10. CMake理论与实践

10.1 什么是CMake?

CMake 是"Cross platform MAke"的缩写

• 一个开源的跨平台自动化建构系统，用来管理程序构建，不相依于特定编译器

• 需要编写CMakeList.txt 文件来定制整个编译流程（需要一定学习时间）

• 可以自动化编译源代码、创建库、生成可执行二进制文件等

• 为开发者节省大量时间，工程实践必备

Write once, run everywhere

CMake 不再使你在构建项目时郁闷地想自杀了 -- 一位开发者

学习资料：官网：www.cmake.org 、《CMake practice》 、《learning CMake》

10.2 CMake有什么优缺点?

优点：

• 开源，使用类 BSD 许可发布

• 跨平台使用，根据目标用户的平台进一步生成所需的本地化 Makefile 和工程文件，如 Unix的Makefile 或Windows 的 Visual Studio 工程

• 能够管理大型项目，比如OpenCV、Caffe、MySql Server

• 自动化构建编译，CMake 构建项目效率非常高

注意：

• 需要根据CMake 专用语言和语法来自己编写CMakeLists.txt 文件

• Cmake 支持很多语言：C、C++、Java 等

• 如项目已经有非常完备的工程管理工具，并且不存在维护问题，没必要迁移到CMake

10.3 CMake如何安装?

Windows下：按提示无脑安装

Linux下

• apt 安装【推荐，够用】 但是Ubuntu源里版本可能比较低

```
sudo apt-get install cmake
sudo apt-get install cmake-gui
```

源码编译【需要最新版本时】， 解压后执行

```
./bootstrap
make -j2
sudo make install
cmake --version
```

Platform	Files
Unix/Linux Source (has \\n line feeds)	cmake-3.19.0-rc2.tar.gz
	cmake-3.19.0-rc2.tar.Z
Windows Source (has \\r\\n line feeds)	cmake-3.19.0-rc2.zip

Binary distributions:

Platform	Files
Windows win64-x64 Installer: **Installer tool has changed. Uninstall CMake 3.4 or lower first!**	cmake-3.19.0-rc2-win64-x64.msi
Windows win64-x64 ZIP	cmake-3.19.0-rc2-win64-x64.zip
Windows win32-x86 Installer: **Installer tool has changed. Uninstall CMake 3.4 or lower first!**	cmake-3.19.0-rc2-win32-x86.msi
Windows win32-x86 ZIP	cmake-3.19.0-rc2-win32-x86.zip
Mac OS X 10.7 or later	cmake-3.19.0-rc2-Darwin-x86_64.dmg
	cmake-3.19.0-rc2-Darwin-x86_64.tar.gz
Linux x86_64	cmake-3.19.0-rc2-Linux-x86_64.sh
	cmake-3.19.0-rc2-Linux-x86_64.tar.gz

10.4 CMake 到底多好用?

举个栗子：OpenCV 在Windows下的配置方法

毛星云的博客里OpenCV配置方法

• 需要手动添加环境变量、 项目中手动添加包含路径、项目中手动添加库路径、项目中手动添加链接库名、Debug 和Release下配置不同、 和OpenCV版本相关、构建好的项目不能直接移植到其他平台，给别人用代码成本也太高

太繁琐！问题多，易出错！刚开始就想放弃了！

CMake 配置方法

• CMake 一键配置、以上所有东西自动关联，并且和OpenCV版本无关、跨平台移植，效率高

无脑安装！超简单！用过的都说好！学习劲头更足了！

10.5 CMake使用注意事项

CMakeLists.txt文件

• CMake 构建专用定义文件，文件名严格区分大小写

• 工程存在多个目录，可以每个目录都放一个CMakeLists.txt文件

• 工程存在多个目录，也可以只用一个CMakeLists.txt文件管理

CMake指令

• 不区分大小写，可以全用大写，全用小写，甚至大小写混合，自己统一风格即可

```
add_executable(hello main.cpp)
ADD_EXECUTABLE(hello main.cpp)
```

参数和变量

• 严格大小写相关。名称中只能用字母、数字、下划线、破折号

• 用${}来引用变量

• 参数之间使用空格进行间隔

10.6 CMake常用指令介绍

> cmake_minimum_required

• 指定要求最小的cmake版本，如果版本小于该要求，程序终止

> project(test)

• 设置当前项目名称为test

> CMAKE_BUILD_TYPE

```
Debug: 调试模式，输出调试信息，不做优化
Release: 发布模式，没有调试信息，全优化
RelWithDebInfo::类似Release，但包括调试信息
MinSizeRel: 一种特殊的Release模式，会特别优化库的大小
```

> CMAKE_CXX_FLAGS

• 编译CXX的设置标志，比如 –std=c++11, -Wall, -O3（优化，使用向量化、CPU流水线，cache等提高代码速度）

• 编译过程中输出警告（warnings）：set(CMAKE_CXX_FLAGS "-Wall")

• 追加，不会丢失之前的定义：set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall")

> include_directories

• 指定头文件的搜索路径，编译器查找相应头文件

• 举例：文件main.cpp中使用到路径 /usr/local/include/opencv/cv.h 中这个文件

• CMakeLists.txt 中添加 include_directories(/usr/local/include)

• 使用时main.cpp前写上 #include “opencv/cv.h" 即可

> set (variable value)

• 用变量代替值

• set (SRC_LST main.cpp other.cpp) 表示定义SRC_LST代替后面的两个cpp文件

> add_executable(hello main.cpp)

• 用指定的源文件为工程添加可执行文件

• 工程会用main.cpp生成一个文件名为 hello 的可执行文件

> add_library(libname STATIC/SHARED sources)

• 将指定的源文件生成链接库文件。STATIC 为静态链接库，SHARED 为共享链接库

> target_link_libraries (target library1 library2 ...)

• 为库或二进制可执行文件添加库链接，要用在add_executable之后。

• 例子如下：

• target_link_libraries (myProject libhello.a)

> add_dependencies (target-name depend)

• 为上层target添加依赖，一般不用

• 若只有一个targets有依赖关系，一般选择使用 target_link_libraries

• 如果两个targets有依赖关系，并且依赖库也是通过编译源码产生的。这时候用该指令可以在编译上层target时，自动检查下层依赖库是否已经生成

> add_subdirectory(source_dir)

• 向当前工程添加存放源文件的子目录，目录可以是绝对路径或相对路径

> aux_source_directory(dir varname)

• 在目录下查找所有源文件

> message(mode "message text")

• 打印输出信息，mode包括FATAL_ERROR、WARNING、STATUS、DEBUG等

• message(STATUS “Set debug mode")

一些预定义好的指令

```
PROJECT_NAME：项目名称，与project( xxx) 一致
PROJECT_SOURCE_DIR：即内含 project() 指令的 CMakeLists 所在的文件夹
EXECUTABLE_OUTPUT_PATH：可执行文件输出路径
LIBRARY_OUTPUT_PATH ：库文件输出路径
CMAKE_BINARY_DIR：默认是build文件夹所在的绝对路径
CMAKE_SOURCE_DIR：源文件所在的绝对路径
```

> find_package(package version EXACT/QUIET/REQUIRED)

• 功能：采用两种模式（FindXXX.cmake和XXXConfig.cmake）搜索外部库

• 示例：find_package(OpenCV 3.4 REQUIRED)

• version：指定查找库的版本号。EXACT：要求该版本号必须精确匹配。QUIET：禁掉没有找到时的警告信息。REQUIRED选项表示如果包没有找到的话，CMake的过程会终止，并输出警告信息。

• 搜索有两种模式：

• Module模式：搜索CMAKE_MODULE_PATH指定路径下的FindXXX.cmake文件，执行该文件从而找到XXX库。其中，具体查找库并给XXX_INCLUDE_DIRS和XXX_LIBRARIES两个变量赋值的操作由FindXXX.cmake模块完成。

• Config模式：搜索XXX_DIR指定路径下的XXXConfig.cmake文件从而找到XXX库。其中具体查找库并给XXX_INCLUDE_DIRS和XXX_LIBRARIES两个变量赋值的操作由XXXConfig.cmake模块完成。

• 两种模式看起来似乎差不多，不过cmake默认采取Module模式，如果Module模式未找到库，才会采取Config模式。

• 如果XXX_DIR路径下找不到XXXConfig.cmake文件，则会找/usr/local/lib/cmake/XXX/中的XXXConfig.cmake文件。总之，Config模式是一个备选策略。通常，库安装时会拷贝一份XXXConfig.cmake到系统目录中，因此在没有显式指定搜索路径时也可以顺利找到。

• 若XXX安装时没有安装到系统目录，则无法自动找到XXXConfig.cmake，需要在CMakeLists.txt最前面添加XXX的搜索路径。

• set(XXX_DIR /home/cxl/projects/OpenCV3.1/build) #添加OpenCV的搜索路径

• 当find_package找到一个库的时候，以下变量会自动初始化:

```
<NAME>_FOUND ： 显示是否找到库的标记
<NAME>_INCLUDE_DIRS 或 <NAME>_INCLUDES ： 头文件路径
<NAME>_LIBRARIES 或 <NAME>_LIBRARIES 或 <NAME>_LIBS ： 库文件
<NAME>_DEFINITIONS:
```

```
find_package(Eigen3 3.1.0 REQUIRED)
find_package(Pangolin REQUIRED)

include_directories(
${PROJECT_SOURCE_DIR}
${PROJECT_SOURCE_DIR}/include
${EIGEN3_INCLUDE_DIR}
${Pangolin_INCLUDE_DIRS}
)
```

```
target_link_libraries(${PROJECT_NAME}
${OpenCV_LIBS}
${EIGEN3_LIBS}
${Pangolin_LIBRARIES}
${PROJECT_SOURCE_DIR}/Thirdparty/DBoW2/lib/libDBoW2.so
${PROJECT_SOURCE_DIR}/Thirdparty/g2o/lib/libg2o.so
)
```

> list

- 列表操作（读、搜索、修改、排序）
- 追加例子：LIST(APPEND CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/cmake_modules)

> If, elseif, endif

- 判断语句，使用和C语言一致

> foreach

- 循环指令

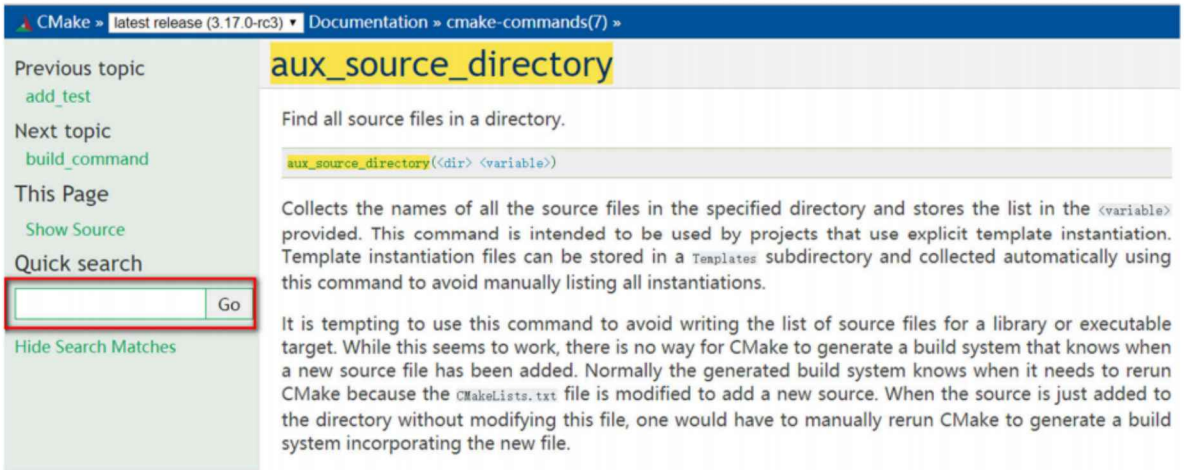
```
# 格式
foreach(<loop_var> <items>)
endforeach()

# 例如
foreach(i 0 1 2 3)
    message(STATUS "current is ${i}")
endforeach(i)

# 输出
-- current is 0
-- current is 1
-- current is 2
-- current is 3
```

```
# 格式2
foreach(loop_var RANGE start stop [step])

# 示例
foreach(i RANGE 0 3 1)
    message(STATUS "current is ${i}")
endforeach(i)

# 输出
-- current is 0
-- current is 1
-- current is 2
-- current is 3
```

```
Reading
  list(LENGTH <list> <out-var>)
  list(GET <list> <element index> [<index> ...] <out-var>)
  list(JOIN <list> <glue> <out-var>)
  list(SUBLIST <list> <begin> <length> <out-var>)

Search
  list(FIND <list> <value> <out-var>)

Modification
  list(APPEND <list> [<element>...])
  list(FILTER <list> {INCLUDE | EXCLUDE} REGEX <regex>)
  list(INSERT <list> <index> [<element>...])
  list(POP_BACK <list> [<out-var>...])
  list(POP_FRONT <list> [<out-var>...])
  list(PREPEND <list> [<element>...])
  list(REMOVE_ITEM <list> <value>...)
  list(REMOVE_AT <list> <index>...)
  list(REMOVE_DUPLICATES <list>)
  list(TRANSFORM <list> <ACTION> [...])

Ordering
  list(REVERSE <list>)
  list(SORT <list> [...])
```

10.7 CMake如何查询指令?

https://cmake.org/cmake/help/latest/genindex.html#



10.8 静态库和共享库

静态库

• 原理：在编译时将源代码复制到程序中，运行时不用库文件依旧可以运行。

• 优点：运行已有代码，运行时不用再用库；无需加载库，运行更快

• 缺点：占用更多的空间和磁盘；静态库升级，需要重新编译程序

共享库（常用）

• 原理：编译时仅仅是记录用哪一个库里面的哪一个符号，不复制相关代码

• 优点：不复制代码，占用空间小；多个程序可以同时调用一个库；升级方便，无需重新编译

• 缺点：程序运行需要加载库，耗费一定时间

	静态库	共享库
Windows	lib	dll
Linux	.a	.so
Mac OS	.a	dylib

10.9 如何安装库?

一般安装库流程,以Pangolin为例:

```
git clone https://github.com/stevenlovegrove/Pangolin.git
cd Pangolin
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=yourdirectory ..
make -j4
sudo make install
```

• cmake ..(注意..代表上一级目录)

• make install 默认安装位置 /usr/bin

• make clean：可对构建结果进行清理

10.10 如何使用库?

当编译一个需要使用第三方库的软件时，我们需要知道:

• 去哪儿找头文件 .h

• 去哪儿找库文件 (.so/.dll/.lib/.dylib/...)

• 需要链接的库文件的名字

• 比如需要一个第三方库 curl，不使用find命令的话， CMakeLists.txt 需要指定头文件目录和库文件:

```
include_directiories(/usr/include/curl)
target_link_libraries(myprogram yourpath/curl.so)
```

• 使用cmake的Modules目录下的FindCURL.cmake，就很简单了，相应的CMakeLists.txt 文件：

```
find_package(CURL REQUIRED)
include_directories(${CURL_INCLUDE_DIR})
target_link_libraries(curltest ${CURL_LIBRARY})
```

11. C++多线程理论与实践

11.1并发 和 并行 的区别?

网上很多解释，鱼龙混杂。并行指的是程序运行时的状态，就是同时运行的意思。并发指的是程序的结构，这个程序程序同时执行多个独立的任务就说这个程序是并发的，实际上，这句话应当表述成“这个程序采用了支持并发的设计”。我们后面讲的都是代码结构，都是指并发。

11.2 理解并发

单核CPU如何产生“并发”

• 单核CPU：某一个时刻只能执行一个任务，由操作系统调度，每秒钟进行多次“任务切换”,来实现并发的假象（不是真正的并发），切换任务时要保存变量的状态、执行进度等，存在时间开销；

• 人脑就是单核运算结构

• 一个人很难做到：一手画圆，一手画方

• 人为什么可以一边走路，一边看手机?

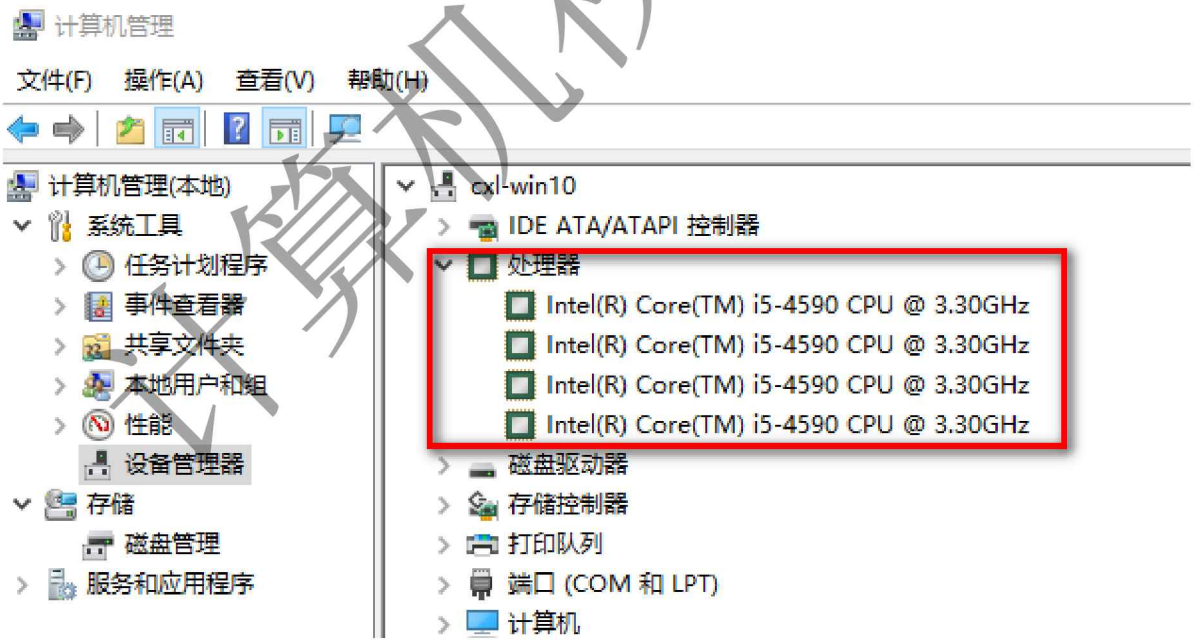
• 专心依次做多件事 和 同时做多件事情 哪个效率高?

• 富士康流水线为什么效率高?

多核CPU

• 双核，4核，8核，10核，能够实现真正的并行执行多个任务（硬件并发）

• 如何查看自己电脑CPU核心数目?



概念理解

可执行程序

• Windows下扩展名为 exe

• Linux下为bin文件

进程

• 进程就是运行起来的可执行程序