



cmake简易指南

本文版权属于重庆大学计算机学院刘骥，禁止转载

cmake简易指南

创建cmake工程

常用cmake指令

- add_executable
- add_library
- include_directories
- link_directories
- target_link_libraries
- add_definitions
- add_subdirectory

使用举例

- 编译库文件
- 使用库文件
- 子项目
- 使用第三方库

其他指令

创建cmake工程

随便建立一个目录（例如test），在根目录下创建 `CMakeLists.txt` 文件，内容如下：

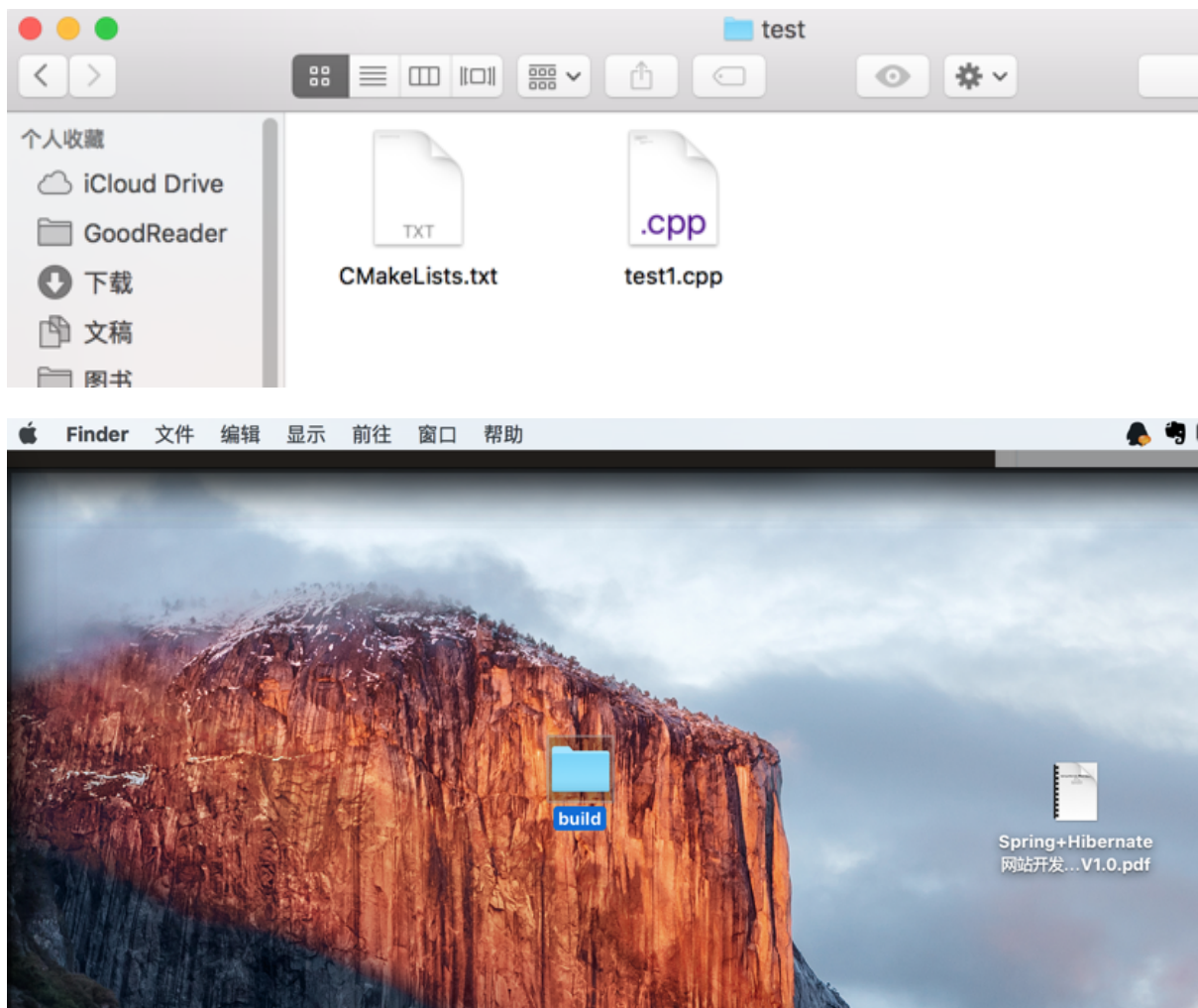
```
add_executable(test1 test1.cpp) #编译test1.cpp，生成可执行文件test1
```

然后编写 `test1.cpp`，内容随意。

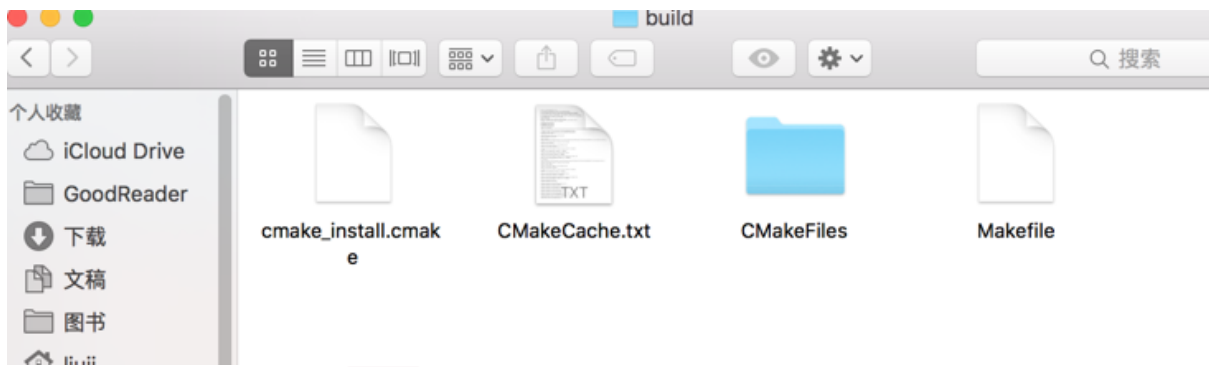
随便找一个目录（随便的意思是随你的便），命令行进入到这个目录，执行如下指令：

```
cmake cmake工程所在的目录
```

例如我在桌面找了一个目录 `build`，工程目录放在 `/Users/liuji/Projects/test`，操作过程如下图：



```
LiuJi-MacBook-Pro:build liuji$ cmake /Users/liuji/Projects/test
-- The C compiler identification is AppleClang 7.3.0.7030031
-- The CXX compiler identification is AppleClang 7.3.0.7030031
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/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: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/liuji/Desktop/build
```



最后在build目录下面执行 `make` 就可以构建程序，如下图：

```
LiuJi-MacBook-Pro:build liuji$ make
Scanning dependencies of target test1
[ 50%] Building CXX object CMakeFiles/test1.dir/test1.cpp.o
[100%] Linking CXX executable test1
[100%] Built target test1
LiuJi-MacBook-Pro:build liuji$ ./test1
test1
LiuJi-MacBook-Pro:build liuji$
```

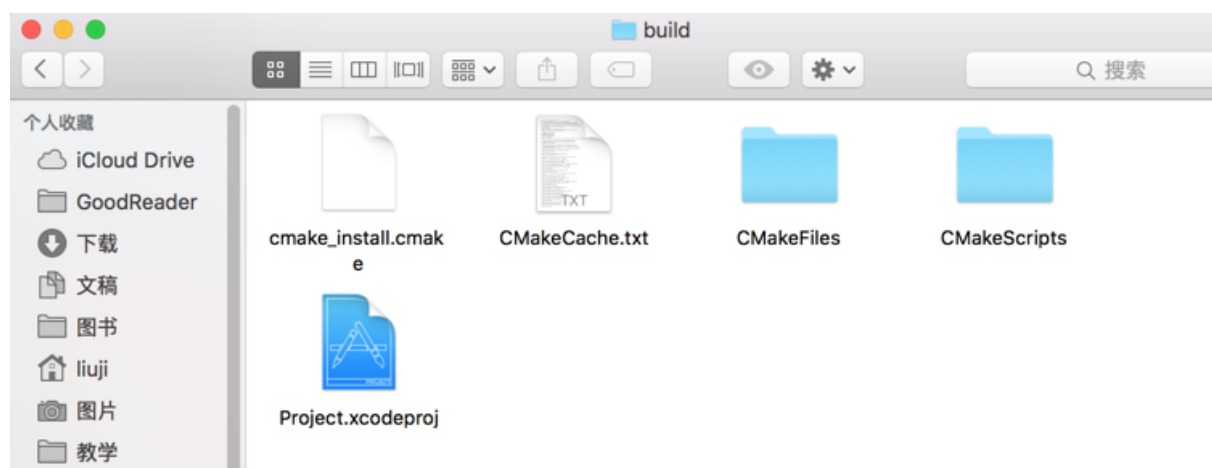
通过这个例子可以看出，cmake可以实现 **源代码目录** 和 **编译目录** 的分离。源代码是源代码，编译文件是编译文件，相互可以处于完全不同的目录。并且编译文件可以根据本机的设置动态生成。不仅如此，除了可以生成unix的makefile文件，也可以生成各种IDE的工程文件。`cmake -G IDE工程文件名 工程目录` 就可以实现这个功能。cmake支持的工程文件包括：

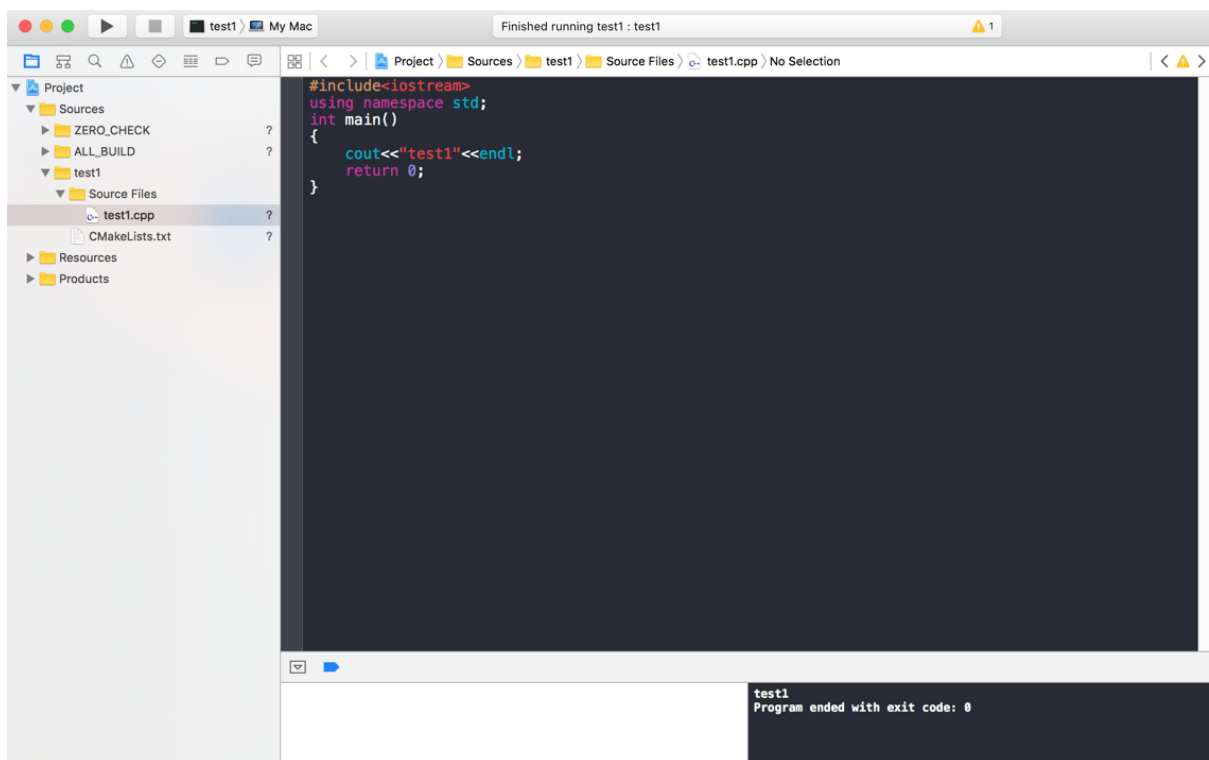
```
LiuJi-MacBook-Pro:build liuji$ cmake -G
CMake Error: No generator specified for -G

Generators
  Unix Makefiles           = Generates standard UNIX makefiles.
  Ninja                   = Generates build.ninja files.
  Xcode                   = Generate Xcode project files.
  CodeBlocks - Ninja      = Generates CodeBlocks project files.
  CodeBlocks - Unix Makefiles = Generates CodeBlocks project files.
  CodeLite - Ninja       = Generates CodeLite project files.
  CodeLite - Unix Makefiles = Generates CodeLite project files.
  Eclipse CDT4 - Ninja    = Generates Eclipse CDT 4.0 project files.
  Eclipse CDT4 - Unix Makefiles = Generates Eclipse CDT 4.0 project files.
  KDevelop3               = Generates KDevelop 3 project files.
  KDevelop3 - Unix Makefiles = Generates KDevelop 3 project files.
  Kate - Ninja            = Generates Kate project files.
  Kate - Unix Makefiles   = Generates Kate project files.
  Sublime Text 2 - Ninja  = Generates Sublime Text 2 project files.
  Sublime Text 2 - Unix Makefiles = Generates Sublime Text 2 project files.
```

下面我执行 `cmake -G Xcode` 生成xcode工程文件，如下：

```
LiujI-MacBook-Pro:build liuji$ cmake -G Xcode /Users/liuji/Projects/test
-- The C compiler identification is AppleClang 7.3.0.7030031
-- The CXX compiler identification is AppleClang 7.3.0.7030031
-- Check for working C compiler using: Xcode
-- Check for working C compiler using: Xcode -- 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 using: Xcode
-- Check for working CXX compiler using: Xcode -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/liuji/Desktop/build
```





从上面的例子可以看出，cmake可以实现 **跨平台和跨工具编译**。什么意思呢？如果有一份源代码，这份源代码想在windows和linux下编译，windows下用vs编译，linux下面用eclipse编译。如果没有cmake，就需要分别建立vs和eclipse的工程，并且针对windows和linux进行编译设置。这就非常麻烦，特别是vs和eclipse的源代码还得是两份，如果修改了其中一份，另外一份也需要跟着改。

有了cmake，这个困扰就没有了。我们只需要维持一份源代码，也不需要编写工程文件。需要在什么平台和IDE下编译就用cmake动态生成，真正实现 **write once, compile anywhere**。

常用cmake指令

add_executable

用于生成可执行文件，指令格式如下：

```
add_executable(可执行文件名 源文件1 源文件2 源文件3...)
```

add_library

用于生成库文件，指令格式如下：

```
add_library(库文件名 源文件1 源文件2 源文件3...)
```

include_directories

用于设置头文件目录，指令格式如下：

```
include_directories(路径1 路径2 路径3...)
```

link_directories

用于设置库文件目录，指令格式如下：

```
link_directories(路径1 路径2 路径3...)
```

target_link_libraries

用于设置链接的库，指令格式如下：

```
target_link_libraries(生成的可执行文件或者库文件名 链接库1 链接库2 链接库3...)
```

add_definitions

设置额外的编译选项，指令格式如下：

```
add_definitions(编译选项)
```

例如 `add_definitions(-DHELLO)`，则在编译时预定义宏 `HELLO`。

add_subdirectory

用于子项目编译，即将不同的项目放置在不同的目录下，每个目录下编写一个 `CMakeLists.txt`，项目的根目录就可以用 `add_subdirectory` 包含子项目。指令格式如下：

```
add_subdirectory(子项目所在目录 子项目输出目录)
```

如果不指定输出目录，那么输出目录与子项目名称相同。

使用举例

编译库文件

编写如下的头文件 `mylib.h`：

```
float add(float a,float b);
```

编写如下的源文件 `mylib.cpp`：

```
#include"mylib.h"
float add(float a,float b)
{
    return a+b;
}
```

编写如下的 `CMakeLists.txt`：

```
add_library(mylib mylib.cpp mylib.h)
```

生成执行cmake，生成库文件，如下：


```

LiuJi-MacBook-Pro:build liuji$ cmake /Users/liuji/Projects/test
-- The C compiler identification is AppleClang 7.3.0.7030031
-- The CXX compiler identification is AppleClang 7.3.0.7030031
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/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: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/liuji/Projects/test/build
LiuJi-MacBook-Pro:build liuji$ make
Scanning dependencies of target mylib
[ 50%] Building CXX object CMakeFiles/mylib.dir/mylib.cpp.o
[100%] Linking CXX static library libmylib.a
[100%] Built target mylib
LiuJi-MacBook-Pro:build liuji$ ls
CMakeCache.txt      CMakeFiles          Makefile            cmake_install.cmake  libmylib.a
LiuJi-MacBook-Pro:build liuji$

```

生成的库文件

使用库文件

增加文件 `test.cpp` :

```

#include "mylib.h"
#include <iostream>
using namespace std;
int main()
{
    cout << add(10, 20) << endl;
}

```

修改 `CMakeLists.txt` :

```

add_library(mylib mylib.cpp mylib.h)
add_executable(test test.cpp)
target_link_libraries(test mylib) #链接mylib

```

执行cmake并且编译


```
liuji@MacBook-Pro:build liuji$ cmake /Users/liuji/Projects/test
-- The C compiler identification is AppleClang 7.3.0.7030031
-- The CXX compiler identification is AppleClang 7.3.0.7030031
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/cc
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/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: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
CMake Warning (dev) at CMakeLists.txt:2 (add_executable):
  Policy CMP0037 is not set: Target names should not be reserved and should
  match a validity pattern. Run "cmake --help-policy CMP0037" for policy
  details. Use the cmake_policy command to set the policy and suppress this
  warning.

  The target name "test" is reserved or not valid for certain CMake features,
  such as generator expressions, and may result in undefined behavior.
  This warning is for project developers. Use -Wno-dev to suppress it.

-- Configuring done
-- Generating done
-- Build files have been written to: /Users/liuji/Projects/test/build
liuji@MacBook-Pro:build liuji$ make
Scanning dependencies of target mylib
[ 25%] Building CXX object CMakeFiles/mylib.dir/mylib.cpp.o
[ 50%] Linking CXX static library libmylib.a
[ 50%] Built target mylib
Scanning dependencies of target test
[ 75%] Building CXX object CMakeFiles/test.dir/test.cpp.o
[100%] Linking CXX executable test
[100%] Built target test
liuji@MacBook-Pro:build liuji$ ls
CMakeCache.txt  CMakeFiles  Makefile  cmake_install.cmake  libmylib.a  test
liuji@MacBook-Pro:build liuji$ ./test
30
```

生成的静态库和可执行文件

子项目

可以将mylib库和test执行文件分为两个子项目。方法是分别建立两个目录，例如mylib和test。将对应文件拷贝到目录下面，在目录下分别编写 `CMakeLists.txt`。目录结构截图如下：

📄 CMakeLists.txt	今天 下午2:44	61 字节	纯文本文稿
▼ 📁 mylib	今天 下午2:42	--	文件夹
📄 CMakeLists.txt	今天 下午2:42	37 字节	纯文本文稿
📄 mylib.cpp	今天 下午2:14	63 字节	C++ Source
📄 mylib.h	今天 下午2:14	28 字节	C Hea...Source
▼ 📁 test	今天 下午2:44	--	文件夹
📄 CMakeLists.txt	今天 下午2:43	113 字节	纯文本文稿
📄 test.cpp	今天 下午2:24	109 字节	C++ Source

根目录下的 `CMakeLists.txt` 内容如下：

```
project(test)
add_subdirectory(mylib)
add_subdirectory(test)
```

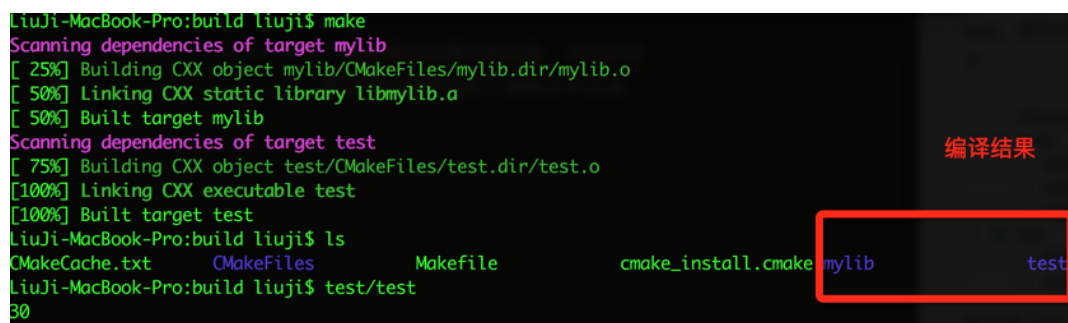
子目录下的 `CMakeLists.txt` 内容分别为：

```
add_library(mylib mylib.cpp mylib.h)
```

和

```
include_directories(${PROJECT_SOURCE_DIR}/mylib)
add_executable(test test.cpp)
target_link_libraries(test mylib)
```

其中 `${PROJECT_SOURCE_DIR}` 表示项目的源代码目录，对应的还有 `${PROJECT_BINARY_DIR}` 表示项目编译后的二进制文件目录。执行cmake，编译后结果如下：



```
LiuJi-MacBook-Pro:build liuji$ make
Scanning dependencies of target mylib
[ 25%] Building CXX object mylib/CMakeFiles/mylib.dir/mylib.o
[ 50%] Linking CXX static library libmylib.a
[ 50%] Built target mylib
Scanning dependencies of target test
[ 75%] Building CXX object test/CMakeFiles/test.dir/test.o
[100%] Linking CXX executable test
[100%] Built target test
LiuJi-MacBook-Pro:build liuji$ ls
CMakeCache.txt  CMakeFiles  Makefile  cmake_install.cmake  mylib  test
LiuJi-MacBook-Pro:build liuji$ test/test
30
```

使用第三方库

假设要在项目中使用OpenCV第三方库，例如 `test.cpp` 如下：

```
#include<opencv2/opencv.hpp>
#include<iostream>
#include"mylib.h"
using namespace std;
using namespace cv;
int main()
{
    cout<<add(10,20)<<endl;
    Mat m(3,3,CV_32F);
    cout<<m<<endl;
    return 0;
}
```

修改 `test` 目录下的 `CMakeLists.txt`，如下：

```
include_directories(${PROJECT_SOURCE_DIR}/mylib)
```

```
include_directories(/usr/local/include) #opencv所在的头文件目录
link_directories(/usr/local/lib) #opencv所在的库文件目录
add_executable(test test.cpp)
target_link_libraries(test mylib opencv_core) #opencv_core是链接的
第三方库
```

程序用cmake编译后执行如下：

```
LiuJi-MacBook-Pro:build liuji$ make
Scanning dependencies of target mylib
[ 25%] Building CXX object mylib/CMakeFiles/mylib.dir/mylib.o
[ 50%] Linking CXX static library libmylib.a
[ 50%] Built target mylib
Scanning dependencies of target test
[ 75%] Building CXX object test/CMakeFiles/test.dir/test.o
[100%] Linking CXX executable test
[100%] Built target test
LiuJi-MacBook-Pro:build liuji$ test/test
30
[0, 0, 0;
 0, 0, 0;
 0, 0, 0]
```

其他指令

cmake还有很多功能，例如install产生安装指令、set命令定义宏，find_package查找第三方库、file进行文件操作、message输出信息等等。网上有很多资料讲解这方面的内容，如果你能够用到这些指令，那么以你的水平看懂网上的资料应该不是什么大问题。