

# GoogleTest初级使用方法

作者/翻译：李玉冰

编写C/C++程序需要进行单元测试，GoogleTest是非常好用的 C++ 程序测试框架。本文大部分内容翻译于官方用户使用手册，有一些微小的改动。

## 1.简介

GoogleTest 遵循 Abseil 生态哲学，Abseil 是Google开源的从Google内部代码块中抽取出来的一系列最基础的软件库。GoogleTest 用于C++程序测试，它基于xUnit架构，与JUnit和PyUnit相似，不仅支持单元测试，还支持其它任意的测试。

GoogleTest 不使用ISTQB术语中的 Test Case，而是使用 Test suite 表达相同的含义。

TEST() 在 Google Test中意为：

Exercise a particular program path with specific input values and verify the results。使用特定输入值执行特定程序路径并验证结果。

GoogleTest使用可参考 用户使用手册，建议从 Google Test Primer 开始。

- (1) Github地址: <https://github.com/google/googletest>
- (2) 用户使用手册: <https://google.github.io/googletest/>
- (3) Abseil地址: <https://github.com/abseil/abseil-cpp>

## 2.安装和简单使用

### 2.1 安装

- (1) 访问release页面: <https://github.com/google/googletest/releases>，下载 source code。
- (2) 构建工具支持 Bazel 和 CMake，由于平时的使用习惯，我们选用 CMake。
  - 使用 Cmake 查阅: <https://google.github.io/googletest/quickstart-cmake.html>
  - 使用 Bazel 查阅: <https://google.github.io/googletest/quickstart-bazel.html>
- (3) 安装cmake.

```
$ sudo snap install cmake
```

## 2.2 简单使用

(1) 为项目创建一个目录。

```
$ mkdir my_project && cd my_project
```

(2) 创建一个 `CMakeLists.txt` 文件并声明 `GoogleTest` 的依赖。

```
$ vim CMakeLists.txt
```

```
cmake_minimum_required(VERSION 3.14)
project(my_project)

# GoogleTest requires at least C++11
set(CMAKE_CXX_STANDARD 11)

include(FetchContent)
FetchContent_Declare(
    googletest
    URL
    https://github.com/google/googletest/archive/609281088cfefc76f9d0ce82e1ff6c30cc3591e5.zip
)
# For Windows: Prevent overriding the parent project's compiler/linker settings
set(gtest_force_shared_crt ON CACHE BOOL "" FORCE)
FetchContent_MakeAvailable(googletest)
```

上面的文件声明了一个从 Github 下载的 `GoogleTest` 依赖，`609281088cfefc76f9d0ce82e1ff6c30cc3591e5` 是用于 `GoogleTest` 版本的 git commit hash 值，我们建议更新这个 hash 以获得最新的版本。

注：`git commit hash` 是每次提交会生成的 hash 值，获取方法见附录 4.1 节。

(3) 创建并运行

将 `GoogleTest` 声明成一个依赖时，就可以在自己的项目中使用 `GoogleTest` 代码。

在 `my_project` 目录下，创建一个名为 `hello_test.cc` 的文件，文件内容如下（声明了一些基础断言）：

```
#include <gtest/gtest.h>

// Demonstrate some basic assertions.
TEST(HelloTest, BasicAssertions) {
    // Expect two strings not to be equal.
    EXPECT_STRNE("hello", "world");
    // Expect equality.
    EXPECT_EQ(7 * 6, 42);
}
```

为构建代码，需要在 `CMakeLists.txt` 文件末尾添加如下内容：

```

enable_testing()

add_executable(
    hello_test
    hello_test.cc
)
target_link_libraries(
    hello_test
    gtest_main
)

include(GoogleTest)
gtest_discover_tests(hello_test)

```

在 `Cmake` 中启用了测试，并且声明了想要构建的二进制C++测试（`hello_test`），并且将它链接到 `GoogleTest`（`gtest_main`）。最后两行使用 `Google Test CMake module` 启用 `CMake` 的测试运行器，并发现二进制文件中的测试。

在 `my_project` 目录下构建并运行测试：

```

$ cmake -S . -B build

-- The C compiler identification is GNU 9.3.0
-- The CXX compiler identification is GNU 9.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found Python: /usr/local/bin/python3.9 (found version "3.9.5") found
components: Interpreter
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Failed
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
-- Configuring done
-- Generating done
-- Build files have been written to: /home/xianyubing/C/my_project/build

```

```

$ cmake --build build

[ 10%] Building CXX object _deps/googletest-
build/googletest/CMakeFiles/gtest.dir/src/gtest-all.cc.o
[ 20%] Linking CXX static library ../../lib/libgtest.a
[ 20%] Built target gtest

```

```
[ 30%] Building CXX object _deps/googletest-  
build/googletest/CMakeFiles/gtest_main.dir/src/gtest_main.cc.o  
[ 40%] Linking CXX static library ../../lib/libgtest_main.a  
[ 40%] Built target gtest_main  
[ 50%] Building CXX object CMakeFiles/hello_test.dir/hello_test.cc.o  
[ 60%] Linking CXX executable hello_test  
[ 60%] Built target hello_test  
[ 70%] Building CXX object _deps/googletest-  
build/googmock/CMakeFiles/gmock.dir/src/gmock-all.cc.o  
[ 80%] Linking CXX static library ../../lib/libgmock.a  
[ 80%] Built target gmock  
[ 90%] Building CXX object _deps/googletest-  
build/googmock/CMakeFiles/gmock_main.dir/src/gmock_main.cc.o  
[100%] Linking CXX static library ../../lib/libgmock_main.a  
[100%] Built target gmock_main
```

```
$ cd build && ctest
```

```
Test project /home/xianyubing/C/my_project/build  
  Start 1: HelloTest.BasicAssertions  
1/1 Test #1: HelloTest.BasicAssertions ..... Passed    0.00 sec  
  
100% tests passed, 0 tests failed out of 1  
  
Total Test time (real) = 0.02 sec
```

至此，我们已经使用 `GoogleTest` 成功构建了一个二进制测试。

了解更多可以阅读 `GoogleTest Guide` 的 `GoogleTest Primer` 和 `GoogleTest Samples`。

## 3.使用方法

### 3.1 程序架构

使用 `GoogleTest` 应从写 `assertion` 开始，

`assertion`：判断语句条件是否为真，结果为 `success`、`nonfatal`、`failure` 或 `fatal failure`；当结果为 `fatal failure` 时，它将当前的函数终止，其它结果程序会继续正常执行。

`test suite`：包含一个或多个测试，测试套件应该反应测试代码的结构。

`test fixture`：同一 `test suite` 中的多个测试需要共享相同的对象和子例程时，可以将它们放到 `test fixture` 类中。

`test program`：包含多个 `test suite`。

## 3.2 断言 (Assertion)

gtest是和函数调用相似的宏，我们通过编写端来测试类或函数的行为。当断言失败时，gtest打印带有故障信息的断言源文件和所在行的序号，还可以提供自定义失败消息，该消息将附加到 gtest 的消息中。

断言成对出现，用来测试相同的事物但造成不同的影响。

- `ASSERT_*`：当此类断言失败时，生成fatal failures并终止当前函数，适用于发生此故障后程序没必要继续运行的情况。
- `EXPECT_*`：通常更倾向于使用此类断言，以便于在一个测试中报告更多故障。

由于 `ASSERT_*` 失败时会立刻从当前函数返回，可能会跳过它之后的 `clean-up` 代码，从而造成内存泄漏，它不一定需要修复，但这可能导致我们获得除断言错误之外的一个堆检查错误。

如果想要自定义断言故障消息，仅需要将它用 `<<` 操作符或一系列这种操作符以流的方式写入宏。下面的例子使用了 `ASSERT_EQ` 和 `EXPECT_EQ` 宏来验证值的相等。

```
ASSERT_EQ(x.size(), y.size()) << "vectors x and y are of unequal length";

for (int i = 0; i < x.size(); ++i) {
    EXPECT_EQ(x[i], y[i]) << "vectors x and y differ at index " << i;
}
```

任何可以流入 `ostream` 的内容都可以流入断言宏，包括 `C string` 和 `string` 对象。如果一个宽字符（如 `wchar_t *`，`std::wstring`）串流入断言，则在打印时将它转换为UTF-8类型。

GoogleTest提供了一组断言以不同方式来验证代码行为。我们可以检查布尔条件，基于关系操作符比较值的大小，验证字符串值，浮点值等等；还可以自定义谓词实现更复杂的状态验证。由GoogleTest提供的断言可以查看 [Assertions Reference](#)。

## 3.3 简单示例

创建一个测试：

- (1) 使用 `TEST()` 宏对测试函数定义并命名，它们是没有返回值的C++函数。
- (2) 在这个函数中，可以使用任何合法的C++语句，并使用不同的gtest断言对值进行检查。
- (3) 测试结果由断言决定，如果测试中的任何一个断言失败（致命或非致命），或测试崩溃，则整个测试失败，否则，测试成功。

```
TEST(TestSuiteName, TestName) {
    ... test body ...
}
```

`TEST()` 参数从一般到具体。第一个参数是测试套件的名称，第二个参数是在这个测试套件内部的测试的名称。它们都必须是合法的C++标识符，并且不能包含下划线（`_`）。测试的全名包括它所在的测试套件名和它自己的名称。不同测试套件的测试可以具有相同的名称。

例如下面的整型函数，返回n的阶乘：

```
int Factorial(int n); // Returns the factorial of n
```

这个函数的测试套件可能像下面这样：

```
// Tests factorial of 0.
TEST(FactorialTest, HandlesZeroInput) {
    EXPECT_EQ(Factorial(0), 1);
}

// Tests factorial of positive numbers.
TEST(FactorialTest, HandlesPositiveInput) {
    EXPECT_EQ(Factorial(1), 1);
    EXPECT_EQ(Factorial(2), 2);
    EXPECT_EQ(Factorial(3), 6);
    EXPECT_EQ(Factorial(8), 40320);
}
```

googletest根据测试套件将测试结果分组，所以逻辑相关的测试应该在同一测试套件中；换句话说，它们 `TEST()` 的第一个参数应该相同。上面的例子中有两个测试，`HandlesZeroInput` 和 `HandlesPositiveInput`，它们同属于测试套件 `FactorialTest`。

当命名测试套件和测试时，我们应当遵循相同的函数和类命名的惯例：[https://google.github.io/styleguide/cppguide.html#Function\\_Names](https://google.github.io/styleguide/cppguide.html#Function_Names)。

在Linux, Windows, Mac上可用。

## 3.4 Test Fixtures

当有两个及以上的测试操作在相同的数据上时，可以使用 `test fixture`，从而对几个不同的测试使用相同的对象配置。

创建一个fixture：

(1) 从 `::testing::Test` 派生一个类，以 `protected:` 作为它主体的起始，因为我们希望从子类访问 `fixture` 成员。

(2) 在类的内部，声明我们想要使用的对象。

(3) 如果有必要，为每个测试编写一个默认的构造器或 `SetUp()` 函数来准备对象。注意不要将 `SetUp()` 错写成 `Setup()`，可以在C++ 11中使用 `override` 来保证拼写正确。

(4) 如果有必要，编写一个析构器或 `TearDown()` 函数来释放在 `SetUp()` 中申请的资源，可以阅读 [FAQ](#)来学习如何使用构造器/析构器以及什么时候使用 `SetUp()/TearDown()`。

(5) 如果有需要，可以为测试定义几个用来共享的子例程。

当使用fixture时，使用 `TEST_F()` 而不是 `TEST()`，因为它允许我们访问test fixture中的对象和子例程。

```
TEST_F(TestFixtureName, TestName) {
    ... test body ...
}
```

和 `TEST()` 相似，第一个参数是测试套件名称，但对于 `TEST_F()` 来说，第一个参数必须是test fixture的类名称。

C++的宏系统不允许创建一个单独宏同时处理这两种类型的测试，使用错误的宏会导致编译器报错。

在 `TEST_F()` 中使用test fixture之前，必须定义好test fixture类，否则编译器会报错：`virtual outside class declaration`。

对于 `TEST_F()` 中定义的每一个测试，googletest会创建一个新的test fixture，并立刻用 `Setup()` 对它初始化，运行该测试，并通过调用 `TearDown()` 进行清理，然后删除该test fixture。在同一测试套件中的不同测试具有不同的test fixture对象，googletest在它创建下一个test fixture对象之前会删除一个test fixture。googletest不会对多个测试重用相同的test fixture。单个测试对fixture的任何改变都不会影响到其它测试。

例如，下面的例子是一个类 `Queue`（FIFO队列）的测试，具有如下接口：

```
template <typename E> // E is the element type.
class Queue {
public:
    Queue();
    void Enqueue(const E& element);
    E* Dequeue(); // Returns NULL if the queue is empty.
    size_t size() const;
    ...
};
```

首先定义一个fixture class。按照惯例，我们应当将其命名为 `FooTest`，其中 `Foo` 是被测试的类。

```
class QueueTest : public ::testing::Test {
protected:
    void SetUp() override {
        q1_.Enqueue(1);
        q2_.Enqueue(2);
        q2_.Enqueue(3);
    }

    // void TearDown() override {}

    Queue<int> q0_;
    Queue<int> q1_;
    Queue<int> q2_;
};
```

在这个例子中，由于我们不需要在每个测试后清理，所以我们不需要 `TearDown()`，除了析构函数已经完成的操作。

现在我们可以使用 `TEST_F()` 和这个fixture写测试。

```
TEST_F(QueueTest, IsEmptyInitially) {
    EXPECT_EQ(q0_.size(), 0);
}

TEST_F(QueueTest, DequeueWorks) {
    int* n = q0_.Dequeue();
    EXPECT_EQ(n, nullptr);

    n = q1_.Dequeue();
    ASSERT_NE(n, nullptr);
    EXPECT_EQ(*n, 1);
    EXPECT_EQ(q1_.size(), 0);
    delete n;

    n = q2_.Dequeue();
    ASSERT_NE(n, nullptr);
```

```
EXPECT_EQ(*n, 2);
EXPECT_EQ(q2_.size(), 1);
delete n;
}
```

上面的测试使用了 `ASSERT_*` 断言和 `EXPECT_*` 断言。经验法则是当我们希望测试在断言失败后继续显示更多错误时使用 `EXPECT_*`，在断言失败后继续运行测试毫无意义时使用 `ASSERT_*`。例如，在 `Dequeue` 测试的第二条断言是 `ASSERT_NE(n, nullptr)`，因为我们接下来要取 `n` 指针指向的值，当 `n` 为 `NULL` 时会导致段错误。

当这些测试运行时，会发生下面的事：

- (1) `googletest` 构造一个 `QueueTest` 对象，称为 `t1`。
- (2) `t1.Setup()` 初始化 `t1`。
- (3) 第一个测试 `IsEmptyInitially` 在 `t1` 上运行。
- (4) `t1.TearDown()` 在测试结束后进行清理。
- (5) `t1` 被解构。
- (6) 在另一个 `QueueTest` 对象上重复上述步骤，这次运行 `DequeueWorks` 测试。

在Linux, Windows, Mac上可用。

## 3.5 调用测试

`TEST()` 和 `TEST_F()` 用 `googletest` 隐式注册它们的测试。所以不必像其它C++测试框架那样为了运行我们定义的测试而重新列出这些测试的列表。

在定义我们的测试以后，可以用 `RUN_ALL_TESTS()` 运行它们，如果所有测试均成功，则返回 0，否则返回 1。注意 `RUN_ALL_TESTS()` 运行连接单元中的所有测试，它们可以来自不同的测试套件甚至是不同的源文件。

当 `RUN_ALL_TESTS()` 宏被调用时：

- 存储所有 `googletest` flags 的状态。
- 为第一个测试创建一个 `test fixture` 对象。
- 通过 `setUp()` 对它初始化。
- 在 `fixture` 对象上运行测试。
- 通过 `TearDown()` 清理 `fixture`。
- 删除 `fixture`。
- 恢复所有 `googletest` flags 的状态。
- 对下一个测试重复上述所有步骤，直至测试运行完毕。

如果出现了 `fatal failure`，那么后续步骤都会被终止。

不能忽视 `RUN_ALL_TESTS()` 的返回值，否则编译器会报错。这种设计的基本原理是自动化测试服务根据其退出代码（而不是其 `stdout/stderr` 输出）来确定测试是否通过，因此 `main()` 函数必须返回 `RUN_ALL_TESTS()` 的返回值。并且只能调用 `RUN_ALL_TESTS()` 一次，多次调用它会与一些高级 `googletest` 功能（例如，线程安全的死亡测试）发生冲突，因此不受支持。

在Linux, Windows, Mac上可用。



## 3.6 编写main()函数

大多用户不应该自己编写 `main` 函数，而应该用 `gtest_main`（与 `gtest` 相反）连接，它定义了合适的入口点。本节的其余部分仅在一些需要在测试运行之前进行的、无法在 `fixture` 和测试套件框架内表达的自定义操作时才使用。

如果自己编写 `main` 函数，它应该返回 `RUN_ALL_TESTS()` 的值。

下面是自定义 `main` 函数的样板：

```
#include "this/package/foo.h"

#include "gtest/gtest.h"

namespace my {
namespace project {
namespace {

// The fixture for testing class Foo.
class FooTest : public ::testing::Test {
protected:
    // You can remove any or all of the following functions if their bodies would
    // be empty.

    FooTest() {
        // You can do set-up work for each test here.
    }

    ~FooTest() override {
        // You can do clean-up work that doesn't throw exceptions here.
    }

    // If the constructor and destructor are not enough for setting up
    // and cleaning up each test, you can define the following methods:

    void SetUp() override {
        // Code here will be called immediately after the constructor (right
        // before each test).
    }

    void TearDown() override {
        // Code here will be called immediately after each test (right
        // before the destructor).
    }

    // Class members declared here can be used by all tests in the test suite
    // for Foo.
};

// Tests that the Foo::Bar() method does Abc.
TEST_F(FooTest, MethodBarDoesAbc) {
    const std::string input_filepath = "this/package/testdata/myinputfile.dat";
    const std::string output_filepath = "this/package/testdata/myoutputfile.dat";
    Foo f;
    EXPECT_EQ(f.Bar(input_filepath, output_filepath), 0);
}
```

```
// Tests that Foo does Xyz.
TEST_F(FooTest, DoesXyz) {
    // Exercises the xyz feature of Foo.
}

} // namespace
} // namespace project
} // namespace my

int main(int argc, char **argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

`::testing::InitGoogleTest()` 函数解析命令行以获取 googletest flags，并删除所有已识别的标志。用户可以通过各种标志来控制测试程序的行为，[AdvancedGuide](#) 介绍了这些标志。必须在调用 `RUN_ALL_TESTS()` 之前调用此函数，否则标志将无法正确初始化。

在 Windows 上，`InitGoogleTest()` 也适用于宽字符串，因此它也可用于以 `UNICODE` 模式编译的程序。

编写所有这些 `main` 函数需要花费大量时间，所以 Google Test 提供 `main()` 函数的基础实现，如果它符合需求，只需要将 `gtest_main` 和测试链接起来即可。

## 3.7 局限性

Google Test 被设计为线程安全的。该实现在 pthreads 库可用的系统上是线程安全的。目前在其他系统（例如 Windows）上同时使用来自两个线程的 Google 测试断言是不安全的。在大多数测试中，这不是问题，因为通常断言是在主线程中完成的。

# 4 附录

---

## 4.1 git commit hash获取方法

(1) 首先在googletest的Git页面找到 `release`，跳转到 `Release` 页面后点击图1红框部分（或点击红框下面的 `e2239ee` 可直接跳转到第3步）；

google / googletest

<> Code Issues 166 Pull requests 32 Discussions Actions Security Insights

Releases Tags

Latest release

release-1.11.0

e2239ee

Compare

## v1.11.0

dinord released this on 12 Jun

## Release Notes

GoogleTest will **not** accept patches of new features to v1.11.0. We recommend building GoogleTest from the latest commit instead. Exceptional critical bug fixes may be considered.

(2) 页面跳转后，点击图2红框部分；

google / googletest

<> Code Issues 166 Pull requests 32 Discussions Actions Security Insights

release-1.11.0 18 branches 15 tags

Go to file Code

Abseil Team and dinord Googletest export ✓ e2239ee on 12 Jun 3,689 commits

.github/ISSUE_TEMPLATE	Googletest export	3 months ago
ci	Googletest export	3 months ago
docs	Googletest export	3 months ago
googlemock	Googletest export	3 months ago
googletest	Googletest export	3 months ago

(3) 页面跳转后，图3红框中的内容即为 commit hash。

github.com/google/googletest/commit/e2239ee6043f73722e7aa812a459f54a28552929

Search or jump to... Pull requests Issues Marketplace Explore

google / googletest Watch 1.2k Star 23.6k Fork 7.8k

<> Code Issues 166 Pull requests 32 Discussions Actions Security Insights

### Googletest export

Release GoogleTest 1.11.0

PiperOrigin-RevId: 378861756

master  
release-1.11.0

Abseil Team authored and dinord committed on 12 Jun 1 parent b050d6c commit e2239ee6043f73722e7aa812a459f54a28552929

