

方亮的专栏

目录视图

摘要视图

RSS 订阅

个人资料



breaksoftware

访问: 673863次

积分: 8696

等级:

BLDG

6

排名: 第2191名

原创: 201篇

转载: 1篇

译文: 0篇

评论: 441条

文章搜索

博客专栏

利器

IT项目研发过程中的利器

文章:4篇

阅读:442

Libev

libev源码解析

文章:6篇

阅读:676

PE

PE文件和COFF文件格式分析

文章:11篇

阅读:28979

动态链接库

DllMain中不当操作导致死锁问题的分析

文章:9篇

阅读:36501

WMI

WMI技术介绍和应用

文章:24篇

阅读:110414

Google Test

GTest源码解析

文章:11篇

阅读:25231

文章分类

DllMain中的做与不做 (9)

赠书 | 异步2周年,技术图书免费送 每周荐书:渗透测试,K8s、架构(评论送书) 项目管理+代码托管+文档协作,开发更流畅

Google Test(GTest)使用方法和源码解析——预处理技术分析和应用

2016-04-07 23:56

975人阅读

评论(0)

收藏

举报

分类: GTest使用方法和源码解析 (10)

版权声明:本文为博主原创文章,未经博主允许不得转载。

目录(?)

[+]

预处理

在《Google Test(GTest)使用方法和源码解析——概况》最后一部分,我们介绍了GTest的预处理就详细介绍该特性的使用和相关源码。(转载请注明出于breaksoftware的csdn博客)

测试特例级别预处理

Test Fixtures是建立一个固定/已知的环境状态以确保测试可重复并且按照预期方式运行的装置。我们可以实现测试特例级别和之后介绍的测试用例级别的预处理逻辑。

举一个比较常见的例子:我们要测试向数据库插入(id,name,location)这样的三个数据,那要先插入数据(0,Fang,Beijing)。我们第一个测试特例可能需要关注于id这个字段,于是它要在基础数据上做出修改,将(1,Fang,Beijing)插入数据库。第二个测试特例可能需要关注于name字段,于是它要在基础数据上做出修改,将(0,Wang,Beijing)插入数据库。第三个测试特例可能需要关注于location字段,于是它要修改基础数据,将(0,Fang,Nanjing)插入数据库。如果做得鲁莽点,我们在每个测试特例前,先将所有数据填充好,再去操作。但是如果我们将其提炼一下,其实我们发现我们只要在每个特例执行前,获取一份基础数据,然后修改其中本次测试关心的一项就可以了。同时这份基础数据不可以在每个测试特例中被修改——即本次测试特例获取的基础数据不会受之前测试特例对基础数据修改而影响——获取的是一个恒定的数据。

我们看下Test Fixtures类定义及使用规则:

1. Test Fixtures类继承于::testing::Test类。

2. 在类内部使用public或者protected描述其成员,为了保证实际执行的测试子类可以使用其成员变量(这个我们后面会分析下)

3. 在构造函数或者继承于::testing::Test类中的SetUp方法中,可以实现我们需要构造的数据。

4. 在析构函数或者继承于::testing::Test类中的TearDown方法中,可以实现一些资源释放的代码(在3中申请的资源)。

5. 使用TEST_F宏定义测试特例,其第一个参数要求是1中定义的类型;第二个参数是测试特例名。

其中4这步并不是必须的,因为我们的数据可能不是申请来的数据,不需要释放。还有就是“构造函数/析构函数”和“SetUp/TearDown”的选择,对于什么时候选择哪对,本文就不做详细分析了,大家可以参看https://github.com/google/googletest/blob/master/googletest/docs/FAQ.md#should-i-use-the-constructordestructor-of-the-test-fixture-or-the-set-uptear-down-function。一般来说就是构造/析构函数里忌讳做什么就不要在里面做,比如抛出异常等。

我们以一个例子来讲解

```
[cpp]
01. class TestFixtures : public ::testing::Test {
02. public:
03.     TestFixtures() {
04.         printf("\nTestFixtures\n");
05.     };
06.     ~TestFixtures() {
07.         printf("\n~TestFixtures\n");
08.     }
09. protected:
10.     void SetUp() {
11.         printf("\nSetUp\n");
12.         data = 0;
13.     };
```

1 of 6

2017年08月23日 15:56

WMI技术介绍和应用 (24)
Apache服务搭建和插件实现 (7)
网络编程模型的分析、实现和对比 (6)
GTest使用方法和源码解析 (11)
PE文件结构和相关应用 (11)
windows安全 (9)
网络通信 (5)
沙箱 (7)
内嵌及定制Lua引擎技术 (3)
IE控件及应用 (7)
反汇编 (15)
开源项目 (16)
C++ (15)
界面库 (3)
python (11)
疑难杂症 (24)
PHP (8)
Redis (8)
IT项目开发过程中的利器 (4)
libev源码解析 (6)

文章存档

2017年08月 (7)
2017年07月 (4)
2017年05月 (9)
2017年02月 (1)
2016年12月 (10)

展开

阅读排行

使用WinHttp接口实现HT (35595)
WMI技术介绍和应用—— (18359)
如何定制一款12306抢票 (13984)
一种标准CSV格式的介绍 (12486)
一种精确从文本中提取UI (12203)
实现HTTP协议Get、Post (11999)
分析两种Dump(崩溃日志 (11576)
一种解决运行程序报“应用 (11171)
实现HTTP协议Get、Post (11158)
反汇编算法介绍和应用— (10676)

评论排行

使用WinHttp接口实现HT (33)
使用VC实现一个“智能”自 (27)
WMI技术介绍和应用—— (23)
WMI技术介绍和应用—— (20)
实现HTTP协议Get、Post (20)
如何定制一款12306抢票 (17)
在windows程序中嵌入Lu (15)
一个分析“文件夹”选择框 (13)
反汇编算法介绍和应用— (12)
使用VC内嵌Python实现的 (10)

推荐文章

* CSDN日报20170817——《如果不从事编程,我可以做什么?》

```
14.     void TearDown() {  
15.         printf("\nTearDown\n");  
16.     }  
17.     protected:  
18.         int data;  
19. };  
20.  
21.     TEST_F(TestFixtures, First) {  
22.         EXPECT_EQ(data, 0);  
23.         data = 1;  
24.         EXPECT_EQ(data, 1);  
25.     }  
26.  
27.     TEST_F(TestFixtures, Second) {  
28.         EXPECT_EQ(data, 0);  
29.         data = 1;  
30.         EXPECT_EQ(data, 1);  
31.     }
```

First测试特例中,我们修改了data的数据(23行),第24行验证了修改的有效性和正确性。在second的测试特例中,一开始就检测了data数据(第28行),如果First特例中修改data(23行)影响了基础数据,则本次测试失败。将First和Second测试特例的实现定义成一样的逻辑,可以避免编译器造成的执行顺序不确定从而影响看下测试输出

```
[plain]  
01. [-----] 2 tests from TestFixtures  
02. [ RUN      ] TestFixtures.First  
03. TestFixtures  
04. SetUp  
05. TearDown  
06. ~TestFixtures  
07. [ OK      ] TestFixtures.First (9877 ms)  
08. [ RUN      ] TestFixtures.Second  
09. TestFixtures  
10. SetUp  
11. TearDown  
12. ~TestFixtures  
13. [ OK      ] TestFixtures.Second (21848 ms)  
14. [-----] 2 tests from TestFixtures (37632 ms total)
```

可以见得,所有局部测试都是正确的,验证了Test Fixtures类中数据的恒定性。我们从输出应该可以看出来,每个测试特例都是要新建一个新的Test Fixtures对象,并在该测试特例结束时销毁它。这样可以保证数据的干净。

我们来看下其实现的源码,首先我们看下TEST_F的实现

```
[cpp]  
01. #define TEST_F(test_fixture, test_name)\  
02.     GTEST_TEST_(test_fixture, test_name, test_fixture, \  
03.                 ::testing::internal::GetTypeId<test_fixture>())
```

我们再回顾下在《Google Test(GTest)使用方法和源码解析——自动调度机制分析》中分析的TEST宏的实现

```
[cpp]  
01. #define GTEST_TEST(test_case_name, test_name)\  
02.     GTEST_TEST_(test_case_name, test_name, \  
03.                 ::testing::Test, ::testing::internal::GetTestTypeId())
```

可以见得它们的区别就是声明的测试特例类继承于不同的父类。同时使用的是public继承方式,所以子类可以使用父类的public和protected成员。这也是我们在介绍Test Fixtures类编写规则时说的,让使用到的变量置于protected域之下的原因。

```
[cpp]  
01. #define GTEST_TEST_(test_case_name, test_name, parent_class, parent_id)\  
02.     class GTEST_TEST_CLASS_NAME_(test_case_name, test_name) : public parent_class {
```

我们再看下Test Fixtures类对象在框架中是怎么创建、使用和销毁的。

在TestInfo::Run()函数中有Test Fixtures对象和销毁的代码

```
[cpp]  
01. // Creates the test object.  
02. Test* const test = internal::HandleExceptionsInMethodIfSupported(  
    test_fixture, &TestInfo::Run, test_fixture, test_name);
```

* 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“通知之”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

使用WinHttp接口实现HTTP协议(
breaksoftware: @qq_34534425:
你过谦了。多总结、多练习、多借鉴就好了。

使用WinHttp接口实现HTTP协议(
qq_34534425: 代码真心nb,感觉自己写的就是渣渣

朴素、Select、Poll和Epoll网络编程
zhangcunli8499:
@Breaksoftware:多谢

朴素、Select、Poll和Epoll网络编程
breaksoftware:
@zhangcunli8499:这篇
http://blog.csdn.net
/breaksoftwa...

朴素、Select、Poll和Epoll网络编程
zhangcunli8499: 哥们,能传一下完整的代码吗?

C++拾趣——类构造函数的隐式\$
breaksoftware: @wuchalilun:多谢鼓励,其实我就想写出点不一样的地方,哈哈。

C++拾趣——类构造函数的隐式\$
Ray_Chang_988: 其他相关的explicit的介绍文章也看了,基本上explicit的作用也都解释清楚了,但是它们都没...

Redis源码解析——字典结构
breaksoftware: @u011548018:
多谢鼓励

Redis源码解析——字典结构
生无可恋只能打怪升级:就冲这图也得点1024个赞

WMI技术介绍和应用——查询系\$
breaksoftware: @hobbyonline:我认为这种属性的信息不准确是很正常的,因为它的正确与否不会影响到系统在不同...

```
03.         factory_, &internal::TestFactoryBase::CreateTest,  
04.         "the test fixture's constructor");  
05.  
06.     // Runs the test only if the test object was created and its  
07.     // constructor didn't generate a fatal failure.  
08.     if ((test != NULL) && !Test::HasFatalFailure()) {  
09.         // This doesn't throw as all user code that can throw are wrapped into  
10.         // exception handling code.  
11.         test->Run();  
12.     }  
13.  
14.     // Deletes the test object.  
15.     impl->os_stack_trace_getter()->UponLeavingGTest();  
16.     internal::HandleExceptionsInMethodIfSupported(  
17.         test, &Test::DeleteSelf_, "the test fixture's destructor");
```

因为测试特例类继承于Test Fixtures类, Test Fixtures类继承于Test类,所以我们可以通过厂类生成一个Test类对象的指针,这就是它创建的过程。在测试特例运行结束后,第16~17行将销毁该对象。

在Test类的Run方法中,除了调用了子类定义的虚方法,还执行了SetUp和TearDown方法

```
[cpp]  
01.     internal::HandleExceptionsInMethodIfSupported(this, &Test::SetUp, "SetUp()");  
02.     // We will run the test only if SetUp() was successful.  
03.     if (!HasFatalFailure()) {  
04.         impl->os_stack_trace_getter()->UponLeavingGTest();  
05.         internal::HandleExceptionsInMethodIfSupported(  
06.             this, &Test::TestBody, "the test body");  
07.     }  
08.  
09.     // However, we want to clean up as much as possible. Hence we will  
10.     // always call TearDown(), even if SetUp() or the test body has  
11.     // failed.  
12.     impl->os_stack_trace_getter()->UponLeavingGTest();  
13.     internal::HandleExceptionsInMethodIfSupported(  
14.         this, &Test::TearDown, "TearDown()");
```

测试用例级别预处理

这种预处理方式也是要使用Test Fixtures。不同的是,我们需要定义几个静态成员:

1. 静态成员变量,用于指向数据。
2. 静态方法SetUpTestCase()
3. 静态方法TearDownTestCase()

举个例子,我们需要自定义测试用例开始和结束时的行为

- 测试开始时输出Start Test Case
- 测试结束时统计结果

```
[cpp]  
01.     class TestFixturesS : public ::testing::Test {  
02.     public:  
03.         TestFixturesS() {  
04.             printf("\nTestFixturesS\n");  
05.         };  
06.         ~TestFixturesS() {  
07.             printf("\n~TestFixturesS\n");  
08.         }  
09.     protected:  
10.         void SetUp() {  
11.         };  
12.         void TearDown() {  
13.         };  
14.  
15.         static void SetUpTestCase() {  
16.             UnitTest& unit_test = *UnitTest::GetInstance();  
17.             const TestCase& test_case = *unit_test.current_test_case();  
18.             printf("Start Test Case %s \n", test_case.name());  
19.         };  
20.  
21.         static void TearDownTestCase() {  
22.             UnitTest& unit_test = *UnitTest::GetInstance();  
23.             const TestCase& test_case = *unit_test.current_test_case();  
24.             int failed_tests = 0;  
25.             int suc_tests = 0;
```

```
26.         for (int j = 0; j < test_case.total_test_count(); ++j) {
27.             const TestInfo& test_info = *test_case.GetTestInfo(j);
28.             if (test_info.result()->Failed()) {
29.                 failed_tests++;
30.             }
31.             else {
32.                 suc_tests++;
33.             }
34.         }
35.         printf("End Test Case %s. Suc : %d, Failed: %d\n", test_case.name(), suc_tests, fail
36.     );
37. };
38. };
39.
40. TEST_F(TestFixturesS, SUC) {
41.     EXPECT_EQ(1,1);
42. }
43.
44. TEST_F(TestFixturesS, FAI) {
45.     EXPECT_EQ(1,2);
46. }
```

测试用例中, 我们分别测试一个成功结果和一个错误的结果。然后输出如下

```
[cpp]
01. [-----] 2 tests from TestFixturesS
02. Start Test Case TestFixturesS
03. [ RUN      ] TestFixturesS.SUC
04. TestFixturesS
05. ~TestFixturesS
06. [       OK ] TestFixturesS.SUC (2 ms)
07. [ RUN      ] TestFixturesS.FAI
08. TestFixturesS
09. ..\test\gtest_unittest.cc(126): error: Expected: 1
10. To be equal to: 2
11. ~TestFixturesS
12. [  FAILED  ] TestFixturesS.FAI (5 ms)
13. End Test Case TestFixturesS. Suc : 1, Failed: 1
14. [-----] 2 tests from TestFixturesS (12 ms total)
```

从输出上看, SetupTestCase在测试用例一开始时就被执行了, TearDownTestCase在测试用例结束前被执行了。我们看下源码中怎么实现的

```
[cpp]
01. // Runs every test in this TestCase.
02. void TestCase::Run() {
03.     .....
04.     internal::HandleExceptionsInMethodIfSupported(
05.         this, &TestCase::RunSetUpTestCase, "SetUpTestCase()");
06.     .....
07.     for (int i = 0; i < total_test_count(); i++) {
08.         GetMutableTestInfo(i)->Run();
09.     }
10.     .....
11.     internal::HandleExceptionsInMethodIfSupported(
12.         this, &TestCase::RunTearDownTestCase, "TearDownTestCase()");
13.     .....
14. }
```

代码之前了无秘密, 以上节选的内容可以说明其执行的先后关系以及执行的区域。

全局级别预处理

顾名思义, 它是在测试用例之上的一层初始化逻辑。如果我们要使用该特性, 则要声明一个继承于::testing::Environment的类, 并实现其SetUp/TearDown方法。这两个方法的关系和之前介绍Test Fixtures类是一样的。

我们看一个例子, 我们例子中的预处理

- 测试开始时输出Start Test
- 测试结束时统计结果

```
[cpp]
01. namespace testing {
02. namespace internal {
```

```

03. class EnvironmentTest : public ::testing::Environment {
04. public:
05.     EnvironmentTest() {
06.         printf("\nEnvironmentTest\n");
07.     };
08.     ~EnvironmentTest() {
09.         printf("\n~EnvironmentTest\n");
10.     }
11. public:
12.     void SetUp() {
13.         printf("\n~Start Test\n");
14.     };
15.     void TearDown() {
16.         UnitTest& unit_test = *UnitTest::GetInstance();
17.         for (int i = 0; i < unit_test.total_test_case_count(); ++i) {
18.             int failed_tests = 0;
19.             int suc_tests = 0;
20.             const TestCase& test_case = *unit_test.GetTestCase(i);
21.             for (int j = 0; j < test_case.total_test_count(); ++j) {
22.                 const TestInfo& test_info = *test_case.GetTestInfo(j);
23.                 // Counts failed tests that were not meant to fail (those without
24.                 // 'Fails' in the name).
25.                 if (test_info.result()->Failed()) {
26.                     failed_tests++;
27.                 }
28.                 else {
29.                     suc_tests++;
30.                 }
31.             }
32.             printf("End Test Case %s. Suc : %d, Failed: %d\n", test_case.name(
33.         }
34.     };
35. };
36. }
37. }
38.
39. GTEST_API_ int main(int argc, char **argv) {
40.     printf("Running main() from gtest_main.cc\n");
41.     ::testing::AddGlobalTestEnvironment(new testing::internal::EnvironmentTest);
42.     testing::InitGoogleTest(&argc, argv);
43.     return RUN_ALL_TESTS();
44. }

```

EnvironmentTest的代码我们就不讲解了,我们可以关注下::testing::AddGlobalTestEnvironment(new testing::internal::EnvironmentTest);这句,我们要在调用RUN_ALL_TESTS之前,使用该函数将全局初始化对象加入到框架中。通过这种方式,可以猜测出,我们可以加入多个对象到框架中。我们看下源码中对它们的调度

```

[cpp]
01. bool UnitTestImpl::RunAllTests() {
02.     .....
03.     ForEach(environments_, SetUpEnvironment);
04.     .....
05.
06.     // Runs the tests only if there was no fatal failure during global
07.     // set-up.
08.     if (!Test::HasFatalFailure()) {
09.         for (int test_index = 0; test_index < total_test_case_count();
10.             test_index++) {
11.             GetMutableTestCase(test_index)->Run();
12.         }
13.     }
14.     .....
15.     std::for_each(environments_.rbegin(), environments_.rend(),
16.                   TearDownEnvironment);
17.     .....
18. }
19. static void SetUpEnvironment(Environment* env) { env->SetUp(); }
20. static void TearDownEnvironment(Environment* env) { env->TearDown(); }

```

截取的源码已经解释的很清楚了。我们看到environments_是个容器,这也印证了我们对于框架中可以有多个Environment的预期。

顶 踩
0 0

上一篇 [Google Test\(GTest\)使用方法和源码解析——断言的使用方法和解析](#)

下一篇 [Google Test\(GTest\)使用方法和源码解析——自定义输出技术的分析和应用](#)

相关文章推荐

- [Google Test\(GTest\)使用方法和源码解析——概况](#)
- [【直播】机器学习之凸优化--马博士](#)
- [深入解析Gtest](#)
- [【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——自定...](#)
- [机器学习&数据挖掘7周实训--韦玮](#)
- [Google Test\(GTest\)使用方法和源码解析——断言...](#)
- [机器学习之数学基础系列--AI100](#)
- [Google Test\(GTest\)使用方法和源码解析——死亡...](#)
- [【套餐】2017软考系统集成项目管理工程师顺利通...](#)
- [Google Test\(GTest\)使用方法和源码解析——结果...](#)
- [【课程】深入探究Linux/VxWorks的设备树--...](#)
- [Google Test\(GTest\)使用方法和源码解析—](#)
- [Google Test\(GTest\)使用方法和源码解析—](#)
- [《Fragmenti详解之二——基本使用方法》...](#)
- [Google Test\(GTest\)和Google Mock\(GMock\)...](#)


查看评论

暂无评论

发表评论

用户名: GreatProgramer

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

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#) | [合作伙伴](#) | [论坛反馈](#)

[网站客服](#) [杂志客服](#) [微博客服](#) webmaster@csdn.net 400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved 