在MacOS（针对Arm架构）里配置OpenGL编程环境（Clion）

H2 安装brew包管理器

brew可以让管理库更加的方便，我们使用国内的镜像安装：

```
/bin/zsh -c "$(curl -fsSL
https://gitee.com/cunkai/HomebrewCN/raw/master/Homebrew.sh)"
```

H2 安装glfw3

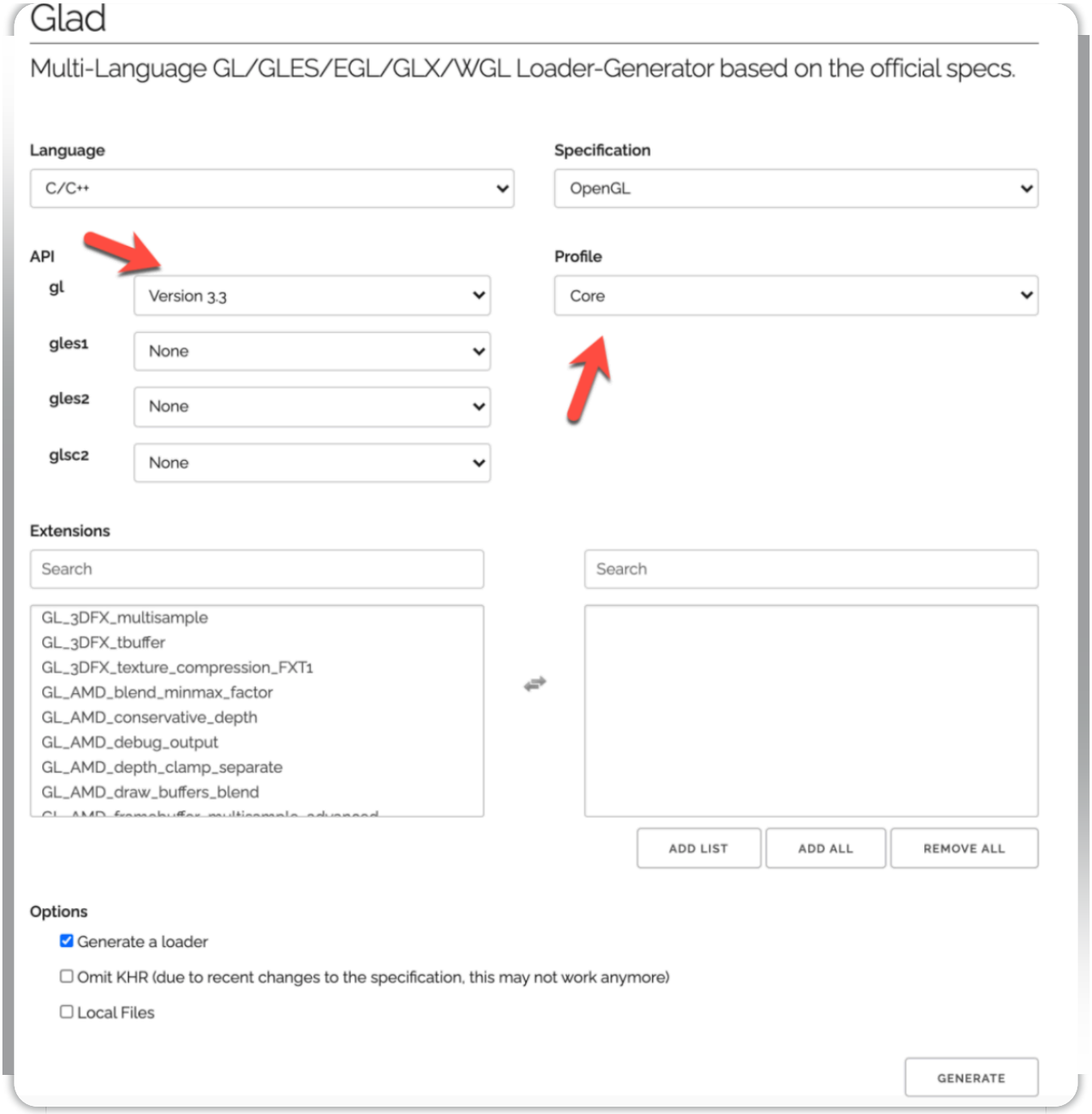
利用brew包管理器：

```
brew install glfw
```

注意一点，如果采用的是arm架构的M系列芯片，brew会默认把所有的文件安装在/opt/homebrew/Cellar/的一个文件夹里。如果是x86架构，那就会在/usr/local/Cellar/。这里以M系列为例子。

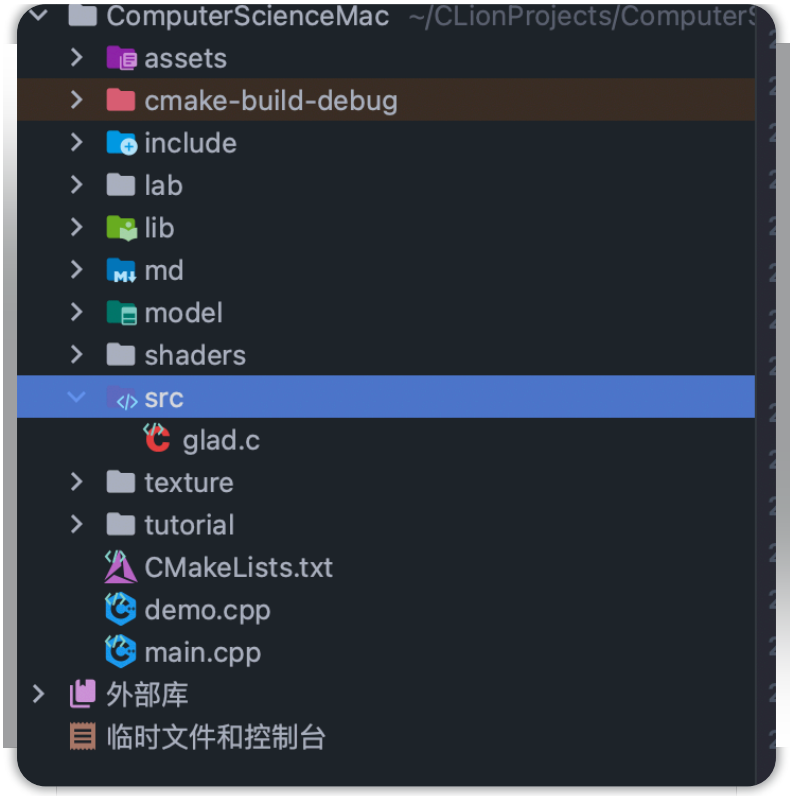
不仅如此，brew 还会在/opt/homebrew/Cellar/的目录里建立一个目录软连接，这个soft link 指向了/opt/homebrew/Cellar/glfw/3.3.8/include 这个目录。这样子，GLFW的头文件就被包含在IDE默认搜索的路径下了。

H3 配置GLAD



将生成好的glad文件夹和KHR文件夹copy到/usr/local/include内（没有的话自己创建），当然也可以将glad文件夹和KHR文件夹copy到本地的项目根目录，这样的话到时候一起编译就好了。这里偏向于第一种：

```
miaokeda@miukedadeMacBook-Air KHR % cd /usr/local/include
miaokeda@miukedadeMacBook-Air include % ls -la
total 16
drwxr-xr-x@ 5 root  wheel   160 10 25 21:09 .
drwxr-xr-x  3 root  wheel    96 10 25 21:09 ..
-rw-r--r--@ 1 root  wheel  6148 10 25 21:09 .DS_Store
drwxr-xr-x@ 3 root  wheel    96 10 25 21:09 KHR
drwxr-xr-x@ 3 root  wheel    96 10 25 21:09 glad
```



H2 配置环境变量

为了方便我们在CMakeLists里调用GLFW和GLAD的路径，我们最好选择配置环境变量，这样的话就会方便许多，不需要在CMakeLists里输入又臭又长的路径了。

因为我们用的是zsh（而不是bash），所以我们在terminal输入：

```
vi ~/.zprofile
```

跟Linux一样，用vi编辑zprofile这个文件，里面存放着各种环境变量，在里面键入：

```
export GLFW_HOME="/opt/homebrew/Cellar/glfw/3.3.8"
export GLAD_HOME="/usr/local/"
```

这样我们定义了两个环境变量，他们的值都是对应库的路径。

编辑完后，我们要激活环境变量，在terminal输入：

```
source ~/.zprofile
```

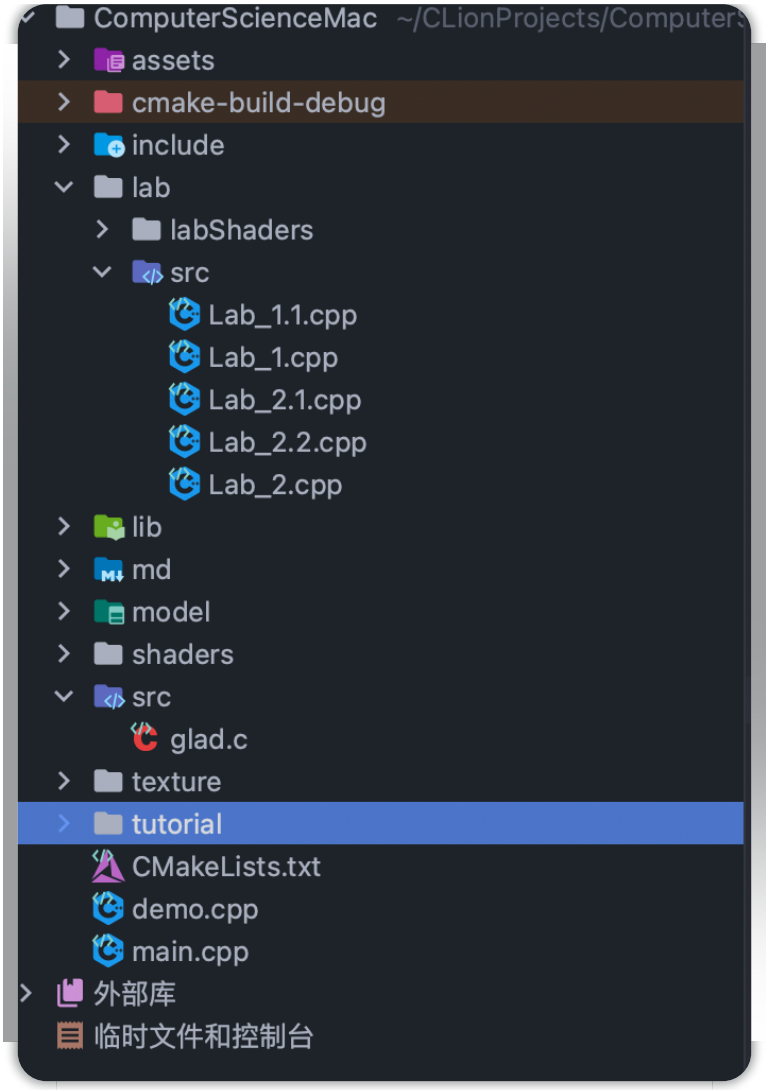
H2 修改CMakeLists.txt

在OpenGL项目中的CMakeLists.txt，加入如下语句：

```
cmake_minimum_required(VERSION 3.23.2)
cmake_policy(VERSION 3.0)
project(ComputerGraphics)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS ON)

if (CMAKE_SYSTEM_NAME MATCHES "Darwin")
    message("Now System is Mac")
    # 检查环境变量
    if (NOT DEFINED ENV{GLFW_HOME})
        message(FATAL_ERROR "found no env named GLFW_HOME")
    endif()
    if (NOT DEFINED ENV{GLAD_HOME})
        message(FATAL_ERROR "found no env named GLAD_HOME")
    endif()

    # 暂存环境变量
    set(GLFW_HOME $ENV{GLFW_HOME})
    set(GLAD_HOME $ENV{GLAD_HOME})

    # 设置头文件目录
    include_directories("${GLFW_HOME}/include")
    include_directories("${GLAD_HOME}/include")
    include_directories("${PROJECT_SOURCE_DIR}/include ")

    # 添加 GLFW3 预编译库
    add_library(glfw SHARED IMPORTED)
    SET_TARGET_PROPERTIES(glfw PROPERTIES IMPORTED_LOCATION
"${GLFW_HOME}/lib/libglfw.dylib")
```

```
    # 链接 GLFW3 预编译库
    link_libraries(glfw)

    add_executable(main main.cpp src/glad.c include/InitShader.cpp
include/Camera/Camera.cpp)
    add_executable(demo demo.cpp src/glad.c include/InitShader.cpp
include/Camera.cpp include/TriMesh.cpp)

    # 利用for循环逐步编译lab/src的文件
    file(GLOB_RECURSE my_c_list RELATIVE ${CMAKE_SOURCE_DIR} "lab/src/*")
    foreach (file_path ${my_c_list})
        string(REPLACE ".cpp" "" new_name ${file_path})
        get_filename_component(filename ${new_name} NAME)
        add_executable(${filename} ${file_path} src/glad.c include/InitShader.cpp
include/Camera/Camera.cpp include/Texture/Texture.cpp  include/TriMesh.cpp)
    endforeach ()

    # 利用for循环逐步编译tutorial的文件
    file(GLOB_RECURSE my_c_list RELATIVE ${CMAKE_SOURCE_DIR} "tutorial/*")
    foreach (file_path ${my_c_list})
        string(REPLACE ".cpp" "" new_name ${file_path})
        get_filename_component(filename ${new_name} NAME)
        add_executable(${filename} ${file_path} src/glad.c
include/Camera/Camera.cpp include/Texture/Texture.cpp)
    endforeach ()
endif ()
```

这里给出项目文件的结构：



H2 坑：要加入宏定义判断

在OpenGL程序中，对于Apple平台一定要加入如下的语句（在创建窗口语句之前加入）：

```
#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif
```

不然会创建窗口失败。

H2 运行程序

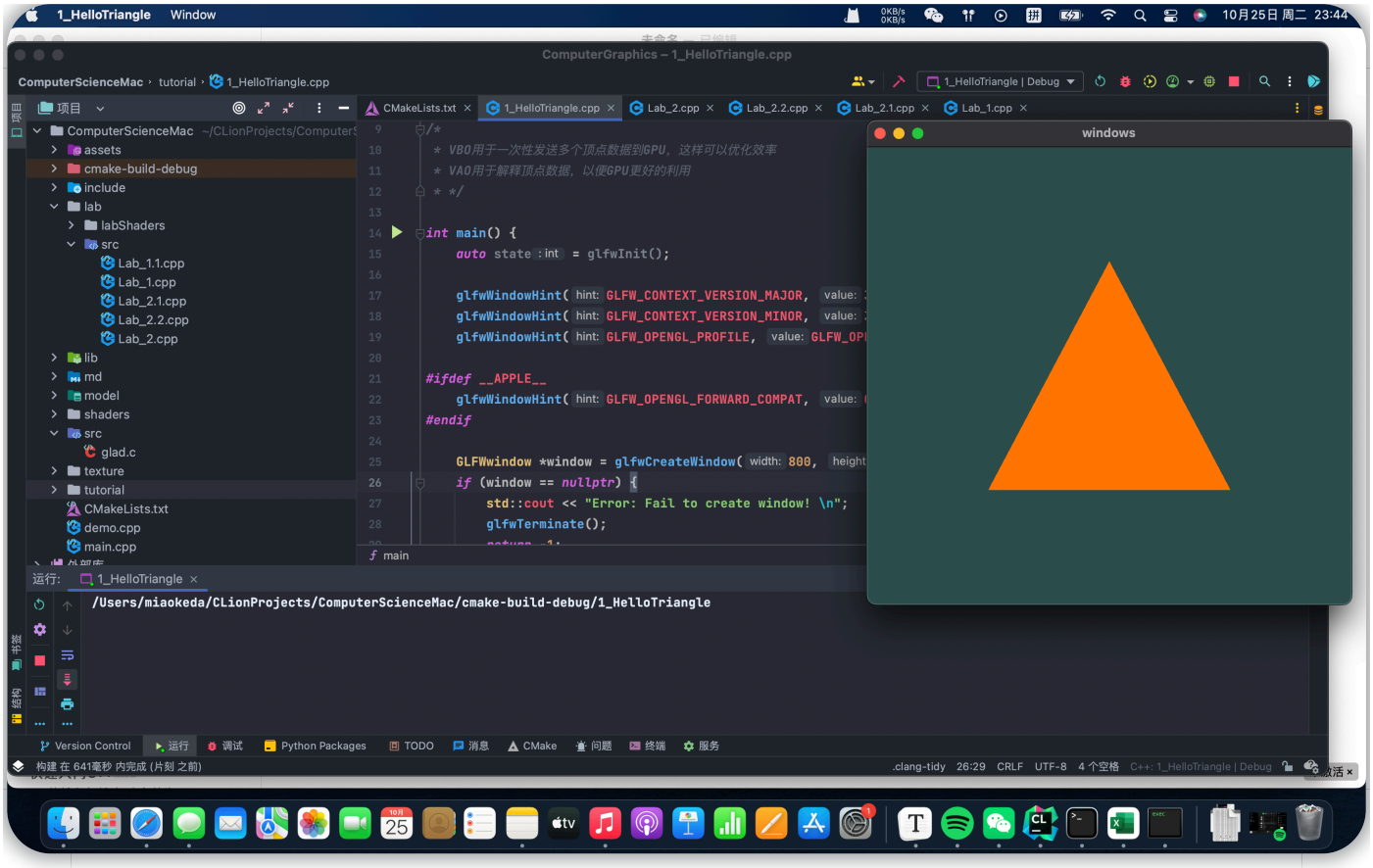
比如要运行：

```
//
// Created by inver on 2022/9/11.
//

#include <iostream>
#include "glad/glad.h"
#include "GLFW/glfw3.h"

/*
 * VBO用于一次性发送多个顶点数据到GPU，这样可以优化效率
 * VAO用于解释顶点数据，以便GPU更好的利用
 * */

int main() {
    auto state = glfwInit();

    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    GLFWwindow *window = glfwCreateWindow(800, 600, "windows", nullptr, nullptr);
    if (window == nullptr) {
        std::cout << "Error: Fail to create window! \n";
        glfwTerminate();
        return -1;
    }

    glfwMakeContextCurrent(window);

    if (!gladLoadGLLoader((GLADloadproc) glfwGetProcAddress)) {
        std::cout << "Error: Fail to initialize GLAD! \n";
        return -1;
    }
```

```
    //End initialize glfwSource and glad

    float vertices[] = {
            -0.5f, -0.5f, 0.0f,
            0.0f, 0.5f, 0.0f,
            0.5f, -0.5f, 0.0f
    };

    //initialize vbo
    unsigned int vbo;
    glGenBuffers(1, &vbo);  //spawn a new vbo
    glBindBuffer(GL_ARRAY_BUFFER, vbo); //bind vbo with ARRAY_BUFFER
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
                 GL_STATIC_DRAW);  //transport vertices data to buffer memory
    //GL_STATIC_DRAW means static data (not dynamic)

    //initialize vao
    unsigned int vao;
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    //set vertices pointer
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void *)
nullptr);
    glEnableVertexAttribArray(0);


    //initialize shader
    const char *vshSource = "#version 330 core\n"
                            "layout (location = 0) in vec3 aPos;\n"
                            "void main()\n"
                            "{\n"
                            "   gl_Position = vec4(aPos.x, aPos.y, aPos.z,
1.0);\n"
                            "}\0";
    const char *fshSource = "#version 330 core\n"
                            "out vec4 FragColor;\n"
                            "void main()\n"
                            "{\n"
                            "   FragColor = vec4(1.0f, 0.5f, 0.2f, 1.0f);\n"
                            "}\n\0";

    //compile vshader
    unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
```

```
    glShaderSource(vertexShader, 1, &vshSource, nullptr);
    glCompileShader(vertexShader);

    //compile fshader
    unsigned int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, &fshSource, nullptr);
    glCompileShader(fragmentShader);

    //spawn shader program
    unsigned int shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);

    while (!glfwWindowShouldClose(window)) {
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);

        glUseProgram(shaderProgram);
        glBindVertexArray(vao);
        glDrawArrays(GL_TRIANGLES, 0, 3);

        glfwSwapBuffers(window);
        glfwPollEvents();
    }
    glDeleteProgram(shaderProgram);
    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);
    glfwTerminate();
}
```

得到的结果：


1_HelloTriangle
Window
10月25日 周二 23:44
ComputerGraphics – 1_HelloTriangle.cpp
ComputerScienceMac › tutorial › 1_HelloTriangle.cpp
1_HelloTriangle | Debug
项目
ComputerScienceMac ~/CLionProjects/ComputerS
assets
cmake-build-debug
include
lab
labShaders
src
Lab_1.1.cpp
Lab_1.cpp
Lab_2.1.cpp
Lab_2.2.cpp
Lab_2.cpp
lib
md
model
shaders
src
glad.c
texture
tutorial
CMakeLists.txt
demo.cpp
main.cpp
CMakeLists.txt
1_HelloTriangle.cpp
Lab_2.cpp
Lab_2.2.cpp
Lab_2.1.cpp
Lab_1.cpp
/*
* VBO用于一次性发送多个顶点数据到GPU，这样可以优化效率
* VAO用于解释顶点数据，以便GPU更好的利用
* */
int main() {
auto state :int = glfwInit();
glfwWindowHint(hint: GLFW_CONTEXT_VERSION_MAJOR, value:
glfwWindowHint(hint: GLFW_CONTEXT_VERSION_MINOR, value:
glfwWindowHint(hint: GLFW_OPENGL_PROFILE, value: GLFW_OP
#ifdef __APPLE__
glfwWindowHint(hint: GLFW_OPENGL_FORWARD_COMPAT, value:
#endif
GLFWwindow *window = glfwCreateWindow(width: 800, height
if (window == nullptr) {
std::cout << "Error: Fail to create window! \n";
glfwTerminate();
f main
windows
运行:
1_HelloTriangle
/Users/miaokeda/CLionProjects/ComputerScienceMac/cmake-build-debug/1_HelloTriangle
Version Control
运行
调试
Python Packages
TODO
消息
CMake
问题
终端
服务
构建 在 641毫秒 内完成 (片刻 之前)
.clang-tidy 26:29 CRLF UTF-8 4 个空格 C++: 1_HelloTriangle | Debug