SirDigit

CMake 用法导览

Preface：本文是CMake官方文档CMake Tutorial (http://www.cmake.org/cmake/help/cmake_tutorial.html) 的翻译。通过一个样例工程从简单到复杂的完善过程，文档介绍了CMake主要模块（cmake, ctest, cpack）的功能和使用环境；从中可以一窥cmake的大体形貌。正文如下：

本文下述内容是一个手把手的使用指南；它涵盖了CMake需要解决的公共构建系统的一些问题。这些主题中的许多主题已经在Mastering CMake一书中以单独的章节被介绍过，但是通过一个样例工程看一看它们如何工作也是非常有帮助的。本指南可以在CMake源码树的Tests/Tutorial路径下找到。每一步都有它自己的子路径，其中包含该步骤的一个完整的指南。

作为基础的起始点（步骤**1**）

最基本的工程是一个从源代码文件中构建可执行文件的例子。对于简单工程，只要一个两行的CMakeLists文件就足够了。这将会作为我们指南的起点。这份CMakeLists文件看起来像是这样：

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

注意到这个例子在CMakeLists文件中使用了小写。CMake支持大写、小写、混合大小写的命令。tutorial.cxx中的源代码用来计算一个数的平方根，并且它的第一版非常简单，如下所示：

```
// A simple program that computes the square root of a number
// 计算一个数的平方根的简单程序
#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;
}
```

我们添加的第一个特性用来为工程和可执行文件指定一个版本号。虽然你可以在源代码中唯一指定它，但是你在CMakeLists文件中指定它可以提供更好的灵活性。如下所示，我么可以通过添加一个版本号来修改CMakeLists文件：

```
cmake_minimum_required (VERSION 2.6)
project (Tutorial)
# 版本号
set (Tutorial_VERSION_MAJOR 1)
```

导航

博客园 首页 联系 订阅 管理

```
set (Tutorial_VERSION_MINOR 0)
# 配置一个头文件，通过它向源代码中传递一些CMake设置。
configure_file (
  "${PROJECT_SOURCE_DIR}/TutorialConfig.h.in"
  "${PROJECT_BINARY_DIR}/TutorialConfig.h"
  )
# 将二进制文件树添加到包含文件的搜索路径中，这样我们可以找到TutorialConfig.h
include_directories("${PROJECT_BINARY_DIR}")
# 添加可执行文件
add_executable(Tutorial tutorial.cxx)
```

由于配置过的文件将会被写到二进制文件目录下，我们必须把该目录添加到包含文件的搜索路径清单中。然后，以下的代码就可以在源目录下创建一份TotorialConfig.h.in文件：

```
// 与tutorial相关的配置好的选项与设置；
#define Tutorial_VERSION_MAJOR @Tutorial_VERSION_MAJOR@
#define Tutorial_VERSION_MINOR @Tutorial_VERSION_MINOR@
```

当CMake配置这份头文件时，@Tutorial_VERSION_MAJOR@和@Tutorial_VERSION_MINOR@的值将会被从CMakeLists文件中传递过来的值替代。下一步，我们要修改tutorial.cxx来包含configured头文件然后使用其中的版本号。修改过的源代码展列于下：

```
// 计算平方根的简单程序。
#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;
}
```

引入库（步骤**2**）

现在我们将会在我们的工程中引入一个库。这个库会包含我们自己实现的计算一个数的平方根的函数。可执行文件随后可以使用这个库文件而不是编译器提供的标准开平方函数。在本指南中，我们将会把库文件放到一个子目录MathFunctions中。它包含下述的单行CMakeLists文件：

2014年1月 (3)
2013年1月 (1)
2012年12月 (1)
2012年10月 (2)
2012年9月 (1)
2012年8月 (2)
2012年7月 (11)
2012年6月 (7)

编程第一线

什么是全栈工程师

全栈工程师是熟悉软件的每个层次（虽然未必精通很多），并且对所有软件技术怀有纯粹的兴趣的那些人。

产品点滴

QQ登陆封面背后

程序员故事

你是一名努力工作的程序员，还是懒惰的程序员？

这家有线电视公司是一个非常少见的实验室，你能够对好的软件设计和坏的软件设计、好的团队行为和坏的团队行为之间的效果有一个直观的比较。

重要参考资源

C++11 FAQ中文版

Bejoney的C++11FAQ的中文版翻译。

最新评论

1. Re:CMake 手册详解（六）

我现在有这么个问题，如果我使用add_subdirectory加入一个子目录，在这个子目录中我生成了一个动态库，这个动态库又用add_subdirectory添加了子目录(子目录生成了静态库)， 那么.....

--1978577456

2. Re:CMake手册详解（十四）

多谢整理翻译。想请教一下，find_path的VAR变量会被存到cache里面，只有变量被清除，才会重新搜索该文件（或目录）是否存在，那么这个cache里面的变量什么情况下才能被清除？有其他comma.....

--慕黎笙丶

3. Re:CMake 用法导览

感谢分享。就是有些跳跃，看的不是很懂。

--seancheer

4. Re:CMake 手册详解（四）

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

源文件mysqrt.cxx有一个叫做mysqrt的函数，它提供了与编译器的sqrt函数类似的功能。为了使用新的库，我们在顶层的CMakeLists中增加一个add_subrirectory调用，这样这个库也会被构建。我们也要向可执行文件中增加另一个头文件路径，这样就可以从MathFunctions/mysqrt.h头文件中找到函数的原型。最后的一点更改是在向可执行文件中引入新的库。顶层CMakeLists文件的最后几行现在看起来像是这样：

```
include_directories ("${PROJECT_SOURCE_DIR}/MathFunctions")
add_subdirectory (MathFunctions)
# 引入可执行文件
add_executable (Tutorial tutorial.cxx)
target_link_libraries (Tutorial MathFunctions)
```

现在，让我们考虑下让MathFunctions库变为可选的。在本指南中，确实没有必要这样画蛇添足；但是对于更大型的库或者依赖于第三方代码的库，你可能需要这种可选择性。第一步是为顶层的CMakeLists文件添加一个选项：

```
# 我们应该使用我们自己的数学函数吗？
option (USE_MYMATH
        "Use tutorial provided math implementation" ON)
```

这将会在CMake的GUI中显示一个默认的ON值，并且用户可以随需改变这个设置。这个设置会被存储在cache中，那么用户将不需要在cmake该工程时，每次都设置这个选项。第二处改变是，让链接MathFunctions库变为可选的。要实现这一点，我们修改顶层CMakeLists文件的结尾部分：

```
# 添加MathFunctions库吗？
if (USE_MYMATH)
  include_directories ("${PROJECT_SOURCE_DIR}/MathFunctions")
  add_subdirectory (MathFunctions)
  set (EXTRA_LIBS ${EXTRA_LIBS} MathFunctions)
endif (USE_MYMATH)
# 添加可执行文件
add_executable (Tutorial tutorial.cxx)
target_link_libraries (Tutorial  ${EXTRA_LIBS})
```

这里用USE_MYMATH设置来决定是否MathFunctions应该被编译和执行。注意到，要用一个变量（在这里是EXTRA_LIBS）来收集所有以后会被连接到可执行文件中的可选的库。这是保持带有许多可选部件的较大型工程干净清爽的一种通用的方法。源代码对应的改变相当直白，如下所示：

```
// 计算一个数平方根的简单程序
#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,
```

不知道楼主还有没有在。。。我是菜鸟 最近在用cmake编译PCL1.8.0 目前遇到一个问题卡住了 我先贴一下错误代码吧：CMake Error at D:/PCL_Program/Codes/.....

--Amoyer

5. Re:CMake 用法导览

感谢分享！

--Daringoo

```
            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，我们也使用了USE_MYMATH。这个宏是由CMake通过TutorialConfig.h.in配置文件中的下述语句行提供给源代码的：

```
+ View Code
```

安装与测试（步骤**3**）

下一步我们会为我们的工程引入安装规则以及测试支持。安装规则相当直白，对于MathFunctions库，我们通过向MathFunctions的CMakeLists文件添加如下两条语句来设置要安装的库以及头文件：

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

对于应用程序，在顶层CMakeLists文件中添加下面几行，它们用来安装可执行文件以及配置头文件：

```
# 添加安装目标
install (TARGETS Tutorial DESTINATION bin)
install (FILES "${PROJECT_BINARY_DIR}/TutorialConfig.h"
         DESTINATION include)
```

这就是要做的全部；现在你应该可以构建tutorial工程了。然后，敲入命令make install（或者从IDE中构建INSTALL目标）然后它就会安装需要的头文件，库以及可执行文件CMake的变量CMAKE_INSTALL_PREFIX用来确定这些文件被安装的根目录。添加测试同样也只需要相当浅显的过程。在顶层CMakeLists文件的的尾部补充许多基本的测试代码来确认应用程序可以正确工作。

```
+ View Code
```

第一个测试用例仅仅用来验证程序可以运行，没有出现段错误或其他的崩溃，并且返回值必须是0。这是CTest所做测试的基本格式。余下的几个测试都是用 PASS_REGULAR_EXPRESSION 测试属性来验证测试代码的输出是否包含有特定的字符串。在本例中，测试样例用来验证计算得出的平方根与预定值一样；当指定错误的输入数据时，要打印用法信息。如果你想要添加许多测试不同输入值的样例，你应该考虑创建如下所示的宏：

```
+ View Code
```

对于每个do_test宏调用，都会向工程中添加一个新的测试用例；宏参数是测试名、函数的输入以及期望结果。

增加系统内省（步骤**4**）

下一步，让我们考虑向我们的工程中引入一些依赖于目标平台上可能不具备的特性的代码。在本例中，我们会增加一些依赖于目标平台是否有log或exp函数的代码。当然，几乎每个平台都有这些函数；但是对于tutorial工程，我们假设它们并非如此普遍。如果该平台有log函数，那么我们会在mysqrt函数中使用它去计算平方根。我们首先在顶层CMakeLists文件中使用宏CheckFunctionExists.cmake测试这些函数的可用性：

```
+ View Code
```

下一步，如果CMake在对应平台上找到了它们，我们修改TutorialConfig.h.in来定义这些值；如下：

```
// 该平台提供exp和log函数吗？
#cmakedefine HAVE_LOG
#cmakedefine HAVE_EXP
```

这些log和exp函数的测试要在TutorialConfig.h的configure_file命令之前被处理，这一点很重要。最后，在mysqrt函数中，如果log和exp在当前系统上可用的话，我们可以提供一个基于它们的可选的实现：

```
1  // 如果我们有log和exp两个函数，那么使用它们<br>#if defined (HAVE_LOG) && defined (HAVE_EXP)
2    result = exp(log(x)*0.5);
3  #else // 否则使用替代方法
```

添加一个生成文件以及生成器（步骤**5**）

在本节，我们会展示你应该怎样向一个应用程序的构建过程中添加一个生成的源文件。在本范例中，我们会创建一个预先计算出的平方根表作为构建过程的一部分。MathFunctions子路径下，一个新的MakeTable.cxx源文件来做这件事。

```
1  // 一个简单的用于构建平方根表的程序
2  #include <stdio.h>
3  #include <stdlib.h><br>#include <math.h>
4
5  int main (int argc, char *argv[])
6  {
7    int i;
8    double result;
9
10   // 确保有足够多的参数
11   if (argc < 2)
12     {
13     return 1;
14     }
15
16   // 打开输出文件
17   FILE *fout = fopen(argv[1],"w");
18   if (!fout)
19     {
20     return 1;
21     }
22
23   // 创建一个带有平方根表的源文件<br>  fprintf(fout,"double sqrtTable[] = {\n");
24   for (i = 0; i < 10; ++i)
25     {
```

```
    result = sqrt(static_cast<double>(i));
    fprintf(fout,"%g,\n",result);
    }

  // 该表以0结尾
  fprintf(fout,"0};\n");
  fclose(fout);
  return 0;
}
```

注意到这个表是由合法的C++代码生成的，并且被写入的输出文件的名字是作为一个参数输入的。下一步是将合适的命令添加到MathFunction的CMakeLists文件中，来构建MakeTable可执行文件，然后运行它，作为构建过程的一部分。完成这几步，需要少数的几个命令，如下所示：

```
# 首先，我们添加生成该表的可执行文件<br>add_executable(MakeTable MakeTable.cxx)
# 然后添加该命令来生成源文件
add_custom_command (
  OUTPUT ${CMAKE_CURRENT_BINARY_DIR}/Table.h
  COMMAND MakeTable ${CMAKE_CURRENT_BINARY_DIR}/Table.h
  DEPENDS MakeTable
  )

# 为包含文件，向搜索路径中添加二进制树路径
include_directories( ${CMAKE_CURRENT_BINARY_DIR} )
<br># 添加main库
add_library(MathFunctions mysqrt.cxx ${CMAKE_CURRENT_BINARY_DIR}/Table.h  )
```

首先，MakeTable的可执行文件也和其他被加入的文件一样被加入。然后，我们添加一个自定义命令来指定如何通过运行MakeTable来生成Table.h。这是通过将生成Table.h增加到MathFunctions库的源文件列表中来实现的。我们还必须增加当前的二进制路径到包含路径的清单中，这样Table.h可以被找到并且可以被mysqrt.cxx所包含。当该工程被构建后，它首先会构建MakeTable可执行文件。然后它会运行MakeTable来生成Table.h文件。最后，它会编译mysqrt.cxx（其中包含Table.h）来生成MathFunctions库。到目前为止，拥有我们添加的完整特性的顶层CMakeLists文件看起来像是这样：

```
cmake_minimum_required (VERSION 2.6)
project (Tutorial)

# 版本号
set (Tutorial_VERSION_MAJOR 1)
set (Tutorial_VERSION_MINOR 0)

# 本系统是否提供log和exp函数?
include (${CMAKE_ROOT}/Modules/CheckFunctionExists.cmake)

check_function_exists (log HAVE_LOG)
check_function_exists (exp HAVE_EXP)

# 我们应该使用自己的math函数吗?
option(USE_MYMATH
  "Use tutorial provided math implementation" ON)

```

```
# 配置一个头文件来向源代码传递一些CMake设置。
configure_file (
  "${PROJECT_SOURCE_DIR}/TutorialConfig.h.in"
  "${PROJECT_BINARY_DIR}/TutorialConfig.h"
  )

# 为包含文件的搜索路径添加二进制树，这样才能发现TutorialConfig.h头文件。
include_directories ("${PROJECT_BINARY_DIR}")

# 添加MathFunctions库吗?
if (USE_MYMATH)
  include_directories ("${PROJECT_SOURCE_DIR}/MathFunctions")
  add_subdirectory (MathFunctions)
  set (EXTRA_LIBS ${EXTRA_LIBS} MathFunctions)
endif (USE_MYMATH)

# 添加可执行文件
add_executable (Tutorial tutorial.cxx)
target_link_libraries (Tutorial  ${EXTRA_LIBS})

# 添加安装的目标
install (TARGETS Tutorial DESTINATION bin)
install (FILES "${PROJECT_BINARY_DIR}/TutorialConfig.h"
         DESTINATION include)

# 测试1 ： 应用程序可以运行吗?
add_test (TutorialRuns Tutorial 25)

# 测试2 ：  使用信息可用吗?
add_test (TutorialUsage Tutorial)
set_tests_properties (TutorialUsage
  PROPERTIES
  PASS_REGULAR_EXPRESSION "Usage:.*number"
  )

# 定义一个可以简化引入测试过程的宏
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
# 执行一系列基于结果的测试
do_test (4 "4 is 2")
```

```
do_test (9 "9 is 3")
do_test (5 "5 is 2.236")<br>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文件看起来像是这样：

```
// Tutorial的配置选项与设置如下
#define Tutorial_VERSION_MAJOR @Tutorial_VERSION_MAJOR@
#define Tutorial_VERSION_MINOR @Tutorial_VERSION_MINOR@
#cmakedefine USE_MYMATH

// 该平台提供exp和log函数吗?
#cmakedefine HAVE_LOG
#cmakedefine HAVE_EXP
```

然后，MathFunctions工程的CMakeLists文件看起来像是这样：

```
+ View Code
```

构建一个安装器（步骤**6**）

下一步假设我们想要向其他人分发我们的工程，这样他们就可以使用它。我们想同时提供在许多不同平台上的源代码和二进制文档发行版。这与之前我们在"安装与测试（步骤3）"做过的安装有一点不同，那里我们仅仅安装我们从源码中构建出来的二进制文件。在本例子中，我们会构建支持二进制安装以及类似于cygwin，debian，RPM等具有包管理特性的安装包。为了完成这个目标，我们会使用CPack来创建Packaging with CPack一章中描述的特定平台的安装器。

```
# 构建一个CPack驱动的安装包
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)
```

需要做的全部事情就这些。我们以包含InstallRequiredSystemLibraries开始。这个模块将会包含许多在当前平台上，当前工程需要的运行时库。第一步我们将一些CPack变量设置为保存本工程的许可证和版本信息的位置。版本信息使用了我们在本指南中先前设置的变量。最后，我们要包含CPack模块，它会使用这些变量以及你所处的系统的一些别的属性，然后来设置一个安装器。下一步是以通常的方式构建该工程然后随后运行CPack。如果要构建一个二进制发行包，你应该运行：

```
cpack -C CPackConfig.cmake
```

为了创建一个源代码发行版，你应该键入：

```
cpack -C CPackSourceConfig.cmake
```

增加对**Dashboard**的支持（步骤**7**）

增加对向一个dashboard提交我们的测试结果的功能的支持非常简单。我们在本指南的先前步骤中已经定义了我们工程中的许多测试样例。我们仅仅需要运行这些测试样例然后将它们提交到dashboard即可。为了包含对dashboards的支持，我们需要在顶层CMakeLists文件中包含CTest模块。

```
# 支持dashboard脚本
include (CTest)
```

我们也可以创建一个CTestConfig.cmake文件，在其中来指定该dashboard的工程名。

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

CTest 将会在运行期间读取这个文件。为了创建一个简单的dashboard，你可以在你的工程下运行CMake，然后切换到二进制树，然后运行ctest -DExperimental. 你的dashboard将会被更新到Kitware的公共dashboard.

<<<---------- 欢迎转载；转载请标明出处。 ---------->>>

标签: CMake 导览

SirDigit
关注 - 18
粉丝 - 65
+加关注

« 上一篇：CMake 手册详解（二十三）
» 下一篇：C++11新特性——The C++ standard library, 2nd Edition 笔记（一）

posted on 2013-01-20 15:39 SirDigit 阅读(30318) 评论(3) 编辑 收藏

评论

#1楼 2013-05-24 12:17 puuuuyan

真是非常的参考文档，，非常感谢

支持(0) 反对(0)

#2楼 2015-05-05 09:35 Daringoo

感谢分享！

支持(0) 反对(0)

#3楼 2016-03-29 15:52 seancheer

感谢分享。就是有些跳跃，看的不是很懂。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部