

# CMake简要教程



作者 行之与亦安 (/u/849a39246ef1) +关注

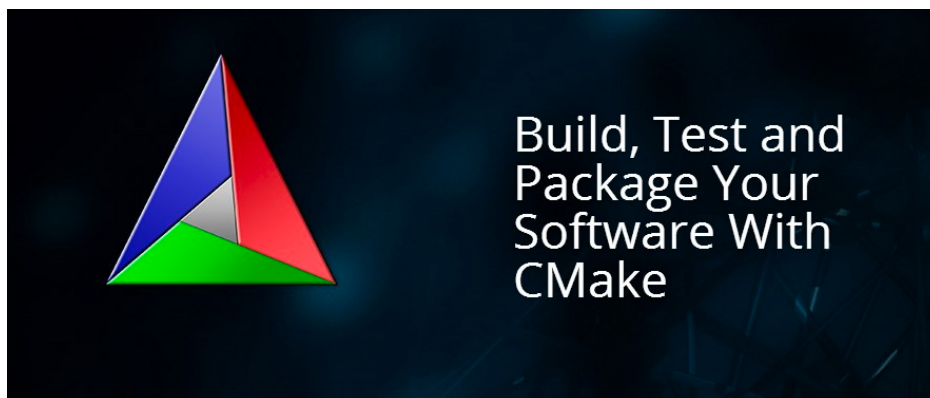
2016.09.18 08:26 字数 2757 阅读 4380 评论 5 喜欢 15

(/u/849a39246ef1)

CMake是我非常喜欢且一直使用的工具。它不但能帮助我跨平台、跨编译器，而且最酷的是，它帮我节约了太多的存储空间。特别是与水银 (<http://www.jianshu.com/p/c3119b57174a>)结合起来使用，其友好的体验，足以给我们这些苦逼码农一丝慰藉。

以下内容翻译自官网教程 (<https://cmake.org/cmake-tutorial/>)

## CMake Tutorial



### A Basic Starting Point (Step 1)

最基本的就是将一个源代码文件编译成一个exe可执行程序。对于一个简单的工程来说，两行的CMakeLists.txt文件就足够了。这将是我们的教程的开始。CMakeLists.txt文件看起来会像这样：

```
cmake_minimum_required (VERSION 2.6)
project (Tutorial)
add_executable(Tutorial tutorial.cxx)
```

注意，在这个例子中，CMakeLists.txt都是使用的小写字母。事实上，CMake命令是大小写不敏感的，你可以用大写，也可以用小写，也可以混写。tutorial.cxx源码会计算出一个数的平方根。它的第一个版本看起来非常简单，如下：



```
// 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;
}
```

## Adding a Version Number and Configured Header File

我们第一个要加入的特性是，在工程和可执行程序上加一个版本号。虽然你可以直接在源代码里面这么做，然而如果用CMakeLists文件来做的话会提供更多的灵活性。为了增加版本号，我们可以如此更改CMakeLists文件：

```
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)
```

由于配置文件必须写到binary tree中，因此我们必须将这个目录添加到头文件搜索目录中。我们接下来在源码目录中创建了TutorialConfig.h.in文件，其内容如下：

```
// the configured options and settings for Tutorial
#define Tutorial_VERSION_MAJOR @Tutorial_VERSION_MAJOR@
#define Tutorial_VERSION_MINOR @Tutorial_VERSION_MINOR@
```

当CMake配置了这个头文件，@Tutorial\_VERSION\_MAJOR@ 和 @Tutorial\_VERSION\_MINOR@ 的值将会被改变。接下来，我们修改了tutorial.cxx来包含配置的头文件并且使用版本号。最终的源代码如下所示：



```
// 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;
}
```

最主要的变更是包含了TutorialConfig.h头文件，并输出了版本号。

## Adding a Library (Step 2)

现在我们给工程添加一个库。这个库会包含我们自己的平方根实现。如此，应用程序就可以使用这个库而非编译器提供的库了。在这个教程中，我们将库放入一个叫MathFunctions的子文件夹中。它会使用如下的一行CMakeLists文件：

```
add_library(MathFunctions mysqrt.cxx)
```

原文件mysqrt.cxx有一个叫做mysqrt的函数可以提供与编译器的sqrt相似的功能。为了使用新的库，我们需要在顶层的CMakeLists 文件中添加add\_subdirectory的调用。我们也要添加一个另外的头文件搜索目录，使得MathFunctions/mysqrt.h可以被搜索到。最后的改变就是将新的库加到可执行程序中。顶层的CMakeLists 文件现在看起来是这样：

```
include_directories ("${PROJECT_SOURCE_DIR}/MathFunctions")
add_subdirectory (MathFunctions)

# add the executable
add_executable (Tutorial tutorial.cxx)
target_link_libraries (Tutorial MathFunctions)
```

现在我们来考虑如何使得MathFunctions库成为可选的。虽然在这个教程当中没有什么理由这么做，然而如果使用更大的库或者当依赖于第三方的库时，你或许希望这么做。第一步是要在顶层的CMakeLists文件中加上一个选择项。

```
# should we use our own math functions?
option (USE_MYMATH
    "Use tutorial provided math implementation" ON)
```

这个选项会显示在CMake的GUI，并且其默认值为ON。当用户选择了之后，这个值会被保存在CACHE中，这样就不需要每次CMAKE都进行更改了。下面一步条件构建和链接MathFunctions库。为了达到这个目的，我们可以改变顶层的CMakeLists文件，使得其看起来像这样：



```
# 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})
```

这里使用了USE\_MYMATH来决定MathFunctions是否会被编译和使用。注意这里变量EXTRA\_LIBS的使用方法。这是保持一个大的项目看起来比较简洁的一个方法。源代码中相应的变化就比较简单了：

```
// 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;
}
```

在源代码中我们同样使用了USE\_MYMATH这个宏。它由CMAKE通过配置文件TutorialConfig.h.in来提供给源代码。

```
#cmakedefine USE_MYMATH
```

## Installing and Testing (Step 3)

接下来我们会为我们的工程增加安装规则和测试支持。安装规则是非常非常简单的。对于MathFunctions库我们安装库和头文件只需要添加如下的语句：

```
install (TARGETS MathFunctions DESTINATION bin)
install (FILES MathFunctions.h DESTINATION include)
```

对于应用程序，我们只需要在顶层CMakeLists 文件中如此配置即可以安装可执行程序 and 配置了的头文件：



```
# add the install targets
install (TARGETS Tutorial DESTINATION bin)
install (FILES "${PROJECT_BINARY_DIR}/TutorialConfig.h"
         DESTINATION include)
```

这就是所有需要做的。现在你就可以编译这个教程了，然后输入make install（或者编译IDE中的INSTALL目标），则头文件、库和可执行程序等就会被正确地安装。CMake变量CMAKE\_INSTALL\_PREFIX被用来决定那些文件会被安装在哪个根目录下。添加测试也是一个相当简单的过程。在最顶层的CMakeLists文件的最后我们可以添加一系列的基礎测试来确认这个程序是否在正确工作。

```
# 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")
```

第一个测试简单地确认应用是否运行，没有段错误或者其它的崩溃问题，并且返回0。这是CTest的最基本的形式。下面的测试都使用了PASS\_REGULAR\_EXPRESSION测试属性来确认输出的结果中是否含有某个字符串。如果你需要添加大量的测试来判断不同的输入值，则你需要考虑创建一个类似于下面的宏：

```
#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")
```

对do\_test的任意一次调用，就有另一个测试被添加到工程中。

## Adding System Introspection (Step 4)

接下来，我们来考虑添加一些有些目标平台可能不支持的代码。在这个样例中，我们将根据目标平台是否有log和exp函数来添加我们的代码。当然大多数平台都是有这些函数的，只是本教程假设这两个函数没有被那么普遍地支持。如果平台有log，那么在mysqrt中，就用它来计算平方根。我们首先使用CheckFunctionExists.cmake来测试这些函数的是否存在，在顶层的CMakeLists文件中：



```
# does this system provide the log and exp functions?
include (CheckFunctionExists.cmake)
check_function_exists (log HAVE_LOG)
check_function_exists (exp HAVE_EXP)
```

接下来我们修改TutorialConfig.h.in来定义CMake是否找到这些函数的宏

```
// does the platform provide exp and log functions?
#cmakedefine HAVE_LOG
#cmakedefine HAVE_EXP
```

重要的一点是，对tests和log的测试必须要在配置文件命令前完成。配置文件命令会使用CMake中的配置立马配置文件。最后在mysqrt函数中我们提供了两种实现方式：

```
// 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
    . . .
```

## Adding a Generated File and Generator (Step 5)

在这一节当中，我们会告诉你如何将一个生成的源文件加入到应用程序的构建过程中。在此例中，我们会创建一个预先计算好的平方根的表，并将这个表编译到应用程序中去。为了达到这个目的，我们首先需要有一个程序来生成这样的表。在MathFunctions这个子目录下一个新的叫做MakeTable.cxx的源文件就是用来干这个的。

```
// 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;
}
```



注意到这张表是由一个有效的C++代码产生的，并且输出文件的名字是由参数代入的。下一步就是添加合适的命令到MathFunctions的CMakeLists文件中来构建MakeTable这个可执行程序，并且作为构建过程中的一部分。完成它需要一些命令，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
  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，就像其它可执行程序一样，MakeTable被添加为可执行程序。然后我们添加了一个自定义命令来详细描述如何通过运行MakeTable来产生Table.h。接下来，我们需要让CMake知道mysqrt.cxx依赖于生成的文件Table.h。这是通过往MathFunctions这个库里面添加生成的Table.h来实现的。我们也需要添加当前的生成目录到搜索路径中，从而Table.h可以被mysqrt.cxx找到。

当这个工程被构建时，它首先会构建MakeTable这个可执行程序。然后运行MakeTable从而生成Table.h。最后，它会编译mysqrt.cxx来生成MathFunctions library。

在这一刻，我们添加了所有的特征到最顶层的CMakeLists文件，它现在看起来是这样的：



```

cmake_minimum_required (VERSION 2.6)
project (Tutorial)

# 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是这样的：





```
// the configured options and settings for Tutorial
#define Tutorial_VERSION_MAJOR @Tutorial_VERSION_MAJOR@
#define Tutorial_VERSION_MINOR @Tutorial_VERSION_MINOR@
#define USE_MYMATH

// does the platform provide exp and log functions?
#define HAVE_LOG
#define HAVE_EXP
```

最后MathFunctions的CMakeLists文件看起来是这样的：

```
# 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)
```

## Building an Installer (Step 6)

最后假设我们想要把我们的工程发布给别人从而让他们去使用。我们想要同时给他们不同平台的二进制文件和源代码。这与步骤3中的install略有不同，install是安装我们从源代码中构建的二进制文件。而在此例中，我们将要构建安装包来支持二进制安装以及cygwin，debian，RPMs等的包管理特性。为了达到这个目的，我们会使用CPack来创建平台相关的安装包。具体地说，我们需要在顶层CMakeLists.txt文件中的底部添加数行。

```
# 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 "${Tutorial_VERSION_MINOR}")
set (CPACK_PACKAGE_VERSION_PATCH "${Tutorial_VERSION_PATCH}")
include (CPack)
```

这就是所有要做的。我们首先包含了InstallRequiredSystemLibraries。这个模块将会包含当前平台所需要的所有运行时库。接下来，我们设置了一些CPack的变量来保存license以及工程的版本信息。版本信息利用了我们在早前的教程中使用到的变量。最后我们包含了CPack这个模块来使用这些变量和你所使用的系统的其它特性来设置安装包。

接下来一步是用通常的方式构建工程，然后在CPack上运行它。如果要构建一个二进制包你需要运行：

```
cpack --config CPackConfig.cmake
```

如果要创建一个源代码包你需要输入

```
cpack --config CPackSourceConfig.cmake
```



Adding Support for a Dashboard (Step 7)

将测试结果上传到dashboard上是非常简单的。我们在早前的步骤中已经定义了一些测试。我们仅需要运行这些例程然后提交到dashboard上。为了包含对dashboards的支持，我们需要在顶层的CMakeLists文件中包含CTest模块。

```
# enable dashboard scripting
include (CTest)
```

我们也创建了一个CTestConfig.cmake文件来指定这个工程在dashobard上的的名字。

```
set (CTEST_PROJECT_NAME "Tutorial")
```

CTest会读这个文件并且运行它。如果要创建一个简单的dashboard，你可以在你的工程上运行CMake，改变生成路径的目录，然后运行ctest -D Experimental。你的dashboard的结果会被上传到Kitware的公共dashboard (https://open.cdash.org/index.php?project=PublicDashboard)中。

最后，可以在这里 (http://oawflafkv.bkt.clouddn.com/xz/code/Tutorial.rar)下载到整个教程的源代码。

编程小事 (nbn9897794) 举报文章 © 著作权归作者所有



行之与亦安 (/u/849a39246ef1)


写了 44484 字，被 134 人关注，获得了 172 个喜欢 (/u/849a39246ef1)

+ 关注

大家好，这里是行之(http://xingzhi.space)与亦安(http://yi-an.space)共同管理的空间，主要发布工科方面的...

您的打赏不是我创作的动力，而是我买买买的动力！^\_^

赞赏支持

 喜欢 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-like-button) | 15




更多分享

(http://cwb.assets.jianshu.io/notes/images/5847779/m



登录后发表评论 (/sign\_in?utm\_source=desktop&utm\_medium=not-signed-in-comment-form)

5条评论 只看作者 按喜欢排序 按时间正序 按时间倒序



谢小帅 (/u/610a0a6eec3b)

2楼 · 2017.07.17 09:51 (/u/610a0a6eec3b)

写的很详细，如果能再讲下 find\_package 就好了

👍 赞

💬 回复



行之与亦安 (/u/849a39246ef1) :

@谢小帅 (/users/610a0a6eec3b) 可以看官方文档呀

2017.07.17 13:12

回复

谢小帅 (/u/610a0a6eec3b) :

@行之与亦安 (/users/849a39246ef1) 正是因为阁下的文章写的太好了，所以养成了依赖心理😂

2017.07.17 13:14

回复

行之与亦安 (/u/849a39246ef1) :

@谢小帅 (/users/610a0a6eec3b) 也不打赏一下👍







2017.07.18 11:46

回复

添加新评论

还有1条评论，展开查看

被以下专题收入，发现更多相似内容

-  程序员 (/c/NEt52a?utm\_source=desktop&utm\_medium=notes-included-collection)
-  开发 (/c/eeef34c46b03?utm\_source=desktop&utm\_medium=notes-included-collection)
-  @IT·互联网 (/c/V2CqjW?utm\_source=desktop&utm\_medium=notes-included-collection)
-  工具癖 (/c/2mvgxp?utm\_source=desktop&utm\_medium=notes-included-collection)
-  计算机技术一锅炖 (/c/007ab4b9baf5?utm\_source=desktop&utm\_medium=notes-included-collection)
-  收藏夹 (/c/d61b7ccabb9b?utm\_source=desktop&utm\_medium=notes-included-collection)

