Google Test(GTest)使用方法和源码解析——断言的使用方法和解析

2016-04-07 23:55 2169人阅读 评论(0) 收藏 举报

分类： GTest使用方法和源码解析（10）

目录(?) [+]

在之前博文的基础上，我们将介绍部分断言的使用，同时穿插一些源码。（转载请指明出于brea 博客）

断言（Assertions）

断言是GTest局部测试中最简单的使用方法，我们之前博文中举得例子都是使用断言去做判断

基础断言

我们先看一个基础的断言

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_TRUE(condition);	EXPECT_TRUE(condition);	condition is true
ASSERT_FALSE(condition);	EXPECT_FALSE(condition);	condition is false

GTest中断言都是成对出现的。即分为两个系列：

1. ASSERT_*系列；
2. EXPECT_*系列；

EXPECT_*系列是比较常用的。在一个测试特例中，如果局部测试使用了EXPECT_*系列函数，它将保证本次局部测试结果不会影响之后的流程。但是ASSERT_*系列在出错的情况下，当前测试特例中剩下的流程就不走了。

```cpp
TEST(BaseCheck, Assert) {
  ASSERT_TRUE(1==1);
  ASSERT_TRUE(2==3);
  ASSERT_TRUE(3==3);
}
TEST(BaseCheck, Expect) {
  EXPECT_TRUE(1==1);
  EXPECT_TRUE(2==3);
  EXPECT_TRUE(3==3);
}
```

上面两个测试特例中，第二个局部测试都是不成立的。由于EXPECT_*不会影响执行流程，所以即使第8行出错，之后的流程（第9行）也执行了。但是ASSERT_*会影响，所以第3行出错后，第4行没有执行。那么GTest是如何做到的呢？我们对比下EXPECT_TRUE和ASSERT_TRUE的实现

```cpp
#define EXPECT_TRUE(condition) \
  GTEST_TEST_BOOLEAN_(condition, #condition, false, true, \
                      GTEST_NONFATAL_FAILURE_)
```

```cpp
#define ASSERT_TRUE(condition) \
  GTEST_TEST_BOOLEAN_(condition, #condition, false, true, \
                      GTEST_FATAL_FAILURE_)
```

可以见得，他们的区别就是在是出错时调用了GTEST_NONFATAL_FAILURE_还是GTEST_FATAL_FAILURE_

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项目研发过程中的利器 (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)
一种准标准CSV格式的介 (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实现自 (10)

推荐文章

* CSDN日报20170817——《如果不从事编程，我可以做什么？》

```cpp
#define GTEST_FATAL_FAILURE_(message) \
  return GTEST_MESSAGE_(message, ::testing::TestPartResult::kFatalFailure)

#define GTEST_NONFATAL_FAILURE_(message) \
  GTEST_MESSAGE_(message, ::testing::TestPartResult::kNonFatalFailure)
```

这儿调用到《Google Test(GTest)使用方法和源码解析——结果统计机制分析》中介绍保存局部测试结果的宏——GTEST_MESSAGE_。但是这个不是重点，重点是GTEST_FATAL_FAILURE_宏调用了return——函数返回了。我们再看下GTEST_TEST_BOOLEAN_的实现

```cpp
#define GTEST_TEST_BOOLEAN_(expression, text, actual, expected, fail) \
  GTEST_AMBIGUOUS_ELSE_BLOCKER_ \
  if (const ::testing::AssertionResult gtest_ar_ = \
      ::testing::AssertionResult(expression)) \
    ; \
  else \
    fail(::testing::internal::GetBoolAssertionFailureMessage(\
        gtest_ar_, text, #actual, #expected).c_str())
```

在出错的情况下，ASSERT_*的else里return了。而EXPECT_*的else没有return。

二进制比较断言

GTest还提供了二进制对比宏

Fatal assertion	Nonfatal assertion	Verifies	全称
ASSERT_EQ(val1,val2);	EXPECT_EQ(val1,val2);	val1 == val2	equal
ASSERT_NE(val1,val2);	EXPECT_NE(val1,val2);	val1 != val2	not equal
ASSERT_LT(val1,val2);	EXPECT_LT(val1,val2);	val1 < val2	less than
ASSERT_LE(val1,val2);	EXPECT_LE(val1,val2);	val1 <= val2	less equal
ASSERT_GT(val1,val2);	EXPECT_GT(val1,val2);	val1 > val2	big than
ASSERT_GE(val1,val2);	EXPECT_GE(val1,val2);	val1 >= val2	big equal

虽然宏很多，但是还是很好记，大家只要记住全称那列，就知道怎么对应关系了。我们再查看下二进制对比系列宏的ASSERT_*和EXPECT_*的区别（以EQ为例）

```cpp
#define ASSERT_EQ(val1, val2) GTEST_ASSERT_EQ(val1, val2)
#define GTEST_ASSERT_EQ(val1, val2) \
  ASSERT_PRED_FORMAT2(::testing::internal:: \
                      EqHelper<GTEST_IS_NULL_LITERAL_(val1)>::Compare, \
                      val1, val2)
```

```cpp
#define EXPECT_EQ(val1, val2) \
  EXPECT_PRED_FORMAT2(::testing::internal:: \
                      EqHelper<GTEST_IS_NULL_LITERAL_(val1)>::Compare, \
                      val1, val2)
```

```cpp
// Binary predicate assertion macros.
#define EXPECT_PRED_FORMAT2(pred_format, v1, v2) \
  GTEST_PRED_FORMAT2_(pred_format, v1, v2, GTEST_NONFATAL_FAILURE_)
#define EXPECT_PRED2(pred, v1, v2) \
  GTEST_PRED2_(pred, v1, v2, GTEST_NONFATAL_FAILURE_)
#define ASSERT_PRED_FORMAT2(pred_format, v1, v2) \
  GTEST_PRED_FORMAT2_(pred_format, v1, v2, GTEST_FATAL_FAILURE_)
#define ASSERT_PRED2(pred, v1, v2) \
  GTEST_PRED2_(pred, v1, v2, GTEST_FATAL_FAILURE_)
```

可以见得它们和基本断言一样——EXPECT在失败的情况下没有return（失败时调用了GTEST_NONFATAL_FAILURE_），而ASSERT在失败的情况下return掉了（失败时调用了GTEST_FATAL_FAILURE_）。

一般来说二进制比较，都是对比其结构体所在内存的内容。C++大部分原生类型都是可以使用二进制对比的。但是对于自定义类型，我们就要定义一些操作符的行为，比如=、<等，我这儿就不举例了。

字符串对比断言

* 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——类构造函数的隐式转
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——类构造函数的隐式转
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——查询系统
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...

对于string类型，可以使用如下宏

Fatal assertion	Nonfatal assertion	Verifies	全称
ASSERT_STREQ(str1,str2);	EXPECT_STREQ(str1,_str_2);	the two C strings have the same content	string equal
ASSERT_STRNE(str1,str2);	EXPECT_STRNE(str1,str2);	the two C strings have different content	string not eq
ASSERT_STRCASEEQ(str1,str2);	EXPECT_STRCASEEQ(str1,str2);	the two C strings have the same content, ignoring case	string (ignori case equal
ASSERT_STRCASENE(str1,str2);	EXPECT_STRCASENE(str1,str2);	the two C strings have different content, ignoring case	string (ignori case not euc

在源码上，string对比宏和二进制对比只是在对比函数的选择上有差异，以Equal为例

```cpp
#define EXPECT_EQ(val1, val2) \
  EXPECT_PRED_FORMAT2(::testing::internal:: \
                      EqHelper<GTEST_IS_NULL_LITERAL_(val1)>::Compare, \
                      val1, val2)
```

```cpp
#define EXPECT_STREQ(s1, s2) \
  EXPECT_PRED_FORMAT2(::testing::internal::CmpHelperSTREQ, s1, s2)
```

浮点**对**比断言

在对比数据方面，我们往往会讨论到浮点数的对比。因为在一些情况下，浮点数的计算精度将影 ... 以这块都会单独拿出来说。GTest对于浮点数的对比也是单独的

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_FLOAT_EQ(val1, val2);	EXPECT_FLOAT_EQ(val1, val2);	the two float values are almost equal
ASSERT_DOUBLE_EQ(val1, val2);	EXPECT_DOUBLE_EQ(val1, val2);	the two double values are almost equal

almost euqal表示两个数只是近似相似，默认的是是指两者的差值在4ULP之内（Units in the Last Place）。我们还可以自己制定精度

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_NEAR(val1, val2, abs_error);	EXPECT_NEAR(val1, val2, abs_error);	the difference between val1 and val2 doesn't exceed the given absolute error

使用方法是

```cpp
ASSERT_NEAR(-1.0f, -1.1f, 0.2f);
ASSERT_NEAR(2.0f, 3.0f, 1.0f);
```

对于浮点数的比较，感兴趣的同学可以查看下GTest的源码，还是有点意思的。

成功失**败**断言

该类断言用于直接标记是否成功或者失败。可以使用SUCCEED()宏标记成功，使用FAIL()宏标记致命错误（同ASSERT_*），ADD_FAILURE()宏标记非致命错误（同EXPECT_*）。举个例子

```cpp
if (Check) {
  SUCCEED();
}
else {
  FAIL();
}
```

我们直接在自己的判断下设置断言。这儿有个地方需要说一下，SUCCEED()宏会调用GTEST_MESSAGE_AT_宏，从而会影响TestResult的test_part_results结构体，这也是唯一的成功情况下影响该结构体的地方。详细的分析可以见《Google Test(GTest)使用方法和源码解析——结果统计机制分析》。

类型**对**比断言

该类断言只有一个::testing::StaticAssertTypeEq<T, T>()。当类型相同时，它不会执行任何内容。如果不同则会引起编译错误。但是需要注意的是，要使代码触发编译器推导类型，否则也会发生编译错误。如

```cpp
template <typename T> class Foo {
 public:
  void Bar() { ::testing::StaticAssertTypeEq<int, T>(); }
};
```

如下的代码就不会引起编译冲突

```cpp
void Test1() { Foo<bool> foo; }
```

但是下面的代码由于引发了编译器的类型推导，所以会触发编译错误

```cpp
void Test2() { Foo<bool> foo; foo.Bar(); }
```

异常断言

异常断言是在断言中接收一定类型的异常，并转换成断言形式。它有如下几种

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_THROW(statement, exception_type);	EXPECT_THROW(statement, exception_type);	statement throws an excep the giv
ASSERT_ANY_THROW(statement);	EXPECT_ANY_THROW(statement);	statem excep any type
ASSERT_NO_THROW(statement);	EXPECT_NO_THROW(statement);	statement doesn't throw a exception

我们举一个例子

```cpp
void ThrowException(int n) {
    switch (n) {
    case 0:
        throw 0;
    case 1:
        throw "const char*";
    case 2:
        throw 1.1f;
    case 3:
        return;
    }
}

TEST(ThrowException, Check) {
    EXPECT_THROW(ThrowException(0), int);
    EXPECT_THROW(ThrowException(1), const char*);
    ASSERT_ANY_THROW(ThrowException(2));
    ASSERT_NO_THROW(ThrowException(3));
}
```

这组测试特例中，我们预期ThrowException在传入0时，会返回int型异常；传入1时，会返回const char*异常。传入2时，会返回异常，但是异常类型我们并不关心。传入3时，不返回任何异常。当然ThrowExeception的实现也是按以上预期设计的。

我们看下源码，我们只看ASSERT_型的，EXPECT_型和ASSERT_型的区别在前文很多次讲到，所以不再罗列代码了。

```cpp
#define ASSERT_THROW(statement, expected_exception) \
  GTEST_TEST_THROW_(statement, expected_exception, GTEST_FATAL_FAILURE_)
#define ASSERT_NO_THROW(statement) \
  GTEST_TEST_NO_THROW_(statement, GTEST_FATAL_FAILURE_)
#define ASSERT_ANY_THROW(statement) \
  GTEST_TEST_ANY_THROW_(statement, GTEST_FATAL_FAILURE_)
```

我们先看最简单的GTEST_TEST_NO_THROW_的实现

```cpp
#define GTEST_TEST_NO_THROW_(statement, fail) \
  GTEST_AMBIGUOUS_ELSE_BLOCKER_ \
  if (::testing::internal::AlwaysTrue()) { \
    try { \
      GTEST_SUPPRESS_UNREACHABLE_CODE_WARNING_BELOW_(statement); \
    } \
    catch (...) { \
      goto GTEST_CONCAT_TOKEN_(gtest_label_testnothrow_, __LINE__); \
    } \
  } else \
    GTEST_CONCAT_TOKEN_(gtest_label_testnothrow_, __LINE__): \
      fail("Expected: " #statement " doesn't throw an exception.\n" \
           "  Actual: it throws.")
```

只要表达式抛出异常，就会goto到else中进行错误处理。

再看下GTEST_TEST_ANY_THROW_的实现

```cpp
#define GTEST_TEST_ANY_THROW_(statement, fail) \
  GTEST_AMBIGUOUS_ELSE_BLOCKER_ \
  if (::testing::internal::AlwaysTrue()) { \
    bool gtest_caught_any = false; \
    try { \
      GTEST_SUPPRESS_UNREACHABLE_CODE_WARNING_BELOW_(statement); \
    } \
    catch (...) { \
      gtest_caught_any = true; \
    } \
    if (!gtest_caught_any) { \
      goto GTEST_CONCAT_TOKEN_(gtest_label_testanythrow_, __LINE__); \
    } \
  } else \
    GTEST_CONCAT_TOKEN_(gtest_label_testanythrow_, __LINE__): \
      fail("Expected: " #statement " throws an exception.\n" \
           "  Actual: it doesn't.")
```

只要抛出异常，就认为是正确的。否则goto到else代码中进行错误处理。

再看下稍微复杂点的GTEST_TEST_THROW_

```cpp
#define GTEST_TEST_THROW_(statement, expected_exception, fail) \
  GTEST_AMBIGUOUS_ELSE_BLOCKER_ \
  if (::testing::internal::ConstCharPtr gtest_msg = "") { \
    bool gtest_caught_expected = false; \
    try { \
      GTEST_SUPPRESS_UNREACHABLE_CODE_WARNING_BELOW_(statement); \
    } \
    catch (expected_exception const&) { \
      gtest_caught_expected = true; \
    } \
    catch (...) { \
      gtest_msg.value = \
          "Expected: " #statement " throws an exception of type " \
          #expected_exception ".\n  Actual: it throws a different type."; \
      goto GTEST_CONCAT_TOKEN_(gtest_label_testthrow_, __LINE__); \
    } \
    if (!gtest_caught_expected) { \
      gtest_msg.value = \
          "Expected: " #statement " throws an exception of type " \
          #expected_exception ".\n  Actual: it throws nothing."; \
      goto GTEST_CONCAT_TOKEN_(gtest_label_testthrow_, __LINE__); \
    } \
  } else \
```

```
    GTEST_CONCAT_TOKEN_(gtest_label_testthrow_, __LINE__): \
      fail(gtest_msg.value)
```

只有接收到了传入的特定类型的异常，否则都会goto到else代码中进行错误处理。

参数名输出断言

在之前的介绍的断言中，如果在出错的情况下，我们会对局部测试相关信息进行输出，但是并不涉及其可能传入的参数。参数名输出断言，可以把参数名和对应的值给输出出来。

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_PRED1(pred1, val1);	EXPECT_PRED1(pred1, val1);	pred1(val1) returns true
ASSERT_PRED2(pred2, val1, val2);	EXPECT_PRED2(pred2, val1, val2);	pred2(val1, val2) returns true
...	...	...

目前版本的GTest支持5个参数的版本ASSERT/EXPECT_PRED5宏。其使用方法是

```cpp
template <typename T1, typename T2>
bool GreaterThan(T1 x1, T2 x2) {
  return x1 > x2;
}
TEST(PredicateAssertionTest, AcceptsTemplateFunction) {
  int a = 5;
  int b = 6;
  ASSERT_PRED2((GreaterThan<int, int>), a, b);
}
```

其输出是

```cpp
error: (GreaterThan<int, int>)(a, b) evaluates to false, where
a evaluates to 5
b evaluates to 6
```

其源码也没什么太多奥秘，只是简单的把结果输出

```cpp
template <typename Pred,
          typename T1,
          typename T2>
AssertionResult AssertPred2Helper(const char* pred_text,
                                  const char* e1,
                                  const char* e2,
                                  Pred pred,
                                  const T1& v1,
                                  const T2& v2) {
  if (pred(v1, v2)) return AssertionSuccess();

  return AssertionFailure() << pred_text << "("
                            << e1 << ", "
                            << e2 << ") evaluates to false, where"
                            << "\n" << e1 << " evaluates to " << v1
                            << "\n" << e2 << " evaluates to " << v2;
}
```

这儿需要注意的是，判断的表达式可以使用if语句判断返回结果，所以最好是bool类型。

子过程中使用断言

经过之前的分析，我们可以想到，如果子过程中使用了断言，则结果输出只会指向子过程，而不会指向父过程中的某个调用。为了便于阅读我们可以使用SCOPED_TRACE宏去标记下位置

```cpp
void Sub(int n) {
    ASSERT_EQ(1, n);
}

TEST(SubTest, Test1) {
    {
        SCOPED_TRACE("A");
```

```
        Sub(2);
    }
    Sub(3);
}
```

其结果输出时标记了下A这行位置，可见如果没有这个标记，是很难区分出是哪个Sub失败的。

```
[plain]
..\test\gtest_unittest.cc(87): error:       Expected: 1
To be equal to: n
      Which is: 2
Google Test trace:
..\test\gtest_unittest.cc(92): A
..\test\gtest_unittest.cc(87): error:       Expected: 1
To be equal to: n
      Which is: 3
```

我们再注意下Sub的实现，其使用了ASSERT_EQ断言，该断言并不会影响Test1测试特例的运行，其原因我们在之前做过分析了。为了消除这种可能存在的误解，GTest推荐使用在子过程中使用ASSERT/EXPECT_NO_FATAL_FAILURE(statement);

如果父过程一定要在子过程发生错误时退出怎么办？我们可以使用::testing::Test::HasFatalFa前线程中是否产生过错误。

```
[cpp]
TEST(SubTest, Test1) {
    {
        SCOPED_TRACE("A");
        Sub(2);
    }
    if (::testing::Test::HasFatalFailure())
        return;
    Sub(3);
}
```


上一篇 Google Test(GTest)使用方法和源码解析——Listener技术分析和应用
下一篇 Google Test(GTest)使用方法和源码解析——预处理技术分析和应用

相关文章推荐

- Google Test(GTest)使用方法和源码解析——私有…
- 用google mock模拟C++对象
- 【直播】机器学习之凸优化--马博士
- 【套餐】2017软考系统集成项目管理工程师顺利通…
- Google Mock(Gmock)简单使用和源码分析——源…
- python3编写简易统计服务器
- 【直播】计算机视觉原理及实战--屈教授
- 【课程】深入探究Linux/VxWorks的设备树--宋宝华
- Google Test(GTest)使用方法和源码解析——Liste…
- Google Test(GTest)使用方法和源码解析——结果…
- 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——自定…
- Google Test(GTest)使用方法和源码解析——死亡…
- Google Test(GTest)使用方法和源码解析——预处…
- 机器学习之数学基础系列--AI100
- google test简介

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