Windows 下搭建 googletest 测试框架(C/C++)

主要分为以下几个部分

- 环境准备
- 源代码准备
- googletest与测试代码
- 编译执行

一、环境准备

1. MinGW

可以直接下载 MinGW， `x86_64-posix-sjlj` 下载地址

也可以通过下载带编译器的 CodeBlocks 来安装 MinGW， 下载地址

安装完成后，将 g++/gcc.exe 所在的目录添加到 环境变量 中，在 cmd 中运行 `gcc -v` ，显示版本信息则成功

```
gcc version 8.1.0 (x86_64-posix-seh-rev0, Built by MinGW-W64 project)
```

2. Cmake

直接在官网下载 Windows x64 Installer 版本， 下载地址

安装时记得勾选添加环境变量选项，同样可以在 cmd 中运行 `cmake --version` ，查看是否配置成功

```
cmake version 3.26.0-rc2
```

CMake 是用于生成 Makefile 的，生成规则写在 CMakeLists.txt 文件中，通过 cmake 命令生成 Makefile，再通过 Makefile 编译项目

在 CMakeLists.txt 中，命令名字是不区分大小写的，而参数和变量是大小写相关的

3. VScode

直接在网上下载即可，后续命令行操作都是在集成终端中演示，鼠标点击文件夹-右键-在集成终端中打开

二、源代码准备

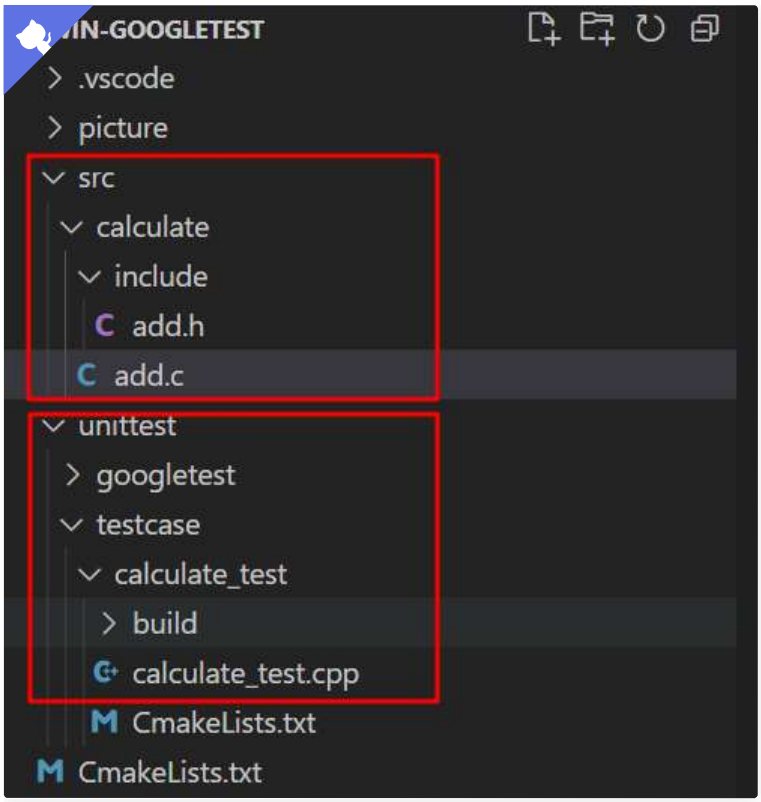
1. 目录结构

首先创建一个 win-googletest 目录，目录下创建 src 和 unittest 两个目录，分别存放源代码文件和测试代码文件

src 下可以创建对应不同功能的文件夹，这里创建一个 calculate 文件夹，存放有关计算操作的 c 代码，以及在 calculate 中创建了一个 include 文件夹存放头文件

unittest 下创建一个 testcase 文件夹用于存放测试用例，testcase 下创建一个对应数组的测试文件夹 calculate_test

```
win-googletest
|-src
   |-calculate
        |-include
        |-add.c
|-unittest
   |-testcase
        |-calculate_test
             |-calculate_test.cpp
```



其实这里的目录结构并不重要，后续在 CMakeLists.txt 中进行对应即可（建议照自己的想法建目录存放源代码与测试代码，后续会介绍链接对应文件，使得编译成功）

2. 源码

add.c 中的代码如下

```c
#include <stdio.h>
#include "../include/add.h"

//加法
int add(int a, int b)
{
    return a+b;
}
```

add.h 中的代码如下

```c
#ifndef __ADD_H_
#define __ADD_H_

int add(int a, int b);

#endif  // __ADD_H_
```

在 src 目录下创建 main.c，在其中调用 add() 函数

```c
#include <stdio.h>
#include "add.h"

int main()
{
    int a=3, b=4;
    printf("a+b= %d\n",add(a, b));
}
```

在 win-googletest 下新建一个 CMakeLists.txt 文件，其内容如下

```
cmake_minimum_required(VERSION 3.0)  # cmake -G "MinGW Makefiles" ..
project(GTESTWIN)

# 头文件目录
include_directories("${CMAKE_CURRENT_SOURCE_DIR}/src/calculate/include")
# 源文件目录
AUX_SOURCE_DIRECTORY("${CMAKE_CURRENT_SOURCE_DIR}/src/calculate" DIR_SRCS)

# 生成可执行文件
add_executable(calculate ./src/main.c ${DIR_SRCS})
```

`include_directories` 添加头文件的搜索路径

`AUX_SOURCE_DIRECTORY` 找到目录下的所有源文件，存在变量 DIR_SRCS 中，变量的引用 ${DIR_SRCS}

`add_executable` 用指定的程序代码(.c)生成指定可执行文件(.exe)

3. 编译与运行

有两种编译方式，

一种直接通过 gcc 指定特定文件生成可执行文件

另一种即是通过 camke 根据 CMakeLists.txt 中的规则生成可执行文件

- gcc

```
# 切换到对应文件夹下 这里为 win-googletest/src
# 生成可执行文件
gcc main.c ./calculate/add.c -I ./calculate/include -o calculate.exe
# 执行
./calculate.exe
```

`-I` 指定包含头文件目录

`-o` 指定生成输出文件

结果如下

```
PS E:\Git\win-googletest\src> gcc main.c ./calculate/add.c -I ./calculate/include -o calculate.exe
PS E:\Git\win-googletest\src> ./calculate.exe
a+b= 7
PS E:\Git\win-googletest\src>
```

可以看到，程序顺利执行

- cmake

在 win-googletest 目录下新建 build 目录，用于存放 cmake 运行后产生的文件

```
# 切换到对应文件夹下 这里为 win-googletest/build
cmake -G "MinGW Makefiles" ..
mingw32-make
```

在 Windows 中，cmake 命令会优先选择 msvc 编译器，因此需要指定编译器为 MinGW，需要添加参数 -G "MinGW Makefiles"。make 指令也需要更改为 mingw32-make

`..` CMakeLists.txt 文件在 build 目录的上一级目录中

结果如下

```
Git\win-googletest\build> cmake -G "MinGW Makefiles" ..
The CXX compiler identification is GNU 8.1.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: [redacted]/MinGW/bin/gcc.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Check for working CXX compiler: [redacted]inGW/bin/c++.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (1.1s)
-- Generating done (0.1s)
-- Build files have been written to: E:/Git/win-googletest/build
PS E:\Git\win-googletest\build> make
make : 无法将“make”项识别为 cmdlet、函数、脚本文件或可运行程序的名称。请检查名称的拼写，如果包括路径，请确保路径正确，然后再试一次。
所在位置 行:1 字符: 1
+ make
+ ~~~~
    + CategoryInfo          : ObjectNotFound: (make:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS E:\Git\win-googletest\build> mingw32-make
[ 33%] Building C object CMakeFiles/calculate.dir/src/main.c.obj
[ 66%] Building C object CMakeFiles/calculate.dir/src/calculate/add.c.obj
[100%] Linking C executable calculate.exe
[100%] Built target calculate
PS E:\Git\win-googletest\build> ./calculate.exe
a+b= 7
PS E:\Git\win-googletest\build>
```

同样成功执行

三、googletest 与测试代码

1. googletest

googletest 下载，github 上直接搜即可，这里也贴上链接， 下载地址

将下载后的 googletest 文件解压后放到 unittest 文件夹下，可以看到 googletest 目录下有 CMakeLists.txt 文件，在其目录下新建 build 文件夹用于 cmake 操作

```
#googletest/build目录下
cmake -G "MinGW Makefiles" ..
mingw32-make
```

会在 build/lib 目录下生成四个库文件 `libgtest.a` 、 `libgtest_main.a` 、 `libgmock.a` 、 `libgmock_main.a` ，将 lib 文件夹复制到 googletest 文件夹下，后续链接库文件时需要用到

```
-- Generating done (0.1s)
-- Build files have been written to: E:/Git/win-googletest/unittest/googletest/build
PS E:\Git\win-googletest\unittest\googletest\build> mingw32-make
[ 12%] Building CXX object googletest/CMakeFiles/gtest.dir/src/gtest-all.cc.obj
[ 25%] Linking CXX static library ..\lib\libgtest.a
[ 25%] Built target gtest
[ 37%] Building CXX object googlemock/CMakeFiles/gmock.dir/src/gmock-all.cc.obj
[ 50%] Linking CXX static library ..\lib\libgmock.a
[ 50%] Built target gmock
[ 62%] Building CXX object googlemock/CMakeFiles/gmock_main.dir/src/gmock_main.cc.obj
[ 75%] Linking CXX static library ..\lib\libgmock_main.a
[ 75%] Built target gmock_main
[ 87%] Building CXX object googletest/CMakeFiles/gtest_main.dir/src/gtest_main.cc.obj
[100%] Linking CXX static library ..\lib\libgtest_main.a
[100%] Built target gtest_main
PS E:\Git\win-googletest\unittest\googletest\build>
```

2. 测试代码

在 calculate_test 文件夹下新建 calculate_test.cpp 测试文件，文件内容如下

```
#include "gtest/gtest.h"
#include "gmock/gmock.h"

extern "C"{
#include "add.h"
}


TEST(ARRAY_TEST, calculate_test) {
    int a=3,b=4;
    EXPECT_EQ(7, add(a, b));
}

#if 1
int main(int argc, char *argv[])
{
  testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
#endif
```

在 calculate_test 文件夹下新建 CMakeLists.txt 文件，用于构建测试文件，内容如下

```
cmake_minimum_required(VERSION 3.0)  # cmake -G "MinGW Makefiles" ..
project(ARRAY_TEST)

# gtest库相关
# 如果把gtest放到test目录下，则使用如下包含关系：
include_directories(../../googletest)  # 编译gtest
include_directories(../../googletest/googletest/include)  # 包含gtest等头文件
include_directories(../../googletest/googlemock/include)  # 包含gmock等头文件
# include_directories(../../googletest/lib)


# 头文件目录
include_directories("../../../src/calculate/include")
# 源文件目录
AUX_SOURCE_DIRECTORY("../../../src/calculate" SRCS)

# 源代码文件
set(SRC_FILES
        ${CMAKE_CURRENT_SOURCE_DIR}/../../../src/calculate/add.c
)

# 测试代码目录
AUX_SOURCE_DIRECTORY("${CMAKE_CURRENT_SOURCE_DIR}" TEST_SRCS)

# 生成测试可执行程序
add_executable(calculate_test ${SRC_FILES} ${TEST_SRCS})

find_library(gtest libgtest.a ../../googletest/lib)
find_library(gtest_main libgtest_main.a ../../googletest/lib)
find_library(gmock libgmock.a ../../googletest/lib)
find_library(gmock_main libgmock_main.a ../../googletest/lib)
# 链接测试库,pthread 库一定要写在 ${GTEST_BOTH_LIBRARIES} 的后面，否则编译时会报错，
# GTEST_BOTH_LIBRARIES表示链接google test的两个库
target_link_libraries( calculate_test
        PRIVATE
        ${gtest}
        ${gtest_main}  # 使用gtest带的main函数,如果检测到外部有main函数，则使用外部main函数,外部main函数要配置gtest初始化。
        ${gmock}
        ${gmock_main} # 使用gmock带的main函数,如果检测到外部有main函数，则使用外部main函数,与gtest_main同时存在则自动配置。
        pthread
        )

```

`set` 将源文件代码路径存储到 SRC_FILES 变量中，后续生成可执行程序时使用 ${SRC_FILES}

`find_library` 作用是查找对应名字的库文件存储到变量中，有三个参数，分别对应变量、查找文件名、查找路径

`t_link_libraries` 链接库文件

四、编译执行

同源代码编译，只是将 main.c 文件替换为测试文件，并添加 gtest 头文件所在目录，以及能找到 gtest 的库文件，还是两种方式

- g++

在 calculate_test 目录下打开集成终端，输入以下命令

```
# 在 calculate_test 目录下 (gcc 好像不行，得用 g++)
g++ calculate_test.cpp ../../../src/calculate/add.c ../../googletest/lib/libgtest.a -I ../../../src/calculate/include -I ../../
# 会在目录下生成 calculate_test.exe 文件
./calculate_test.exe
```

如果出现未定义错误，需要将测试代码中 `extern {` 以及 `}` 这两行注释掉，只保留 `#include "add.h"` ，原因能是 cpp 和 c 混合编译导致的，具体原因大家可以去查一查，哈哈😊，具体结果如下

```
PS E:\Git\win-googletest\unittest\testcase\calculate_test> g++ calculate_test.cpp ../../../src/calculate/add.c ../../google
test/lib/libgtest.a -I ../../../src/calculate/include -I ../../googletest/googletest/include -I ../../googletest/googlemock
/include -o calculate_test.exe
PS E:\Git\win-googletest\unittest\testcase\calculate_test> ./calculate_test.exe
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from ARRAY_TEST
[ RUN      ] ARRAY_TEST.calculate_test
[       OK ] ARRAY_TEST.calculate_test (0 ms)
[----------] 1 test from ARRAY_TEST (2 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (10 ms total)
[  PASSED  ] 1 test.
PS E:\Git\win-googletest\unittest\testcase\calculate_test>
```

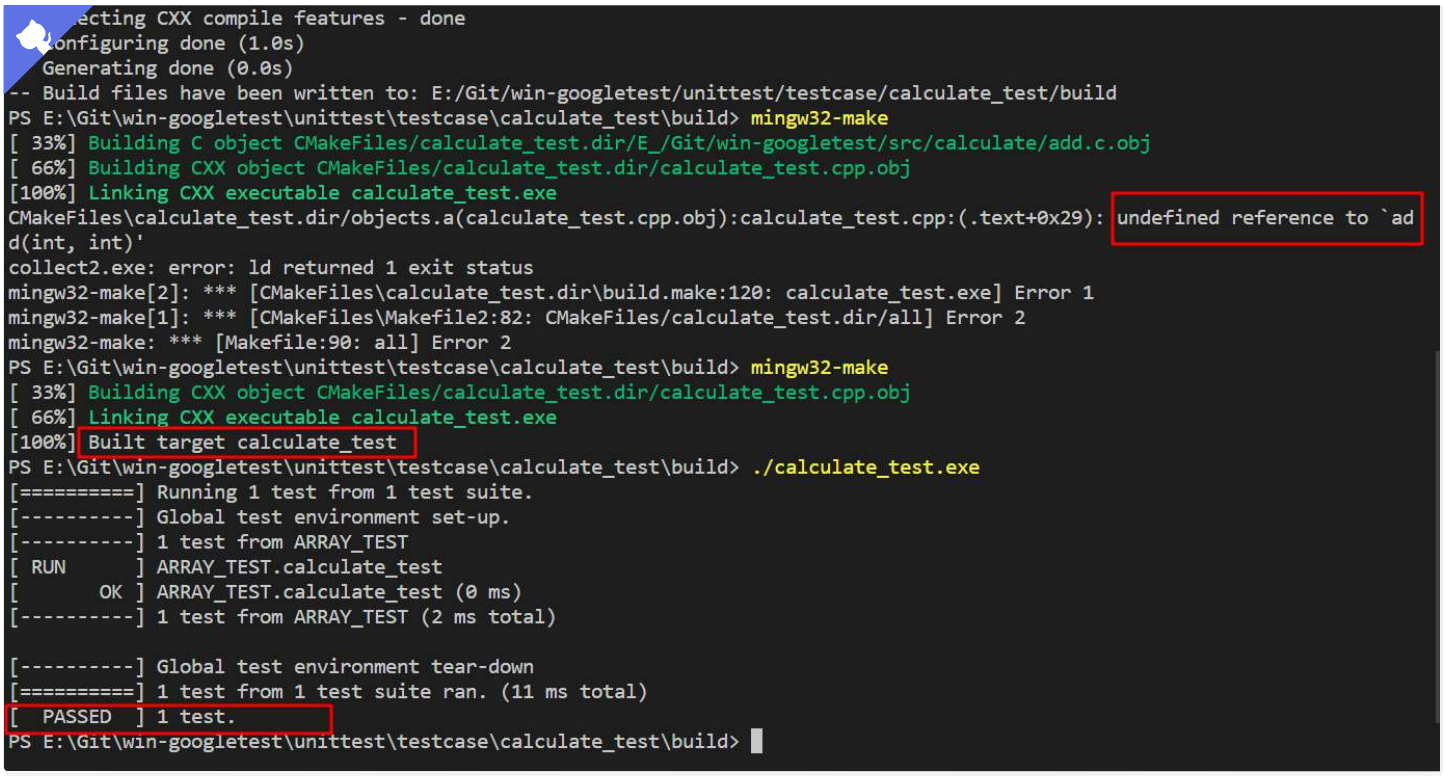
- cmake

可以看到通过 g++ 指定文件编译的方法需要输入太多的目录，每次运行的时候都输入这么多的话就显得很麻烦，cmake 的作用就是预先设定这些目录，自动化编译，从设定的目录中寻找需要的文件

在 calculate_test 文件夹下创建 build 文件夹存放 cmake 操作产生的文件，切换到 build 目录下打开集成终端，执行 cmake 和 make 操作

```
cmake -G "MinGW Makefiles" ..
mingw32-make
```

如果出现未定义错误，需要将测试代码中 `extern {` 以及 `}` 这两行注释打开



可以看到，成功执行

如果后续需要继续添加不同功能的源代码和测试代码，可以直接将 `calculate` 以及 `calculate_test` 在对应文件夹下复制一份，修改一下名字，并在 CMakeLists.txt 文件中设定好自己需要编译的源代码路径以及头文件路径，修改生成的可执行程序名称，然后再运行 cmake 以及 make 即可

总结

总的来说，主要是要了解清楚编译过程中需要哪些头文件和库文件，如果缺乏，就在 CMakeLists.txt 文件中指定路径，让其在自动编译时能够找到它😆

源码：github

参考链接

https://www.bilibili.com/read/cv17586887/

https://github.com/TonsenWei/gtest_win

> 作者：augustine0654
> 出处：https://www.cnblogs.com/augustine0654/p/17156412.html
> 版权：本作品采用「署名-非商业性使用-相同方式共享 4.0 国际」许可协议进行许可。

分类：c 语言

0 0

« 上一篇： Ubutun 安装 zsh 和 oh-my-zsh
» 下一篇： Linux 中文乱码