<https://blog.csdn.net/dabenxiong666/article/details/53998998>

## CMake简介

CMake是一个跨平台的、开源的构建工具。cmake是makefile的上层工具，它们的目的正是为了产生可移植的makefile，并简化自己动手写makefile时的巨大工作量.目前很多开源的项目都可以通过CMake工具来轻松构建工程，例如博客之前分享的openHMD、hidapi、OSVR-Core等等，代码的分享者提供源代码和相应的Cmake配置文件，使用者就可以非常方便的在自己的电脑上构建相应的工程，进行开发和调试。

## CMake链接

* [CMake官网](https://cmake.org/)
* [CMakeWiki](https://cmake.org/Wiki/CMake)
* [CMake3.7 Document](https://cmake.org/cmake/help/v3.7/manual/cmake.1.html)

## CMake教程

[官方英文教程](https://cmake.org/cmake-tutorial/),黑体部分是本人在操作过程中,觉得需要额外提示的地方,整理翻译如下:

### 第一步：一个简单的起点

最简单的工程就是将一些源文件编译为可执行程序。对于简单工程来说，只需要在CMakeList.txt添加2行内容即可。这就是我们这个教程的起点，CMakeLists.txt文件内容如下：

cmake\_minimum\_required (VERSION 2.6)

project (Tutorial)

add\_executable(Tutorial tutorial.cxx)

注意：CMake文件是不区分大小写的，这个例子中CMakeLists.txt文件中都使用小写字母。源文件”tutorial.cxx”用来计算一个数的平方根，它的第一个版本非常简单，如下所示：

// A simple program that computes the square root of a number

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

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

{

if (argc < 2)

{

fprintf(stdout,"Usage: %s number\n",argv[0]);

return 1;

}

double inputValue = atof(argv[1]);

double outputValue = sqrt(inputValue);

fprintf(stdout,"The square root of %g is %g\n",

inputValue, outputValue);

return 0;

}

**添加版本号和配置头文件**   
我们添加的第一个特性就是为我们的可执行程序和工程提供一个版本号.你可以在源代码中写入版本号,然而在CMakeLists.txt中提供(版本号)更灵活.修改CMakeLists.txt增加一个版本号,内容如下：

cmake\_minimum\_required (VERSION 2.6)

project (Tutorial)

# The version number.

set (Tutorial\_VERSION\_MAJOR 1)

set (Tutorial\_VERSION\_MINOR 0)

# configure a header file to pass some of the CMake settings

# to the source code

configure\_file (

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

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

)

# add the binary tree to the search path for include files

# so that we will find TutorialConfig.h

include\_directories("${PROJECT\_BINARY\_DIR}")

# add the executable

add\_executable(Tutorial tutorial.cxx)

因为配置文件将会被写入到二进制树当中，我们需要把这个目录添加到头文件路径当中。我们接着创建一个”TutorialConfig.h.in”文件，内容如下：

// the configured options and settings for Tutorial

#define Tutorial\_VERSION\_MAJOR @Tutorial\_VERSION\_MAJOR@

#define Tutorial\_VERSION\_MINOR @Tutorial\_VERSION\_MINOR@

当CMake配置这个头文件是，”@Tutorial\_VERSION\_MAJOR@”和”@Tutorial\_VERSION\_MINOR@”的值将会被CMakeLists.txt文件中的值替换。下一步我们修改”tutorial.cxx”来包含配置头文件来使用版本号。修改后的代码如下:

// A simple program that computes the square root of a number

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include "TutorialConfig.h"

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

{

if (argc < 2)

{

fprintf(stdout,"%s Version %d.%d\n",

argv[0],

Tutorial\_VERSION\_MAJOR,

Tutorial\_VERSION\_MINOR);

fprintf(stdout,"Usage: %s number\n",argv[0]);

return 1;

}

double inputValue = atof(argv[1]);

double outputValue = sqrt(inputValue);

fprintf(stdout,"The square root of %g is %g\n",

inputValue, outputValue);

return 0;

}

上面代码最主要的修改就是包含了TutorialConfig.h头文件，并且打印出版本号。

### 第二步：添加一个库

现在我们为我们的工程添加一个库。这个库包含了自己实现的一个用来计算数的平方根函数。应用程序可以使用这个库来计算平方根，而不是使用编译器提供的标准库。教程里我们会把这个库放到一个叫做”MathFunctions”的子文件夹中。它包含一个只有一行内容的CMakeLists.txt文件：

add\_library(MathFunctions mysqrt.cxx)

源文件”mysqrt.cxx”有一个叫做mysqrt的函数，它跟编译器提供的sqrt函数类似。为了使用这个新库，我们要在顶层的CMakeLists.txt中添加”add\_subdirectory”调用,以便使这个库能够被编译到.我们还需要添加另外一个包含文件夹以便找到包含在头文件”MathFunctions/MathFunctions.h”中的函数原型.最后把新库添加到应用程序当中,在顶层的CMakeLists.txt文件中添加如下内容:

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

add\_subdirectory (MathFunctions)

# add the executable

add\_executable (Tutorial tutorial.cxx)

target\_link\_libraries (Tutorial MathFunctions)

**注意:这里需要自己手动在MathFunctions文件夹里添加”mysqrt.cxx”和”MathFunctions.h”文件,这2个文件里的代码可以参考文章最后提供的附件**   
现在让我们把MathFunctions库做成可选的.当然在这个教程的工程中这个操作有点多余.但是在你的其他工程中,多数情况会有大量的依赖第三方的库文件.添加一个可选库,需要在最高层的CMakeLists.txt文件中添加一些内容:

# should we use our own math functions?

option (USE\_MYMATH

"Use tutorial provided math implementation" ON)

这个USE\_MYMATH将会显示在CMake GUI当中,并且默认值为ON,这个设置将会被保存到CACHE当中,所以用户每次打开工程时不必重新配置.下一步需要做一些修改,以便使MathFunctions库参与编译和链接.这个修改是在顶层的CMakeLists.txt文件中,内容如下:

# add the MathFunctions library?

#

if (USE\_MYMATH)

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

add\_subdirectory (MathFunctions)

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

endif (USE\_MYMATH)

# add the executable

add\_executable (Tutorial tutorial.cxx)

target\_link\_libraries (Tutorial ${EXTRA\_LIBS})

通过配置USB\_MYMATH来决定是否编译和使用MathFunctions.注意上面的”EXTRA\_LIBS”变量是用来收集每一个可选库的,以便他们链接到可执行程序中.这是一个保持大项目工程可选库清洁的通用方法.相应的源代码修改是很直观的,如下:

// A simple program that computes the square root of a number

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include "TutorialConfig.h"

#ifdef USE\_MYMATH

#include "MathFunctions.h"

#endif

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

{

if (argc < 2)

{

fprintf(stdout,"%s Version %d.%d\n", argv[0],

Tutorial\_VERSION\_MAJOR,

Tutorial\_VERSION\_MINOR);

fprintf(stdout,"Usage: %s number\n",argv[0]);

return 1;

}

double inputValue = atof(argv[1]);

#ifdef USE\_MYMATH

double outputValue = mysqrt(inputValue);

#else

double outputValue = sqrt(inputValue);

#endif

fprintf(stdout,"The square root of %g is %g\n",

inputValue, outputValue);

return 0;

}

在源文件中,我们也使用了USE\_MYMATH,CMake通过修改配置文件TutorialConfig.h.in配置文件来为源文件这个提供支持,添加内容如下:

#cmakedefine USE\_MYMATH

### 第三步:安装和测试

下一步我们将为我们的工程添加安装规则和测试.安装规则相当简单,为了安装MathFunctions库和头文件,我们需要在MathFunctions文件夹的CMakeLists.txt文件中,添加如下内容:

install (TARGETS MathFunctions DESTINATION bin)

install (FILES MathFunctions.h DESTINATION include)

对于应用程序,需要在最顶层的CMakeLists.txt文件中安装可执行程序和配置头文件.

# add the install targets

install (TARGETS Tutorial DESTINATION bin)

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

DESTINATION include)

**注意:上面这几行内容,需要添加在如下代码之后,否则编译会报错,提示找不到”Tutorial”**

# add the executable

add\_executable (Tutorial tutorial.cxx)

target\_link\_libraries (Tutorial ${EXTRA\_LIBS})

这就是所有要做的.到这里你应该可以构建这个教程了,然后输入”make install”(或则使用IDE编译出INSTALL目标),它将安装适当的头文件,库,和可执行程序.CMake变量”CMAKE\_INSTALL\_PREFIX”用来决定这些文件将被安装的根路径.添加测试也是很简单的.在顶层的CMakeLists.txt文件中添加一些简单的测试,来确认应用程序工作正常.

include(CTest)

# does the application run

add\_test (TutorialRuns Tutorial 25)

# does it sqrt of 25

add\_test (TutorialComp25 Tutorial 25)

set\_tests\_properties (TutorialComp25 PROPERTIES PASS\_REGULAR\_EXPRESSION "25 is 5")

# does it handle negative numbers

add\_test (TutorialNegative Tutorial -25)

set\_tests\_properties (TutorialNegative PROPERTIES PASS\_REGULAR\_EXPRESSION "-25 is 0")

# does it handle small numbers

add\_test (TutorialSmall Tutorial 0.0001)

set\_tests\_properties (TutorialSmall PROPERTIES PASS\_REGULAR\_EXPRESSION "0.0001 is 0.01")

# does the usage message work?

add\_test (TutorialUsage Tutorial)

set\_tests\_properties (TutorialUsage PROPERTIES PASS\_REGULAR\_EXPRESSION "Usage:.\*number")

在构建完成之后需要运行”ctest”命令来运行上面的测试.第一个测试用例用来确保应用程序是否有段错误,crash,返回值非0.这个是一个简单的CTest测试.接下来的几个测试都是利用”PASS\_REGULAR\_EXPRESSION”测试属性来验证输出是否包含特定字符串.在这个例子当中验证平方根计算是否正确,以及当输入错误信息时候输出使用信息.如果你希望添加更多测试来测试不同的输入值,你可以考虑创建一个宏像如下这样:

#define a macro to simplify adding tests, then use it

macro (do\_test arg result)

add\_test (TutorialComp${arg} Tutorial ${arg})

set\_tests\_properties (TutorialComp${arg}

PROPERTIES PASS\_REGULAR\_EXPRESSION ${result})

endmacro (do\_test)

# do a bunch of result based tests

do\_test (25 "25 is 5")

do\_test (-25 "-25 is 0")

do\_test的每一次调用,就会有一次测试被添加到工程当中,测试的名字、输入、结果,基于传递的参数.

### 第四步:添加系统自检

下一步让我们考虑添加一些代码到我们的工程,以支持目标平台没有的特性.在这个例子当中,我们将添加一些代码来验证目标平台是否具有log和exp函数.当然几乎所有的平台都有这些函数,教程假设一下这种少数情况.如果平台有log函数,那么我们在mysqrt函数中用它来计算输出.第一步我们利用CheckFunctionExists.cmake宏测试这些函数是否有效,在顶层CMakeLists.txt文件中添加如下内容:

# does this system provide the log and exp functions?

include (CheckFunctionExists)

check\_function\_exists (log HAVE\_LOG)

check\_function\_exists (exp HAVE\_EXP)

接着我们修改”TutorialConfig.h.in”来定义那些CMake在平台上查找到的值,修改如下:

// does the platform provide exp and log functions?

#cmakedefine HAVE\_LOG

#cmakedefine HAVE\_EXP

在使用log和exp之前,确定他们是否被定义,是非常重要的.在CMake中,配置文件的命令会立刻使用目前的配置.最后在mysqrt函数中,我们可以提供一个基于log和exp函数的实现,代码如下:

// if we have both log and exp then use them

#if defined (HAVE\_LOG) && defined (HAVE\_EXP)

result = exp(log(x)\*0.5);

#else // otherwise use an iterative approach

. . .

**注意:这里需要在”mysqrt.cxx”文件里添加头文件#include “TutorialConfig.h”,否则找不到定义的宏**

### 第5步:添加/生成一个通用的文件

这一节我们将演示如何添加一个源文件到一个应用程序当中.在这个例子当中,我们将会创建一个已经计算好的平方根表,它作为构建过程的一部分,并且编译进我们的应用程序.为了实现这个功能,我们需要一个程序来生成这个表.在MathFunctions子文件夹中我们添加一个叫做MakeTable.cxx的文件,它的内容如下:

// A simple program that builds a sqrt table

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

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

{

int i;

double result;

// make sure we have enough arguments

if (argc < 2)

{

return 1;

}

// open the output file

FILE \*fout = fopen(argv[1],"w");

if (!fout)

{

return 1;

}

// create a source file with a table of square roots

fprintf(fout,"double sqrtTable[] = {\n");

for (i = 0; i < 10; ++i)

{

result = sqrt(static\_cast<double>(i));

fprintf(fout,"%g,\n",result);

}

// close the table with a zero

fprintf(fout,"0};\n");

fclose(fout);

return 0;

}

注意,这个表格是用C++代生成的,并且文件的名字是通过该程序的运行参数传入的.下一步就是添加一个适当的命令到MathFunctions的CMakeLists.txt文件来构建MakeTable应用程序,然后在构建过程中运行它.完成这些操作,我们需要添加如下命令:

# first we add the executable that generates the table

add\_executable(MakeTable MakeTable.cxx)

# add the command to generate the source code

add\_custom\_command (

OUTPUT ${CMAKE\_CURRENT\_BINARY\_DIR}/Table.h

COMMAND MakeTable ${CMAKE\_CURRENT\_BINARY\_DIR}/Table.h

DEPENDS MakeTable

)

# add the binary tree directory to the search path for

# include files

include\_directories( ${CMAKE\_CURRENT\_BINARY\_DIR} )

# add the main library

add\_library(MathFunctions mysqrt.cxx ${CMAKE\_CURRENT\_BINARY\_DIR}/Table.h )

首先,可执行程序MakeTable像所有可执行程序一样被添加.接着我们添加一些命令来让MakeTable生成Table.h.然后我们让CMake知道mysqrt.cxx依赖于生成的Table.h.这需要为MathFunctions库把生成的Table.h文件添加到源文件列表当中,我们还需要添加当前的二进制文件夹添加到include文件夹列表当中,以便Table.h文件可以被mysqrt.cxx找到并包含.当工程被构建时,它将会先构建MakeTable可执行程序.它将运行MakeTable来生成Table.h文件.最后,它编译包含了Table.h文件的mysqrt.cxx来生成MathFunctions库.为了达到这个目的,顶层的CMakeLists.txt需要添加如下代码:

cmake\_minimum\_required (VERSION 2.6)

project (Tutorial)

include(CTest)

# The version number.

set (Tutorial\_VERSION\_MAJOR 1)

set (Tutorial\_VERSION\_MINOR 0)

# does this system provide the log and exp functions?

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

check\_function\_exists (log HAVE\_LOG)

check\_function\_exists (exp HAVE\_EXP)

# should we use our own math functions

option(USE\_MYMATH

"Use tutorial provided math implementation" ON)

# configure a header file to pass some of the CMake settings

# to the source code

configure\_file (

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

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

)

# add the binary tree to the search path for include files

# so that we will find TutorialConfig.h

include\_directories ("${PROJECT\_BINARY\_DIR}")

# add the MathFunctions library?

if (USE\_MYMATH)

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

add\_subdirectory (MathFunctions)

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

endif (USE\_MYMATH)

# add the executable

add\_executable (Tutorial tutorial.cxx)

target\_link\_libraries (Tutorial ${EXTRA\_LIBS})

# add the install targets

install (TARGETS Tutorial DESTINATION bin)

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

DESTINATION include)

# does the application run

add\_test (TutorialRuns Tutorial 25)

# does the usage message work?

add\_test (TutorialUsage Tutorial)

set\_tests\_properties (TutorialUsage

PROPERTIES

PASS\_REGULAR\_EXPRESSION "Usage:.\*number"

)

#define a macro to simplify adding tests

macro (do\_test arg result)

add\_test (TutorialComp${arg} Tutorial ${arg})

set\_tests\_properties (TutorialComp${arg}

PROPERTIES PASS\_REGULAR\_EXPRESSION ${result}

)

endmacro (do\_test)

# do a bunch of result based tests

do\_test (4 "4 is 2")

do\_test (9 "9 is 3")

do\_test (5 "5 is 2.236")

do\_test (7 "7 is 2.645")

do\_test (25 "25 is 5")

do\_test (-25 "-25 is 0")

do\_test (0.0001 "0.0001 is 0.01")

TutorialConfig.h.in包含如下内容:

// the configured options and settings for Tutorial

#define Tutorial\_VERSION\_MAJOR @Tutorial\_VERSION\_MAJOR@

#define Tutorial\_VERSION\_MINOR @Tutorial\_VERSION\_MINOR@

#cmakedefine USE\_MYMATH

// does the platform provide exp and log functions?

#cmakedefine HAVE\_LOG

#cmakedefine HAVE\_EXP

MathFunctions文件夹中的CMakeLists.txt文件,内容如下:

# first we add the executable that generates the table

add\_executable(MakeTable MakeTable.cxx)

# add the command to generate the source code

add\_custom\_command (

OUTPUT ${CMAKE\_CURRENT\_BINARY\_DIR}/Table.h

DEPENDS MakeTable

COMMAND MakeTable ${CMAKE\_CURRENT\_BINARY\_DIR}/Table.h

)

# add the binary tree directory to the search path

# for include files

include\_directories( ${CMAKE\_CURRENT\_BINARY\_DIR} )

# add the main library

add\_library(MathFunctions mysqrt.cxx ${CMAKE\_CURRENT\_BINARY\_DIR}/Table.h)

install (TARGETS MathFunctions DESTINATION bin)

install (FILES MathFunctions.h DESTINATION include)

### 第6步:构建一个安装器

接着假设我们想把工程发布给其他人使用.我们希望提供不同平台的二进制文件和源文件.这里与我们之前在第三步中的安装有一些不同,那时我们安装我们已经从源代码构建出来的二进制文件.在这个例子中,我们将构建一个安装包,它支持二进制安装,以及cygwin、 debian、RPMs等工具的包管理特性.为了实现这个功能,我们会使用CPack来创建平台相关的安装包.我们需要在顶层的CMakeLists.txt文件中的最后添加一些代码,内容如下:

# build a CPack driven installer package

include (InstallRequiredSystemLibraries)

set (CPACK\_RESOURCE\_FILE\_LICENSE

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

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

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

include (CPack)

**官方教程中没有License.txt文件,所有要手动在顶层目录下添加一个”License.txt”文件**   
以上就是我们所有要做的.以包含InstallRequiredSystemLibraries为开始.这个模块将包含工程在目前平台上所需要的所有运行时库.接着我们设置一些CPack变量,包括我们保存license的地方,工程的版本信息.版本信息就是用了我们教程中之前设置的变量值.最后我们包含CPake模块,它将使用这些变量和一些系统的其他属性来设置安装包.   
下一步是以通常的方法构建工程,然后在CPack上运行它.为了构建二进制发行包,你需要运行如下命令:

cpack --config CPackConfig.cmake

为了创建一个源文件发行包,你需要运行:

cpack --config CPackSourceConfig.cmake

**注意:这里的2条命令需要在构建工程的目录下运行!**

### 第7步:添加对Dashboard支持

将我们的测试结果添加到Dashboard上是非常容易的.在教程的前面步骤当中我们已经为我们的工程定义了一些测试.我们只需要运行这些测试,并且提交他们到一个dashboard上.为了支持dashboard我们需要在顶层的CMakeLists.txt文件中包含CTest模块.

# enable dashboard scripting

include (CTest)

我们还需要创建一个CTestConfig.cmake文件,用来为dashboard指定工程名字.

set (CTEST\_PROJECT\_NAME "Tutorial")

CTest运行时将读取这个文件.你可以在你的工程中运行CMake来生成一个简单的dashboard,进入构建目录下,然后执行”ctest -D Experimental”,将会把你的dashboard结果上传到Kitware’s工程dashboard上.

到这里教程完毕,[这里](https://pan.baidu.com/s/1jIwQyJc)有7步操作的文件,有需要的点击获取.

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/dabenxiong666/article/details/53998998