

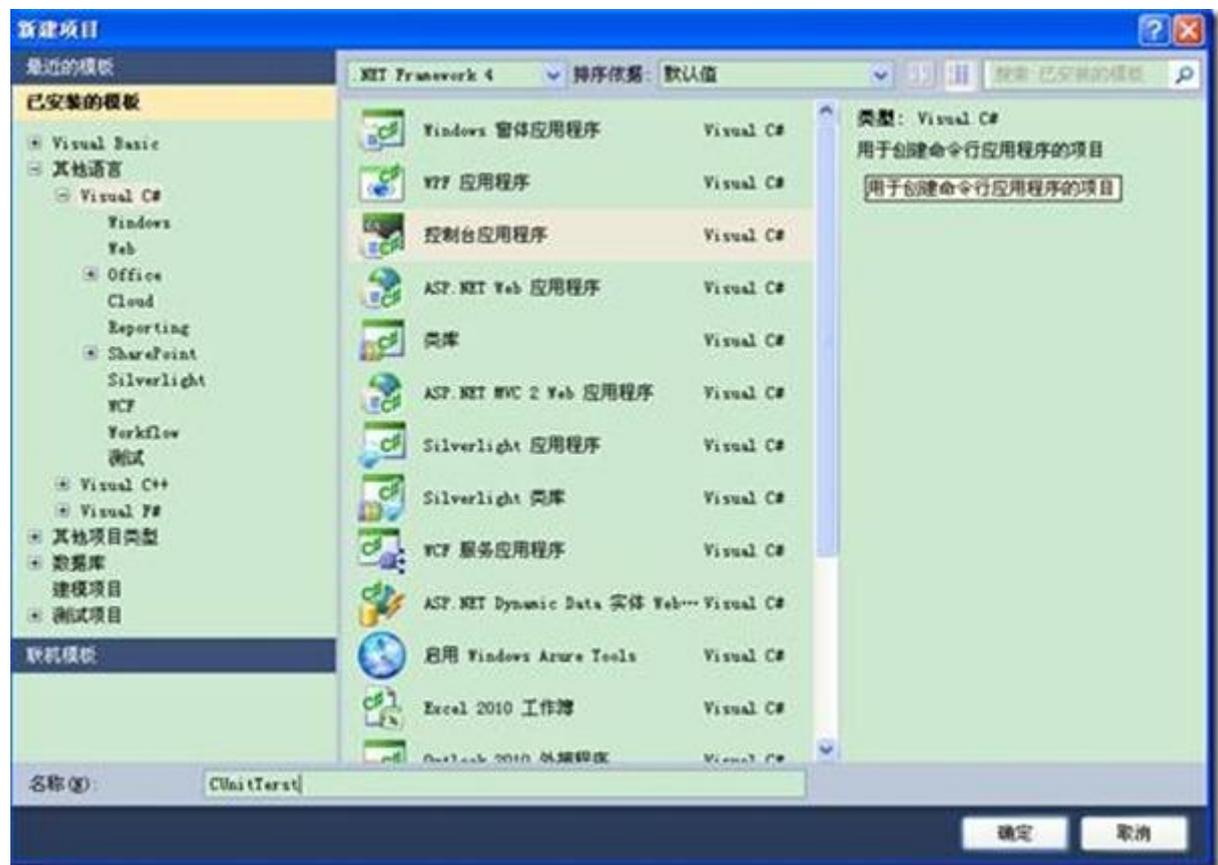
C#_在 VS2010 下进行单元测试

在 VS2010 中，单元测试的功能很强大，使得建立单元测试和编写单元测试代码，以及管理和运行单元测试都变得简单起来，通过私有访问器可以对私有方法也能进行单元测试，并且支持数据驱动的单元测试。

1、建立单元测试项目

1.1、从被测试代码生成单元测试

1) 实例：创建 VC#模式下的控制台应用程序，工程名为 CUnitTest

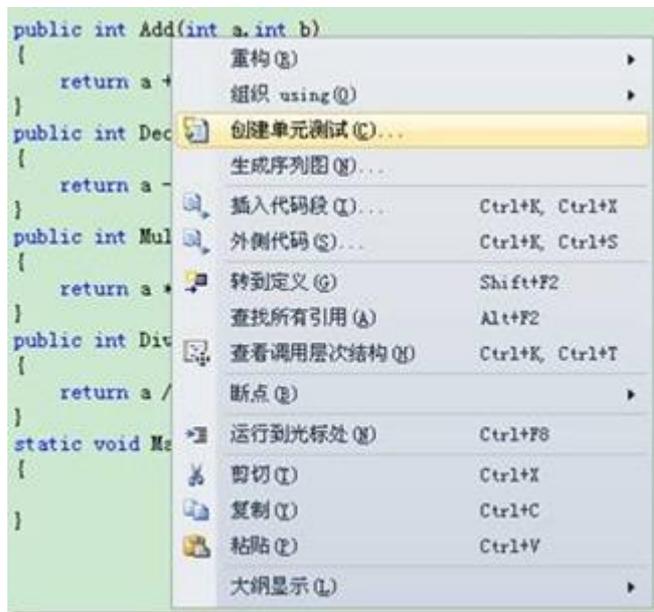


2) 输入简单的加、减、乘、除函数代码，如下图所示

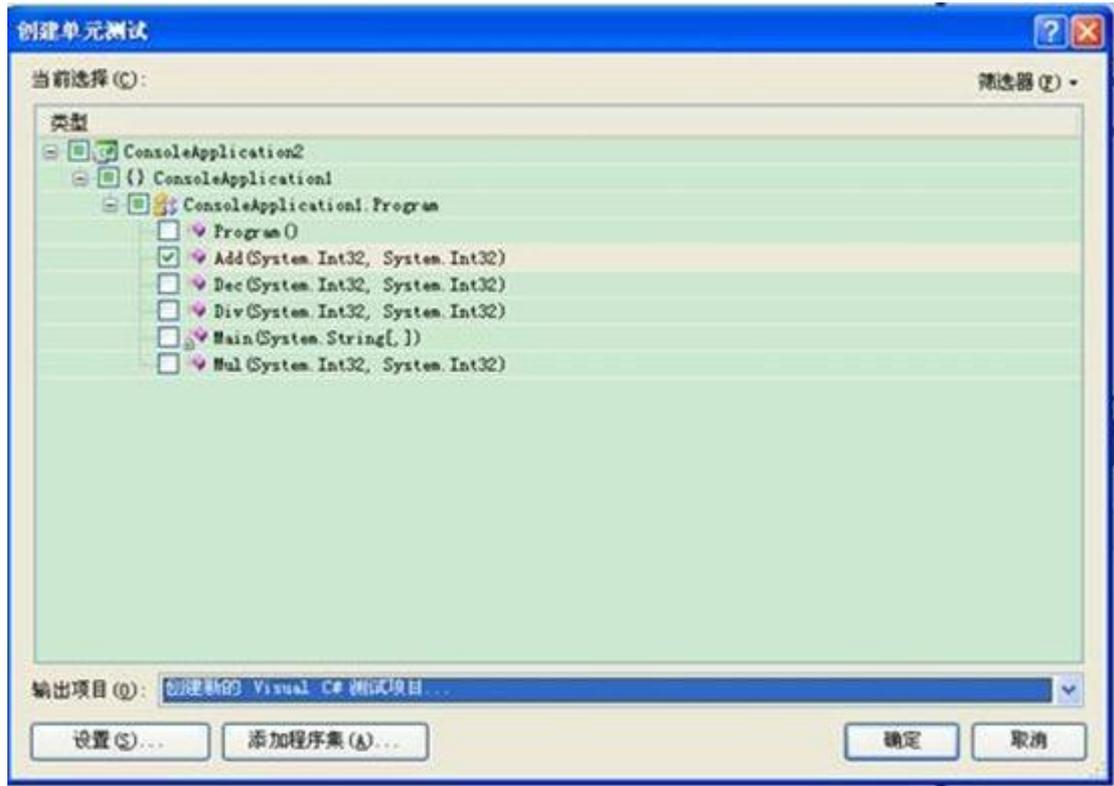
```
namespace ConsoleApplication1
{
    class Program
    {
        public int Add(int a, int b)
        {
            return a + b;
        }
        public int Dec(int a, int b)
        {
            return a - b;
        }
        public int Mul(int a, int b)
        {
            return a * b;
        }
        public int Div(int a, int b)
        {
            return a / b;
        }
        static void Main(string[] args)
        {
        }
    }
}
```

3) 可按如下步骤建立单元测试

(1) 在 Add 方法体内，单击鼠标右键，在菜单中选择"创建单元测试"，



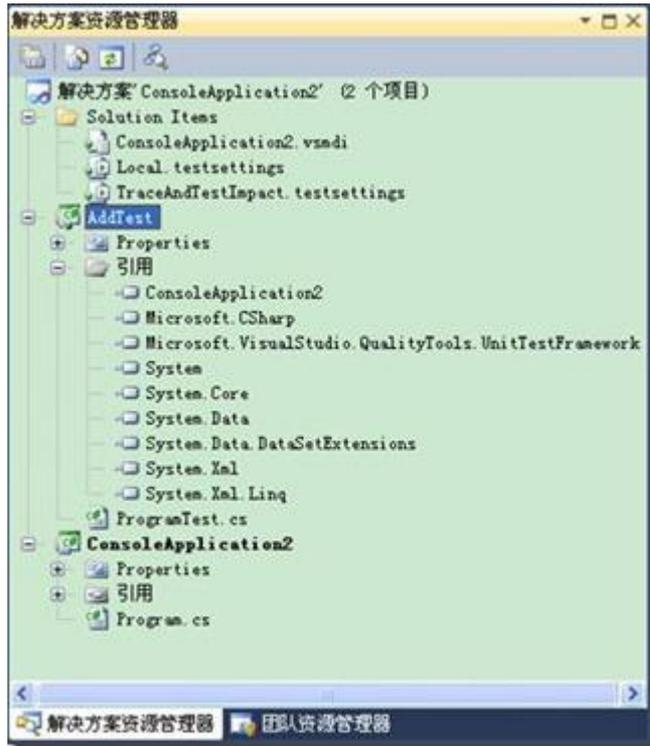
(2) 在出现的"创建单元测试"界面中，Add 方法被自动勾选，表示要为这个方法创建单元测试代码的基本框架，单击确定按钮



(3) 点击确定后，在新建测试项目中，输入需要创建的单元测试的新项目的名称，然后单击"创建"按钮，则自动创建一个新的单元测试代码项目。



(4) 在"解决方案资源管理器"中可以看到多了一个"AddTest"项目，可以看出"AddTest"项目引用了被测项目的程序集，和单元测试框架 Microsoft.VisualStudio.QualityTools.UnitTestFrame，并且自动产生两个 C#代码文件 AssemblyInfo.cs 和 ProgramTest.cs



(5) ProgramTest.cs 的代码如下图所示，从图中可以看到，自动产生了一个"ProgramTest"类，并使用[TestClass()]标识为一个单元测试类，以及一个"AddTest"测试方法，并用[TestMethod()]标识。

```
using ConsoleApplication1;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
namespace AddTest
{
    /// <summary>
    /// 这是 ProgramTest 的测试类, 旨在
    /// 包含所有 ProgramTest 单元测试
    /// </summary>
    [TestClass()]
    public class ProgramTest
    {
        private TestContext testContextInstance;
        /// <summary>
        /// 获取或设置测试上下文, 上下文提供
        /// 有关当前测试运行及其功能的信息。
        /// </summary>
        public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }
        附加测试特性
        /// <summary>
        /// Add 的测试
        /// </summary>
        [TestMethod()]
        public void AddTest()
        {
            Program target = new Program(); // TODO: 初始化为适当的值
            int a = 0; // TODO: 初始化为适当的值
            int b = 0; // TODO: 初始化为适当的值
            int expected = 0; // TODO: 初始化为适当的值
            int actual;
            actual = target.Add(a, b);
            Assert.AreEqual(expected, actual);
            Assert.Inconclusive("验证此测试方法的正确性。");
        }
    }
}
```

(6) ProgramTest.cs 代码文件详讲

[TestMethod()]: 说明了以下代码是一个测试用例

```
int a = 0; // TODO: 初始化为适当的值
```

```
int b = 0; // TODO: 初始化为适当的值
```

这两句是被测函数的输入参数, 需要我们去修改它的值, 也就是我们输入测试用例的地方。

```
double expected = 0; // TODO: 初始化为适当的值
```

```
double actual;
```

这两句话浅显易懂, 前一句话是定义了期望值和对它进行初始化, 后一句话是定义了实际值。

默认

```
Assert.AreEqual(expected, actual);
```

Assert 在这里可以理解成断言：在 VSTS 里做单元测试是基于断言的测试。

默认代码中 Assert.Inconclusive 表明这是一个未经验证的单元测试。在实际的程序中可以注释掉。

1.2、添加单元测试项目

(1) 另外一种单元测试方法是独立添加单元测试项目，在解决方案中添加一个新的项目，选择项目类型为"测试项目"，



(2) 单击确定后，自动产生一个新的单元测试项目，在"解决方案资源管理器"中可看到新添加的测试项目"TestProject2"。对比"TestProject2"和"AddTest"可发现，"TestProject2"少了对被测项目程序集的引用，仅仅引用了单元测试框架的 DLL "Microsoft.VisualStudio.QualityTools.UnitTestFrame"

2、编写测试方法

单元测试的基本方法是调用被测代码的函数，输入函数的参数值，获取返回结果，然后与预期测试结果进行比较，如果相等则认为测试通过，否则认为测试不通过。

1、Assert 类的使用

`Assert.Inconclusive()` 表示一个未验证的测试；

`Assert.AreEqual()` 测试指定的值是否相等，如果相等，则测试通过；

`AreSame()` 用于验证指定的两个对象变量是指向相同的对象，否则认为是错误

`AreNotSame()` 用于验证指定的两个对象变量是指向不同的对象，否则认为是错误

`Assert.IsTrue()` 测试指定的条件是否为 `True`，如果为 `True`，则测试通过；

`Assert.IsFalse()` 测试指定的条件是否为 `False`，如果为 `False`，则测试通过；

`Assert.IsNull()` 测试指定的对象是否为空引用，如果为空，则测试通过；

`Assert.IsNotNull()` 测试指定的对象是否为非空，如果不为空，则测试通过；

2、CollectionAssert 类的使用

用于验证对象集合是否满足条件

StringAssert 类的使用

用于比较字符串。

`StringAssert.Contains`

`StringAssert.Matches`

`StringAssert.StartsWith`

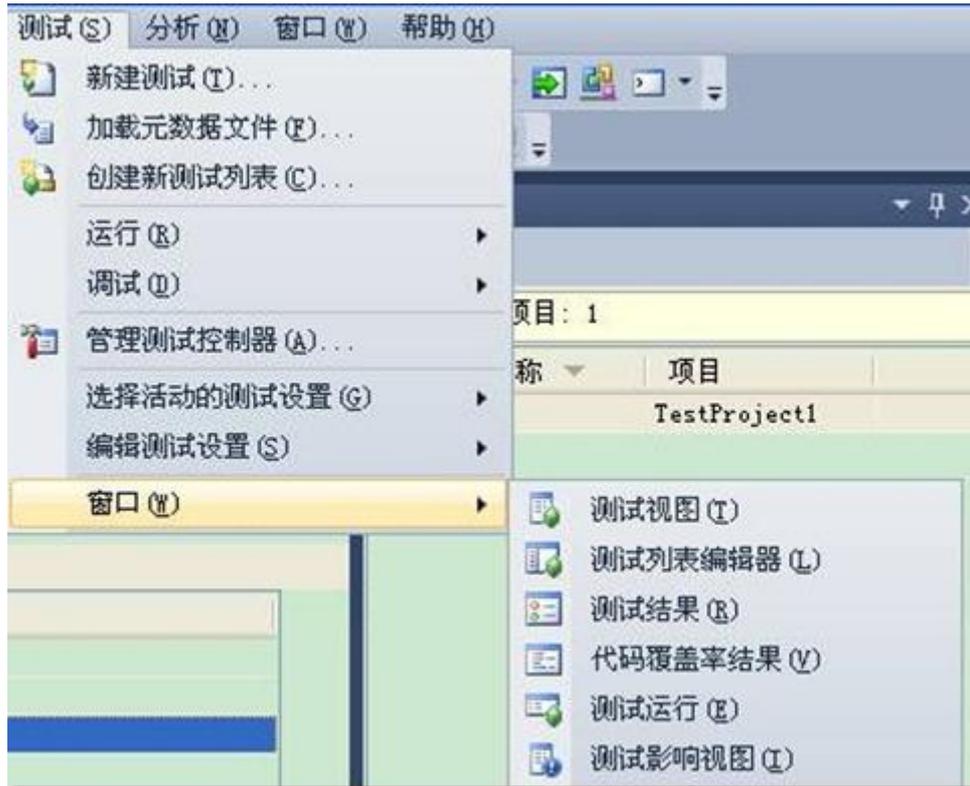
3、数据驱动的单元测试

数据驱动的单元测试是指单元测试的输入数据遍历一个数据源的所有行。从数据源的每一行

读入数据并传入给测试方法使用

3.1、ACCESS 数据驱动单元测试

1) 打开测试视图窗口，选择测试视图



2) 在测试视图窗口中选择需要配置成数据驱动方式的单元测试方法，然后按 F4，打开单元测试的属性窗口



3) 编辑"数据连接字符串"属性, 在"属性"窗口中单击该属性, 然后单击省略号 (...). 这将打开"选择数据源"对话框, 其中列出了若干个可能的数据源, 包括 ODBC、Microsoft SQL Server 和 Microsoft Access. 选择一个数据源后将打开一个特定于该数据源类型的对话框; 可以使用此对话框配置该数据源的连接属性。配置完数据连接后, 连接字符串会作为"数据连接字符串"的值出现。此字符串还会作为单元测试方法的一个属性存储起来



4) 在这个界面中, 选择一个 Access 表 data.mdb, 单击"确定"按钮完成设置, 回到"单元测试属性"窗口。可以看到数据源的已经设置好。



5) 在建立与数据源的连接之后，可以选择一个数据表。当您单击"属性"窗口的值列中的下拉列表时，将会列出所连接的数据库中的表。从此列表中选择的就是在运行单元测试时将检索其中的行的表。与"数据连接字符串"等其他属性一样，"数据表名称"也会作为单元测试方法的一个属性存储起来。

6) 在"数据访问方法"，请选择"顺序"或"随机"；默认值为"顺序"。此设置表示从数据源的表中检索记录的顺序。

可以看到，在测试方法前面已经添加了一行：

```
[DataSource("System.Data.OleDb", "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=|DataDirectory|\\data.mdb",
|", DataAccessMethod.Sequential), DeploymentItem("TestProject1\\data.mdb"), TestMethod()]
```

7) 数据源的使用

通过 TestContext 类的 DataRow 和 DataConnection 属性将数据提供给正在运行的单元测试。下面为使用 TestContext 类的 DataRow 属性来读入数据行

```

/// <summary>
/// Add 的测试
/// </summary>
[DataSource("System.Data.OleDb", "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=[DataDirectory]\\data.mdb",
    "", Data class System.String keyment Item("TestProject1\\data.mdb"), TestMethod())]
public void 表示文本,即一系列 Unicode 字符。
{
    Program target = new Program(); // TODO: 初始化为适当的值
    int a = Int32.Parse(TestContext.DataRow["参数1"].ToString()); // TODO: 初始化为适当的值
    int b = Int32.Parse(TestContext.DataRow["参数2"].ToString()); // TODO: 初始化为适当的值
    int expected = Int32.Parse(TestContext.DataRow["结果"].ToString()); // TODO: 初始化为适当的值
    int actual;
    actual = target.Add(a, b);
    Assert.AreEqual(expected, actual);
    // Assert.Inconclusive("验证此测试方法的正确性。");
}
    
```

8) Access 数据源中的表为

参数1	参数2	结果
1	2	3
2	3	4
3	1	4
0	0	0

记录: 4 共有记录数: 4

3.2、读取 Excel 的方法:

- 1) 在桌面新建一个 txt 文件, 更改文件名为 data.dsn
- 2) 选中"数据库连接字符串", 单击右边列的按钮, 更改数据源为 Microsoft ODBC 数据源, 点击"确定"按钮



3) 选择使用连接字符串，点击生成



4) 选择 Excel 数据源的驱动程序，点击"下一步"



5) 选择 data.dsn 为数据源保存文件，一直选择"下一步"。



6) 在弹出的选择工作簿中，选择用例的输入文件 data.txt,点击"确定"



7) 选择用例所在的 Sheet 页，选择"完成"



8) 数据源的使用代码

```

public void AddTest()
{
    Program target = new Program(); // TODO: 初始化为适当的值
    int a = Int32.Parse(TestContext.DataRow.ItemArray[0].ToString()); // TODO: 初始化为适当的值
    int b = Int32.Parse(TestContext.DataRow.ItemArray[1].ToString()); // TODO: 初始化为适当的值
    int expected = Int32.Parse(TestContext.DataRow.ItemArray[2].ToString()); // TODO: 初始化为适当的值
    int actual;
    actual = target.Add(a, b);
    Assert.AreEqual(expected, actual);
    //Assert.Inconclusive("验证此测试方法的正确性。");
}
    
```

4、单元测试的运行

单元测试的运行有两种方式：调试和运行。可以像调试普通代码一样对单元测试代码进行调试，当然也可以直接运行，单元测试的结果将在"测试结果"界面中展示，双击测试结果，可以得到测试结果的详细信息。单元测试的代码覆盖率可以在"代码覆盖率结果"界面中展示。



5、附加测试属性

"附加测试属性"。默认都是被注释掉的，只要我们取消注释就可以使用了。这个功能的加入，很大程度上是为了增加测试的灵活性。具体的属性有：

[ClassInitialize()]在运行类的第一个测试前先运行代码

[ClassCleanup()]在运行完类中的所有测试后再运行代码

[TestInitialize()]在运行每个测试前先运行代码

[TestCleanup()]在运行完每个测试后运行代码

如在执行测试时，将测试执行时间输入到日志中，代码如下

```
[ClassInitialize()]
public static void MyClassInitialize(TestContext testContext)
{
    StreamWriter sw = new StreamWriter(@"D:\TestLog.txt");
    sw.Write("Test Time, ");
    sw.WriteLine(DateTime.Now);
    sw.Flush();
    sw.Close();
}
```