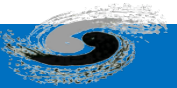
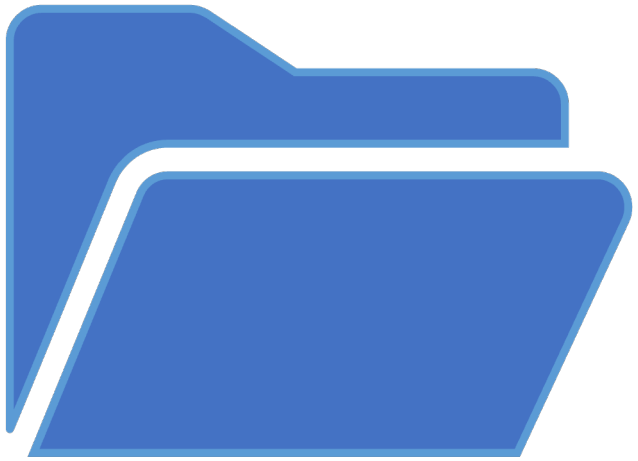
源代码的编译安装以及CMAKE应用

田浩来

IHEP School of Computing 2021

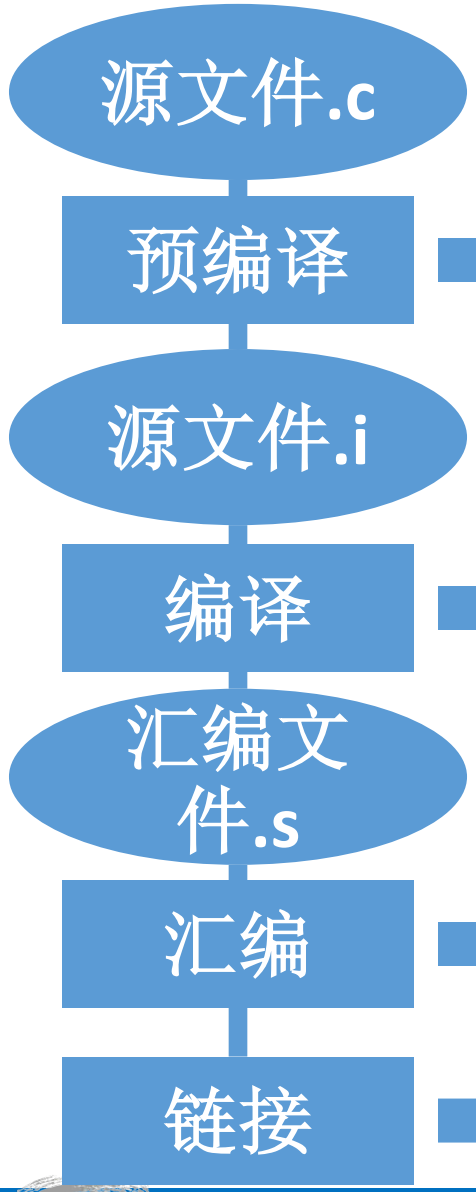


目录



- 源代码的编译和安装
- CMake的基本概念和安装
  CMake编译安装源代码
- Case study： CMakeList文件入门
- CMake一些深入的用法

编译链接的基本过程



- 将所有#define等宏定义展开
- 判断条件编译指令: #if #ifdef #elif #else #endif
- 根据#include预编译指令，插入头文件
- 其他预编译任务

- 词法分析=>语法分析=>语义分析
- 中间代码=>目标代码=>代码优化

- 汇编器as，汇编代码=>二进制的目标文件(.o)

- 链接器ld，目标文件=>可执行文件(.exe)

安装执行的基本过程

- 安装：
  - cp(1)可执行文件(2)动态链接库(3)头文件
- 执行：

```
[tianhl@localhost test]$ ls
bin  include  lib  lib64
```

  - 可执行程序的搜索路径

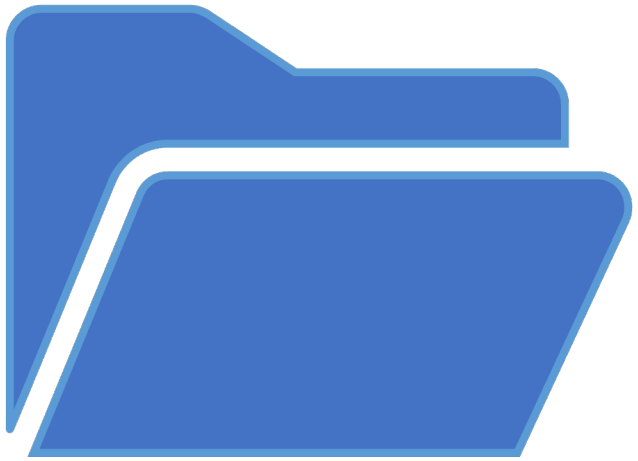
    环境变量PATH变量是Linux系统存储可执行文件搜索路径的地方

    export PATH=$PATH:/home/other/test/bin
  - 动态链接库的搜索路径

    环境变量LD_LIBRARY_PATH指定的动态库搜索路径

    export LD_LIBRARY_PATH=/home/other/test/lib:$LD_LIBRARY_PATH
- 配置搜索路径：
  - 系统启动配置脚本：/etc/profile; /etc/bashrc; /etc/profile.d/*.sh; etc
  - 用户启动配置脚本： ~/.bash_profile; ~/.bashrc; etc

目录



- 源代码的编译和安装
- CMake的基本概念和安装
  CMake编译安装源代码
- Case study： CMakeList文件入门
- CMake一些深入的用法

CMAKE基本知识

- 什么是CMAKE？

  CMake 是一个跨平台的编译构建工具，用来自动化生成构建文件。

- CMAKE执行步骤：

安装CMAKE

- 下载CMAKE安装包：https://cmake.org/download/
- wget https://cmake.org/files/v3.10/cmake-3.10.0-Linux-x86_64.sh
- chmod a+x cmake-3.10.0-Linux-x86_64.sh
- ./cmake-3.10.0-Linux-x86_64.sh

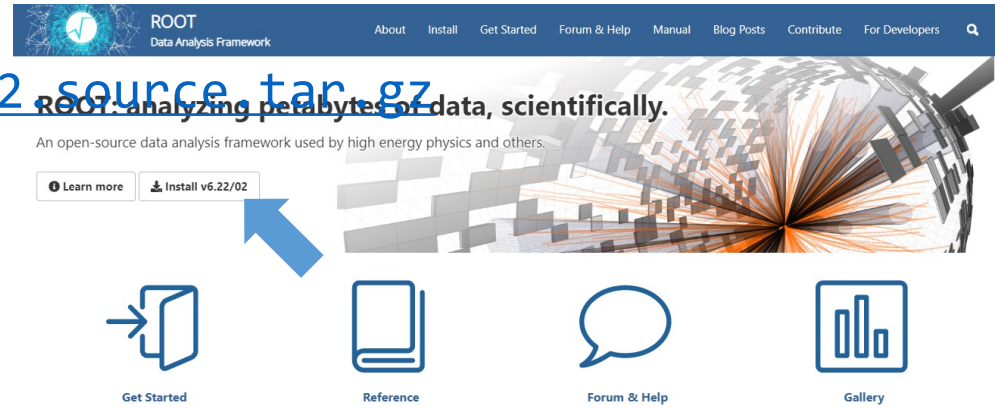
```
[tianhl@localhost cmake]$ wget https://cmake.org/files/v3.10/cmake-3.10.0-Linux-x86_64.sh
--2020-08-18 13:25:09--  https://cmake.org/files/v3.10/cmake-3.10.0-Linux-x86_64.sh
Resolving cmake.org (cmake.org)... 66.194.253.19
Connecting to cmake.org (cmake.org)|66.194.253.19|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 34221700 (33M) [text/x-sh]
Saving to: ‘cmake-3.10.0-Linux-x86_64.sh.1’

100%[======================================================================================================>]

2020-08-18 13:25:18 (4.34 MB/s) - ‘cmake-3.10.0-Linux-x86_64.sh.1’ saved [34221700/34221700]

[tianhl@localhost cmake]$ chmod a+x cmake-3.10.0-Linux-x86_64.sh
[tianhl@localhost cmake]$ ./cmake-3.10.0-Linux-x86_64.sh
CMake Installer Version: 3.10.0, Copyright (c) Kitware
This is a self-extracting archive.
The archive will be extracted to: /home/tianhl/workarea/CMakeWorkshop/cmake

If you want to stop extracting, please press <ctrl-C>.
CMake - Cross Platform Makefile Generator
Copyright 2000-2017 Kitware, Inc. and Contributors
All rights reserved.
```

- 可执行文件：

```
[tianhl@localhost cmake]$ ls cmake-3.10.0-Linux-x86_64/bin/
ccmake  cmake  cmake-gui  cpack  ctest
```

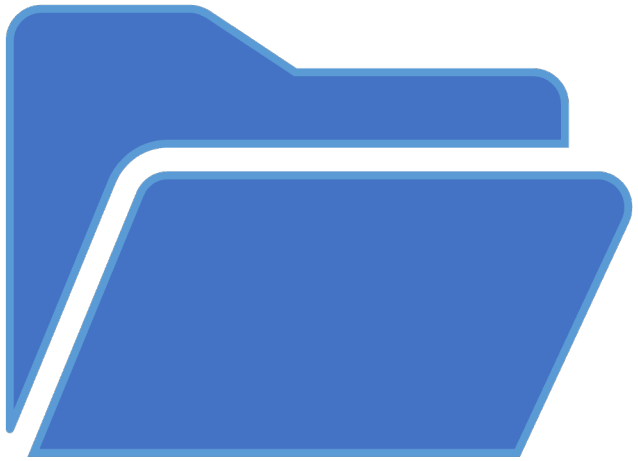
利用CMAKE编译ROOT

- 下载ROOT安装包：https://root.cern.ch/
- wget https://root.cern/download/root_v6.22.02.source.tar.gz
- tar -xzvf root_v6.22.02.source.tar.gz
- cd root-6.22.02/build/
- cmake .. ; make -j40
- source bin/thisroot.sh




```
[tianhl@localhost build]$ ../../../cmake/cmake-3.10.0-Linux-x86_64/bin/cmake .. ; make -j40
```

目录



- 源代码的编译和安装
- CMake的基本概念和安装
  CMake编译安装源代码
- Case study： CMakeList文件入门
- CMake一些深入的用法

CASE Study 1: g++/make编译可执行文件

- g++编译可执行文件

```cpp
#include <iostream>

using namespace std;


int main(int argc,char** argv)
{
    cout << "Hello for cmake!" << endl;
    return 0;
}

```

CPP源代码

```
[tianhl@localhost cmake1]$ g++ -o helloworld helloworld.cpp
[tianhl@localhost cmake1]$ ./helloworld
Hello for cmake!
```

编译执行

- make编译可执行文件

```
helloMake: helloworld.cpp
	g++ helloworld.cpp -o helloMake

clean:
	rm helloMake  -rf

```

Makefile

```
[tianhl@localhost cmake1]$ make
g++ helloworld.cpp -o helloMake
[tianhl@localhost cmake1]$ ./helloMake
Hello for cmake!
[tianhl@localhost cmake1]$ make clean
rm helloMake  -rf
[tianhl@localhost cmake1]$
```

编译执行

CASE Study 1: cmake编译可执行文件

```
1 cmake_minimum_required(VERSION 2.8.11)
2 project(HELLO)
3 add_definitions(-w)
4
5 add_executable(helloworld helloworld.cpp)
6
7
```

CMakeLists.txt

```
# Build rule for target.
helloworld: cmake_check_build_system
	$(MAKE) -f CMakeFiles/Makefile2 helloworld
.PHONY : helloworld

# fast build rule for target.
helloworld/fast:
	$(MAKE) -f CMakeFiles/helloworld.dir/build.make CMakeFiles/helloworld.dir/build
.PHONY : helloworld/fast

helloworld.o: helloworld.cpp.o
.PHONY : helloworld.o

# target to build an object file
helloworld.cpp.o:
	$(MAKE) -f CMakeFiles/helloworld.dir/build.make CMakeFiles/helloworld.dir/helloworld.cpp.o
.PHONY : helloworld.cpp.o

helloworld.i: helloworld.cpp.i
.PHONY : helloworld.i

# target to preprocess a source file
helloworld.cpp.i:
	$(MAKE) -f CMakeFiles/helloworld.dir/build.make CMakeFiles/helloworld.dir/helloworld.cpp.i
.PHONY : helloworld.cpp.i

helloworld.s: helloworld.cpp.s
.PHONY : helloworld.s

# target to generate assembly for a file
helloworld.cpp.s:
	$(MAKE) -f CMakeFiles/helloworld.dir/build.make CMakeFiles/helloworld.dir/helloworld.cpp.s
.PHONY : helloworld.cpp.s
```

Makefile

```
[root@localhost build]# cmake ..
-- The C compiler identification is GNU 4.8.5
-- The CXX compiler identification is GNU 4.8.5
-- 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
-- Configuring done
-- Generating done
```

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

编译

```
[root@localhost build]# ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  helloworld  Makefile
[root@localhost build]# ./helloworld
Hello for cmake!
```

执行

CASE Study 2: 编译动态链接库

```
#ifndef __WORLD_H__
#define __WORLD_H__

void say_hello();

#endif

```

头文件源代码

```
#include <iostream>
#include "sayhello.hpp"


void say_hello()
{
    std::cout << "Say Hello from DL builded by CMake" << std::endl;
}

```

CPP源代码

```
cmake_minimum_required(VERSION 2.8.11)
project(SayHelloLib)

include_directories("sayhello")
add_library(sayhello SHARED sayhello.cpp)


INSTALL(TARGETS   sayhello LIBRARY DESTINATION ${PROJECT_SOURCE_DIR}/../lib)
INSTALL(DIRECTORY sayhello DESTINATION ${PROJECT_SOURCE_DIR}/../inc)

```

CMakeLists.txt

```
[tianhl@localhost build]$ cmake ..
-- The C compiler identification is GNU 4.8.5
-- The CXX compiler identification is GNU 4.8.5
-- 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
-- Configuring done
-- Generating done
-- Build files have been written to: /home/tianhl/workarea/CMakeWorkshop/cmake2/build
[tianhl@localhost build]$ make
Scanning dependencies of target sayhello
[100%] Building CXX object CMakeFiles/sayhello.dir/sayhello.cpp.o
Linking CXX shared library libsayhello.so
[100%] Built target sayhello
[tianhl@localhost build]$ make install
[100%] Built target sayhello
Install the project...
-- Install configuration: ""
-- Installing: /home/tianhl/workarea/CMakeWorkshop/cmake2/../lib/libsayhello.so
-- Installing: /home/tianhl/workarea/CMakeWorkshop/cmake2/../inc/sayhello
-- Installing: /home/tianhl/workarea/CMakeWorkshop/cmake2/../inc/sayhello/sayhello.hpp
```

编译=>安装

```
[tianhl@localhost build]$ ls ../../inc/sayhello/
sayhello.hpp
[tianhl@localhost build]$ ls ../../lib/
libsayhello.so
```

动态链接库/头文件

CASE Study 3: 链接动态链接库

```cpp
#include <iostream>
#include "sayhello/sayhello.hpp"

using namespace std;


int main(int argc,char** argv)
{
    cout << "Hello for cmake!" << endl;
    say_hello();
    return 0;
}

```

CPP源代码

```
cmake_minimum_required(VERSION 2.8.11)
project(CallHELLO)

link_directories(${PROJECT_SOURCE_DIR}/../lib)
include_directories(${PROJECT_SOURCE_DIR}/../inc)
add_executable(helloDL helloDL.cpp)

target_link_libraries(helloDL sayhello)


INSTALL(TARGETS helloDL DESTINATION ${PROJECT_SOURCE_DIR}/../bin)

```

CMakeLists.txt

```
tianhl@localhost build]$ cmake ..
- The C compiler identification is GNU 4.8.5
- The CXX compiler identification is GNU 4.8.5
- 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
- Configuring done
- Generating done
```

```
tianhl@localhost build]$ make
canning dependencies of target helloDL
100%] Building CXX object CMakeFiles/helloDL.dir/helloDL.cpp.o
inking CXX executable helloDL
```

```
tianhl@localhost build]$ make install
100%] Built target helloDL
nstall the project...
- Install configuration: ""
- Installing: /home/tianhl/workarea/CMakeWorkshop/cmake3/../bin/helloDL
```

编译=>安装

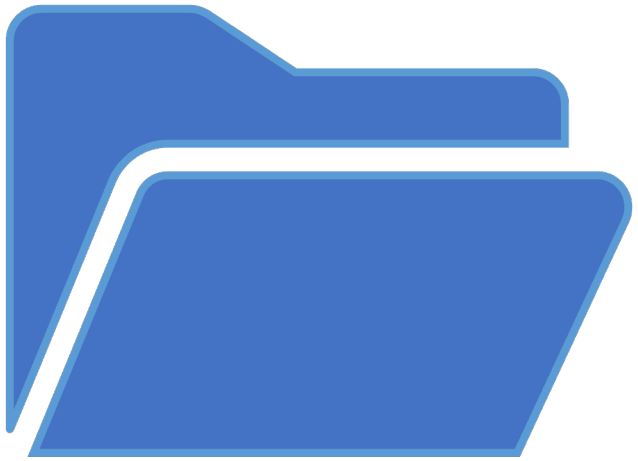
```
export PATH=/home/tianhl/workarea/CMakeWorkshop/bin:$PATH
export LD_LIBRARY_PATH=/home/tianhl/workarea/CMakeWorkshop/lib:$LD_LIBRARY_PATH

```

设置环境变量

```
[tianhl@localhost build]$ helloDL
Hello for cmake!
Say Hello from DL builded by CMake
```

执行

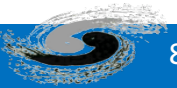
目录



- 源代码的编译和安装
- CMake的基本概念和安装
  CMake编译安装源代码
- Case study： CMakeList文件入门
- CMake一些深入的用法

设置环境变量(搜索路径)

- 关键的搜索路径
  - 头文件路径
  - 动态链接库路径
- 脚本设置
  - export CMAKE_INCLUDE_PATH=/opt/boost/include:$CMAKE_INCLUDE_PATH
  - export CMAKE_LIBRARY_PATH=/opt/boost/lib:$CMAKE_LIBRARY_PATH
- CMakeLists.txt设置
  - include_directories(${PROJECT_SOURCE_DIR}/inc)
  - link_directories((${PROJECT_SOURCE_DIR}/lib)
- CMake命令设置
  - -DXXX_INCLUDE_DIR=$(……)
  - -DXXX_LIBRARY=$(……)



查找动态链接库以其头文件

```
# Find Package
FIND_PACKAGE(Boost REQUIRED COMPONENTS python3)
FIND_PACKAGE(PythonLibs)
IF(Boost_FOUND AND PYTHONLIBS_FOUND)
  INCLUDE_DIRECTORIES(${Boost_INCLUDE_DIRS} ${PYTHON_INCLUDE_PATH})
  SET(Boost_USE_STATIC_LIBS OFF)
  SET(Boost_USE_MULTITHREADED ON)
  SET(Boost_USE_STATIC_RUNTIME OFF)
  FIND_PACKAGE(Boost  COMPONENTS python REQUIRED)
ELSEIF(NOT Boost_FOUND)
  MESSAGE(FATAL_ERROR "Unable to find Boost. Did you set BOOST_ROOT?")
ELSEIF(NOT PYTHONLIBS_FOUND)
  MESSAGE(FATAL_ERROR "Unable to find Python. Did you set PYTHONPATH?")
ENDIF()
```

- Cmake Module查找
  - 设置动态链接库

    ${XXX_LIBRARIES}
  - 设置头文件

    ${XXX_INCLUDE_DIR}

- 链接时调用

```
# Compile Library for CPP
add_library(SniperKernel SHARED ${CPPLIB_SRC_LIST})
target_link_libraries(SniperKernel ${Boost_LIBRARIES})

# Compile Library for Python
ADD_LIBRARY(SniperPython SHARED ${PYBLIB_SRC_LIST})
TARGET_LINK_LIBRARIES(SniperPython  ${Boost_LIBRARIES})
TARGET_LINK_LIBRARIES(SniperPython  SniperKernel)
```

自定义CMAKE Module

```
# Look for the header file.
SET(CMAKE_INCLUDE_PATH ${CMAKE_INSTALL_PREFIX})
SET(CMAKE_LIBRARY_PATH ${CMAKE_INSTALL_PREFIX})

FIND_PATH(Dim_INCLUDE_DIR NAMES dim.h)
MARK_AS_ADVANCED(Dim_INCLUDE_DIR)

# Look for the library.
FIND_LIBRARY(Dim_LIBRARY NAMES dim)
MARK_AS_ADVANCED(Dim_LIBRARY)

# handle the QUIETLY and REQUIRED arguments and set SniperKernel_FOUND to TRUE if
# all listed variables are TRUE
INCLUDE(FindPackageHandleStandardArgs)
FIND_PACKAGE_HANDLE_STANDARD_ARGS(Dim DEFAULT_MSG Dim_LIBRARY Dim_INCLUDE_DIR)

MESSAGE(STATUS "Dim include path: " ${Dim_INCLUDE_DIR})

IF(DIM_FOUND)
        SET(Dim_LIBRARIES    ${Dim_LIBRARY})
        SET(Dim_INCLUDE_DIRS ${Dim_INCLUDE_DIR})
ELSE(Dim_FOUND)
        SET(Dim_LIBRARIES)
        SET(Dim_INCLUDE_DIRS)
ENDIF(DIM_FOUND)
```

- 搜索头文件
  - FIND_PATH
  - XXX.h
- 搜索动态链接库
  - FIND_LIBRARY
  - libXXX.so
- 设置CMAKE环境变量
  - XXX_DOUND
  - XXX_INCLUDE_PATH
  - XXX_LIBRARIES


enjoy it!