAME)
  foreach(comp ${${_NAME}_FIND_COMPONENTS})
    if(NOT ${_NAME}_${comp}_FOUND)
      if(${_NAME}_FIND_REQUIRED_${comp})
        set(${_NAME}_FOUND FALSE)
      endif()
    endif()
  endforeach()
endmacro()

####################################################################################

set( PLUS_LIBRARIES  plus)
set( PLUS_INCLUDE_DIRS  ${PACKAGE_PREFIX_DIR}/include)
set( PLUS_LIBRARY_DIRS ${PACKAGE_PREFIX_DIR}/lib)

check_required_components(${PROJECT_NAME})
```

2. 使用

我们在别的地方进行使用我们生成的xxxConfig.cmake文件。另外起一个单独的项目，其CMakeLists.txt内容如下：

```
cmake_minimum_required(VERSION 3.4)
project(Te)

# 设置find_package的查找路径
set(CMAKE_PREFIX_PATH ${CMAKE_CURRENT_SOURCE_DIR}/plus/install/lib/cmake/myli

find_package(PLUS REQUIRED)
if(PLUS_FOUND)
    message(STATUS "PLUS_FOUND = ${PLUS_FOUND}")
    message(STATUS "PLUS_INCLUDE_DIRS = ${PLUS_INCLUDE_DIRS}")
    message(STATUS "PLUS_LIBRARIES = ${PLUS_LIBRARIES}")
    message(STATUS "PLUS_LIBRARY_DIRS = ${PLUS_LIBRARY_DIRS}")
endif()


add_executable(ppp ${CMAKE_CURRENT_SOURCE_DIR}/main.cpp)

target_link_directories(ppp PUBLIC ${PLUS_LIBRARY_DIRS})
target_link_libraries(ppp plus)
target_include_directories(ppp PUBLIC ${PLUS_INCLUDE_DIRS})
```

合集: CMake

标签: CMake

好文要顶 关注我 收藏该文 微信分享


ReRound
粉丝 - 0 关注 - 0
+加关注

0 0

升级成为会员

posted @ 2025-01-25 00:17 ReRound 阅读(120) 评论(0) 收藏 举报

刷新页面 返回顶部

登录后才能查看或发表评论，立即 登录 或者 逛逛 博客园首页

【推荐】100%开源！大型工业跨平台软件C++源码提供，建模，组态！
【推荐】FlashTable：表单开发界的极速跑车，让你的开发效率一路狂飙
【推荐】Flutter适配HarmonyOS 5知识地图，实战解析+高频避坑指南
【推荐】博客园的心动：当一群程序员决定开源共建一个真诚相亲平台
【推荐】轻量又高性能的 SSH 工具 IShell：AI 加持，快人一步

编辑推荐：
· MySQL查询执行顺序：一张图看懂SQL是如何工作的
· 为什么PostgreSQL不自动缓存执行计划?
· 于是转身独立开发者
· C#.Net筑基-泛型T & 协变逆变
· dotnet 代码调试方法

阅读排行：
· 【Cursor保姆级教程】零基础小白从安装到实战，手把手教你玩转AI编程神器！
· Cursor 实战万字经验分享，与 AI 编码的深度思考
· MySQL查询执行顺序：一张图看懂SQL是如何工作的
· 用 AI 制作超长视频，保姆级教程！
· GIM 1.5发布了！ 支持Windows系统了