陶辉的专栏

linux下分布式海量数据处理。个人网站：taohui.org.cn

如何用googletest写单元测试

标签: googletest 单元测试 google testing 测试 跨平台

2012-03-12 10:55 27266人阅读 评论(4) 收藏 举报

分类:

C/C++(17) 技术分享(43)

googletest是一个用来写C++单元测试的框架,它是跨平台的,可应用在windows、Linux、Mac等
面,我来说明如何使用最新的1.6版本gtest写自己的单元测试。
本文包括以下几部分:1、获取并编译googletest(以下简称为gtest);2、如何编写单元测试用例;3
测试。4、google test内部是如何执行我们的单元测试用例的。

1. 获取并编译gtest

gtest试图跨平台,理论上,它就应该提供多个版本的binary包。但事实上,gtest只提供源码和相应平台的编译方式,这是为什么呢?google的解释是,我们在编译出gtest时,有些独特的工程很可能希望在编译时加许多flag,把编译的过程下放给用户,可以让用户更灵活的处理。这个仁者见仁吧,反正也是免费的BSD权限。

源码的获取地址:http://code.google.com/p/googletest/downloads/list

目前gtest提供的是1.6.0版本,我们看看与以往版本1.5.0的区别:

```
[cpp]

Changes for 1.6.0:

* New feature: ADD_FAILURE_AT() for reporting a test failure at the
  given source location -- useful for writing testing utilities.
。。。 。。。
* Bug fixes and implementation clean-ups.
* Potentially incompatible changes: disables the harmful 'make install'
  command in autotools.
```

就是最下面一行,make install禁用了,郁闷了吧?UNIX的习惯编译方法:./configure;make;make install失灵了,只能说google比较有种,又开始挑战用户习惯了。
那么怎么编译呢?
先进入gtest目录(解压gtest.zip包过程就不说了),执行以下两行命令:

```
[cpp]

g++ -I./include -I./ -c ./src/gtest-all.cc
ar -rv libgtest.a gtest-all.o
```

之后,生成了libgtest.a,这个就是我们要的东东了。以后写自己的单元测试,就需要libgtest.a和gtest目录下的include目录,所以,这1文件1目录我们需要拷贝到自己的工程中。
编译完成后怎么验证是否成功了呢?(相当不友好!)

```
[cpp]

cd ${GTEST_DIR}/make
  make
```

2016年10月 (1)
2014年06月 (1)
展开

阅读排行

浅谈时间函数gettimeofda (36259)
高性能网络编程(一)----a (34878)
“惊群”,看看nginx是怎么解 (33187)
推荐我的新书《深入理解I (31222)
详解rsync算法--如何减少 (29704)
设计模式在C语言中的应 (28618)
高性能网络编程3----TCP (27834)
高性能网络编程7--tcp连接 (27326)
一个低级Illegal instructio (27241)
如何用googletest写单元测 (27232)

评论排行

推荐我的新书《深入理解I (41)
高性能网络编程2----TCP (37)
高性能网络编程(一)----a (36)
详解rsync算法--如何减少 (33)
高性能网络编程3----TCP (33)
谈谈守护进程与僵尸进程 (29)
paxos分布式一致性算法- (25)
高性能网络编程7--tcp连接 (23)
“惊群”,看看nginx是怎么解 (21)
paxos算法如何容错的--讲 (21)

推荐文章

* CSDN日报20170817——《如果不从事编程,我可以做什么?》
* Android自定义EditText:你需要一款简单实用的SuperEditText(一键删除&自定义样式)
* 从JDK源码角度看Integer
* 微信小程序——智能小秘“遥知之”源码分享(语义理解基于olami)
* 多线程中断机制
* 做自由职业者是怎样的体验

最新**评论**

高性能网络编程6--reactor反应堆
lantmayi: epoll怕不是同步IO ?
高性能网络编程6--reactor反应堆
lantmayi: 这不还是同步的么?nginx既然用的是reactor 怎么就称自己是一部非阻塞了
paxos算法如何容错的--讲述五虎
水田如雅: mark
paxos分布式一致性算法--讲述诸
水田如雅: interesting
高性能网络编程(一)----accept建
third_handsome: 讲的特别清楚,但是有一个地方还是每太明白:如果请求建立连接的速度是1000/s,非阻塞socket ...
坑爹的list容器size方法--为了spli
vtfbztvl: 这个问题,其实在Effective STL里面是有提到的
高性能网络编程7--tcp连接的内存

```
03.    ./sample1_unittest
```

如果看到:

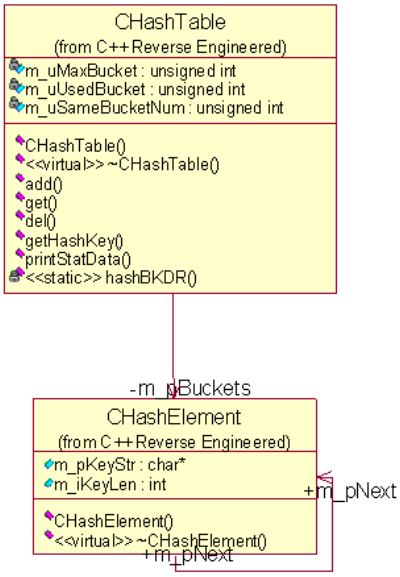
```
[cpp]

01.  Running main() from gtest_main.cc
02.  [==========] Running 6 tests from 2 test cases.
03.  [----------] Global test environment set-up.
04.  [----------] 3 tests from FactorialTest
05.  [ RUN      ] FactorialTest.Negative
06.  [       OK ] FactorialTest.Negative (0 ms)
07.  [ RUN      ] FactorialTest.Zero
08.  [       OK ] FactorialTest.Zero (0 ms)
09.  [ RUN      ] FactorialTest.Positive
10.  [       OK ] FactorialTest.Positive (0 ms)
11.  [----------] 3 tests from FactorialTest (0 ms total)
12.
13.  [----------] 3 tests from IsPrimeTest
14.  [ RUN      ] IsPrimeTest.Negative
15.  [       OK ] IsPrimeTest.Negative (0 ms)
16.  [ RUN      ] IsPrimeTest.Trivial
17.  [       OK ] IsPrimeTest.Trivial (0 ms)
18.  [ RUN      ] IsPrimeTest.Positive
19.  [       OK ] IsPrimeTest.Positive (0 ms)
20.  [----------] 3 tests from IsPrimeTest (0 ms total)
21.
22.  [----------] Global test environment tear-down
23.  [==========] 6 tests from 2 test cases ran. (0 ms total)
24.  [  PASSED  ] 6 tests.
```

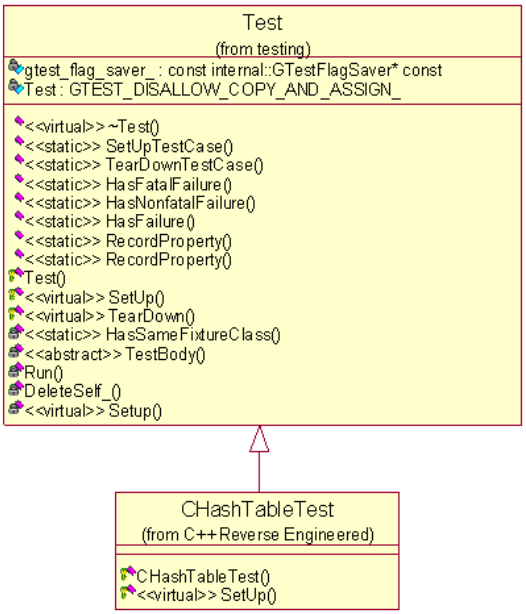
那么证明编译成功了。

2、如何编写单元测试用例

以一个例子来说。我写了一个开地址的哈希表,它有del/get/add三个主要方法需要测试。在测试的时候,很自然,我只希望构造一个哈希表对象,对之做许多种不同组合的操作,以验证三个方法是否正常。所以,gtest提供的TEST方式我不会用,因为多个TEST不能共享同一份数据,而且还有初始化哈希表对象的过程呢。所以我用TEST_F方式。TEST_F是一个宏,TEST_F(classname, casename){}在函数体内去做具体的验证。



上面是我要执行单元测试的类图。那么,我需要写一系列单元测试用例来测试这个类。用gtest,首先要声明一个类,继承自gtest里的Test类:



代码很简单：

```cpp
class CHashTableTest : public ::testing::Test {
protected:
    CHashTableTest():ht(100){

    }
    virtual void SetUp() {
        key1 = "testkey1";
        key2 = "testkey2";
    }

    // virtual void TearDown() {}
    CHashTable ht;

    string key1;
    string key2;
};
```

然后开始写测试用例，用例里可以直接使用上面类中的成员。

```cpp
TEST_F(CHashTableTest, hashfunc)
{
    CHashElement he;

    ASSERT_NE(\
            ht.getHashKey((char*)key1.c_str(), key1.size(), 0),\
            ht.getHashKey((char*)key2.c_str(), key2.size(), 0));

    ASSERT_NE(\
            ht.getHashKey((char*)key1.c_str(), key1.size(), 0),\
            ht.getHashKey((char*)key1.c_str(), key1.size(), 1));

    ASSERT_EQ(\
            ht.getHashKey((char*)key1.c_str(), key1.size(), 0),\
            ht.getHashKey((char*)key1.c_str(), key1.size(), 0));
}
```

注意，TEST_F宏会直接生成一个类，这个类继承自上面我们写的CHashTableTest类。

gtest提供ASSERT_和EXPECT_系列的宏，用于判断二进制、字符串等对象是否相等、真假等等。这两种宏的区

别是，ASSERT_失败了不会往下执行，而EXPECT_会继续。

3、如何执行单元测试

首先，我们自己要有一个main函数，函数内容非常简单：

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

int main(int argc, char** argv) {
    testing::InitGoogleTest(&argc, argv);

    // Runs all tests using Google Test.
    return RUN_ALL_TESTS();
}
```

InitGoogleTest会解析参数。RUN_ALL_TESTS会把整个工程里的TEST和TEST_F这些函数全部作为测试用例执行一遍。

执行时，假设我们编译出的可执行文件叫unittest，那么直接执行./unittest就会输出结果到屏幕，例

```cpp
[==========] Running 4 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 4 tests from CHashTableTest
[ RUN      ] CHashTableTest.hashfunc
[       OK ] CHashTableTest.hashfunc (0 ms)
[ RUN      ] CHashTableTest.addget
[       OK ] CHashTableTest.addget (0 ms)
[ RUN      ] CHashTableTest.add2get
testCHashTable.cpp:79: Failure
Value of: getHe->m_pNext==NULL
  Actual: true
Expected: false
[  FAILED  ] CHashTableTest.add2get (1 ms)
[ RUN      ] CHashTableTest.delget
[       OK ] CHashTableTest.delget (0 ms)
[----------] 4 tests from CHashTableTest (1 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test case ran. (1 ms total)
[  PASSED  ] 3 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] CHashTableTest.add2get
```

```cpp

```

可以看到，对于错误的CASE，会标出所在文件及其行数。

如果我们需要输出到XML文件，则执行./unittest --gtest_output=xml，那么会在当前目录下生成test_detail.xml文件，内容如下：

```cpp
<?xml version="1.0" encoding="UTF-8"?>
<testsuites tests="3" failures="0" disabled="0" errors="0" time="0.001" name="AllTests">
  <testsuite name="CHashTableTest" tests="3" failures="0" disabled="0" errors="0" time="
    <testcase name="hashfunc" status="run" time="0.001" classname="CHashTableTest" />
    <testcase name="addget" status="run" time="0" classname="CHashTableTest" />
    <testcase name="delget" status="run" time="0" classname="CHashTableTest" />
  </testsuite>
</testsuites>
```

如此，一个简单的单元测试写完。因为太简单，所以不需要使用google mock模拟一些依赖。后续我再写结合google mock来写一些复杂的gtest单元测试。

下面来简单说下gtest的工作流程。

4、google test内部是如何执行我们的单元测试用例的

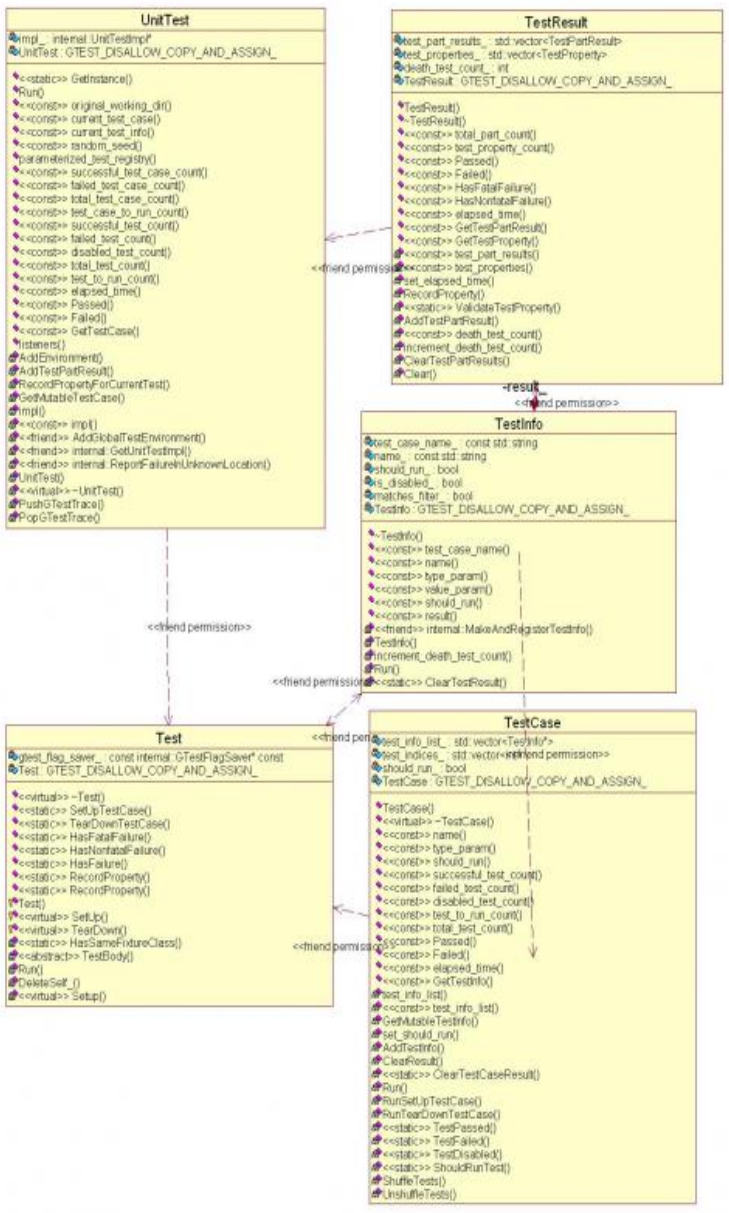
首先从main函数看起。

我们的main函数执行了RUN_ALL_TESTS宏，这个宏干了些什么事呢？

```cpp
#define RUN_ALL_TESTS()\
  (::testing::UnitTest::GetInstance()->Run())

}  // namespace testing
```

原来是调用了UnitTest静态工厂实例的Run方法！在gtest里，一切测试用例都是Test类的实例！所以，Run方法将会执行所有的Test实例来运行所有的单元测试，看看类图：



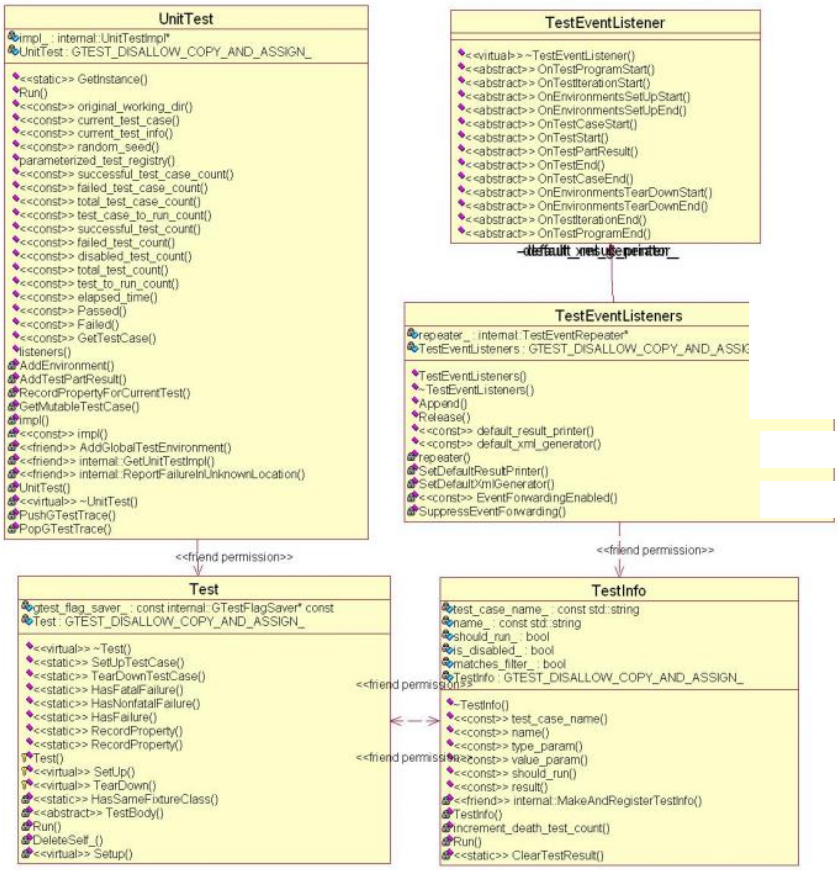
为什么说一切单元测试用例都是Test类的实例呢？

我们有两种写测试用例的方法，一种就是上面我说的TEST_F宏，这要求我们要显示的定义一个子类继承自Test类。在TEST_F宏里，会再次定义一个新类，继承自我们上面定义的子类（两重继承哈）。

第二种就是TEST宏，这个宏里不要求用户代码定义类，但在google test里，TEST宏还是定义了一个子类继承自

Test类。
所以，UnitTest的Run方法只需要执行所有Test实例即可。

每个单元测试用例就是一个Test类子类的实例。它同时与TestResult, TestCase, TestInfo关联起来，用于提供结果。
当然，还有EventListen类来监控结果的输出，控制测试的进度等。



以上并没有深入细节，只是大致帮助大家理解，我们写的几个简单的gtest宏，和单元测试用例，到底是如何被执行的。接下来，我会通过gmock来深入的看看google单元测试的玩法。

上一篇 详解rsync算法--如何减少同步文件时的网络传输量
下一篇 用google mock模拟C++对象

相关文章推荐

- 用google mock模拟C++对象
- 【直播】机器学习之凸优化--马博士
- google test简介
- 【直播】计算机视觉原理及实战--屈教授
- Google Test(GTest)使用方法和源码解析——概况
- 机器学习&数据挖掘7周实训--韦玮
- 如何编译google test的例子？
- 机器学习之数学基础系列--AI100
- 使用Google Test的一个简单例子
- 【套餐】2017软考系统集成项目管理工程师顺利通...
- 在mingw项目中引入googletest
- 【课程】深入探究Linux/VxWorks的设备树--宋宝华
- googletest 学习笔记
- 小试Googletest记二
- googletest试用
- Google Test(GTest)使用方法和源码解析——模板...

查看评论

4楼 Messagezsl 2017-01-12 20:32发表

学习了！

3楼 Nestler 2015-03-01 12:52发表

老师，执行RUN_ALL_TESTS的main文件和所有测试用例都放在一个目录下，然后用一个Makefile单独编译吧？知道怎么用了，但不知道如何融入到自己的工程中。

2楼 陶辉 2012-03-12 21:46发表

都需要继承框架提供的类，框架类做了大量幕后工作。

1楼 margaretMYQ 2012-03-12 21:37发表

恩，Ruby也提供了一个Test的类，有些类似

发表评论

用 户 名： GreatPrograme r

评论内容：

提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场