请将您的问题发送到googlemock讨论组。如果编译器错误需要帮助，请确保首先尝试了谷歌模拟医生。

## 当我在模拟对象上调用一个方法时，实际对象的方法将被调用。是什么问题?

为了模拟一个方法，它必须是虚拟的，除非使用高性能依赖项注入技术。

## 我写了一些匹配器。当我升级到谷歌Mock的新版本后，它们不再编译。这是怎么呢

谷歌Mock的1.4.0版本发布后，我们有了一个想法，如何使编写匹配器更容易，从而有效地生成信息消息。我们尝试了这个想法，并且喜欢我们所看到的。因此，我们决定实施它。

不幸的是，这意味着如果您通过实现MatcherInterface或使用make多态性matcher()定义了自己的匹配器，那么您的定义将不再编译。使用MATCHER\*宏家族定义的匹配器不受影响。

抱歉，如果你的比赛受到影响。我们相信，尽早做出这一改变符合所有人的长远利益。幸运的是，将现有的匹配器迁移到新的API通常并不困难。以下是你需要做的:

如果你这样写你的匹配器:

//旧的匹配器定义，与最新的//谷歌模拟不工作。  
  
测试使用::::MatcherInterface;…  
  
类MyWonderfulMatcher: public MatcherInterface<MyType> {public:…  
  
  
虚拟bool匹配(MyType值)const{//如果值匹配，则返回true。  
  
getfoo () > 5;  
}……};

你需要把它改成:

与最新的谷歌模拟一起工作的新的匹配器定义。  
测试使用::::MatcherInterface;使用::测试::MatchResultListener;…  
  
  
类MyWonderfulMatcher: public MatcherInterface<MyType> {public:…  
  
  
如果值匹配，则返回true。  
  
  
getfoo () > 5;  
}……};

(例如，将Matches()重命名为MatchAndExplain()，并给它第二个MatchResultListener\*类型的参数。)

如果你也使用ExplainMatchResultTo()来改进matcher消息:

//旧的匹配器定义，与最新的//谷歌模拟不工作。  
  
测试使用::::MatcherInterface;…  
  
类MyWonderfulMatcher: public MatcherInterface<MyType> {public:…  
  
  
虚拟bool匹配(MyType值)const{//如果值匹配，则返回true。  
  
getfoo () > 5;  
{//打印一些有用的信息到操作系统，以帮助//用户理解为什么值匹配(或不匹配)。  
  
  
  
  
  
\*os << "the Foo property is " << value.GetFoo();  
}……};

您应该将ExplainMatchResultTo()的逻辑移到MatchAndExplain()中，使用MatchResultListener参数，其中使用了::std::ostream::

与最新的谷歌模拟一起工作的新的匹配器定义。  
测试使用::::MatcherInterface;使用::测试::MatchResultListener;…  
  
  
类MyWonderfulMatcher: public MatcherInterface<MyType> {public:…  
  
  
如果值匹配，则返回true。  
  
  
\*listener << "the Foo property is " << value.GetFoo();  
getfoo () > 5;  
}……};

如果你的匹配器是使用make多态性匹配器()定义的:

//旧的匹配器定义，与最新的//谷歌模拟不工作。  
  
测试使用::::MakePolymorphicMatcher;…  
  
MyGreatMatcher {public:…  
  
  
如果值匹配，则返回true。  
  
getbar () < 42;  
}……},…  
  
  
MakePolymorphicMatcher (MyGreatMatcher ())…

您应该将Matches()方法重命名为MatchAndExplain()，并添加一个MatchResultListener\*参数(与实现MatcherInterface定义的matchers相同):

与最新的谷歌模拟一起工作的新的匹配器定义。  
测试使用::::MakePolymorphicMatcher;使用::测试::MatchResultListener;…  
  
  
MyGreatMatcher {public:…  
  
  
bool MatchAndExplain(MyType值，MatchResultListener\* listener) const{//如果值匹配，返回true。  
  
  
getbar () < 42;  
}……},…  
  
  
MakePolymorphicMatcher (MyGreatMatcher ())…

如果你的多态匹配器使用ExplainMatchResultTo()来获得更好的失败消息:

//旧的匹配器定义，与最新的//谷歌模拟不工作。  
  
测试使用::::MakePolymorphicMatcher;…  
  
MyGreatMatcher {public:…  
  
  
如果值匹配，则返回true。  
  
getbar () < 42;  
{//将一些有用的信息打印到操作系统，以帮助//用户理解为什么值匹配(或不匹配)。  
  
  
  
  
  
  
  
\*os << "the Bar property is " << value.GetBar();}…  
  
MakePolymorphicMatcher (MyGreatMatcher ())…

你需要将ExplainMatchResultTo()中的逻辑移动到MatchAndExplain():

与最新的谷歌模拟一起工作的新的匹配器定义。  
测试使用::::MakePolymorphicMatcher;使用::测试::MatchResultListener;…  
  
  
MyGreatMatcher {public:…  
  
  
bool MatchAndExplain(MyType值，MatchResultListener\* listener) const{//如果值匹配，返回true。  
  
  
\*listener << "the Bar property is " << value.GetBar();  
getbar () < 42;  
}……},…  
  
  
MakePolymorphicMatcher (MyGreatMatcher ())…

要了解更多信息，您可以从cookbook中阅读这两个食谱。如果您需要任何帮助，欢迎您在googlemock@googlegroups.com上提出问题。

## 当使用谷歌Mock时，我必须使用谷歌测试作为测试框架吗?我有我最喜欢的测试框架，不想切换。

谷歌Mock通过谷歌测试可以开箱即用。但是，很容易配置它来与您选择的任何测试框架一起工作。[以下是我的解释。](ForDummies.md#using-google-mock-with-any-testing-framework)

## 我该如何理解这些可怕的模板错误呢?

如果您对gcc抛出的编译器错误感到困惑，请首先尝试咨询谷歌模拟医生工具。它的作用是扫描stdin以查找gcc错误消息，并对代码中出现的问题(我们称之为疾病)进行诊断。

要“安装”，请运行命令:

别名gmd='<路径到googlemock>/scripts/gmock\_doctor.py'

要使用它，请做到:

<your-favorite-build-command> <your-test> 2>&1 | gmd

例如:

使my\_test 2>和1 | gmd

或者您可以运行gmd并复制-n-粘贴gcc的错误消息到它。

## 我可以嘲笑可变参数函数吗?

您不能在谷歌模拟中直接模拟可变参数函数(即使用省略号(…)参数的函数)。

问题是，通常情况下，模拟对象无法知道有多少参数被传递给可变参数方法，以及参数的类型是什么。只有基类的作者知道这个协议，我们无法了解他的想法。

因此，要模拟这样的函数，用户必须教会模拟对象如何计算参数的数量及其类型。一种方法是提供函数的重载版本。

省略参数是从C继承而来的，并不是真正的c++特性。它们使用起来不安全，并且不能使用具有构造函数或析构函数的参数。因此，我们建议在c++中尽量避免它们。

## 当我定义一个带有const参数的mock方法时，MSVC向我发出C4301或C4373警告。为什么?

如果你用Microsoft Visual c++ 2005 SP1编译这个:

类Foo{…  
  
类MockFoo: public Foo{…  
  
  
  
  
MOCK\_METHOD1(Bar, void(const int i));};

你可能会得到以下警告:

警告:C4301: 'MockFoo::Bar':覆盖虚函数只是不同于const/volatile限定符的'Foo::Bar'

这是一个MSVC错误。例如，相同的代码在gcc中可以很好地编译。如果你使用Visual c++ 2008 SP1，你会得到警告:

警告C4373: 'MockFoo::Bar':虚函数覆盖'Foo::Bar'，以前的版本的编译器没有覆盖参数只有不同的const/volatile限定符

在c++中，如果使用const参数声明函数，则会忽略const修饰符。因此，上面的Foo基类等价于:

类Foo{…  
  
虚虚条(int i) = 0;// int还是const int?没有区别。};

实际上，您可以使用int参数声明Bar()，并使用const int参数定义它。编译器仍然会匹配它们。

由于在方法声明中创建参数const是没有意义的，我们建议在Foo和MockFoo中都删除它。这应该可以解决VC bug。

注意，我们在这里讨论的是顶级const修饰符。如果函数参数是通过指针或引用传递的，那么将指针或仲裁声明为const仍然是有意义的。例如，下列两项声明并不等同:

空白栏(int \* p);// p和\*p都不是常量。  
空栏(const int\* p);// p不是常量，但是\*p是常量。

## 我有一个巨大的模拟类，而Microsoft Visual c++在编译它时耗尽了内存。我能做什么?

我们注意到，当使用/clr编译器标志时，Visual c++在编译模拟类时使用的内存是它的5~6倍。我们建议在编译原生c++模拟时避免/clr。

## 我不明白为什么谷歌Mock认为我的期望没有得到满足。我该怎么办?

您可能希望使用——gmock\_verbose=info运行测试。此标志允许谷歌模拟打印它接收到的每个模拟函数调用的跟踪。通过研究跟踪，您将了解为什么您设置的期望没有得到满足。

## 如何断言一个函数从未被调用?

EXPECT\_CALL (foo, Bar(\_))同学(0);

## 我有一个失败的测试，其中谷歌Mock两次告诉我某个特定的期望没有得到满足。这不是多余的吗?

当谷歌Mock检测到失败时，它会打印相关信息(模拟函数参数、相关期望的状态等)来帮助用户调试。如果检测到另一个故障，谷歌Mock将执行相同的操作，包括打印相关期望的状态。

有时，期望的状态在两次失败之间并没有改变，您将两次看到相同的状态描述。但是它们不是多余的，因为它们指的是不同的时间点。它们是一样的，这是个有趣的信息。

## 在使用模拟对象时，会出现堆检查失败，但使用实际对象则没有问题。会有什么问题呢?

您要模拟的类(希望是纯接口)是否有一个虚拟析构函数?

无论何时派生基类，请确保其析构函数是虚的。否则坏事就会发生。考虑以下代码:

类基{public: //不是虚拟的，但应该是。  
  
  
~基地(){…类派生:public Base {public:…  
  
  
  
  
  
  
私人:std:: string value\_;},…  
  
  
  
  
Base\* p = new Derived;  
…  
删除p;/ /惊喜!将调用~Base()，但是不会// - value\_被泄漏。

通过将~Base()更改为virtual，在执行delete p时将正确地调用~Derived()，堆检查器将很高兴。

## “更新的期望覆盖旧的期望”规则使编写期望变得很困难。为什么谷歌Mock会这样做?

当人们抱怨这一点时，他们通常指的是这样的代码:

bar()应该被调用两次，第一次返回1，第二次返回// 2。  
然而，我必须把期望写成//相反的顺序。  
这太糟糕了!!  
() .WillOnce(Return(2)) . reti谐振饱和点();(). EXPECT\_CALL(foo, Bar()) .WillOnce(Return(1)) . reti谐振饱和点();

问题是他们没有选择最好的方式来表达测试的意图。

默认情况下，期望不必以任何特定的顺序匹配。如果希望它们按一定的顺序匹配，则需要显式。这是谷歌Mock(和jMock)的基本原理:很容易意外地过度指定您的测试，我们想让它更难做到这一点。

有两种更好的方法来编写测试规范。

bar()应该被调用两次，第一次返回1，第二次返回// 2。  
使用序列，我们可以按照自然顺序来编写期望//。  
  
{InSequence年代;  
  
.WillOnce(Return(1)) . reti谐振饱和度();  
  
  
.WillOnce(Return(2)) . reti谐振饱和度();}

或者你可以把行动的顺序放在相同的期望中:

bar()应该被调用两次，第一次返回1，第二次返回// 2。  
  
.WillOnce(Return(1)) .WillOnce(Return(2)) . reti谐振饱和度();

回到最初的问题:为什么谷歌模拟从后面到前面搜索期望(和ON\_CALLs) ?因为这允许用户在早期(例如在mock的构造函数或测试装置的设置阶段)为常见情况设置mock的行为，然后使用更具体的规则对其进行自定义。如果谷歌模拟从前到后进行搜索，就不可能使用这种非常有用的模式。

## 当调用没有EXPECT\_CALL的函数时，即使我使用ON\_CALL设置了它的行为，谷歌Mock也会打印一个警告。在这种情况下不显示警告是否合理?

在整洁和安全之间做选择时，我们倾向于后者。所以答案是，我们认为最好显示警告。

人们通常在模拟对象的构造函数或SetUp()中编写ON\_CALLs，因为默认行为很少在测试之间改变。然后在测试体中，他们设置期望，这些期望对于每个测试通常是不同的。在测试的设置部分有一个ON\_CALL并不意味着调用是预期的。如果没有EXPECT\_CALL，并且调用了该方法，那么它可能是一个错误。如果我们在不通知用户的情况下悄悄让调用通过，bug可能会悄悄潜入。

但是，如果您确定这些调用是正常的，则可以进行编写

EXPECT\_CALL (foo, Bar (\_)) .WillRepeatedly (…);

而不是

ON\_CALL (foo, Bar (\_)) .WillByDefault (…);

这告诉谷歌Mock您确实希望调用，并且不应该打印任何警告。

此外，还可以使用——gmock\_verbose标志控制冗长。如果在调试时发现输出太吵，只需选择一个不太冗长的级别。

## 如何在操作中删除模拟函数的参数?

如果需要直接执行一些谷歌Mock不支持的操作，请记住可以使用MakeAction()或make多态性action()定义自己的操作，或者可以编写一个存根函数并使用invoke()调用它。

## MOCK\_METHODn()的第二个参数看起来很有趣。你为什么不使用MOCK\_METHODn(方法，return\_type, arg\_1，…arg\_n)语法呢?

什么? !我认为它很漂亮。:-)

虽然在某种程度上，哪种语法看起来更自然是一个主观问题，但是选择谷歌Mock的语法是因为它有几个实际的优点。

试着模拟一个以地图为参数的函数:

(const map<int, std::string>& m);

使用建议的语法，它将是:

MOCK\_METHOD1(GetSize, int, const map<int, std::string>& m);

你猜怎么着?你会得到一个编译器错误，因为编译器认为const map<int, std::string>& m是两个参数，而不是一个。要解决这个问题，可以使用typedef为映射类型指定一个名称，但这妨碍了您的工作。谷歌Mock的语法避免了这个问题，因为函数的参数类型被保护在一对括号中:

//这个编译得很好。  
MOCK\_METHOD1(GetSize, int(const map<int, std::string>& m));

如果返回类型包含不受保护的逗号，则仍然需要typedef，但这种情况比较少见。

其他优点包括:

1. MOCK\_METHOD1(Foo、int、bool)可能会让读者怀疑该方法返回的是int还是bool，而使用谷歌Mock的语法不会产生这样的混淆。
2. 谷歌Mock描述函数类型的方式并不新鲜，尽管许多人可能不熟悉它。C中使用了相同的语法，而tr1中的函数库广泛地使用了这种语法。由于tr1将成为STL新版本的一部分，我们非常乐意与它保持一致。
3. 在谷歌Mock的API的其他部分(例如action接口)也使用了函数类型语法，以使实现易于处理。为了利用谷歌Mock更高级的特性，用户无论如何都需要学习它。我们还不如在MOCK\_METHOD\*中使用相同的语法!

## 我的代码调用一个静态/全局函数。我可以嘲笑它吗?

你可以，但是你需要做一些改变。

一般来说，如果您发现自己需要模拟一个静态函数，这是您的模块耦合太紧密的一个标志(并且不太灵活、可重用性差、可测试性差，等等)。您最好定义一个小接口并通过该接口调用函数，这样就可以轻松地模拟它。这是一个小的工作，但通常很快支付自己。

这篇谷歌测试博客文章很好地说明了这一点。检查出来。

## 我的模拟对象需要做一些复杂的事情。指定动作是很痛苦的。Google模拟糟透了!

我知道这不是一个问题，但无论如何你都可以得到一个免费的答案。:-)

使用谷歌Mock，您可以轻松地在c++中创建模拟。人们可能会到处使用它们。有时候它们很好用，有时候你会发现，嗯，用起来很麻烦。那么，后一种情况有什么问题呢?

当您在不使用mock的情况下编写测试时，您将测试代码并断言它返回正确的值或系统处于预期的状态。这有时被称为“基于状态的测试”。

模拟非常适合一些所谓的“基于交互”测试:而不是最后检查系统状态,模拟对象调用验证他们的正确方法和报告一个错误就出现了,给你一个处理精确的环境引起的错误。这通常比基于状态的测试更有效、更经济。

如果您正在进行基于状态的测试，并且使用一个测试替身来模拟真实的对象，那么您最好使用一个假的。在这种情况下使用mock会导致问题，因为它不是mock执行复杂操作的强项。如果您经历过这种情况，并且认为模拟很糟糕，那么您只是没有使用正确的工具来解决问题。或者，你可能试图解决错误的问题。:-)

## 我得到了一个警告“遇到了无趣的函数调用-采取了默认的操作..”我应该恐慌吗?

当然不是!只是供参考。

这意味着你有一个模拟函数，你没有对它设置任何期望(根据谷歌模拟规则，这意味着你对这个函数的调用不感兴趣，因此它可以被调用任意次数)，它被调用了。没关系——你没有说调用这个函数是不可以的!

如果您实际上打算不允许调用这个函数，但是却忘记编写EXPECT\_CALL(foo, Bar()).Times(0)，那该怎么办?虽然有人会说这是用户的错，但谷歌Mock会尽量友好一些，并打印一个通知给您。

因此，当您看到消息并相信不应该有任何无趣的调用时，您应该调查发生了什么。为了简化您的工作，当遇到不感兴趣的调用时，谷歌Mock会打印函数名和参数。

## 我想定义一个自定义动作。我应该使用Invoke()还是实现action接口?

任何一种方式都可以——你想要选择一种更适合你的情况的方式。

通常，如果您的操作针对特定的函数类型，那么使用Invoke()定义它应该更容易;如果您的操作可以用于不同类型的函数(例如，如果您正在定义Return(value))，那么make多态性操作()是最简单的。有时候你想要精确的控制action可以用在什么类型的函数中，而实现ActionInterface就是解决这个问题的方法。参见include/gmock/gmock-actions中Return()的实现。以h为例。

## 我使用的是set-argument-pointee操作，编译器会抱怨“指定的返回类型冲突”。这是什么意思?

您得到这个错误，因为谷歌Mock不知道在调用Mock方法时应该返回什么值。SetArgPointee()表示副作用是什么，但没有说明返回值应该是什么。您需要DoAll()将SetArgPointee()与Return()链接起来。

有关详细信息和示例，请参阅本菜谱。

## 我的问题不在你的FAQ里!

如果你不能在这个常见问题解答中找到你的问题的答案，你可以使用一些其他的资源:

1. 阅读其他文档,
2. 搜索邮件列表存档，
3. 在googlemock@googlegroups.com上提问，就会有人回答(为了防止垃圾邮件，我们要求你在发帖之前加入讨论组)。

请注意，在问题跟踪器中创建一个问题并不是获得答案的好方法，因为它很少被少数人监控。

在提问时，提供尽可能多的信息是有帮助的(如果你的问题中没有足够的信息，人们无法帮助你):

* 您使用的谷歌Mock的版本(或者直接从SVN中检出的版本号)(谷歌Mock正在积极开发中，所以您的问题可能在以后的版本中解决了)，
* 你的操作系统,
* 编译器的名称和版本，
* 你给编译器的完整命令行标志，
* 完整的编译器错误消息(如果问题是关于编译的)，
* 遇到问题的实际代码(理想情况下是最小但完整的程序)。