说明

CMake安装

CMake一个HelloWord

CMake一个HelloWord-的语法介绍

PROJECT关键字

SET关键字

MESSAGE关键字

ADD_EXECUTABLE关键字

语法的基本原则

语法注意事项

内部构建和外部构建

外部构建方式举例

让Hello World看起来更像一个工程

将目标文件放入构建目录的 bin 子目录 ADD_SUBDIRECTORY 指令 更改二进制的保存路径

安装

如何安装HelloWord

安装文件COPYRIGHT和README

安装脚本runhello.sh

安装 doc 中的 hello.txt

安装过程

静态库和动态库的构建

构建实例

ADD LIBRARY

同时构建静态和动态库

SET_TARGET_PROPERTIES

动态库的版本号

安装共享库和头文件

使用外部共享库和头文件

解决: make后头文件找不到的问题

解决:找到引用的函数问题

特殊的环境变量 CMAKE_INCLUDE_PATH 和 CMAKE_LIBRARY_PATH

本人所有视频和笔记都是免费分享给大家的,制作视频和笔记要花费大量的时间成本 我也有老婆和孩子要养,恳求各位观众老爷们有经济实力的稍微打赏一下小弟,但不强求,再一次感谢。

您的打赏、会让我今后有更大的动力、做出更优质的视频、感谢大家的支持

说明

cmake的定义是什么? -----高级编译配置工具

当多个人用不同的语言或者编译器开发一个项目,最终要输出一个可执行文件或者共享库(dll, so等等)这时候神器就出现了----CMake!

所有操作都是通过编译CMakeLists.txt来完成的一简单

官 方网站是 www.cmake.org,可以通过访问官方网站获得更多关于 cmake 的信息

学习CMake的目的,为将来处理大型的C/C++/JAVA项目做准备

CMake安装

- 1、绝大多数的linux系统已经安装了CMake
- 2、Windows或某些没有安装过的linux系统,去<u>http://www.cmake.org/HTML/Download.html</u> 可以下载安装

CMake一个HelloWord

1、步骤一,写一个HelloWord

```
#main.cpp

#include <iostream>

int main(){
    std::cout << "hello word" << std::endl;
}</pre>
```

2、步骤二,写CMakeLists.txt

```
#CMakeLists.txt

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})
```

3、步骤三、使用cmake, 生成makefile文件

```
m出:
[root@localhost cmake]# cmake .

CMake Warning (dev) in CMakeLists.txt:
Syntax Warning in cmake code at

/root/cmake/CMakeLists.txt:7:37

Argument not separated from preceding token by whitespace.
```

```
This warning is for project developers. Use -Wno-dev to suppress it.
-- The C compiler identification is GNU 10.2.1
-- The CXX compiler identification is GNU 10.2.1
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- 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 /root/cmake
-- This is SOURCE dir /root/cmake
-- Configuring done
-- Generating done
-- Build files have been written to: /root/cmake
```

目录下就生成了这些文件-CMakeFiles, CMakeCache.txt, cmake_install.cmake 等文件,并且生成了Makefile. 现在不需要理会这些文件的作用,以后你也可以不去理会。最关键的是,它自动生成了Makefile.

4、使用make命令编译

```
root@localhost cmake]# make
Scanning dependencies of target hello
[100%] Building CXX object CMakeFiles/hello.dir/main.cpp.o
Linking CXX executable hello
[100%] Built target hello
```

5、最终生成了Hello的可执行程序

CMake一个HelloWord-的语法介绍

PROJECT关键字

可以用来指定工程的名字和支持的语言,默认支持所有语言

PROJECT (HELLO) 指定了工程的名字,并且支持所有语言—建议

PROJECT (HELLO CXX) 指定了工程的名字,并且支持语言是C++

PROJECT (HELLO C CXX) 指定了工程的名字,并且支持语言是C和C++

该指定隐式定义了两个CMAKE的变量

_BINARY_DIR,本例中是 HELLO_BINARY_DIR

_SOURCE_DIR, 本例中是 HELLO_SOURCE_DIR

MESSAGE关键字就可以直接使用者两个变量、当前都指向当前的工作目录、后面会讲外部编译

问题: 如果改了工程名, 这两个变量名也会改变

解决:又定义两个预定义变量:PROJECT_BINARY_DIR和PROJECT_SOURCE_DIR,这两个变量和HELLO_BINARY_DIR,HELLO_SOURCE_DIR是一致的。所以改了工程名也没有关系

SET关键字

用来显示的指定变量的

SET(SRC_LIST main.cpp) SRC_LIST变量就包含了main.cpp

也可以 SET(SRC_LIST main.cpp t1.cpp t2.cpp)

MESSAGE关键字

向终端输出用户自定义的信息

主要包含三种信息:

- 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.

ADD_EXECUTABLE关键字

生成可执行文件

ADD_EXECUTABLE(hello \${SRC_LIST}) 生成的可执行文件名是hello,源文件读取变量SRC_LIST中的内容也可以直接写 ADD_EXECUTABLE(hello main.cpp)

上述例子可以简化的写成

PROJECT(HELLO) ADD_EXECUTABLE(hello main.cpp)

注意:工程名的 HELLO 和生成的可执行文件 hello 是没有任何关系的

语法的基本原则

- 变量使用\${}方式取值,但是在 IF 控制语句中是直接使用变量名
- 指令(参数 1 参数 2...) 参数使用括弧括起,参数之间使用空格或分号分开。 以上面的 ADD_EXECUTABLE 指令为例,如果存在另外一个 func.cpp 源文件

就要写成: ADD_EXECUTABLE(hello main.cpp func.cpp)或者ADD_EXECUTABLE(hello main.cpp;func.cpp)

• 指令是大小写无关的,参数和变量是大小写相关的。但,推荐你全部使用大写指令

语法注意事项

- SET(SRC_LIST main.cpp) 可以写成 SET(SRC_LIST "main.cpp"),如果源文件名中含有空格,就必须要加双引号
- ADD_EXECUTABLE(hello main) 后缀可以不行,他会自动去找.c和.cpp,最好不要这样写,可能会有这两个文件main.cpp和main

内部构建和外部构建

- 上述例子就是内部构建, 他生产的临时文件特别多, 不方便清理
- 外部构建,就会把生成的临时文件放在build目录下,不会对源文件有任何影响强烈使用外部构建方式

外部构建方式举例

```
//例子目录, CMakeLists.txt和上面例子一致
[root@localhost cmake]# pwd
/root/cmake
[root@localhost cmake]# 11
total 8
-rw-r--r-- 1 root root 198 Dec 28 20:59 CMakeLists.txt
-rw-r--r-- 1 root root 76 Dec 28 00:18 main.cpp
```

- 1、建立一个build目录,可以在任何地方,建议在当前目录下
- 2、进入build,运行cmake ... 当然..表示上一级目录,你可以写CMakeLists.txt所在的绝对路径,生产的文件都在build目录下了
- 3、在build目录下,运行make来构建工程

注意外部构建的两个变量

- 1、HELLO_SOURCE_DIR 还是工程路径
- 2、HELLO_BINARY_DIR 编译路径 也就是 /root/cmake/bulid

让Hello World看起来更像一个工程

- 为工程添加一个子目录 src, 用来放置工程源代码
- 添加一个子目录 doc, 用来放置这个工程的文档 hello.txt
- 在工程目录添加文本文件 COPYRIGHT, README
- 在工程目录添加一个 <u>runhello.sh</u> 脚本,用来调用 hello 二进制
- 将构建后的目标文件放入构建目录的 bin 子目录
- 将 doc 目录 的内容以及 COPYRIGHT/README 安装到/usr/share/doc/cmake/

将目标文件放入构建目录的 bin 子目录

每个目录下都要有一个CMakeLists.txt说明

PROJECT(HELLO)

ADD SUBDIRECTORY(src bin)

src下的CMakeLists.txt

ADD_EXECUTABLE(hello main.cpp)

ADD_SUBDIRECTORY 指令

ADD_SUBDIRECTORY(source_dir [binary_dir] [EXCLUDE_FROM_ALL])

- 这个指令用于向当前工程添加存放源文件的子目录,并可以指定中间二进制和目标二进制存放的位置
- EXCLUDE_FROM_ALL函数是将写的目录从编译中排除,如程序中的example
- ADD_SUBDIRECTORY(src bin)

将 src 子目录加入工程并指定编译输出(包含编译中间结果)路径为bin 目录 如果不进行 bin 目录的指定,那么编译结果(包括中间结果)都将存放在build/src 目录

更改二进制的保存路径

SET 指令重新定义 EXECUTABLE_OUTPUT_PATH 和 LIBRARY_OUTPUT_PATH 变量 来指定最终的目标二进制的位置

SET(EXECUTABLE_OUTPUT_PATH \${PROJECT_BINARY_DIR}/bin) SET(LIBRARY_OUTPUT_PATH \${PROJECT_BINARY_DIR}/lib)

思考:加载哪个CMakeLists.txt当中

哪里要改变目标存放路径,就在哪里加入上述的定义,所以应该在src下的CMakeLists.txt下写

安装

- 一种是从代码编译后直接 make install 安装
- 一种是打包时的指定 目录安装。
 - 简单的可以这样指定目录: make install DESTDIR=/tmp/test
 - 稍微复杂一点可以这样指定目录: ./configure -prefix=/usr

如何安装HelloWord

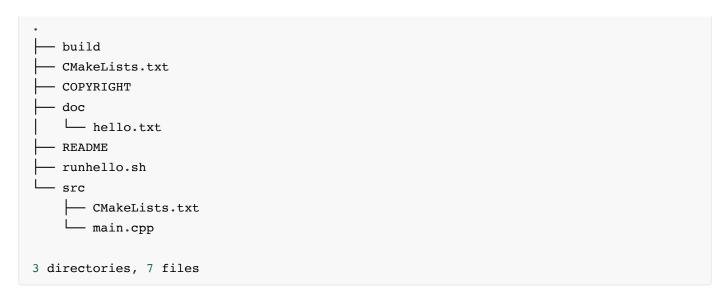
使用CMAKE一个新的指令: INSTALL

INSTALL的安装可以包括:二进制、动态库、静态库以及文件、目录、脚本等

使用CMAKE一个新的变量: CMAKE_INSTALL_PREFIX

// 目录树结构

[root@localhost cmake]# tree



安装文件COPYRIGHT和README

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

FILES: 文件

DESTINATION:

1、写绝对路径

2、可以写相对路径,相对路径实际路径是:\${CMAKE_INSTALL_PREFIX}/<DESTINATION 定义的路径>

CMAKE_INSTALL_PREFIX 默认是在 /usr/local/

cmake -DCMAKE_INSTALL_PREFIX=/usr 在cmake的时候指定CMAKE_INSTALL_PREFIX变量的路径

安装脚本runhello.sh

PROGRAMS: 非目标文件的可执行程序安装(比如脚本之类)

INSTALL(PROGRAMS <u>runhello.sh</u> DESTINATION bin)

说明:实际安装到的是/usr/bin

安装 doc 中的 hello.txt

• 一、是通过在 doc 目录建立CMakeLists.txt ,通过install下的file

• 二、是直接在工程目录通过

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

DIRECTORY 后面连接的是所在 Source 目录的相对路径

注意: abc 和 abc/有很大的区别

目录名不以/结尾: 这个目录将被安装为目标路径下的

目录名以/结尾:将这个目录中的内容安装到目标路径

安装过程

cmake ..

make

make install

静态库和动态库的构建

任务:

- 1,建立一个静态库和动态库,提供 HelloFunc 函数供其他程序编程使用,HelloFunc 向终端输出 Hello World 字符串。
- 2, 安装头文件与共享库。

静态库和动态库的区别

- 静态库的扩展名一般为".a"或".lib";动态库的扩展名一般为".so"或".dll"。
- 静态库在编译时会直接整合到目标程序中,编译成功的可执行文件可独立运行
- 动态库在编译时不会放到连接的目标程序中,即可执行文件无法单独运行。

构建实例

hello.h中的内容

```
#ifndef HELLO_H
#define Hello_H

void HelloFunc();
#endif
```

hello.cpp中的内容

```
#include "hello.h"
#include <iostream>
void HelloFunc(){
    std::cout << "Hello World" << std::endl;
}</pre>
```

项目中的cmake内容

```
PROJECT(HELLO)
ADD_SUBDIRECTORY(lib bin)
```

lib中CMakeLists.txt中的内容

```
SET(LIBHELLO_SRC hello.cpp)
ADD_LIBRARY(hello SHARED ${LIBHELLO_SRC})
```

ADD_LIBRARY

ADD_LIBRARY(hello SHARED \${LIBHELLO_SRC})

- hello: 就是正常的库名,生成的名字前面会加上lib, <u>最终产生的文件是libhello.so</u>
- SHARED, 动态库 STATIC, 静态库
- \${LIBHELLO_SRC}: 源文件

同时构建静态和动态库

```
// 如果用这种方式,只会构建一个动态库,不会构建出静态库,虽然静态库的后缀是。a
ADD_LIBRARY(hello SHARED ${LIBHELLO_SRC})
ADD_LIBRARY(hello STATIC ${LIBHELLO_SRC})

// 修改静态库的名字,这样是可以的,但是我们往往希望他们的名字是相同的,只是后缀不同而已
ADD_LIBRARY(hello SHARED ${LIBHELLO_SRC})

ADD_LIBRARY(hello_static STATIC ${LIBHELLO_SRC})
```

SET_TARGET_PROPERTIES

这条指令可以用来设置输出的名称,对于动态库,还可以用来指定动态库版本和 API 版本同时构建静态和动态库

```
SET(LIBHELLO_SRC hello.cpp)

ADD_LIBRARY(hello_static STATIC ${LIBHELLO_SRC})

//对hello_static的重名为hello

SET_TARGET_PROPERTIES(hello_static PROPERTIES OUTPUT_NAME "hello")

//cmake 在构建一个新的target 时,会尝试清理掉其他使用这个名字的库,因为,在构建 libhello.so 时,就会清理掉 libhello.a

SET_TARGET_PROPERTIES(hello_static PROPERTIES CLEAN_DIRECT_OUTPUT 1)

ADD_LIBRARY(hello SHARED ${LIBHELLO_SRC})

SET_TARGET_PROPERTIES(hello PROPERTIES OUTPUT_NAME "hello")

SET_TARGET_PROPERTIES(hello PROPERTIES CLEAN_DIRECT_OUTPUT 1)
```

动态库的版本号

一般动态库都有一个版本号的关联

```
libhello.so.1.2
libhello.so.1->libhello.so.1.2
```

CMakeLists.txt 插入如下

```
SET_TARGET_PROPERTIES(hello PROPERTIES VERSION 1.2 SOVERSION 1)
```

VERSION 指代动态库版本, SOVERSION 指代 API 版本。

安装共享库和头文件

本例中我们将 hello 的共享库安装到/lib目录,

将 hello.h 安装到/include/hello 目录

```
//文件放到该目录下
INSTALL(FILES hello.h DESTINATION include/hello)

//二进制,静态库,动态库安装都用TARGETS

//ARCHIVE 特指静态库,LIBRARY 特指动态库,RUNTIME 特指可执行目标二进制。
INSTALL(TARGETS hello hello_static LIBRARY DESTINATION lib ARCHIVE DESTINATION lib)
```

注意:

安装的时候,指定一下路径,放到系统下

```
cmake -DCMAKE_INSTALL_PREFIX=/usr ..
```

使用外部共享库和头文件

准备工作,新建一个目录来使用外部共享库和头文件

main.cpp

```
#include <hello.h>
int main(){
   HelloFunc();
}
```

解决: make后头文件找不到的问题

PS: include <hello/hello.h> 这样include是可以,这么做的话,就没啥好讲的了

关键字: INCLUDE_DIRECTORIES 这条指令可以用来向工程添加多个特定的头文件搜索路径,路径之间用空格分割

在CMakeLists.txt中加入头文件搜索路径

INCLUDE_DIRECTORIES(/usr/include/hello)

解决:找到引用的函数问题

报错信息: undefined reference to `HelloFunc()'

关键字: LINK_DIRECTORIES 添加非标准的共享库搜索路径

关键字: TARGET_LINK_LIBRARIES 添加需要链接的共享库

在CMakeLists.txt中插入链接共享库,主要要插在executable的后面

TARGET_LINK_LIBRARIES(main libhello.so)

查看main的链接情况

```
[root@MiWiFi-R4CM-srv bin]# ldd main
linux-vdso.so.1 => (0x00007ffedfda4000)
libhello.so => /lib64/libhello.so (0x00007f41c0d8f000)
libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007f41c0874000)
libm.so.6 => /lib64/libm.so.6 (0x00007f41c0572000)
libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f41c035c000)
libc.so.6 => /lib64/libc.so.6 (0x00007f41bff8e000)
/lib64/ld-linux-x86-64.so.2 (0x00007f41c0b7c000)
```

链接静态库

TARGET LINK LIBRARIES (main libhello.a)

特殊的环境变量 CMAKE INCLUDE PATH 和 CMAKE LIBRARY PATH

注意:这两个是环境变量而不是 cmake 变量,可以在linux的bash中进行设置 我们上面例子中使用了绝对路径INCLUDE_DIRECTORIES(/usr/include/hello)来指明include路径的位置

我们还可以使用另外一种方式,使用环境变量export CMAKE_INCLUDE_PATH=/usr/include/hello

本人所有视频和笔记都是免费分享给大家的,制作 视频和笔记要花费大量的时间成本

我也有老婆和孩子要养,恳求各位观众老爷们有经 济实力的稍微打赏一下小弟,但不强求,再一次感 谢。

您的打赏,会让我今后有更大的动力,做出更优质 的视频,感谢大家的支持



推荐使用支付宝



lewis(**斯)

打开支付宝[扫一扫]

申请官方收钱码: 拨打95188-6