近期现学现做一个在 Ubuntu 环境下将 C++程序封装成动态.so 库文件和静态.a 库文件的小项目，期间不知道掉了多少坑，所以在这里记录下来与诸君共勉。

一、静态库和动态库

本质上来说库是一种可执行代码的二进制形式，可以被操作系统载入内存执行，库可分成静态库和动态库（共享库），在 Windows 系统下，分别对应着 xx.lib 和 xx.dll 文件，在 linux 系统下，分别是 xx.a 和 xx.so 文件。

1.静态库

静态库的命名形式是 libname.a.静态库的代码在编译过程中已经被载入可执行程序，因此体积较大。它的优点是，编译后的执行程序不需要外部的函数库支持，因为所有使用的函数都已经被编译进可执行文件了。同样它的不足，如果静态函数库改变了，那么你的程序必须重新编译，而且体积也较大。

2.动态库

动态库名字一般是 libname.so.相对于静态函数库，共享库的代码是在可执行程序运行时才载入内存的，在编译过程中仅简单的引用，因此代码体积较小。由于函数库没有被整合进你的程序，而是程序运行时动态申请并调用，所以程序的运行环境中必须提供相应的库。动态函数库的改变并不影响你的程序，所以动态函数库的升级比较方便。而且如果多个应用程序都要使用同一函数库，动态库就非常适合，可以减少应用程序的体积。

静态库和动态库的主要区别在于：静态库是在程序编译时被链接到目标代码中，而动态库是在程序运行时才被载入。

二、生成动态库和静态库

一开始找的 MTCNN 的源代码是依赖于 opencv 和 openblas 环境，说起配置环境又是一把辛酸泪，在这里不再赘述，源码中头文件和 cpp 文件数量不多，所以采用的是用 g++的方式生成库文件，可是在测试的时候，能够在没有配置 opencv 的环境下直接调用程序所需要的库文件，但是却不能在没有 openblas 的环境下调用 libopenblas 库文件，所以当时就放弃了这种方案。

1.用 g++方式分别创建静态和动态库文件

在 linux 系统下，是用命令 ar 处理 A.o 目标文件生成静态库文件，需要指令如下：

```
1.g++ -c  A.cpp  -o A.o

2.ar -cr libA.a A.o

3.ar -r libABC.a  *.o
```

第一条指令是编译 A.cpp 生成 A.o 文件

第二条指令是生成静态库文件，在-cr 后面的参数就是库文件的名字

第三条指令是将目录下的所有的.o 文件合并生成静态库

在 linux 下编译时，通过 -shared 参数可以生成动态库.so 文件，如下：

```
g++ -shared -fPIC -o libA.so A.o
```

-shared 该选项指定生成动态连接库，不用该标志外部程序无法连接。相当于一个可执行文件
-fPIC：表示编译为位置独立的代码，不用此选项的话编译后的代码是位置相关的，所以动态载入时是通过代码拷贝的方式来满足不同进程的需要，而不能达到真 正代码段共享的目的。

2.使用 cmake 方式创建静态库和动态库文件

很幸运地是，我们又找到了一个不依赖任何第三方库的 C++源程序代码，可是此源码的头文件和 CPP 文件的数量巨大，而且代码具有层次感，其中还有子文件夹，所以在这个时候，用 Cmake 方式创建库文件是很高效间接的手段。

采用 out-of-source 编译的方式，按照习惯，建立一个 build 目录，将源程序文件放入 build 目录下，并在 build 目录下编写 CMakeLists.txt，这个文件是 cmake 的构建定义文件，文件名是大小写相关的。为了能同时生成动态库文件和静态库文件，CMakeLists.txt 文件中的相应内容如下：

```
1.add_library(name SHARED source1, source2, ..., sourceN)

2.add_library(name_static STATIC source1, source2, ... , sourceN)

3.set_target_properties(name_static PROPERTIES OUTPUT_NAME "name")

4.set_target_properties(name_static PROPERTIES CLEAN_DIRECT_OUTPUT 1)

5.set_target_properties(name PROPERTIES CLEAN_DIRECT_OUTPUT 1)

6.set_target_properties(name PROPERTIES VERSION 1.2 SOVERSION 1)

7.install(TARGETS name name_static

         LIBRARY DESTINATION lib

         ARCHIVE DESTINATION lib)

8.install(DIRECTORY ${titile_H} DESTINATION include/tH)
```

第一条指令是生成动态库（扩展名为.so），类型关键字是 SHARED，并不需要写全 libname.so，只需要填写 name 即可，cmake 系统会自动生成 libname.so。

第二条指令是在支持动态库的基础上为工程添加一个静态库，因为静态库和动态库同名时，构建静态库的指令是无效的，所以把上面的 name 修改为 name_static，就可以构建一个 libname_static 的静态库；然而我们需要的是名字相同的静态库和动态库，因为 target 的唯一性，所以就不能通过 add_library 指令实现，所以用到第三条指令

第三条指令是为了能够同时得到 libname.so/libname.a 两个库文件，但是因为 cmake 在构建一个新的 target 时，会尝试清理掉具有相同命名的库文件，所以，在构建 libname.a 的时候会将 libname.so 库文件清理掉，因此需要再次使用 set_target_properties 定义的 CLEAN_DIRECT_OUTPUT 属性，如第四条和第五条指令所示，至此，我们再次进行构建，就会发现在目录中同时生成 libname.so 动态库文件和 libname.a 静态库文件

第六条指令是因为按照规则，动态库是应当包含一个版本号的， 为了实现动态库版本号，仍然需要使用 SET_TARGET_PROPERTIES 指令，其中 VERSON 指代动态库版本，SOVERSION 指代 API 版本。

第七条指令是将动态库和静态库文件安装到系统目录，才能够真正地让其他人开发使用，我们将库文件安装到<prefix>/lib 目录下

第八条指令是将头文件安装到<prefix>/include/tH 目录中。

在终端进入 build 目录的上级目录，输入命令行，命令如下：

```
cmake build

make

sudo make install
```

至此，我们就可以将头文件和库文件分别安装到系统目录/usr/local/include/tH/和 usr/local/lib 中了。

三、外部引用动态库和静态库和头文件

构建和安装动态库和静态库之后，为了测试库文件是否被外部调用，需要编写源文件 main.cpp 进行函数调用测试。同样，我们还是使用 cmake 方式进行编译

3.1 外部引用静态库文件

```
1.INCLUDE_DIRECTORIES(头文件在系统中的位置)

2.ADD_EXECUTABLE(main source/main.cpp)

3.TARGET_LINK_LIBRARIES(main libfaceDetection.a)
```

第一条指令是引用头文件搜索路径

第二条指令的作用是生成一个名为 main 的可执行文件

第三条指令是位 target 添加静态库

3.2 外部引用动态库文件

因为编译安装将动态库安装到/usr/local/lib 目录下，对于动态库的外部引用有些麻烦，稍后补上