Nunit 做 C#的单元测试

作者：林东峰
MSN：dongfengl@hotmail.com
QQ：430369
E-mail：lindongfeng@21cn.com
日期：2006 年 2 月 28 日

前言

大部分书籍介绍的内容，看了一百页其中有用的也就十多页的内容，本着取其精华的原则，写了下面的学习心得，同时在学习心得中加入了部分自己对单元测试的一些看法。可能看法过于片面，对本文有不同看法的朋友们，可以联系我（晚上一般 MSN 会在线），我们一起做一下讨论。

1. 测试组如何做单元测试

单元测试是针对最小的可测试软件元素（单元）的，它所测试的内容包括单元的内部结构（如逻辑和数据流）以及单元的功能和可观测的行为。使用白盒测试方法测试单元的内部结构，使用黑盒测试方法测试单元的功能和可观测的行为。

由于开发方式的不同，单元测试一般划分方法如下：

1. 面向对象的软件开发：以Class(类)作为测试的最小单元。以方法的内部结构作为测试的重点。
2. 结构化的软件开发：以模块（函数、过程）作为测试的最小单元。

对于中国目前的情况，开发人员对于单元测试（包括测试）并不是很认同的情况下，大部分公司会把这部分工作推给测试组，那么测试组进行单元测试的时候，又存在着什么问题呢？测试组做单元测试的时候又有那些必要条件呢？

1.代码的可读性

开发人员在写代码的时候都有自己的习惯，比如在写一个新的类的时候，类的名称有的用英语，有的用汉语拼音。而对于注释而言，有的存在部分，有的从头到尾根本没有注释（这样的代码让开发人员过半年自己再看，他自己也可能不知道这段代码实现的是什么功能）那么是否有一段可用代码编写规范在执行，开发人员是否都按规范来进行开发，一段开发人员自己都看不明白的代码，测试组的成员又怎么能看明白？可见规范的代码编写是进行单元测试必不可少的一部分。

2.设计单元测试

在进行单元测试之前需要弄清楚被测试代码实现的功能和相应的逻辑关系，同时还要考虑到测试的输入内容。也就是说，测试用例的设计。用例的设计要保证面面俱到，是否覆盖了每一条路径。而如何做到面面俱到这就需要测试组的成员们对每个函数进行详细的分析，将分析和讨论的结果归入相关的测试库中。初期工作的进度慢并不要紧，只要能做得很详细对于以后的测试还是有很大的帮助。或许以后的测试中，只要直接调用原来写过的测试类库，修改部分简单的语句就可以实现新模块的单元测试了。

3.对开发和测试语言的理解

测试人员必竟没有进行系统的开发培训，而对于单元测试来说，其中相当的一部分工作就是写代码，把讨论和分析的测试方法用测试软件的语言来进行编码的实现。刚开始这部分工作的时候，对于测试人员来说可能是比较困难的。补吧，练吧，做多了，自然就会写了。对于单元测试中的白盒测试又存在着分支的选择，这里也就涉及到了独立路径的选择。那么是否每一条路径都走到了，就要看测试人员对代码的理解程度了。而黑盒测试对于测试组成员来说可能就相对的简单些了，测试出模块是否按设计的要求实现了相关功能就可以了。

而本文在进行单元测试的时候，介绍NUnit和VS2005beta2本身所集成的测试功能（在集成中，并不只有单元测试这一部分）。

2. Nunit 的安装

Nunit 是一款开源的 C#单元测试工具，下载解压之后，一般会有如下三个文件

NUnit-2[1].2.7-net-1.1.msi	1.1 框架的安装程序
NUnit-2[1].2.7-net-2.0.msi	2.0 框架的安装程序
TestDriven.NET-2.0.1438d.exe	可以直接集成到.net 开发环境中

根据自己安装的.NET 框架进行 Nunit 的安装就可以，对于本机框架的版本，可以启动 Microsoft Visual Studio .NET 2003 之后，到“帮助”中的“关于”中查看。
在装完 Nunit 之后，再把 TestDriven.NET-2.0.1438d.exe 安装一下。
安装的过程只要认识几个英文的就可以操作下去，就不多做介绍。

2.1 第一个测试

现在我们启动 vs2003，建立一个“控制台程序”程序，让程序从一组数列中得到最大数，输入下面的代码（阴影部分）

```
using System;
namespace ConsoleApplication2
{
    /// <summary>
    /// Class1 的摘要说明。
    /// </summary>
    ///
    public class cmp
    {
        static public int Largest(int[] list)
        {
            int index, max=Int32.MaxValue;
            for (index = 0; index < list.Length-1; index++)
            {
                if (list[index] > max)
                {
                    max = list[index];
                }
            }
            return max;
        }
    class Class1
        {
            /// <summary>
            /// 应用程序的主入口点。
            /// </summary>
            [STAThread]
            static void Main(string[] args)
```

```
            {
                //
                // TODO: 在此处添加代码以启动应用程序
                //
            }
        }
    }
}
```

之后选择“项目（Project）”—“添加引用（Add Reference…）”弹出窗口，在.net 标签页下面选择 nunit.framework。如图 2-1 所示，之后点击选择，再点击确定。这样我们就把

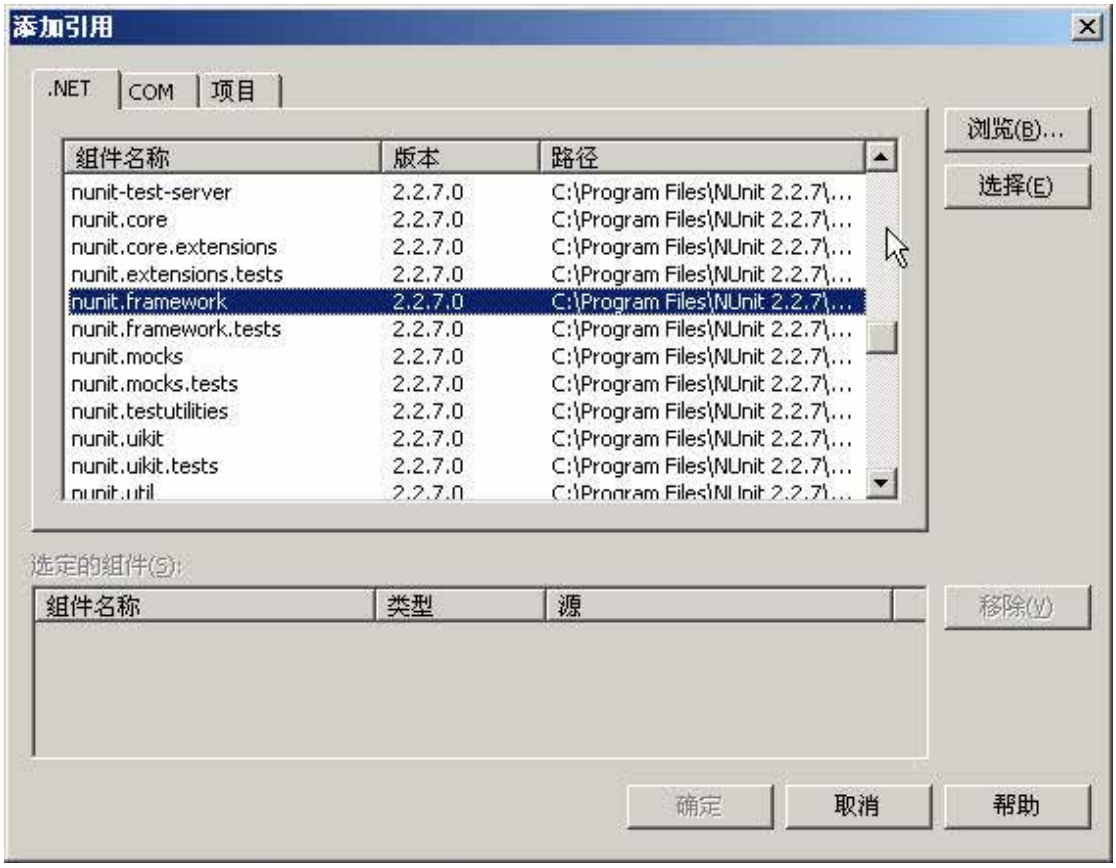

图 2-1 选择 nunit.framework

Nunit 的框架加到 vs 中了，现在我们在原来的程序上增加如下代码

```
using System;
using NUnit.Framework;          //调用Nunit框架
namespace ConsoleApplication2
{
    /// <summary>
    /// Class1 的摘要说明。
    /// </summary>
    ///
    public class cmp
    {
        [TestFixture]
```

```
        public class TestLargest
    {
        [Test]
        public void Largestof3()
        {
            Assert.AreEqual(9,cmp.Largest(new int[] {8,9,7}));
            }
    }
    static public int Largest(int[] list)
    {
        int index, max=Int32.MaxValue;
        for (index = 0; index < list.Length-1; index++)
        {
            if (list[index] > max)
            {
                max = list[index];
            }
        }
        return max;
    }
class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {

        }
    }
}
```

注意：代码中有[TestFixture]和[Test]两个属性，这两个属性为为NUnit所规定必须要添加的。

[TestFixture]表示这个类包含了测试代码（这个特性可以被继承）。这个类必须是公有的，但他的父类并不受限制。这个类还必须有一个默认构造函数。

[Test]表示它是一个测试方法。测试方法的返回值必须为void并且不能带有参数。在我们的测试方法中，执行了被测试的方法并检查了对象的状态。Assert类定义了一组方法用于检查给定的条件，例子中使用了AreEqual()方法（这个方法有很多重载）。

将文件进行编译，得到exe文件。启动NUnit-Gui，启动之后如图2-2所示，

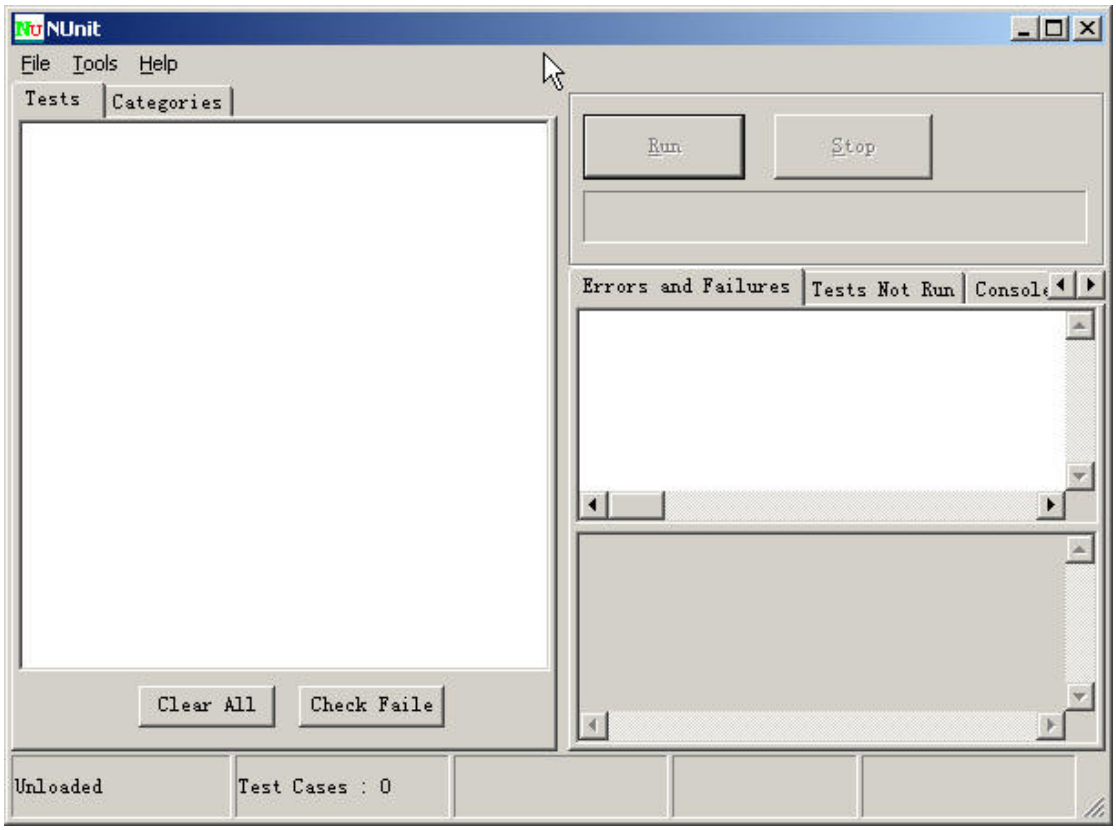

图 2-2 Nunit启动界面

我们选择File-open打开我们刚才编译完的ConsoleApplication2.exe文件。右侧的RUN变为可点状态，点击RUN开始执行测试。结果如图2-3所示，测试没有通过。

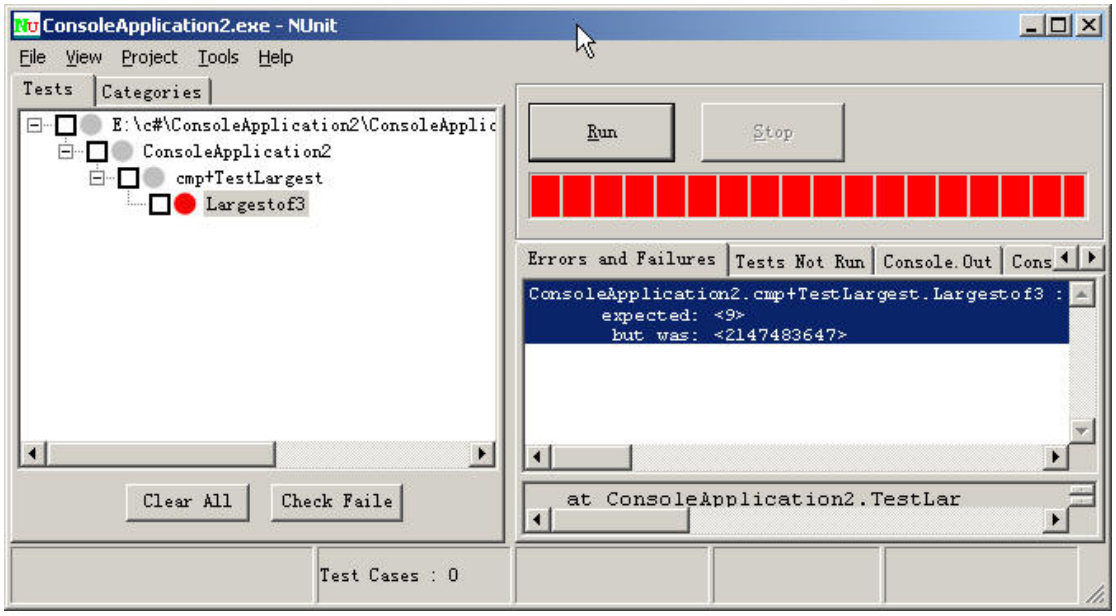

图 2-3 测试没有通过

我们可以看到，期望值是9，但实际返回值是2147483647。看看代码中

int index, max=Int32.MaxValue; 原来我们把max的值设置错了，直接指向了最大值，把代码改成

int index, max=0;再运行一下测试。重新编译一下，之后再加载到nunit中，之后，我们发现，测试可以正常通过了。如图2-4

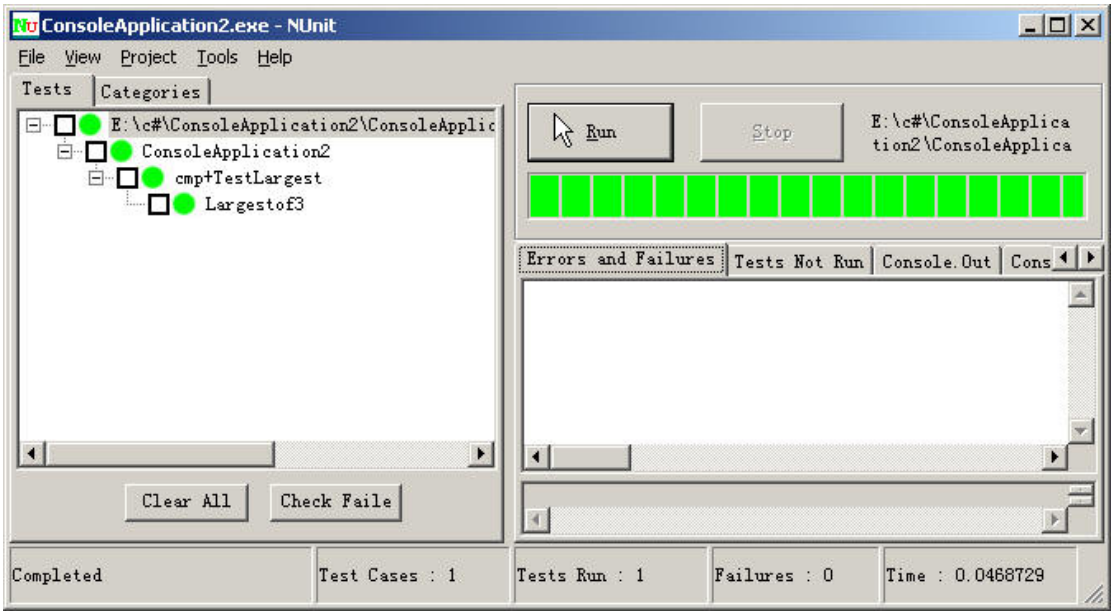

图 2-4 测试通过

2.2 用 VS2005 的测试功能重复第一个测试

现在来启动Microsoft Visual Studio 2005 Beta 2版，启动之后的界面和2003没什么太大的区别，但是注意，在菜单栏中多了一个“Test”的菜单。如图2-5所示

图 2-5 多了一个Test菜单

我们新建一个Windows Console Application 应用程序：在系统自动生成的“Program”上面，加入如下代码：

```
public class cmp
    {
        static public int Largest(int[] list)
        {
            int index, max = Int32.MaxValue;
            for (index = 0; index < list.Length-1; index++)
            {
                if (list[index] > max)
                {
                    max = list[index];
                }
            }
            return max;
        }
        }
```

编译成功后，选择“Test”-“New-Test...”弹出如图2-6所示窗口，在窗口最下面的Add To Test Project: 中我们选择“Create a new Visual C# Test Project”在“Templates”中我们选择“Unit Test Wizard”

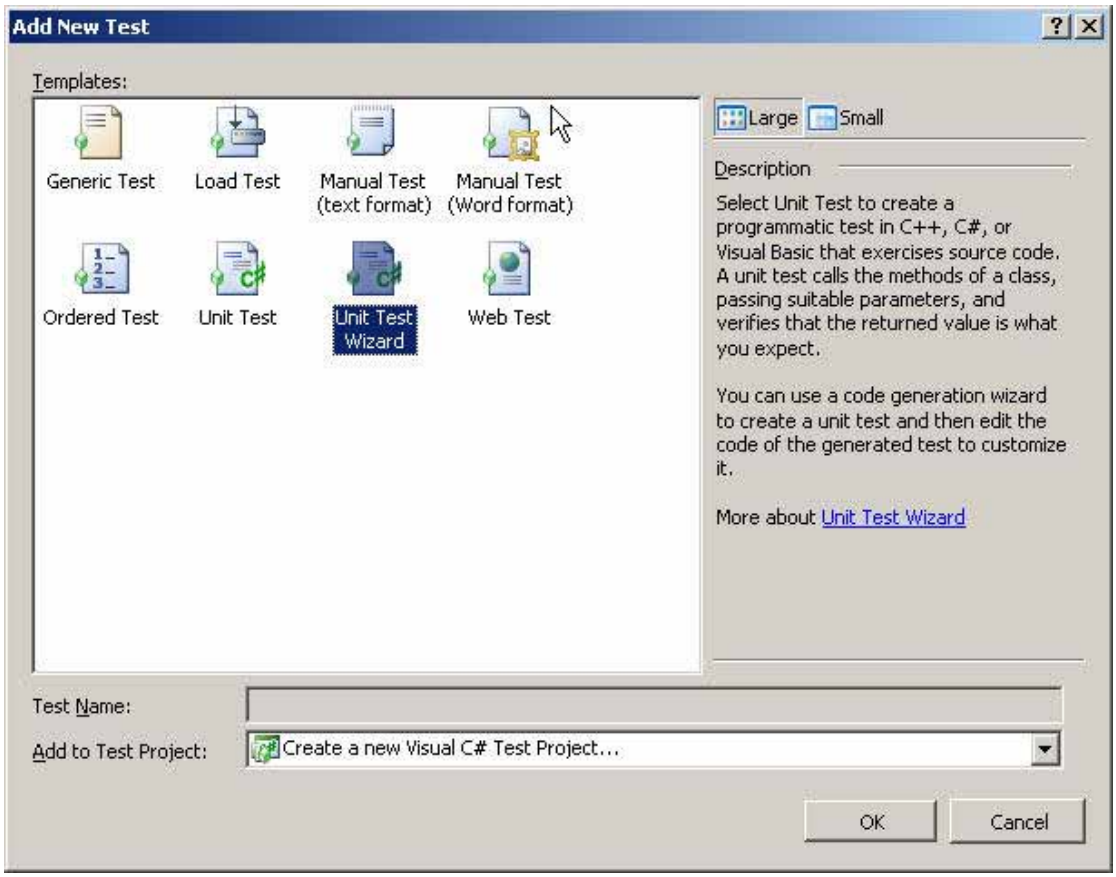

图 2-6 创建测试

点击“OK”弹出图2-7所示对话框。

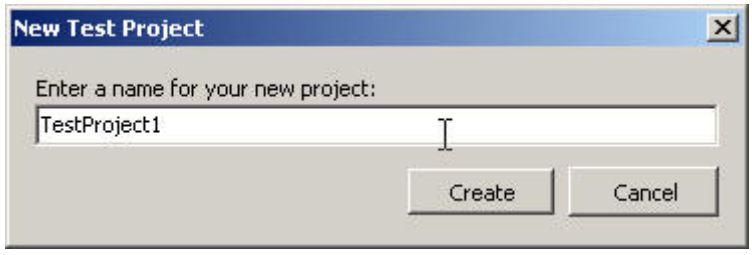

图 2-7 测试取名对话框

点击“Create”弹出如图2-8所示窗口。

我们可以看到，应用程序的类名，成员函数，都在下面显示出来，我们只选择测试的类，即CMP这个类。在窗口的左下角，有一个“Setting...”点击弹出如图2-9所示窗口。

其中：Name Settings 主要是对测试的文件名、测试的类名、测试的方法名进行设置文件名是直接在程序的文件后面加上test，类名也一样。

在下面的普通选项中，主要是设置测试的方法和过程等等，这些我们保持默认不变，点击“OK”关闭此窗口，再点击图2-8中的“OK”VS开始创建相关的测试框架。创建完成之后，我们会发现，主窗口中多了三个文件“ProgramTest.cs”、“AuthoringTests.txt”、“UnitTest1.cs”。对于“AuthoringTests.txt”文件没什么用，我们直接关闭就可以了。

下面我们看“ProgramTest.cs”文件的最后一部分代码。

```
public void LargestTest()   {
            int[] list = null; // TODO: Initialize to an appropriate value
            int expected = 0;
            int actual;
```

```
            actual = ConsoleApplication4.cmp.Largest(list);
            Assert.AreEqual(expected, actual, "ConsoleApplication4.cmp.Largest did not return
the expected value.");
            Assert.Inconclusive("Verify the correctness of this test method.");        }
```

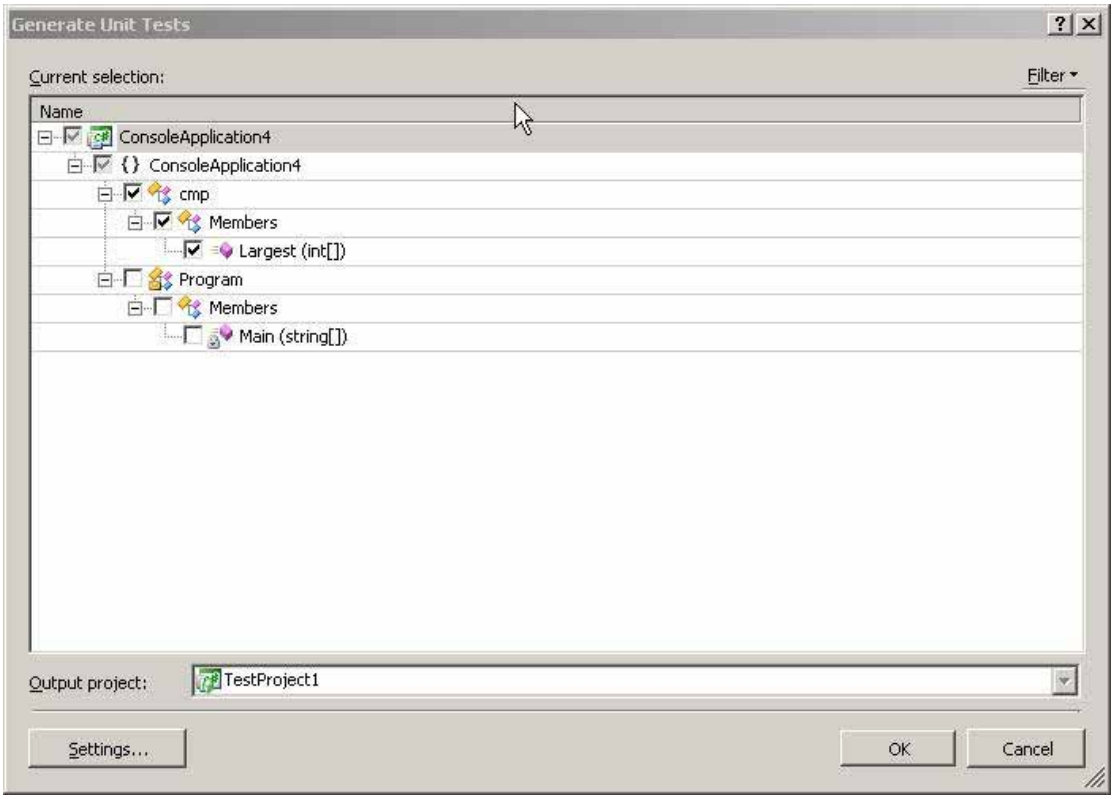

图 2-8 选择测试的类

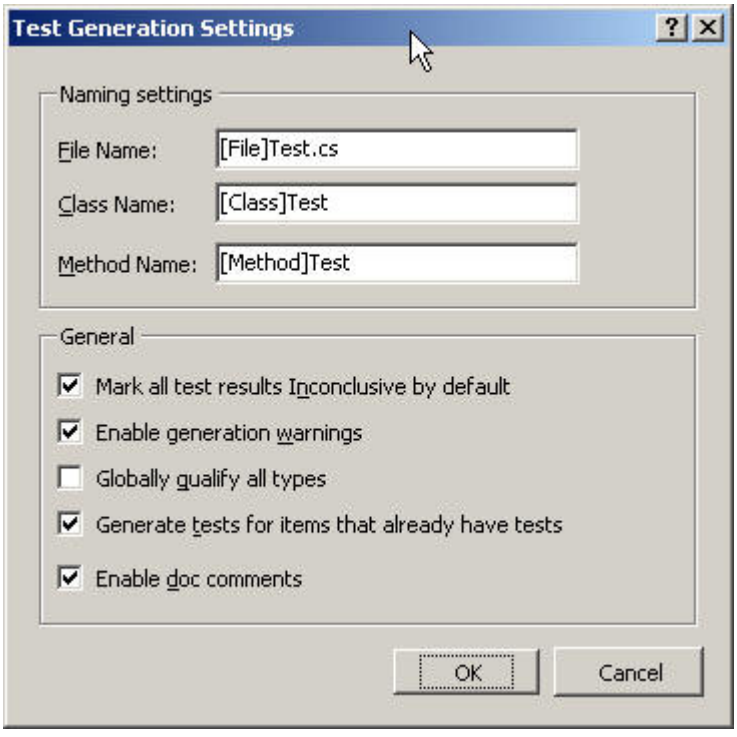

图 2-9 设置测试的基本参数

在这部分代码中，系统自己定义了list数组初始值为空，expected（期望值），actual

（实际值）实际值是从Largest(list)中得到，现在我们对这部分代码进行修改。

```
int[] list = { 9,8,7}; // TODO: Initialize to an appropriate value
int expected = 9;
```

其它保持不变，调试后无错误输出，现在点击，“Test”下的“Start Selected Test Project with Debugger”.现在开始对程序进行测试，调试结果会在下面显示出来，如图2-10所示。

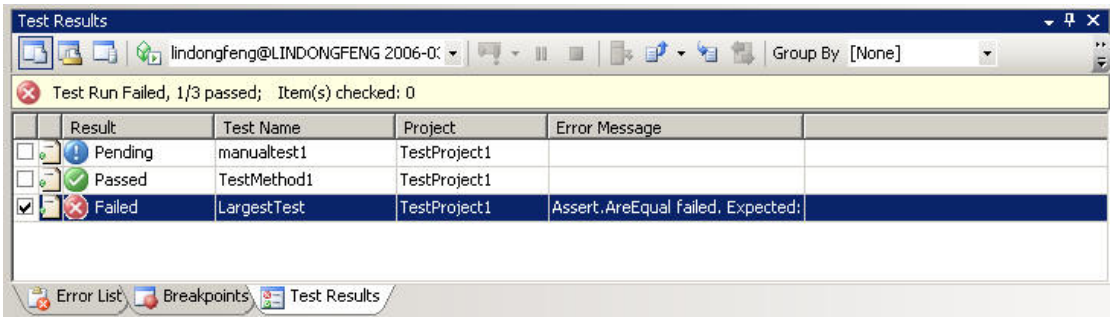

图 2-10 测试结果

双击Failed项，出现如图2-11所示界面

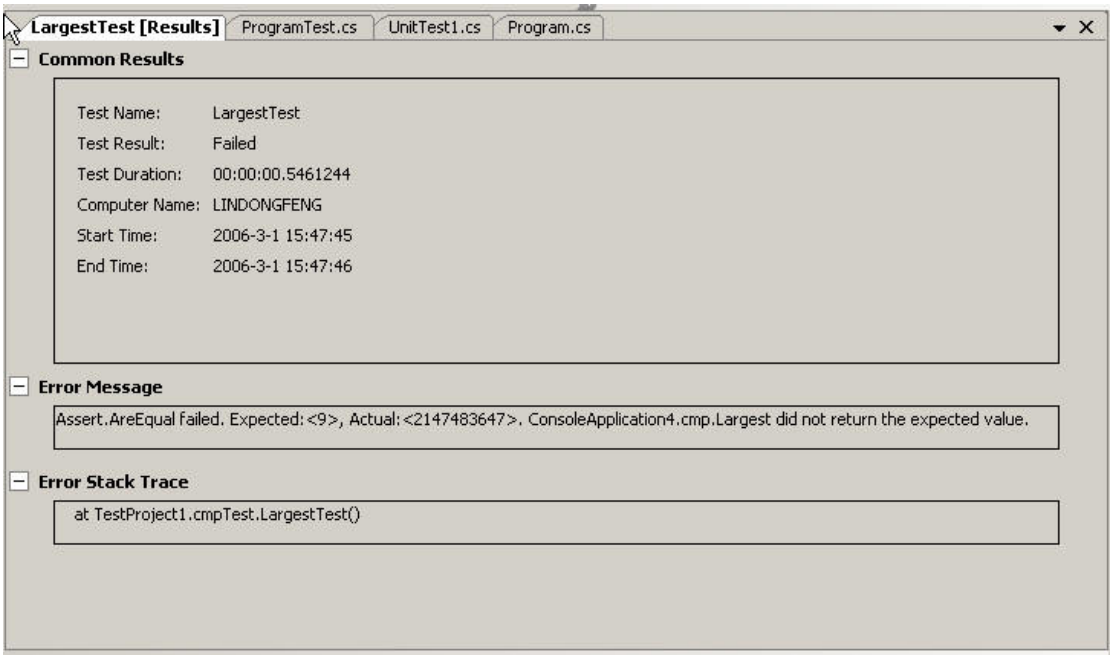

图 2-11 错误报告

这里我们可以看出来，我们期望的输出值是9，可实际值是2147483647。我们把程序中的`int index, max = Int32.MaxValue;`改为`int index, max = Int32.MinValue;`再执行一下测试。

我们看到，LargestTest类上面出现了一个？非确定性错误。并且错误提示如下

Assert.Inconclusive failed. Verify the correctness of this test method.

既然是非确定的，我们把程序的最后一行，

`Assert.Inconclusive("Verify the correctness of this test method.");`直接注释掉，再次执行测试程序，测试通过。接下来我们做负数组的测试，把`int[] list = { 9,8,7};`改为`int[] list = { -9,-8,-7};`在修改完成之后，我们也可修改我们将要得到的期望值`int expected = 9;`改为`int expected = -7;`执行测试程序，程序报错，

Assert.AreEqual failed. Expected:<-7>, Actual:<-8>. ConsoleApplication4.cmp.Largest did not return the expected value.

为什么是-8而不是-7？看看我们的循环语句，

```
for (index = 0; index < list.Length-1; index++)
```

少循环一位，把它改成`for (index = 0; index < list.Length; index++)`

再次执行测试，OK程序通过。接着分别测试以下数组

{1}、{9,8,9,7}测试全部通过，对于在数组中找最大数的输入条件全部可以通过。

下面我们来看看Assert.AreEqual它的方法

Assert.AreEqual(object,objcet)

那们上面那么长的代码我们也可以直接写成下面的形式

```
Assert.AreEqual(9, ConsoleApplication4.cmp.Largest(new int[] { 9,8,7}));
```

一句话代替，其中9是期望得到的值，9后面是从函数largest中经过比较返回的值。这样写的好处是，可以在一段代码中直接输入所有可能的输入量，例

```
Assert.AreEqual(9, ConsoleApplication4.cmp.Largest(new int[] { 9,8,7}));
Assert.AreEqual(9, ConsoleApplication4.cmp.Largest(new int[] { 9,8,9,7}));
Assert.AreEqual(-7, ConsoleApplication4.cmp.Largest(new int[] { -9,-8,-7}));
Assert.AreEqual(1, ConsoleApplication4.cmp.Largest(new int[] { 1}));
```

在测试程序运行的时候，是按顺序进行，当出现错误时，会指定具体的返回行，没有错误的地方直接跳过。

对于以上二者的相对来说，VS中的测试程序是单独写在另一个测试的工程文件夹中，而用Nunit写的程序是和开发的代码套在一起的，而且Nunit做测试的时候所写的测试代码不能调试，VS写的测试代码可以进行调试。并且大多数代码，只是用户自己输入数据就可以，需要编码的部分很少。相比之下，VS做C#的单元测试更灵活些，

一天的思想先写这么多，随时准备更新。