CmakeLists学习笔记

最常用的指令

```
# 本CMakeLists.txt的project名称
# 会自动创建两个变量，PROJECT_SOURCE_DIR和PROJECT_NAME
# ${PROJECT_SOURCE_DIR}：本CMakeLists.txt所在的文件夹路径
# ${PROJECT_NAME}：本CMakeLists.txt的project名称
project(xxx)

# 获取路径下所有的.cpp/.c/.cc文件，并赋值给变量中
aux_source_directory(路径 变量)

# 给文件名/路径名或其他字符串起别名，用${变量}获取变量内容
set(变量 文件名/路径/...)

# 添加编译选项
add_definitions(编译选项)

# 打印消息
message(消息)

# 编译子文件夹的CMakeLists.txt
add_subdirectory(子文件夹名称)

# 将.cpp/.c/.cc文件生成.a静态库
# 注意，库文件名称通常为libxxx.so，在这里只要写xxx即可
add_library(库文件名称 STATIC 文件)

# 将.cpp/.c/.cc文件生成可执行文件
add_executable(可执行文件名称 文件)

# 规定.h头文件路径
include_directories(路径)

# 规定.so/.a库文件路径
link_directories(路径)

# 对add_library或add_executable生成的文件进行链接操作
# 注意，库文件名称通常为libxxx.so，在这里只要写xxx即可
target_link_libraries(库文件名称/可执行文件名称 链接的库文件名称)
```

基本结构

```
project(xxx)                                    #必须

add_subdirectory(子文件夹名称)                  #父目录必须，子目录不必
```

```
add_library(库文件名称 STATIC 文件)                          #通常子目录(二选一)
add_executable(可执行文件名称 文件)                          #通常父目录(二选一)

include_directories(路径)                                   #必须
link_directories(路径)                                      #必须

target_link_libraries(库文件名称/可执行文件名称 链接的库文件名称)        #必须
```

除了这些之外，就是些set变量的语句，if判断的语句，或者其他编译选项的语句，但基本结构都是这样的。

步骤

CMakeLists.txt的创建

在需要进行编译的文件夹内编写CMakeLists.txt，即含有.cpp/.c/.cc的文件夹内

CMakeLists.txt的编写

如果项目的CMakeLists.txt的文件数量是2个，目录层次结构为上下层关系。通常的解决方案，就是将下层目录编译成一个静态库文件，让上层目录直接读取和调用，而上层目录就直接生成一个可执行文件。

上层CMakeLists.txt的内容为：

```
cmake_minimum_required(VERSION 3.0)
project(example_person)

# 如果代码需要支持C++11，就直接加上这句
SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++0x")
# 如果想要生成的可执行文件拥有符号表，可以gdb调试，就直接加上这句
add_definitions("-Wall -g")

# 设置变量，下面的代码都可以用到
set(GOOGLE_PROTOBUF_DIR ${PROJECT_SOURCE_DIR}/protobuf)
set(PROTO_PB_DIR ${PROJECT_SOURCE_DIR}/proto_pb2)
set(PROTO_BUF_DIR ${PROJECT_SOURCE_DIR}/proto_buf)

# 编译子文件夹的CMakeLists.txt
add_subdirectory(proto_pb2)

# 规定.h头文件路径
include_directories(${PROJECT_SOURCE_DIR} ${PROTO_PB_DIR} ${PROTO_BUF_DIR}
)

# 生成可执行文件
add_executable(${PROJECT_NAME} example_person.cpp )

# 链接操作
target_link_libraries(${PROJECT_NAME} general_pb2)
```

如果是初学者，这一段可能看不懂两个地方，第一是链接操作的general_pb2，第二是按照上文的CMakeLists.txt的流程，并没有规定link_directories的库文件地址。这两个其实是一个道理，add_subdirectory起到的作用。

当运行到add_subdirectory这一句时，会先将子文件夹进行编译，而libgeneral_pb2.a是在子文件夹中生成出来的库文件。子文件夹运行完后，父文件夹就已经知道了libgeneral_pb2.a这个库，因而不需要link_directories了。

同时，另一方面，在add_subdirector之前set的各个变量，在子文件夹中是可以调用的。

CMakeLists.txt的编译

一般CMakeLists.txt是，在最顶层创建build文件夹，然后编译。即：

```
mkdir build && cd build
cmake ..
make
```

最终生成可执行文件example_person。

可以通过以下命令来运行该可执行文件：

```
./example_person
```