
.NET for Visual FoxPro® Developers



Kevin McNeish

*Edited by
Cathi Gero*

Hentzenwerke Publishing

作者: Kevin McNeish

编辑: Cathi Gero

ISBN: 1-930919-30-1

页数: 508 页

Copyright © 2002 by Kevin McNeish

出版: [Hentzenwerke Publishing](#)

翻译: xinjie

目 录

概述	10
第一章 .NET 简介	11
什么是 .NET?.....	11
什么是 .NET Framework?	11
.NET Framework 类库	12
公共语言运行库 (CLR)	13
VFP 开发者这为什么要学 .NET ?	14
市场	14
ASP.NET	15
建立中间层组件.....	15
语言互操作性	16
抽象操作系统服务	16
多线程.....	16
你的确可以学习 .NET.....	16
托管代码 (Managed Code)	17
程序集 (Assemblies)	17
清单 (Manifests)	18
私有和共享程序集 (Private and shared assemblies)	20
在全局程序集缓存中查看程序集	21
命名空间 (Namespaces)	21
.NET 程序语言.....	22
Visual C# .NET	22
Visual Basic .NET	23
命名规范	23
常规的开发步骤	24
需求调查.....	24
开发计划.....	24
设计和创建商业对象	24

建立数据模型	25
创建用户界面	25
创建 XML Web 服务	25
总结	26
第二章 Visual Studio .NET	27
起始页	27
开始 (Get Started)	28
新增内容 (What's New)	28
在线社区 (Online Community)	29
新闻提要 (Headlines)	29
在线搜索 (Search Online)	29
下载 (Downloads)	30
XML Web Services	30
虚拟主机 (Web Hosting)	30
我的配置文件 (My Profile)	31
显示起始页	31
解决方案和项目	32
创建一个新项目	32
项目模板	33
示例: 建立一个 Visual Basic Windows 应用程序	33
在解决方案资源管理器中检查新的解决方案	35
检查新项目	36
在已有解决方案中添加其他项目	37
指定启动项目	38
生成一个解决方案	39
生成配置	40
生成选项	41
生成示例解决方案	42
检查生成的输出文件	42
运行编译的程序	43

创建并安装共享程序集	43
为共享程序集创建一个强名称（strong name）	43
安装程序集到全局程序集缓存	45
动态帮助	45
属性窗口	46
属性窗口中的排序	47
显示事件	48
代码编辑	49
智能感应	50
大纲显示（Outlining）	52
书签	53
格式化文本	54
Visual Studio .NET 窗口	55
工具窗口	56
文档窗口	56
查找和替换	57
查找和替换	58
在文件中查找和替换	58
查找符号	60
设置 IDE 选项	61
对象浏览器	62
类视图窗口	63
任务列表	65
命令窗口	65
收藏夹	65
工具箱	66
服务器资源管理器	67
数据连接	68
服务器	69
源代码管理	69

宏资源管理器	70
宏 IDE	71
总结	72
第三章 C# 入门	73
弱类型 vs 强类型	73
一个简单的 C# 程序	75
C# 语法	76
大小写敏感	76
分号和多行语句	77
代码位置	77
语句的分组	77
注释	78
命名空间	79
定义一个简单的类	80
默认基类	80
定义类方法	81
定义变量	81
字段	81
字段修饰符(field modifiers)	82
值类型和引用类型(Value and reference types)	83
理解栈和堆 (stack and heap)	83
值类型和引用类型的性能	85
字符串类型 (The string type)	85
将一个字符串赋予另一个字符串	86
枚举	87
数组	88
声明数组	89
在数组中存储值	89
排序数组	89
多维数组	90

类型转换	90
隐式类型转换	91
使用 <code>casts</code> 进行显式类型转换	91
转换到字符串	92
使用 <code>Parse</code> 来转换字符串	92
装箱和取消装箱(Boxing and unboxing values)	92
<code>is</code> 运算符	93
<code>if</code> 语句	94
<code>switch</code> 语句	94
<code>for</code> 循环	95
<code>while</code> 循环	95
<code>do</code> 循环	96
<code>foreach</code> 循环	96
XML 文档化	96
不安全代码(Unsafe Code)	100
C# 是一个国际化的行业标准语言	100
总结	101
第四章 Visual Basic .NET 入门	102
针对 Visual Basic 6 的向后兼容性改进	102
弱类型 vs 强类型	103
Option Explicit 和 Option Strict	103
一个简单的 Visual Basic .NET 程序	105
Visual Basic .NET 语法	106
大小写敏感	106
行终止符和续行符	107
语句分组	107
注释	107
命名空间	108
定义一个简单的类	109
默认基类	110

定义类方法.....	110
从方法中返回值.....	111
声明变量	111
成员变量	112
成员变量修饰符.....	113
值类型和引用类型	113
字符串类型.....	113
作为引用类型的字符串	114
枚举.....	115
数组.....	116
声明数组	116
数组的数组空间重分配（Redimensioning arrays）	117
在数组中存储值.....	117
数组排序	117
多维数组	118
类型转换	118
隐式类型转换	119
显式类型转换	119
CType 函数	121
转换到字符串	121
使用 Val 转换字符串	122
Visual Basic .NET 中的装箱和取消装箱.....	122
TypeOf 操作符.....	122
If 语句.....	123
Select...Case 语句	123
For...Next 循环	124
While 循环.....	124
Do 循环.....	125
For Each 循环	125
With 语句.....	125

XML 文档化工具	126
总结	127

概述

现在已经可以在线阅读由 Kevin McNeish 所著, Cathi Gero 编辑的《.NET For Visual Developers》一书。在 Hentzenwerke 出版社、微软公司和本书作者 Kevin McNeish 的合作下, 这本书已经出版。

Visual FoxPro 是创建桌面、C/S 和 Web 应用的最好的工具之一。无论如何, 它没有被纳入 .NET 是一个错误。微软已经投入了大量的资源来使 .NET 成为创建桌面和 Internet 应用程序的革命性平台。

如果你仅仅对 .NET 的意图好奇, 这本书将针对 .NET Framework、C# 以及 Visual Basic .NET 提供一个完整的预览, 它可以让你通过 Visual FoxPro 的视角来评估这些新的技术。如果你已经转换开发语言并打算学习如何使用 .NET 开发项目的细节, 这本书将提供许多的“如何去做”、“循序渐进”以及“最优方法”的信息, 这些将帮助你减少学习 .NET 的学习曲线并快速的应用它。

第一章是后面其他章节的基础, 它回答了诸如“什么是 .NET?”、“为什么要对 .NET 感兴趣?”这样的问题。其他章节将带你学习 C# 和 Visual Basic .NET 的详细内容, 以帮助你来决定你到底要学哪个语言。在此之后的其他章节, 将循序渐进的带你来创建你的第一个 Windows 应用程序、Web Forms 应用以及 XML Web Service 。

第一章 .NET 简介

每隔六七年，微软在技术上都有一个巨大的飞跃。在 2002 年 2 月出现的是 .NET 。
什么是 .NET ？它对 Visual FoxPro 开发者意味着什么？这一章将介绍 .NET、.NET Framework 及语言，并解释为什么针对你的软件开发项目要调研 .NET 。

.NET 大张旗鼓的突然出现在软件开发领域，并伴随着来自微软的大量营销。在许多方面，.NET 和微软先前提供的技术背道而驰；在其他方面，它有着更多的演变，以至于是一场革命。

尽管存在一个崭新的 .NET 语言 C#，但是它也包括一个经过完全修改的 Visual Basic .NET，以及包括 COBOL 在内的其他许多语言的 .NET 化，Visual FoxPro 不是 .NET 语言，并且微软也不打算这么做。作为一个 Visual FoxPro 开发者，这对你意味着什么？为什么要使用 .NET 技术？什么时候使用？在哪里使用？这一章将告诉你这些问题的答案。

什么是 .NET？

最初，微软对 .NET 的定位是“一个建设、部署、运行、整和、强大的 Web 服务平台”。这样的定义使大多数人认为它只能建立 Web 应用程序。此外，因为 Web 服务流行的很慢，它导致了很多开发者的观望。你可以把这比喻为，仅仅因为 Visual FoxPro 具有建立 Web 服务的能力而大力销售它——尽管你可以使用 VFP 来建立 Web 服务，但是这不是它的主要功能。

几个月后，微软承认了它市场营销的错误，并在 2002 年 8 月由比尔·盖茨提供了一个关于 .NET 的新的定义：“.NET 是用来连接信息、人、系统和设备的软件。”我想这个主题思想可能更好些，但是我并不确信它是足够清晰或者说足够明确的。我认为更好的解释是——至少从一个开发者的观点来说——.NET 是微软提供的针对建立桌面、移动以及基于 Web 应用的新的编程模型。

什么是 .NET Framework？

当很多人提及“.NET”时，他们通常指的是 .NET Framework 。在 .NET Framework 中，实际上包含下面三种含义：

-
- 一个统一标准的核心类库（统一编程类），用于应用程序的流水作业。
 - 提供用于建立 ASP.NET 和 Windows Forms 应用的类。
 - 公共语言运行库(Common Language Runtime, 简称 CLR)，你的 .NET 程序执行的环境。

.NET Framework 类库

在 .NET 之前,Windows API 是你访问 Windows 操作系统服务的基本方式。Windows API 已经发展了好多年，要学习和使用它是非常困难，它已经成为实现功能的泥潭。相反，.NET Framework 提供了一套类，它们具有属性、事件、方法，你可以通过它们用一种更具有逻辑性更直观的方式来访问同样的服务。

例如，.NET Framework 有一个 Environment 类，你可以使用它来得到和设置应用程序运行平台和其环境的信息。下面包含在 Environment 类的属性看上去很直观，并且是自解释的：

- CurrentDirectory
- MachineName
- OSVersion
- SystemDirectory
- UserDomainName

你不需要像专家那样理解 Environment 类中下面的这些方法如何运行的：

- GetCommandLineArgs
- GetEnvironmentVariables
- GetLogicalDrives

获取同样的信息，这种方式相比于去学习使用 Windows API 是一个更快更容易的方式。

也就是说，针对 .NET 的最大学习曲线是 .NET Framework 类。这里有一个比较——Visual FoxPro 中，你可以在你的应用程序中使用不同的 35 个基类，包括文本框、组合框、表格、容器、自定义类、数据工作期等等。与此相反，.NET Framework 却拥有超过 2000 个基类！你需要领会哪些类对你是有用的，在特定的情况下使用哪些，这些是学习曲线所在。

然而，一个在 Visual FoxPro 和 .NET Framework 类之间公正的比较是这样的：在 Visual FoxPro 中，你需要记忆超过 500 个函数、大约 430 个命令以及超过 75 个系统

变量。相比而言，.NET Framework 类的结构可以使你更容易的找到你需要的功能。例如，在 Visual FoxPro 中，存在几十个用于字符串处理的命令。当你第一次学习 Visual FoxPro 时，你可以学习了这些命令中很少的部分（也许你到现在也没有完全掌握它们!）。

相反，.NET 执行所有这些字符串处理是通过 String 类的单独方法来进行。一旦你使用了 String 类，你就可以随心所欲的进行字符串处理。在 .NET Framework 类中，你会发现所有相似的不同类型的功能。

公共语言运行库（CLR）

除了一些关键性的差异外，公共语言运行库（CLR）和 Visual FoxPro 的运行库的功能相似。Visual FoxPro 是一个解释型语言；实际执行的是伪代码（pseudo-code, 简写 p-code）。当它执行时，伪代码被 Visual FoxPro 运行库解释为机器代码（图 1）。

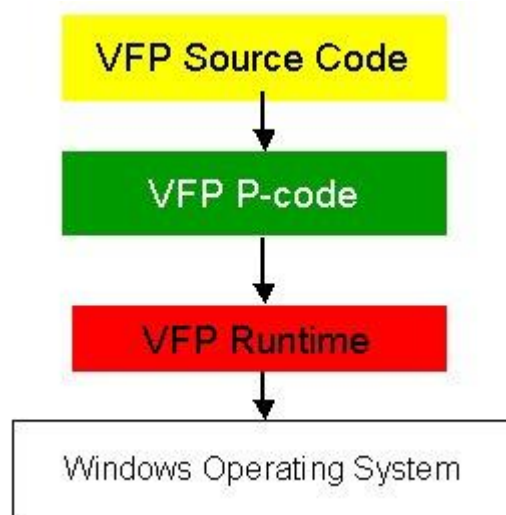


图 1. Visual FoxPro 运行库在运行时刻解释伪代码为机器代码

在 .NET 中，代码被编译两次——第一次在你开发用的机器上被编译为 MS 中间语言（MS Intermediate Language, 简写 MSIL），第二次是在运行时被 CLR 编译。正如你在图 2 中所见，无论源代码使用哪种语言书写，它也许是 C#，也许是 Visual Basic .NET，也许是 C++，它都被编译为同样的中间语言（IL）。通常情况下，你分发这个中间语言给最终用户。

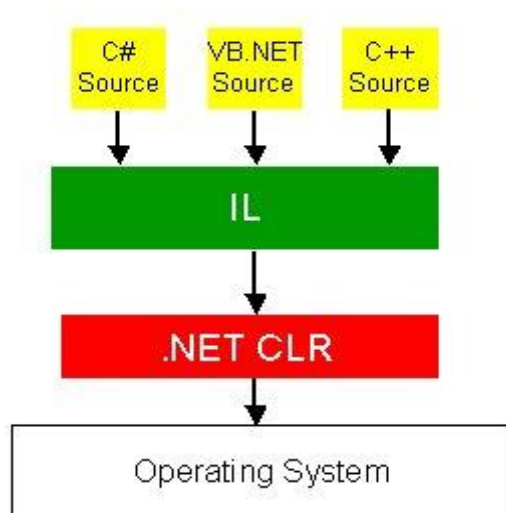


图 2. .NET 代码被编译两次。第一次为 MSIL，第二次在运行时刻被 CLR 编译。

当 MSIL 在最终用户的机器上执行时，它通过一个 .NET Just-In-Time (JIT) 编译器再次编译为机器代码。这种方法的好处是，特定 CPU 的 JIT 编译器可以在最终用户的机器上编译 MSIL 代码到机器代码。所以，如果编译代码的机器具有更新型的处理器和更先进的能力，那么 JIT 编译器就可以利用这些优势。

JIT 编译器仅在代码执行时编译代码。代码执行的结果被缓存起来直到应用程序结束，所以，当同样的代码被再次执行时并不需要再次编译。这和 Visual FoxPro 在运行时解释伪代码到机器代码是不同的（并且，不会将作为结果的机器代码像 .NET 那样缓存起来）。

VFP 开发者这为什么要学 .NET ？

Visual FoxPro 是建立桌面应用、Web 应用以及 Web 服务的伟大的开发工具。所以，有什么理由让 VFP 开发者对学习使用 .NET 感兴趣呢？这一节将列出我已经找到的一些令人信服的理由。

市场

你或许了解这样一件事，由 Visual FoxPro 创建的新的项目只是占已经被创建的软件系统的总数中很小的一个比例。相反，.NET 开发者的需求不断增加，并且会随着时间的推移持续增加。同时拥有 Visual FoxPro 和 .NET 的技术将会使你作为一个员工或顾问获得更多的工作机会。

ASP.NET

如果你还没接触过 .NET，你或许会对使用 ASP.NET 来建立 Web 应用感兴趣。微软已经在 ASP.NET 投入了很多的精力以使它比 ASP 更好的工作。相对于使用脚本语言来创建 Web 应用，你现在可以使用像 C# 或 VB.NET 这样的完全的面向对象的 .NET 语言。

如果你学会了如何使用 Visual Studio .NET 来创建 Windows Forms 应用，那么你可以使用同样的技巧和熟悉的 IDE 来创建 Web Forms 应用和 XML Web 服务，这可以很大程度的减少你的学习曲线。

建立中间层组件

Visual FoxPro 基于它的数据访问和字符串处理速度（尤其对 XML 而言），可以成为创建中间层组件的一个伟大的工具。然而，Visual FoxPro 组件是基于 COM（Component Object Model）基础的。在使用 COM 几年后，我现在可以告诉你 COM 瓶颈的痛苦！这里是我在使用 VFP COM 组件时受到的“三个打击”：

1. 它们调试起来非常痛苦。要搞清楚 COM 组件内部的工作，你通常必须使用 STRTOFILE() 将变量值输出到一个文本文件，以此来确定 COM 是否工作。这真的让人感到非常的沮丧。

2. 它们将会把你带入 DLL 地狱。首先，COM DLLs 必须在 Windows 注册表中进行注册，并且有时（因为一些未知的原因）注册会不成功。此外，如果你在同一机器上有同一 DLL 的不同的版本，你还会遭遇版本维护的问题。

3. 你不能利用 COM+ 对象池，因为 Visual FoxPro 的线程模式——在 Visual Basic 6 中也是如此。对象池允许 COM+ 回收或重新利用中间层组件。当一个中间层组件释放自身的时候，COM+ 会将它放入对象池中以便其他客户端可以再次使用它。注意，即使是 .NET 组件，COM+ 仍旧使用同样的技术来控制中间层组件！

这些问题在 .NET 中的状况：

1. 你可以很容易的使用 Visual Studio .NET 来调试 .NET 组件。实际上，调试器允许你单步调试任何 .NET 语言所书写的组件。例如，你可以单步调试一个 VB.NET 所写的客户端，它调用了用一个用 C# 所写的组件的一个方法。

2. .NET 组件 DLLs 是自描述的，并且不需要在 Windows 注册表中注册。通常情况下，

你只需要拷贝一个 .NET DLL 到一个机器上后，它就可以工作！

3. .NET 组件可以在 COM+ 环境中被控制，并且可以被池化。

语言互操作性

在许多大型的软件开发公司，不同的开发组使用不同的开发语言。.NET 在一个完整的新的层次提供了语言的交互性。你可以用一种 .NET 语言创建一个类，然后使用另一种 .NET 语言来创建它的子类。你也可以在同一个 Visual Studio .NET IDE 中使用不同的语言来进行工作。

有趣的是，其他的供应商（除了微软）正在为诸如 COBOL、PERL、Eiffel、以及其他很少谈论的旧的程序设计语言创建 .NET 版本。

抽象操作系统服务

前面已经提到，.NET Framework 类提供了面向对象的方式来访问底层操作系统的服务。这是一种比调用 Windows API 更好的方式，因为调用 Windows API 假设你运行在 Windows 操作系统上！

.NET Framework 类库增加了一个抽象层，它最终可以允许你的代码移植到非 Windows 硬件平台，例如无线设备和手持设备。

多线程

Visual FoxPro 的一个局限性就是无法创建多线程的应用。如果这对你很重要，那么，.NET 会让你很容易的创建多线程，允许你的应用程序在后台执行任务，例如打印、计算或者发送/检索电子邮件。

你的确可以学习 .NET

如果你已经经历了 Visual FoxPro 的学习曲线，那么你学习 .NET 就已经有了一个很好的开端——这比 Visual Basic 开发者要好的多。这是因为 VB 开发者转移到 .NET 的最大的学习曲线是面向对象。尽管 VB6 是基于对象的，但它并不具有真正的继承（参看第五章 C# 和 Visual Basic .NET 中的面向对象）。相比而言，Visual FoxPro 开发者学习 C# 或

VB.NET 的语法的学习曲线要短的多。

托管代码 (Managed Code)

从 .NET 的角度出发, 世界上存在两种类型的代码: **托管代码** 和 **非托管代码**。

托管代码 是由公共语言运行库 (CLR) 执行和管理。托管代码包含一些元数据, 它提供了一些信息, 允许在运行时刻提供诸如内存管理、安全、垃圾回收这样的服务。所有的 MSIL 代码都是托管代码。关于元数据的更多信息, 参看本章后面的 “清单 (Manifests)” 小节。关于垃圾回收的更多信息, 参看第五章 “C# 和 Visual Basic .NET 中的面向对象”。

非托管代码 是在 .NET 之外运行的代码。这包括位于 COM 服务中的 Visual FoxPro 代码。(图 3)。

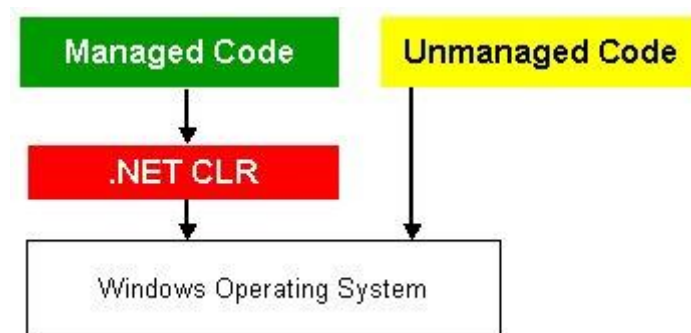



图 3. 在 .NET 公共语言运行库外运行的非托管代码

 代码和非托管代码是不同的。关于不安全的代码的有关信息, 参看第三章 “C# 入门”。

程序集 (Assemblies)

在 Visual FoxPro 中, 依赖于你编译的项目类型, 结果会是 APP、EXE 或 DLL 文件。在 .NET 中, Windows Forms 和 控制台项目被编译成包含 MSIL 的 EXE 文件; Web Forms、Web Services 和类库项目 (包含 Windows 控件库) 被编译成包含 MSIL 的 DLLs。这些 EXE 和 DLL 文件被称为程序集 (assemblies)。

程序集是 .NET 应用程序的主要构成部分。程序集这个术语比物理设计要更合乎逻辑, 因为, 尽管一个程序集通常是由一个单一文件组成, 但是它也可以由一个或多个文件组成。(图 4)。

多文件程序集允许你将程序集分解为更小的单元，它们很容易维护并具有易于下载的很小的文件尺寸。它也允许你创建一个程序集，其中的组件由多种语言构建。

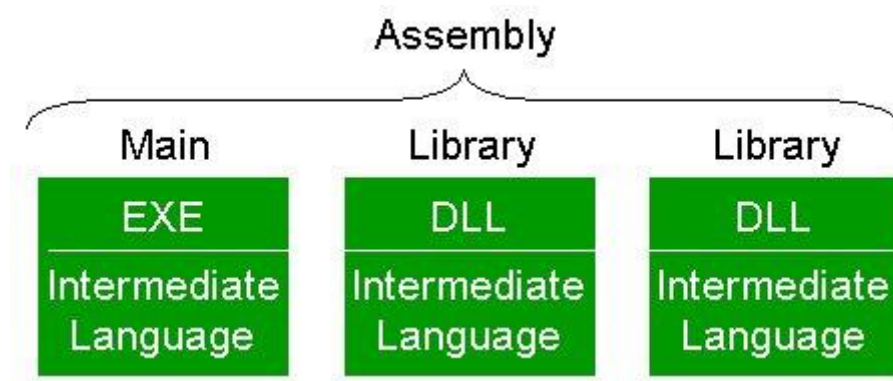


图 4. 一个程序集可以由一个或多个物理文件组成。

程序集是自描述的——它并不需要在 Windows 注册表中注册，因为它拥有一个包含关于程序集元数据的清单（manifest）。

清单（Manifests）

程序集的清单包含一些诸如程序集身份的信息（名称、版本以及区域性（culture）），在程序集中有所有文件的列表、所有引用的程序集以及所有类及成员的详细资料。

查看程序集清单的最好方式是使用 .NET IL 反汇编工具。IL 反汇编程序（ildasm.exe）可以在 FrameworkSDK\Bin 路径下找到，该路径在包含 .NET Framework 的目录下。（译者注：该文件在 VS2003 和 VS2005 中可找到，在 VS2008 中未找到。且，打开的文件，需是对应 VS 版本生成的文件。）你只需要在资源管理器中双击它就可以运行。要查看一个程序集，请执行 **文件|打开**，然后选择一个 .NET 程序集。反汇编结果显示了程序集清单以及在程序集中声明的任何命名空间（namespaces）（图 5）。关于命名空间（namespaces）的更多信息，参看本章后面的“命名空间（Namespaces）”一节。

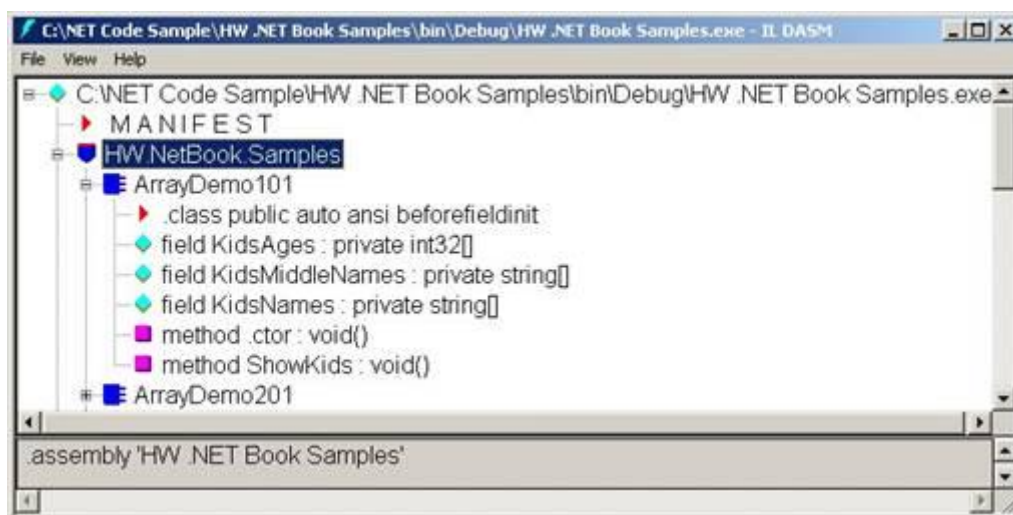


图 5. IL 反汇编程序允许你查看一个程序集的内容，包括任意的 MSIL 代码。

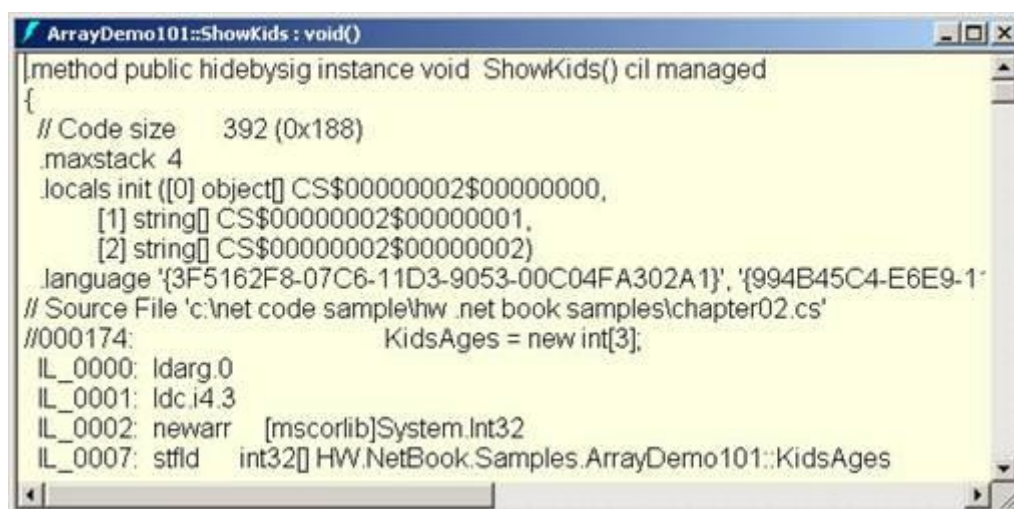
如果你展开命名空间节点，将会显示属于该命名空间的所有的类。图 6 显示了一些在反汇编程序中使用的图标列表及其描述。你可以在反汇编工具的帮助中找到这个列表。针对这些内容的详细描述（如类、接口、方法、静态方法等）请参看第五章 “C# 和 Visual Basic .NET 中的面向对象”。

命名空间：		（蓝色盾状图形）
类：		（带有三条突出短线的蓝色矩形）
接口：		（带有三条突出短线的蓝色矩形，并且有“I”标记）
值类：		（带有三条突出短线的棕色矩形）
枚举：		（带有三条突出短线的棕色矩形，并且有“E”标记）
方法：		（紫红色矩形）
静态方法：		（带“S”标记的紫红色矩形）
字段：		（青色菱形）
静态字段：		（带“S”标记的青色矩形）
事件：		（向下指的绿色三角形）
属性：		（向上指的红色三角形）
清单或类信息项：		（向右指的红色三角形）

图 6. IL 反汇编程序使用不同的图标来表示程序集中不同的项目。

如果你双击反汇编器中树状视图的 Manifest 节点，将会打开一个子窗口，在其中显示程序集的清单。当你打开程序集后，在窗口的顶部，你或许能看到程序集所引用的外部程序集列表（例如：“assembly extern System.Windows.Forms”）。你看到的第一行没有“extern”关键字的程序集引用表示从那里开始是清单的“identity”小节。在清单的 identity 段指定了程序集的名字。Identity 段也包含程序集的版本号。版本号显示在 “.ver” 的后面。

IL 反汇编程序最酷的特性大概是可以预览 MSIL 代码。如果你双击一个类的方法，它会打开一个窗口来显示 MSIL 代码。如果你想看到图 7 所显示的 IL 代码，你可以选择 视图|显示源行 菜单。对许多开发者来说，相对于工具能带来的其他好处而言，可以看到源代码是所有想做的很酷的事情之一。



```
ArrayDemo101::ShowKids : void()
.method public hidebysig instance void ShowKids() cil managed
{
    // Code size 392 (0x188)
    .maxstack 4
    .locals init ([0] object[] CS$000000002$00000000,
        [1] string[] CS$000000002$00000001,
        [2] string[] CS$000000002$00000002)
    .language '3F5162F8-07C6-11D3-9053-00C04FA302A1', '{994B45C4-E6E9-1'
    // Source File 'c:\net code sample\hw .net book samples\chapter02.cs'
    //000174: KidsAges = new int[3];
    IL_0000: ldarg.0
    IL_0001: ldc.i4.3
    IL_0002: newarr [mscorlib]System.Int32
    IL_0007: stfld int32[] HW.NetBook.Samples.ArrayDemo101::KidsAges
}
```

图 7. IL 反汇编程序允许你预览实际的 IL 代码、并随意的查看源代码。

私有和共享程序集（Private and shared assemblies）

存在两种类型的 .NET 程序集，**私有的** 和 **共享的**。一个私有程序集仅被用于一个应用程序。程序集存储在应用程序目录中。这使它很容易的安装和使用——你所要做的只是拷贝它到应用程序目录中。你也可以将一个程序集放在应用程序的子目录中，你只需要设置文件夹名和程序集文件名相同即可。例如，如果你有一个名为“MyLibrary”的程序集，你可以将它放在目录名为“MyLibrary”的子目录下。这样，你就可以将外部的程序集和你的项目文件相分离。

共享程序集可以用于多个应用程序。共享程序集存储在一个称为 **全局程序集缓存**（Global Assembly Cache，简写 GAC）特别的目录中。默认情况下，这个目录是 c:\winnt\assembly 或 c:\windows\assembly。共享程序集的一个例子就是 .NET Framework。所有的 .NET 应用程序都需要访问 .NET Framework 程序集，所以，它们都位于全局程序集缓存中。

在全局程序集缓存中查看程序集

当你在你的计算机中安装 .NET Platform SDK 时，它自动载入一个被称为程序集缓存查看器的 Windows shell 扩展，这个扩展允许你在 Windows 资源管理器中查看全局程序集缓存。你所需要做的就是打开 Windows 资源管理器并定位到 <windows directory>\assembly 目录，然后在其中查看程序集（图 8）。

程序集名称	版本	区域性	公钥标记
System	1.0.5000.0		b77a5c561934e089
System.Configuration.Install	1.0.5000.0		b03f5f7f11d50a3a
System.Configuration.Install.resources	1.0.5000.0	zh-CHS	b03f5f7f11d50a3a
System.Data	1.0.5000.0		b77a5c561934e089
System.Data.OracleClient	1.0.5000.0		b77a5c561934e089
System.Data.resources	1.0.5000.0	zh-CHS	b77a5c561934e089
System.Design	1.0.5000.0		b03f5f7f11d50a3a
System.Design.resources	1.0.5000.0	zh-CHS	b03f5f7f11d50a3a
System.DirectoryServices	1.0.5000.0		b03f5f7f11d50a3a
System.DirectoryServices.resources	1.0.5000.0	zh-CHS	b03f5f7f11d50a3a
System.Drawing	1.0.5000.0		b03f5f7f11d50a3a

图 8. 你存储在全局程序集缓存中的共享程序集，默认路径为 c:\winnt\assembly 或 c:\windows\assembly 。

关于创建和安装共享程序集的更多信息，请参看第 2 章 “Visual Studio .NET” 中的“创建和安装共享程序集”一节。

命名空间（Namespaces）

在 .NET 中，依靠**命名空间**（namespaces）来避免类名的重复。命名空间是一种命名机制，它允许你针对你的类有逻辑的予以声明。例如，在 .NET Framework 中，存在像下面这样的命名空间：

- System.Drawing
- System.Data.SqlClient
- System.Windows.Forms
- System.Web.Services.Protocols

一开始的时候，许多 Visual FoxPro 开发者认为命名空间等同于 VFP 的类库，但是它们之间真的是有很大的不同。一个 VFP 类库是控制一个或多个类的物理容器。而一个 .NET

命名空间却和类的物理位置无关——它纯粹是一个将类区别开来的逻辑名称。

命名空间的组成通常是从左至右、先常规后具体的方式。这在概念上和生物学的分类方法很相似。例如，图 9 显示了红狐的分类层次。国家、门、亚门等等，每个层次都是从常规到特定的种类。这样的约定可以对整个生物学进行分类。

```
Kingdom Animalia
  Phylum Chordata-- chordates
    Subphylum Vertebrata-- vertebrates
      Class Mammalia-- mammals
        Subclass Theria
          Infraclass Eutheria
            Order Carnivora-- carnivores
              Suborder Caniformia
                Family Canidae-- coyotes, dogs
                  Genus Vulpes-- kit foxes, red foxes
                    Species Vulpes vulpes-- red fox, red fox
```

图 9. 命名空间允许你像生物学分类方法那样对你的类进行分类。

微软建议你用以下的方式来定义你的命名空间：命名空间的第一部分是你的公司名，第二部分是你的产品名，第三部分是特定的类的分类名，如此等等。例如，我定义了我公司的所有商业对象都隶属于“OakLeaf.MM.Business”。Oak Leaf 是我公司的名字，MM 是产品名（Mere Mortals Framework），Business 指商业对象类。

关于如何指派类到一个命名空间，请参看第 3 章“C# 入门”和第 4 章“Visual Basic.NET 入门”。

.NET 程序语言

正如这里所说的一样，有三种主要的 .NET 语言你可以选择：Visual Basic.NET、Visual C#.NET、Visual C++.NET。由于大多数 Visual FoxPro 开发者选择 Visual Basic.NET 或 Visual C#.NET 作为他们的软件开发语言，所以这本书将着重于这两种语言，分别提供示例，以便帮助你作出适合自己的决定。

Visual C# .NET

尽管“Visual C#.NET”是语言的“官方”名称，但是你也经常看到它被简单的称呼为

“C#”（读音为 C sharp），在本书中我也将使用这个约定。

C# 是针对 .NET 所写的全新的语言，并且它也带来很多振奋人心的东西。事实上，微软使用它创建了 .NET Framework 基类。截止本书为止，已经有很多的 Visual FoxPro 开发者选择 C# 作为学习 .NET 的开发语言。

因为 C# 是“C”家族之中的一员，它的语法和 C++ 很相似，但是它更多的更像 Java。C# 也被用来创建 Windows Forms、Web Form 应用、XML Web 服务、控制台应用程序以及类库等等，更重要的是，C# 在设计时结合了 C 和 C++ 的能力和 control，并结合了 Visual Basic 的易用性来获得更高的生产力。关于 C# 的详细资料，请参看第 3 章“C# 入门”。

Visual Basic .NET

Visual Basic .NET 是微软 Visual Basic 程序语言的全新版本。它和 VB6 的差异基本与 VFP 和 FoxPro 2.6 之间的差异一样大——甚至更多。

与它的前身 Visual Basic 6 不同，VB.NET 是一个真正的面向对象语言，它已经针对 .NET Framework 完全的重新设计。你可以使用 VB.NET 创建 Windows Forms 应用、Web Form 应用、XML Web 服务、控制台应用程序、类库等等。已赋予新的生命的 VB.NET 也充当着“粘和”的角色，填充了一些空白并与应用程序紧密相连。你可以使用 Visual Basic .NET 创建在 Visual Studio .NET IDE 中使用的宏。

如你所愿，VB.NET 拥有很好的易用性，并且，你会在后续版本的 .NET 中享受更多这样的便利。

关于 Visual Basic .NET 的详细信息，请参看第 4 章“Visual Basic .NET 入门”。

命名规范

微软已经提出了一个针对 .NET 语言的命名规范。他们希望更多的开发者可以接受这个规范，以便于在开发人员之间增强代码的可读性。一个令很多开发者感到惊讶的规定是停止使用匈牙利命名法。

你可以在下面的站点找到正式的命名规范：

<http://msdn.microsoft.com/library/en-us/cpgenref/html/cpconnamingguidelines.asp>

（翻译者注：最新版的中文版命名规范：

<http://msdn.microsoft.com/zh-cn/library/ms229002.aspx>)

常规的开发步骤

在这一章，在一个更高的层次上给出创建一个 .NET 应用的开发步骤是恰当的。从全局角度讲，全书的每一章都和开发步骤有关联。

需求调查

在你开始书写代码之前，你需要收集针对应用的需求。这些年以来，许多的开发商都开始在为这一目标使用统一建模语言（Unified Modeling Language，简写 UML）和统一软件过程（Rational Unified Process，简写 RUP）。微软已经认可了这种分析的重要性，并计划通过在他们的 Visio diagramming tool 中增加 UML diagrams 来使 Visio 和 Visual Studio .NET 更好的协同工作。



在本节中我使用的 UML 数据你或许并不熟悉。要学习关于统一建模语言的知识，请查阅书籍，例如由 Addison-Wesley 所著的《The Unified Modeling Language User Guide》。

第一步的需求收集也包括向用户展示的软件系统最终如何使用的文档。这里使用的主要手段就是使用 UML 的 case diagrams 。

开发计划

在调查完需求后，你需要建立一个开发计划。这包括确定项目周期以及组件的实现顺序。如果你使用 UML，你就可以更进一步的分析、设计和实施。

设计和创建商业对象

在实现设计期间，你可以使用 cases 并针对每个使用需求建立商业对象。这意味着在商业对象类中增加方法以包含大多数的应用逻辑。这种方法和现在许多 Visual FoxPro 开发者使用的方法大不相同——他们将大多数应用逻辑放置于用户界面。然而，使用商业对象可以使你的应用程序更灵活、更易于伸缩、更具有可维护性。

通常情况下，你需要在 C# 或 VB.NET 中创建一个类库项目来包含你的商业对象。这个项目可以被编译为一个程序集以便其他应用程序可以使用它（Windows Forms、Web Forms、

Web 服务等等)。

关于设计实现商业对象的更多信息，请参看第 8 章 “.NET 商业对象”。

建立数据模型

与设计商业对象一样，你要开始考虑应用的数据模型。即使我在设计一个 C/S 架构的应用时，我也经常发现在 Visual FoxPro 中创建原型表是一个伟大的“概念验证”技术。因为 Visual FoxPro 开发者倾向于从数据的角度来看这个世界，把数据放入表中可以帮助他们更快的找到对象模型和用其他方法无法找到的缺陷。

最终，在你真正开始使用所选择的语言创建商业对象前，你要完成和建模相关的工作。

关于在 .NET 应用中访问数据的更多信息，请参看第 7 章“使用 ADO.NET 进行数据访问”。

创建用户界面

如果你正在创建一个 Windows Forms 或者是 Web Forms 应用，你就需要去设计实现用户界面。你可以使用 Visual Studio .NET 来创建新的 Windows Forms 或 Web Forms，并在项目中增加一个到商业对象库的引用，以便你的用户界面可以使用这些类。

正如前面提到的一样，你并不需要在你的界面中放置应用逻辑。把你的用户界面当做你的应用的“skin”，它可以很容易的使用其他的 skin 来替换。

你可以在你的用户界面中放置一些代码来实例化商业对象，并调用它们的方法去完成像检索数据、操作和保存数据、进行计算等等这些服务。

关于建立应用程序用户界面的更多信息，请参看第 9 章“建立 .NET WinForm 应用程序”和第 10 章“使用 ASP.NET 建立 Web 应用”。

创建 XML Web 服务

如果你正在创建一个 XML Web 服务，那么你可以在 Visual Studio .NET 中创建一个新的 Web 服务项目。然后，你可以在 Web 服务项目中增加对商业对象的引用，以便 Web 服务可以使用这些类。

关于建立 XML Web 服务的更多信息，请参看第 12 章“XML Web 服务”。

总结

微软正在 .NET 上下赌注。他们已经在 .NET 上投入了大量的资源，并取得了一些非常令人深刻的技术。在技术上它是一个巨大的转变，并引起了所有的关注——包括 Visual FoxPro 开发者。本书的其余部分将帮助你做出明智的决定——为什么要使用 .NET？什么时候使用？在哪里使用它进行你的软件开发？

第二章 Visual Studio .NET

Visual Studio .NET 是微软最新的软件开发工具套件。VS.NET 小组已经做了大量的工作使开发人员获得顶级的体验，无论他们使用哪种 .NET 语言。这一章将带领你在 .NET 中漫游，熟悉它的功能，以便可以快速的使用它。

在 .NET 之前，Visual Studio 中的工具是是一个个松散的相互关联的产品安装包——Visual Studio .NET 结束了这样的历史。开发团队无论是使用 C#，还是 Visual Basic .NET，抑或是 C++，现在都使用统一的集成开发环境（Integrated Development Environment，简写 IDE），在其中工作是一件很愉快的事。

起始页

学习 .NET 的最大挑战之一是查找资讯。Visual Studio .NET 的起始页完全可以帮助你找到快速启动和运行的信息。很多开发者不曾花一些时间来仔细查看起始页，所以，这里我将带领你快速的对其予以浏览。

当你运行 VS.NET 时，起始页看起来像 图 1 中显示的那样。这个页面被显示在一个 Web 浏览标签中（标签是指 VS.NET IDE 中一个选项卡式文档），它包含一个具有导航按钮的工具栏和一个组合框，在那里，你可以键入任何有效的 URL。

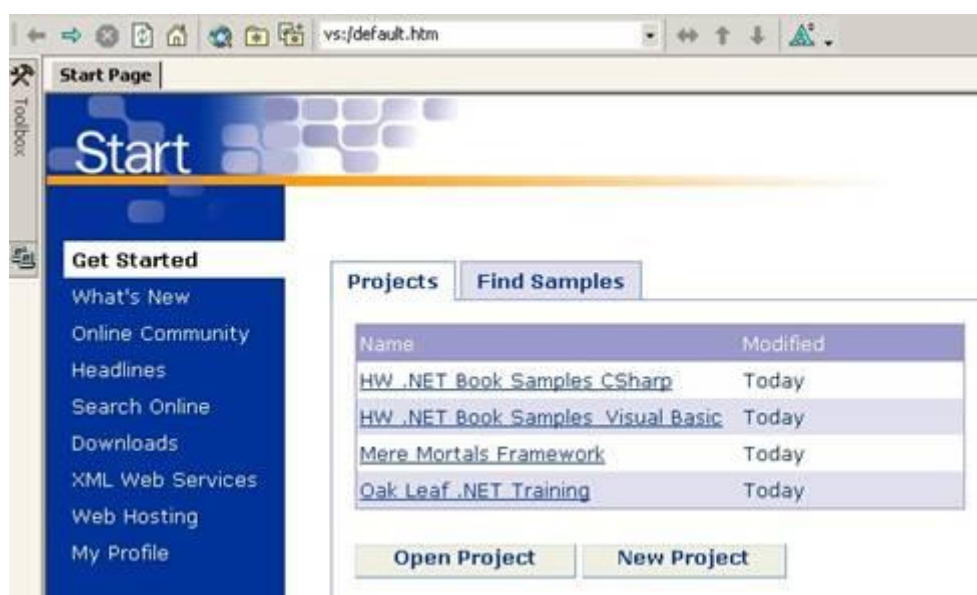


图 1. VS .NET 的起始页显示了你最近使用的项目。

开始（Get Started）

开始窗格显示为一个页框，它包含两个标签——项目 (Projects) 和查找示例 (Find Samples)。项目标签允许你打开一个项目或创建一个新项目。查找示例标签允许你查找在线的语言 (C++、C#、Visual Basic .NET) 示例或其他 Visual Studio 开发者提供的示例，并显示在其中。在这里，有大量的示例可供你参考。我个人极力推荐你使用它。

设置一个筛选器

当你查看在线社区、新闻提要 (Headlines) 和下载 (Downloads) 窗格时，你可以设置一个筛选器来限制有那些信息显示出来 (图 2)。这个过滤器在 VS.NET 运行期间都是有效的。如果你设置筛选器上的语言，我建议你选择 语言+相关信息(language “and related”) 选项 (例如，“Visual C# and related”)。如果你选择了例如 “Visual C#” 这样的选项，你会错过很多重要的信息。

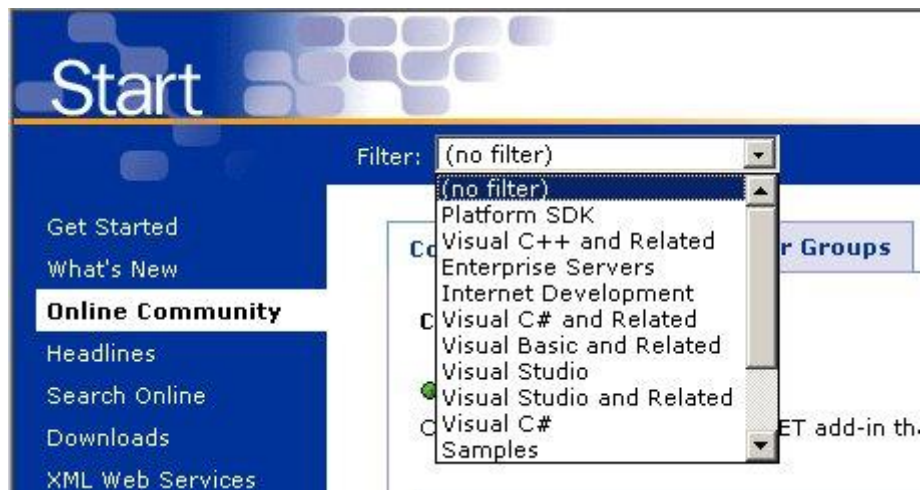


图 2. 你可以在开始页的在线选项里设置一个筛选器来显示所看到的信息。

新增内容（What's New）

当你的计算机并没有连接到 Internet 时，如过你选择了新增内容窗格，你可以看到一个 .NET 帮助主题清单，它包含了在 .NET 中哪些方面有新的内容，当然，也包括语言方面

的内容。

然而,如果你的计算机连接到了 Internet ,你可以看到一个技术(Technology)标签,它包含有一个详细的选项清单,这些选项包括 Visual Studio .net Service packs、Windows security 和 Platform SDK updates 的链接,也有许多包含关于 .NET 信息的 Web 页面链接(大多是微软的网页)。你还可以看到一个训练和事件(Training and Events)标签,它链接到即将举办的 .NET 培训活动。另一个更酷的功能是它包含一个技巧和演示(Tips and Walkthroughs)标签,它链接到基于 Web 的演示,用来解释基本和高级的 .NET 概念。

在线社区 (Online Community)

在线社区窗格允许你定位到在线代码示例、新闻组、用户组、查找特定 .NET 技术专家以及查找 .NET 的第三方插件。图 3 显示了如果你所在地区查找 .NET 用户组。



图 3. 在线社区|VS.NET 起始页的用户组选项卡允许你查找本地区的用户组。

新闻提要 (Headlines)

起始页的新闻提要窗格向你显示了在[微软 MSDN 站点](#)当前的在线新闻提要。这是一个很棒的地方,可以让你和 .NET 及其他微软技术保持同步。实际上,我已经将它作为我的 IE 默认主页很长一段时间了。它是和新技术保持联系的很好的一个方式。

在线搜索 (Search Online)

在线搜索窗格允许你在 MSDN 在线知识库中进行搜索。如果你点击了高级 (Advanced)


按钮, 它将打开一个新的 VS.NET Web 浏览标签, 你就可以在 Microsoft.com 中进行搜索, 查看热门下载、技术支持链接和热门搜索。

下载 (Downloads)

下载窗格包含了丰富的下载链接——包含免费下载和 MSDN 订户下载。

XML Web Services

如果你想在 UDDI 注册中搜索一个特定类型的 XML Web 服务, 你就可以在 VS.NET 中来做。XML Web Services 窗格(图 4)允许你指定一个类别并增加搜索条件, 指定是在 UDDI 产品环境还是在 UDDI 测试环境中进行搜索。关于 UDDI 的更多信息, 参看第 12 章“XML Web Services”。



Find a Service **Register a Service**

UDDI is the Yellow Pages equivalent for XML Web services. This free public registry enables you to search for registered XML Web services to include in your applications. It also lets you publish your own XML Web services to the community. Here you can query UDDI by category or keyword and directly add a reference to your project.

Search in: ☒ UDDI Production Environment ☐ UDDI Test Environment

Category: **Search for:**

Financial credit card **Go** **Advanced**

图 4. 你可以基于类别和附加的搜索条件在 VS.NET 中搜索 Web Services。

虚拟主机 (Web Hosting)

如果你想为你的 .NET Web 应用查找一个 Internet 服务提供商, 虚拟主机窗格可以让你找到提供 .NET Web 服务的 ISP。这些提供商通常提供一个环境供你上传和测试 XML Web 服务或 ASP.NET 应用。

我的配置文件（My Profile）

我的配置文件窗格允许你针对 VS.NET 设置你的工作环境参数。配置文件允许你对键盘方案、窗口布局和帮助筛选器。键盘方案设置列表显示了你可以选择的快捷键配置。窗口布局用来设置一个默认的 VS.NET 窗口布局。帮助筛选器指定你如何使用 VS.NET 的帮助文件（图 5）。显示的帮助选项允许你指定你是否想在 IDE 或是否想在 IDE 之外的一个单独窗口中显示 VS.NET 帮助。At Startup 选项允许你指定当你第一次运行 Visual Studio 时的外观。



如果你设置帮助文件针对某个语言进行过滤，最好选择 语言+“and Related”选项（图 5）。如果你仅过滤 C# 或 Visual Basic .NET，那么很多重要的帮助主题就会被过滤掉。



图 5. 你可以针对 VS.NET 帮助文件指定一个默认过滤设置。

显示起始页

当你在 VS.NET 中查看帮助主题时，它通常会关闭起始页。如果你想再次打开它，请从菜单中选择 **帮助 | 显示起始页**。

现在，你已经知道可以在哪里找到学习 .NET 的帮助，我会继续告诉你使用 Visual Studio .NET 的方法。

解决方案和项目

你需要理解的第一个概念就是 Visual Studio .NET 组织和管理 .NET 应用程序的方式。作为最高层次的组织方式，VS.NET 使用 **解决方案** 以及隶属于解决方案的 **项目**（来组织和管理）。

一个解决方案是一个容器，它可以控制一个或多个相关的项目。解决方案允许你在一个 VS.NET 实例中操作多个项目，也可以设置选项以及针对一组项目设置常用的工作环境。当你创建一个新的项目时，VS.NET 会自动创建一个解决方案来容纳这个新的项目。

VS.NET 中的一个项目，非常类似于 Visual FoxPro 中的一个项目。它也拥有相关的文件，例如源代码、窗体（表单）、用户界面控件、文本文件、图象等等。你可以编译一个项目为一个 EXE、DLL 或者是其他项目的一个单元。当你编译一个解决方案时，针对解决方案中的每个项目都会生成一个单独的输出文件。

创建一个新项目

要创建一个新项目，在 VS.NET 起始页单击“New Project”按钮。这会运行一个新建项目对话框（图 6）。在项目类型窗格中选择你所想创建的项目类型。



图 6. 你可以使用你选择的语言，基于多种项目模板来创建新的项目。

你可以选择去创建一个 Visual Basic 项目、C#项目、C++ 项目或 安装和部署项目，或者是一个 Visual Studio 解决方案（一个空的解决方案）。“其他项目”节点中，允许你选择创建数据库项目（其中包含脚本和查询访问多种数据库）；企业级模板项目用来创建分布式应用；Visual Studio Analyzer 项目用来收集事件信息并执行项目分析；扩展项目用于创建 VS.NET 的插件；此外还有一个 Application Center Test 项目可选。

项目模板

在 Visual FoxPro 中，当你创建一个新项目时，项目是空的。但是在 Visual Studio .NET 中却提供有多种类型的项目模板可供选择。如果你在项目类型窗格中选择了 Visual Basic 或 C# 项目，在右侧的模板窗格中就会显示一系列模板（表 1）。

表 1. Visual Basic .NET 和 C# 项目模板

项目模板	描述
Windows 应用程序	传统的 Windows 应用程序 (WinForms)
类库	可以被其他项目引用的类库
Windows 控件库	用于 Windows Forms 的 Windows 控件类库
ASP.NET Web 应用程序	Web Forms 应用程序
ASP.NET Web 服务	XML Web 服务
Web 控件库	用于 Web Forms 页的 Web 控件类库
控制台应用程序	命令行应用程序
Windows 服务	无用户界面的 Windows Service 应用程序
空项目	空的 Windows 项目
空 Web 项目	空的 Web 项目
在现有文件夹中创建新项目	在已存在的文件夹中创建空项目

创建 WinForms 应用时常用的模板是 Windows 应用程序(参看第 9 章“建立 .NET WinForm 应用程序”)、ASP.NET Web 应用程序模板(参看第 10 章“使用 ASP.NET 建立 Web 应用”以及 ASP.NET Web 服务模板(参看第 12 章“XML Web 服务”)。

示例：建立一个 Visual Basic Windows 应用程序

为了帮助你理解解决方案和项目，我将告诉你如何从头创建一个 Visual Basic Windows 应用程序。这一节所描述的内容，同样适用于 C# Windows 应用程序。

要创建一个新的 Visual Basic .NET Windows 应用程序，需要下列步骤：

1. 在 VS.NET 起始页点击“New Project”按钮。这会显示一个新建项目对话框。

2. 在项目类型中选择“Visual Basic 项目”，在模板中选择“Windows 应用程序”。
3. 在“名称”文本框中，键入新项目的名称。我这里是“My First VB Windows App”。
4. 在“位置”文本框，选择新项目所在的文件夹。当你第一次安装 Visual Studio .NET 时，这个对话框显示一个指定的文件夹作为默认项目目录，但是你可以更改它为你所希望的位置。

如果这时在 VS.NET 中已经打开了一个解决方案，你会看到一个额外的选项，它可以允许你将在哪里增加新项目，或者是关闭解决方案(参看图 6)。如果你选择了“关闭解决方案”，VS.NET 会为你的新项目自动创建一个新的解决方案。
5. 默认情况下，新的解决方案具有和项目同样的名称。但是，如果你定义为另外的名字，你可以单击“更多”按钮。这会显示一个“创建解决方案目录”的选择框。如果你选定了它，“新解决方案的名称”文本框就变为可用，你可以在这里指定一个新的解决方案名称。这两种方式，都会在底部显示一个消息“将在……处创建新项目”，消息中包含项目将被创建到的目录。注意：VS.NET 会在“位置”文本框中指定的目录下创建解决方案和项目的新路径。
6. 单击“确定”按钮创建新项目和与之关联的解决方案。稍等片刻，新的解决方案和项目就会显示在 VS.NET IDE 中(图 7)。

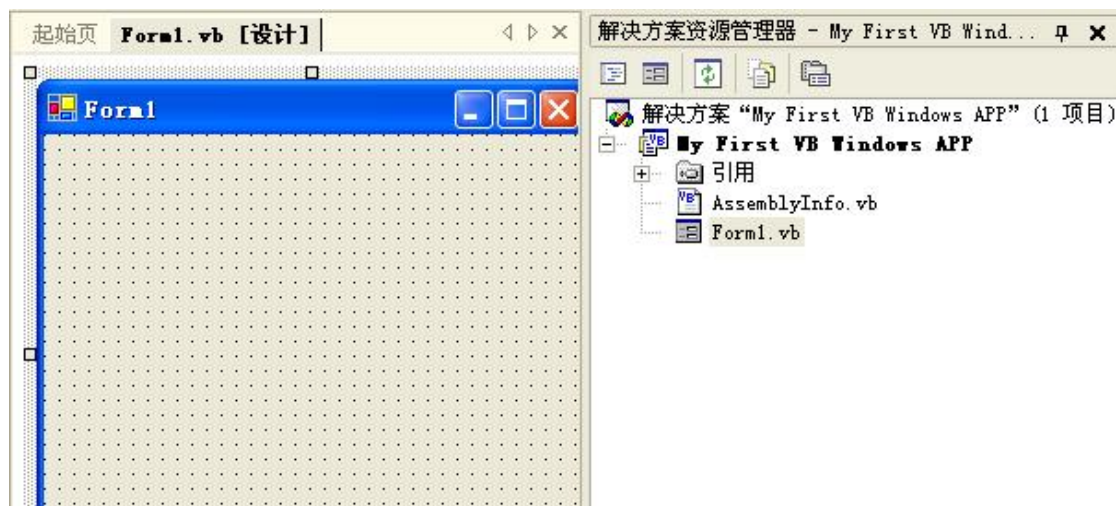


图 7. VS .NET 基于你选择的模板创建新的项目。解决方案资源管理器可以用来有组织的查看项目和其中的文件。

在左边，VS.NET 显示了一个新的名为 Form1 的空窗体。窗体上的标签显示出是在设计模式，窗体被存储在名为 Form1.vb 的文件中。



所有 Visual Basic .NET 源文件(窗体、类、单元、组件、控件)都具有 .vb 扩展名。所有 C# 源文件都具有 .cs 扩展名。这和 VFP 不同类型的文件具有不同扩展名是不同的。

在解决方案资源管理器中检查新的解决方案

在 IDE 右侧，你会看到解决方案资源管理器。如果没看到，请选择 **视图|解决方案资源管理器** 菜单。解决方案资源管理器提供了项目有组织的视图并显示相关的文件。

解决方案资源管理器的第一个节点包含了新解决方案的名称以及在解决方案中包含多少项目（图 7）。



默认情况下，并不是所有条目都显示在解决方案资源管理器中。要查看所有条目，单击解决方案资源管理器顶部的“显示所有文件”按钮（图 8）。



图 8. 要查看解决方案中的所有文件，单击“显示所有文件”按钮。

本质上，一个解决方案定义被存储在两个不同的文件中，一个是 .sln，一个是 .suo。例如，刚才创建的新项目是由“my first vb windows app.sln”和“my first vb windows app.suo”两个文件组成。.sln 文件存储解决方案元数据的详细设计信息，项目是与解决方案相关联的，附加的条目被增加到解决方案层（这比附加到单独的项目要好），然后建立配置文件。.suo 文件存储特定解决方案的用户选项（例如文档窗口位置），这些选项可以让你自定义 IDE。如果你很好奇，你可以在 Visual FoxPro 中打开 .sln 文件（它实际上是一个文本文件）来查看其中的内容。

检查新项目

解决方案资源浏览器的第二个节点包含新项目的名称。默认情况下，它和解决方案同名，这里是“**My First VB Windows App**”。展开节点下的“引用”节点，你将看到已被自动添加到项目的 .NET 组件 DLLs 列表（图 9）。



图 9.当你创建新项目时，VS.NET 自动增加所引用的 .NET 组件 DLLs 到你的项目中。

如果你单击一个引用，相关 DLL 的信息就会显示在属性窗口中。如果属性窗口不可见，请选择 **视图|属性窗口** 菜单。

在“引用”节点下面，存在一个名为 `AssemblyInfo.vb` 的文件。如果你双击它，将会打开一个包含文件内容的标签。在这个文件中，你要注意两个地方。第一个是程序集属性的

列表（图 10）。属性允许你在其中增加一些描述，用来注释程序集、类、方法、属性和其他元素。当你编译项目时，这些属性会被当做附加的描述信息由编译器增加到 IL 中。关于属性的更多信息，请参看第 5 章“C# 和 Visual Basic .NET 中的面向对象”。



```
<Assembly: AssemblyTitle("")>
<Assembly: AssemblyDescription("")>
<Assembly: AssemblyCompany("")>
<Assembly: AssemblyProduct("")>
<Assembly: AssemblyCopyright("")>
<Assembly: AssemblyTrademark("")>
<Assembly: CLSCompliant(True)>
```

图 10. 你可以设置程序集属性值来作为你的程序集的附加信息来提供给用户。

除了显示在图 10 的属性之外，文档中还包含在创建新项目时自动生成的 Guid 属性（全局唯一标识符）。如果项目从一个 COM 进行访问，它将用于你的程序集的类型库，关于和 COM 交互操作的详细信息，请参看第 15 章“和 Visual FoxPro 的交互操作”。在 AssemblyInfo.vb 的最后部分，有一个 AssemblyVersion 属性。注释中说明你可以忽视这些属性，并使用星号来作为次要的版本号，VS.NET 会自动生成新的版本号。否则，你要移除星号并写上版本号。

到这里，还有 Windows Form 文件 Form1.vb 没有讨论，这将在第 9 章“建立 .NET WinForm 应用程序”中进行讨论。

在已有解决方案中添加其他项目

在这一节，我将通过创建一个新的 C# 项目来展示 .NET 的语言互操作性，你可以增加它到已存在的解决方案中。

要增加一个项目到已存在的解决方案，首先要在 Visual Studio .NET 中打开一个解决方案（如果它还没有被打开），然后，按照下面的步骤来操作：

1. 在解决方案资源管理器中，在解决方案（最上面的第一个节点）上右击鼠标并选择 **增加|新项目** 菜单，或者在起始页点击“新建项目”按钮。这将显示一个“添加新项目”对话框。
2. 选择 Visual C# 项目，并在模板中选择“类库”。

3. 在名称文本框中输入“My CSharp Class Library”。注意，这里要拼写成“CSharp”，因为项目名称不能包含符号(#)。
4. 点击“确定”来创建新项目。少等片刻，在解决方案资源管理器中就可以看到新的 C# 项目。(图 11)。



图 11. 你可以像这样增加多个项目或不同语言的项目到一个解决方案。

指定启动项目

注意：“My First VB Windows App”项目是高亮显示，而新的 C# 项目则不是。这表示 VB 项目是启动项目。启动项目是特定的项目，或者当你开始执行 Visual Studio 调试器时的项目。如果你仅想指定解决方案中的一个项目为启动项目，你只需要右击解决方案管理器中的项目并在快捷菜单中设置其为启动项目。

当你运行 VS.NET 调试器时，如果想建立、运行和调试多于一个项目，你可以在解决方案属性页对话框中指定多个启动项目。要运行这个对话框，你只需在解决方案资源管理器中右击解决方案节点并选择“属性”菜单。要查看启动项目属性，需展开左边通用属性并选择“启动项目”（图 12）。



图 12. 解决方案属性页对话框允许你在解决方案中指定一个或多个启动项目。

默认情况下，一个解决方案仅有一个启动项目。如果你想指定多个，可以选择“多启动项目”选项。下一步，单击“操作”，然后设置你想作为启动项目的项目为启动项目，并选择“启动”或“开始执行（不调试）”。通常情况下，在设计模式中，你要选择“启动”以便你而已调试你的项目。如果你指定了多个启动项目，当你返回到解决方案资源管理器时，解决方案名会变为黑体显示。

针对示例项目，我将忽略其他设置，将 VB Windows 项目设置为启动项目。

生成一个解决方案

尽管第一个 .NET 解决方案是如此的简单，但是它确实是学习如何编译一个解决方案和/或项目的一个很好的起点。

生成配置

生成配置允许你指定解决方案中的项目如何被生成以及最终如何部署。在 Visual Studio .NET 中，存在两个不同层次的配置——解决方案配置和项目配置。

解决方案配置

当你创建新的解决方案时，VS.NET 自动创建了两个配置——一个“Debug”配置和一个“Release”配置。

要查看解决方案的配置，右击解决方案资源管理器中的解决方案并在快捷菜单中选择“配置管理器”菜单。这将运行配置管理器（图 13），它显示了解决方案中的所有项目以及它们的配置、目标平台以及一个表示项目是否被生成的选择框。

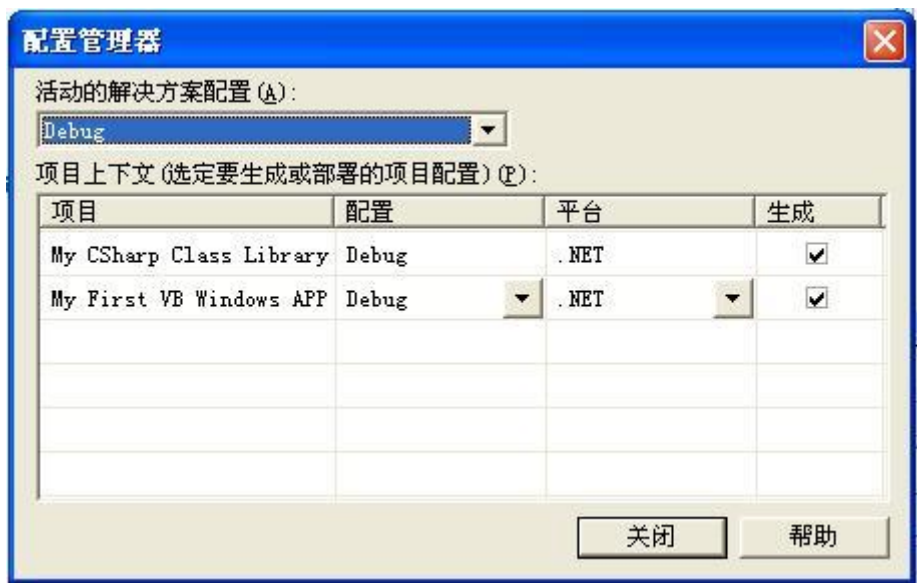


图 13. 配置管理器允许你查看、编辑并创建解决方案的生成配置。

默认情况下，Debug 配置显示在组合框中。在你设计和生成应用程序期间，通常情况下活动配置为 Debug。如果你准备为部署生成应用程序，它统称被配置为 Release。一个被编译后的应用通常不允许你进行调试，但是和 Visual FoxPro 一样，如果关闭调试可以使它运行的更快。要查看解决方案的默认 release 配置，请在“活动的解决方案配置”组合框选择“Release”。

项目配置

项目配置允许你指定项目如何生成。它包含是否包含调试信息到生成的文件、如果或如何优化输出文档、定位输出文件的位置等等。这很类似于 Visual FoxPro 的项目信息对话框。

要查看项目的生成配置，右击解决方案资源管理中的项目并选择“属性”快捷菜单。这将运行项目属性对话框（图 14）。在“配置属性”下有四种类型的配置设置列表：调试、优化、生成、部署。

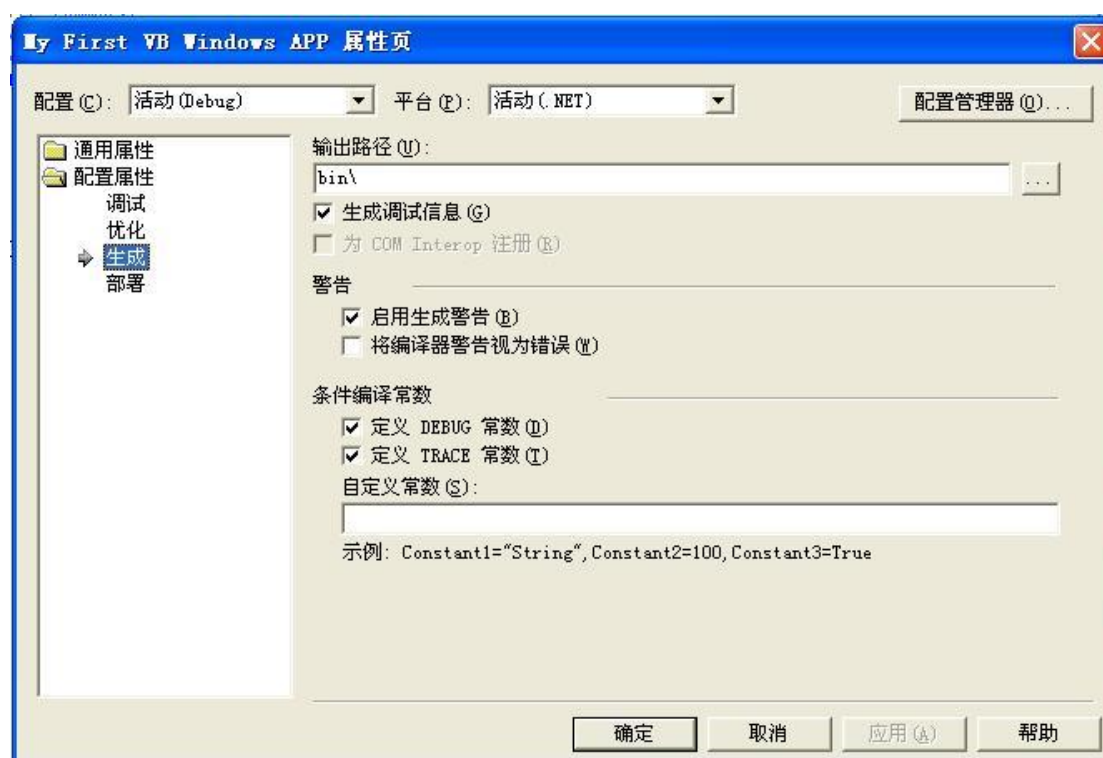


图 14. 你可以设置一个解决方案中每个项目的多种生成选项。

关于这些配置的附加信息，请在选择这些项目后单击“帮助”按钮。

生成选项

要生成一个项目，请选择“生成菜单”，共有四个生成选项：

- 生成解决方案 - 生成解决方案，包括解决方案中的所有项目

- 重新生成解决方案 - 重新生成解决方案，清理解决方案配置文件
- 生成项目 - 生成一个指定的项目
- 重新生成项目 - 重新生成一个指定的项目，清理项目配置文件

生成示例解决方案

如果你已经按照上面的步骤创建了一个实例解决方案，那么现在就可以生成它。你只需要执行 **生成|生成解决方案** 菜单。

这时，Visual Studio .NET 按照每个项目的配置为你生成项目输出文件。当生成是一个 Debug 配置时，默认输出文件会放置在项目的 bin\Debug 子文件夹。

当你生成解决方案时，如果编译器遇到错误，就会显示在输出窗口。如果成功编译，它会在输出窗口显示一个“已完成”信息，在其中显示没有任何错误发生（图 15）。

```
----- 已启动生成: 项目: My First VB Windows APP, 配置: Debug .NET -----  
  
正在准备资源...  
正在更新引用...  
正在执行主编译...  
正在生成附属程序集...  
  
----- 已启动生成: 项目: My CSharp Class Library, 配置: Debug .NET -----  
  
正在准备资源...  
正在更新引用...  
正在执行主编译...  
  
生成完成 -- 0 个错误, 0 个警告  
正在生成附属程序集...  
  
----- 完成 -----  
  
生成: 2 已成功, 0 已失败, 0 已跳过
```

图 15. VS .NET 在输出窗口显示的生成进程的过程和结果。

关于控制编译错误的更多信息，请参看第 13 章“ .NET 中的错误处理和调试”。

检查生成的输出文件

如果你查看“My First VB Windows App”的输出路径，你会看到两个文件——My First Windows App.exe 和 My First Windows App.pdb。第一个文件很明显是一个 Windows 可执

行文件。第二个文件具有 PDB 的扩展名。它表示“程序调试数据库”(program debug database)。

当你生成一个项目时，如果它包含调试信息，VS.NET 会创建一个程序调试数据库，以便你映射 MSIL 到源代码。通常情况下，你不需要直接打开 PDB 文件——它是在 VS.NET 调试应用程序时使用的。

如果你查看“My CSharp Class Library”的输出路径，你也会看到两个文件——My CSharp Class Library.dll 和 My First CSharp Class Library.pdb。因为 C# 项目是一个类库，所以就直接生成了一个 DLL 而不是 EXE 。

运行编译的程序

要运行编译后的示例解决方案，选择 **调试|启动** 菜单。或者按 F5 键，也可以单击标准工具栏上的“启动”按钮（图 16）



图 16. 单击标准工具栏上的启动按钮来执行一个已编译的应用程序。

当应用程序运行时，你会看到一个标题为“Form1”的 Windows 窗体。要关闭这个窗体，只需要点击窗体右上角的关闭按钮。

创建并安装共享程序集

在第一章“.NET 简介”中提到，你可以创建私有和共享的程序集。在私有和共享程序集之间主要的不同是它们的物理位置。共享程序集被存储在全局程序集缓存中，它默认的路径是 <windows directory>\assembly 文件夹。

为共享程序集创建一个强名称（strong name）

因为共享程序集保存在公共的全局程序集缓存中，它可能会和其他程序集出现命名冲突。要避免这个，共享程序集基于私有密钥被赋予一个强名称。



私有密钥的细节已经超出了本书的范围。关于这个主题的细节，请参看 .NET 帮助主题“加密概述”。

为一个共享程序集创建程序集有几个方法。最容易的方法只需要三步。

首先，你需要生成一对公匙—私匙并存储它到一个强名称。.NET Framework 有一个强名称命令行工具(Sn.exe)可以供你使用。要访问它，只需在命令提示符下运行下面的命令：

```
sn -k <outfile>
```

现在，为了生成一对新密钥并存储它们到名为 keyPair.snk 中（snk 扩展名的意思是“强名称密钥”（strong name key）），你需要运行下面的命令：

```
sn -k keyPair.snk
```

下一步，你需要在程序集中引用强名称密钥。要达到这个目的，你需要在 Visual Studio .NET 中打开已创建的程序集。在解决方案资源管理器中，双击 AssemblyInfo.cs 或 AssemblyInfo.vb 文件，并编辑它。

如果你使用的是 C#，你会看到下面的程序集属性：

```
[assembly: AssemblyKeyFile("")]
```

像下面这样在双引号间指定强名称密钥文件：

```
[assembly: AssemblyKeyFile("keyPair.snk")]
```

如果文件位于项目输出路径以外的其他目录，你需要指定一个相对路径，以便于 VS.NET 可以找到它。路径是相对于项目输出路径的，所以，如果你有一个强名称密钥文件位于项目的主目录下，而项目的输出路径为 <Project Directory>\bin\debug，你就需要像下面这样类指定：

```
[assembly: AssemblyKeyFile("../..\keyPair.snk")]
```

在 Visual Basic .NET 中，这个 assembly 属性并不是已经存在的，所以你需要手动添加它。这就是在 VB.NET 中要做的——注意，我已经指定了一个完整的路径。尽管 .NET 的帮助说我可以为密钥文件指定一个相对路径，但它看上去好象并不会很好的工作，所以，目前我指定了一个完整路径：

```
<Assembly: AssemblyKeyFile("../MySharedAssemblyVB\bin\keyPair.snk")>
```

最后一步是编译项目。这将使用位于强名称密钥文件中的强名称“标记”程序集。

安装程序集到全局程序集缓存

和私有程序集不同，你不能简单的拷贝程序集到全局程序集缓存——你必须安装它。有三种方式你可以选择：

- Microsoft Windows Installer 2.0（推荐用于运行的机器）
- .NET 命令行工具 GacUtil.exe（仅推荐用于开发者机器）。
- 程序集缓存查看器（仅用于开发者机器）

GacUtil 很容易使用。你仅需要在命令提示符下运行下面的命令：

```
gacutil -I <assembly name>
```

例如，你想安装一个名为 MySharedAssembly 的程序集到全局程序集缓存，你就需要运行下面的命令：

```
gacutil -I MySharedAssembly.dll
```

如果在开发的机器上安装了 .NET Framework SDK，那么程序集缓存查看器（在第 1 章“.NET 简介”的“在全局程序集缓存中查看程序集”一节有详细描述）是安装程序集到全局程序集缓存的最容易方式。你仅需要拖放强名称的 .NET 程序集到 Windows 资源管理器的全局程序集缓存，该程序集自动在全局程序集缓存中进行注册。

动态帮助

Visual Studio.NET 有一个很好的小的功能称为动态帮助，它“观察”你在 IDE 里的操作，并显示它认为对你有用的帮助主题。

要尝试动态帮助，你可以单击动态帮助窗口——默认情况下，它位于屏幕右下角，和属性窗口停靠在一起。如果窗口不可见，你仅需要执行 帮助|动态帮助菜单。下一步，在解决方案资源浏览器中单击你的解决方案，它会显示如图 17 的帮助主题。



图 17. 动态帮助窗口显示了有助于你在 IDE 里当前所选操作的帮助主题。

如果在解决方案资源管理器中已经打开了“`My First VB Windows App`”解决方案，请右击 `Form1.vb` 并选择“查看代码”菜单。尝试在 VB.NET 源文件中单击不同的单词，例如“`Class`”和“`Inherits`”。对 C# 的 `Class1.cs` 做同样的操作并单击“`using`”、“`namespace`”、“`public`”以及“`class`”。你也可以尝试单击 IDE 中的其他窗口——你会在动态帮助窗口看到恰当的帮助主题。

如果要自定义动态帮助，需要执行 **工具 | 选项** 菜单来打开选项对话框。动态帮助的设置位于“环境”项目下。

属性窗口

Visual Studio .NET 属性窗口(图 18)很像 Visual FoxPro 的属性窗口，然而，它们有很大的不同。在 VFP 中，你可以使用属性窗口查看/编辑一个对象的属性、事件和方法。在 Visual Basic .NET 中，你仅可以使用它查看/编辑属性。在 C# 中，你可以使用它查看/编辑属性和事件。

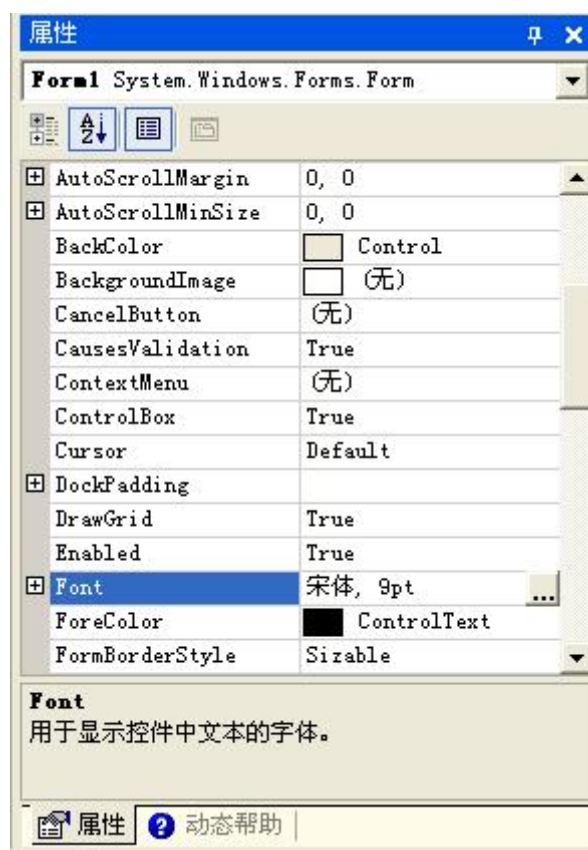


图 18. 你可以像这样按窗口字母进行排序，也可以按分类排序。

属性窗口顶部的组合框中显示了对象名和它基于的完整的类名（完整的命名空间和类名）。

和 Visual FoxPro 一样，如果你更改了一个属性的默认值，它就会在属性窗口中高亮显示。和 Visual FoxPro 不同的是，它不能仅仅显示被更改设置的属性。

如果一个属性有一系列值，你可以通过双击属性来在这些值中循环以选择合适的值。如果你选择了默认值，属性在属性窗口中会被标记为“未更改”。这和 VFP 是不同的（甚至更好），即使你输入一个默认属性值，它仍旧被标记为“非默认”。

属性窗口中的排序

在属性窗口中有两种排序方法——按字符排序或按分类排序。图 18 中显示了按字母排序的属性窗口（这是我的个人喜好）。

要按字母进行排序，可以单击“字母”按钮（属性窗口顶部从左至右查第二个按钮）。

当你在属性窗口中按分类进行排序时，相关的属性被分组到几个不同的分类里。例如，

当你在属性窗口中按分类进行排序时，看到的是可访问性、外观、行为、配置、数据、设计、焦点、布局、杂项和窗口样式。要按分类在属性窗口中进行排序，你需要单击“分类”按钮（属性窗口顶部从左至右查第一个按钮）。每个分类中的属性都是按字母顺序排序的。

显示事件

如果你是在 C# 中编辑一个对象，属性窗口就可以让你查看对象的事件（图 19）。

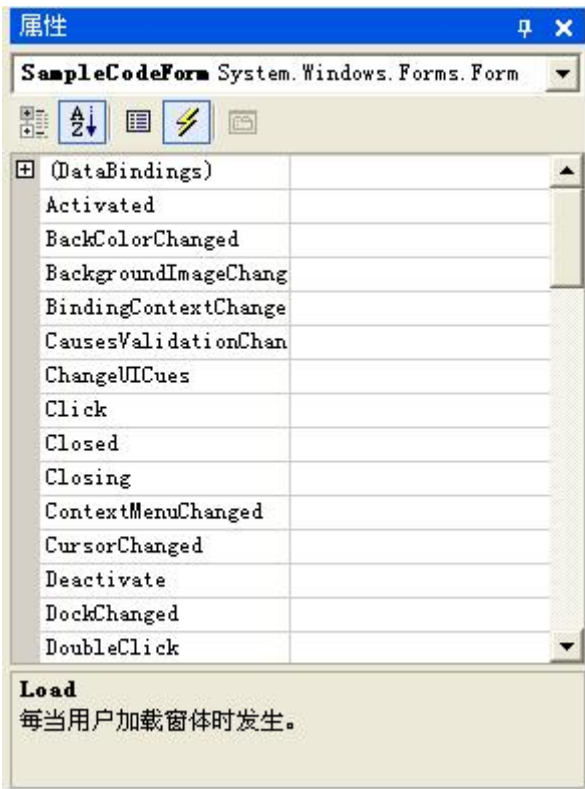


图 19. 如果你使用 C#，你也可以在属性窗口中查看一个对象的事件。

你可以通过单击“事件”按钮在属性窗口中查看事件（属性窗口顶部从左至右查第四个按钮——“闪电标识”）。要查看属性，单击“属性”按钮（从右边查第三个）。

在 Visual Basic .NET 中，尽管你不能在属性窗口中查看事件，但是你可以在代码编辑窗口顶部的组合框中达到相同的目的。如果你在坐侧的组合框中选择了 (Base Class Events)（图 20），一系列和其对应的事件将会显示在右侧的组合框中（图 21）。

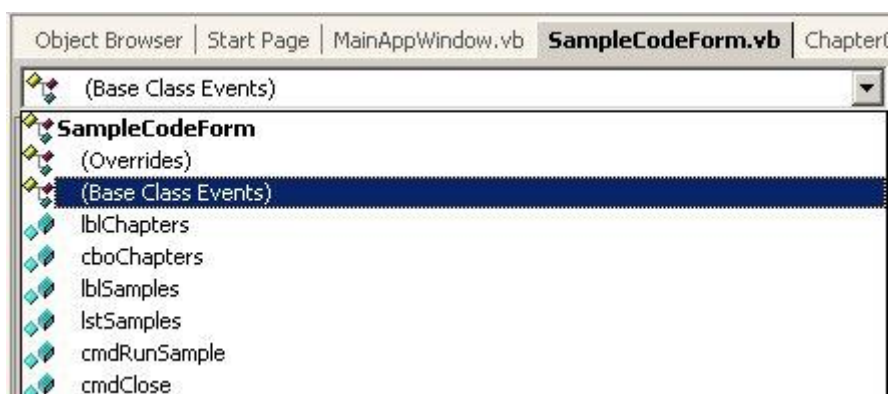


图 20. 要查看 VB.NET 中的事件，从代码编辑窗口顶部左侧的组合框中选择 (Base Class Events)。

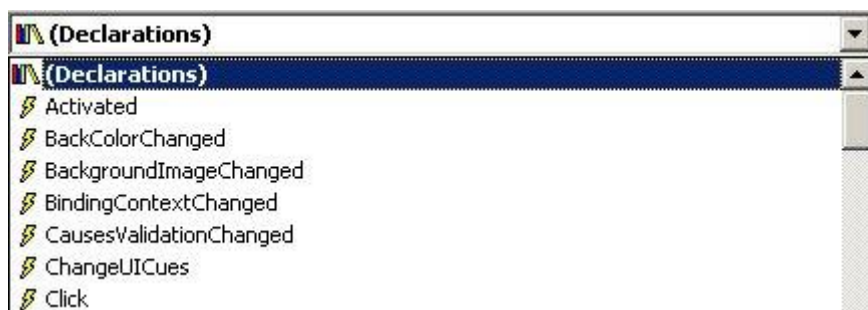


图 21. 在 VB .NET 中，当你从组合框中选择(Base Class Events)时，和对应的事件会显示在代码窗口顶部右侧的组合框中。

关于事件的更多信息，参看第 5 章“C# 和 Visual Basic .NET 中的面向对象”。

代码编辑

要显示在 Visual Studio .NET 中代码编辑的能力，你需要在解决方案资源管理器中右击“`My CSharp Class Library`”项目的 `Class1.cs` 文件，选择“查看代码”菜单。这样会在 IDE 里打开一个代码编辑窗口（图 22）。



图 22. 代码编辑窗口允许你很容易的选择要编辑的类和方法，并且还提供像 Visual FoxPro 那样的语法着色。

在代码编辑窗口顶部左边是一个可以显示所有源代码文件中类列表的组合框。右侧组合框是当前所选类的元数据。当你在源代码中进行翻页时，组合框会自动显示当前类和光标所位于的元数据。如果你在这些组合框中选择一个项目，光标就会被放置在源代码中合适的位置。

智能感应

如果你很熟悉 Visual FoxPro 7 及以后版本中的智能感应，你会觉得在 Visual Studio.NET 中使用它很舒服。智能感应以元数据列表、快速信息、完成单词和自动括号匹配 (brace matching) 形式来工作。

元数据列表

当你输入类名或结构名再输入一个句点时，智能感应在一个可滚动的列表中显示所有有效的元数据，你可以在这个列表中选择你使用的元数据。当你选择的元数据在列表中高亮显示时，你可以用几种不同的方法将它插入到代码中。首先，你可以下面的字符，例如一个逗号、空格或圆括号。或者，你可以按 TAB 键、Ctrl+Enter、Enter 或双击高亮项目。

要关闭元数据列表，你只需要按 Escape 键。一个很容易重新显示元数据列表的方式是按 Ctrl+J 。

参数信息

当你在一个方法（function）后键入一个左括号时，VS.NET 会在一个弹出窗口中显示参数列表，当前显示的参数是高亮显示的。如果一个方法（method）是重载的（overloaded），它会显示所有可用的方法和他们的相关参数，你可以通过使用 UP 和 Down 方向键来滚动它。关于重载方法的解释，请参看第 5 章“C# 和 Visual Basic .NET 中的面向对象”。

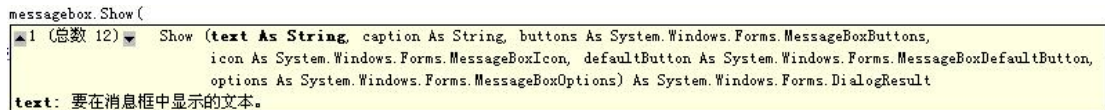


图 23. 如果一个方法是重载的，智能感应会显示一个可用方法的列表以及它们相关的参数。

要关闭参数列表，可以按 Escape 键。一个很容易重新显示参数列表的方式是按 Ctrl+Shift+Space 键。

快速信息

当你将鼠标悬停在代码中的任意标识符上时，VS.NET 就会在一个黄色的弹出式对话框显示标识符的完整声明。

如果要以手动方式来显示弹出式快速信息，你需要按 Ctrl+K，然后按 Ctrl+I 。

完成单词

VS.NET 可以自动完成变量、命令或方法名，当然，你需要输入足够多的字符来使自己的名称和其他的有所区别。要使用单词的自动完成，首先要输入变量、命令或方法名的前几个字母，然后按 Alt+RightArrow。如果你输入的字符不能唯一的标识一个单词，VS.NET 会显示一个滚动列表以供你选择。

例如，如果你在代码窗口键入“Oper”并按 Alt+RightArrow，VS.NET 就会在代码中插入“OperatingSystem”。

自动括号匹配 (Brace matching)

当你键入一个结束括号时，在开始和结束括号间的部分都会高亮显示，直到你敲击其他的键或移动光标。自动括号匹配包括 ()、[]、{ } 以及 < >。

如果你使用 C#，你可以将光标放置在括号间的任意部分，然后按 Ctrl+]，这样就可以匹配的括号间来回的移动光标。这个特性在 VB.NET 中是无效的。

Visual Basic .NET—特殊的智能感应

当你使用 Visual Basic .NET 时，你可以使用附加的智能感应特性，例如完成关键字 goto、Implements、Option 和 Declare。你也可以完成 Enum 和 Boolean，以及语法提示。

编辑智能感应

默认情况下，所有的智能感应都会出现在 IDE 的某个地方。然而，如果你想关闭自动的智能感应并在需要时手动管理它，你可以遵循下面的步骤来设置智能感应选项。

要关闭自动智能感应，请执行 **工具|选项** 菜单。展开“文本编辑器”并选择你想自定义的语言。然后选择语言下的“常规”，针对你想关闭的智能感应选项清除对应的选择框。

大纲显示 (Outlining)

Visual Studio .NET 的代码编辑器允许你依赖被称之为大纲显示 (outlining) 的特性来隐藏和显示代码。在某些时候，VS .NET 的代码编辑器被称为 **可折叠编辑器** (folding editor)。

如果你查看代码编辑窗口(图 24)，你会看到 VS .NET 自动增加一个减号(-)到代码块的起始位置，例如命名空间、类定义、方法定义。当你单击减号时，在你单击的减号和它下一个减号间的代码会被隐藏，并且会在下一个代码块的前面显示一个加号，并且在代码行末尾显示一个“...”的方框(图 24)。然后使用一个可视的指示器来显示所隐藏的代码。

特别好的一个地方是当你将鼠标放置在包含“...”的方框上时，它会显示一个包含所折叠的代码的快速信息。

要展开已经被折叠的代码，你或者单击加号，或者双击包含“...”的方框。



图 24. Visual Studio .NET 的 outlining 特性允许你隐藏代码以便更容易的只去查看你所想查看的代码。

除了 VS .NET 的自动大纲显示外，你也可以创建你自己的大纲显示。要达到这个目的地，你只需要选择你想设置大纲显示的文件，右击所选文本，并选择 **大纲显示|隐藏选定内容** 菜单

在“大纲显示”快捷菜单中共有六个可用选项：

- **隐藏选定内容** - 隐藏当前选定的文本。在 C# 中，只有当自动大纲停止被关闭或“停止大纲显示”被选择后，才能看到此选项。
- **切换大纲显示展开** - 反转被选定代码的当前折叠或展开状态，或者使当前光标所在小节处于大纲显示状态。
- **切换所有大纲显示** - 设置文档中所有代码大纲显示为同一状态。
- **停止大纲显示** - 关闭整个文档的大纲显示。
- **停止隐藏当前区域** - 从当前所选的用户定义区域移除大纲显示信息。在 C# 中，只有当自动大纲停止被关闭或“停止大纲显示”被选择后，才能看到此选项。
- **折叠到定义** - 创建文档中所有过程的大纲显示区域并折叠它们。

书签

你可以像在 Visual FoxPro 中几乎一样的方式为你的代码做书签标记，在 Visual FoxPro 中，当你退出 VFP 并再次打开 VFP 时，所标记的书签就会消失，而在 Visual

Studio .NET 中则不会有这种情况发生。

要创建一个书签，你需要将光标放到某一行代码（或者选择一行或多行代码），然后执行 **编辑|书签|切换书签** 菜单。你也可以按两次 Ctrl+K 来切换书签。

要在文档中移动到下一个书签，你需要按 Ctrl+K ，然后按 Ctrl+N 。要移动到前面的书签，你需要按 Ctrl+K ，然后按 Ctrl+P。要清除文档中所有的书签，你需要按 Ctrl+K ，然后按 Ctrl+L 。这些热键功能和 **编辑|书签** 下的菜单项是等效的。

格式化文本

编辑|高级 菜单提供了一些格式化文本的选项。**表 2** 是这些选项的简要描述。

表 2. 有多种格式化文本选项可用，有一些选项在 Visual FoxPro 中不存在。

项目模板	描述	快捷键
编排文档格式	对输入的文档智能应用整套规则。仅对 VB.NET 源代码有效	Ctrl+K, Ctrl+D
格式化选定内容	对所选代码智能应用整套规则	Ctrl+K, Ctrl+F
制表符替换空格	转换空格为制表符	–
空格替换制表符	转换制表符为空格	–
转换成大写	所选文本全部大写。它和“转换为小写”在区分大小写的语言中使用时（例如 C#）要小心慎重	Ctrl+Shift+U
转换成小写	所选文本全部小写	Ctrl+U
删除水平空白	在所选文本中删除水平空格。它并不会删除空行	Ctrl+K, Ctrl+\
查看空白	在当前文档中以可见的方式显示空格。制表符以箭头表示，空格以小圆点表示	Ctrl+R, Ctrl+W
自动换行	在当前文档中打开自动换行功能	Ctrl+R, Ctrl+R
渐进式搜索	在当前文档中打开渐进搜索功能。依据你输入的字符，光标会移动到第一个匹配的单词	Ctrl+I
注释选定内容	注释所选文本	Ctrl+K, Ctrl+C
取消注释选定内容	对所选文本取消注释	Ctrl+K, Ctrl+U
增加行缩进	所选文本增加缩进	–
减少行缩进	所选文本减少缩进	–

默认情况下，减少行缩进、增加行缩进、注释选定内容、取消注释选定内容也在文本编辑器工具栏中提供(图 25)。



图 25. 文本编辑器工具栏中的减少行缩进、增加行缩进、注释选定内容、取消注释选定内容

Visual Studio .NET 窗口

Visual Studio .NET 在自定义 IDE 窗口上提供了很大的弹性，你可以选择最适合你自己的方式——包括自动隐藏、可停靠、窗口拆分、选项卡式和环绕 IDE 的窗口拖动。我建

议你都尝试下，以确定哪种方式最适合你。这一节提供了这些特性的一个快速预览。更多信息请参看 Visual Studio .NET 的帮助文件。

在 Visual Studio .NET 中存在两种主要的窗口类型——工具窗口和文档窗口。

工具窗口

在 VS.NET 的 **视图** 菜单可以找到工具窗口，它们包括解决方案资源管理器、类视图、服务器资源管理器、资源视图、属性窗口、工具箱、Web 浏览器、宏资源管理器、对象浏览器、文档大纲、任务列表、命令窗口、输出窗口、查找结果窗口以及收藏夹。它们都可以自动隐藏、停靠、浮动、以选项卡形式放在其他工具窗口上，甚至可显示在另外的显示器中。要设置一个工具窗口的运行方式，右击窗口并从快捷菜单中选择恰当的设置。

这里提到的每个工具窗口在本章的以后部分会予以讨论。

在其他显示器中显示工具窗口

如果你的计算机有多于一个显示器，那么，你就可以简单的拖放一个工具窗口到另外的显示器中。

文档窗口

当你在解决方案中打开项目时，文档窗口就会自动被创建。你可以选择文档窗口工作在选项卡式文档（默认情况下），也可以作为 MDI 模式（多文档界面）。你可以通过执行 **工具 | 选项** 菜单，在选项对话框的“环境”中的“常规”选项下，更改文档窗口的模式。在窗体的顶部列出了文档窗口可设置的模式。

拆分窗口

如果你需要同时查看文档的两个部分，你而可以对窗口进行拆分(图 26)。

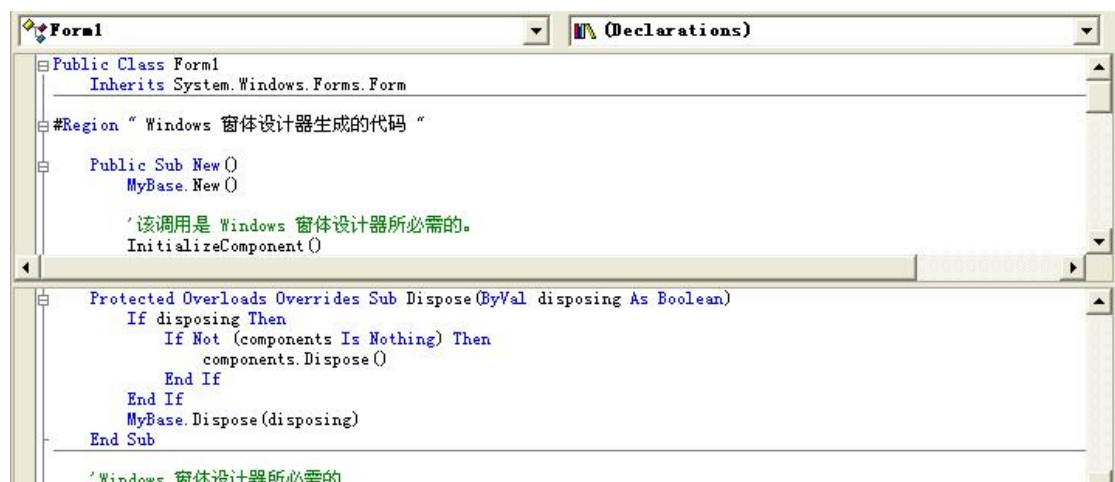


图 26. 如果你需要同时查看文档中的两个部分的代码，你可以拆分窗口。

要拆分文档窗口，你需要选择 **窗口|拆分** 菜单。要取消拆分，你需要选择 **窗口|移除拆分** 菜单。

全屏模式

你可以通过选择 **视图|全屏** 来在全屏模式下查看文档窗口。这会临时隐藏其他所有工具窗口并将当前所选择的文档在全屏模式显示。

要返回到常规的查看模式，你只需要点击漂浮的全屏工具栏（当进入全屏模式时它会自动显示出来）。

查找和替换

Visual Studio .NET 具有强大的搜索和替换能力。如果你选择了 **编辑|查找和替换** 菜单，你会看到有五个选项：

- 查找
- 替换
- 在文件中查找
- 在文件中替换
- 查找符号

查找和替换

无论你选择 **查找** 还是 **替换** 菜单，你都会看到同一个对话框。如果你打开的是查找对话框，当你单击“替换”按钮后，查找对话框就会变为替换对话框（图 27）。运行查找对话框的快捷键是 Ctrl+F，运行替换对话框的快捷键是 Ctrl+H。



图 27. 替换对话框为搜索和替换文本提供了多种选项。

有一系列的选择框和选项组来控制如何执行搜索。你或许不熟悉“搜索隐藏文本”。如果你选择了它，VS.NET 就会搜索隐藏的文本，例如设计时刻控件的元数据，或者一个大纲显示的文档中隐藏/折叠的区域。

搜索选项组允许你缩小搜索范围到当前文档。你也可以扩展搜索范围到所有打开的文档。

如果你运行查找或替换对话框时，在一行中有一个或多个被选择的单词，那么它们会自动显示在“查找内容”中。

在你选择或输入搜索条件后，使用“查找下一个”、“替换”、“全部替换”和“标记全部”就可以进行搜索或随意的替换。“标记全部”会在包含特定文本的代码行前增加一个书签。你可以在书签间浏览所找到的每个实例。具体请参看本章前面的小节“书签”。

在文件中查找和替换

选择 **在文件中查找** 和 **在文件中替换** 也会得到相同的界面。如果你单击在文件中查找对话框中的“替换”按钮，它会变为在文件中替换对话框（图 28）。运行在文件中查找的快捷键是 Ctrl+Shift+F，运行在文件中替换对话框的快捷键是 Ctrl+Shift+H。



图 28. 在文件中替换对话框允许你将它作为替换对话框在同一位置搜索文本，你也可以当前项目的搜索文件、一个特定文件或是一个特定的文件夹。

在对话框中存在一系列的选项来决定如何进行搜索。一个令人感兴趣的选项是“全部替换后保持将已修改的文件打开”。这样就可以打开文件以便确认更改，并可有选择性的撤消它们。

“查找范围”组合框允许你指定在哪里搜索文本。你可以当它是替换对话框在同一位置搜索文本，但是，你也可以在当前项目中搜索所有文件。此外，你可以手动键入你想搜索的文件名或路径。“在子文件中查找”选择框指定搜索位置包含子文件夹。你也可以指定要搜索的文件类型。默认情况下是所有文件 (*.*)。

在执行查找后，搜索的结果会显示在一个查找结果窗口（图 29）。如果你双击任意一个搜索结果，它会自动打开相关的文件并将光标放置在包含搜索文本的行上。



图 29. 查找结果窗口显示了搜索条件和搜索结果。

“显示在查找结果 2”选择框指定搜索的结果显示在另外打开的“查找结果 2”窗口。这允许你查看多个搜索集。

查找符号

查找符号对话框(图 30)允许你搜索符号，它位于对象(类、结构、接口、命名空间等等)的上下文以及它们的元数据中。这个选项允许你缩小搜索范围，以便搜索范围不包含注释和方法。运行查找符号对话框的快捷键是 Alt+F12 。



图 30. 查找符号对话框允许你在对象和它们的元数据中搜索指定文本。

“查找范围” 组合框允许你指定是在“活动项目” 中搜索还是在“选定的组件” 中进行搜索。如果你指定在“选定的组件” 中搜索，并单击浏览按钮 (...)，将运行选定的组件对话框 (图 31)，这里你可以指定要搜索的组件。



图 31. 选定的组件对话框允许你指定针对指定的符号在哪个组件中进行搜索。

当你执行查找时，结果会显示在查找符号结果窗口（**图 32**）。结果列表包含文件名、行号和你所搜索的字符的位置。

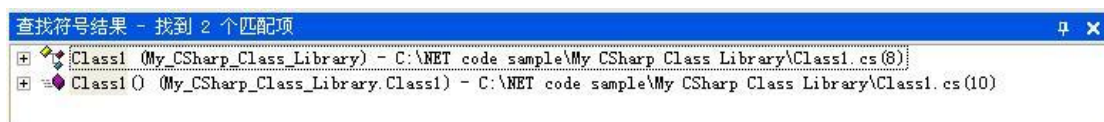


图 32. 查找符号结果窗口显示了文件名、行号和所搜索的字符的位置。

查找符号的另一个很酷的特性就是你可以搜索 .NET 类、接口等等。如果是那样，搜索结果会显示在 VS.NET 的对象浏览器中。关于对象浏览器的更多信息，请参看本章后面的“对象浏览器”一节。

设置 IDE 选项

Visual Studio .NET IDE 具有高度的灵活性。选项对话框（**图 33**）允许你更改 IDE 下面这些的表现：

- 环境
- 源代码管理
- 文本编辑器
- Analyzer
- 数据库工具
- 调试
- HTML 设计器
- 项目
- Windows 窗体设计器
- XML 设计器

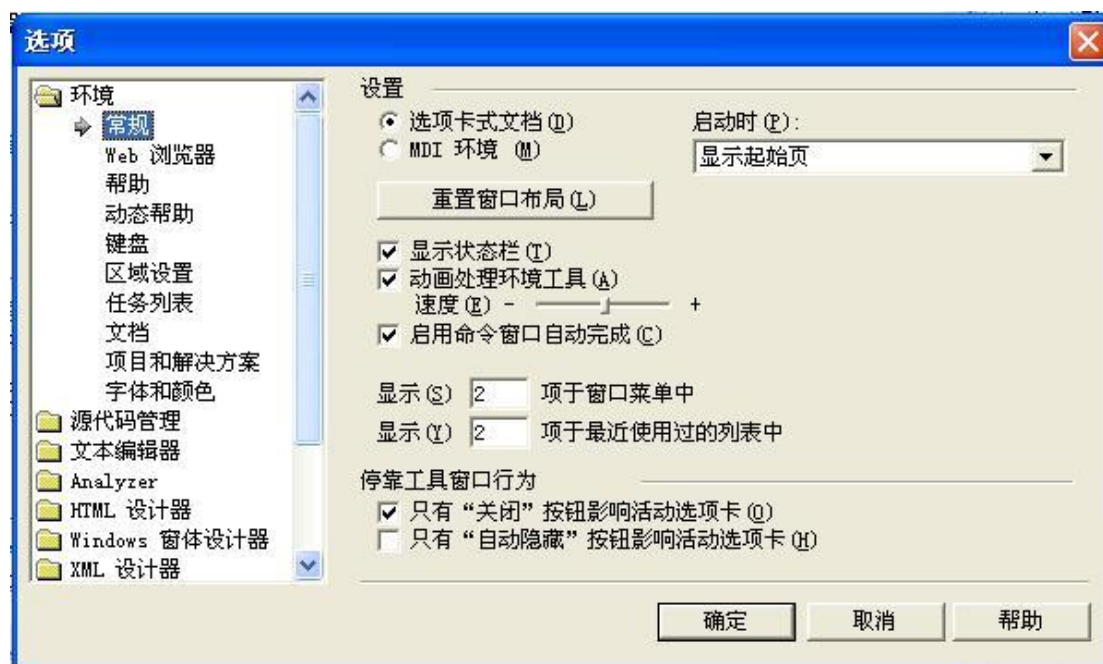


图 33. 请花一些时间来查看选项对话框中的所有设置，它允许你定制 IDE 为你自己最适合的工作方式。

要运行选项对话框，请执行 **工具|选项** 菜单。对于每个设置的详细情况，请先选择分类，然后单击“帮助”按钮。

对象浏览器

Visual Studio .NET 的对象浏览器（图 34）和 Visual FoxPro 的对象浏览器在功能上很相似。它允许你查看 .NET 组件和 COM 组件。你可以通过执行 **视图|其他窗口|对象浏览器** 菜单来运行它，或者使用快捷键 **Ctrl+Alt+J**。

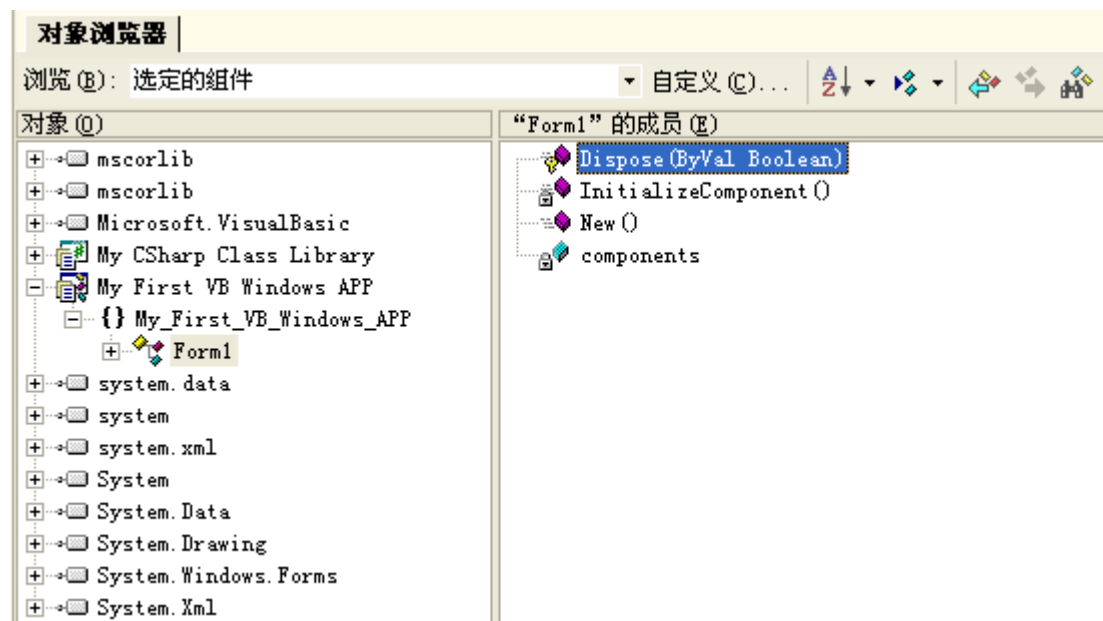


图 34.对象浏览器可用来查看 .NET 和 COM 组件。

当你运行对象浏览器时，它会列出当前解决方案中的所有项目，以及在这些项目中引用的所有组件。你可以在左侧面板中的树视图中深入的了解项目和组件。一旦你选择了一个对象，它的元数据就会显示在右侧的元数据面板中。如果你单击一个元数据，那么在底部描述面板中就会显示该元数据的描述。

在对象浏览器中使用了很多的图标。它们的描述，请参看帮助文件中“类视图和对象浏览器图标”主题。

对象浏览器具有一项很有用的功能，当你右击一个自定义类并选择“转到定义”菜单时，如果源代码可用，它就可以自动的打开包含类的文件并将光标定位到类定义开始的位置。

类视图窗口

类视图窗口（图 35）很类似于对象浏览器，但是它仅仅用于查看解决方案中的符号而不是外部组件。此外，它提供了符号的分层查看（按项目和命名空间）。要执行类视图，你需要执行 **视图|类视图** 菜单，或使用快捷键 **Ctrl+Shift+C**。

如果你在类视图中双击类或类成员，它会自动打开相关的源代码文件（如果它还没有被打开），并将光标置于对应的代码上。这是它工作的一个方式。如果你在代码编辑窗口，右击一个类或类成员，并选择“同步类视图”，那么类视图窗口会得到焦点，并高亮显示你所选的项目。



图 35. 类视图按照项目和命名空间的分类显示解决方案中的符号。

C# 类视图的特性

如果你右击的是一个 C# 类，你会看到一个额外的选项，它在 VB.NET 中是不可见的。那是让人很感兴趣的“添加”选项（图 36），它允许你在类视图窗口中向类添加方法、属性、字段或者索引器。（译者注：在 VS2005 和 VS2008 中，这种操作是在类关系图中进行。）



图 36. C# 有一个特殊的功能，它允许你在类视图窗口中直接增加方法、属性、字段或索引器。

关于方法、属性、字段和索引器的更多信息，请参看第 5 章“C# 和 Visual Basic.NET 中的面向对象”。

任务列表

VS.NET 的任务列表（图 37）非常类似于 Visual FoxPro 的任务列表。它是一个可以帮助你管理项目任务的工具。任务列表通常位于 VS.NET IDE 的底部。如果你没有看到它，请执行 **视图|其他窗口|任务列表** 菜单，或者按快捷键 **Ctrl+Alt+K**。



图 37. 任务列表包含 VS.NET 自动增加的任务和你自己输入的任务。

VS.NET 会向任务列表自动增加条目。例如，当你在项目中增加一个新类并且 VS.NET 创建一个构造器方法（它类似于 Visual FoxPro 类的 **Init** 方法）时，它会被增加一条 **TODO** 项目来告诉你要在构造器方法中添加代码。

你也可以通过单击窗口顶部手动输入任务描述。此外，你还能通过右击代码行，从快捷菜单中向其添加任务项目。

命令窗口

如果你认为 Visual Studio .NET 的命令窗口也和 Visual FoxPro 的命令窗口相似，你会大失所望！你可以通过执行 **视图|其他窗口|命令窗口** 菜单或使用快捷键 **Ctrl+Alt+A** 来运行它。关于命令窗口的更多信息，请参看第 13 章“**.NET 中的错误处理和调试**”。

收藏夹

收藏夹窗口用来在 VS.NET IDE 中显示 IE 浏览器的收藏夹（图 38）。你可以执行 **视图|其他窗口|收藏夹** 菜单或按快捷键 **Ctrl+Alt+F** 来打开收藏夹窗口。如果你在收藏夹窗口上右击某一项目，你会看到和 IE 中同样的快捷菜单，它允许你删除收藏夹、重命名收藏夹、创建新收藏夹等等。

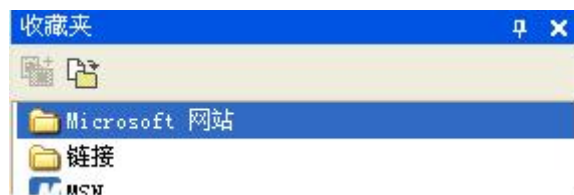


图 38. 收藏夹窗口用来在 VS.NET IDE 中显示 IE 浏览器收藏夹。

工具箱

默认情况下，VS.NET 工具箱在 IDE 中是自动隐藏的。要显示工具箱，你只需要移动鼠标到桌面左侧的工具箱图标上（图 39）。



图 39. 移动鼠标到工具箱图标以便自动显示工具箱窗口。

如果你没有看到这个图标，你可以执行 **视图 | 工具箱** 菜单来打开工具箱。

工具箱包根据你在当前 IDE 中的选择含了几个不同的标签来显示多个项目。例如，如果你选择了一个 Windows 窗体，它会显示 Windows 窗体标签，其中包含了很多用户界面控件，你可以见它们添加到你的窗体中（图 40）。

在本书的其他各章，你会看到其他标签是如何使用的。例如，第 9 章“生成 .NET WinForm 应用程序”会详细描述如何使用 Windows 窗体标签。

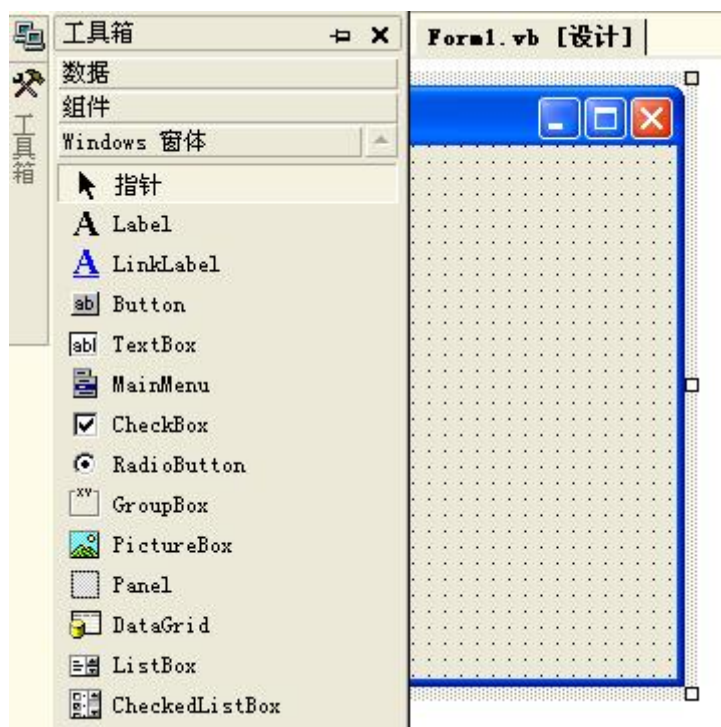


图 40. 工具箱窗口依赖你在 IDE 里的选择来显示多个项目。

服务器资源管理器

服务器资源管理器是 Visual Studio .NET 中一个很好的特性。它允许你打开数据连接、登录服务器并使用数据库开始工作。你可以使用 SQL Server 和 Oracle ——也可以通过服务器资源管理器来使用 MSDE(Microsoft Database Engine)!

默认情况下，服务器资源管理器自动隐藏在 VS.NET IDE 的左侧。你可以将鼠标移动到服务器资源管理器标签上，或执行 **视图|服务器资源管理器** 菜单来使它可见。

服务器资源管理器树视图包含两个主要节点——数据连接和服务器(图 41)。

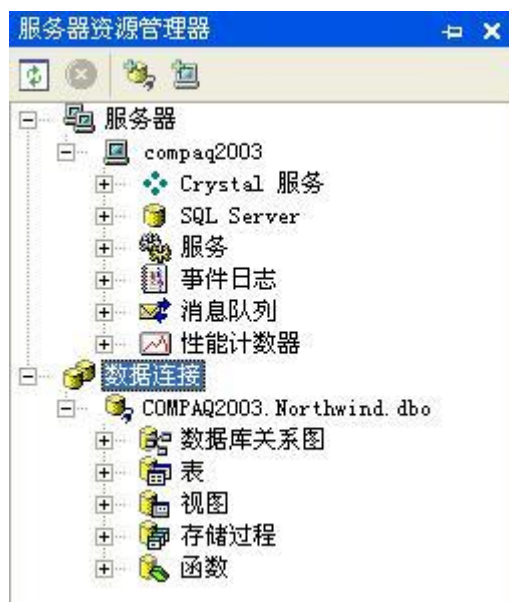


图 41. 服务器资源管理器允许你创建到服务器的连接、查看数据库、事件日志、消息队列、性能计数器和系统服务。

数据连接

你可以通过右击“数据连接”节点并选择“添加连接...”快捷菜单来增加网络所能访问的任意数据库的连接。这个操作会执行“数据链接属性”对话框。你可以在“提供程序”标签选择一个所提供的程序，然后在“连接”、“高级”、“所有”标签设置属性。如果你需要设置连接的帮助，请点击“帮助”按钮。

一旦你创建了一个数据的连接，你就可以无须离开 Visual Studio .NET IDE 来查看和控制数据。例如，如果你创建了一个到 SQL Server Northwind 数据库的连接，那么展开连接下的“表”节点后，你就会看到数据库中所有表的列表。如果你右击一个表，你会看到“从表中检索数据”、“设计表”、“新建表”等选项。

图 42 显示了右击 Employees 表并选择“从表中检索数据”菜单后的结果。

起始页 Class1.cs Form1.vb Form1.vb [设计]				dbo.Employee...Northwind)		
	EmployeeID	LastName	FirstName	Title	TitleOfCourtesy	BirthDate
▶	1	Davolio	Nancy	Sales Represent	Ms.	1948
	2	Fuller	Andrew	Vice President,	Dr.	1952
	3	Leverling	Janet	Sales Represent	Ms.	1963
	4	Peacock	Margaret	Sales Represent	Mrs.	1937
	5	Buchanan	Steven	Sales Manager	Mr.	1955
	6	Suyama	Michael	Sales Represent	Mr.	1963
	7	King	Robert	Sales Represent	Mr.	1960
	8	Callahan	Laura	Inside Sales Co	Ms.	1958
	9	Dodsworth	Anne	Sales Represent	Ms.	1966
*						

图 42. 如果你使用服务器资源管理器来检索数据，那么结果将显示在 VS.NET IDE 中。

当结果标签获得焦点时，查询工具栏会自动显示出来，它包含了诸如“运行查询”、“验证 SQL 语法”、“降序排序”、“移除筛选器”等按钮。

服务器

服务器资源管理器的服务器节点显示了当前可用的服务器列表。在每个服务器节点下是 Crystal 服务(Crystal 报表选项)、“事件日志”、“消息队列”、“性能计数器”和“SQL Server”。

关于服务器资源管理器的更多信息，请查阅 Visual Studio.NET 的帮助文件中“服务器资源管理器简介”主题。

源代码管理

如你所料，微软综合了 Visual Studio .NET 和 Visual SourceSafe 来提供源代码控制的能力。Visual SourceSafe 6.0c 是一个“再发行”软件，就微软而言，它是随 Visual Studio 6.0 SP5（不久之前发布）一起发售。（译者注：VS2003 提供的是 Visual SourceSafe 6.0d）。此外，6.0c 修复了一些 bug，提高了速度，并和 VS.NET IDE 融于一体。

你可以将一个项目或整个解决方案添加到源代码管理中。要增加一个项目到源代码管理：

- 在解决方案资源管理器中选择一个项目
- 执行 文件|源代码管理|将选定项目增加到源代码管理... 菜单

与 Visual FoxPro 相同，当你将一个项目增加到源代码管理后，项目条目前的图标就会表示它们的源代码控制状态，例如“签入”和“签出（图 43）。



图 43. 你可以将单独的项目或整个解决方案添加到源代码控制里。

要增加整个解决方案到源代码控制：

- ◆ 在解决方案资源管理器中右击解决方案，然后选择“将解决方案添加到源代码管理...”菜单。
 - 如果出现提示，登录到源代码管理
 - 当“ADD to SourceSafe Project”对话框显示时，它显示了你的解决方案名称，选择（或创建）你想存储项目的目录，然后单击 OK 按钮

关于 VS.NET 和 Visual SourceSafe 的详细可用资料，请访问站点：

http://msdn.microsoft.com/library/?url=/library/en-us/dnbda/html/tdlg_rm.asp?frame=true

或者参看 .NET 帮助文件“Source Control Basics”主题，它包含了其他的源代码控制主题的链接。关于 Visual SourceSafe 工作的更多信息，可以查阅由 Hentzenwerke 出版社出版，Ted Roche 所著的《Essential SourceSafe》一书。这本书已增加了关于 Visual Studio .NET 和 SourceSafe 6.0c 的特定内容，预定该书请访问 Hentzenwerke 站点。

宏资源管理器

宏提供了自动重复操作的能力，或者在开发期间提供了一些按键的操作。Visual Studio.NET 将创建和管理宏带入到一个新的层次。宏资源管理器显示了针对一个解决方案的所有可用宏。在 VS.NET 中，宏是被存储在宏项目中的，它可以被其他多个解决方案所共享。宏只能由 Visual Basic .NET 所创建——其他 .NET 语言无法创建宏。

要查看宏资源管理器（图 44），请选择 **视图|其他窗口|宏资源管理器** 菜单。宏资源管理器允许你创建新的宏项目，载入已有宏项目或者卸载宏项目。



图 44. 宏资源管理器用来创建和管理 Visual Studio .NET 宏。

宏 IDE

VS.NET 包含一个特殊的宏 IDE，它允许你创建、管理并运行宏（图 45）。这个 IDE 包含自己的项目资源管理器、类视图、属性窗口、工具箱、Web Browser、动态帮助、调试器等窗口。

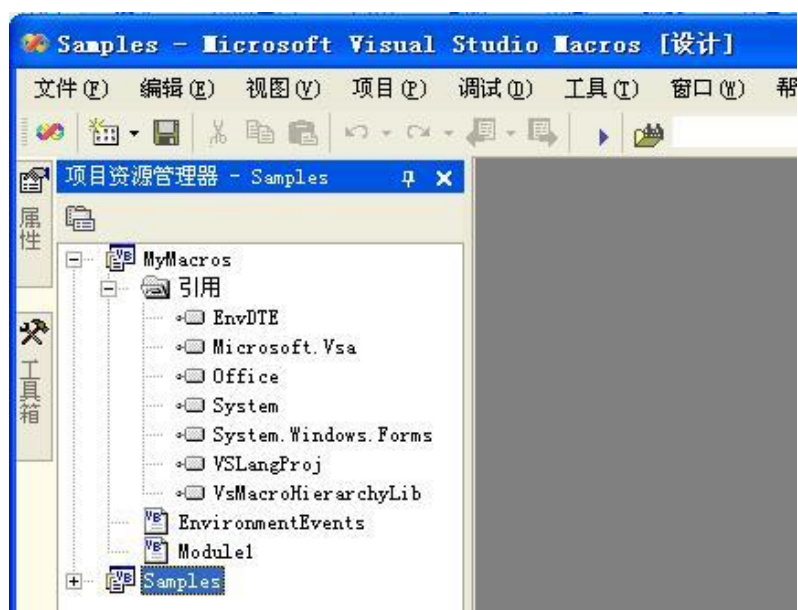


图 45. VS .NET 拥有一个单独的宏 IDE，你可以使用它创建和管理宏。

要运行宏 IDE，请右击宏资源管理器顶部的“宏”节点，并选择“宏 IDE...”快捷菜单。关于创建、运行和管理宏的细节，请参看 .NET 帮助文件中的“管理宏”主题。

总结

本章快速的浏览了 Visual Studio.NET 的许多特性。你可以看到微软已经投入了大量的努力使 VS.NET 成为一个强大的软件开发环境。然而，如果要覆盖 VS.NET 的所有特性，本章篇幅可能要增加一倍。我建议你花一些时间浏览 VS.NET 帮助，以便学习到更多有用的知识。

第三章 C# 入门

一些开发者基于 C 语言高级、难以理解和难于使用的声誉“害怕”C#。本章的目的就是要打消这种疑虑，并将 C# 和 Visual FoxPro 加以比较——你会惊讶于它们的相似。它还将帮助里利用 Visual FoxPro 的已有知识来学习 C#。

在微软最新的编程语言 C# 中有很多令人兴奋的事。它是一个干净、优雅的面向对象的 .NET 语言。然而，新的并不意味着它是没有经过考验的或“少不更事”的。微软放出 C# 进行测试并使用它书写了 .NET Framework 的基类（根据记录，.NET Framework 或许已经用 Visual Basic .NET 重新书写。只所以使用 C# 来写是因为许多使用 .NET Framework 的开发者是 C++ 开发者，并且，C# 编译器是在 VB.NET 编译器之前被编译的）。

在现有的编程语言中，C# 是最像 Java 的编程语言。2000 年 10 月 Java™ 的报告指出：“C# 是简单继承的 Java。具备自动内存管理，43 个操作中有 41 个是相同的，完全一样的控制流语句、超级 foreach，在 50 个关键字中有 38 个是相同的，如果你允许同义词，那么在 50 个关键字中有 46 个是相同的”。

尽管 C# 是一门新语言，它仍旧有许多功能和其他许多成熟的语言有联系，例如 Visual FoxPro。在这一章，你将从学习弱类型和强类型的对比开始，对变量、常数、数据类型、条件语句、控制流语句、跳转语句以及数据类型的转换予以学习。

尽管在本章将涉及 C# 中面向对象的基本基本要素，例如定义一个简单的类，完整的 C# 和 VB.NET 中的面向对象内容却在第 5 章“C# 和 Visual Basic .NET 中的面向对象”中。



C#、Visual Basic .NET 以及 Visual FoxPro 的操作和关键字的比较，请查看“附录 A—语言对照表”。每种语言的数据类型比较，请参看“附录 B—数据类型的比较”。

弱类型 vs 强类型

在 C# 和 Visual FoxPro 之间最大的区别就是弱类型和强类型。Visual FoxPro 是一个弱类型语言。这意味着你可以将任意类型的值存储到一个变量中。例如，像下面代码那样，你可以存储一个日期时间型的值到一个变量，也可以将字符型的值存储到同一变量中，然后

还可以将完全不同的一个整数仍旧存储在该变量中，Visual FoxPro 的编译器不会有什么抱怨：

```
LOCAL MyVariable  
MyVariable = DATETIME()  
MyVariable = "Hello World!"  
MyVariable = 6
```

在这种难以置信的灵活中，它有自己的缺点。首先，从易读性的角度来看，最好是创建三个变量来存储每一种类型的值，而不是重复使用一个。其次，更重要的是，弱类型允许 bug 偷偷的躲藏起来。例如，如果你存储一个整数到变量 MyVariable，然后试图运行下面的代码，你会在运行时刻得到一个“操作符/操作数类型不匹配”的错误：

```
REPLACE CustomerName WITH "Customer" + MyVariable IN Customer
```

当你运行代码时你也许很快就会发现这个 bug——但问题是——“你在运行代码的时候”。如果直到它到最终用户手中这段代码如果从没运行过怎么办？

相比之下，C# 是强类型语言（VB.NET 也是强类型语言，但是 C# 是更强类型的。详细资料参看第 4 章“Visual Basic.NET 入门”）。在变量的世界里，强类型意味着你必须定义所有变量的类型。例如，下面的代码创建了名为 MyDate 的日期变量、名为 MyString 的字符变量和一个名为 MyInteger 的整数变量：

```
DateTime MyDate;  
string MyString;  
int MyInteger;
```

如果你没有定义变量类型，C# 编译器会告诉你一个错误，并在编译完成你的应用程序前退出。

另一个在 Visual FoxPro 代码中容易引起 bug 的常规错误是，你在使用一个变量前没有事先定义。例如，下面的代码在使用 MyVariable 前没有事先定义：

```
MessageBox("嗨，你好!" + MyVariable, 0+48, "弱类型示例")
```

Visual FoxPro 编译器并不能捕获这种类型的错误。然而，当这个代码在运行时刻执行时，你的用户就会看到一个“变量‘MyVariable’没有发现”的消息。

相比之下，C# 在你没有定义和初始化之前是不允许你使用它的。例如，如果你创建了下面的代码：

```
string MyString;  
MessageBox.Show(MyString, "强类型示例");
```

编译器会显示错误“使用未分配的局部变量‘MyString’”。

要更改这些代码以便恰当的工作，你需要为变量 `MyString` 分配一个初始值：

```
string MyString = "我爱强类型！";  
MessageBox.Show(MyString, "强类型示例");
```

C# 的强类型也应用于存储对象引用的变量。这方面的内容在第 5 章“C# 和 VB.NET 中的面向对象”。

在编码过程中的实际价值为——在 Visual FoxPro 中，当我写了一整页的代码时，变量会存在一些 bugs。直到我烛行调试代码时我才能发现它们。对于 .NET 语言来说，在书写同样数量的代码时，编译器会捕捉到大部分的语法 bugs (很抱歉，没有任何一个编译器可以捕捉到你应用程序逻辑的 bugs)。

一个简单的 C# 程序

尽管这一章不涉及面向对象，但是为了创建一个简单的示例程序，你还是需要知道一些基本知识，这是了解 C# 如何使用的一个基本前提。所以，现在必须要创建一个“Hello World”程序。在一个 .NET 控制台程序中调用 `Console.WriteLine` 来显示“Hello.NET World!”：

```
using System;  
class MyFirstCSharpProgram  
{  
    public static int Main()  
    (  
        Console.WriteLine("Hello .NET World!");  
        return 0;  
    }  
}
```

在这段代码中你要注意几点。每一个 C# 程序必须要有一个 `Main` 方法。当程序第一次被执行时此方法被调用，它必须返回 `void` 或 `int` 型的值。可选的，你可以指定 `Main` 方法接收一个字符型数组参数。任何传递的命令行参数，都被放入这个数组：

```
using System;
class MyFirstCSharpProgram
{
    public static int Main(string[] args)
    {
        // 这里是注释
        Console.WriteLine("Hello .NET World"); //这里是其他的注释
        // 处理参数
        if (args.Length > 0)
        {
            // 生成一个字符串包含所有参数
            String Parms = "";
            foreach (string parameter in args)
            {
                Parms += parameter + " ";
            }
            // 显示参数
            Console.WriteLine("\nParameters: " + Parms);
        }
        return 0;
    }
}
```

此段代码中处理参数所包含的一些 C# 特性我还没有讲到，例如数据处理和 foreach 循环。你可以在本章后面的小节找到关于这些特性的信息。

C# 语法

本节，我将在和 Visual FoxPro 相比较的基础上给出 C# 语法的概述。我将在 C# 中使用最多的两个开始向你予以展示。

大小写敏感

Visual FoxPro 不是一个大小写敏感的语言。下面的代码定义了一个名为 “MyString” 的变量并访问了它的值。在第二行代码中，我使用了这个变量，但是我称它为 “mystring” (全部小写)。在 Visual FoxPro 中，它们表示同一个变量。

```
MyString = "Visual FoxPro 不是大小写敏感的"
MESSAGEBOX(mystring, 0+48, "大小写敏感")
```

C# 是一个大小写敏感的语言。“MyString” 和 “mystring” 表示两个完全不同的变量。此外，所有的关键字都是小写。如果你在使用关键字时使用大小写混合的方式，编译器会认为它是一个错误。有些开发者要花很长的时间来适应 C# 的区分大小写。然而，如果你在 Visual FoxPro 中已经以区分大小写的方式书写代码，那么，这对你来说不是一个大问题。

分号和多行语句

在 Visual FoxPro 中，你不需要在每行代码结尾放置一个结束符。但是，如果一个单一的语句包含多行代码，你就需要增加一个分号，它是 Visual FoxPro 的续行符：

```
MESSAGEBOX("这个命令包含" +  
"多行", 0+48, "续行符")
```

在 C# 中这恰恰是相反的。就像前面小节中的示例代码，所有的 C# 语句必须以一个分号做结尾。这允许包含多行的命令可以不需要续行符。例如：

```
MessageBox.Show("这个命令包含" +  
"多行代码，但是不需要续行符");
```

对于使用 C# 的 Visual FoxPro 开发者来说，起初，这多少会让人感到些讨厌，但是，几乎不用考虑的键入分号并不会花费你太多的时间。

代码位置

在 Visual FoxPro 中，你可以在类中、单独的函数、过程、程序中书写代码。这允许你混合使用面向过程和面向对象。

在 C#，*所有的代码都必须放在类或类似于类的结构中*。关于结构的讨论在第 5 章“C# 和 Visual Basic.NET 中的面向对象”。如果非要选择的话，我认为 C# 的这个方法比 VFP 的方法要好，虽然在 VFP 中它可以带来更多的灵活性，但是，在大多数情况下，它是一种比较低劣的面向对象编程。

语句的分组

在 Visual FoxPro 中如果要区分一组语句，通常在开始和结束位置使用关键字。例如，你会在 DEFINE CLASS 和 ENDDFINE 关键字之间列出类的元数据。你会在 PROCEDURE 和 ENDPROC 或者 FUNCTION 和 ENDFUNC 关键字之间区分类方法中的代码：

```
DEFINE CLASS MyClass AS Session

    PROCEDURE MyProcedure
    ENDPROC

    FUNCTION MyFunction
    ENDFUNC

ENDDDEFINE
```

C# 使用大括号将代码括到一起。这里有一个代码示例，它定义了一个名为 CodeGroupingDemo 的类，这个类包含两个方法 MyMethod1 和 MyMethod2：

```
public class CodeGroupingDemo
{
    public void MyMethod1()
    {
    }

    public void MyMethod2()
    {
    }
}
```

尽管刚开始你可能觉得这样风格的代码阅读起来比较困难，但它并不比 Visual FoxPro（和 VB.NET 就这一点来说）显的更罗嗦，这意味着你可以少打好多字，并且，它贯穿整个语言。

注释

在 Visual FoxPro 中，你可以在行首使用熟悉的星号(*)来标记注释。你也可以在行末使用双符号(&&)来标记注释。

C# 无论在行首还是在行尾都可以使用双反斜杠来标记注释。对于多行注释，它在第一行注释的行首使用一个反斜杠和一个星号(/*)作为开始标记，在末行使用一个星号和反斜杠(*/)作为结束标记。

```
public class CommentDemo
{
    /* 多行注释
       这是注释的第二行 */

    // 注释
    public void MyCommentedMethod() // 另一个注释
    {
    }
}
```

C# 有一个非常酷的特性，它允许你使用 XML 来文档化你的代码。详细内容参看本章后面的“XML 文档化”一节。

命名空间

.NET 中所有的类都属于一个命名空间。一定要牢记，命名空间不是一个物理存在的类，它和 Visual FoxPro 中的项目或类库是完全不同的。它是对类进行逻辑分组的简单命名约定。例如，本书中所有相关源代码都隶属于命名空间 `HW.NetBook.Samples`。关于命名空间的更多信息，请参看第 1 章“`.NET` 简介”。

要将一个类指派给一个命名空间，你所需要做的仅仅是将类放置于当前命名空间定义的大括号中：

```
namespace HW.NetBook.Samples
{
    public class MyDemoClass
    {
    }

    public class MyOtherDemoClass
    {
    }
}
```

因为命名空间并不是一个物理定义，所以你可以将代码放到不同的物理源文件或程序集中，并赋予它们和命名空间相同的名字（译者注：这就是为什么我们在全局程序集缓存或其他位置，看到具有和命名空间相同名字的文件名的原因）。这一点是和 Visual FoxPro 不同的，在 Visual FoxPro 中，类是由它的物理位置来指定的——一个类库或一个 PRG 文件。

在 Visual FoxPro 中，在实例化一个类之前，你需要使用 `SET CLASSLIB` 或者 `SET PROCEDURE` 来打开包含类的类库或 PRG 文件。同样的，在 C# 中，你必须要使用“`using`”来指定类所隶属的命名空间。下面的代码使用了两个不同的命名空间 `System` 和 `System.Windows.Forms`：

```
using System;
using System.Windows.Forms;

namespace HW.NetBook.Samples
{
}
```

当你声明你使用 `System` 命名空间时,它并不会赋予你自动访问在 `System` 命名空间下的命名空间(例如 `System.Windows.Forms`)。你必须明确的声明你所使用的每一个命名空间。`using` 声明告诉 C# 到哪里去找到在你代码中引用的类。

定义一个简单的类

正如我承诺的,本章我不会讲太多关于面向对象的知识,但是为了更好的理解本章的示例代码,你至少要理解如何定义一个简单的类及其方法。

这是用 C# 定义的最简单的类(几乎什么都没有):

```
class MyFirstCSharpClass
{
}
```

下面的代码是具有一个单一简单方法的类。注意,我把它放到了命名空间(`HW.NetBook.Samples`)中,并且引用了 `System.Windows.Forms` 命名空间。因为 `MessageBox` 类隶属于这个命名空间,所以这是必须的。

```
using System.Windows.Forms;

namespace HW.NetBook.Samples
{
    public class MyFirstCSharpClass
    {
        public void SayHello()
        {
            MessageBox.Show("Hello .NET World!", "C#");
        }
    }
}
```

默认基类

在 Visual FoxPro 中,“父类”用于描述类是从哪个类派生的。在 .NET 世界,使用“基类”来表述同样的意思。在 VFP 中,‘基类’被称为最顶层的 Visual FoxPro 类(例如文本框类、组合框类、自定义类、容器类等等)。我只所以说这些,是因为在定义 `MyFirstCSharpClass` 时,没有指定它的父类或基类。

在 C# 中,如果你没有指定更多的东西,类就假定是源于 `Object` 类。`Object` 类位于 .NET 类层次的最顶端(所有的基类都源于它)。`.NET Framework` 中的每个类都继承自 `Object` 类。

定义类方法

你已经看到了一个简单类的示例。这里是类方法的官方语法：

```
[modifiers] returnType MethodName([parmType parmName])
{
    // Method body
}
```

在 C#，返回值类型和参数类型必须被明确指定。这是为了适应 C# 的强类型语言特性的。方法如果没有返回值，那么就使用 `void` 返回类型。

定义变量

在方法中定义的变量是局部变量（不存在其他类型的变量）。这意味着它们仅在定义它们的方法中是可见的。基于此，在定义一个变量时，你不需要指定它的作用域（scope）（局部、私有、公共）。

要定义一个变量，首先指定它的类型，然后是变量名：

```
int Count;    // 定义一个名为”Count”的整型变量
Count = 5;    // 指派变量 Count 的值为 5
```

你也可以在一条语句中定义变量并为其赋值：

```
// 定义一个名为 Height 的 int 变量并将赋予其值 6
int Height = 6;
```

如果你想在一条语句中定义多个同类型的变量，你可以使用逗号进行分割：

```
int Top = 5, Left = 39;
```

字段

在前面的小节提到，在方法层次上定义的变量是局部变量。然而，像下面代码这样，在类层次定义的变量就被称为 **字段**。

```

public class FieldsAndLocalVars
{
    string Test1 = "Test1";    // 定义一个字段

    public void ShowFieldsAndVars()
    {
        string Test2 = "Test2"; // 定义一个局部变量
        MessageBox.Show("Displaying Field " + Test1);           // 显示字段
        MessageBox.Show("Displaying Local Variable " + Test2);  // 显示变量
    }
}

```

刚开始你或许认为字段等同于 Visual FoxPro 中的属性，但事实并非如此。除了字段之外，C# 类也有属性，相关内容参看第 5 章“C# 和 Visual Basic .NET 中的面向对象”。现在，你暂时认为字段是在类层次定义的变量就行，它可以被类中所有的方法访问。

另一个需要注意的是，你可以像引用变量那样的方式来引用字段。你并不需要像 Visual FoxPro 那样使用一个 `this` 修饰符，当然，你也可以使用 `this.Test1` 来引用 `Test1` 字段，但这并不是必须的。



在大量的 C# 类教学活动中，我极力推荐使用“`this`”。因为 C# 是一个大小写敏感的语言，使用“`this`”可以应用智能感应，允许你从元数据列表中进行选择，而不是手工输入它们，因为你有可能输入不正确。

如果你定义的局部变量和类所包含的字段重名，在局部变量的作用域中，它会掩盖字段。在这种情况下，如果你需要引用字段，你就必须要在字段名前使用 `this` 前缀。

字段修饰符(field modifiers)

字段修饰符是用来修正字段定义的。例如，这里有 5 个可用的字段修饰符（表 1），它们指定了一个字段的可见度：

表 1. 字段修饰符允许你指定字段的可见度

修饰符	修饰符中文含义 (译者添加)	描述
public	公共	访问无限制
internal	内部	访问被限制在当前项目中
protected	保护	访问被限制在包含它的类中或类的子类中
protected internal	受保护的内部	访问被限制在当前项目或类的子类中
private	私有	访问被现在包含字段的类中

将字段作为 `private` 是经过慎重考虑的很好的面向对象组织形式。为了加强这个约定，如果你不指定一个修饰符，C# 在默认情况下就会将字段看成是 `private`。如果你想允许其

他的类访问这个字段，你可以创建一个 `protected` 或 `public` 的字段（详细资料参看第 5 章“C# 和 Visual Basic .NET 中的面向对象”中的“属性”一节）。

值类型和引用类型(Value and reference types)

.NET 在值类型和引用类型之间有严格的区分。值类型变量直接存储数据，而引用类型变量存储的是能找到相关资料数据的地址。引用型变量实际就是指想一个对象的指针——但是你不能直接的访问指针。要更清晰的理解这个概念，你需要学习 C#在后台如何分配内存。

理解栈和堆（stack and heap）

.NET 运行时为你的数据类型提供两种内存分配方式——栈和堆。栈是一个内存区域，它用于存储固定长度的值类型。这包含例如 `integers`、`boolean`、`long`、`float`、`double` 和 `character` 这些基本数据类型。**图 1** 是它如何工作的概念示意图。当你定义 `integer` 变量“`x`”时，运行时会在栈里为变量分配空间。当你为变量赋予一个值时，.NET 运行时就会在栈中已分配的空间中存储这个值。

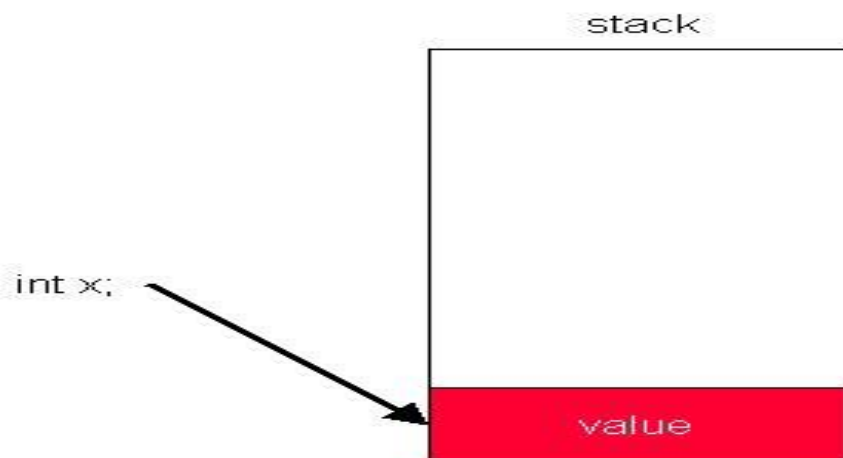


图 1. 当你定义一个值类型变量时，.NET 运行时在栈中为变量分配空间。当你对变量赋值时，该值就被存储到栈中已分配的空间中。

当你像下面这样将一个值类型变量拷贝到另一个值类型变量时会发生什么呢？

```
int x;  
int x = 5;  
int y = x;
```

这会在栈中创建同一数据的两份拷贝。当你声明变量 `x` 时，它在栈中被分配了空间。当你赋予变量 `x` 的值为 `5` 时，它将值 `5` 存储到栈中为变量 `x` 分配的空间中。当你将 `x` 赋予 `y` 时，它会在栈中为 `y` 分配空间并从 `x` 的空间中拷贝值 `5` 到为变量 `y` 分配的空间中。这样就在栈中存储了同一数据的两份拷贝。

堆是内存中用以存储例如类、数组、结构这种引用类型变量的区域。当你声明一个引用类型变量时，运行时在栈中为引用型变量分配空间——就像为值类型分配空间一样。然而，当你实例化引用型对象时（参看图 2），相比于将对象存储在栈来说更好的处理方法是将它存储到堆，然后一个指向对象数据地址的指针被存储于被存储到引用型变量在栈中的空间。

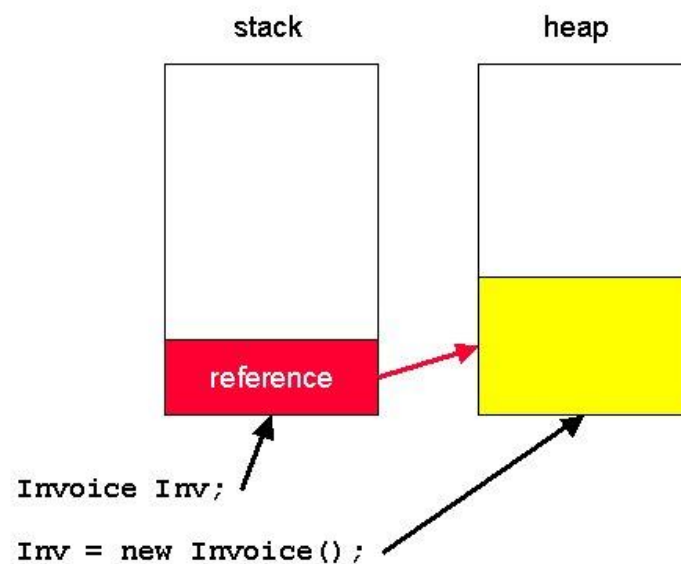


图 2. 当你声明一个引用型变量时，.NET 运行时在栈中为其分配空间。然而，当你实例化对象时，它在堆中存储对象数据，在栈中引用型变量所分配的空间中存储指向堆数据的指针。

当你像下面这样将一个引用型变量拷贝到另一引用型变量时会发生什么？

```
Invoice Inv;  
Inv = new Invoice();  
Invoice Inv2 = Inv;
```

这会在堆里创建两个到同一数据的引用。当你声明变量 `Inv` 时，运行时会在栈里为引用型变量分配空间。当你实例化 `Invoice` 对象时，它会在堆里存储对象的数据，并在栈中为 `Inv` 变量分配的空间里存储到 `Invoice` 对象数据的指针。当你向变量 `Inv2` 拷贝 `Inv` 变量时，它首先在栈中为 `Inv2` 变量分配空间。然后，它会在栈中该空间中增加一个到 `Inv` 变量在堆中存储对象数据的指针。

值类型和引用类型的性能

像 `integers`、`bool`、`decimal` 以及 `char` 都是基本数据类型 (Primitive base types)，`char` 是值类型并提供更好的性能，因为在你每次使用这些类型时在栈中没有创建对象的开销。

字符串类型（The string type）

`string` 类是一个引用类型，虽然表面上，你仍旧可以使用熟练的命令像值类型那样操作它。

你可以使用双引号的形式将字符串赋予一个 `string` 变量：

```
string Caption = ".NET for VFP Developers";
```

和 Visual FoxPro 不同，你不能使用单引号或者方括号。连接两个字符串的方法在 C# 和 Visual FoxPro 中完全相同，都是使用一个加号 (+)：

```
string Title = "Sir"  
string Name = Title + "Laurence Olivier";
```

随之而来的问题就是“我如何创建一个带双引号的字符串，例如 “Satchmo” 这样的”？这就需要用到字符串转义。

表 2 显示了一些特定的字符，它们被称为转义符，它们可以被增加到字符串中。例如要在一个字符串中增加一个断行符，你可以使用 `\n` 转义符。下面的代码在城市/国家和邮政编码之间增加一个新行：

```
string CityStateZip = "Charlottesville, VA \n22903";
```

要解决最初的问题，你可以通过使用双引号的转义符 `\"` 在字符串中增加双引号：

```
string ManlyMen = "Louis \"Satchmo\" Armstrong";
```

Table 2. 本表显示了常用转义符和特殊字符，你可以把它们增加到你的字符串中。

转义符	描述
\a	Alarm (bell)
\t	Tab
\r	回车
\n	新行
\"	双引号

你可以通过在字符串前加一个前缀符号 `@` 来告诉 C# 忽略转义符：

```
string Folder = @"c:\temp";
```

将一个字符串赋予另一个字符串

.NET 运行时认为字符串不同于其他的引用类型。首先，当你将一个字符串赋予另一个字符串时：

```
string State1 = "California";  
string State2 = State1;
```

它像预期的那样工作。这个代码在栈中创建了两个变量，它们都指向在堆中的同一个“California”。

然而，如果你更改了第一个字符串：

```
string State1 = "Sunny California";
```

.NET 运行时会在堆中创建一个新的 `string` 对象，并在栈中存储一个到新数据的引用，`State2` 仍旧指向在堆中的旧有的数据。利用这些资料，你可以在如何控制字符串以便在堆中不去创建大量的字符串就有了更好的选择，使用 .NET `StringBuilder` 类来代替使用 `string` 是最好的选择。`StringBuilder` 类有一个 `Append` 方法将字符串连接到一起：

```
StringBuilder StrBuilder = new StringBuilder();  
StrBuilder.Append("Texas, ");  
StrBuilder.Append("New Mexico, ");  
StrBuilder.Append("Colorado");  
MessageBox.Show(StrBuilder.ToString());
```

枚举

C# 允许你定义你自己的复杂的值类型。其中的一种值类型就是枚举（另一个是结构，请参看第 5 章“C# Visual Basic .NET 中的面向对象”）。

枚举是 C# 中一个很方便的特性，它允许你将基于 integer 的相关常量组合在一起。你可以认为它是常量数组。在 Visual FoxPro 中没有和它对应的内容，但是你很容易就能看出它们是多么的有用了。

下面的代码是 FoxPro.h 文件中定义的一组常量（这个文件位于 Visual FoxPro 主目录）。这些常量定义了 VFP 游标可能具有的“Buffering”值：

```
#DEFINE DB_BUF OFF      1
#DEFINE DB_BUF LOCK RECORD 2
#DEFINE DB_BUF OPT RECORD 3
#DEFINE DB_BUF LOCK TABLE 4
#DEFINE DB_BUF OPT TABLE 5
```

当你在一个命令中使用这些常量时，例如 CursorSetProp()，你经常需要打开 FoxPro.h 文件来查看有哪些值可用。枚举就可以帮助你做这样的事。

在 C# 中，你可以定义一个枚举，它包含所有针对 Buffering 的可能设置：

```
public enum Buffering
{
    BufOff = 1,
    BufLockRecord = 2,
    BufOptRecord = 3,
    BufLockTable = 4,
    BufOptTable = 5
}
```

你可以通过引用枚举的名字来访问枚举。编译器会将它转换为和它对应的 integer 值：

```
Buffering.BufOptRecord
```

如图 3 所示，枚举可 hook 到 .NET 智能感应。当你键入枚举名和一个句点时，智能感应就会显示在枚举中所有可用的值。

```
public class EnumerationTest
{
    Buffering.|
}
```



图 3. 枚举可 hook 到 VS.NET 的智能感应，这样就可以从一个有限的列表中很容易的选择可用值。

尽管枚举是基于 integer 的，如果你试图存储一个枚举到一个 integer 变量，编译器仍旧会抱怨：

```
int BufferMode = Buffering.BufOptRecord;
```

编译器无法隐式的转换“Buffering”到一个 integer。尽管你可以使用 cast 显式的转换枚举值到一个 integer。（请参看本章后面的“Casting”小节）你也可以通过指定变量类型为“Buffering”来绕开这个问题：

```
Buffering BufferMode = Buffering.BufOptRecord;
```

尽管枚举默认情况下是 integer 类型，你可以指定它们为 byte、sbyte、short、ushort、int、uint、long 或 ulong。例如：

```
public enum Buffering : long
```

在这个语句中，: long 指定 Buffering 类是基于 long 类型的。关于子类和继承，请参看第 5 章“C# 和 Visual Basic .NET 中的面向对象”。

数组

Visual FoxPro 中的数组和 C# 中的数组有很多相似之处。它们之间最大的不同是，在 C# 中数组被作为对象，允许有自己的方法和属性。另一个不同是 Visual FoxPro 允许你在一个数组中存储多种数据类型（integers、strings、dates 等等），但是在 C# 中所有的数组元素必须是同一类型。



你可以通过声明数组包含的项目类型是“Object”来绕开数组元素必须是同一种类型的限制，因为 .NET Framework 中所有的类都是基于它。

第三个不同是 C# 的数组是从 0 开始的，第一个数组元素是 0。在 Visual FoxPro 中，第一个数组元素是 1。

声明数组

在 C#，数组的生命方式和其他变量相同，除了在变量类型和变量名之间加上方括号 ([])。例如，下买内的语句声明了一个名为 “KidsAges” 的数组：

```
int [] KidsAges;
```

你也可以将声明和设置数组的大小在一个语句中完成：

```
string [] KidsNames = new string[3];
```



一旦你声明了数组的长度，你就不能再调整它。如果你想在运行时刻使用一个可以灵活调整大小的数组，你需要使用 **ArrayList** 对象。你可以在 **System.Collections** 命名空间中找到它。详细信息参看 **.NET** 帮助文件。

你可以通过调用数组的 **GetLength** 方法来获得任意数组的长度，也可以使用它来传递你所想使用的数组元素：

```
MessageBox.Show("Number of kids = " + KidsNames.GetLength(0));
```

在数组中存储值

当你声明数组时，你可以设置数组元素的初始值：

```
string [] KidsMiddleNames = {"Christopher", "Mark", "James"};
```

或者，你可以在声明数组之后在数组元素中存储值：

```
string [] KidsNames = new string[3];  
KidsNames[0] = "Jordan";  
KidsNames[1] = "Timothy";  
KidsNames[2] = "Alexander";
```

排序数组

通过调用数组的 **Sort** 方法，你就可以很容易的排序数组：

```
Array.Sort(KidsNames);
```

通过调用 **Array** 类的 **Reverse** 方法，你可以对数组进行逆序排序：

```
Array.Reverse(KidsNames);
```

多维数组

迄今为止在本书中看到的数组都是一维数组。Visual FoxPro 允许你创建一维和二维数组。在 C# 中更进了一步，允许你创建多维数组——数组可以是三维或更多维！无论如何，除非你有特别的需求，你大概不会需要多于二维的数组。

多维数组或者是矩形数组（每行都有同样数量的列），或者是交错数组（每行有不同数量的列）。Visual FoxPro 中的数组是矩形数组，而没有交错数组。

定义多维矩形数组

你可以通过在声明数组语句的方括号中增加一个或多个逗号来定义一个矩形数组。每增加一个逗号，数组就增加一维。这里定义的是二维数组，它具有两列三行：

```
string[,] ArrayKids = { {"Jordan", "McNeish"},  
                        {"Timothy", "McNeish"},  
                        {"Alexander", "McNeish"} };
```

定义交错数组

交错数组（jagged array）是一个“数组的数组”。它是多维的，但是，和矩形数组不同，每行都有不同数量的列。通过在每一维增加额外的方括号，你可以定义一个交错数组。在下面的示例中，在第一维，交错数组具有三个元素。它的第二维具有和其他行不同的列数。第一行有三列，第二行有两列，第三行有一列：

```
string[][] Musicians = new string [3][];  
Musicians[0] = new string[] {"Stevie", "Ray", "Vaughn"};  
Musicians[1] = new string[] {"Jimi", "Hendrix"};  
Musicians[2] = new string[] {"Bono"};
```

类型转换

使用强类型语言时，在不同的数据类型之间进行转换是一个很重要的事。C# 提供了隐式转换和显式转换两种方式。



C# 数据类型的比较，以及和 VB.NET 、 VFP 数据类型的比较，请参看“附录 C—数据类型对照表”。

隐式类型转换

C# 可以隐式转换一些值。在下面的示例中，C# 自动转换 short 值到一个 integer ，然后再转换为 double ：

```
short x = 10;
int y = x;
double z = y;
```

表 3 列出了在 C# 中可被（自动）隐式转换的类型。

Table 3. 在 C# 中可隐式转换的数据类型

从	到
byte	ushort, short, uint, int, ulong, long, float, double, decimal
sbyte	short, int, long, float, double, decimal
short	int, long, float, double, decimal
ushort	uint, int, ulong, long, float, double, decimal
char	ushort, uint, int, ulong, long, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
float	double

使用 casts 进行显式类型转换

如果编译器拒绝隐式转换，你可以使用 cast 来替代。像下面示例一样，在你所想转换的值前面加上使用圆括号括起来的目标类型这样的前缀，将值从一个类型转换到另一个类型。下面的代码转换 integer 类型到 byte 类型（编译器不会在它们之间进行隐式转换）：

```
int x = 20;
byte y = (byte)x;
```

转换到字符串

如果你想转换一个值到 `string` 类型，`Object` 类有一个 `ToString` 方法（它被 `.NET Framework` 中所有的类继承），它可以返回对象的字符串形式。所以，要得到 `integer` 的字符串形式，你可以这样做：

```
int i = 10;
string s = i.ToString();
```

使用 `Parse` 来转换字符串

如果你需要转换 `strings` 到其他值类型，你可以使用目标类型的 `Parse` 方法。例如，如果你想转换 `string` 到 `integer`，你可以使用 `int` 类型的 `Parse` 方法：

```
string StringAmt = "100";
int IntAmt = int.Parse(StringAmt);
```

如果你想转换 `string` 到 `long`，你可以使用 `long` 类型的 `Parse` 方法：

```
string StringAmt = "100";
long IntAmt = long.Parse(StringAmt);
```

你也可以使用 `Parse` 方法将 `string` 转换到 `Boolean`。然而，字符串的值必须是“True”或者“False”（不区分大小写），否则，在运行时你会得到错误“字符串不能被验证为一个有效的布尔值”。

装箱和取消装箱(Boxing and unboxing values)

装箱指将一个值类型转换为引用类型的过程。取消装箱是将一个应用类型的值还原为值类型的过程。

当你将一个 `integer` 类型赋值给一个对象时隐式发生：

```
int i = 10;
object o = i;
```

图 4 描述了当这个代码执行时会发生什么。在第一行，`int` “i” 在栈中被分配空间，并将值 10 赋予变量。在第二行，对象 “o” 的引用变量在栈中被分配，值 10 被拷贝到位于堆的对象 “o” 的数据空间中。

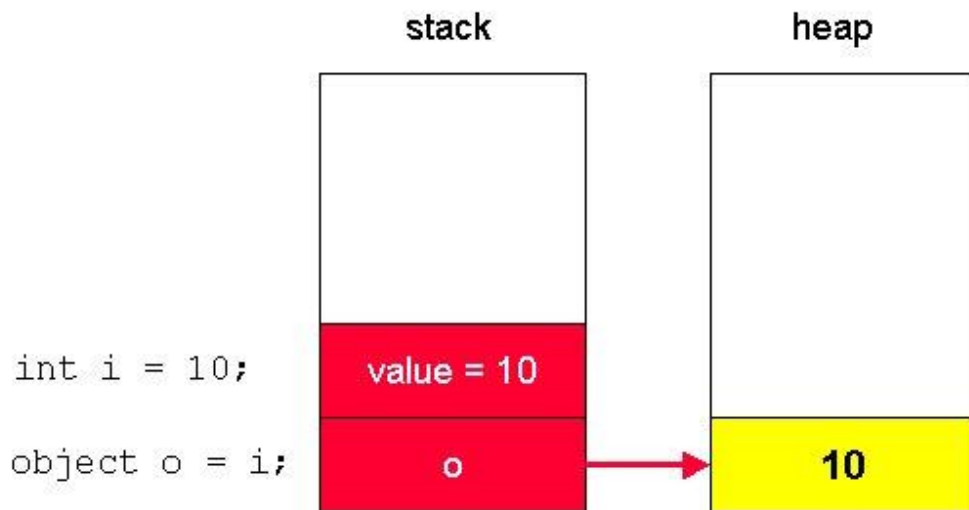


图 4. 当你存贮一个 `integer`(或其他)类型的值类型到一个对象时, 值类型就被隐式的装箱。

取消装箱只有显式的发生。你仅可以将先前的装箱取消装箱。例如：

```
// Boxing
int MyInteger = 10;
object MyObject = MyInteger;

// Unboxing
int MyUnboxedInt = (int)MyObject;
```

is 运算符

你可以使用 `is` 运算符来确定一个对象是否是指定的类型。这在方法接受参数时是非常有用的，它可以用来确定所接收的常规对象类型是否是指定的类型。例如，下面的方法接受一个参数类型 `object` 。然后检查它是否是 类型 `MyClass1` ：

```
public void CheckClass(object testObject)
{
    if (testObject is MyClass1)
    {
        MessageBox.Show("TestObject 是 MyClass1",
            "is 运算符");
    }
    else
    {
        MessageBox.Show("TestObject 不是 MyClass1",
            "is 运算符");
    }
}
```

if 语句

C# 的 if 语句和 Visual FoxPRO 的 IF...ENDIF 语句很像。如果在 if 后面圆括号中的表达式结果为真，就执行当前的代码块。可选的，你可以在 if 语句中增加 else 语句，当 if 的条件表达式为假时也能得到处理。C# 也允许嵌套 if 语句。

如果在 if 或 else 后仅有一条语句，你并不需要使用大括号，然而，这样做是一个很好的习惯。

```
bool HavingFun = true;

if (HavingFun)
{
    MessageBox.Show("我们非常高兴！", "if 示例");
}
else
{
    MessageBox.Show("我们并不高兴", "if 示例");
}
```

switch 语句

C# 的 switch 语句和 Visual FoxPro 的 DO CASE 语句等效。基于表达式的值，case 语句之一被执行。你可以指定一个 default 分支，它和 VFP 的 OTHERWISE 子句作用相同。如果 switch 表达式和所有 case 都不匹配，控制将被传递到 default 分支。不过不存在 default，控制将被传递到 switch 之后的第一个命令。

```
testValue int = 2;
switch (testValue)
{
    case 1:
        MessageBox.Show("testValue = 1", "switch");
        break;
    case 2:
        MessageBox.Show("testValue = 2", "switch");
        break;
    case 3:
    case 4:
        MessageBox.Show("testValue = 3 or 4", "switch");
        break;
    default:
        MessageBox.Show("testvalue is not 1, 2, 3 or 4",
            "switch");
        break;
}
```

正如上面示例显示的，你不能从一个 case 到另一个 case，除非它是空的。这意味着你必须使用下面之一的跳转语句：

- break - 分支结束的标志，控制被传递到 switch 后的语句。
- goto - 指定控制传递到指定的分支（例如，“goto case 2”），或者到 default 分支（例如，“goto default”）。
- return - 分支结束的标志，并结束当前方法的执行。

for 循环

C# 的 for 循环等效于 Visual FoxPro 的 FOR...ENDFOR 命令。当特定的条件为真时，它重复执行循环。

在 for 语句后面的圆括号中，有三个不同的项目。第一个是初始化（initializer）（例如 i=0;）。在这部分语句中你可以放任意变量的初始化，尽管通常情况下初始化指定的单一变量。第二部分是要被执行的表达式（例如 i < 4）。在循环被执行时，这个表达式被计算出结果。第三部分是迭代器（iterators）（例如 i++）。这是用来表示增量或减量的循环计数器。

在 C# 中，++ 运算符用来递增一个变量，-- 运算符用来递减一个变量。关于 C# 运算符的更多信息参看“附录 A—语言对照表”。

```
int i;

string Message = "";

for (i=0; i < 4; i++)
{
    Message += "Message " + i + "\n";
}
MessageBox.Show(Message, "For 循环");
```

while 循环

C# 的 while 循环等效于 Visual FoxPro 的 DO...WHILE 循环。它是预检测循环，也就是说，如果条件表达式为假，则循环就不会被执行。While 循环通常用于在未知的时间内重复运行代码块。

```
bool Condition = true;
while (Condition)
{
    // 这个循环仅执行一次
    int i = 1;
    MessageBox.Show("While 循环计数 " + i, "while 循环");
    i++;
    Condition = false;
}
```

do 循环

C# 的 do 循环和其自身的 while 循环很相似,它的条件检测是在循环的末尾而不是在开始。这意味着循环至少要执行一次。这和 Visual FoxPro 的 DO...WHILE 循环是不同的,它是在循环的开始进行检查。

```
bool Condition = false;
do
{
    // 这个循环仅执行一次
    // 因为条件的检查是在循环的末尾
    int i = 1;
    MessageBox.Show("Do While 循环计数 " + i, "do...while 循环");
    i++;
} while(Condition);
```

你可以使用 break、goto 或 return 语句退出 do 循环。

foreach 循环

C# 的 foreach 循环针对数组的每一个元素或集合中的每个项目执行一组语句。它等效于 Visual FoxPro 的 FOR EACH 命令。

```
string VersionList = "";
string[] VSNetVersions = {"Standard", "Professional",
                          "Enterprise Developer", "Enterprise Architect"};

foreach (string Version in VSNetVersions)
{
    VersionList += "VS .NET " + Version + "\n";
}
MessageBox.Show(VersionList, "ForEach loop");
```

XML 文档化

C# 有一个很酷的特性,它允许你使用 XML 文档化你的代码。

如果在一个用户自定义类型（例如类、结构、枚举、委托或接口）或一个元数据（例如字段、属性或方法）的前面键入三个一行的反斜杠，XML 文档模板就会自动被插入到代码中以便你增加注释。例如，假设你有下面这样的 C# 类：

```
public class CommentDemo2
{
    public string GetString(int value)
    {
        return value.ToString();
    }
}
```

如果你在类定义和方法定义之前开始输入三个斜杠，下面的 XML 文档模板就会增加到你的代码中：

```
/// <summary>
///
/// </summary>
public class CommentDemo2
{
    public string GetString(int value)
    {
        /// <summary>
        ///
        /// </summary>
        /// <param name="value"></param>
        /// <returns></returns>
        return value.ToString();
    }
}
```

你可以在两个<summary>标记之间输入类的描述和方法的描述。注意，C# 已经为参数和返回值创建了 XML 标记。你也可以在 <param> 和 <returns> 标记间添加注释。例如：

```
/// <summary>
/// 注释示例类
/// </summary>
public class CommentDemo2
{
    /// <summary>
    /// 转换 integer 值到 string
    /// </summary>
    /// <param name="value">转换 integer 值到 string </param>
    /// <returns>转换后的 string 值</returns>
    public string GetString(int value)
    {
        return value.ToString();
    }
}
```

那么，XML 格式的注释对你有什么用呢？在 Visual Studio .NET 中的 C# 开发环境可

以读取你的 XML 文档并自动转换它到“注释 Web 页”!

要建立这些 Web 页，在 VS.NET 中打开你的 C# 解决方案，然后选择 工具|生成注释 Web 页 菜单。这会显示“生成注释 Web 页”对话框（图 5）。这个对话框允许你指定是针对整个解决方案还是所选项目创建创建。你甚至可以把它增加到你的 IE 浏览器的收藏夹中。



图 5. 你可以针对整个解决方案或单一项目生成注释 Web 页。

当你单击“确定”按钮时，Web 注释页就会生成。当它完成后，你会在 Visual Studio.NET 中的代码注释 Web 报表标签看到它（图 6）。



图 6. 在生成注释 Web 页后, 会显示一个包含所有项目列表的代码注释 Web 报告页。

如果你单击项目链接, 一个显示项目中所有命名空间的 Web 页会显示出来。如果你展开命名空间节点, 就会显示出隶属于命名空间的类。选择某一个类会显示出类及其所有成员的信息, 这些都源于你在 C# 源代码中插入的注释 (图 7)



图 7. 注释 Web 页显示了源于插入 C# 源代码中注释的类的信息。

如果你单击一个元数据, 例如方法, 它会显示关于方法参数和返回值的详细信息 (图 8)。

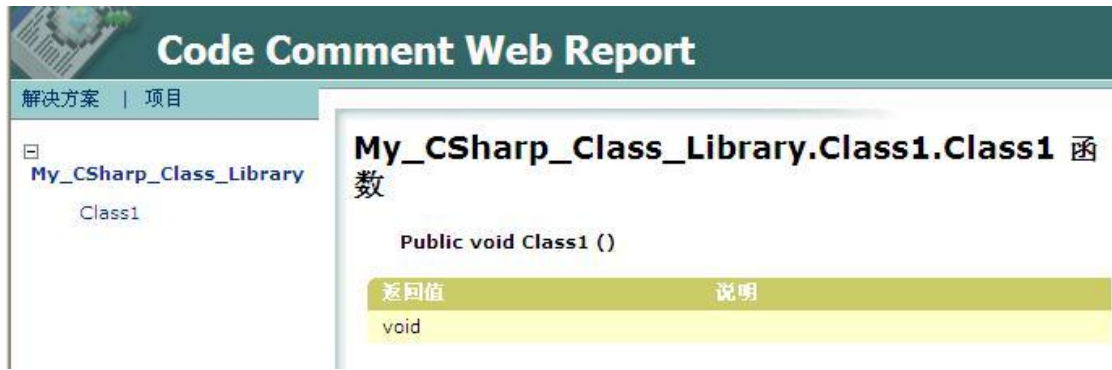


图 8. 注释 Web 页也显示方法参数和返回值的详细信息。

注释 Web 页是 Visual Studio .NE 很好的增强, 因为它不仅仅是一个工具, 而且, 它还鼓励开发者文档化他们的代码!

不安全代码(Unsafe Code)

C# 有一个在 Visual Basic.NET 中没有的高级特性——可以创建不安全代码。

不安全代码往往被误解为非托管代码。在第 1 章“.NET 简介”提到，非托管代码是在 CLR 之外执行的，它无法直接使用诸如垃圾回收这样的 .NET 服务。运行非托管代码的示例是在一个 Visual FoxPro COM 服务中调用一个对象的方法。

相比之下，不安全代码仍旧是托管代码，但是它允许你依赖指针直接访问内存。术语不安全并不意味着代码对你会存在危害。它仅意味着代码在运行时是不能确认它的安全。

所以，你为什么要书写不安全代码呢？这里有一些理由。最常见的理由是：提高性能。当你可以使用指针而不是引用型变量访问内存地址时，你少经历了一层，这提供了性能优势。如果你的应用程序对运行时间有要求，以获得更高的运行速度，你或许需要考虑不安全代码。

另一个或许创建不安全代码的理由是可以访问遗留的 DLL，例如一个 Windows API 调用，或者用做将指针作为参数传递而调用一个遗留的 COM DLL。

关于在 C# 中创建不安全代码的更多信息，参看来自微软的 Eric Gunnerson 在 MSDN 上文章：

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncscol/html/csharp10182001.asp>

C# 是一个国际化的行业标准语言

Microsoft 已经做了和 C# 语言有关的大量有意思的事。在 2000 年 10 月，微软向欧洲计算机制造协会（European Computer Manufacturers Association，简写 ECMA）提交了 C# 的审查报告，这是一个国际标准组织，希望可以使 C# 成为一个工业标准语言。在 2001 年 12 月 13 日，ECMA 批准了 C# 和公共语言规范（Common Language Infrastructure，简写 CLI）为工业标准。（CLI 类似于 Java 的虚拟机——它包含了库和允许 C# 运行于非 Windows 操作系统的计算机）。

这对微软和 C# 开发者意味着什么？首先，它意味着 C# 和 CLI 是 ECMA 官方管理和认可的标准，虽然微软保留了向谁发放技术许可的权利。此外，虽然有一些理论指出微软这样做“仅仅为了显示”（Sun Microsystems 也向 ECMA 提交了它们的 Java 编程语言，但被驳回），最后，这意味着你使用 C# 所写的程序或许可以运行在非 Windows 操作系统中，例

如 Linux、Unix 和 Palm——这是 Java 开始以来已具有的关键特性。这让你具有了和现在使用 C++ 创建多平台应用人具有了类似的能力。

总结

希望本章的 C# 语法、数据类型、类型转换、条件语句和控制流语句能让你对了解 C# 有好的帮助。尽管首先在本章提出了大括号和分号，如果开发者仍旧惊讶于 C# 的易学易用。

尽管在本章我覆盖了 C# 的基本语法，你在由 Wrox 出版社出版的《Professional C# - Second Edition》可以学习到更多的知识。

第四章 Visual Basic .NET 入门

很多 Visual FoxPro 开发者厌恶“Visual Basic”。本章的目的是让你抛弃原来的偏见，并注意 Visual Basic 的新的化身——Visual Basic .NET。由于你可以针对 Visual FoxPro 和 Visual Basic .NET 进行比较阅读，所以你将看到 Visual Basic 的成长，它已经是像 C# 和 C++ 一流的编程语言。

Visual Basic 已经存在了很多年。它是微软向大众提供的第一个 Windows 编程语言。在 Visual FoxPro 和 Visual Basic 之间经常发生争论，两个阵营都具有宗教般的狂热。

争论的最大焦点包含已经被 VB 开发者所断言但被 VFP 开发者极力反对的一个问题：Visual Basic 6 是一个面向对象编程语言。现在，事情已经有所改变。对所有的意图和目的来说，Visual Basic .NET 是一个全新的语言。它的语法类似于 Visual Basic，但是却被完全重写的 .NET 编程语言。

在下一章，我将详细介绍 C# 和 VB.NET 中的面向对象特性。本章，我将通过与 VFP 比较来介绍 VB.NET 的语法。所以，希望你能放下先入为主的观念来看看 VB.NET 提供了哪些新的东西。



Visual Basic .NET 和 Visual FoxPro 的操作符及关键字对照，请参看“附录 A—语言对照表”。关于每种语言的数据类型，请参看“附录 B—数据类型对照表”。

针对 Visual Basic 6 的向后兼容性改进

正如你在学习 Visual Basic .NET 时，你会感觉在 VB.NET 中有一些东西不同于 .NET 规范。在 Visual Studio .NET Beta 1 发布后，微软基于 Visual Basic 社区的反馈在 VB.NET 中重新加入了一些 VB6 的一些功能。这提了一个从 Visual Basic 6 到 Visual Basic .NET 的比较容易的升级方式。这里是三个主要的改进：

- **True 的值** - 在几乎每一种编程语言中，当一个逻辑值 True 被转换为整数时，它的值都是 1。在 Visual Basic 中，却一直是 -1。原来，微软打算更改它以便和其他 .NET 语言保持一致，但是依据从社区反馈的信息，在 VB.NET 里，True 仍旧被当作 -1。

- **And/Or/Not/XOr 的行为** - 起初，微软打算更改 And、Or、Not、XOr 位操作符（它们已存在于 Visual Basic 6）为逻辑操作符。基于社区的反馈，他们保留了这些并增加了

两个新的运算符：AndAlso 和 OrElse 。

- **声明数组** - 为了使 VB.NET 和其他的 .NET 语言保持一致，微软有意更改数组的声明方式，使它不同于 Visual Basic 6 。然而，为了保持向后兼容性，它们又将其更改回 Visual Basic 6 的方式。详细资料参见本章后面的“数组”一节。

弱类型 vs 强类型

VB.NET 是一种强类型语言，C# 是一种更强类型的语言。它们之间的不同在于，在 C# 中使用变量前声明变量及它们的类型。然而在 VB.NET 中，这些是可选的。

Option Explicit 和 Option Strict

VB.NET 有两个设置来指定编译器如何处理变量—Option Explicit 和 Option Strict。

Option Explicit 设置指定你是否可以使用未声明的变量—不仅仅指变量类型，而且也包括变量自身。默认情况下，Option Explicit 的设置是 “Off”。它允许你无须声明即可使用变量。当设置为 “On” 时，VB.NET 的行为就像 C# 那样，强迫你在使用变量前要声明所有的变量。



尽管关闭变量声明的检查似乎能够带来灵活性，实际上并非如此。你需要设置 **Option Explicit** 为 “ON”，以便使编译器可以捕获未声明的变量。如果设置它为 “OFF”，编译器将不能在运行时刻捕获变量名和变量类型导致的未知行为。

即使当 Option Strict 被设置为 “On”，Visual Basic .NET 和 C# 在使用变量时仍旧有一些不同。在 Visual Basic .NET 中，你可以使用一个变量而无须事先为它赋值（VB.NET 在你声明变量时为其赋予一个默认值）。依赖于你的个人喜好：强制编译器检查或需要更多的灵活性，你可以考虑这些设置是有益还是有害！

Option Strict 的设置指定编译器是否严格检查变量类型。如果 Option Strict 设置为 “OFF”（默认设置），编译器不需要你在声明变量时指定变量类型。如果 Option Strict 设置为 “ON”，那么 VB.NET 的行为就像 C#，强制你声明所有变量的类型。Option Strict 也指定在两种不同类型变量间转换时编译器怎样来执行检查。如果 Option Strict 被设置为 “OFF”，那么在变量类型转换时不执行检查。如果设置为 “ON”，在转换结果会丢失一些数据时，编译器会向你发出警告。例如，如果 Option Strict 为 “On”，当你拷贝一个 Long

变量的值到一个 Integer 类型的变量时，编译器就会报错，因为 Long 类型的变量所存储值的大小要大于 Integer。然而，如果你做相反的转换，编译器则不会抱怨。

通常情况下，最好设置 Option Strict 为“On”。这将允许编译器可以捕获在变量类型间转换时是否有数据丢失，这将比在运行时刻捕获它们要好的多。此外，显式的指定类型转换要比（自动的）隐式转换更有效率。这是因为在运行时刻隐式的转换需要需要额外的工作来确定在转换时所需的类型。

假设 Option Strict 和 Option Explicit 都被设置为“On”，你声明变量时就需要指定变量名以及变量类型。例如，下面的代码声明了一个名为 MyDate 的 Data 类型变量、一个名为 MyString 的 string 类型变量以及一个名为 MyInteger 的 Integer 类型变量：

```
Dim MyDate As DateTime
Dim MyString As String
Dim MyInteger As Integer
```

你可以在 VS.NET 的解决方案资源管理器中，在应用程序级别来设置 Option Explicit 和 Option Strict。仅需要右击项目，在快捷菜单中选择 **属性** 菜单，然后在公共属性页的树视图中选择生成节点（图 1）。然后你就可以对 Option Explicit 和 Option Strict 进行设置，这会影响你项目中的所有代码。

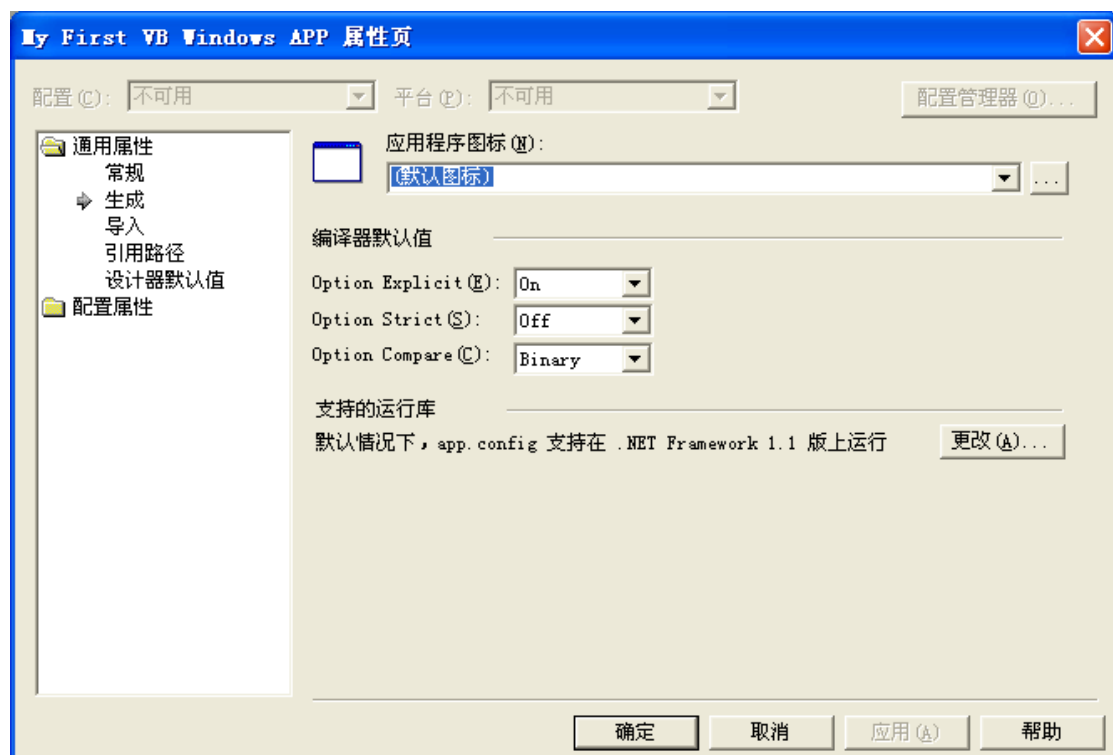


图 1. 通常情况下，你要设置 **Option Explicit** 和 **Option Strict** 为 “On”，以便编译器可以捕获未声明的变量、变量类型以及类型转换错误。

你可以通过在你的源文件的头部声明一个不同的 `Option Explicit` 或 `Option Strict` 设置来覆盖项目级别的设置。例如：

```
Option Explicit Off
Public Class MyForm
    Inherits System.Windows.Forms.Form
End Class
```

基于使用强类型的“最优方法”和显式变量声明，本书中的 Visual Basic .NET 示例代码假设这些设置都为 “ON”。

一个简单的 Visual Basic .NET 程序

尽管这一章并不介绍 Visual Basic .NET 的面向对象特性，但是如果不借助于学习如何创建简单的程序，那么学习它的语法就会比较困难。这里是一个用 VB.NET 所写的 “Hello .NET World” 程序：

```
Module MyFirstVBNetProgram

    Sub Main()
        Console.WriteLine("Hello .NET World")
    End Sub

End Module
```

在这个程序里你要注意一些东西。首先，每个 VB.NET 程序都必须要有个名为 `Main` 的过程。它必须是返回一个 `Integer` 或是没有返回值。如果你想返回一个 `Integer`，你必须指定 `Main` 是一个函数过程，而不是一个 `Sub` 过程：

```
Module MyFirstVBNetProgram

    Function Main() As Integer
        Console.WriteLine("Hello .NET World")
        Return 0
    End Function

End Module
```

你也可以指定 `Main` 接收一个字符型数组参数。当程序执行时，如果有命令行参数被传递，它们就会被放入这个数组：

```

Module MyFirstVBNetProgram

    Function Main(ByVal CmdArgs() As String) As Integer
        Console.WriteLine("Hello .NET World")

        ' 处理参数
        If CmdArgs.Length > 0 Then
            ' 建立一个包含所有参数的字符串
            Dim Parms, Parameter As String
            For Each Parameter In CmdArgs
                Parms += Parameter & " "
            Next
            ' 显示参数
            Console.WriteLine(Parms)
        End If

        Return 0
    End Function

End Module

```

注意 += 操作符的使用。在 Visual Basic .NET 中，它是下面方式的简写：

```
Parms = Parms + Parameter
```



尽管在技术上要求在 VB.NET 程序中需要有一个名为 Main 的过程，实际上，在你的源代码中或许看不到它。例如，如果你使用 Visual Studio .NET 创建一个新的 Visual Basic .NET Windows 应用程序，你就找不到 Main 方法。编译器在后台已经向 IL 中增加了一个 Main 方法。你可以忽略它并手工创建一个 Main 方法。

Visual Basic .NET 语法

本节在和 Visual FoxPro 比较的基础上给出 VB .NET 语法的一个概述。

大小写敏感

Visual Basic .NET 和 Visual FoxPro 一样是大小写不敏感的。尽管在 VB.NET 中约定关键字必须小写，你也可以用大写或大小写混合，编译器会认为它们是相同的。在 C# 中，关键字必须小写。如果你使用大写或大小写混合，编译器就会报错。

为了验证 VB.NET 的大小写不敏感，在下面的代码中我声明了一个变量 “MyString” 并赋予它一个值。在第二行代码中，我引用这个变量，但是我用全部小写的形式 “mystring”。在 VB.NET 和 Visual FoxPro 中，这两个名字都是引用同一个变量：

```

Dim MyString As String = "Visual Basic .NET is case-insensitive"
Console.WriteLine(mystring)

```

相反，正如在第三章“C#入门”中提到的，C# 是区分大小写的。如果你在键入代码时比较随意，这就会成为你选择 Visual Basic .NET 还是 C# 的一个影响因素。如果你严格的遵循大小写（并让智能感应“填充空白”），那么这对你并没有什么影响。

行终止符和续行符

在行终止符和续行符方面，Visual Basic .NET 更是很像 Visual FoxPro。在 VB.NET 中，一个 CR+LF 就表示一行的结束。如果你有多行的语句，你就必须使用续行符。在 Visual FoxPro 中，续行符是分号(;)，但是在 Visual Basic .NET 中，它是下划线（_）。

语句分组

Visual Basic .NET 中语句的分组是依靠配套的开始和结束关键字。例如，类定义以 Class 开始，以 End Class 结束。子程序以 Sub 开始，以 End Sub 结束：

```
Class MyClass
    Public Sub MySubroutine()
    End Sub
End Class
```

这很像 Visual FoxPro 中的开始和结束关键字：

```
DEFINE CLASS MyClass AS Session

    PROCEDURE MyProcedure
    ENDPROC

    FUNCTION MyFunction
    ENDFUNC

ENDDDEFINE
```

注释

在 VB .NET 中，你可以使用单引号来表示注释的开始。无论它是位于代码行的开始，还是位于代码行的后面：

```
' 这里是行首的注释
Dim MyInteger As Integer = 100    ' 这里是语句的注释
```

因为在 Visual Basic .NET 中语句的分行需要续行符，所以在 VB.NET 中就没有像 C# 那样的多行注释。

命名空间

在 Visual Basic .NET 中，使用 Imports 关键字来指定编译器到哪里来寻找当前命名空间中定义的类。例如：

```
Imports System.Data.OleDb
Imports System.Data.SqlClient
Imports System.Data.SqlTypes
```

所有的 Visual Basic .NET 项目都自动的导入几个默认的命名空间。一旦在项目层次导入它们，你就不需要在别处导入这些命名空间。不同类型的项目会自动导入不同的命名空间。要查看项目中默认导入的命名空间，你可以在解决方案资源浏览器的项目上右击项目，并在快捷菜单中选择 **属性**。选择 通用属性 节点下的 导入 节点，你将看到项目导入列表框中一系列默认导入的命名空间（图 2）。Visual Studio .NET 允许你增删这个列表。

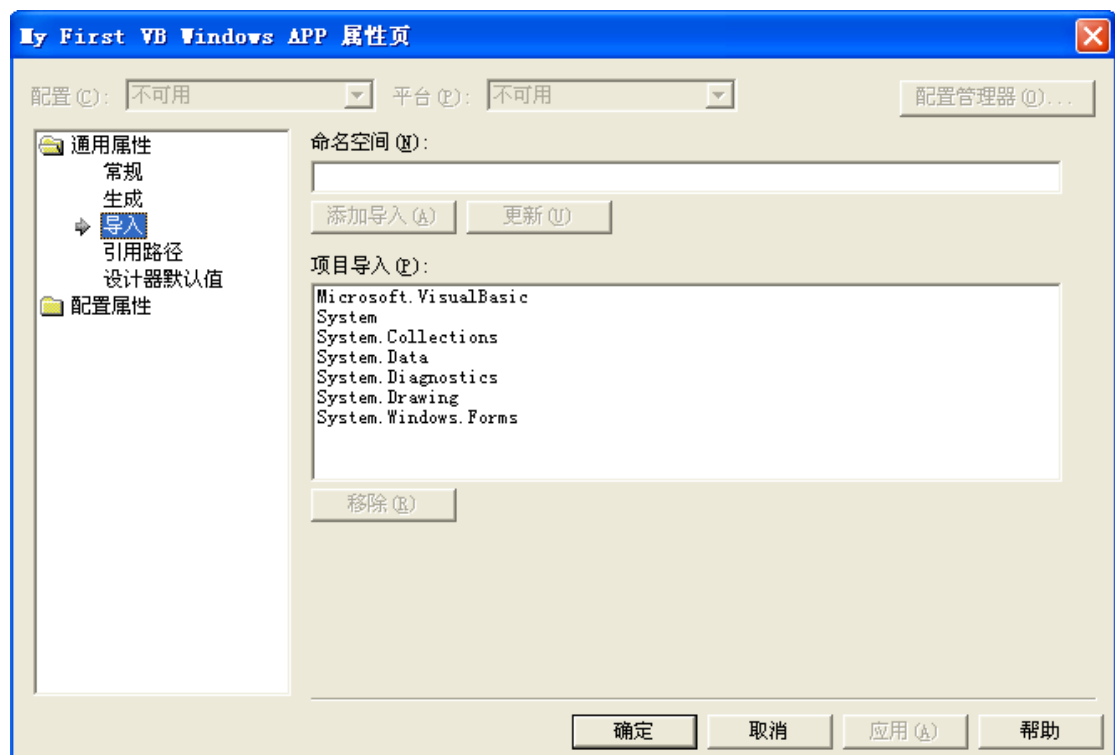


图 2. Visual Basic .NET 会在你创建的每个项目中自动增加一些默认的命名空间。列表中存在的命名空间，你不需要在使用它的地方明确的声明。

Visual Basic .NET 具有一个有趣的命名空间特性—项目根命名空间。要查看项目的根命名空间,右击解决方案资源管理器中的项目并在快捷菜单中选择 **属性**。选择 **通用属性** 节点下的 **常规** 节点,你将看到项目的根命名空间(图 3)。



图 3. 在项目属性对话框中你可以指定项目的根命名空间。

默认情况下,项目中所有的类都隶属于根命名空间。根命名空间是项目中所有其他声明的命名空间的前缀。例如,如果项目的根命名空间是“HW.NetBook”,在下面的代码中声明“Business”命名空间和 Employee 类,“Business”命名空间的完整名称是 HW.NetBook.Business。

```
Namespace Business

    Public Class Employee
    End Class

End Namespace
```

定义一个简单的类

尽管在下一章才介绍 Visual Basic .NET 中的面向对象,但是结合类定义还是比较容

易理解 VB.NET 的语法。这里定义了一个简单的 VB.NET 类：

```
Class MyFirstVBClass
End Class
```

下面的示例类具有一个单一、简单的方法。注意，我已经把类加入到命名空间 Samples 中。如果项目的根命名空间是 HW.NetBook（它是本书示例代码的根命名空间），那么这个类的完整的命名空间就是 HW.NetBook.Samples 。

```
Namespace Samples

Public Class MyFirstVBClass
    Public Sub SayHello()
        MessageBox.Show("Hello .NET World!", "Visual Basic .NET")
    End Sub
End Class

End Namespace
```

和上一章 C# 示例代码不同，你不需要为了使用 MessageBox 类来明确导入 System.Windows.Forms 命名空间，因为在默认情况下，它已经在项目层次被导入。

默认基类

如果你没有指定父（基）类，系统就假设它派生于 Object 类。正如前面提到的，Object 类是 .NET Framework 体系的超类。

定义类方法

Visual Basic .NET 具有两种不同的方法——子程序和函数。如果你的方法不需要返回 值，你必须将它声明为一个子程序。如果有返回值，你就必须声明它为函数。

这里是子程序的声明语法：

```
[Attributes][ProcedureModifier] Sub Identifier [( [ FormalParameterList ] )]
    ' Method body
End Sub
```

这里是函数的声明语法：

```
[Attributes][ProcedureModifier] Function Identifier
[( [ FormalParameterList ] )]
[ As [Attributes] TypeName]
```

```
' Method body  
End Function
```

注意，函数具有一个额外的 `As` 子句来指定返回值及其类型。正如本章前面“Option Strict 和 Option Explicit”一节所说，如果 Option Strict 被设置为“ON”，你就必须指定返回值和参数的类型。如果被设置为“OFF”，并且没有指定函数的返回类型，那么系统会自动返回一个 `Object` 类型的值。

从方法中返回值

从一个函数返回值有两种不同的方法。你可以使用 `Return` 语句(在 Visual Basic.NET 中新增的)：

```
Public Function ReturnValue1() As Integer  
    Dim intReturnValue As Integer = 1  
    Return intReturnValue  
End Function
```

你可以通过设置函数名的值来从一个方法中返回值：

```
Public Function ReturnValue2() As Integer  
    Dim intReturnValue As Integer = 2  
    ReturnValue2 = intReturnValue  
End Function
```

我推荐使用 `Return` 语句从 Visual Basic.NET 方法中返回值，因为这比使用函数名来返回值更直观。这可以使别人更容易阅读你的代码并明白发生了什么。

关于方法语法更详细的内容请参阅第五章“C# 和 Visual Basic.NET 中的面向对象”。

声明变量

方法中定义的变量被认为是局部变量（不存在其他类型的变量）。它们仅在定义它们的方法中可见。这一点与 Visual FoxPro 不同，在 Visual FoxPro 中允许定义局部变量、私有变量或全局变量。



在 Visual Basic.NET 和 C# 中仅可以使用局部变量并不是一种束缚。良好的编程技法告诉我们应该声明方法变量为局部。这比在 Visual FoxPro 中使用私有变量要好的多，如果你想让一个 `sub` 方法可以访问一个变量，那么你需要明确的以参数方式传递。

要声明一个局部变量，需要使用 Dim 语句，后跟变量名、As 关键字以及变量类型：

```
Dim Count As Integer ' 声明一个名为 Count 的 integer 型变量
Count = 5             ' 为变量赋值
```

你也可以在单独一行语句中声明变量并赋值：

```
Dim Height As Integer = 6 ' 声明一个 int 型变量并赋值
```

如果你想在一行中声明多个同类型的变量，你可以使用逗号来分割它们：

```
Dim Top, Left As Integer
```

当你在一行代码中声明多个变量时，你不能像在 C# 中那样指定默认值。然而，在 Visual Basic.NET 中允许你做一些在 C# 中无法做到的事——在一行代码中声明不同类型的变量：

```
Dim Bottom As Integer, FieldName As String
```

成员变量

正如先前提到的那样，在方法中声明的变量是局部变量。然而，在类层次声明的变量是作为成员变量或实例变量：

```
Public Class MemberAndLocalVariables

    Private Test1 As String = "Test1" ' 声明一个成员变量

    Public Sub DeclareVariables()
        Dim Test2 As String = "Test2" ' 声明一个局部变量

        MessageBox.Show("Displaying " & Test1, 'Member Variable')
        MessageBox.Show("Displaying " & Test2, 'Local Variable')
    End Sub
End Class
```

首先，成员变量看上去好象和 Visual FoxPro 中的属性相似，但是它们是不同的。在 VB.NET 类中，除了有成员变量外，也可以有属性，这些在第五章“C# 和 Visual Basic.NET 中面向对象”来讨论。目前，仅需要认为成员变量是在类层次声明的变量，并且可被类内部的所有方法访问即可。

另外一点需要注意的是，你可以用引用局部变量的方式来引用成员变量。你并不需要像在 Visual FoxPro 中那样使用“this”。在前面的例子中，你可以引用 Test1 字段(译者注：原文如此，下同。另，VB.NET 中的成员变量和 C#中的字段为同一含义)为“Me.Test1”。Visual Basic .NET 中的关键字 Me 等同于 Visual FoxPro 中的“This”。

成员变量修饰符

你可以对成员变量应用五种不同的修饰符（表 1）。

表 1. 字段修饰符允许你指定字段的可视性

修饰符	修饰符中文含义 (译者添加)	描述
public	公共	访问不受限制
friend	朋友	访问被限制在当前项目
protected	保护	访问被限制在所在类或类的子类
protected friend	保护 朋友	访问被限制在当前项目或类的子类
private	私有	访问被限制在所在的类

通常情况下最好指定成员变量为 private 。为了加强这个约定，如果你在声明变量时未使用修饰符，Visual Basic .NET 默认就把它作为 private 。如果你想让其他的类可以访问这个字段，想比于使用修饰符而言，最好创建一个 protected 或 public 属性（详细信息，请参见第五章“C# 和 Visual Basic .NET 中的面向对象”中的“属性”一节）。

值类型和引用类型

在第三章“C#入门”的“值类型和引用类型”一节已经讨论了在 .NET Framework 中两种类型的区别。这种不同是针对所有 .NET 语言的一包括 Visual Basic .NET. 详细资料参看前一章的同样小节。

字符串类型

尽管使用命令控制字符串时，字符串看上去很像值类型，但实际上它是引用类型。
你可以通过使用在双引号内的一串字符来给字符型变量赋值：

```
Dim Caption As String = ".NET for VFP Developers"
```

和 Visual FoxPro 不同，你不能使用单引号或方括号来界定字符串（在 VB.NET 中，单引号是注释专用的）。在 Visual Basic .NET 中，你可以使用 & 或 + 号（和 VFP 中一样）来连接字符串：

```
MessageBox.Show(str1 & " " & str2, "Concatenate with &")
MessageBox.Show(str1 + " " + str2, "Concatenate with +")
```

接下来的问题就是“我如何才能创建一个包含双引号的字符串，例如“Satchmo” Armstrong”？你可以很容易的通过使用连续两个双引号来解决这个问题：

```
Dim ManlyMen As String = "Louis ""Satchmo"" Armstrong"
```

Visual Basic .NET 并不能识别字符串中任何其他特别的字符转义（例如 C# 中的“\n”表示新的一行）。详细信息，参看第三章“C#入门”中的“字符串类型”一节。

相比于使用字符转义，VB .NET 提供了一系列枚举值（表 2）（译者注：这里的“枚举值”实际是指“输出和显示常数”）。详细信息，参看本章的“枚举”一节。

表 2. VB .NET 提供了可包含在字符串中的枚举值。本表包含了部分枚举值。

枚举	描述
CrLf	回车 / 换行
Cr	回车
Lf	换行
NewLine	换行字符(等同于 CrLf)
Tab	Tab
FormFeed	换页

如果你导入 Microsoft.VisualBasic.ControlChars 命名空间，你就可以很简单的使用表 2 中“枚举”列中的值。例如：

```
MessageBox.Show("Line 1" + NewLine + "Line 2", "New Line Demo")
```

作为引用类型的字符串

.NET 运行时在处理字符串时和其他引用类型有一些不同。从实际应用的角度来说，这只会造成一个问题，也就是说在将一些字符串连接到一起时（例如在一个循环中），因为运行时可以在堆中过度增生大量的字符串数据。在这样的情况下，你最好使用 .NET 里的 StringBuilder 类而不是使用 Strings 。

StringBuilder 类具有一个 Append 方法，你可以使用它来连接字符串：

```
Dim StrBuilder As New StringBuilder()  
StrBuilder.Append("Texas, ")  
StrBuilder.Append("New Mexico, ")  
StrBuilder.Append("Colorado")  
MessageBox.Show(StrBuilder.ToString(), "StringBuilder")
```

枚举

Visual Basic .NET 允许你定义自己的复杂的值类型。这些值类型就包括枚举和结构(参阅第五章“C# 和 VB.NET 中的面向对象”)。

Enumerations 是 Visual Basic .NET 中一个很方便的特性，它允许你可以将相关的常数做为一组。关于枚举特性的更多信息，请参看第三章“C#入门”中的“枚举”一节。

这里是在 Visual Basic .NET 中声明一个枚举的例子：

```
Public Enum Months As Integer  
    January = 1  
    February = 2  
    March = 3  
    April = 4  
    May = 5  
    June = 6  
    July = 7  
    August = 8  
    September = 9  
    October = 10  
    November = 11  
    December = 12  
End Enum
```

你可以通过名字来引用枚举并指定你所想指定的成员。编译器会将它转换为相关联的 integer 值：

```
Public Class EnumerationTest  
    Public Function GetBookPubMonth() As Integer  
        ' 访问枚举  
        Dim PubMonth As Months = Months.September  
        Return PubMonth  
    End Function  
End Class
```

尽管递增的整数是枚举的常见形式，但是枚举常量值也可以是任意的整数，包括负数，并且可以是任意的顺序。例如，下面的枚举按照字母的顺序列出了三个人的姓氏，这就导致他们的帆船运行水平 (windsurfing ability) 的等级并不是按顺序来排序：

```
Public Enum WindSurfingAbility As Integer
    Egger = 9
    Fabulous_Ferguson = 8
    McNeish = -3
    Strahl = 10
End Enum
```

尽管枚举在默认情况下是一个 Integer 型，你也可以指定它们为 Byte、Integer、Long 或 Short 。例如：

```
Public Enum HairLength As Long
```

As Long 指定 HairLength 类是源于类型 “Long” 。关于派生和继承的详细信息请参看第五章 “C# 和 Visual Basic .NET 中的面向对象”。

数组

VB.NET 中所有的数组都是从 0 开始的，这也就是说数组的第一个元素是 0，这比在 Visual FoxPro 中以 1 开始要好。此外，所有的数组中值的类型必须相同。你不能在一个单一的数组中混合使用 integers、strings、dates 等等类型。

声明数组

在 Visual Basic .NET 中，数组的声明和变量的声明是相同的，除了需要在数组名后加上括号：

```
Dim KidsAges() As Integer
```

你也可以同时声明数组及其大小：

```
Dim KidsNames(2) As String
```

如果你将这行 VB.NET 代码和 C# 代码作比较，你会注意到声明数组大小的方式的不同。在 C# 中，我设置数组的大小为 3。这里，看上去好象我声明数组的大小为 2——然而，差别并不在此。为了向后兼容 Visual Basic 6，在 Visual Basic .NET 中声明数组的方式和其他 .NET 语言的方式有细微的差别。

当你在 VB.NET 中声明一个数组的大小时，你就定义了数组的上限，这比定义有多少个元素要好的多。例如，这里声明的 KidsNames 数组，它具有三个元素。因为数组是从 0 开

始的，所以第一个元素是 0，第二个元素是 1，最后一个元素是 2。在 VB.NET 中，我使用这样的上限元素值来指定数组的大小。

尽管这很不标准，但是这是使微软的 VB.NET 引擎兼容 Visual Basic 6 的让步。

数组的数组空间重分配（Redimensioning arrays）

和 C# 不同，你可以使用 VB.NET 的 ReDim 语句来更改任意数组的大小。下面的代码展示了如何分配和重新分配一个数组：

```
Dim ScaryAnimals() As String = {"Lions", "Tigers"}  
ReDim Preserve ScaryAnimals(2)
```

代码的第一行已经分配数组的大小为 2，因为我已经指定了两个字符元素 (Lions 和 Tigers)。第二行代码重分配了数组以便容纳三个元素 (谨记 VB.NET 的特性—你可以指定最高的元素值而不是元素的个数)。注意这里使用的关键字 Preserve。如果你不增加这个关键字，VB.NET 会在重分配数组空间时清空其他的数组元素。另一点需要注意的是，ReDim 只能用于已有的数组—你不能使用 ReDim 初始化一个新的数组。同样，你可以使用 ReDim 来仅仅更改数组维数；但是你不能更改维数的编号。

在数组中存储值

你可以在声明数组时初始化数组元素：

```
Dim KidsMiddleNames() As String = {"Christopher", "Mark", "James"}
```

或者，你可以在声明数组后在数组元素中存储值：

```
Dim KidsNames(2) As String  
KidsNames(0) = "Jordan"  
KidsNames(1) = "Timothy"  
KidsNames(2) = "Alexander"
```

数组排序

你可以通过调用数组类的 Sort 方法很容易的对数组元素进行排序：

```
Array.Sort(KidsNames)
```

你也可以通过调用数组的 `Reverse` 方法来进行逆序排序：

```
Array.Reverse(KidsNames)
```

多维数组

在 Visual FoxPro 中，你可以创建一维和两维数组。Visual Basic .NET 允许你创建多维数组—数组可以是三维或更多维。

多维数组可以是矩形数组（每行都具有相同的列）或交错数组（每行可以有不同的列）。

定义多维矩形数组

在 VB.NET 中，声明一个多维数组时，你需要在声明的数组的括号内增加一个或多个逗号。你增加的每个逗号都是额外的数组维数。这里定义一个二维矩形数组：

```
Dim ArrayKids(,) As String = {{ "Jordan", "McNeish"}, _  
                                { "Timothy", "McNeish"}, _  
                                { "Alexander", "McNeish" }}  
Defining jagged arrays
```

交错数组是一个“数组的数组”。它也是多维数组，但是每行具有不同的列数。当定义一个交错数组时，通过在每一维后面增加额外的括号来实现。在下面的例子中，交错数组的第一维具有三个元素，第二维具有和其他维不同的尺寸。第一行有三列，第二行有两列，第三行仅有一列：

```
Dim Musicians()() As String = {New String() { "Stevie", "Ray", "Vaughn"}, _  
                                New String() { "Jimi", "Hendrix"}, _  
                                New String() { "Bono" }}  
  
' Show all three names of the first musician  
MessageBox.Show(Musicians(0)(0) + " " & _  
Musicians(0)(1) & " " & _  
Musicians(0)(2), "Jagged Arrays")
```

类型转换

当使用强类型语言时，必须考虑不同数据类型间的转换。VB.NET 提供了隐式转换和显式转换。



VB.NET 的数据类型列表以及 VB.NET、C# 和 Visual FoxPro 的数据类型比较, 请参看“附录 C-数据类型比较”。

隐式类型转换

Visual Basic .NET 可以隐式的转换一些数据类型。下面的示例中, VB.NET 转换一个 short 值到一个 double 值:

```
Dim x As Short = 10
Dim y As Integer = x
Dim z As Double = y
```

表 3 列出了可进行 (自动) 隐式转换的数据类型。

表 3. Visual Basic .NET 中可隐式转换的数据类型。

从	到
byte	ushort, short, uint, int, ulong, long, float, double, decimal
sbyte	short, int, long, float, double, decimal
short	int, long, float, double, decimal
ushort	uint, int, ulong, long, float, double, decimal
char	ushort, uint, int, ulong, long, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
float	double

显式类型转换

如果编译器拒绝隐式转换数据类型, 你可以自己显式的转换它。Visual Basic .NET 提供了一些方式来完成这样的操作。

第一个方法使用 Visual Basic .NET 的转换关键字 (**表 4**)

表 4. Visual Basic .NET 提供的用于数据类型转换的关键字。

关键字	描述
CBool	转换任何数值类型、String、Object 到 Boolean
CByte	转换任何数值类型、任何枚举类型、Boolean、String、Object 到 Byte
CChar	转换 String、Object 到 Char
CDate	转换 String、Object 到 Date
Cdbl	转换任何数值类型、Boolean、String、Object 到 Double
CDec	转换任何数值类型、Boolean、String、Object 到 Decimal
CInt	转换任何数值类型、Boolean、String、Object 到 Integer
CLng	转换任何数值类型、Boolean、String、Object 到 Long
CObj	转换任意类型到 Object
CShort	转换任何数值类型、Boolean、String、Object 到 Short
CSng	转换任何数值类型、Boolean、String、Object 到 Single
CStr	转换任意数值类型、Boolean、Char、Char() 数组、Date、Object 到 String

例如，要转换 Integer 到 Byte，你可以使用 CByte 关键字：

```
Dim x As Integer = 20
Dim y As Byte = CByte(x)
```

你也可以使用 System.Convert 类来完成显式转换。Convert 类有一些不同的转换方法来
来完成转换（表 5）。例如，要完成上面示例中同样的转换，你可以这样做：

```
Dim x As Integer = 20
Dim y As Byte = Convert.ToByte(x)
```



使用 System.Convert 将比使用转换函数体验到更好的性能。这是因为 Convert 类具有重载方法来明确指定所转换的类型。然而，类型转换函数例如 CStr 和 CDate 都要考虑到当前系统的设置，所以，如果这个功能对你来说是重要的话，它或许就会影响性能。

表 5. Convert 类包含一些方法来用于类型转换。这里是部分列表。

关键字	描述
ToBoolean	转换一个值到 Boolean
ToByte	转换一个值到 8-bit 无符号 integer
ToChar	转换一个值到 Unicode character
DateTime	转换一个值到 DateTime
ToDecimal	转换一个值到 Decimal
ToDouble	转换一个值到 Double
ToInt16	转换一个值到 16-bit 有符号 Integer
ToInt32	转换一个值到 32-bit 有符号 Integer
ToInt64	转换一个值到 64-bit 有符号 integer
ToSByte	转换一个值到 8-bit signed integer
ToSingle	转换一个值到单精度浮点
ToUInt16	转换一个值到 16-bit 无符号 integer
ToUInt32	转换一个值到 32-bit 无符号 integer
ToUInt64	转换一个值到 64-bit 无符号 integer

CType 函数

上一节讨论的转换关键字和 System.Convert 类对于转换基本值类型是很好的，但是如果你想转换更复杂的对象，也许你需要一些其他的东西。它也许是 CType 函数。

CType 是一个 Visual Basic .NET 函数，它允许你转换一个表达式到 data、object、structure、class 或 interface。CType 函数接收两个参数。第一个参数是要被转换的表达式。通常情况下第一个参数包含一个对象引用。第二个参数是目标类型。例如，下面的代码转换一个 integer 到 decimal：

```
Dim x As Integer = 100
Dim y As Decimal = CType(x, Decimal)
```

转换到字符串

如果你需要转换一个值到字符串，Object 类有一个 ToString 方法，它被 .NET Framework 中的所有类继承。这个方法返回对象的字符串形式。所以，就可以像下面这样，得到 Integer 的字符串形式：

```
Dim i As Integer = 10
Dim s As String = i.ToString()
```

使用 Val 转换字符串

如果你需要转换字符串到值类型，你可以使用“显式类型转换”一节中所列的转换关键字。此外，你还可以使用 Val 函数，它更多的是用于转换字符串到数值形式—它可以让你转换包含数字和字符的字符串。

Val() 遇到数字中不可识别的部分时停止转换。在下面的例子中，字符串为“567 Test”，使用 Val() 函数转换的结果是 567：

```
Dim i As Double
Dim z As String = "567 Test"
i = Val(z)      ' 返回 567
```

Visual Basic .NET 中的装箱和取消装箱

如前所述，装箱是转换值类型到引用类型的过程。取消装箱是一个相反的过程。

例如，当你将一个 integer 型的值存储到一个对象时装箱操作就会在后台自动进行：

```
Dim x As Integer
Dim y As Object = x
```

对于一个值被装箱时在内存中到底会发生什么，请参看第三章“C#入门”中的“装箱和取消装箱”一节。

在 Visual Basic .NET 中，你不能使用像 C# 中那样的方式明确的取消装箱。



因为 Visual Basic .NET 不能明确的取消装箱，所以它需要依赖 Microsoft.VisualBasic.Helpers 命名空间中的 helper 函数，它远不及 C# 中明确取消装箱有效。

typeof 操作符

你可以通过使用 typeof 操作符来确定一个对象是否是一个特定的类型。当一个方法接收一个常规对象参数并且希望确定它是否是特定的类型时非常有用。例如，下面的方法接收一个参数。然后检查参数的类型是否是 MyClass1：

```
Public Class TypeOfOperatorDemo
    Public Sub CheckClass(ByVal TestObject As Object)
        If TypeOf TestObject Is MyClass1 Then
            MessageBox.Show("TestObject is MyClass1", "typeof operator")
        Else
        
```

```
        MessageBox.Show("TestObject is not MyClass1", "TypeOf operator")
    End If
End Sub
End Class
```

If 语句

Visual Basic .NET 的 If 语句和 Visual FoxPro 的 IF..ENDIF 语句很相似。如果被执行的表达式为真，那么后面的代码将被执行。可选的，你也可以增加一个 Else 语句到其中，如果条件表达式为假，则执行其中的代码。你也可以嵌套 If 语句。

```
Dim HavingFun As Boolean = True
If HavingFun Then
    MessageBox.Show("We are having fun!", "if demo")
Else
    MessageBox.Show("We are NOT having fun!", "if demo")
End If
```

如果你想在 Else 中检测一个额外的条件，请使用 ElseIf：

```
Dim MyInteger As Integer = 99
If MyInteger < 10 Then
    MessageBox.Show("MyInteger is less than 10", "If...End If")
ElseIf MyInteger < 100 Then
    MessageBox.Show("MyInteger is more than 9, but less than 100", "If...End If")
End If
```

Select...Case 语句

Visual Basic .NET 的 Select...Case 语句等效于 Visual FoxPro 的 DO CASE 语句。基于表达式的值，其中一个 Case 语句被执行。你可以指定一个 Case Else，它的作用和 VFP 的 OTHERWISE 语句相同。如果控制表达式与任何 case 都不匹配，控制就会被传递到 Case Else。如果不存在 Case Else 语句，控制就会被传递到 End Select 后的第一条语句。

```
Dim TestValue As Integer = 3
Select Case TestValue
    Case 1
        MessageBox.Show("TestValue = 1", "Select...Case")
    Case 2
        MessageBox.Show("TestValue = 2", "Select...Case")
    Case 3, 4
        MessageBox.Show("TestValue = 3 or 4", "Select...Case")
    Case Else
        MessageBox.Show("TestValue is not 1, 2, 3, or 4", "Select...Case")
End Select
```

```
End Select
```

和 C# 不同的是，你不需要在每个分支中的末尾增加一个 `break` 语句，这个特性很不错！

For...Next 循环

Visual Basic .NET 的 `For...Next` 循环等效于 Visual FoxPro 的 `FOR...ENDFOR` 命令。当特定的条件为真时它会重复循环。

在 `For...Next` 循环中，循环执行计数器所指定的次数。`Next` 语句常用于计数器值的递增或递减。

```
Dim i As Integer
Dim Message As String

For i = 0 To 3
    Message += "Message " + i.ToString() + NewLine
Next i
```

如果退出 `For...Next` 循环，你可以在任意位置使用 `Exit For` 语句来达到目的。

While 循环

`While` 循环等效于 Visual FoxPro 的 `DO...WHILE` 循环。它是预检测循环，也就是说，如果条件表达式为假，循环根本就不会执行。`While` 循环常用于未知次数的循环执行代码块。

```
Public Sub WhileLoopDemo()
    Dim Condition As Boolean = True
    While Condition
        Dim i As Integer = 1
        MessageBox.Show("While loop count " + i.ToString(), _
            "While loop")
        i += 1
        Condition = False
    End While
End Sub
```

你可以通过在任意位置使用 `Exit While` 语句退出循环。

Do 循环

Visual Basic.NET 的 Do 循环在表达式为真或直到一个表达式为真时重复执行代码块。如果你使用 Do While 结构，那么当表达式为真时执行循环。如果你使用的是 Do Until 结构，循环在表达式为真之前执行。

While 和 Until 关键字可被用于循环的开始和结束语句。如果用于循环的开始，在循环执行前检查表达式。如果被放置为末尾，就像下面的例子一样，那么表达式在循环开始执行后检测。这和 Visual FoxPro 的 DO WHILE 循环在开始检查表达式是不同的。

```
Dim Condition As Boolean = False
Do
    Dim i As Integer = 1
    MessageBox.Show("Do While loop count " + i.ToString(), _
        "Do...While loop")
    i += 1
Loop While Condition
```

你可以在任意位置使用 Exit Do 语句退出 Do 循环。

For Each 循环

Visual Basic.NET 的 For Each 循环针对数组的每个元素或者集合中的每一项执行一组语句。它等效于 Visual FoxPro 的 FOR EACH 命令。

```
Dim VersionList, Version As String
Dim VSNetVersions() As String = {"Standard", "Professional", _
    "Enterprise Developer", _
    "Enterprise Architect"}
For Each Version In VSNetVersions
    VersionList += "VS .NET " + Version + NewLine
Next
MessageBox.Show(VersionList, "ForEach loop")
```

你可以在任意位置使用 Exit For 语句来退出 For Each 循环。

With 语句

Visual Basic.NET 的 With 语句等效于 Visual FoxPro 的 WITH...ENDWITH 命令。你可以在 With 语句的开始指定一个对象引用，然后在语句中通过指定对象名来访问对象的属性和方法。在 With 语句中执行的任意以句点开始的表达式都会被当作具有对象名前缀。例

如：

```
Dim ReturnDemo As New ReturnValuesDemo()  
Dim ReturnValue As Integer  
With ReturnDemo  
    ReturnValue = .ReturnValue1()  
    ReturnValue = .ReturnValue2()  
End With
```

XML 文档化工具

尽管 XML 文档化并不存在于 Visual Studio .NET for VB.NET，在 VS.NET 发布几个月后，微软单独发布了一个工具，以允许你创建 VB.NET 项目的 XML 文档。要获得此工具，你可以在 VB.NET 的“GotDotNet”站点 (<http://www.gotdotnet.com/team/vb>) 下载。（译者注：自 VS2005 开始，VS.NET 内置 For VB.NET 的 XML 文档化工具。使用方法类似于 C#，具体参看 MSDN。）

这个工具和用于 C# 的 XML 文档化工具有一些不同。对于 C# XML 文档化，你需要直接在代码中输入 XML 注释。这些注释被编译进程序集，并且可以在对象浏览器中借助智能感应来查看。此外，VS.NET 可以从这些 XML 注释自动生成 XML 注释 Web 页（详细信息参看第三章“C#入门”）。

对于 VB.NET XML 文档化工具，XML 注释并不存储在源代码中，而是在一个单独的 XML 文件中。像 C# 一样，可以很容易从这些 VB.NET XML 注释创建注释 Web 页。此外，因为注释并不存储在源代码中，这就意味着：

1. 你可以在源代码中不放置任何注释（不推荐）。
2. 或者，你可以复制注释——一份在源代码中，一份在 XML 文件中。

我认为这是此工具的一个严重的限制。为了你和他人的利益，你应该将注释放置在源代码中，否则别人就不得不检查你的代码。

在下载了 VB.NET XML 文档化工具之后，你可以双击“XML Documentation Tool.exe”以启动它。**图 4** 是运行时的对话框。

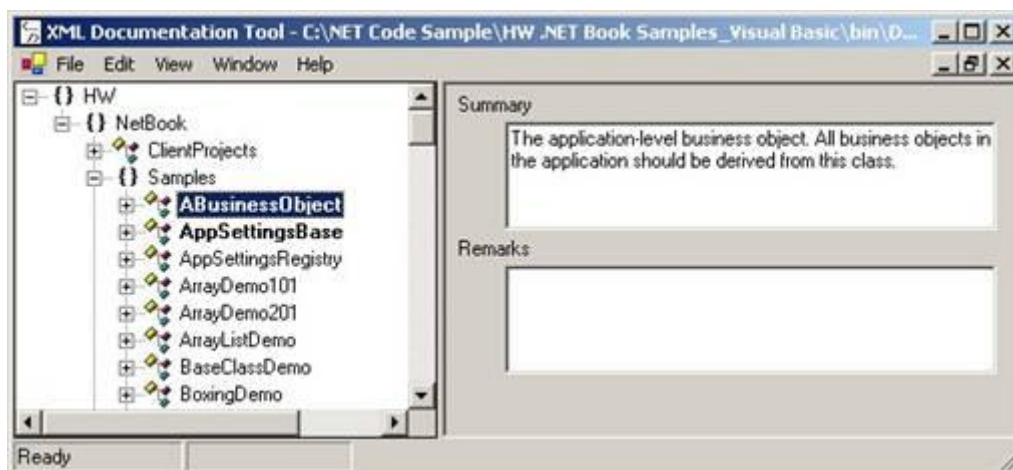


图 4. XML Documentation Tool for Visual Basic .NET 允许开发者创建 XML 注释，这样，其他开发者就不需要查看你的源代码。

相关的 ReadMe.htm 文件介绍了如何使用这个工具来创建 XML 注释。

总结

Visual Basic .NET 和 Visual Basic 6 有相当大的差距，并且它已经是一个强大的、完全面向对象的语言。这种差别就像 FoxPro 2.6 和 Visual FoxPro 的差别（甚至比这个差别还大）。所有的 .NET 语言的语法都很像，正如你对 Visual Basic 的期待，它具有许多方便的特性，这使它可以作为你所使用的程序语言的选择。由于微软一直强调 Visual Basic 的易用性，可以预见在将来这些增强将变的更容易使用。