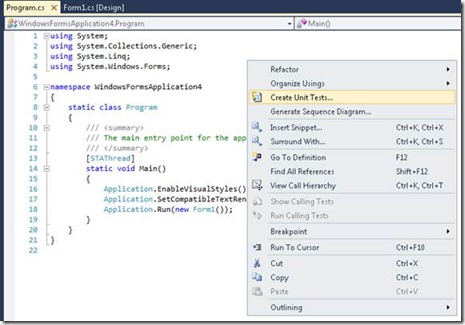
**自动化测试的实现**

      编写自动化测试也许对很多测试人员来说比较陌生。所幸的是Visual Studio中为实现自动化测试提供了一系列的工具，单元测试（Unit Test）、编码UI测试（Coded UI Test）、压力测试（Stress Test）、网页性能测试（Web Performance Test）、数据库单元测试（Database Unit Test）等等，让实现自动化测试变得轻松。这里我想着重介绍2种最基本的，也是在我们的产品开发中最常用的测试：单元测试和编码UI测试。

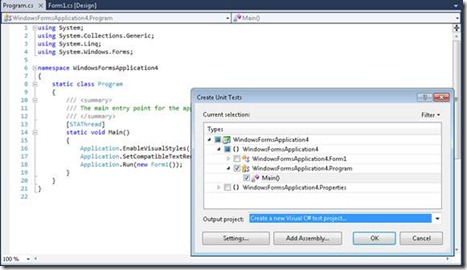
**1. 单元测试**

      单元测试是Visual Studio中最基本、应用最广泛的一种测试。通常开发人员可以选择为一个方法或是一个部件创建单元测试，来保证其逻辑正确。

      要在Visual Studio中创建单元测试，可以在源代码的上下文菜单中选择“创建单元测试”，并在弹出的窗口中选择需要为其创建单元测试的方法（如图一、图二所示）。这样Visual Studio就会自动创建出一系列单元测试的代码框架，以及针对private/internal等无法直接调用的方法的访问器（Accessor），用户只需修改或添加具体测试逻辑即可。访问器会随着源代码的每一次编译自动更新，为用户节省了不少麻烦。当然，用户也可以使用单元测试向导创建，或是直接添加一个单元测试（测试->新建测试）文件再自行添加逻辑代码。

[](http://images.cnblogs.com/cnblogs_com/stbchina/WindowsLiveWriter/VisualStudio_81A8/clip_image002_2.jpg)

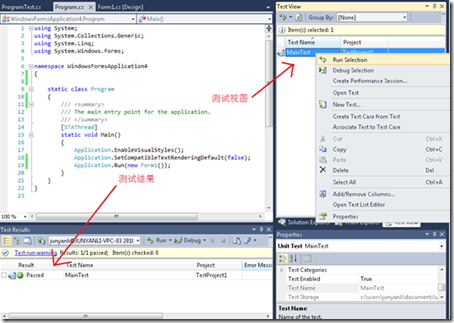
图一 创建单元测试

[](http://images.cnblogs.com/cnblogs_com/stbchina/WindowsLiveWriter/VisualStudio_81A8/clip_image004_2.jpg)

图二 创建单元测试对话框

      单元测试通常以[TestClass]属性来表示一个测试类，在测试类中使用5种不同的属性标示方法：[ClassInitialize]、[TestInitialize]、[TestMethod]、[TestCleanup]、[ClassCleanup]。一个测试类中可包含多个测试方法（Test Method），但是仅可以有一个类初始化方法（Class Initialize）、一个测试初始化方法（Test Initialize）、一个测试清理方法（Test Method）、一个类清理方法（Class Cleanup）。在测试运行时，类的初始化会被首先调用，然后在运行每一个测试方法之前运行测试初始化，之后运行测试清理，在测试方法运行结束后，类清理方法将被运行。除测试方法外，其他的辅助方法都不是必须的。大家可以根据实际需要来安排代码逻辑。

      成功编译后，所有测试方法都会在测试视图（Test View）窗口中列出，在该窗口中还可以对测试方法进行过滤、查询和排序，选择一个或多个测试方法后，可以运行或调试测试用例。测试的结果（是否通过）会显示在测试结果（Test Result）窗口中，双击任意一条测试结果都会打开具体的测试结果日志以获取更详细的信息，如图三所示。单元测试还可以通过直接在测试方法代码中右键选择“运行测试”，或是在命令行中直接执行mstest命令来运行。

[](http://images.cnblogs.com/cnblogs_com/stbchina/WindowsLiveWriter/VisualStudio_81A8/clip_image006_2.png)

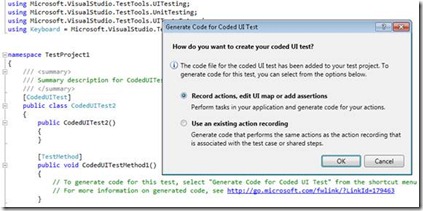
图三 测试视图和测试结果

      此外，单元测试工具不仅可以用作单元测试的目的，也可以作为一种载体，来实现验收测试或是功能测试。我们在实践中大量利用了Visual Studio对单元测试的管理、运行、日志等功能，通过在测试代码中实现验收测试、功能测试的具体逻辑来完成各种不同类型的测试。

**2. 编码UI测试**

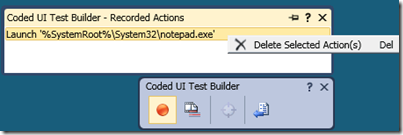
      虽然单元测试框架适用于各种不同的测试，不过其本身却没有提供太多对测试代码实现上的支持。对于自动化测试中常常令人无从下手的UI操作的自动化，Visual Studio 2010中添加了一种新的测试类型——编码UI测试，以帮助用户克服这一难题。编码UI测试是一种能轻松上手，迅速创建出UI测试的框架。

      一种最简单的创建UI测试的方法是直接从手动测试入手。如果此前我们曾在Test Manager中创建了测试用例，并曾在手动执行时录制过其测试步骤，那么我们就可以直接将录制的步骤转化为编码UI测试的代码。在Visual Studio中选择创建一个编码UI测试后，会跳出一个对话框询问用户是使用已有的操作录制还是重新录制，选择第二项“Use an existing action recording（使用现有操作录制）”后即可通过查询测试用例工作项将相应的测试转化为自动化测试代码（见图四）。

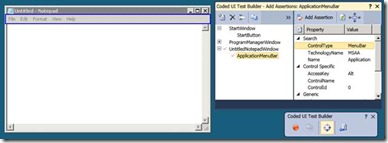
[](http://images.cnblogs.com/cnblogs_com/stbchina/WindowsLiveWriter/VisualStudio_81A8/clip_image008_2.jpg)

图四 创建编码UI测试

      如果之前没有录制过测试步骤，或是想重新创建测试的话，可以在图四对话框中选择第一项“Record actions, edit UI map or add assertions（录制操作、编辑 UI 映射或添加断言）”，这样编码UI测试生成器（Coded UI Test Builder）就会出现。在编码UI测试生成器中，用户可以自由选择为测试录制操作步骤（图五）、手动添加某些UI控件或是断言（图六），然后就可以为这些内容生成代码。这一过程可以通过在代码的上下文菜单中选择“Generate Code for Coded UI Test（为编码UI测试生成代码） ”反复执行，需要提醒用户的一点是每一次所有的代码都将被重新生成，所以手动修改生成的代码是没有意义的，除非此后不再借助编码UI测试生成器生成代码。

[](http://images.cnblogs.com/cnblogs_com/stbchina/WindowsLiveWriter/VisualStudio_81A8/clip_image009_2.png)

图五 编码UI测试生成器——录制

[](http://images.cnblogs.com/cnblogs_com/stbchina/WindowsLiveWriter/VisualStudio_81A8/clip_image011_2.jpg)

图六 编码UI测试生成器——添加UI控件和断言

      此外，用户还可以不借助Visual Studio提供的这些工具，直接利用编码UI测试提供的API（Microsoft.VisualStudio.QualityTools.CodedUITestFramework等）编写代码，实现UI自动化测试。

      编码UI测试的运行方法、运行结果等都与单元测试类似，此处不再赘述。

      这里要强调的是自动生成的自动化UI测试并不能解决UI测试固有的不稳定的问题。尤其是这种编码UI测试是通过UI控件之间的包含关系来寻找控件并对其执行操作的，就导致了如果运行测试时UI排列与录制时不尽相同时，测试可能无法正确运行。确保运行时UI环境的一致、在各操作步骤之间添加对UI控件状态的判断、在生成的代码的基础上编写自己的代码是能提高编码UI测试稳定性的一些方法。

**3. 其他类型测试**

      除了上述两种常用的测试类型之外，Visual Studio针对不同类型的测试以及测试对象，提供了各种其他的测试工具。例如，网页性能测试通过记录用户每一步操作选择的地址和发送的信息来实现网页测试的自动化；负载测试帮助用户模拟多用户各种不同测试环境下的负载；数据库单元测试提供了直接针对数据库的测试支持。这里我就不再一一详细介绍了，有兴趣的读者可以自己在MSDN上查询使用方法或者直接试用这些功能。

**自动化测试的管理**

      对于手动测试，测试用例工作项已经能很好的描述测试的内容以及记录测试的结果。而自动化测试的不同之处在于其需要代码的支持。我们通常将测试代码和产品代码一起保存在Team Foundation Server的源代码控制中，这样一方面便于代码的统一管理，另一方面让测试用例也能利用到TFS提供的版本控制、搁置集等功能。另外，我们还可以通过设置TFS的测试用例工作项中包含的“关联的自动化测试”域的值将测试计划中的测试用例和实际的代码联系起来。

**小结**

      在这一篇中，我们讨论了手动测试和自动化测试各自的优势和局限性，两者互补和平衡能帮助测试人员更好的在敏捷开发的环境中完成测试任务。此外，我们还了解了如何借助Visual Studio中提供的一些工具来实现并管理自动化测试。在介绍了自动化测试的方法和工具后，我将在下一篇中进一步为大家介绍如何计划和执行自动化的测试用例。