# CMake学习笔记

## 1.单个源文件的编译配置

## 2.同一个目录 多个源文件的编译配置

1.在项目中新建一个MathFunctions.c文件，存放一个函数 power，代码如下

double power(double base, int exponent)

{

int result = base;

int i;

if (exponent == 0) {

return 1;

}

for(i = 1; i < exponent; ++i){

result = result \* base;

}

return result;

}

2.在项目中新建一个MathFunctions.h文件，代码如下

#ifndef CMAKE\_STUDY\_MATHFUNCTIONS\_H

#define CMAKE\_STUDY\_MATHFUNCTIONS\_H

extern double power(double base, int exponent);

#endif //CMAKE\_STUDY\_MATHFUNCTIONS\_H

3.在main.c中添加MathFunctions的头文件，代码如下

#include "MathFunctions.h"

4.编写CMakeLists.txt文件，进行编译配置，代码如下

aux\_source\_directory(. DIR\_SRCS)

add\_executable(Cmake\_Study ${DIR\_SRCS})

总结：同一个项目添加多个源文件的方法

方法1：add\_executable(Cmake\_Study main.c MathFunctions.c)

方法2：

#自动查找目录下的所有源文件

#并将名称保存到 DIR\_SRCS 变量，注意. DIR\_SRCS中间有一个空格

#aux\_source\_directory(. DIR\_SRCS)

# 指定生成目标

#add\_executable(Cmake\_Study ${DIR\_SRCS})

注：add\_executable添加相对路径（单个简单项目推荐，不需要再配置其他CMakeLists）

## 3.多个目录，多个源文件

1.新建一个链接库math(project→new→directory)

2.把MathFunctions.c和MathFunctions.h文件放入math文件夹中

3.在math文件夹中新建一个CMakeLists.txt（我们将其记为math/CMakeLists.txt） ，在math/CMakeLists.txt编辑如下代码

# 查找当前目录下的所有源文件

# 并将名称保存到 DIR\_LIB\_SRCS 变量

aux\_source\_directory(. DIR\_LIB\_SRCS)

# 指定生成 MathFunctions 链接库

add\_library (MathFunctions ${DIR\_LIB\_SRCS})

注：#命令add\_library 将 src 目录中的源文件编译为静态链接库。

4.编辑CMakeLists.txt（main.c的编辑配置）代码如下

cmake\_minimum\_required(VERSION 3.14)

project(Cmake\_Study C)

set(CMAKE\_C\_STANDARD 99)

# 查找当前目录下的所有源文件

# 并将名称保存到 DIR\_SRCS 变量

aux\_source\_directory(. DIR\_SRCS)

# 指定生成目标

add\_executable(Cmake\_Study ${DIR\_SRCS})

# 添加 math 子目录

add\_subdirectory(math)

# 添加链接库

target\_link\_libraries(Cmake\_Study MathFunctions)

该文件添加了下面的内容: 使用命令 add\_subdirectory 指明本项目包含一个子目录 math，这样 math 目录下的 CMakeLists.txt 文件和源代码也会被处理 。使用命令 target\_link\_libraries 指明可执行文件 main 需要连接一个名MathFunctions 的链接库。

## 4.自定义编译选项

CMake 允许为项目增加编译选项，从而可以根据用户的环境和需求选择最合适的编译方案。

例如，可以将 MathFunctions 库设为一个可选的库，如果该选项为 ON ，就使用该库定义的数学函数来进行运算。否则就调用标准库中的数学函数库。

#### 1.修改 CMakeLists 文件

我们要做的第一步是在顶层的 CMakeLists.txt 文件中添加该选项：

cmake\_minimum\_required(VERSION 3.13)

project(CMake\_Study C)

set(CMAKE\_C\_STANDARD 99)

# 加入一个配置头文件，用于处理 CMake 对源码的设置

configure\_file (

"${PROJECT\_SOURCE\_DIR}/config.h.in"

"${PROJECT\_BINARY\_DIR}/config.h"

)

# 是否使用自己的 MathUtils 库

option (USE\_MATH\_LIB

"Use provided math implementation" ON)

# 是否加入 MathUtils 库

if (USE\_MATH\_LIB)

include\_directories ("${PROJECT\_SOURCE\_DIR}/math")

add\_subdirectory (math)

set (EXTRA\_LIBS ${EXTRA\_LIBS} MathFunctions)

endif (USE\_MATH\_LIB)

# 查找当前目录下的所有源文件

# 并将名称保存到 DIR\_SRCS 变量

aux\_source\_directory(. DIR\_SRCS)

# 指定生成目标

add\_executable(CMake\_Study ${DIR\_SRCS})

target\_link\_libraries (CMake\_Study ${EXTRA\_LIBS})

其中：

1. configure\_file 命令用于加入一个配置头文件 config.h ，这个文件由 CMake 从 config.h.in生成，通过这样的机制，将可以通过预定义一些参数和变量来控制代码的生成。
2. option 命令添加了一个 USE\_MATH\_LIB选项，并且默认值为ON。
3. 根据 USE\_MATH\_LIB变量的值来决定是否使用我们自己编写的 MathFunctions 库。

#### 2.修改 main.c 文件

之后修改 main.c文件，让其根据 USE\_MYMATH 的预定义值来决定是否调用标准库还是

#include <stdio.h>

#include <config.h>

#ifdef USE\_MATH\_LIB

#include <MathFunctions.h>

#else

#include <math.h>

#endif

int main(int argc, char \*argv[]) {

#ifdef USE\_MATH\_LIB

printf("Now we use our own Math library. \n");

hello();

#else

printf("Now we use the standard library. \n");

printf("Hello, World !\n");

#endif

printf("Hello, World Ending!\n");

return 0;

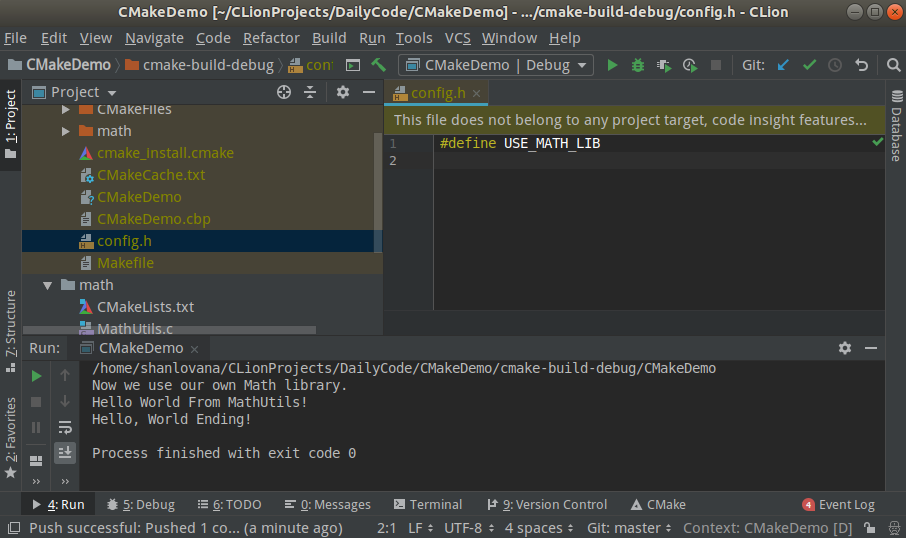
}

#### 3.编写 config.h.in文件

上面的程序值得注意的是第2行，这里引用了一个 config.h 文件，这个文件预定义了 USE\_MATH\_LIB的值。但我们并不直接编写这个文件，为了方便从 CMakeLists.txt 中导入配置，我们编写一个 config.h.in文件，内容如下：

#cmakedefine USE\_MATH\_LIB

这样 CMake 会自动根据 CMakeLists 配置文件中的设置自动生成 config.h 文件。



## 5.安装和测试

CMake 也可以指定安装规则，以及添加测试。这两个功能分别可以通过在产生 Makefile 后使用 make install 和 make test 来执行。在以前的 GNU Makefile 里，你可能需要为此编写 install 和 test 两个伪目标和相应的规则，但在 CMake 里，这样的工作同样只需要简单的调用几条命令。

1. 首先先在 math/CMakeLists.txt 文件里添加下面两行：

# 指定 MathFunctions 库的安装路径

install (TARGETS MathFunctions DESTINATION bin)

install (FILES MathFunctions.h DESTINATION include)

2.修改根目录的 CMakeLists 文件，在末尾添加下面几行：

# 指定安装路径

install (TARGETS CMake\_Study DESTINATION bin)

install (FILES "${PROJECT\_BINARY\_DIR}/config.h"

DESTINATION include)

生成的 CMake\_Study 文件和 MathFunctions 函数库 libMathFunctions.o 文件将会被复制到 /usr/local/bin 中，而 MathFunctions.h 和生成的 config.h 文件则会被复制到 /usr/local/include 中.

3.验证

[ehome@xman CMake\_Study]$ sudo make install

[ 50%] Built target MathFunctions

[100%] Built target CMake\_Study

Install the project...

-- Install configuration: ""

-- Installing: /usr/local/bin/CMake\_Study

-- Installing: /usr/local/include/config.h

-- Installing: /usr/local/bin/libMathFunctions.a

-- Up-to-date: /usr/local/include/MathFunctions.h

[ehome@xman CMake\_Study]$ ls /usr/local/bin

CMake\_Study libMathFunctions.a

[ehome@xman CMake\_Study]$ ls /usr/local/include

config.h MathFunctions.h

为工程添加测试

添加测试同样很简单。CMake 提供了一个称为 CTest 的测试工具。我们要做的只是在项目根目录的 CMakeLists 文件中调用一系列的 add\_test 命令。

# 启用测试

enable\_testing()

# 测试程序是否成功运行

add\_test (test\_run CMake\_Study 5 2)

# 测试帮助信息是否可以正常提示

add\_test (test\_usage CMake\_Study)

set\_tests\_properties (test\_usage

PROPERTIES PASS\_REGULAR\_EXPRESSION "Usage: .\* base exponent")

# 测试 5 的平方

add\_test (test\_5\_2 CMake\_Study 5 2)

set\_tests\_properties (test\_5\_2

PROPERTIES PASS\_REGULAR\_EXPRESSION "is 25")

# 测试 10 的 5 次方

add\_test (test\_10\_5 CMake\_Study 10 5)

set\_tests\_properties (test\_10\_5

PROPERTIES PASS\_REGULAR\_EXPRESSION "is 100000")

# 测试 2 的 10 次方

add\_test (test\_2\_10 CMake\_Study 2 10)

set\_tests\_properties (test\_2\_10

PROPERTIES PASS\_REGULAR\_EXPRESSION "is 1024")

上面的代码包含了四个测试。第一个测试 test\_run 用来测试程序是否成功运行并返回 0 值。剩下的三个测试分别用来测试 5 的 平方、10 的 5 次方、2 的 10 次方是否都能得到正确的结果。其中 PASS\_REGULAR\_EXPRESSION 用来测试输出是否包含后面跟着的字符串。

测试结果：

[ehome@xman CMake\_Study]$ make test

Running tests...

Test project /home/ehome/Documents/programming/C/power/ CMake\_Study

Start 1: test\_run

1/4 Test #1: test\_run ......................... Passed 0.00 sec

Start 2: test\_5\_2

2/4 Test #2: test\_5\_2 ......................... Passed 0.00 sec

Start 3: test\_10\_5

3/4 Test #3: test\_10\_5 ........................ Passed 0.00 sec

Start 4: test\_2\_10

4/4 Test #4: test\_2\_10 ........................ Passed 0.00 sec

100% tests passed, 0 tests failed out of 4

Total Test time (real) = 0.01 sec

如果要测试更多的输入数据，像上面那样一个个写测试用例未免太繁琐。这时可以通过编写宏来实现：

# 定义一个宏，用来简化测试工作

macro (do\_test arg1 arg2 result)

add\_test (test\_${arg1}\_${arg2} CMake\_Study ${arg1} ${arg2})

set\_tests\_properties (test\_${arg1}\_${arg2}

PROPERTIES PASS\_REGULAR\_EXPRESSION ${result})

endmacro (do\_test)

# 使用该宏进行一系列的数据测试

do\_test (5 2 "is 25")

do\_test (10 5 "is 100000")

do\_test (2 10 "is 1024")

## 6.支持Gdb

让 CMake 支持 gdb 的设置也很容易，只需要指定 Debug 模式下开启 -g 选项：

set(CMAKE\_BUILD\_TYPE "Debug")

set(CMAKE\_CXX\_FLAGS\_DEBUG "$ENV{CXXFLAGS} -O0 -Wall -g -ggdb")

set(CMAKE\_CXX\_FLAGS\_RELEASE "$ENV{CXXFLAGS} -O3 -Wall")

之后可以直接对生成的程序使用 gdb 来调试。

## 7.添加环境检查

有时候可能要对系统环境做点检查，例如要使用一个平台相关的特性的时候。在这个例子中，我们检查系统是否自带 pow 函数。如果带有 pow 函数，就使用它；否则使用我们定义的 power 函数。

1.添加 CheckFunctionExists 宏

首先在顶层 CMakeLists 文件中添加 CheckFunctionExists.cmake 宏，并调用 check\_function\_exists 命令测试链接器是否能够在链接阶段找到 pow 函数。

# 检查系统是否支持 pow 函数

include (${CMAKE\_ROOT}/Modules/CheckFunctionExists.cmake)

check\_function\_exists (pow HAVE\_POW)

将上面这段代码放在 configure\_file 命令前

2.预定义相关宏变量

接下来修改 [config.h.in](http://config.h.in/) 文件，预定义相关的宏变量。

// does the platform provide pow function?

#cmakedefine HAVE\_POW

3.在代码中使用宏和函数

最后一步是修改 [main.cc](http://main.cc/) ，在代码中使用宏和函数：

#ifdef HAVE\_POW

printf("Now we use the standard library. \n");

double result = pow(base, exponent);

#else

printf("Now we use our own Math library. \n");

double result = power(base, exponent);

#endif

## 8.添加版本号

给项目添加和维护版本号是一个好习惯，这样有利于用户了解每个版本的维护情况，并及时了解当前所用的版本是否过时，或是否可能出现不兼容的情况。

1.首先修改顶层 CMakeLists 文件，在 project 命令之后加入如下两行  
  
set (Demo\_VERSION\_MAJOR 1)

set (Demo\_VERSION\_MINOR 0)

分别指定当前的项目的主版本号和副版本号。

2.为了在代码中获取版本信息，我们可以修改 [config.h.in](http://config.h.in/) 文件，添加两个预定义变量

// the configured options and settings for Tutorial

#define Demo\_VERSION\_MAJOR @Demo\_VERSION\_MAJOR@

#define Demo\_VERSION\_MINOR @Demo\_VERSION\_MINOR@

3.直接在代码中打印版本信息了

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include "config.h"

#include "math/MathFunctions.h"

int main(int argc, char \*argv[])

{

if (argc < 3){

// print version info

printf("%s Version %d.%d\n",

argv[0],

Demo\_VERSION\_MAJOR,

Demo\_VERSION\_MINOR);

printf("Usage: %s base exponent \n", argv[0]);

return 1;

}

double base = atof(argv[1]);

int exponent = atoi(argv[2]);

#if defined (HAVE\_POW)

printf("Now we use the standard library. \n");

double result = pow(base, exponent);

#else

printf("Now we use our own Math library. \n");

double result = power(base, exponent);

#endif

printf("%g ^ %d is %g\n", base, exponent, result);

return 0;

}

## 9.生成安装包

本节将学习如何配置生成各种平台上的安装包，包括二进制安装包和源码安装包。为了完成这个任务，我们需要用到 CPack ，它同样也是由 CMake 提供的一个工具，专门用于打包。

1.首先在顶层的 CMakeLists.txt 文件尾部添加下面几行：

# 构建一个 CPack 安装包

include (InstallRequiredSystemLibraries)

set (CPACK\_RESOURCE\_FILE\_LICENSE

"${CMAKE\_CURRENT\_SOURCE\_DIR}/License.txt")

set (CPACK\_PACKAGE\_VERSION\_MAJOR "${Demo\_VERSION\_MAJOR}")

set (CPACK\_PACKAGE\_VERSION\_MINOR "${Demo\_VERSION\_MINOR}")

include (Cpack)

上面的代码做了以下几个工作：

1. 导入 InstallRequiredSystemLibraries 模块，以便之后导入 CPack 模块；
2. 设置一些 CPack 相关变量，包括版权信息和版本信息，其中版本信息用了上一节定义的版本号；
3. 导入 CPack 模块。

2.接下来的工作是像往常一样构建工程，并执行 cpack 命令。

生成二进制安装包：

cpack -C CpackConfig.cmake

生成源代码安装包：

cpack -C CPackSourceConfig.cmake

在生成项目后，执行 cpack -C CPackConfig.cmake 命令：

[ehome@xman Cmake\_Study]$ cpack -C CPackSourceConfig.cmake

CPack: Create package using STGZ

CPack: Install projects

CPack: - Run preinstall target for: Cmake\_Study

CPack: - Install project: Cmake\_Study

CPack: Create package

CPack: - package: /home/ehome/Documents/programming/C/power/Cmake\_Study/Cmake\_Study-1.0.1-Linux.sh generated.

CPack: Create package using TGZ

CPack: Install projects

CPack: - Run preinstall target for: Cmake\_Study

CPack: - Install project: Cmake\_Study

CPack: Create package

CPack: - package: /home/ehome/Documents/programming/C/power/Cmake\_Study/Cmake\_Study-1.0.1-Linux.tar.gz generated.

CPack: Create package using TZ

CPack: Install projects

CPack: - Run preinstall target for: Cmake\_Study

CPack: - Install project: Cmake\_Study

CPack: Create package

CPack: - package: /home/ehome/Documents/programming/C/power/Cmake\_Study/Cmake\_Study-1.0.1-Linux.tar.Z generated.

此时会在该目录下创建 3 个不同格式的二进制包文件：

[ehome@xman Cmake\_Study]$ ls Cmake\_Study-\*

Cmake\_Study-1.0.1-Linux.sh Cmake\_Study-1.0.1-Linux.tar.gz Cmake\_Study-1.0.1-Linux.tar.Z

这 3 个二进制包文件所包含的内容是完全相同的。我们可以执行其中一个。此时会出现一个由 CPack 自动生成的交互式安装界面：

[ehome@xman Cmake\_Study]$ sh Cmake\_Study-1.0.1-Linux.sh

Demo8 Installer Version: 1.0.1, Copyright (c) Humanity

This is a self-extracting archive.

The archive will be extracted to: /home/ehome/Documents/programming/C/power/ Cmake\_Study

If you want to stop extracting, please press <ctrl-C>.

The MIT License (MIT)

Copyright (c) 2013 Joseph Pan(http://hahack.com)

Permission is hereby granted, free of charge, to any person obtaining a copy of

this software and associated documentation files (the "Software"), to deal in

the Software without restriction, including without limitation the rights to

use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of

the Software, and to permit persons to whom the Software is furnished to do so,

subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS

FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR

COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER

IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN

CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Do you accept the license? [yN]:

y

By default the Demo8 will be installed in:

"/home/ehome/Documents/programming/C/power/Cmake\_Study/Cmake\_Study-1.0.1-Linux"

Do you want to include the subdirectory Cmake\_Study-1.0.1-Linux?

Saying no will install in: "/home/ehome/Documents/programming/C/power/Cmake\_Study" [Yn]:

y

Using target directory: /home/ehome/Documents/programming/C/power/Cmake\_Study/Cmake\_Study-1.0.1-Linux

Extracting, please wait...

Unpacking finished successfully

完成后提示安装到了 Cmake\_Study-1.0.1-Linux 子目录中，我们可以进去执行该程序

[ehome@xman Cmake\_Study]$ ./Cmake\_Study-1.0.1-Linux/bin/Cmake\_Study 5 2

Now we use our own Math library.

5 ^ 2 is 25

## 10.添加外部库到项目中

下面以添加osg库到项目Study\_Osg中为例。

配置文件编辑如下：

cmake\_minimum\_required(VERSION 3.14)

project(Study\_Osg)

set(CMAKE\_CXX\_STANDARD 14)

//找包

find\_package(OpenSceneGraph 3.2 REQUIRED COMPONENTS osgGA osgViewer osgDB osgUtil osg osgQt osgFX)

//查找当前目录下的所有源文件：命名为 OSGWidgets\_SRCS

aux\_source\_directory(. OSGWidgets\_SRCS)

//指定所有源文件生成动态链接库 OSGWidgets

add\_library(OSGWidgets SHARED ${OSGWidgets\_SRCS})

//配置库的头文件到链接库 OSGWidgets中

target\_include\_directories(OSGWidgets PUBLIC ${OPENSCENEGRAPH\_INCLUDE\_DIRS} )

//设置main.cpp为项目 Study\_Osg 的入口

add\_executable(Study\_Osg main.cpp)

//将库文件链接到项目中

target\_link\_libraries(Study\_Osg ${OPENSCENEGRAPH\_LIBRARIES})