导航

C++博客
首页
新随笔
联系
XML聚合
管理

< 2011年11月 >

日	一	二	三	四	五	六
30	31	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	1	2	3
4	5	6	7	8	9	10

常用链接

我的随笔
我的评论
我参与的随笔
留言簿(1)

给我留言
查看公开留言
查看私人留言
随笔档案(10)

2012年11月 (1)
2011年11月 (3)
2011年10月 (1)
2011年9月 (1)
2011年8月 (2)
2011年6月 (1)
2011年5月 (1)
文章分类(2)

c++(1)(rss)
Visual Studio 使用(1)(rss)
搜索技术相关(rss)
文章档案(1)

2011年5月 (1)
搜索

搜索
最新随笔

1. WeiboHotGather Download
2. Cmake的介绍和使用 Cmake实践
3. Makefile 解析
4. C++编译器 GCC G++ 使用
5. 详尽的Mysql安装过程 讲解
最新评论 XML

1. re: Cmake的介绍和使用 Cmake实践
感谢分享
--cx
2. re: Cmake的介绍和使用 Cmake实践
请问如何添加COPYRIGHT、README、和run.sh，是添加三个空文件么？还是说run.sh是需要内容的？
--smellflower
3. re: C++编译器 GCC G++ 使用
Nice work!
--tangb4c
4. re: select 和pselect的区别 [未登录]
这超过来超过去TTMD的 没意思
--小石

Cmake的介绍和使用 Cmake实践

Cmake优点：

1. 开发源代码，实用类BSD许可发布。
2. 跨平台，并可以生成native编译配置文件，在linux/unix平台，生成makefile，在mac平台可以生成xcode，在windows平台可以生成msvc工程的配置文件。
3. 能够管理大型项目
4. 简化编译构建过程和编译过程，只需要cmake+make就可以
5. 高效率
6. 可扩展，可以为cmake编写特定功能的模块，扩充cmake功能

如何安装cmake

1. Cmake的安装可以使用autotools进行安装，点击cmake-2.8.6.tar.gz 链接，可以对软件进行下载。
2. ./configure
3. make
4. sudo make install

Cmake的原理

Helloworld cmake

//main.cpp

```
#include<cstdio>


int main()
{
    printf("hello world from main\n");
    return 0;
}
```

创建CMakeLists.txt（注意大小写一个字母都不能错）

向该文件中加入以下几行（稍后会做解释）

```
PROJECT (HELLO)
SET(SRC_LIST main.cpp)
MESSAGE(STATUS "This is BINARY dir " ${HELLO_BINARY_DIR})
MESSAGE(STATUS "This is SOURCE dir "${HELLO_SOURCE_DIR})
ADD_EXECUTABLE(hello ${SRC_LIST})
```

运行以下命令：

cmake . (别忘记加上这个点，表示当前目录)

```
[chenjl@Processor006 cmake_demo]$ cmake .
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- This is BINARY dir /home/chenjl/Demo/cmake_demo
-- This is SOURCE dir /home/chenjl/Demo/cmake_demo
-- Configuring done
-- Generating done
-- Build files have been written to: /home/chenjl/Demo/cmake_demo
```

注意执行完这句话之后会生成几个文件如下：

```
[chenjl@Processor006 cmake_demo]$ ls
CMakeCache.txt  CMakeFiles  CMakeLists.txt  Makefile  cmake_install.cmake  main.cpp
```

CMakeFiles, CMakeCache.txt, cmake_install.cmake等文件，并且生成了Makefile

然后执行make 就可以生成可执行文件hello

```
[chenjl@Processor006 cmake_demo]$ make
Scanning dependencies of target hello
[100%] Building CXX object CMakeFiles/hello.dir/main.cpp.o
Linking CXX executable hello
```

这是当前目录下就会生成可执行文件如下图：

```
[chenjl@Processor006 cmake_demo]$ ls
CMakeCache.txt  CMakeFiles  CMakeLists.txt  Makefile  cmake_install.cmake  hello  main.cpp
[chenjl@Processor006 cmake_demo]$
```

对例子的解释：

CMakeLists.txt的内容如下：

```
PROJECT (HELLO)

SET(SRC_LIST main.cpp)

MESSAGE(STATUS "This is BINARY dir " ${HELLO_BINARY_DIR})

MESSAGE(STATUS "This is SOURCE dir "${HELLO_SOURCE_DIR})

ADD_EXECUTABLE(hello ${SRC_LIST})
```

Project的指令的语法是：

```
PROJECT(projectname [CXX] [C] [JAVA])
```

这个执行是用来定义工程的名称的和定义工程支持的语言。这个指令也隐式的定义了两个cmake变量：<projectname>_BINARY_DIR以及<projectname>_BINARY_DIR,这里就是HELLO_BINARY_DIR和HELLO_SOURCE_DIR，两个变量指的都是当前工程的路径。

SET指令的语法：

```
SET（VAR[VALUE] [CACHE TYPE DOCSTRING [FORCE]]）
```

Set指令是用来显式的定义变量的，我们之前用到的是SET(SRC_LIST main.cpp)如果有多个源文件，也可以定义成SET(SRC_LIST main.cpp t1.cpp t2.cpp)。

MESSAGE指令的语法是：

```
MESSAGE([SEND_ERROR | STATUS | FATAL_ERROR] "message to display" ...)
```

这个指令用于向终端输出用户信息，包含三种类型：

SEND_ERROR，产生错误，生成过程被跳过。

SATUS,输出前缀为-的信息。

FATAL_ERROR，立即终止所有cmake过程。

我们在这里使用的是STATUS信息输出，显示了由PROJECT指令项一项两个饮食变量HELLO_BINARY_DIR和HELLO_SOURCE_DIR。

```
ADD_EXECUTABLE(hello ${SRC_LIST})
```

定义了这个工程会生成一个文件名为hello的可执行文件，相关的源文件是SRC_LIST中定义的源文件列表，本例中你可以直接写成ADD_EXECUTABLE(hello main.c)。

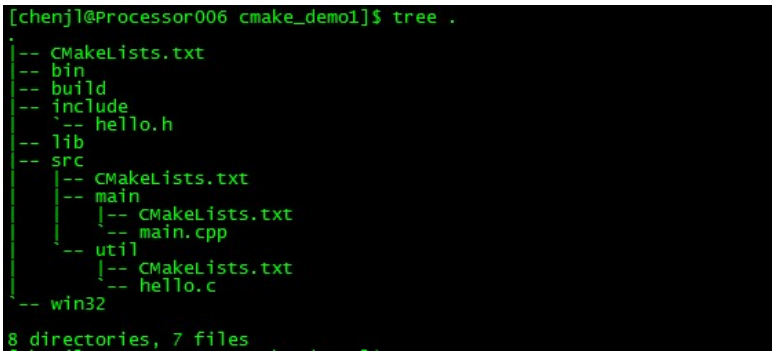
将本例改写成一个最简化的CMakeLists.txt：

```
PROJECT(HELLO)

ADD_EXECUTABLE(hello main.c)
```

下面我们介绍一个比较实用的例子，即包含生成静态库又包含引入外部头文件和链接库的cmake demo。

先按照工程规范建立工程目录，并编写代码，以下面的工程目录为例进行解释这个例子,工程的目录结构为：



编译工程要实现的目标：

1. 添加子目录doc，用以放置这个工程的文档hello.txt
2. 生成hello的静态库，并在main可执行程序链接hello静态库
3. 在这个工程中添加COPYRIGHT,README
4. 在工程目录中添加一个run.sh的脚本，用以调用生成的二进制可执行文件
5. 将生成的二进制文件生成到bin子目录中
6. 编写安装程序

1. 编写CMakeLists.txt

   由于一个工程目录中包含多个项目，其中在此项目中包含util项目和main项目，其中util项目是用以生成main程序需要的静态库，main是用以生成可执行文件。

   在工程项目中的父目录向有一个CMakeLists.txt是用以声明定义工程需要的Cmake设置还定义了子目录src，用以递归的调用src中的MakeLists.txt。其中工程目录的CMakeLists.txt内容定义如下：

   ```
   PROJECT(HELLO)

   ADD_SUBDIRECTORY(src)
   ```

   在src里面的CMakeLists.txt是用以定义src目录包含的两个工程的依赖关系分别进行编译。

   util目录里面的CMakeLists.txt是用以定义生成util静态库的规则，其中内容如下：

   ```
   SET(LIBRARY_OUTPUT_PATH ${HELLO_SOURCE_DIR}/lib)

   SET(CMAKE_C_COMPILER g++)

   SET(SRC_LIST hello.c)


   INCLUDE_DIRECTORIES(${HELLO_SOURCE_DIR}/include)

   ADD_LIBRARY(util STATIC ${SRC_LIST})
   ```

   其中SET(LIBRARY_OUTPUT_PATH ${HELLO_SOURCE_DIR}/lib)定义了库生成的路径，LIBRARY_OUTPUT_PATH是一个内部变量，存放库生成路径。

   SET(SRC_LIST hello.c)是用来定义库文件需要的源文件。

   INCLUDE_DIRECTORIES(${HELLO_SOURCE_DIR}/include)是用来定义非标准库头文件要搜索的路径。其中INCLUDE_DIRECTORIES命令的格式为：

   ```
   INCLUDE_DIRECTORIES([AFTER|BEFORE] [SYSTEM] dir1 dir2 ...)
   ```

   ADD_LIBRARY(util STATIC ${SRC_LIST})是用来定义生成的库的名字，以及生成库的类型和生成库需要的源文件，其中ADD_LIBRARY命令格式为：

```
ADD_LIBRARY(libname [SHARED|STATIC|MODULE]
[EXCLUDE_FROM_ALL]
source1 source2 ... sourceN)
```

SET(CMAKE_C_COMPILER g++)是用来定义c的编译器为g++，防止出现C和C++代码在不指定C编译器的情况下默认使用gcc，导致系统编译混乱。

在main目录中的CMakeLists.txt是用来定义可执行程序编译和链接时所需要的一些命令或环境。内容如下：

```
SET(EXECUTABLE_OUTPUT_PATH ${HELLO_SOURCE_DIR}/bin)

SET(SRC_LIST main.cpp)


INCLUDE_DIRECTORIES(${HELLO_SOURCE_DIR}/include)

LINK_DIRECTORIES(${HELLO_SOURCE_DIR}/lib)


ADD_EXECUTABLE(hello ${SRC_LIST})

TARGET_LINK_LIBRARIES(hello util)
```

INCLUDE_DIRECTORIES命令是定义工程的include文件夹，其中存放使用到的库的头文件，LINK_DIRECTORIES是定义工程的库文件，其中存放着库文件，ADD_EXECUTABLE是定义生成的可执行文件，
TARGET_LINK_LIBRARIES用以定义链接时需要的库文件。

2. 在工程目录下创建build目录，并采用out-of-source方式编译项目。执行命令make ..，执行结果如下：

```
[chenjl@Processor006 build]$ cmake ..
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc
-- Check for working C compiler: /usr/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
CMake Warning (dev) in CMakeLists.txt:
  No cmake_minimum_required command is present.  A line of code such as

    cmake_minimum_required(VERSION 2.8)

  should be added at the top of the file.  The version specified may be lower
  if you wish to support older CMake versions for this project.  For more
  information run "cmake --help-policy CMP0000".
This warning is for project developers.  Use -Wno-dev to suppress it.

-- Configuring done
-- Generating done
-- Build files have been written to: /home/chenjl/Demo/cmake_demo1/build
```

执行make，这时在build目录下生成了中间编译文件：

```
[chenjl@Processor006 build]$ tree .
.
|-- CMakeCache.txt
|-- CMakeFiles
|   |-- CMakeCCompiler.cmake
|   |-- CMakeCXXCompiler.cmake
|   |-- CMakeDetermineCompilerABI_C.bin
|   |-- CMakeDetermineCompilerABI_CXX.bin
|   |-- CMakeDirectoryInformation.cmake
|   |-- CMakeOutput.log
|   |-- CMakeSystem.cmake
|   |-- CMakeTmp
|   |   `-- CMakeFiles
|   |       `-- cmTryCompileExec.dir
|   |-- CompilerIdC
|   |   |-- CMakeCCompilerId.c
|   |   `-- a.out
|   |-- CompilerIdCXX
|   |   |-- CMakeCXXCompilerId.cpp
|   |   `-- a.out
|   |-- Makefile.cmake
|   |-- Makefile2
|   |-- TargetDirectories.txt
|   |-- cmake.check_cache
|   `-- progress.marks
|-- Makefile
|-- cmake_install.cmake
`-- src
    |-- CMakeFiles
    |   |-- CMakeDirectoryInformation.cmake
    |   `-- progress.marks
    |-- Makefile
    |-- cmake_install.cmake
    |-- main
    |   |-- CMakeFiles
    |   |   |-- CMakeDirectoryInformation.cmake
    |   |   |-- hello.dir
    |   |   |   |-- DependInfo.cmake
    |   |   |   |-- build.make
    |   |   |   |-- cmake_clean.cmake
    |   |   |   |-- depend.make
    |   |   |   |-- flags.make
    |   |   |   |-- link.txt
    |   |   |   `-- progress.make
    |   |   `-- progress.marks
    |   |-- Makefile
    |   `-- cmake_install.cmake
    `-- util
        |-- CMakeFiles
        |   |-- CMakeDirectoryInformation.cmake
        |   |-- progress.marks
        |   `-- util.dir
        |       |-- DependInfo.cmake
        |       |-- build.make
        |       |-- cmake_clean.cmake
        |       |-- cmake_clean_target.cmake
```

执行make命令，结果如下：

```
[chenjl@Processor006 build]$ make
Scanning dependencies of target util
[ 50%] Building C object src/util/CMakeFiles/util.dir/hello.o
In file included from /home/chenjl/Demo/cmake_demo1/src/util/hello.c:1:
/home/chenjl/Demo/cmake_demo1/include/hello.h:7:7: warning: no newline at end of file
/home/chenjl/Demo/cmake_demo1/src/util/hello.c:14:2: warning: no newline at end of file
Linking C static library ../../../lib/libutil.a
[ 50%] Built target util
Scanning dependencies of target hello
[100%] Building CXX object src/main/CMakeFiles/hello.dir/main.o
In file included from /home/chenjl/Demo/cmake_demo1/src/main/main.cpp:1:
/home/chenjl/Demo/cmake_demo1/include/hello.h:7:7: warning: no newline at end of file
/home/chenjl/Demo/cmake_demo1/src/main/main.cpp:9:2: warning: no newline at end of file
Linking CXX executable ../../../bin/hello
[100%] Built target hello
```

可以看到工程创建和编译成功了。

2. 安装

在工程目录下添加COPYRIGHT、README、和run.sh，重新编辑工程目录下的CMakeLists.txt。在CMakeLists.txt中添加如下命令：

INSTALL(FILES COPYRIGHT README DESTINATION share/doc/cmake_demo)

INSTALL(PROGRAMS run.sh DESTINATION bin)

INSTALL(PROGRAMS bin/hello DESTINATION bin)

INSTALL(DIRECTORY doc/ DESTINATION share/doc/cmake_demo)

这些命令表示在执行make install命令时，安装程序会拷贝相应的文件、目录或程序到指定的前缀开始的目录中，cmake执行命令如下：

cmake -DCMAKE_INSTALL_PREFIX=~/data/cmake_demo ..这时将工程目录安装到~/data/cmake_demo目录下。执行结果如下：

```
[chenjl@Processor006 build]$ cmake -DCMAKE_INSTALL_PREFIX=~/data/cmake_demo ..
CMake Warning (dev) in CMakeLists.txt:
  No cmake_minimum_required command is present.  A line of code such as

    cmake_minimum_required(VERSION 2.8)

  should be added at the top of the file.  The version specified may be lower
  if you wish to support older CMake versions for this project.  For more
  information run "cmake --help-policy CMP0000".
This warning is for project developers.  Use -Wno-dev to suppress it.

-- Configuring done
-- Generating done
-- Build files have been written to: /home/chenjl/Demo/cmake_demo1/build
[chenjl@Processor006 build]$ make
[ 50%] Built target util
[100%] Built target hello
[chenjl@Processor006 build]$ make install
[ 50%] Built target util
[100%] Built target hello
Install the project...
-- Install configuration: ""
-- Installing: /home/chenjl/data/cmake_demo/share/doc/cmake_demo/COPYRIGHT
-- Installing: /home/chenjl/data/cmake_demo/share/doc/cmake_demo/README
-- Installing: /home/chenjl/data/cmake_demo/bin/run.sh
-- Installing: /home/chenjl/data/cmake_demo/bin/hello
-- Installing: /home/chenjl/data/cmake_demo/share/doc/cmake_demo
-- Installing: /home/chenjl/data/cmake_demo/share/doc/cmake_demo/hello.txt
[chenjl@Processor006 build]$
```

其中cmake编译c、c++工程完毕。

posted on 2011-11-17 20:18 Roger 阅读(35817) 评论(2) 编辑 收藏 引用

评论

 # re: Cmake的介绍和使用 Cmake实践 回复 更多评论

请问如何添加COPYRIGHT、README、和run.sh，是添加三个空文件么？还是说run.sh是需要内容的？

2014-04-04 17:20 | smellflower

 # re: Cmake的介绍和使用 Cmake实践 回复 更多评论

感谢分享

2015-07-22 22:17 | cx

刷新评论列表

找优秀程序员，就在博客园

标题 re: Cmake的介绍和使用 Cmake实践

姓名

主页

验证码 * 1271

内容(提交失败后,可以通过“恢复上次提交”恢复刚刚提交的内容)

Remember Me?

提交 登录 使用高级评论 新用户注册 返回页首 恢复上次提交

[使用Ctrl+Enter键可以直接提交]

网站导航: 博客园 IT新闻 BlogJava 知识库 程序员招聘 管理