单元测试基础

北京理工大学计算机学院
金旭亮

软件测试概述

当年我年轻时……

- 我不懂测试，“单元测试”？这是什么东东啊？……
- 后来，总算知道了“单元测试”是什么，但总觉得没有必要，一目了然的代码，还测啥？
- 时间那么紧，功能都做不完，哪还有时间写一堆测试代码？
- 赶紧把程序做完领钱是正事！

后来我慢慢地明白了……

那么一些“老掉牙”的东西，是很有道理的……

欲速则不达

磨刀不误砍柴功

凡事预则立，不预则不立

想写出好的软件，必须遵循科学的方法

现在我再写代码，已经习惯于在编写功能代码的同时，同步编写“单元测试”了……

为什么测试是重要的？

手机厂家正在进行按键测试

手机黑窝点里的“生产流水线”

产品质量靠测试！

软件测试之十八般兵器

此图摘自《构建之法——现代软件工程》邹欣著 2014

理解单元测试

问题

如何判定一段代码是“正确的”？

```
//求一个数的倍数
3 references | 0/2 passing
public int DoubleValue(int i)
{
    return i * 2;
}
```

（参看类库项目ClassLib中的MyClass）

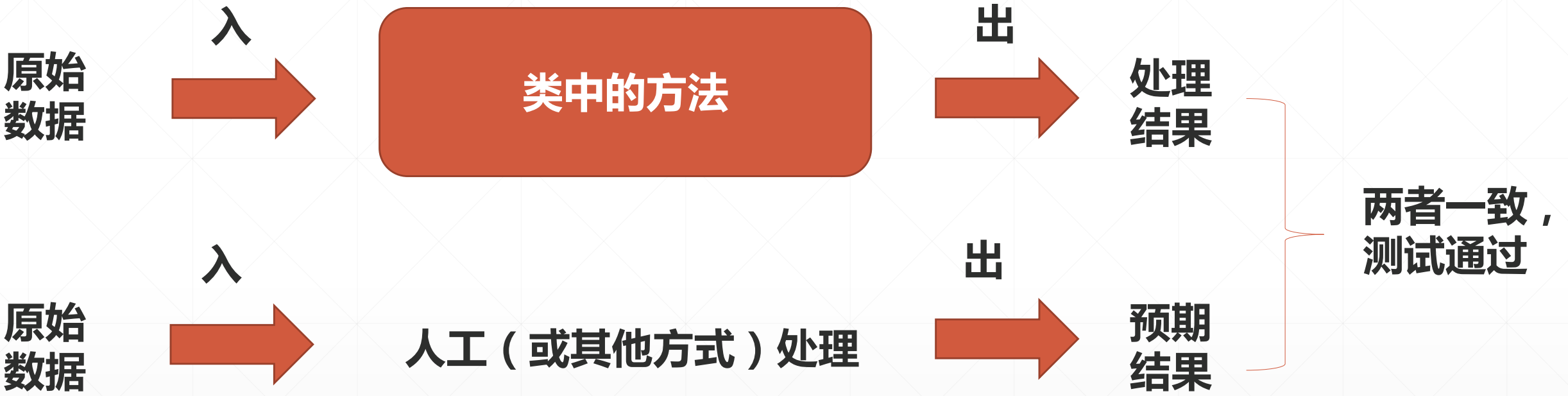
这个代码是否正确地完成了相应的数据处理工作？

```
MyClass obj = new MyClass();
int result = obj.DoubleValue(1);
if (result == 2)
{
    Console.WriteLine("The Code is Correct.");
}
else
{
    Console.WriteLine("The Code is not Correct");
}
```

人工编写的“测试代码正确性”的代码

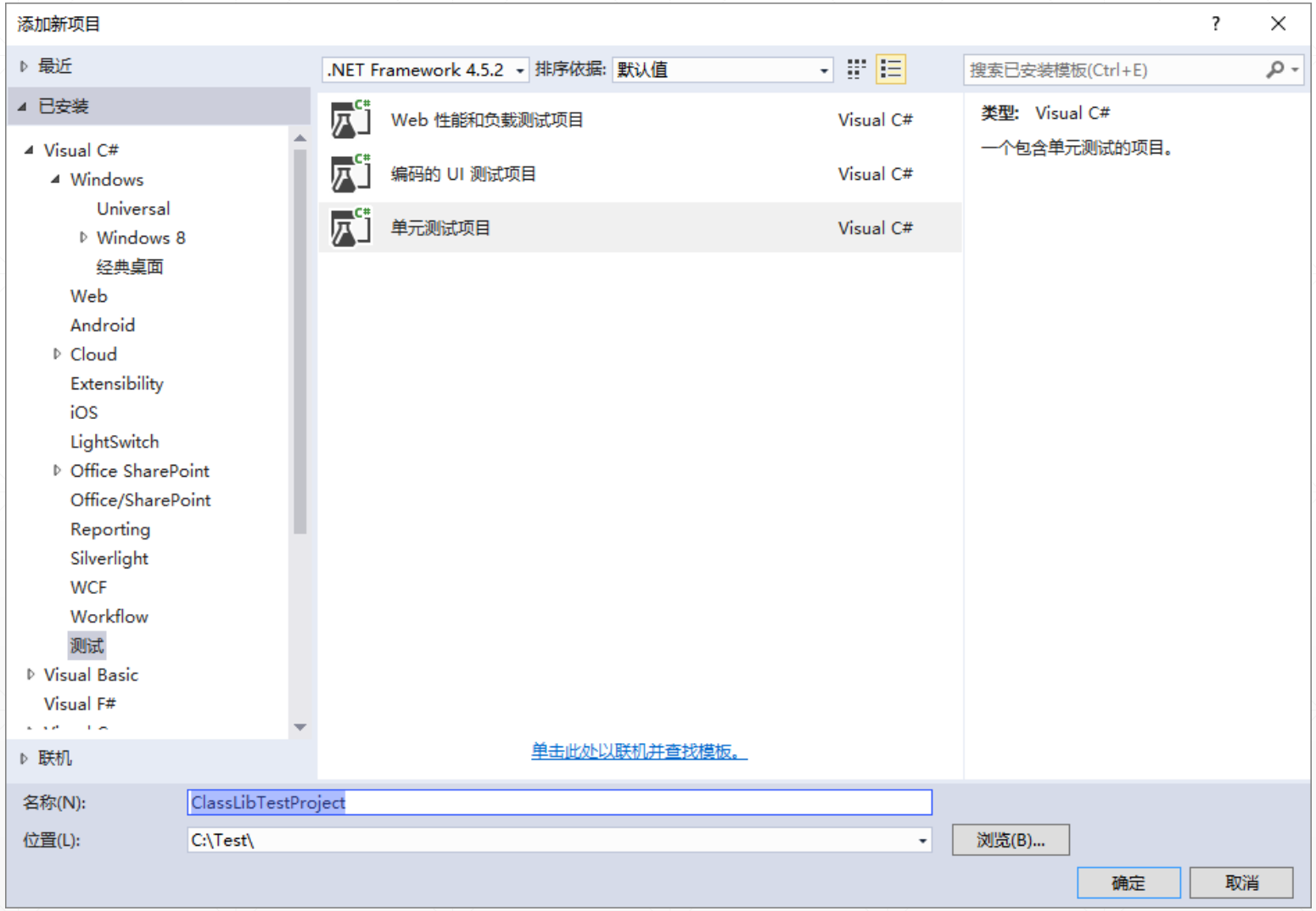
（参看控制台项目UseClassLib）

单元测试是对类中方法"正确性"的测试



输入的数据 + 预期的结果 = 测试用例（Test Case）

Visual Studio“单元测试”项目模板



```
//求一个数的倍数
2 references | 0/2 passing
public int DoubleValue(int i)
{
    //return i * i;
    return i * 2;
}
```

待测试方法

（参看类库项目ClassLib中的MyClass）

单元测试三部曲：

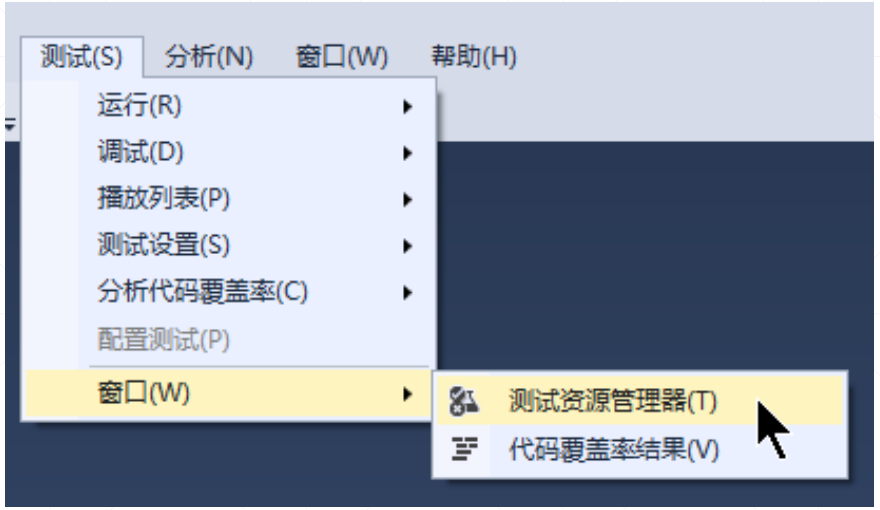
Arrange、Act、Assert

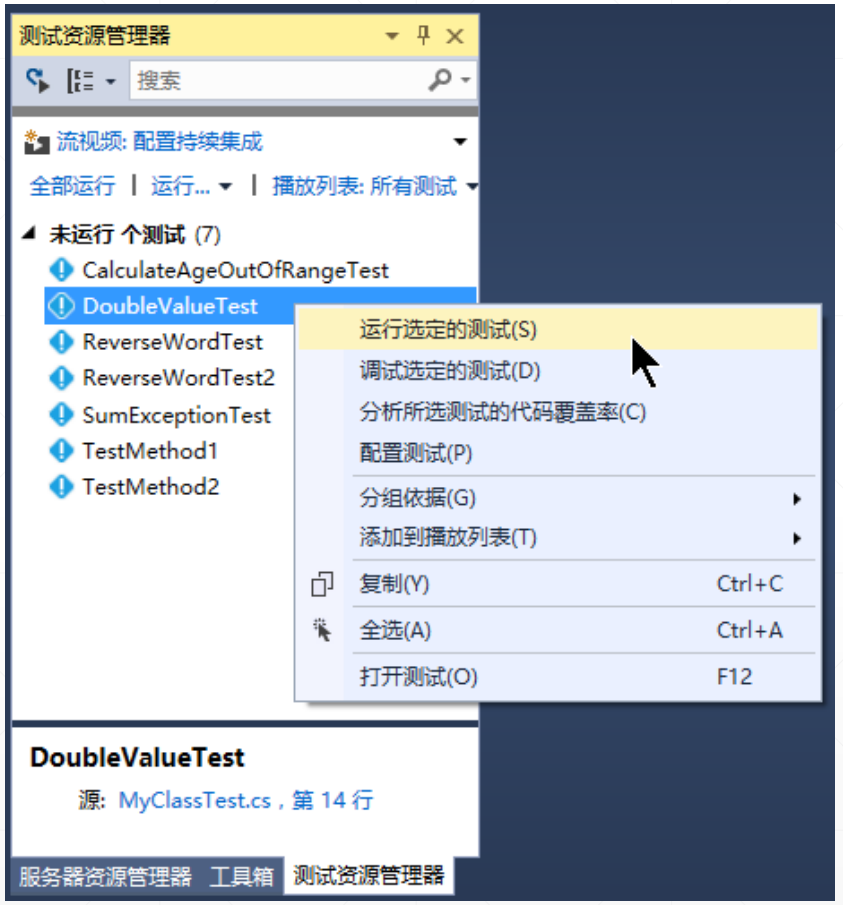
```
[TestMethod]
public void DoubleValueTest()
{
    //1. Arrange:准备
    MyClass target = new MyClass();

    //2. 设定测试用例
    int value = 1;
    int expected = 2;

    //3. Act:执行
    int actual = target.DoubleValue(value);

    //4. Assert:断言
    Assert.AreEqual(expected, actual);

}
```

测试方法

（参看测试项目ClassLibTestProject中的MyClassTest类）

单元测试的运行


测试(S)
分析(N)
窗口(W)
帮助(H)
运行(R)
调试(D)
播放列表(P)
测试设置(S)
分析代码覆盖率(C)
配置测试(P)
窗口(W)
测试资源管理器(T)
代码覆盖率结果(V)


测试资源管理器
搜索
流视频: 配置持续集成
全部运行 | 运行... | 播放列表: 所有测试
未运行 个测试 (7)
CalculateAgeOutOfRangeTest
DoubleValueTest
ReverseWordTest
ReverseWordTest2
SumExceptionTest
TestMethod1
TestMethod2
运行选定的测试(S)
调试选定的测试(D)
分析所选测试的代码覆盖率(C)
配置测试(P)
分组依据(G)
添加到播放列表(T)
复制(Y) Ctrl+C
全选(A) Ctrl+A
打开测试(O) F12
DoubleValueTest
源: MyClassTest.cs，第 14 行
服务器资源管理器
工具箱
测试资源管理器


测试资源管理器
搜索
流视频: 配置持续集成
全部运行 | 运行... | 播放列表: 所有测试
已通过 个测试 (1)
DoubleValueTest 8 毫秒
未运行 个测试 (6)
CalculateAgeOutOfRangeTest
ReverseWordTest
ReverseWordTest2
SumExceptionTest
TestMethod1
TestMethod2
DoubleValueTest
源: MyClassTest.cs，第 14 行
测试 已通过 - DoubleValueTest
运行时间:8 毫秒
服务器资源管理器
工具箱
测试资源管理器

单元测试编写技巧

```
[TestMethod]
0 references
public void DoubleValueTest()
{
    //Arrange:准备
    MyClass target = new MyClass();

    //设定测试用例
    int value = 1;
    int expected = 2;

    //Act:执行
    int actual = target.DoubleValue(value);

    //Assert:断言
    Assert.AreEqual(expected, actual);
}
```

Visual Studio使用[TestClass]和[TestMethod]标识单元测试代码

MSTest测试框架中，为Assert类设计了一系列的方法实现“断言”

Assert所提供的断言方法分类

断言方法分类	方法名
“相等性”断言	AreEqual , AreNotEqual
“对象引用”断言	AreSame , AreNotSame , Contains
“满足条件”断言	IsTrue, IsFalse, IsNull, IsNotNull, IsNaN, IsEmpty, IsNotEmpty
“对象类型”断言	IsInstanceOf<T>, IsNotInstanceOf<T>, IsAssignableFrom<T>, …
“抛出异常”断言	Assert.Throws

测试期望抛出的异常

```
//计算从[from,to]中所有整数的和
1 reference | 0/1 passing
public int Sum(int from, int to)
{
    if (from > to)
    {
        throw new ArgumentException("参数from必须小于to");
    }
    int sum = 0;
    for (int i = from; i <= to; i++)
    {
        sum += i;
    }
    return sum;
}
```

```
[TestMethod]
[ExpectedException(typeof(ArgumentException))]
0 references
public void SumExceptionTest()
{
    target.Sum(100, 50);

}
```

另两个相关Assert类型

断言类型	相关的断言方法
StringAssert	Contains, StartsWith, EndsWith, AreEqualIgnoringCase, IsMatch(regex)
CollectionAssert	AllItemsAre, InstancesOfType, Unique, Equal, Equivalent…

```
public const String AgeErrorString = "生日必须小于当前日期";
//给定一个人的生日，计算他的年纪
1 reference | 1/1 passing
public int CaculateAge(DateTime Birthday)
{
    if (DateTime.Now < Birthday)
    {
        throw new ArgumentOutOfRangeException(AgeErrorString);
    }
    return DateTime.Now.Year - Birthday.Year;
}
```

```
[TestMethod]
0 references
public void CalculateAgeOutOfRangeTest()
{
    try
    {
        //Arrange:准备
        MyClass target = new MyClass();
        target.CaculateAge(DateTime.Now.AddDays(1));
    }
    catch (ArgumentOutOfRangeException e)
    {
        StringAssert.Contains(e.Message, MyClass.AgeErrorString);
        return;
    }
    Assert.Fail("No exception was thrown.");
}
```

每次测试前执行特定的代码

```
[TestInitialize]
0 references
public void TestInitialize()
{
    Console.WriteLine("Begin Test");
}
[TestCleanup]
0 references
public void TestCleanup()
{
    Console.WriteLine("Test finished");
}

[TestMethod]
0 references
public void TestMethod1()
{
    Console.WriteLine("Running TestMethod1");
}
```

测试名称: TestMethod1

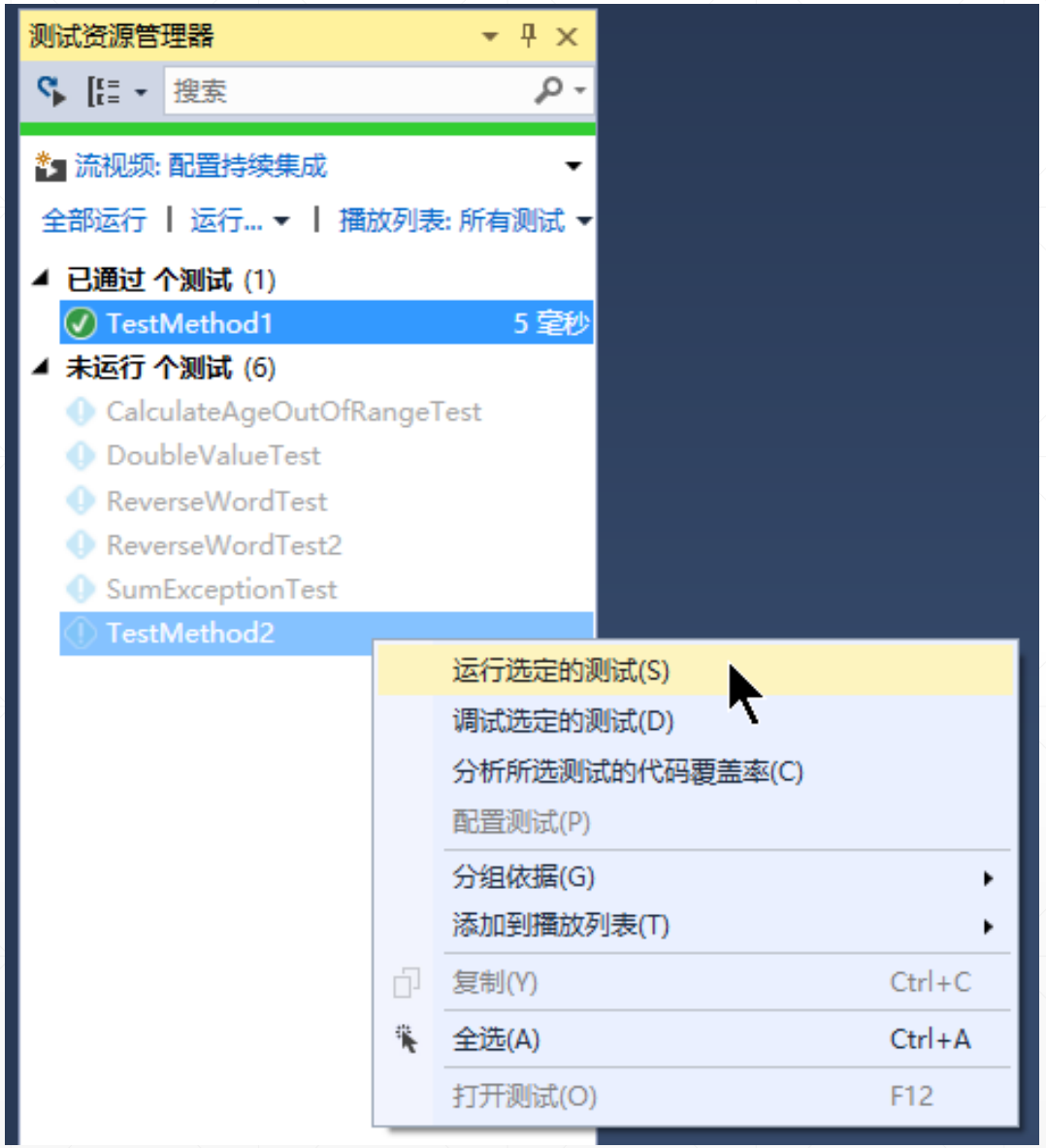
测试结果: 已通过

标准输出

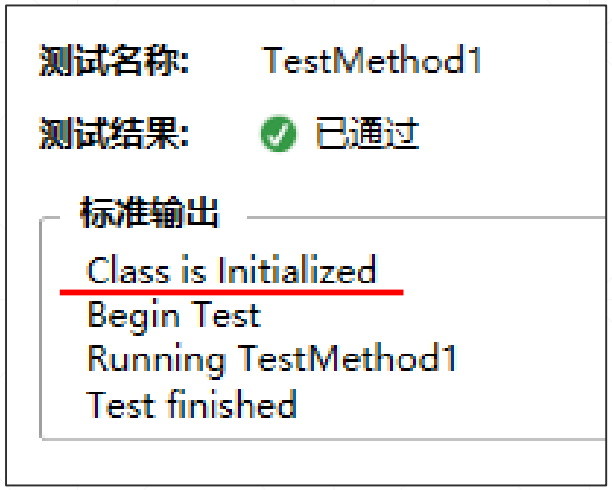
Begin Test
Running TestMethod1
Test finished

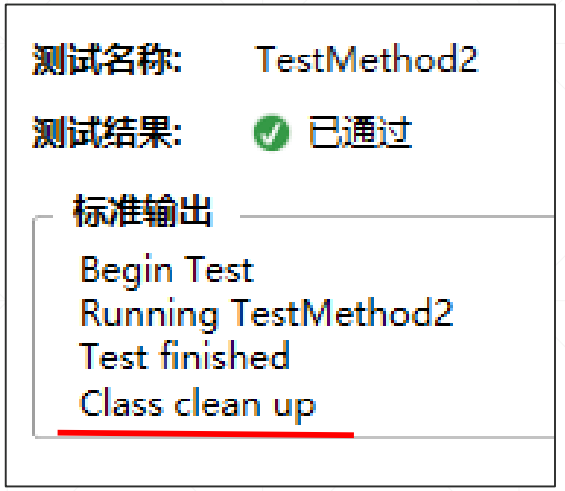
所有测试前/后执行代码

```
[ClassInitialize]
0 references
public static void MyClassInitialize(TestContext context)
{
    Console.WriteLine("Class is Initialized");
}
[ClassCleanup]
0 references
public static void MyClassCleanup()
{
    Console.WriteLine("Class clean up");
}

[TestMethod]
0 references
public void TestMethod1()
{
    Console.WriteLine("Running TestMethod1");
}
[TestMethod]
0 references
public void TestMethod2()
{
    Console.WriteLine("Running TestMethod2");
}
```



测试结果





在执行第一个测试代码之前，执行了有**[ClassInitialize]**标识的方法

在执行所有测试方法代码之后，执行了有**[ClassCleanup]**标识的方法

关于单元测试……

编写单元测试代码注意事项

1. 每次只针对一个特性编写测试代码
2. 使用足够多的“断言（Assert）”验证代码的正确性
3. 不测试过于简单的方法：比如get和set方法
4. 不测试私有方法

编写单元测试的FIRST原则

- **Fast**
  - 单元测试必须运行得很快
  - 没有被反复运行的单元测试，毫无价值
- **Independent**
  - 单元测试方法的执行顺序无关紧要
  - 单元测试的各个方法之间不应该相互依赖
- **Repeatable**
  - 功能代码不改的前提下，相同的测试代码多次运行，应该得到相同的结果
- **Self-validating**
  - 单元测试方法只有两个可能的运行结果：通过或失败，没有第三种情况.
- **Timely**
  - 功能代码与测试代码“配对”编写，步步为营，小步快走

仔细考虑单元测试的必要性

1. 是否应该进行单元测试，取决于具体情况。
2. 通常情况下，对于正式的需要长期维护与持续改进的软件项目，进行单元测试是必需的。
3. 即使一个人写的程序，进行单元测试也是有益的，编写这些测试代码的时间，将会在Debug、功能扩充、算法替换等后期阶段成倍地节省回来。

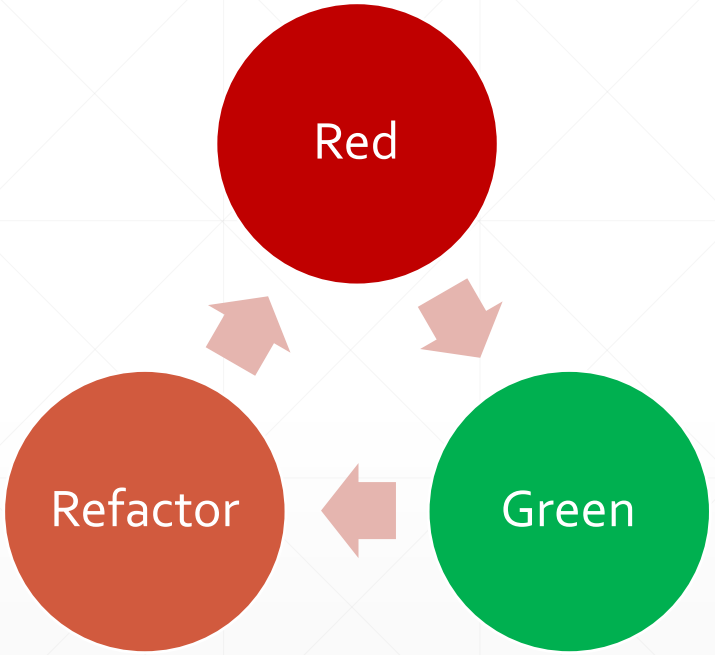
测试驱动的开发方式

- TDD：Test Driven Development，测试驱动的开发
- TDD主要通过“单元测试（Unit Test）”进行

在编写功能代码之前，先编写测试代码，之后再编写功能代码，让其通过测试

所以，是“测试（Test）驱动（Drive）开发（Development）”。

TDD的具体步骤



TDD的过程通常被划分为以下阶段：

1. **Red**：创建一个“注定要失败”的单元测试
2. **Green**：写“必要”的功能代码让测试通过
3. **Refactor**：重构测试代码和功能代码，完成所需的功能
4. **Repeat**：重复上述1、2、3三步

关于TDD的对话-1

- **为什么我要先写那些“注定”失败的测试代码？**

  先写功能代码，会诱使你添加太多的功能，而这些功能可能你会“忘记”测试它们。

- **采用TDD，到底有何好处？**

  它强迫你去想：我怎样去调用这个方法？这样一来，有助于你仔细思索你的方案，设计出好的方法接口。

关于TDD的对话-2

- **测试代码难道不是应该由测试人员负责的吗？**

  所有程序员都应该写单元测试，先通过自己的测试，再由测试人员测试，这样才能保证有足够高的开发效率。

- **采用TDD，是否我应该写完所有测试代码才开始写功能代码？**

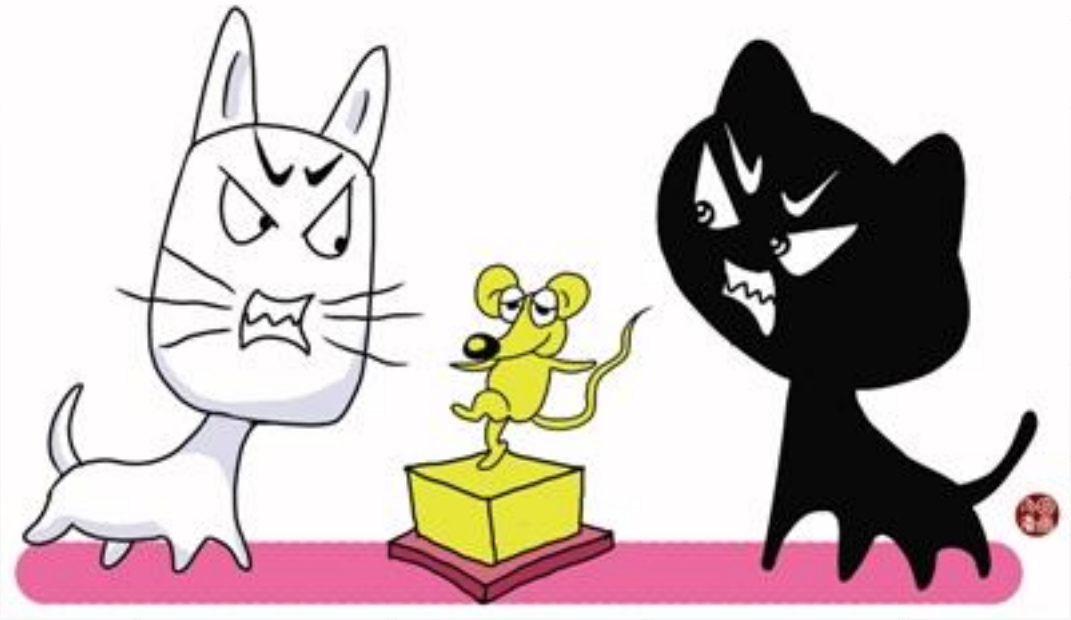
  TDD的步伐是非常小的，写测试代码，看着它失败，接着编写功能代码让测试代码工作，……，整个过程通常在几分钟内结束。

关于TDD的对话-3

- 是否我一定要采用TDD？

有些场景，是不能用TDD的。可以Test First，也可以Test Last.

是你写代码，照你的直觉和想法去做。



不管白猫黑猫，
能逮着老鼠就是好猫！