随笔-234 文章-10 评论-2047

玩转Google开源C++单元测试框架Google Test系列(gtest)之四 - 参数化

一、前言

在设计测试案例时，经常需要考虑给被测函数传入不同的值的情况。我们之前的做法通常是写一个通用方法，然后编写在测试案例调用它。即使使用了通用方法，这样的工作也是有很多重复性的，程序员都懒，都希望能够少写代码，多复用代码。Google的程序员也一样，他们考虑到了这个问题，并且提供了一个灵活的参数化测试的方案。

二、旧的方案

为了对比，我还是把旧的方案提一下。首先我先把被测函数IsPrime帖过来(在gtest的example1.cc中)，这个函数是用来判断传入的数值是否为质数的。

```
// Returns true iff n is a prime number.
bool IsPrime(int n)
{
    // Trivial case 1: small numbers
    if (n <= 1) return false;

    // Trivial case 2: even numbers
    if (n % 2 == 0) return n == 2;

    // Now, we have that n is odd and n >= 3.

    // Try to divide n by every odd number i, starting from 3
```

随笔档案


```
    for (int i = 3; ; i += 2) {
        // We only have to try i up to the squre root of n
        if (i > n/i) break;

        // Now, we have i <= n/i < n.
        // If n is divisible by i, n is not prime.
        if (n % i == 0) return false;
    }
    // n has no integer factor in the range (1, n), and thus is prime.
    return true;
}
```

假如我要编写判断结果为True的测试案例，我需要传入一系列数值让函数IsPrime去判断是否为True（当然，即使传入再多值也无法确保函数正确，呵呵），因此我需要这样编写如下的测试案例：

```
TEST(IsPrimeTest, HandleTrueReturn)
{
    EXPECT_TRUE(IsPrime(3));
    EXPECT_TRUE(IsPrime(5));
    EXPECT_TRUE(IsPrime(11));
    EXPECT_TRUE(IsPrime(23));
    EXPECT_TRUE(IsPrime(17));
}
```

我们注意到，在这个测试案例中，我至少复制粘贴了4次，假如参数有50个，100个，怎么办？同时，上面的写法产生的是1个测试案例，里面有5个检查点，假如我要把5个检查变成5个单独的案例，将会更加累人。

2014年5月 (2)
2014年4月 (2)
2011年5月 (1)
2011年3月 (1)
2011年1月 (1)
2010年12月 (3)
2010年11月 (3)
2010年10月 (2)
2010年9月 (6)
2010年8月 (2)
2010年7月 (4)
2010年6月 (3)
2010年5月 (4)
2010年4月 (9)
2010年3月 (6)
2010年2月 (3)
2010年1月 (16)
2009年12月 (6)
2009年11月 (3)
2009年10月 (4)
2009年9月 (3)
2009年8月 (2)
2009年7月 (7)
2009年6月 (2)
2009年4月 (12)
2009年3月 (5)
2009年2月 (2)
2009年1月 (3)
2008年12月 (7)

接下来，就来看看gtest是如何为我们解决这些问题的。

三、使用参数化后的方案

1. 告诉gtest你的参数类型是什么

你必须添加一个类，继承testing::TestWithParam<T>，其中T就是你需要参数化的参数类型，比如上面的例子，我需要参数化一个int型的参数

```
class IsPrimeParamTest : public::testing::TestWithParam<int>
{

};
```

2. 告诉gtest你拿到参数的值后，具体做些什么样的测试

这里，我们要使用一个新的宏（嗯，挺兴奋的）：TEST_P，关于这个"P"的含义，Google给出的答案非常幽默，就是说你可以理解为"parameterized" 或者 "pattern"。我更倾向于 "parameterized"的解释，呵呵。在TEST_P宏里，使用GetParam()获取当前的参数的具体值。

```
TEST_P(IsPrimeParamTest, HandleTrueReturn)
{
    int n =  GetParam();
    EXPECT_TRUE(IsPrime(n));
}
```

嗯，非常的简洁！

3. 告诉gtest你想要测试的参数范围是什么

使用INSTANTIATE_TEST_CASE_P这宏来告诉gtest你要测试的参数范围：

2008年11月 (9)
2008年9月 (8)
2008年8月 (7)
2008年7月 (8)
2008年6月 (9)
2008年5月 (33)
2008年4月 (6)
2008年2月 (1)
2007年12月 (3)
2007年11月 (3)
2007年10月 (7)
2007年9月 (1)

系列文章

Python天天美味系列
攻击方式学习系列
瘦客户端那些事
玩转gtest系列

读书笔记

Python网络编程
xUnit Test Patterns
卓有成效的程序员

友情链接

积分与排名

积分 - 547334
排名 - 200

最新评论

```
INSTANTIATE_TEST_CASE_P(TrueReturn, IsPrimeParamTest, testing::Values(3, 5, 11, 23, 17));
```

第一个参数是测试案例的前缀，可以任意取。

第二个参数是测试案例的名称，需要和之前定义的参数化的类的名称相同，如：IsPrimeParamTest

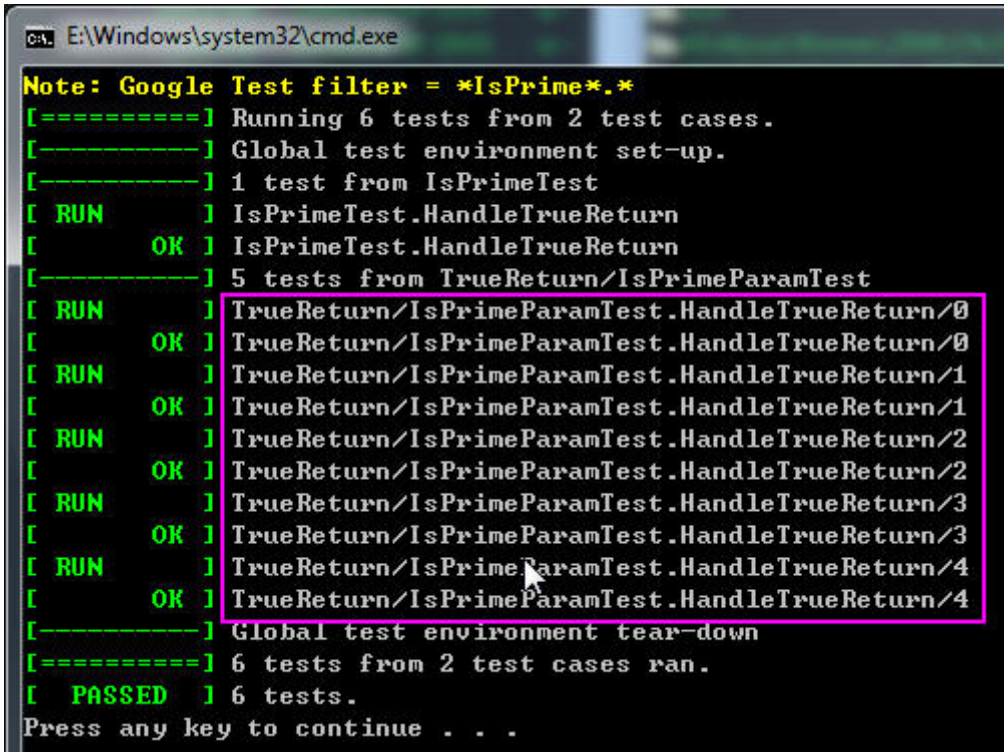
第三个参数是可以理解为参数生成器，上面的例子使用test::Values表示使用括号内的参数。Google提供了一系列的参数生成的函数：

Range(begin, end[, step])	范围在begin~end之间，步长为step，不包括end
Values(v1, v2, ..., vN)	v1,v2到vN的值
ValuesIn(container) and ValuesIn(begin, end)	从一个C类型的数组或是STL容器，或是迭代器中取值
Bool()	取false 和 true 两个值
Combine(g1, g2, ..., gN)	这个比较强悍，它将g1,g2,...gN进行排列组合，g1,g2,...gN本身是一个参数生成器，每次分别从g1,g2,..gN中各取出一个值，组合成一个元组(Tuple)作为一个参数。 说明：这个功能只在提供了<tr1/tuple>头的系统中有效。gtest会自动去判断是否支持tr/tuple，如果你的系统确实支持，而gtest判断错误的话，你可以重新定义宏GTEST_HAS_TR1_TUPLE=1。

四、参数化后的测试案例名

1. Re:gtest参数化测试代码示例
博客园的链接改了，是这个地址：
--canbeing
2. Re:玩转Google开源C++单元测试框架Google Test系列(gtest)之一 - 初识gtest
@xiao_1bai编译的目标就是生成lib文件，你已经成功了。现在可以在你的项目引用gtestd.lib...
--cnbloghzc
3. Re:ViEmuVS2013-3.2.1 破解
安装失败，提示：
所需要的.NET Framework 没有
--xiake007
4. Re:玩转Google开源C++单元测试框架Google Test系列(gtest)之七 - 深入解析gtest
谢谢，作者哥哥，这么多章，最精彩这章。领教了，谢谢
--$JackChen
5. Re:最常用的Emacs的基本操作
如果Emacs入手都算有些难度。。。那VIM怎么办？
--震灵
6. Re:PyQt4学习资料汇总
大神，我下了你的财务管理系统，那个数据库文件提示打不开，怎么解决啊~~急用~~
--lzgst

因为使用了参数化的方式执行案例，我非常想知道运行案例时，每个案例名称是如何命名的。我执行了上面的代码，输出如下：



从上面的框框中的案例名称大概能够看出案例的命名规则，对于需要了解每个案例的名称的我来说，这非常重要。 命名规则大概为：

prefix/test_case_name.test.name/index

五、类型参数化

gtest还提供了应付各种不同类型的数据时的方案，以及参数化类型的方案。我个人感觉这个方案有些复杂。首先要了解一下类型化测试，就用gtest里的例子了。

首先定义一个模版类，继承testing::Test：

```
template <typename T>
class FooTest : public testing::Test {
 public:
  ...
  typedef std::list<T> List;
  static T shared_;
  T value_;
};
```

接着我们定义需要测试到的具体数据类型，比如下面定义了需要测试char,int和unsigned int ：

```
typedef testing::Types<char, int, unsigned int> MyTypes;
TYPED_TEST_CASE(FooTest, MyTypes);
```

又是一个新的宏，来完成我们的测试案例，在声明模版的数据类型时，使用TypeParam

```
TYPED_TEST(FooTest, DoesBlah) {
  // Inside a test, refer to the special name TypeParam to get the type
  // parameter.  Since we are inside a derived class template, C++ requires
  // us to visit the members of FooTest via 'this'.
  TypeParam n = this->value_;

  // To visit static members of the fixture, add the 'TestFixture::'
  // prefix.
  n += TestFixture::shared_;

  // To refer to typedefs in the fixture, add the 'typename TestFixture::'
  // prefix.  The 'typename' is required to satisfy the compiler.
```

4. 玩转Google开源C++单元测试框架Google Test系列(gtest)之三 - 事件机制(68517)
5. 玩转Google开源C++单元测试框架Google Test系列(gtest)之六 - 运行参数(54532)
6. 玩转Google开源C++单元测试框架Google Test系列(gtest)之四 - 参数化(54400)
7. C# 中使用JSON - DataContractJsonSerializer(52402)
8. PyQt4学习资料汇总(49575)
9. 玩转Google开源C++单元测试框架Google Test系列(gtest)之七 - 深入解析gtest(49311)
10. 代码覆盖率浅谈(47554)

评论排行榜

1. PyQt4学习资料汇总(152)
2. 开源Granados介绍 - SSH连接远程Linux服务器(C#)(66)
3. (原创)攻击方式学习之(1) - 跨站式脚本(Cross-Site Scripting) (49)
4. 三年之痒(44)
5. NancyBlog - 我的Google App Engine Blog(42)
6. 创业三年来的一些感想 - 游戏篇(40)
7. CCNET+MSBuild+SVN实时构建的优化总结(40)

```
  typename TestFixture::List values;
  values.push_back(n);
  ...
}
```

上面的例子看上去也像是类型的参数化，但是还不够灵活，因为需要事先知道类型的列表。gtest还提供一种更加灵活的类型参数化的方式，允许你在完成测试的逻辑代码之后再去考虑需要参数化的类型列表，并且还可以重复的使用这个类型列表。下面也是官方的例子：

```
template <typename T>
class FooTest : public testing::Test {
  ...
};

TYPED_TEST_CASE_P(FooTest);
```

接着又是一个新的宏TYPED_TEST_P类完成我们的测试案例：

```
TYPED_TEST_P(FooTest, DoesBlah) {
  // Inside a test, refer to TypeParam to get the type parameter.
  TypeParam n = 0;
  ...
}

TYPED_TEST_P(FooTest, HasPropertyA) { ... }
```

8. CoderZh首款Python联机对战游戏 - NancyTetris1.0倾情发布（一）(37)
9. 代码安全系列(1) - Log的注入(35)
10. 程序员的信仰(35)

推荐排行榜

1. 玩转Google开源C++单元测试框架Google Test系列(gtest)(总)(24)
2. 创业三年来的一些感想 - 游戏篇(14)
3. 程序员的共鸣 - 读《卓有成效的程序员》(12)
4. 代码覆盖率浅谈(12)
5. 《xUnit Test Patterns》学习笔记5 - xUnit基础(10)
6. 三年之痒(9)
7. 玩转Google开源C++单元测试框架Google Test系列(gtest)之一 - 初识gtest(9)
8. 优美的测试代码 - 行为驱动开发(BDD)(8)
9. Python天天美味(总)(7)
10. PyQt4学习资料汇总(6)

接着，我们需要我们上面的案例，使用REGISTER_TYPED_TEST_CASE_P宏，第一个参数是testcase的名称，后面的参数是test的名称

```
REGISTER_TYPED_TEST_CASE_P(FooTest, DoesBlah, HasPropertyA);
```

接着指定需要的类型列表：

```
typedef testing::Types<char, int, unsigned int> MyTypes;
INSTANTIATE_TYPED_TEST_CASE_P(My, FooTest, MyTypes);
```

这种方案相比之前的方案提供更加好的灵活度，当然，框架越灵活，复杂度也会随之增加。

六、总结

gtest为我们提供的参数化测试的功能给我们的测试带来了极大的方便，使得我们可以写更少更优美的代码，完成多种参数类型的测试案例。

系列链接：

1.玩转Google开源C++单元测试框架Google Test系列(gtest)之一 - 初识gtest

2.玩转Google开源C++单元测试框架Google Test系列(gtest)之二 - 断言

3.玩转Google开源C++单元测试框架Google Test系列(gtest)之三 - 事件机制

4.玩转Google开源C++单元测试框架Google Test系列(gtest)之四 - 参数化

5.玩转Google开源C++单元测试框架Google Test系列(gtest)之五 - 死亡测试

6.玩转Google开源C++单元测试框架Google Test系列(gtest)之六 - 运行参数

7.玩转Google开源C++单元测试框架Google Test系列(gtest)之七 - 深入解析gtest

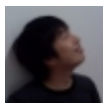
8.玩转Google开源C++单元测试框架Google Test系列(gtest)之八 - 打造自己的单元测试框架

作者：CoderZh
公众号：hacker-thinking（一个程序员的思考）
独立博客：http://blog.coderzh.com
博客园博客将不再更新，请关注我的「微信公众号」或「独立博客」。
作为一个程序员，思考程序的每一行代码，思考生活的每一个细节，思考人生的每一种可能。

微信扫一扫交流

分类: 单元测试,C/C++

标签: Google Test



CoderZh
关注 - 10
粉丝 - 790
+加关注

3 0

« 上一篇：玩转Google开源C++单元测试框架Google Test系列(gtest)之三 - 事件机制
» 下一篇：Google App Engine已经支持JAVA了

posted @ 2009-04-08 02:05 CoderZh 阅读(54400) 评论(15) 编辑 收藏

评论列表

#1楼 2009-05-14 14:19 haonan

老兄的文章思路清晰，比Gtest的原版文章感觉还好

那么难理解的东西，看过实现一下就会了。

这里还有点小的不明之处，向您请教：

在http://code.google.com/p/googletest/wiki/GoogleTestAdvancedGuide#Value_Parameterized_Tests ，提到了如下的内容：

To distinguish different instances of the pattern (yes, you can instantiate it more than once), the first argument to INSTANTIATE_TEST_CASE_P is a prefix that will be added to the actual test case name. Remember to pick unique prefixes for different instantiations. The tests from the instantiation above will have these names:

InstantiationName/FooTest.DoesBlah/0 for "meeny"
InstantiationName/FooTest.DoesBlah/1 for "miny"
InstantiationName/FooTest.DoesBlah/2 for "moe"
InstantiationName/FooTest.HasBlahBlah/0 for "meeny"
InstantiationName/FooTest.HasBlahBlah/1 for "miny"
InstantiationName/FooTest.HasBlahBlah/2 for "moe"

我发现在这个例子中，输出的时候有 for "meeny"、for "miny"之类的文字，能够将参数化的值显示出来，而不仅仅是0、1、2、3这样的序列化数字。

一直想实现，却做不到。包括getst的sample貌似也没有具体的说明。

向您请教，还请指点则个！

支持(0) 反对(0)

#2楼[楼主] 2009-05-14 23:05 CoderZh

@haonan
谢谢。
Google的文档中后面的 for "meeny"等应该是一些辅助说明的内容，并不是案例名称。如：
InstantiationName/FooTest.DoesBlah/0 for "meeny"
其实是说，案例名称是：InstantiationName/FooTest.DoesBlah/0，这个案例是参数为"meeny"的案例。
不过，说实话，我觉得参数化后的案例名称怪怪的，不知道有没有更好的命名方法。

支持(0) 反对(0)

#3楼 2009-05-15 17:51 haonan

@CoderZh
3ks

每次看了过于不过，都不清楚具体的输入了。

如果Gtest这个能够改进，该有多好啊 ：）

支持(0) 反对(0)

#4楼 2009-05-18 16:31 haonan

@CoderZh
试过了，可以建立自己的EXPECT宏，实现自定义输出 ：）

支持(0) 反对(0)

#5楼 2009-08-28 13:45 gsmagic

看了博主的文章受益匪浅
有个问题想请教您，被测函数的输入参数是结构化的指针不知gtest有什么好的办法组织参数呢？
谢谢！

#6楼[楼主] 2009-08-28 16:41 CoderZh

@ gsmagic
你指的“组织参数”是？

支持(0) 反对(0)

#7楼 2009-08-28 17:22 gsmagic

引用

CoderZh：@gsmagic
你指的“组织参数”是？

比如我的被测函数是RecvOrder(struct abc *){}
strcut abc{
int a；
int b；

char c[10];
char d[20];
}
有什么好的办法去组织参数呢？
十分感谢博主！

#8楼[楼主] 2009-08-28 19:24 CoderZh

@ gsmagic
class XXXTest : public::testing::TestWithParam<abc*>
{
}

支持(0) 反对(0)

#9楼 2010-10-09 15:26 三犬风

博主你好，我是看着你文章结合Google的samples入门的，在此真诚感谢...
在上面你给的关于类型参数化得例子中：
TYPED_TEST(FooTest, DoesBlah) {
// Inside a test, refer to the special name TypeParam to get the type
// parameter. Since we are inside a derived class template, C++ requires
// us to visit the members of FooTest via 'this'.
TypeParam n = this->value_;

// To visit static members of the fixture, add the 'TestFixture::'
// prefix.
n += TestFixture::shared_;

// To refer to typedefs in the fixture, add the 'typename TestFixture::'
// prefix. The 'typename' is required to satisfy the compiler.
typename TestFixture::List values;
values.push_back(n);
}
如果该测试宏TYPED_TEST中也含有大量的：
EXPECT_TRUE(IsPrime(3));
EXPECT_TRUE(IsPrime(5));
EXPECT_TRUE(IsPrime(11));
EXPECT_TRUE(IsPrime(23));
EXPECT_TRUE(IsPrime(17));
重复性输入时，能否继续采用参数生成器test::Values来产生参数列表...十分感谢！

支持(0) 反对(0)

#10楼[楼主] 2010-10-10 00:01 CoderZh

@ 三犬风
你好！感谢你的关注。
你提的问题的确很有意思，即数据参数化和类型参数化一起怎么用。我又重温了一遍gtest代码，做了几番尝试之后，觉得在现有的框架基础下很难实现了。除非自己再扩展一下。

支持(0) 反对(0)

#11楼 2010-10-26 20:57 ybtyoyo

类型参数化，我怎么觉得方案一与二没什么区别呢？不知道具体灵活在哪里？求解？？

支持(0) 反对(0)

#12楼 2010-12-06 12:10 2毛

@ CoderZh
请教您一下：如果入参是多个呢？应该怎么去写啊？testing::TestWithParam<T>这个还行么？
还有文中的：使用参数化后的方案里的确三点要同时使用才能达到效果是么？
谢谢~~~~

#13楼 2013-04-12 11:17 zelad

想问一下博主，如果我要使用filter来只运行参数化测试的测试用例，该怎么指定？我试了IsPrimeParamTest.HandleTrueReturn 、 IsPrimeParamTest.* 都不行，运行0个测试

支持(0) 反对(0)

#14楼 2013-04-25 17:25 Dippig

我有一个case，继承testing::TestWithParam<T>，但是使用TEST_P的时候，就是不run这个测试代码， 把他修改成TEST_F, 就没有问题，不知道问题出在那里，望指教！

支持(0) 反对(0)

#15楼 2016-11-02 14:47 超超boy

转眼都到了2016啦，博主的博文还是这么流行。。
前来膜拜。。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

» 更多知识库文章...

站长统计