Cmake 学习

什么是 Cmake?

CMake 意为 cross-platform make，可用于管理 c/c++工程。CMake 解析配置文件 CMakeLists.txt 生成 Makefile，相比直接用 Makefile 管理工程，CMake 更灵活和简单。

cmake 是一个跨平台、开源的构建系统。它是一个集软件构建、测试、打包于一身的软件。它使用与平台和编译器独立的配置文件来对软件编译过程进行控制。现在许多跨平台的开源软件都转向了用 cmake 来做构建工具，如 KDE，Kdevelop，hypertable 等，使用 cmake，我们可以不用关心如何去创建编译可执行文件和动态库。它为了支持不同的平台，提供了以下特性：跨库依赖检查，并行构建和简单的头文件结构，这使它大大减少了跨平台软件的开发和维护过程的复杂性。

CMake 的两大功能

1，配置和生成各大平台的工程（vs 的 vcxproj，Qt 的 Pro）：

比如设置输出目录，设置编译对象的 debug 后缀，设置源码在工程中的那个文件夹（Filter），配置需要依赖的第三方的头文件目录，库目录等等属性

2，生成 makefile 文件

计算机编译源文件的时候是一条指令一条指令的发送给编译器执行的，这样效率很低下，所以就产生了一种文件，把所有的命令写到一个文件中，这个文件就是 makefile。CMake 生成了这个 makeFile 之后，各大平台的编译器都会拿到这个 makeFile 然后解析它。将他的命令解析出来一条一条执行。

cmake 常见命令:

cmake_minimum_required(VERSION 2.8)　　#指定需要的最小的 cmake 版本

aux_source_directory(　)　　#查找源文件并保存到相应的变量中

```
#查找当前目录下所有源文件并保存至SRC_LIST变量中
aux_source_directory(. SRC_LIST)
```

add_library　#添加一个库

（1）添加一个名为<name>的库文件

```
add_library(<name> [STATIC | SHARED | MODULE] [EXCLUDE_FROM_ALL] source1 source2 ...
```

#指定 STATIC, SHARED, MODULE 参数来指定要创建的库的类型, STATIC 对应的静态库(.a), SHARED 对应共享动态库(.so)

source1 source2 ... sourceN 用来指定源文件

（2）导入已有的库

```
add_library(<name> [STATIC | SHARED | MODULE | UNKNOWN] IMPORTED)
```

#导入了一个已存在的<name>库文件, 导入库一般配合 set_target_properties 使用, 这个命令用来指定导入库的路径,

```
add_library(test SHARED IMPORTED)
set_target_properties(  test #指定目标库名称
                        PROPERTIES IMPORTED_LOCATION #指明要设置的参数
                        libs/src/${ANDROID_ABI}/libtest.so #设定导入库的路径)
```

Set 命令　#作用是用于设置 cmake 的变量

```
# 设置可执行文件的输出路径(EXCUTABLE_OUTPUT_PATH是全局变量)
set(EXECUTABLE_OUTPUT_PATH [output_path])
```

```
# 设置库文件的输出路径(LIBRARY_OUTPUT_PATH是全局变量)
set(LIBRARY_OUTPUT_PATH [output_path])
```

```
# 设置C++编译参数(CMAKE_CXX_FLAGS是全局变量)
set(CMAKE_CXX_FLAGS "-Wall std=c++11")
```

```
# 设置源文件集合(SOURCE_FILES是本地变量即自定义变量)
set(SOURCE_FILES main.cpp test.cpp ...)
```

include_directories　#用于设置头文件的位置

```
# 可以用相对货绝对路径，也可以用自定义的变量值
include_directories(./include ${MY_INCLUDE})
```

add_executable　#添加可执行文件

```
add_executable(<name> ${SRC_LIST})
```

target_link_libraries　#将若干库链接到目标库文件

```
target_link_libraries(<name> lib1 lib2 lib3)
```

将 lib1, lib2, lib3 链接到<name>上，注意：链接的顺序应当符合 gcc 链接顺序规则，被链接的库放在依赖它的库的后面，即如果上面的命令中，lib1 依赖于 lib2, lib2 又依赖于 lib3，则在上面命令中必须严格按照 lib1 lib2 lib3 的顺序排列，否则会报错

add_subdirectory　#为当前目录添加子目录

```
# sub_dir指定包含CMakeLists.txt和源码文件的子目录位置
# binary_dir是输出路径，  一般可以不指定
add_subdirecroty(sub_dir [binary_dir])
```

File #相关文件操作命令

```
# 将message写入filename文件中,会覆盖文件原有内容
file(WRITE filename "message")

# 将message写入filename文件中，会追加在文件末尾
file(APPEND filename "message")

# 从filename文件中读取内容并存储到var变量中，如果指定了numBytes和offset，
# 则从offset处开始最多读numBytes个字节，另外如果指定了HEX参数，则内容会以十六进制形式存储在var变量中
file(READ filename var [LIMIT numBytes] [OFFSET offset] [HEX])

# 重命名文件
file(RENAME <oldname> <newname>)

# 删除文件，  等于rm命令
file(REMOVE [file1 ...])

# 递归的执行删除文件命令，等于rm -r
file(REMOVE_RECURSE [file1 ...])

# 根据指定的url下载文件
# timeout超时时间；下载的状态会保存到status中；下载日志会被保存到log;
# sum指定所下载文件预期的MD5值,如果指定会自动进行比对，如果不一致，则返回一个错误;
# SHOW_PROGRESS，进度信息会以状态信息的形式被打印出来
file(DOWNLOAD url file [TIMEOUT timeout] [STATUS status] [LOG log] [EXPECTED_MD5 sum] [SHOW_PROGRESS])
```

```
# 创建目录
file(MAKE_DIRECTORY [dir1 dir2 ...])

# 会把path转换为以unix的/开头的cmake风格路径,保存在result中
file(TO_CMAKE_PATH path result)

# 它会把cmake风格的路径转换为本地路径风格：windows下用"\"，而unix下用"/"
file(TO_NATIVE_PATH path result)

# 将会为所有匹配查询表达式的文件生成一个文件list，并将该list存储进变量variable里，
# 如果一个表达式指定了RELATIVE，返回的结果将会是相对于给定路径的相对路径，查询表达式例子：*.cxx，*.vt?
NOTE：按照官方文档的说法，不建议使用file的GLOB指令来收集工程的源文件
file(GLOB variable [RELATIVE path] [globbing expressions]...)
```

set_directory_properties　#设置某个路径的一种属性

```
# prop1 prop代表属性
# 取值为：INCLUDE_DIRECTORIES  LINK_DIRECTORIES  INCLUDE_REGULAR_EXPRESSION  ADDITIONAL_MAKE_CLEAN_FILES
set_directory_properties(PROPERTIES prop1 value1 prop2 value2)
```

set_property　#在给定的作用域内设置一个命名的属性

```
set_property(<GLOBAL |
             DIRECTORY [dir] |
             TARGET [target ...] |
             SOURCE [src1 ...] |
             TEST [test1 ...] |
             CACHE [entry1 ...]>
              [APPEND]
              PROPERTY <name> [value ...])
# 第一个参数决定了属性可以影响的作用域,必须为以下值：
# GLOBAL 全局作作用域,不接受名字
# DIRECTORY 默认为当前路径，但是同样也可以用[dir]指定路径
# TARGET 目标作用，可以是0个或多个已有的目标
# SOURCE 源作用域，  可以是0个过多个源文件
# TEST 测试作用域，可以是0个或多个已有的测试
# CACHE 必须指定0个或多个cache中已有的条目
PROPERTY参数是必须的
```

循环：CMake 中的循环有两种：foreach()...endforeach()和 while()...endwhile()

```
set(mylist "a" "b" c "d")
foreach(_var ${mylist})
    message("当前变量是：${_var}")
endforeach()
```

```
set(result 0)
foreach(_var RANGE 0 100)
    math(EXPR result "${result}+${_var}")
endforeach()
message("from 0 plus to 100 is:${result}")
//计算从0一直加到100的结果
```

条件判断：

```
#CMake中，判断的用法如下：
if(expression)
    # then section.
    COMMAND1(ARGS ...)
    COMMAND2(ARGS ...)
    ...
elseif(expression2)
    # elseif section.
    COMMAND1(ARGS ...)
    COMMAND2(ARGS ...)
    ...
else(expression)
    # else section.
    COMMAND1(ARGS ...)
    COMMAND2(ARGS ...)
    ...
endif(expression)
```

注意：对于 expression 里面的值，True if the constant is 1, ON, YES, TRUE, Y, or a non-zero number. False if the constant is 0, OFF, NO, FALSE, N, IGNORE, "", or ends in the suffix '-NOTFOUND'.如下

```
if(WIN32)
    message("this operation platform is windows")
elseif(UNIX)
    message("this operation platform is Linux")
endif()
```

CMake 中也有宏和函数的概念，关键字分别为"macro"和"function"，CMake 中的函数("function")与宏唯一的区别就在于，函数不能像宏那样直接将计算结果传出来，并且函数中的变量是局部的，而宏中的变量在外面也可以被访问到。

```
# 宏
macro( [arg1 [arg2 [arg3 ...]]])
    COMMAND1(ARGS ...)
    COMMAND2(ARGS ...)
    ...
endmacro()


# 函数
function( [arg1 [arg2 [arg3 ...]]])
    COMMAND1(ARGS ...)
    COMMAND2(ARGS ...)
    ...
endfunction()
```

```
macro(sum outvar)
    set(_args ${ARGN})
    set(result 0)

    foreach(_var ${_args})
        math(EXPR result "${result}+${_var}")
    endforeach()

    set(${outvar} ${result})
endmacro()
sum(addResult 1 2 3 4 5)
message("Result is :${addResult}")
```

更多相关命令:

https://gitlab.kitware.com/cmake/community/wikis/home

CMake 实战

推荐教程:

CMake 官网教程地址：https://cmake.org/cmake-tutorial/

中文教程：http://www.hahack.com/codes/cmake/

源码：https://github.com/wzpan/cmake-demo

注意:

1、每一个需要进行 cmake 操作的目录下面，都必须存在文件 CMakeLists.txt 。

2、cmake 指令不区分大小写。本文为了醒目，笔者把 cmake 指令都作大写处理。

3、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；

4、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；

一：基本开始_从一个 demo 开始

1，新建项目文件夹，创建 cpp 文件

```
mkdir test
cd test
touch main.cpp
touch CMakeLists.txt
```

2，创建 CMakeLists.txt ，内容如下：

```
# 用于说明CMake最低版本要求。
cmake_minimum_required (VERSION 2.6)

# 指定项目名,这里项目名字为GOC
project (GOC)

# Demo是最终的可执行文件名，main.cpp生成该可执行文件的源文件
add_executable(Demo main.cpp)
```

3，执行以下命令

```
cmake .
make
./Demo
```

Cmake .命令后生成了 makefile，make 是执行 makefile，

上面就是单个源文件的 demo，后面多个源文件，自定义编译选项，安装和测试

支持 gdb，添加环境检查，添加版本号，生成安装包等几个 demo，可以查看上述的网址进行学习。

与其它相似工具的对比

autotools

优点：历史悠久，使用广泛；可以自动生成 Makefile 文件

缺点：它需要有支持 shell 和 m4；只支持 Unix 平台，而 Visual Studio,Borland 等在 Win32 平台上的都不支持；依赖发现大部分依赖手工操作；生成的文件很大，一般很难看明白。

Jam

优点：跨平台；支持平行连接

缺点：最初的实现很粗糙；依赖发现机制大部分依赖手工

SCons

优点：跨平台；

缺点：可扩展性不好；