**cmake 构建多文件多目录项目**

一：所使用的环境

cmake version 3.5.1

Linux lei-virtual-machine 4.13.0-39-generic #44~16.04.1-Ubuntu SMP Thu Apr 5 16:43:10 UTC 2018 x86\_64 x86\_64 x86\_64 GNU/Linux

gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.9)

二： cmake 常用的环境变量

Cmake 的常用的环境变量可以参考下面的这边博客

<https://blog.csdn.net/gubenpeiyuan/article/details/8667279>

下面主要列举我常用的几个变量：

CMAKE\_C\_FLAGS ：用来设置gcc 编译c文件的编译选项

CMAKE\_CXX\_FLAGS ： 用来设置g++编译c++ 文件的编译选项

CMAKE\_CURRENT\_SOURCE\_DIR： 表示的是所在的当前目录

三： cmake 常用的几个函数

1、ADD\_DEFINITIONS(-std=c++11) 用来设置g++或者gcc 的编译选项

利用此函数可以添加-g选项如ADD\_DEFINITIONS(-g)，否则无法调试编译过的代码

2、set(CMAKE\_C\_FLAGS "-fPIC") 用来设置变量的值

3、MESSAGE(消息类型 消息内容) 用来打印消息的类似shell的echo

message("需要安装的头文件的目录" ${include\_path})

1. find\_package (OpenCV REQUIRED )

此函数主要是用于查找相应软件的配置文件 OpenCVConfig.cmake，OpenCVModules.cmake 利用配置文件的中的路径查找软件的头文件和库文件。路径查找的方式类似与Linux 库的查找方式。

1. file(GLOB include\_path "${CMAKE\_CURRENT\_SOURCE\_DIR}/\*.h")

获取当前目录下的所有的头文件到 include\_path 变量中

1. install(FILES ${include\_path} DESTINATION include)

将变量中的文件安装到系统的include目录下

install(TARGETS scanlib

RUNTIME DESTINATION bin

LIBRARY DESTINATION lib

ARCHIVE DESTINATION lib

)

将文件安装到相应的目录下

1. add\_subdirectory(utils)

使用该函数在编译的时候会进入到子目录下调用子目录的cmakeList.txt 文件，这对于程序的分层很有作用，这个在构造项目的编译环境很有作用。

8、SET\_TARGET\_PROPERTIES(zbar\_b PROPERTIES LINKER\_LANGUAGE CXX)

系统会自动识别文件的类型，如果文件的实现的方式都写在头文件中可以采用上述函数进行指定编译的语言CXX 表示的是c++语言， C 表示c语言。

9、使用linux系统的环境变量要使用$ENV{HOME} 进行获取

IMPORTED\_LOCATION\_RELEASE "$ENV{HOME}/opencv-2.4.9/lib/libopencv\_calib3d.so.2.4.9"

四：头文件路劲，添加库，添加库文件路径

1、添加头文件路径：

include\_directories(./ ./barcode ./censor ./DataCollector ./optimizeObjBlock ./template ./third/jsoncpp ./third/jsoncpp/include ./third/cvBlob ./third ./thread ./utils)

2、添加库路径：

对应的函数叫LINK\_DIRECTORIES

3、添加库文件：

target\_link\_libraries(scanlib barcode censor DataCollector optimizeObjBlock template jsoncpp zxing cvblob utils tesseract lept) 注：去掉前后缀

五：cmakeList.txt 的目标文件

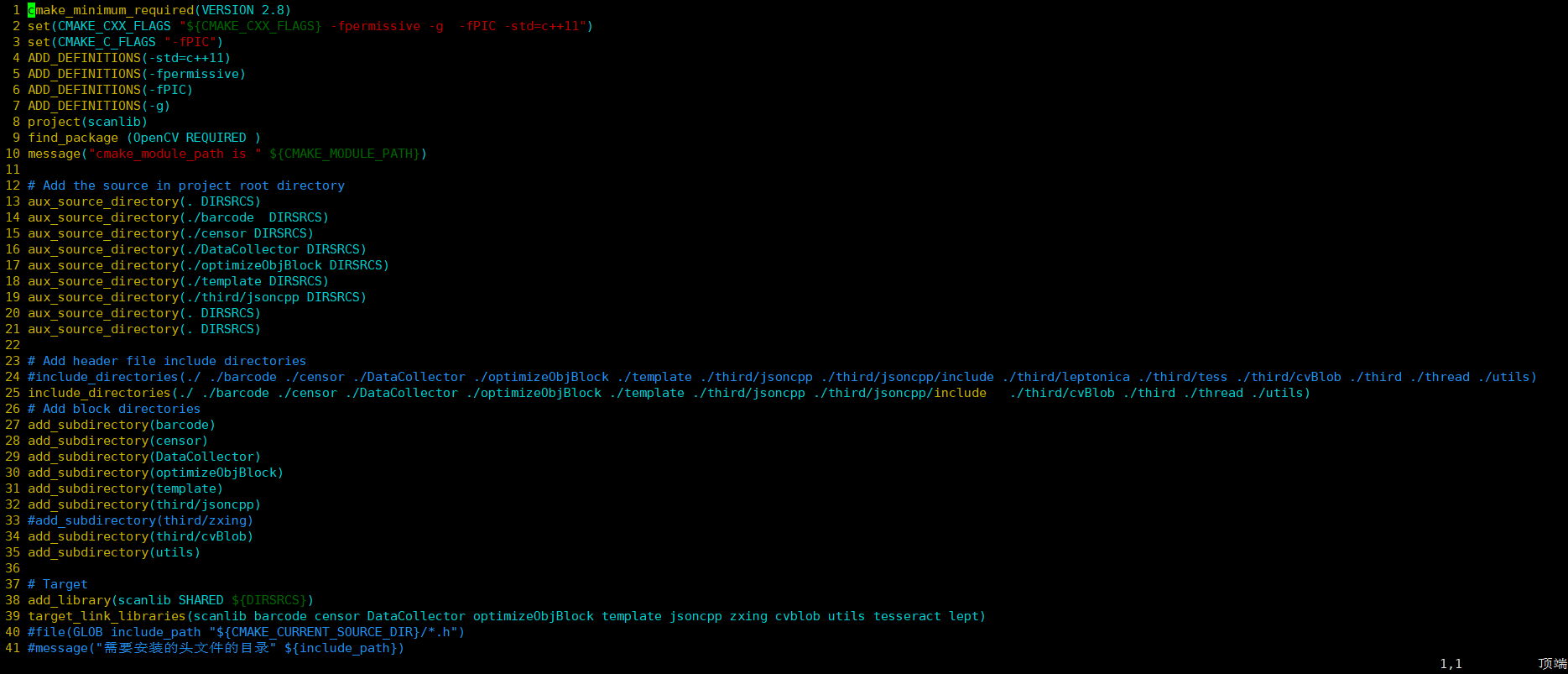
add\_library(scanlib SHARED ${DIRSRCS}) 这个是使的此cmakeList.txt 编译出scanlib的动态库

add\_executable(libtest ${DIRSRCS\_LIBTEST}) 这个是生成libtest 的可执行文件

add\_library(censor STATIC ${DIR\_CENSOR\_SRCS}) 这个是生成censor的静态库

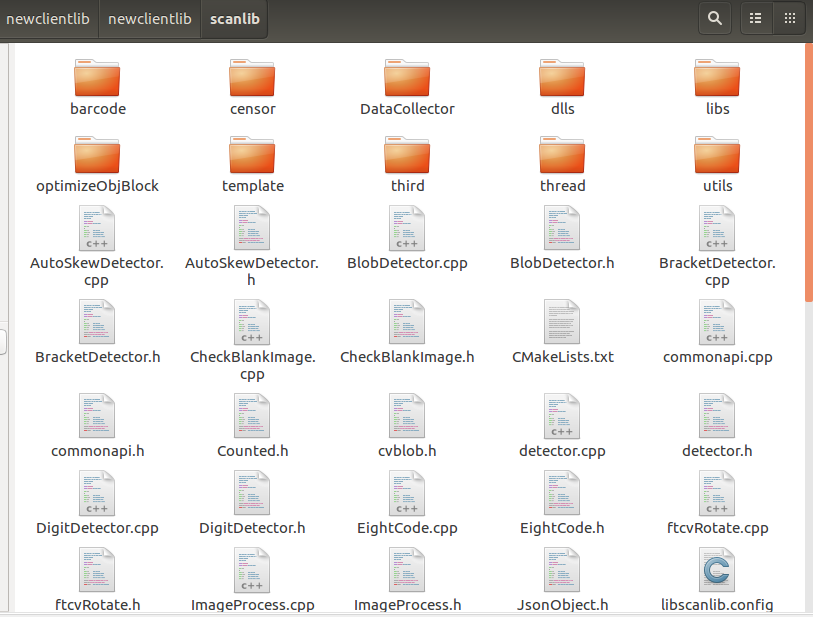
我试过这三个必须要有一个目标文件，cmakelist.txt 才能编译

上述讲了很多的基本知识，下面我们看些cmake 需要的指导文件长什么样

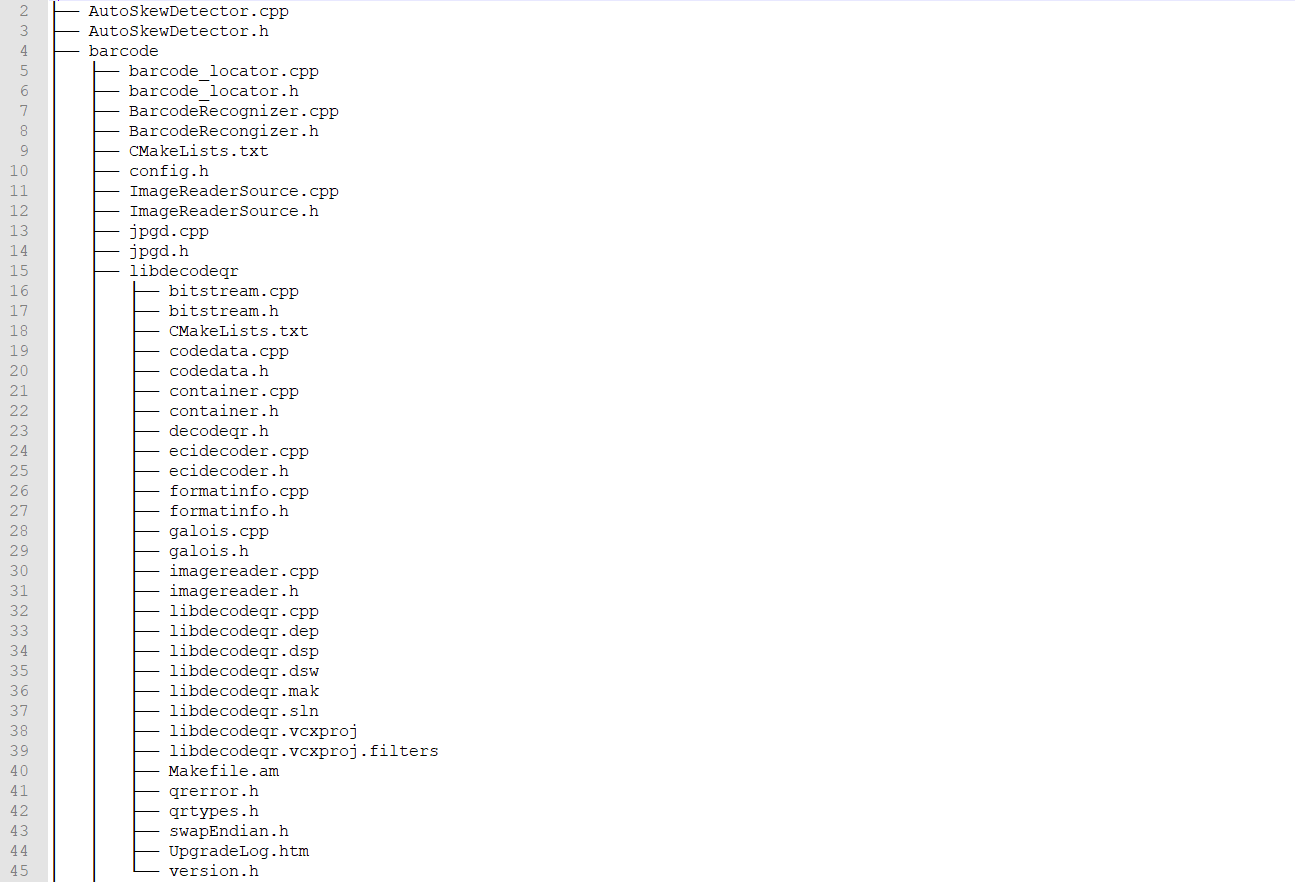


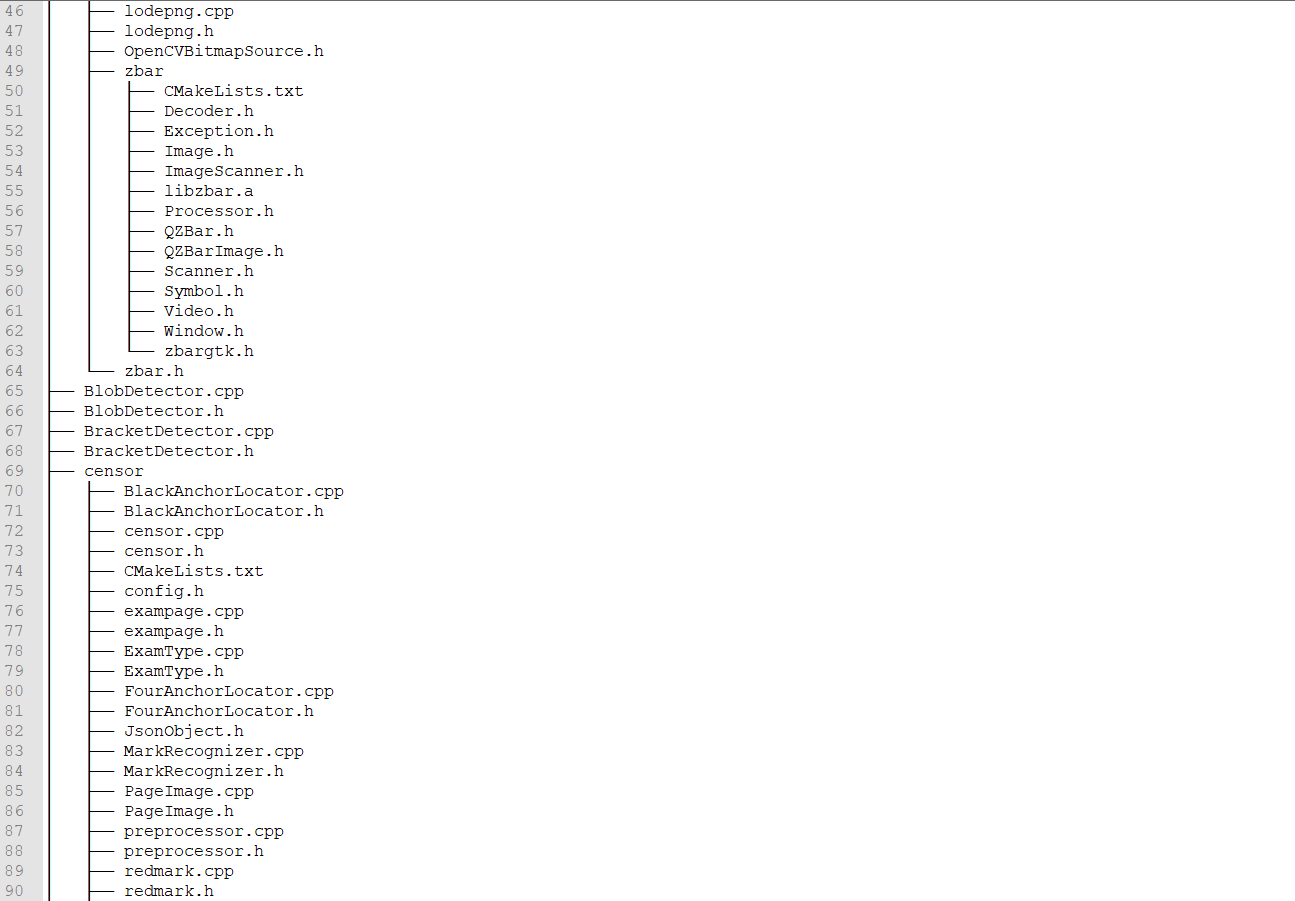
具体的文件为[1]

六 cmake 多文件多目录的scanlib 编译实例（包含多个开源 库）



Scanlib 的目录结构：





这里只列出了一部分的目录结构，有需要的可以查看整个scanlib目录结构[2]

可以发现在主要的目录结构下都会cmakeList.txt 文件，主要的一个思想就是将自己写的子目录的程序编译成静态库，外部的第三方软件则以动态库的形式存在，然后利用生成的子目录的静态库和第三方动态库在主目录下生成scanlib.so 库，最后利用libtest 进行测试是有效的。具体的实现方式参考scanlib下的cmakelist.txt和各个子目录下cmakelist.txt 文件。此时这个add\_subdirectory()非常有用，它会进入到子目录下找到相应的cmakelist.txt 文件进行编译

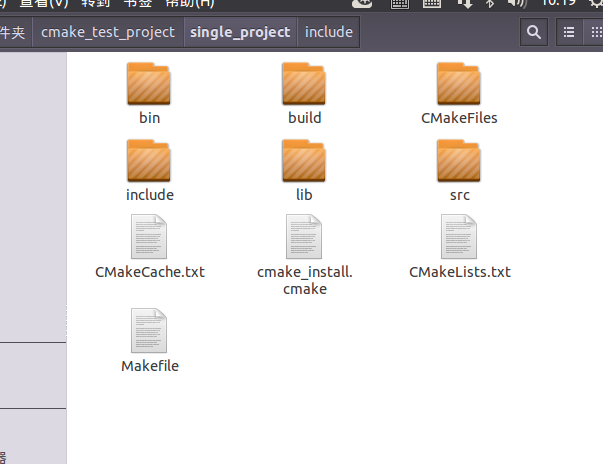
**上面已经讲解了编写CMakeList的基本技巧和构建scanlib项目实现的例子，对于新手可以参考下面的例子进行快速的学习进（它和scanlib 构建框架都是一致的，但是专注于项目的构建）**

1. **cmake 多文件多目录的学习的简易例子（供学习使用 ）**

1、第一个是我们经常会将头文件和实现文件以及lib库文件分开存放的一个例子实现的例子

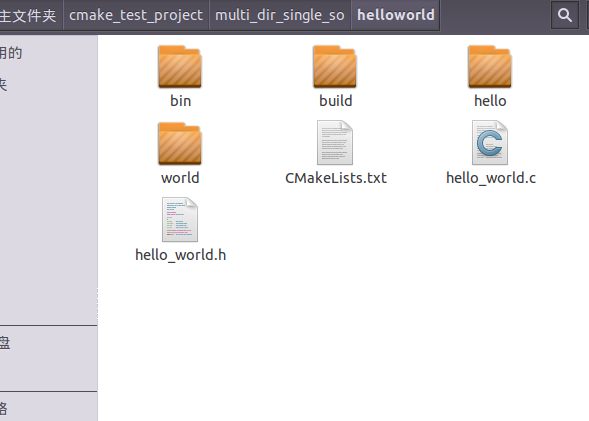
目录结构如下：

整个的实现方式见[3]



1. 这个例子是和我们scanlib 匹配的例子，它是按功能模块进行存放，通常一个子目录会表示一个特定的意思，很适合分成结构。

目录结构如下：



其中hello目录和word 目录分别表示一个模块，里面既有头文件也有实现文件。通常这是软件排布的最合理的方式，建议按此方式构建项目和编码。

整体的实现方式见[4]

基础知识以及整体的构建方式和例子都已经给出了，下面吐吐槽自己的所遇到的问题和解决方式。

1. **构建项目所思所想以及困惑的地方**
2. 面对大型的项目的时候，也许你已经构建完成了但是你的代码运行出的结构却不是你想象的那样，当你去调试的时候却发现不能调，要重新添加编译参数，修改各个目录下的cmakeList.txt 文件费时费力，建议总是使用ADD\_DEFINITIONS添加-g选项和-std=c++11.
3. 如果你构建别人写的代码，一个子目录下只有头文件你怎么编写cmakeList.txt (定义和实现都写在头文件中)?

①利用file(GLOB path\_file\_1 "${CMAKE\_CURRENT\_SOURCE\_DIR}/\*.h")就能将所有目录下的头写入到变量path\_file\_1 中这样就获取到了所有的头文件

②SET\_TARGET\_PROPERTIES(B\_lib PROPERTIES LINKER\_LANGUAGE CXX) 利用这个函数告诉编译器以什么方式进行编译。

1. 如果你构建的项目应用了第三方库，而第三方库采用的makefile 的编译架构。怎么和现在的cmake 编译方式融合

第一种方式：编译第三方库并安装，然后修改cmakeList.txt 的库文件和头文件路径然后再进行编译。（注：一定不要去修改第三方库的编译架构将makefile 改变成cmakeList.txt)

或者下个cmake 编译版本的库也可以一起进行编译（注：可能性比较小）

第二种方式：修改你项目的cmake 方式为makefile 构建方式然后融合第三方的库，可以实现简单的编译。

我当时遇到的问题是我已经利用cmake 方式架设编译过程完成了大部分，但是缺少了三个库zbar,zxing以及tesseract库，有两个是第三方库是利用makefile 构建，一个是cmake构建。一般来说不要试图去修改第三方的库的构建方式，最后只能采用第一种的实现方式编写脚本去实现。

1. 如果你构建项目的话有多个子目录，那子目录是生成动态库还是静态库呢？

我当时思考这个问题，采用动态库它是运行的时候再链接程序可以是得到包比较小，但是运行的时候关注的库会比较多，如果编译成静态库用起来会方便很多我们也不用关注那么多库它会将代码编译进来。最后我采用的是如果我们目录下的文件不是很多就编译成静态库，如果库文件比较大则编译成静态库。有两个需要注意的地方：

一、上层编译的动态库可以子目录的静态库进行生成，不要使用动态库生成静态库哦（需要这样的自行测试）

二、进行静态库编译时加上 -fPIC 选项，否则利用静态库生成的动态会有问题。

**[1] scanlib 项目顶层的CMakeLists.txt 文件**



**[2] scanlib 项目的整个目录结构**

****

**[3] 头文件，实现文件分开存放的例子**

****

**[4] 分层构建项目的例子**

