**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

[GoogleTest记录文档 I](#_Toc10054)

[编辑历史 I](#_Toc10270)

[目录 II](#_Toc3171)

[1. 引言 1](#_Toc5978)

[1.1 概述 1](#_Toc19085)

[1.2 GoogleTest 1](#_Toc605)

[2. 安装 2](#_Toc15778)

[3. 使用 3](#_Toc7909)

[3.1 官方样例 3](#_Toc20568)

[3.2 用例构成 4](#_Toc20727)

[3.3 编译参数 4](#_Toc4644)

[3.4 文件执行选项 4](#_Toc16307)

[3.4.1 测试套件和用例的筛选 4](#_Toc30004)

[3.4.2 测试运行状态 5](#_Toc9685)

[3.4.3 测试运行结果 5](#_Toc15739)

**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。