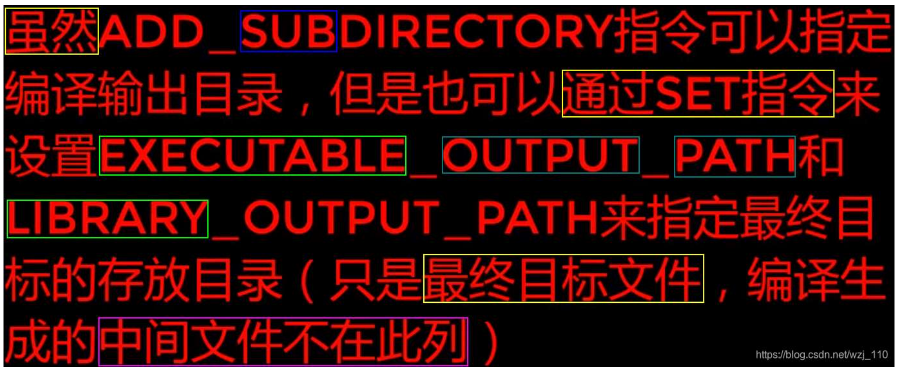
cmake(七)Cmake指定目标保存文件

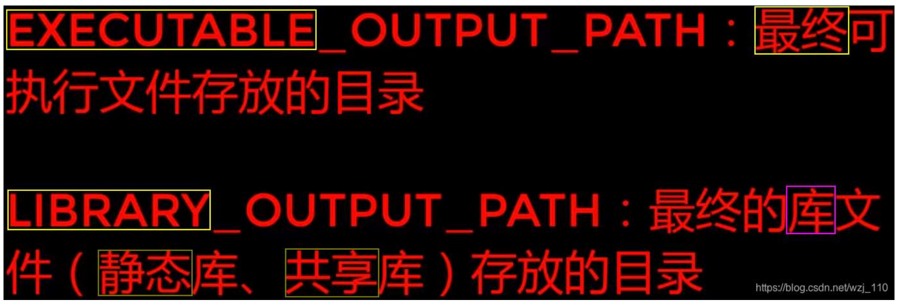
原创 wzj_110 于 2021-04-18 10:10:27 发布 阅读量1.1k 收藏 3 点赞数 1 版权
分类专栏： cmake DSL语言

cmake DSL语言 专栏收录该内容 38 篇文章 已订阅

一 两种方法保存编译输出的对比


虽然ADD_SUBDIRECTORY指令可以指定编译输出目录，但是也可以通过SET指令来设置EXECUTABLE_OUTPUT_PATH和LIBRARY_OUTPUT_PATH来指定最终目标的存放目录（只是最终目标文件，编译生成的中间文件不在此列）

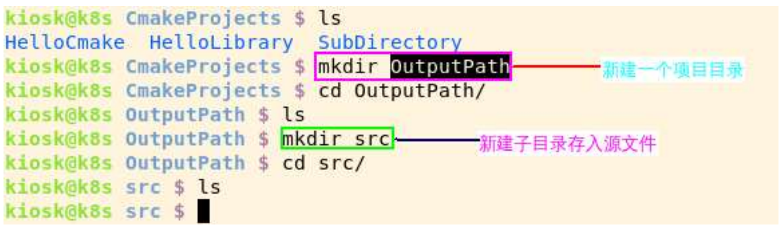
```
核心差异性： 'add_subdirectory'除了保存'最终可执行文件',还会保存'编译生成的中间件'
```


EXECUTABLE_OUTPUT_PATH：最终可执行文件存放的目录

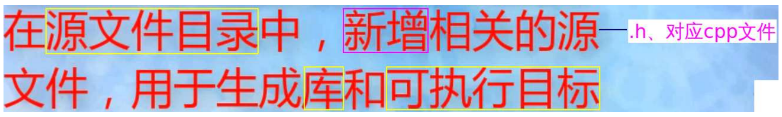
LIBRARY_OUTPUT_PATH：最终的库文件（静态库、共享库）存放的目录

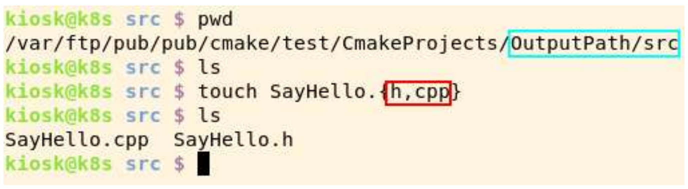
二 实践

① 项目初始化工作

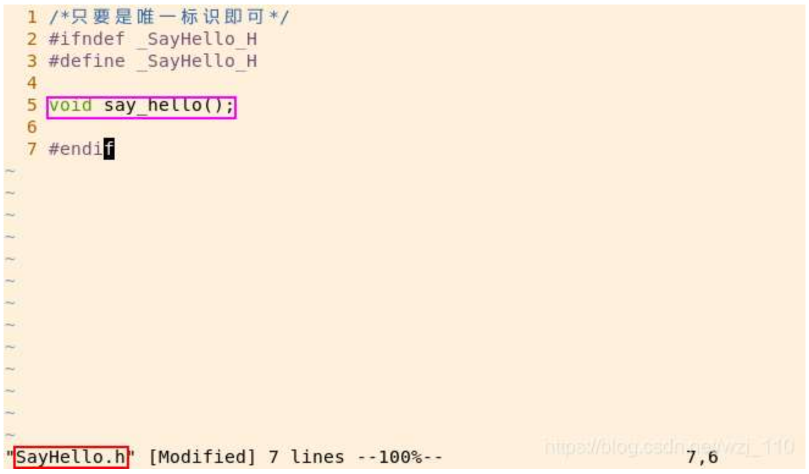

```
kiosk@k8s CmakeProjects $ ls
HelloCmake  HelloLibrary  SubDirectory
kiosk@k8s CmakeProjects $ mkdir OutputPath
kiosk@k8s CmakeProjects $ cd OutputPath/
kiosk@k8s OutputPath $ ls
kiosk@k8s OutputPath $ mkdir src
kiosk@k8s OutputPath $ cd src/
kiosk@k8s src $ ls
kiosk@k8s src $ 
```

② 新增相关源文件


在源文件目录中，新增相关的源文件，用于生成库和可执行目标


```
kiosk@k8s src $ pwd
/var/ftp/pub/pub/cmake/test/CmakeProjects/OutputPath/src
kiosk@k8s src $ ls
kiosk@k8s src $ touch SayHello.{h,cpp}
kiosk@k8s src $ ls
SayHello.cpp  SayHello.h
kiosk@k8s src $ 
```

```
说明： 声明一个'函数'
```



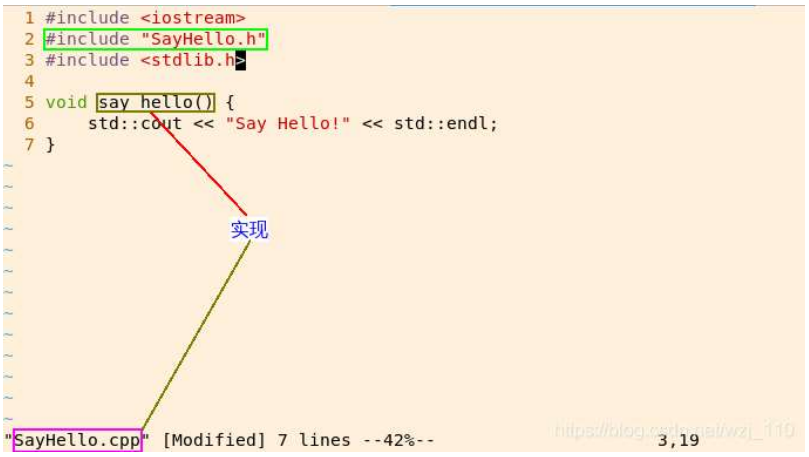
```
/*只要是唯一标识即可*/
#ifndef _SayHello_H
#define _SayHello_H

void say_hello();

#endif

"SayHello.h" [Modified] 7 lines --100%--                7,6
```

编辑与头文件对应的源文件，实现头文件中声明的函数



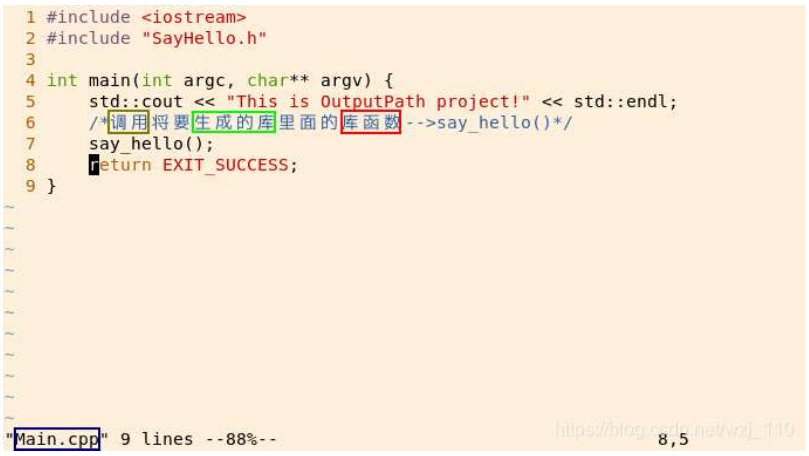
```
#include <iostream>
#include "SayHello.h"
#include <stdlib.h>

void say_hello() {
    std::cout << "Say Hello!" << std::endl;
}

"SayHello.cpp" [Modified] 7 lines --42%--          3,19
```

添加源文件用于可执行目标

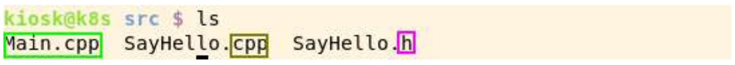
备注： 缺少'#include <stdlib.h>',进行'add'添加,否则后续'报错'



```
#include <iostream>
#include "SayHello.h"

int main(int argc, char** argv) {
    std::cout << "This is OutputPath project!" << std::endl;
    /*调用将要生成的库里面的库函数-->say_hello()*/
    say_hello();
    return EXIT_SUCCESS;
}

"Main.cpp" 9 lines --88%--                         8,5
```



```
kiosk@k8s src $ ls
Main.cpp  SayHello.cpp  SayHello.h
```

③ 添加CMakeLists.txt文件

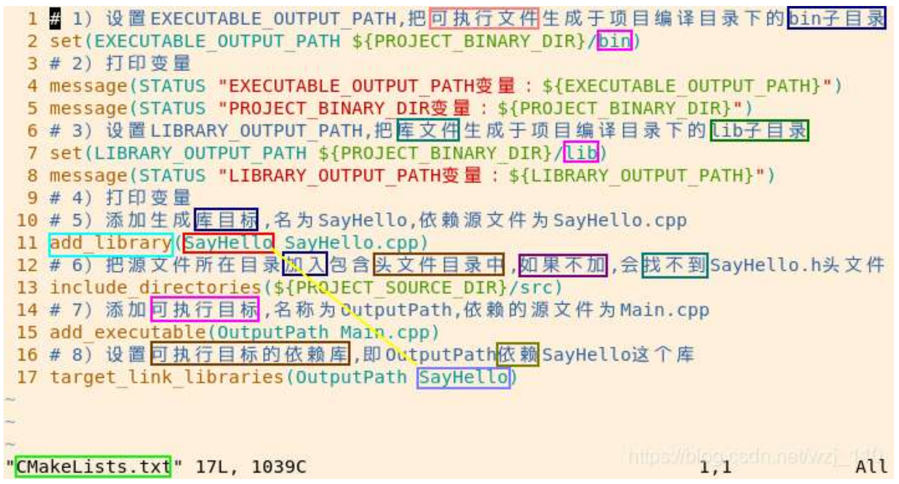
为此src目录添加CMakeLists.txt

```
备注： 'add_library'不指定'库类型'默认是'STTAIC' --> '静态类型'

补充： target_link_libraries'添加链接库'要在add_executable命令'下方',否则会'报错'

```

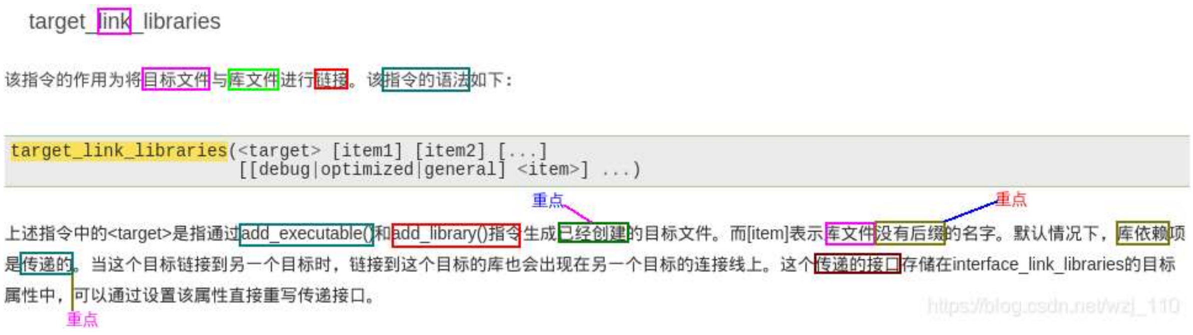
```
注意： 11、13、15、17'关系'
```



```
 1 # 1) 设置EXECUTABLE_OUTPUT_PATH,把可执行文件生成于项目编译目录下的bin子目录
 2 set(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR}/bin)
 3 # 2) 打印变量
 4 message(STATUS "EXECUTABLE_OUTPUT_PATH变量：${EXECUTABLE_OUTPUT_PATH}")
 5 message(STATUS "PROJECT_BINARY_DIR变量：${PROJECT_BINARY_DIR}")
 6 # 3) 设置LIBRARY_OUTPUT_PATH,把库文件生成于项目编译目录下的lib子目录
 7 set(LIBRARY_OUTPUT_PATH ${PROJECT_BINARY_DIR}/lib)
 8 message(STATUS "LIBRARY_OUTPUT_PATH变量：${LIBRARY_OUTPUT_PATH}")
 9 # 4) 打印变量
10 # 5) 添加生成库目标,名为SayHello,依赖源文件为SayHello.cpp
11 add_library(SayHello SayHello.cpp)
12 # 6) 把源文件所在目录加入包含头文件目录中,如果不加,会找不到SayHello.h头文件
13 include_directories(${PROJECT_SOURCE_DIR}/src)
14 # 7) 添加可执行目标,名称为OutputPath,依赖的源文件为Main.cpp
15 add_executable(OutputPath Main.cpp)
16 # 8) 设置可执行目标的依赖库,即OutputPath依赖SayHello这个库
17 target_link_libraries(OutputPath SayHello)
~
~
~
"CMakeLists.txt" 17L, 1039C                                   1,1           All
```

target_link_libraries

其它参考



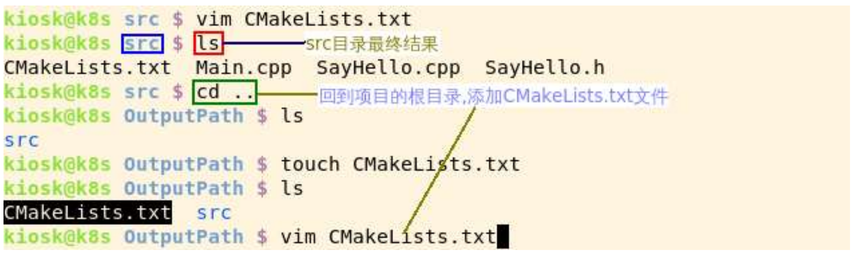
target_link_libraries

该指令的作用为将目标文件与库文件进行链接。该指令的语法如下：

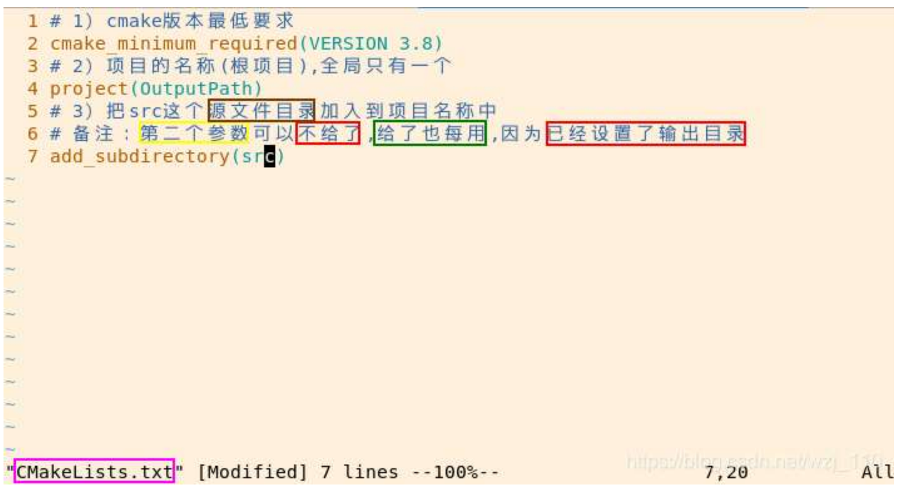
```
target_link_libraries(<target> [item1] [item2] [...]
                      [[debug|optimized|general] <item>] ...)
```

上述指令中的<target>是指通过add_executable()和add_library()指令生成已经创建的目标文件。而[item]表示库文件没有后缀的名字。默认情况下，库依赖项是传递的。当这个目标链接到另一个目标时，链接到这个目标的库也会出现在另一个目标的连接线上。这个传递的接口存储在interface_link_libraries的目标属性中，可以通过设置该属性直接重写传递接口。

④ 项目根目录添加CMakeLIsts.txt文件



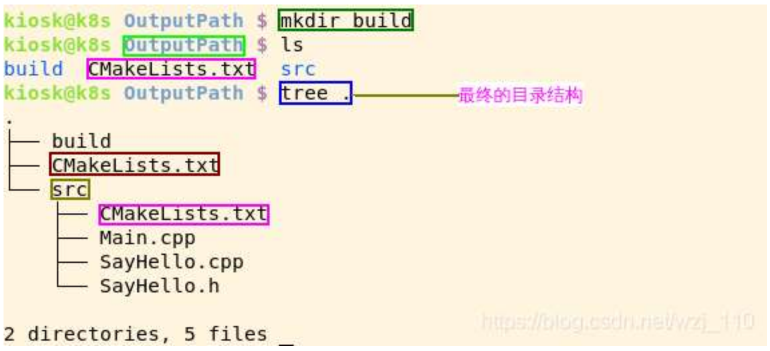
```
kiosk@k8s src $ vim CMakeLists.txt
kiosk@k8s src $ ls
CMakeLists.txt  Main.cpp  SayHello.cpp  SayHello.h
kiosk@k8s src $ cd ..
kiosk@k8s OutputPath $ ls
src
kiosk@k8s OutputPath $ touch CMakeLists.txt
kiosk@k8s OutputPath $ ls
CMakeLists.txt  src
kiosk@k8s OutputPath $ vim CMakeLists.txt
```



```
# 1) cmake版本最低要求
cmake_minimum_required(VERSION 3.8)
# 2) 项目的名称(根项目),全局只有一个
project(OutputPath)
# 3) 把src这个源文件目录加入到项目名称中
# 备注：第二个参数可以不给了,给了也每用,因为已经设置了输出目录
add_subdirectory(src)
~
"CMakeLists.txt" [Modified] 7 lines --100%--                  7,20          All
```

project解读

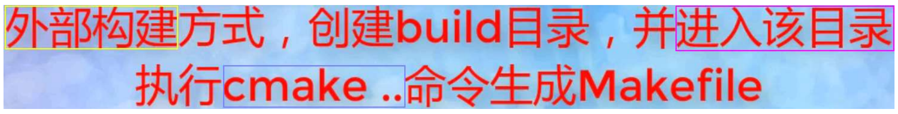
⑤ 外部构建



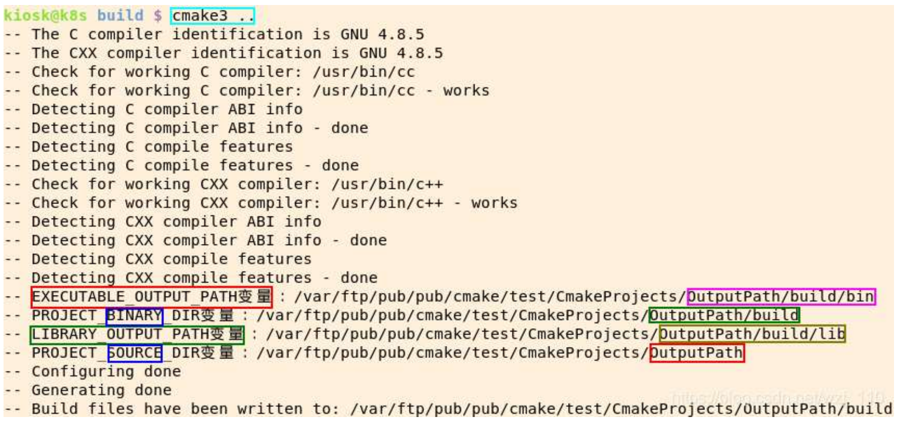
```
kiosk@k8s OutputPath $ mkdir build
kiosk@k8s OutputPath $ ls
build  CMakeLists.txt  src
kiosk@k8s OutputPath $ tree .
.
├── build
├── CMakeLists.txt
└── src
    ├── CMakeLists.txt
    ├── Main.cpp
    ├── SayHello.cpp
    └── SayHello.h

2 directories, 5 files _
```

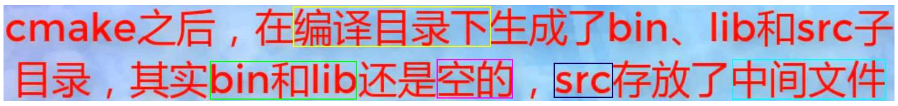
⑥ cmake ..



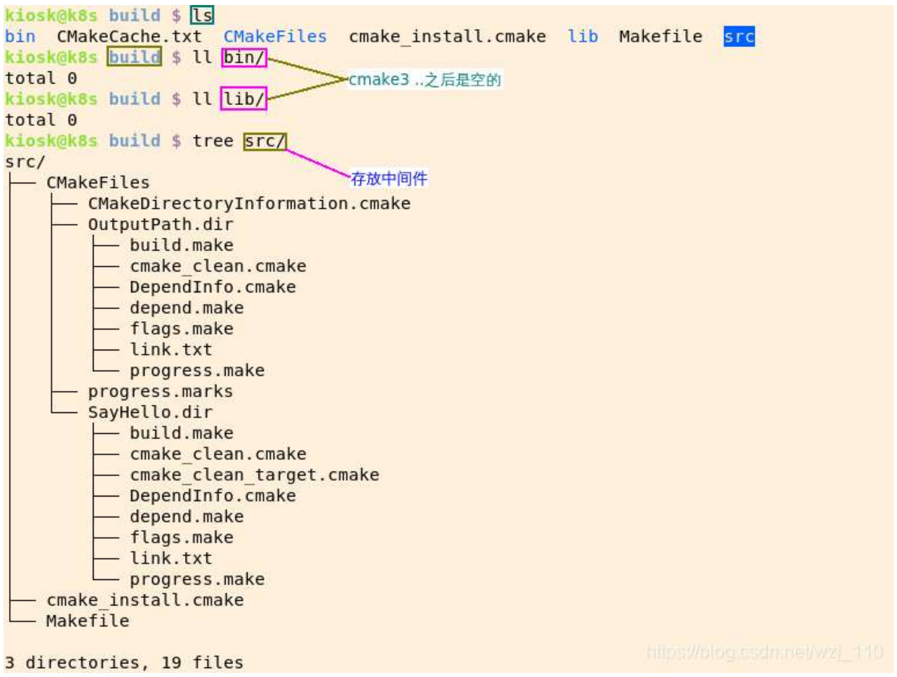
外部构建方式，创建build目录，并进入该目录执行cmake ..命令生成Makefile



```
kiosk@k8s build $ cmake3 ..
-- The C compiler identification is GNU 4.8.5
-- The CXX compiler identification is GNU 4.8.5
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc - works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ - works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- EXECUTABLE_OUTPUT_PATH变量 : /var/ftp/pub/pub/cmake/test/CmakeProjects/OutputPath/build/bin
-- PROJECT_BINARY_DIR变量 : /var/ftp/pub/pub/cmake/test/CmakeProjects/OutputPath/build
-- LIBRARY_OUTPUT_PATH变量 : /var/ftp/pub/pub/cmake/test/CmakeProjects/OutputPath/build/lib
-- PROJECT_SOURCE_DIR变量 : /var/ftp/pub/pub/cmake/test/CmakeProjects/OutputPath
-- Configuring done
-- Generating done
-- Build files have been written to: /var/ftp/pub/pub/cmake/test/CmakeProjects/OutputPath/build
```



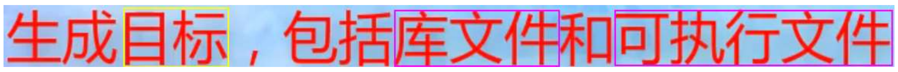
cmake之后，在编译目录下生成了bin、lib和src子目录，其实bin和lib还是空的，src存放了中间文件



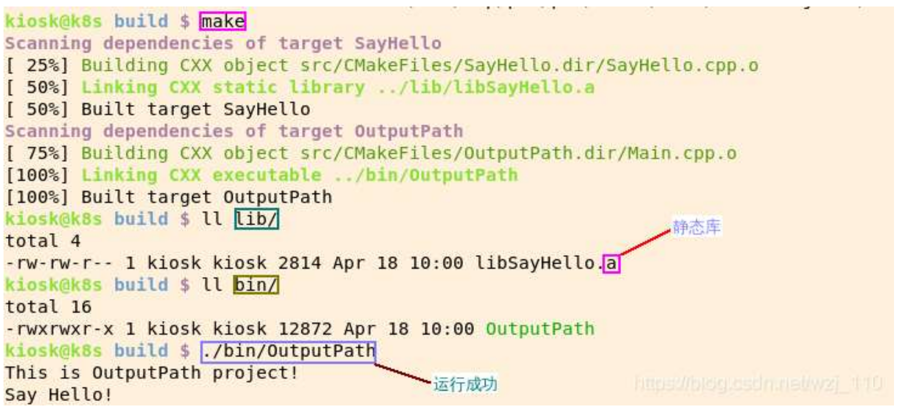
```
kiosk@k8s build $ ls
bin  CMakeCache.txt  CMakeFiles  cmake_install.cmake  lib  Makefile  src
kiosk@k8s build $ ll bin/
total 0
kiosk@k8s build $ ll lib/
total 0
kiosk@k8s build $ tree src/
src/
├── CMakeFiles
│   ├── CMakeDirectoryInformation.cmake
│   ├── OutputPath.dir
│   │   ├── build.make
│   │   ├── cmake_clean.cmake
│   │   ├── DependInfo.cmake
│   │   ├── depend.make
│   │   ├── flags.make
│   │   ├── link.txt
│   │   └── progress.make
│   ├── progress.marks
│   └── SayHello.dir
│       ├── build.make
│       ├── cmake_clean.cmake
│       ├── cmake_clean_target.cmake
│       ├── DependInfo.cmake
│       ├── depend.make
│       ├── flags.make
│       ├── link.txt
│       └── progress.make
├── cmake_install.cmake
└── Makefile

3 directories, 19 files
```

⑦ make



生成目标，包括库文件和可执行文件



```
kiosk@k8s build $ make
Scanning dependencies of target SayHello
[ 25%] Building CXX object src/CMakeFiles/SayHello.dir/SayHello.cpp.o
[ 50%] Linking CXX static library ../lib/libSayHello.a
[ 50%] Built target SayHello
Scanning dependencies of target OutputPath
[ 75%] Building CXX object src/CMakeFiles/OutputPath.dir/Main.cpp.o
[100%] Linking CXX executable ../bin/OutputPath
[100%] Built target OutputPath
kiosk@k8s build $ ll lib/
total 4
-rw-rw-r-- 1 kiosk kiosk 2814 Apr 18 10:00 libSayHello.a
kiosk@k8s build $ ll bin/
total 16
-rwxrwxr-x 1 kiosk kiosk 12872 Apr 18 10:00 OutputPath
kiosk@k8s build $ ./bin/OutputPath
This is OutputPath project!
Say Hello!
```