GoogleTest 记录文档

编辑历史

更新日期	作者	更新内容
2024.02.29	胡益华	Initialization

目录

1. 引言

1.1 概述

单元测试指对软件中最小的可测试单元进行检查和功能验证。这里的最小单元没有明确的界限，一个类，一个函数，或者几个函数组成的一个功能模块在测试中都可以视为一个单元，只要它是可测试的。

单元测试需要具备可重复性，举个例子，我编写了一个数据结构用于信息的增删改查，是用链表实现的，基于信息的增删改查功能，我编写了相应的单元测试测试用例；后来，我改用红黑树实现了这个功能。此时为了验证信息的增删改查功能是否正确，我可以直接使用之前的测试用例而不需要重写测试用例。同时，在用例失败时，需要尽可能地提供测试失败信息，以便定位问题所在。

1.2 GoogleTest

GoogleTest 是一款开源的用于 C/C++单元测试的软件。gitbub 地址如下：
https://github.com/google/googletest

官方使用手册地址如下：
https://google.github.io/googletest/

GoogleTest 是用 C++编写的，主要用于 C++代码的单元测试，当然一定程度上对 C 也是支持的，不过一些 C++专用的断言语句 C 代码就不能使用了。

2. 安装

以 Linux，centos7 环境为例，下载源码后解压。

(1) Gtest 提供了 Cmake 构建编译的方式，运行如下命令。

```
mkdir -p build && cd build
cmake ..
make edit_cache
```

(2) 打开 CMake 编译界面。然后在编辑界面中输入 t，会显示详细编辑界面。

(3) 默认生成静态库，若需要动态库，可以开启 BUILD_SHARED_LIBS。

(4) 选择编译器，默认的编译器一般是/bin 目录下的。若/bin 目录下的编译器版本过低或需要交叉编译等，可以将其中的 C 和 CXX 编译器分别修改为对应平台的 C 和 CXX 编译器。以 gcc/g++为例。使用 which 命令查看 gcc/g++的路径，然后将 CMAKE_C_COMPILER 和 CMAKE_CXX_COMPILER 选项分别改成对应可执行文件路径即可。其余的 ar、ranlib 等工具一般会被自动添加。

(5) 生成物的路径默认一般是/usr/local 目录，若没有写权限或希望自行指定生成物路径，可以修改 CMAKE_INSTALL_PREFIX 选项。

(6) 按顺序输入 c、g。显示无错误即可。

```
make -B -j && make install
```

3. 使用

3.1 官方样例

Gtest 提供了一些官方样例。位于./googletest/samples 目录下。这里以 x86 平台静态库为例。进入 samples 目录，以如下命令编译。

```
g++ -o sample1 ../src/gtest_main.cc sample1.cc sample1_unittest.cc \
-lpthread -lgtest \
-I../include/ -L/GoogleTest/static_lib_dir/
```

若看到如下打印界面，那么恭喜，你已经掌握 gtest 的核心了。

```
Running main() from ../src/gtest_main.cc
[==========] Running 6 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 3 tests from FactorialTest
[ RUN      ] FactorialTest.Negative
[       OK ] FactorialTest.Negative (0 ms)
[ RUN      ] FactorialTest.Zero
[       OK ] FactorialTest.Zero (0 ms)
[ RUN      ] FactorialTest.Positive
[       OK ] FactorialTest.Positive (0 ms)
[----------] 3 tests from FactorialTest (0 ms total)

[----------] 3 tests from IsPrimeTest
[ RUN      ] IsPrimeTest.Negative
[       OK ] IsPrimeTest.Negative (0 ms)
[ RUN      ] IsPrimeTest.Trivial
[       OK ] IsPrimeTest.Trivial (0 ms)
[ RUN      ] IsPrimeTest.Positive
[       OK ] IsPrimeTest.Positive (0 ms)
[----------] 3 tests from IsPrimeTest (0 ms total)

[----------] Global test environment tear-down
[==========] 6 tests from 2 test suites ran. (0 ms total)
[  PASSED  ] 6 tests.
```

3.2 用例构成

打开官方样例的文件简单浏览，第一个样例并不复杂。Gtest 每个用例一般分为 test suite 和 test case，即测试套件和测试用例，测试用例是测试套件的子集。官方建议测试用例和测试套件名称中不要含有下划线，实际我尝试时发现含有下划线也不会出错，不过还是尽量避免使用下划线好了。

3.3 编译参数

(1) 编译时 -lgtest 用于连接 gtest 库，同理也可以加上 -lgmock 参数。
(2) 编译时会需要加上-lpthread 参数。
(3) 尽管 Gtest 对 C 在一定程度上是支持的，但编译时需要使用 C++的编译工具，不可使用 C 的编译工具。

3.4 文件执行选项

编译得到可执行文件后，运行时可以指定环境变量或添加相应的选项来控制测试用例的执行。官方文档地址如下。

https://google.github.io/googletest/advanced.html

文档比较长，可以 ctrl F 定位一下具体位置。这里记录一些常用的参数选项。

3.4.1 测试套件和用例的筛选

(1) ./unitTest

运行所有测试套件中的所有测试用例。

(2) ./unitTest --gtest_filter=*

运行所有测试套件中的所有测试用例。

(3) ./unitTest --gtest_filter=utSuite.*

运行名称为 utSuite 测试套件中的所有测试用例。

(4) ./unitTest --gtest_filter=utSuite.utCase

运行名称为 utSuite 测试套件中名称为 utCase 的测试用例。

(5) ./unitTest --gtest_filter=utSuite*.utCase*

运行名称以 utSuite 开头的测试套件中的名称以 utCase 开头的测试用例。实际上就和 Linux 中通配符的使用一样。

(6) ./unitTest --gtest_filter=*utCase1*:* utCase2*

仅运行名称中含有 utCase1 和 utCase2 的测试套件。

(7) ./unitTest --gtest_filter=-*utCase*

运行所有的测试套件，但不运行名称中含有 utCase 的测试套件。

(8) ./unitTest --gtest_filter=utSuite.*-utSuite.utCase

运行名称为 utSuite 测试套件中的所有测试用例，但不运行名称为 utSuite 测试套件中名称为 utCase 的测试用例。

(9) ./unitTest --gtest_filter=\
utSuite1.*: utSuite2.*-utSuite1.utCase1:utSuite2.utCase2

运行名称为 utSuite1 测试套件中的所有测试用例和运行名称为 utSuite2 测试套件中的所有测试用例，但不运行名称为 utSuite1 测试套件中名称为 utCase1 的测试用例，也不运行名称为 utSuite2 测试套件中名称为 utCase2 的测试用例。

3.4.2 测试运行状态

(1) ./unitTest --gtest_fail_fast

测试过程中发现错误后立刻停止后续测试。这里的停止是针对测试套件而言的。举个例子，若有两个测试套件 utSuite1 和 utSuite2，若 utSuite1 出错，那么 utSuite2 将不会被执行。若 utSuite1 中有两个测试用例，utCase1 和 utCase2，若 utCase1 出错，那么 utCase2 不会被停止。后续没有被执行的测试用例显示 skipped。

(2) ./unitTest --gtest_repeat=1000

重复运行测试用例 1000 次。

--gtest_fail_fast 会影响当前这次测试套件的运行情况，但是不会影响下一次测试套件的运行。

(3) ./unitTest --gtest_repeat=-1

重复次数为负数，那么将永远重复运行测试用例。

(4) ./unitTest --gtest_repeat=1000 --gtest_break_on_failure

重复运行测试用例 1000 次，在测试用例出错时停止所有的重复测试。

--gtest_break_on_failure 会在测试用例出错后立刻停下，即使当前测试套件中还有其它未被执行的测试用例。

(5) ./unitTest --gtest_shuffle

Gtest 运行时，默认以测试套件的顺序执行。添加--gtest_shuffle，那么测试套件的执行顺序会被打乱。若希望测试套件间不存在依赖关系，可以使用此选项；若测试套件间存在依赖关系，那么不能添加此参数。

3.4.3 测试运行结果

(1) ./unitTest --gtest_brief=1

Gtest 运行时，默认会将所有的测试用例测试结果打印。添加--gtest_brief=1 选项，那么运行时只会打印出错的测试用例。

(2)_1 ./unitTest --gtest_output=xml:/report_dir

(2)_2 ./unitTest --gtest_output=xml:/report_dir/report_name

在测试完成后输出一份 xml 格式的报告。若指定了目录，没有指定文件名，那么文件名默认为可执行文件文件名的前缀加上.xml 后缀；若指定了文件名，那么就输出相应文件名的文件。

实际上 xml 报告看起来也是比较费劲的，后续可以使用 xsltproc 等工具对其进行转化，生成 html 格式的报告。

(3) ./unitTest --gtest_output=json:/report_dir

在测试完成后输出一份 json 格式的报告。同上，不再赘述。