C++ C++11 C++98 C++14 单元测试

如何在 C++ 中对私有函数进行单元测试?

马塞尔·利普

4.38/5 (6 票)

2019 年 10 月 28 日 中央政治局 阅读时间：3分钟 42.4千

本技巧展示了如何对 C++ 中的私有函数进行单元测试。

介绍

自动化软件测试越来越流行。得益于测试驱动开发（TDD）的理念，单元测试的使用越来越频繁。

要实现单元测试，首先我们必须定义什么是单元？在面向对象环境中，通常将类视为一个单元。因此，在单元测试的情况下，我们正在测试类的行为。

这里的一种思路是，只`public`需要测试接口，因为接口仅由类的函数`private`使用，因此如果有人从外部使用我们的类，他/她只会使用接口，而不知道接口。这也是正确的。`publicpublicprivate`

另一方面，一个重要的 KPI 是代码覆盖率，要达到 100% 的代码覆盖率，最简单的方法是`private`单独测试各个功能。在某些情况下，`private`功能确实很复杂，很难通过`public`功能测试所有极端情况。

因此需要测试`private`函数。但测试起来并不容易，因为`private`函数无法从类外部调用。

我收集了一些可能的解决方案来测试它们。这些解决方案专用于 C++，但大多数也可用于其他面向对象编程语言。

可能的解决方案

朋友类

在 C++ 中，`friend`类的类也可以看到其`private`属性和函数。因此，将测试夹具类设为`friend`被测试的类，然后可以对其`private`函数进行单元测试。

但是它的缺点是你必须修改被测试类的代码（通过添加`friend`类的那行）。所以这个解决方案不太优雅。

C++ 收缩 ▲

```
#include

class UnitTestClass;

class ToBeTested
{

private:

   int Calculate() { return 0; }
   friend UnitTestClass;
};

class UnitTestClass
{
public:
    void RunTest()
    {
        ToBeTested object_under_test;
        if (object_under_test.Calculate() == 0)
        {
            std::cout << "Test passed" << std::endl;
        }
        else
        {
            std::cout << "Test failed" << std::endl;;
        }
    }
};

int main()
{
  UnitTestClass unit_tester;
  unit_tester.RunTest();

  return 0;
}
```

定义私人公共

在 C++ 中，预处理器指令为您提供了很多作弊的机会。您可以将这样的代码放入测试文件中：

C++

```
#define private public
#include
#undef private
```

很丑吧？不过还是有用……

使其受保护并继承

您可以将`private`函数更改为`protected`，这样它们的行为就会相同，直到您不再从类继承为止。现在，您可以从被测类继承测试装置类，这样您就可以访问其所有`protected`函数。

这个解决方案的缺点是，您只是为了测试目的而更改生产代码，并且在继承的情况下会降低其安全性。

C++ 收缩 ▲

```
#include

class ToBeTested
{
protected:
    int Calculate() { return 0; }
};

class UnitTestClass : ToBeTested
{
public:
    void RunTest()
    {
        if (Calculate() == 0)
        {
            std::cout << "Test passed" << std::endl;;
        }
        else
        {
            std::cout << "Test failed" << std::endl;;
        }
    }
};

int main()
{
  UnitTestClass unit_tester;
  unit_tester.RunTest();

  return 0;
}
```

将其移至单独的类

让你的代码以最面向对象和最干净的方式完成的解决方案是将要测试的函数移到一个新类中。因此，如果你`Calculate``private`的类中有一个复杂的函数，需要进行单元测试，那么只需将其移到一个新`Calculator`类中即可。或者移到一个辅助函数集合的类中。然后将`private`新类的成员添加到旧类中。这样，你的代码将更具可测试性和可重用性，并且得益于依赖注入等技术，你可以在将来以更优雅的方式交换算法，还可以轻松地模拟它以对类的其他部分进行单元测试。这是我更喜欢的解决方案，因为这样，你将拥有更小的类，并具有清晰、定义良好的职责。

C++ 收缩 ▲

```
#include

class UnitTestClass;

class Calculator
{
    int GetResult() { return 0; }
};

class ToBeTested
{
private:
    Calculator calculator;
};

class UnitTestClass
{
public:
    void RunTest()
    {
        Calculator calculator_under_test;
        if (calculator_under_test.GetResult() == 0)
        {
            std::cout << "Test passed" << std::endl;
        }
        else
        {
            std::cout << "Test failed" << std::endl;;
        }
    }
};

int main()
{
  UnitTestClass unit_tester;
  unit_tester.RunTest();

  return 0;
}
```

概括

还有其他一些技巧可让您测试`private`函数，例如使用预处理器指令或使用`FRIEND_TEST`选项`GTEST`，但归根结底，这些都只是我之前提到的解决方案的版本。

最好的和最受欢迎的解决方案是绝对独立于语言的：让您的架构干净整洁。这始终是创建优秀软件的好方法。

历史

- 2019 年 10 月28日：初始版本

执照