个人资料

breaksoftware

访问：673865次
积分：8696
等级：BLOG 6
排名：第2191名

原创：201篇
转载：1篇
译文：0篇
评论：441条

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

2016-04-07 23:57 1305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(?) [+]

在介绍自定义输出机制之前，我们先了解下AssertResult类型函数。(转载请指明出于breaksol
客)

在函数中使用AssertionResult

AssertionResult只有两种类型：

1. AssertionSuccess()
2. AssertionFailure()

要么成功，要么失败，我们就可以使用基础断言来判断

```cpp
::testing::AssertionResult IsEven(int n) {
    if ((n % 2) == 0)
        return ::testing::AssertionSuccess() << n << " is even";
    else
        return ::testing::AssertionFailure() << n << " is odd";
}

TEST(TestAssertResult, Check) {
    EXPECT_FALSE(IsEven(0));
    EXPECT_TRUE(IsEven(1));
}
```

我们在IsEven函数中输出了额外的内容（Actual中），便于我们之后查看结果

```cpp
Value of: IsEven(0)
  Actual: true (0 is even)
Expected: false
error: Value of: IsEven(1)
  Actual: false (1 is odd)
Expected: true
```

自定义输出断言

如果默认的输出结果不能满足我们的需要，或者我们的类型不支持字符流输出，我们就需要自定义输出。我们可以使用

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_PRED_FORMAT1(pred_format1, val1);	EXPECT_PRED_FORMAT1(pred_format1, val1);	pred_format1(val1) is successful
ASSERT_PRED_FORMAT2(pred_format2, val1, val2);	EXPECT_PRED_FORMAT2(pred_format2, val1, val2);	pred_format2(val1, val2) is successful
...	...	...

这一系列GTest也是有5对函数，最高是ASSERT/EXPECT_PRED_FORMAT5。我们看下使用的例子

```cpp
::testing::AssertionResult IsEven2(const char* expr, int n) {
```

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 (10)

推荐文章

* CSDN日报20170817——《如果不从事编程，我可以做什么？》

```
02.     if ((n % 2) == 0)
03.         return ::testing::AssertionSuccess() << expr << " = " << n << " is even";
04.     else
05.         return ::testing::AssertionFailure() << expr << " = " << n << " is odd";
06. }
07.
08. TEST(TestAssertResult, Check2) {
09.     int a = 0;
10.     int b = 1;
11.     EXPECT_PRED_FORMAT1(IsEven2, a);
12.     EXPECT_PRED_FORMAT1(IsEven2, b);
13. }
```

我们发现，用于判断的表达式要求返回类型是AssertionResult。因为源码底层是

```
[cpp]

01. #define GTEST_ASSERT_(expression, on_failure) \
02.   GTEST_AMBIGUOUS_ELSE_BLOCKER_ \
03.   if (const ::testing::AssertionResult gtest_ar = (expression)) \
04.     ; \
05.   else \
06.     on_failure(gtest_ar.failure_message())
```

其次要求用于判断的表达式第一个参数要是一个const char*类型数据，它用于传递参数的名字。测试输出是

```
[cpp]

01. error: b = 1 is odd
```

自定义类型输出

一些情况下，我们自定义类型可能是个复杂的符合结构。C++编译器并不知道怎么输出它，这个时候我们就需要告诉GTest如何去输出了。目前有两种方式

定义输出运算符函数

比如待测类是class Bar。我们只要定义一个方法

```
[cpp]

01. ::std::ostream& operator<<(::std::ostream& os, const Bar& bar) {
02.     return os << bar.DebugString();  // whatever needed to print bar to os
03. }
```

通过Bar暴露出来的方法将该对象输出。我们看一个例子

```
[cpp]

01. #include <vector>
02. #include <string>
03. using namespace std;
04. class Bar {
05.     class Data {
06.     public:
07.         Data() {
08.             strData = "17";
09.             intData = 11;
10.         }
11.     public:
12.         std::string strData;
13.         int intData;
14.     };
15. public :
16.     std::string DebugString() const {
17.         std::string output = "Bar.Data.strData = ";
18.         output += data.strData;
19.         output += "\t";
20.         output += "Bar.Data.intData = ";
21.         char intBuffer[16] = {0};
22.         itoa(data.intData, intBuffer, 10);
23.         output += string(intBuffer);
24.         return output;
25.     }
26.     Data data;
27. };
28.
29. ::std::ostream& operator<<(::std::ostream& os, const Bar& bar) {
```

* 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...

```
30.     return os << bar.DebugString();  // whatever needed to print bar to os
31. }
32.
33. bool IsCorrectBarIntVector(vector<pair<Bar, int> > bar_ints) {
34.     return false;
35. }
36.
37. TEST(TestSelfDefineOutput, Test1) {
38.     vector<pair<Bar, int> > bar_ints;
39.     Bar bar;
40.     bar_ints.push_back(pair<Bar, int>(bar, 1));
41.     EXPECT_TRUE(IsCorrectBarIntVector(bar_ints))
42.         << "bar_ints = " << ::testing::PrintToString(bar_ints);
43. }
```

我们将Bar设计为一个较为复杂的结构，然后定义了一个函数DebugString用于输出其包含的变量。我们让断言进入出错状态，查看其输出

```
[cpp]

01.   Actual: false
02. Expected: true
03. bar_ints = { (Bar.Data.strData = 17     Bar.Data.intData = 11, 1) }
```

可以看出来，GTest将Vector类型的数据格式化输出（使用了PrintToString方法），并使用我们自己定义的DebugString输出了自定义结构。

这儿有个有趣的地方，PrintToString的实现，比如它是如何判断它是个容器的

```
[cpp]

01. template <typename T>
02. void PrintTo(const T& value, ::std::ostream* os) {
03.   DefaultPrintTo(IsContainerTest<T>(0), is_pointer<T>(), value, os);
04. }
```

```
[cpp]

01. typedef int IsContainer;
02. template <class C>
03. IsContainer IsContainerTest(int /* dummy */,
04.                             typename C::iterator* /* it */ = NULL,
05.                             typename C::const_iterator* /* const_it */ = NULL) {
06.   return 0;
07. }
08.
09. typedef char IsNotContainer;
10. template <class C>
11. IsNotContainer IsContainerTest(long /* dummy */) { return '\0'; }
```

编译器遇到这种情况时，会试着用返回IsContainer的方法去匹配方法，但是如何发现class C没有迭代器，则用返回IsNotContaner的函数取匹配。这样就可以区分模板类是否是容器了。

还有个一is_pointer模板方法，用于判断是否是指针。

```
[cpp]

01. template <typename T>
02. struct is_pointer : public false_type {};
03.
04. template <typename T>
05. struct is_pointer<T*> : public true_type {};
```

在我们的测试例子中，由于数据是个容器，且不是指针。那么将会匹配到

```
[cpp]

01. template <typename C>
02. void DefaultPrintTo(IsContainer /* dummy */,
03.                     false_type /* is not a pointer */,
04.                     const C& container, ::std::ostream* os) {
```

其实DefaultPrintTo方法还有其他两个，只是本次没有匹配到

```
[cpp]

01. template <typename T>
02. void DefaultPrintTo(IsNotContainer /* dummy */,
```

```
03.                     true_type /* is a pointer */,
04.                     T* p, ::std::ostream* os) {
```

```cpp
template <typename T>
void DefaultPrintTo(IsNotContainer /* dummy */,
                    false_type /* is not a pointer */,
                    const T& value, ::std::ostream* os) {
```

定义PrintTo方法

有些时候，输出运算符可能被其他业务逻辑占用了。GTest就提供了一个针对性的方法，定义PrintTo方法，我们可以这么去做

```cpp
void PrintTo(const Bar& bar, ::std::ostream* os) {
    *os << bar.DebugString();  // whatever needed to print bar to os
}
```

那么它在什么时候被调用的呢？PrintToString最终会调到如下的函数中，

```cpp
template <typename C>
void DefaultPrintTo(IsContainer /* dummy */,
                    false_type /* is not a pointer */,
                    const C& container, ::std::ostream* os) {
......
  for (typename C::const_iterator it = container.begin();
       it != container.end(); ++it, ++count) {
......
    internal::UniversalPrint(*it, os);
  }
......
}
```

```cpp
template <typename T1, typename T2>
void PrintTo(const ::std::pair<T1, T2>& value, ::std::ostream* os) {
  *os << '(';
  // We cannot use UniversalPrint(value.first, os) here, as T1 may be
  // a reference type.  The same for printing value.second.
  UniversalPrinter<T1>::Print(value.first, os);
  *os << ", ";
  UniversalPrinter<T2>::Print(value.second, os);
  *os << ')';
}
```

```cpp
template <typename T>
class UniversalPrinter {
 public:
  GTEST_DISABLE_MSC_WARNINGS_PUSH_(4180)
  static void Print(const T& value, ::std::ostream* os) {
    PrintTo(value, os);
  }
  GTEST_DISABLE_MSC_WARNINGS_POP_()
};
```

其中UniversalPrinter<T1>::Print(value.first, os)会被我们定义的PrintTo匹配到，从而被调用。

上一篇 Google Test(GTest)使用方法和源码解析——预处理技术分析和应用
下一篇 Google Test(GTest)使用方法和源码解析——私有属性代码测试技术分析

相关文章推荐

- gtest单元测试
- 【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——预处…
- 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--宋宝华
- Google Test(GTest)使用方法和源码解析——Liste…
- gtest和gmock
- gtest实现架构简单分析
- Google Test(GTest)使用方法和源码解析——私有…

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