数据驱动测试在 Nunit 框架中的应用

王 敏[1]，陈亚光[2]
（1. 武昌理工学院 信息工程学院，湖北 武汉 430223；
2. 中南民族大学 生物医学工程学院，湖北 武汉 430074）

摘 要：为了解决单元测试工具 Nunit 本身不支持数据驱动测试的问题，提出了在 Nunit 框架下实现数据驱动测试的方法。该方法首先将测试类所使用的测试数据基本信息设定在 ini 文件中，将输入数据及预期结果存放于 Excel 文件中。随后通过属性标签[TestFixtureSetUp]标记的方法动态读取 ini 文件中的基本信息，再根据这些基本信息读取 Excel 文件中的测试数据，并将测试数据保存于自定义的结构体数组中供各测试方法使用。该方法有效地实现了测试数据与测试脚本的分离，能降低测试脚本的维护工作量，提高测试效率。

关键词：单元测试；Nunit 框架；测试脚本；测试数据
中图分类号：TP311.1 **文献标识码**：A **文章编号**：1674-7720(2012)22-0010-03

Application for data-driven test in the Nunit framework

Wang Min[1], Chen Yaguang[2]
(1. College of Information Engineering, Wuchang University of Technology, Wuhan 430223, China;
2. College of Biomedical Engineering, South-Central University for Nationalities, Wuhan 430074, China)

Abstract: In order to solve the problem that unit testing tool Nunit does not support data-driven test, we put forward a data-driven test method in the Nunit framework. In this method, the basic informations of test data are stored in an ini file, input data and expected results are stored in an Excel file. When starting unit test, some basic informations are dynamically read out from the ini file by the procedure defined in the attribute label [TestFixtureSetUp]. Next, according to these basic informations, it can read out the test data from the Excel file, and save the test data in structured array for unit test. This method can separate the test data and the test script effectively, it can reduce the workload of the test script maintenance and improve test efficiency.

Key words: unit test; Nunit framework; test script; test data

软件测试是保证软件质量的重要手段，作为软件测试基础的单元测试是软件开发过程中最低级的测试活动。据业界统计，单元测试一般可以发现大约 80%的软件缺陷[1]。由于软件缺陷发现得越早，其修复的代价就越小，因此，单元测试在软件测试过程中占据着非常重要的地位，有效地实施单元测试能有效节约后续测试时间、降低测试成本、保证软件质量。

Nunit 是微软.Net 平台下的单元测试框架，也是在.Net 平台下进行测试驱动开发的重要工具；其提供的图形用户接口 GUI 操作简单、内容直观，因此 Nunit 被广泛地运用于.Net 平台项目的单元测试中。利用 Nunit 进行单元测试，首先要进行脚本开发，目前常用的方法是将测试数据以常量的方式赋值给变量，再由变量参与测试[2-3]。采用这种方式，测试数据与测试脚本共存于一个测试脚本文件中，不利于测试脚本的维护。而测试脚本的维护是自动化测试的重要环节，适当地调整和增强测试脚本，能提高测试脚本的灵活性以及应对测试对象变更的能力，数据驱动方式的测试脚本开发是解决这类问题的重要手段[4]。由于 Nunit 框架本身并不具备数据驱动测试的功能，因此需要通过增强测试脚本的功能来实现 Nunit 框架下的数据驱动测试。为此，本文提出一种测试设计方法，将测试用例的输入数据、预期结果及执行条件编写在 Excel 工作簿中，工作簿中每一个工作表的数据对应一个测试类；工作簿存放的位置、测试类对应的工作表名、输入数据数目及预期结果数目等信息按一定格式保存于执行路径下的 ini 文件中。在测试脚本中，

利用 Nunit 的[TestFixtureSetUp]属性标签，在该标签标记的方法中实现测试类所有测试数据的一次性读入，并将其保存于自定义的结构体数组中，供各测试方法使用，在后续测试方法的脚本编写中，只需按结构体数组下标获取测试数据及预期结果便能完成测试脚本的开发；测试执行时将按照 Excel 文件中设计的数据自动执行测试脚本。这一测试设计方法能有效地分离测试脚本和测试数据，从而降低测试脚本的维护工作量。

1 数据驱动测试的设计思路

数据驱动测试是一种数据被包含在输入测试数据文件中，并且以数据来控制自动化测试脚本执行的流程和动作的测试[5]。在 Nunit 框架中，[TestFixtureSetUp] 和[SetUp]属性标签标记的方法分别在测试类和各测试方法执行前被执行，[Test] 属性标签标记的方法按其在脚本中的先后顺序自动执行；此外，通过 Nunit 的 GUI 界面，可以指定被执行的方法，因此 Nunit 自身已经具备控制测试脚本执行流程的功能。这里主要从用数据驱动脚本的动作出发来实现数据驱动测试。

数据驱动测试使用存档的测试数据来驱动自动化测试过程，这些数据通常以简单的文本文件或 Excel 文件形式存在[5]。鉴于 Excel 文件具有以表格形式呈现、结构清晰直观的特点，本文采用 Excel 的工作表来存储测试数据。同时，为使测试脚本具有较高的灵活性，将测试数据文件的存取路径、测试数据所在的工作表名等信息存放于 ini 文件中，以便在测试环境和数据发生改变时，测试脚本可以保持不变，从而降低测试脚本的维护成本。

1.1 Excel 文件格式的设计

图 1 为 Excel 文件结构与测试类及测试方法之间的对应关系图，图中的数据是根据 Nunit 自带的一个实例 Moneybag 类所做的部分测试数据。

图 1 数据表与测试类及方法的对应关系图

图 1 中，工作簿 Moneytest.xls 中的工作表 Moneytest 用来存放测试类 Moneytest 中各测试方法所需的数据。由于各测试方法常常使用一些公共的输入数据来进行测试验证，因此将公共数据设定在第一行，从第二行数据开始每一行数据对应一个测试方法，每一个单元格存放一个基本数据（字符串类型或数值）。结构数据类型都可以看成是基本数据类型的组合，一个结构数据类型可分成多个单元格来存放。由于每一个测试方法所需的输入和预期值数目各不相同，在测试数据设计时，可按需要增减工作表列的数目来增减输入和预期值的数目（图 1 中 3~6 行的 A-H 列为空，是因为 3~6 行对应的测试方法全部采用公共输入数据）。测试数据的最后一列为测试说明，用于对测试条件等辅助信息加以说明，以供测试人员或维护人员参考，测试脚本不读取该列数据。

1.2 ini 文件的设计

使用 ini 文件是为了使这一实现方法具有较高的灵活性，最大限度地减少因测试环境和测试数据的变化带来的测试脚本的维护工作量。这里约定 ini 文件以标记的测试类命名，并且保存于执行文件所在目录下，这样可通过编程来动态读取路径。因此，除了 ini 文件名外，其他所有数据的来源都是通过测试脚本执行动态获取的，这一实现方法能达到较高程度的测试脚本与测试数据的分离。

ini 文件设计如下：

```
[FILE]
    path= "D:\NUNIT\Moneytest.xls"
    sheet= [Moneytest$]
[DATA]
    input=8
    except=4
[ROWS]
    Common=0
    BagSimpleAdd=1
    BagSubtract=2
    BagMultiply=3
    BagNegate=4
```

节[FILE]用于存放 Excel 数据文件信息，参数 path 的值表示数据文件存放的路径，参数 sheet 的值表示测试数据所在的工作表名；节[DATA]用于存放输入数据和预期值的数目，这里按该测试类中使用输入数据和预期值数目最多的方法进行设置，用于动态定义结构体内数组的维数，这样定义会损失一定的数组空间，但给测试脚本的编写带来了方便；节[ROWS]用于存放每个测试方法对应的测试数据所在的行编号（结构体数组下标），在各测试方法中通过读取 ini 文件中的行编号来取得对应的测试数据，Common=0 表示将公用数据设在数据表第一行，读入到下标为 0 的数组中。

2 数据驱动测试的实现

Nunit 采用属性标签来标记测试类和方法，其中[TestFixture] 用于标记测试类，[Test]、[SetUp]、[TestFixtureSetUp]用于标记测试方法。[SetUp]标记的方法是为了避免代码的冗余，该方法将各测试方法中的重复代码提取出来，组织成一个共用的方法，在每个[Test]标记的测试方法执行前被执行一次，与它成对使用的是[TearDown]属性，用来释放[SetUp]中初始化的变量。[TestFixtureSetUp]与[SetUp]属性类似，但此属性标记的方法用来实现整个测试类的初始化，它在整个测试类执行前执行一次，与它成对使用的是 [TestFixtureTearDown] 属性，用来释放

[TestFixtureSetUp]中初始化的变量。

在本设计中，测试数据被存储在外部数据文件中，每一个方法执行前都需要外部文件中的测试数据，如果将读取外部文件的操作分别写到各个测试方法中，这无疑会产生大量冗余代码，也会增加 I/O 方面的开销；如果将这部分处理写到[SetUp]属性方法中，可减少测试脚本中重复代码的数量，但 I/O 方面的开销与前一种方法相比，没有任何改善。故本设计将测试数据读取的代码编写在[TestFixtureSetUp]属性标记的方法中，通过自定义一个结构体，将测试数据一次性读出并存放于结构体数组中，即在整个测试类的方法执行之前执行一次测试数据的读取操作。这样，既能减少测试脚本中重复代码的数量，也能减少 I/O 方面的开销。

2.1 自定义结构体的实现

```
public struct testdata
    {
        public string[] strIn ;
        public string[] strExcept;
    }
```

上面是采用 C# 定义的结构体 testdata，其中包含两个字符串类型的动态数组，分别用来存放各测试方法的输入数据和预期结果。因为测试类最大的输入数据和预期结果数目在测试用例维护时可能发生变化，故采用定义动态数组的方式。

2.2 测试类初始化的实现

[TestFixtureSetUp] 属性标记的方法用于整个测试类的初始化，其处理流程设计如图 2 所示。

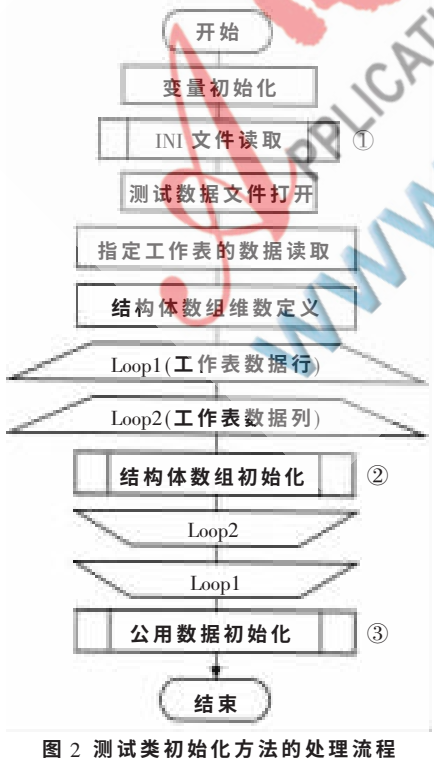

图 2 测试类初始化方法的处理流程

过程①中读取[FILE]和[DATA]节中的值，用于定位测试数据文件和定义结构体数组维数；过程②按行列依次循环，将数据表中的数据依次全部读入结构体数组；过程③初始化各测试方法的公用数据，下面是按照本文提出的方法，以 Nunit 自带的样例测试脚本 MoneyTest.cs 为例对公共输入数据进行设定的相关代码：

```
f12CHF = new Money(Convert.ToInt32(tdTest[i].strIn[0]),
tdTest[i].strIn[1]);
f7USD = new Money(Convert.ToInt32(tdTest[i].strIn[2]),
tdTest[i].strIn[3]);
f14CHF = new Money(Convert.ToInt32(tdTest[i].strIn[4]),
tdTest[i].strIn[5]);
f21USD = new Money(Convert.ToInt32(tdTest[i].strIn[6]),
tdTest[i].strIn[7]);
fMB1 = new MoneyBag(f12CHF, f7USD);
fMB2 = new MoneyBag(f14CHF, f21USD);
```

如果约定下标为 0 的结构体数组中存放公共测试数据，则在上面的代码前可直接将 i 赋值为 0，也可读取 ini 文件的 Common 值来设定 i 值。

2.3 自定义方法的实现

每一个方法执行之前都需要读取结构体数组中对应行的数据，虽然读取数据的代码可以写在各测试方法中，但这无疑会产生重复代码。通过分析各测试方法读取数据的相似部分，可将这部分代码写成一个公用方法，[SetUp]中定义的方法能保证各方法执行之前自动执行一次，但是该方法不能带参数。因此，这里自定义一个带参数的方法供各测试方法调用。同样以 Moneybag 类的测试为例，该测试类中，各方法的输入基本采用公共数据，各方法的预期结果各不相同，因此各方法均有一个读取预期结果的公共处理，将这部分处理用自定义方法实现，参数 i 为各方法对应的测试数据的数组下标。代码如下：

```
private void SetValue(int i)
{
mexcept1 = new Money(Convert.ToInt32(tdTest[i].strExcept
[0]), tdTest[i].strExcept[1]);
mexcept2 = new Money(Convert.ToInt32(tdTest[i].strExcept
[2]), tdTest[i].strExcept[3]);
mbexcept = new MoneyBag(mexcept1, mexcept2);
}
```

数据表中的数据全部以字符串类型读入构造体数组中，使用时，根据具体情况将字符串类型转换为对应数据类型参与测试。公共处理以外的个性数据读取可在各测试方法中实现。

2.4 各测试方法的实现

将 Nunit 自带的样例测试脚本按照本文提出的方法进行改编，下面是实现单一货币钱包加法的改编脚本（其他方法基本类似）：

```
[Test]
```

```
public void BagSimpleAdd()
{
int intI = GetPrivateProfileString ("ROWS", "BagSimpleAdd", "", tmpRow, 500, striniPath + "\\moneytest.ini");
int i = Convert.ToInt32(Convert.ToString(tmpRow));
SetValue(i);
Assert.AreEqual(mbexcept, fMB1.Add(f14CHF));
}
```

利用本文提出的方法对 Nunit 自带测试类 Monytest.cs 中的所有方法进行改编，执行结果与自带测试类一致。与原来的方法相比较，本文提出的方法实现了测试脚本与测试数据的分离，对测试用例的维护基本可通过对 Excel 数据文件和 ini 文件的维护来实现，从而降低了测试脚本维护过程中产生的维护成本。在后续的研究中，将根据本文提出的方法，进一步探索测试脚本自动生成的实现，以进一步提高测试脚本的开发效率。

参考文献

[1] 刘德宝.软件测试工程师培训教材[M].北京：科学出版社，2009.

[2] 林勤花.使用 NUnit 在.net 编程中进行单元测试[J].科技信息，2008(24)：410-411.

[3] 陆复名. NUnit.NET 项目测试点评[J].程序员，2004(11)：128-129.

[4] 陈技能.QTP 自动化测试实践[M]. 北京：电子工业出版社，2009.

[5] 刘晓丹，武君胜，刘博.基于数据驱动的自动化测试平台设计[J].科学技术与工程，2008,8(3):779-782.

(收稿日期：2012-08-19)

作者简介：

王敏，女，1968 年生，副教授，主要研究方向：软件测试技术。

陈亚光，男，1951 年生，教授，主要研究方向：计算机应用技术。