HZ BOOKS
微软公司核心技术书库
Microsoft Press

Inside C#
Second Edition

C# 技术揭秘

(美) Tom Archer
Andrew Whitechapel 著
马朝晖 等译

Microsoft .net

机械工业出版社
China Machine Press

微软公司核心技术书库

Inside C#
Second Edition

C#技术揭秘

(美) Tom Archer
Andrew Whitechapel 著

马朝晖 等译

Microsoft .net

机械工业出版社
China Machine Press

本书详细介绍了C#的特点、新增功能以及它与C和C++的异同，是一本真正揭示C#技术内幕的书籍。

本书既适合C#的初学者阅读，也适合已经使用过C#语言进行编程并希望掌握高级功能的开发人员参考。

Tom Archer, Andrew Whitechapel: Inside C#, Second Edition (ISBN 0-7356-1648-5).

版权登记号：图字：01-2002-5523

图书在版编目（CIP）数据

C#技术揭秘/（美）阿奇（Archer, T.），（美）怀特切普（Whitechapel, A.）著；马朝晖等译．-北京：机械工业出版社，2003.7

（微软公司核心技术书库）

书名原文：Inside C#, Second Edition

ISBN 7-111-12257-7

Ⅰ.C… Ⅱ.①阿…②怀…③马… Ⅲ.C语言-程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2003）第043717号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：周 睿

北京瑞德印刷有限公司印刷·新华书店北京发行所发行

2003年7月第1版第1次印刷

787mm×1092mm 1/16 · 38.75印张

印数：0 001—4 000册

定价：69.00元（附光盘）

译 者 序

刚听说 C# 时，有人告诉我 C# 是一种简化版的C++。但是了解了 C# 之后，特别是在翻译完本书之后，我发现这种说法相当不准确。有这种说法似乎是因为 C# 限制了（并没有取消）对指针的使用，并且取消了一些最难于理解的C++概念。依我之见，C# 是C++语言向 Java 妥协的标志。这种说法也许有点儿夸张，但是你看过本书就会或多或少地体会到我的意思。书中多处提到 Java 程序员可以猜出代码的效果，而C++程序员会吃惊。实际情况确实如此，如果你同时掌握 Java 和C++，那么当你看到某些 C# 功能时你会情不自禁地说："这太 Java 了。"所以，我甚至可以建议：如果你只具有C++背景，那么不妨先学习 Java，再回头学习 C#。

好在，我几年前已经从C++转向了 Java，所以我觉得学习 C# 并不难。但是我想C++程序员会很不习惯，因此我自作主张将书中提到的 C# 与C++的差异收集起来，供C++程序员参考（见下表）；其中主要列出了改变的地方，增加的功能总结得不全。这些改变的影响各有不同，有的非常好，有的可能会造成麻烦，有的可以不用理会；我想你有能力判断各种改变的影响属于什么性质。但是，增加的功能基本上都很出色。

C# 与C++ 的差异	相关章节
在 C# 中，所有方法都是在类定义中定义的	1.2.1 节
所有对象都隐式地派生自一个基类，System.Object	2.1 节 2.6.2 节
在类的末尾不必放分号，但是也可以加分号	3.1 节
除非希望成员采用默认的访问修饰符 private，否则必须为它指定一个访问修饰符	3.3 节
在哪个类中定义 Main 方法对于 C# 编译器并无影响，而且你选择的类也不影响编译的次序。这与C++不同，在C++中在建立应用程序时必须密切关注依赖性	3.4 节
C# 中的命令行参数数组并不将应用程序名作为它的第一个条目	3.4.1 节
C# 和C++中的对象实例化过程不一样	3.5 节
所有方法都被封装在某个类或结构中；C# 不支持全局方法	第 4 章
即使函数的其他重载版本存在于基类中而不是在当前类中，重载方法仍然被认为是重载	4.3.2 节
在 C# 中，数组是以 System.Array 类作为基类定义的对象	5.2 节
在 C# 中，声明数组的语法是在类型和变量名之间放一对空的方括号	5.2.1 节
C# 增加了属性	第 6 章
C# 没有从数值数据到字符数据的隐式转换；但是可以进行显式转换	8.4 节
当使用关系操作符比较两个对象时，C# 编译器并不比较对象的内容。相反，它比较这两个对象的地址	8.6 节
C# 在进行对象赋值时是对引用进行赋值	8.7 节
if 语句中被运算的表达式必须产生一个布尔值	9.1.1 节
C# 不支持在 switch 语句中省略 break 语句时的穿透效果	9.1.2 节
能够在 C# 中重载的操作符不如C++中多	13.1.2 节
C# 增加了 XML 注释	第 15 章
C# 中没有 delete 操作符，也没有析构器，析构器由 Finalize 方法替代	20.2 节
只能在标明 unsafe（不安全）的代码中使用指针	20.8 节
C# 不支持 . * 和 - > * 操作符	20.9.2 节

Microsoft 在 C# 中所做的主要改进在以下几个方面：

- 首先是增强了 C 语言面向对象的程度。C++ 在近几年似乎丧失了“面向对象”的荣誉，而降成了“基于对象的”。的确，C# 在这方面大有改进。单根的对象层次结构，所有对象都最终派生自 System.Object，取消了全局方法，等等；这些都提高了面向对象的程度。
- 提高代码的优雅程度。准确地说，主要是提高客户代码的优雅程度。第 5 章开头说的话非常正确，“系统被简化到一定程度之后，进一步简化的惟一方法就是转移复杂性，将最复杂的工作转移到适合处理它的部分中去。”对于语言来说，就是让类的编写者付起更大的责任，让类的客户更轻松。优雅指的是采用自然而且简便的方式。像特性这样的功能确实很优雅。当然，转移复杂性并不能减小复杂性，客户轻松了，类的编写者就费劲了。但这种原则是正确的，毕竟，最了解类的人是类的编写者。在崇尚可重用性的今天，迫使客户详细研究类，还要记住一堆特殊规则，对于重用是很不利的。力求优雅和自然，这可能是 C# 给我的最大启示了，我们在自己的程序设计工作中应该时刻想着客户。打个比方，你要是买了一台电视机，你肯定不愿意查看它的电路图，然后拿起螺丝刀（甚至是电烙铁）在机器里折腾一小时后才能看电视。你希望插上电源和天线，一按遥控器就行了，而且希望遥控器按钮的功能一目了然。这就是自然。
- 提供了 .NET 语言之间的互操作性。这主要是通过 MSIL 和通用类型系统实现的。MSIL 借用了 Java 字节代码的中间语言概念，但仅仅是借用了概念，它还是局限于 Windows 的范围。而通用类型系统算是一个成果，而不是成就；既然 Microsoft 控制着 VC、VB 等语言，那么这些语言的数据类型不能完全兼容是很荒谬的，现在 Microsoft 做了早就应该做的事。

这本书的原文中许多地方非常晦涩，用的词汇也常常很偏，翻译起来很难，我们用了很长时间才完成翻译工作。我们在翻译过程中不得不进行了一定程度的意译，我们确信已经尽力保持了原文的意思，而且纠正了原书的一些明显错误。如果你有幸得到本书的原文，希望你不要纯粹从字面评价本译稿（这对程序员是没有意义的）。当然，因为我们的水平有限，而且不可能保证没有疏漏，所以欢迎你指出书中的实质性错误。由于原文比较晦涩，加之我们的语言水平有限，译稿可能不够流畅，请见谅。

原文虽然难读，但是作者的某些见解还是很精彩的。比如，作者指出 goto 语句并不是毫无可取之处，这的确很有道理。我和大家一样在学习编程的时候，就被告之要避免使用 goto 语句，这已经形成了一种教条。我也习惯了不使用 goto 语句，这往往是因为有些语言取消了 goto 语句；即使语言支持 goto 语句，我也不用它，这可能是因为怕别人嘲笑我无知。我说这么多，并不是想说 goto 语句是多么必要，而是感到很佩服作者挑战教条的勇气。作为程序员，我们不但应该掌握进行创造性工作的技能，而且应该具有相应的勇气。

另外，本书中使用的术语没有完全按照 .NET 中文版的译法，因为我们认为其中的一些译法并不合适。比如 assembly，.NET 中文版译为“程序集”，这个译文实在太模糊，太松散了，所以我们使用“配件”这个词。对于新术语，我们参考了 .NET 中文版、技术网站和讨论组中的译法，经过讨论之后确定译法（我们做决定的标准在很大程度上也是“是否自然”）。也许你所习惯的译文与我们的不同，但是我们希望这些不一致的地方不会影响你阅读本书；毕竟，作

为一个有实力学习技术内幕类书籍的程序员，你所关注的肯定不是文字，而是文字的含义。

公平地说，C# 对C++ 进行了很多很有价值的改进，是一门值得学习的语言。但是，有些地方确实值得商榷。比如，对象赋值的效果对于C++ 程序员是很难习惯的，而且C++ 原来的方式并不太差（至少提供了选择方案）。

C 语言系列的这种发展方向是否是好现象？答案肯定是见仁见智的。可以肯定的是，这种发展至少在一定程度上是正确的，因为 Java 确实具有 C 所缺乏的一些特质。问题只是这种同化的程度。从好的方面来说，C 和 Java 的同化降低了程序员同时掌握这两种语言的难度，或者说降低了同时使用它们时出错的可能性。从消极的方面来说，这减小了程序员进行选择的空间；试想一下，如果最终 C 和 Java 成了一样的语言（可能只剩下语法的细节差异），那么程序员就不用选择了。在这里我要声明：我很喜欢 Java，但是不希望 C 变成 Java。我希望 C 和 Java 保持一定的距离，保持各自特有的优点，以便在不同场合选用。一般情况下，在希望得到最佳性能时 C 是最合适的，这主要是由于 C 直接使用指针；在希望得到最好的可移植性时 Java 是最好的。对于编程来说，C++ 和 Java 之间的主要差异就在于对指针的使用。近年来，由于 Java 的流行，指针大受非议，但是在有些场合指针确实很方便；而且对于指针不安全等问题，C++ 并不是没有解决方案，“收益越大，风险越高”是很自然的，我认为不应该“因噎废食”。如果 C 语言完全放弃指针，实在是可惜，幸好 C# 只是限制了指针的使用。Microsoft 这么做的原因是很值得思考的，它们可能也不舍得放弃指针，但是又要向 Java 风气做妥协。

好了，我说得太多了，这些只是我的个人意见。作为有经验的程序员，你一定会有自己的看法和好恶。和以往一样，每当出现一种新语言时，我们只能研究它，决定是否采用它，至于它应该如何如何我们是无权决定的。只要我们能够利用它的长处编写出更好的程序，或者更快地编写出程序就可以了。衷心希望你能够通过本书掌握 C#，利用它编写出精品程序，同时享受编程的乐趣。

本书主要由马朝晖、陈美红翻译，参与翻译、录入、审校的还有刘丽珍、王建芬、杨帆、邹辉、潘浩、楼涵、董小蕾、王悦、李军、罗伟、鲍广华、瞿兰、陆明、宋丽、杨立军、李鸣、马晓云、成荣光等。

序　一

我在 Microsoft 的全部工作就是致力于改进开发人员的工作方式，特别是提高开发人员的生产效率。我的工作涉及许多产品和技术，但是我从来没有像现在这样因为所做的工作大大改进了开发人员的工作方式而极度兴奋。Microsoft.NET 提供的技术其广度和深度是令人吃惊的。我们提供了一种非常新颖的语言，打破了过去将开发人员隔离在各种语言范围内的壁垒，而且使 Web 站点能够满足用户的需求。这几点都很有趣，然而把它们结合起来才真正令人兴奋。

我们来看看 .NET 的关键构件和一些相关的技术：

- **C#，一种新语言**　C# 是 C 和C++ 语言系列中第一种面向组件的语言。它是一种从 C 和C++ 发展出来的简单、现代、面向对象以及类型安全的编程语言。C# 结合了 Microsoft Visual Basic 的高生产率和C++ 的底层能力。
- **通用语言运行时环境**　高性能的通用语言运行时环境包含一个执行引擎、一个垃圾收集器、即时编译机制、一个安全系统和内容丰富的类框架（.NET 框架）。运行时环境是以支持多种语言为目标而重新设计的。
- **通用语言规范**　通用语言规范（CLS）定义了一组共有的语言功能。CLS 规定的限制非常少，这有助于建立一系列与 CLS 兼容的语言。这些语言具有双重好处：对 .NET 框架功能的完全访问以及与其他兼容语言之间丰富的互操作能力。例如，Visual Basic 可以继承 C# 类并且重定义它的虚拟方法。
- **多种可以利用运行时环境的语言**　Microsoft 提供的可以利用运行时环境的语言包括 Visual Basic、具有托管扩展的 Visual C++ 、Visual C# 和 JScript。第三方开发商还提供了许多其他语言。
- **Web 服务**　当今的 WWW 主要由单独的网站组成。用户可能需要访问多个网站才能完成某个任务，比如安排一个旅行团的旅行方案，而这些网站通常没有进行协作。下一代 Web 将由相互协作的 Web 站点的网络组成。原因很简单：相互协作的 Web 站点更能满足用户的需求。Microsoft 的 Web 服务技术支持通过基于 XML 的标准协议（这些协议既独立于语言，也独立于平台）进行通信，这可以促进 Web 站点之间的协作。许多重要的 Web 服务将基于 C# 和在 Windows 上运行的通用语言运行时环境，但是这个体系结构是开放的。
- **Visual Studio.NET**　Visual Studio.NET 将以上所有构件有机地结合在一起，它简化了使用各种编程语言编写组件、应用程序和服务的过程。

既然我已经谈到了与 C# 相关的一些重要技术，现在就来谈谈 C# 本身。开发人员在所使用的语言上已经花费了大量精力，所以一种新语言要证明它是值得开发人员学习和使用的，则应该保留了开发人员常用的一些功能，并且进行了一定的改进和革新。

保留功能

Hippocrates（古希腊医学家，被尊称为“医学之父”。——译者注）说：“习惯有两种作用：提供帮助，或者至少不带来损害。”我们在设计 C# 时充分考虑到了“不带来损害”的原则。如果一个 C 或C++ 功能出色地解决了一个问题，我们就原封不动地保留它。最重要的是，C# 从 C 和C++ 那里借用了一些核心功能，比如表达式、语句和总体语法。因为典型的程序主要是由这些功能组成的，所以 C 和C++ 开发人员会立即适应 C# 。

改进

C# 进行了许多改进，改进的地方太多了，无法在这篇简短的序言中一一列举；但是有几个可以消除常见且浪费时间的 C 和C++ 错误的修改应该提一下：

- 变量在使用之前必须被初始化，所以消除了由未初始化变量导致的错误。
- if 和 while 这样的语句要求使用布尔值，所以如果开发人员偶然使用赋值操作符（=）替代相等操作符（==），那么他会在编译时发现这个错误。
- 不允许在 switch 语句中进行穿透，所以如果开发人员偶然漏掉了 break 语句，那么他会在编译时发现这个错误。

革新

在 C# 的类型系统中进行了几个重大的革新，包括以下几点：

- C# 类型系统采用了自动的内存管理，开发人员不必再进行既费时又容易出错的手工内存管理。与大多数类型系统不同，C# 类型系统允许直接操作指针类型和对象地址（这些手工内存管理技术只允许在某些安全的上下文中使用）。
- C# 类型系统是统一的——所有东西都是对象。通过使用装箱和开箱等概念，C# 在值类型和引用类型之间建立桥梁，使任何数据都能够被作为对象对待。
- 特性、方法和事件属于基础结构。许多语言没有对特性和事件的固有支持，这在语言和相关联的框架之间造成了不必要的不匹配问题。例如，如果框架支持特性而语言不支持，那么对特性进行递增就很麻烦（例如 o.SetValue（o.GetValue() + 1））。如果语言也支持特性，操作就简单了（例如 o.Value++ ）。
- C# 支持属性，可以使用属性定义和使用关于组件的声明性信息。定义新型声明性信息的能力对于语言设计者来说是强大的工具。现在所有 C# 开发人员都拥有了这种能力。

Scott Wiltamuth

Microsoft 公司 Visual C# .NET 程序组经理

序　二

我认为序言可能是最难写的文章了。想想吧，序言的篇幅非常有限，而在其中要表达出这本书的广度和深度，还要谈到它的编写特色，这样才能激发读者的阅读兴趣。因此，我认为序言需要用一句话评价这本书的特点。对于本书，这句话就是："它深入地讨论了 C# 的细节信息。"

Scott 已经解释 C# 为什么是一种出色的语言，而我现在要花一点儿时间解释为什么阅读本书是学习 C# 的合适手段。多年以来，我使用许多语言进行过大量编程工作，我学到了两条令我难忘的真理：不可能写出"最好的"代码，只可能在了解细节的情况下写出"更好"的代码。首先来解释一下第一句话。软件工程一旦超过 100 行，可能的正确实现数目（即完成任务的不同方案数目）会急剧增长，以致于基本上不可能找出最快、最高效的解决方案。而且，人们对于什么是最好的解决方案持有不同的见解，有人会说："我的代码只有一行，非常容易维护。"而有人会说："我的代码有 300 行，但是执行速度快 30 倍。"即使不考虑这一点，结论也应该是明确的：我们作为软件开发人员的责任是建立优秀的解决方案，而不必建立最好的。

第二句话的意思是：如果不了解细节，就无法建立优秀的解决方案。不要误解我的意思，的确，没有好的算法，也不可能写出好代码。但是无论使用哪种算法，如果在实现它们时不了解平台和语言的知识，那么实现出色的解决方案肯定是不可能的。所以本书是值得阅读的，因为本书作者 Tom 深入解释了 C# 的细节，而这些细节会帮助你编写出更好的代码。

Anson Horton

Microsoft 公司 Visual C# .NET 程序组经理

前　言

为什么要写这本书

20年来，我为各种平台开发软件（从System/38，到AS/400，到OS/2和现在的Microsoft Windows），我可以毫无保留地说我已经完成了使命。向往夜晚和周末的日子成了遥远的记忆；这种向往并不是因为我想回家，而是因为在家可以毫无干扰地整日整夜工作。我感到所有值得做的事都已经完成了。这种感觉在2000年初发生了变化。

2000年，就在Microsoft在佛罗里达州奥兰多发布.NET之前的几个月，我的一位好朋友告诉我一个关于很“酷”的C#语言和运行时环境框架（名为NGWS）的秘密。尽管他谈论这个话题时充满了热情，但是我并不以为然，我回想起了那些失败的总体体系结构（比如IBM的SAA和Microsoft的DNA）。结果，我抱着怀疑的态度开始接触C#和.NET。

但是随着我逐渐了解了这种新语言和平台，我原来的热情逐渐回来了。近两年，我起床时不再带着对重复编写烦人的应用程序的恐惧，而是满心期望学习.NET的令人兴奋的新功能，而且我现在仍然在学习。实际上，我学到的关于.NET的所有新东西都打开了我的眼界。

这听起来不可思议，但是大多数程序员开始编程并不是为了钱，而只是为了学习和建立某些东西所带来的成就感和满足感。对于我来说，C#和.NET框架使我找回了已经失去的热情。所以，我写这本书是为了与你分享我重新编程所感到的乐趣。我希望你在阅读本书并开始使用C#时同样感到愉快和兴奋。

本书的这个版本有什么新东西

如果你读过本书的第一版（谢谢!），你可能会奇怪这一版为什么厚了一倍。这是因为我写第一版的时间比较早。在.NET和C#刚发布之后的一段时期，写这方面的书的作者几乎都是在边学习边写作。而且与任何软件工程一样，你每学一点儿东西，就希望马上实践它。很明显，这对于书是不现实的，所以我知道需要编写第二版来使本书更完整。下面是对这一版的新内容的概述。我想你阅读完以下内容后就会理解我为什么这么渴望编写这一版了。

- **大量的新资料，包括新添加的章节**　读者对第一版提供了大量反馈意见，包括他们自己对C#语言的各个方面的体会，以及建议应该更深入地讨论哪些语言功能。作为对这些反馈的回应，我们重新审视了最常被问到的一些语言和框架功能，而且特意增强了本书对这些主题的讨论。许多章节经历了较大的修改，而且许多新添加的章节讨论了第一版没有涉及的问题。例如，应读者的请求，新添加的“安全”一章讨论了许多安全问题，包括代码访问安全、基于角色的安全、可检验的类型安全、代码签名、加密/数据签名和隔离存储。其他新添加的章包括“字符串处理和正则表达式”、“用流进行文件I/O”、“数值处理和Math类”、“集合和对象枚举”（我最喜欢的一章）、“固定和内存管理”、“从C#应用程序使用COM”和“在非托管代码中使用.NET组件”。另外，

“使用 XML 进行文档记录”这个新章讨论了从 C# 源代码注释产生 XML 文档的能力。

- **本书是一本真正的“技术内幕”** 第二版最主要的目标是成为更加名副其实的“技术内幕”。我在第一版中犯的错误是，我只有在了解到 MSIL 可能影响我们使用 C# 编写代码的方式时才进入 MSIL 层。但是，许多读者很快指出，我们这些程序员都是很好奇的人，有时候我们希望了解某些东西只是因为我们不了解它们。在这一版的几乎每一章中，我都是先解释主题，然后深入研究编译器产生的 MSIL。如果你花点儿时间研究幕后的情况，你会为某些功能的实现方式感到吃惊（至少我很吃惊）。另外，章节的组织方式使得不关心底层 MSIL 的人可以轻易跳过 MSIL 部分。另一方面，已经了解某个功能的语法并且知道如何使用它的人不必阅读相关的解释，而是可以直接跳到讨论底层 MSIL 的部分。
- **“为什么”以及“如何”** 许多书详细说明了使用语言功能所需的语法，然后给出一系列例子。但是你我可以从联机帮助中得到语法说明。我认为一本书提供的应该不止这些。因此，在每一章中我都试图解释给定的语言功能为什么存在，以及设计它是为了解决代码中的什么问题。毕竟，如果你从来没有使用过某种功能（比如接口或委托），那么不解释为什么需要使用它而只给出语法的话，你就无法充分利用它。
- **更好且更现实的例子** 我个人喜欢比 foo/bar 型例子更现实的例子，但是在第一版的示例应用程序中只有大约 10% 算得上是现实的例子。在本书中，这个比例是大约 70%。例如，在第 13 章中提供了大量例子。前两个例子（操作符重载）讲解如何重载加法（+）操作符来对发票对象进行总计。另一个演示程序讲解如何重载几个操作符来创建一个便于产生渐变填充值的 RGB 类。然后，转换演示程序讲解如何转换各种温度制对象以及如何创建和使用一个三态控件。这些类不是完整的实现，但是我认为向程序员提供现实的演示程序有助于他们理解在应用程序中何处以及如何使用给定的功能。

谁应该阅读本书

本书的内容已经进行了较大的更新，添加了许多中高级信息，同时在组织各章内容时尽量使不熟悉 C# 的开发人员也能够理解这些主题。各章往往先解释 C# 或 .NET 的某个功能、语言中为什么需要这个功能以及它解决的问题。然后，提供一个与此功能相关的简单的演示应用程序，接着解释所需的语法。在讨论了这个主题的基础知识之后，提供一些比较高级的演示程序和资料，比如编译器产生的 MSIL。因此，本书既适合 C# 和 .NET 的初学者，也适合已经使用过 C# 语言进行编程并且希望学习更高级的语言功能的开发人员。

对本书读者惟一的要求是，你们应该知道如何使用 C、C++ 或 Java 编写简单的程序。另一个先决条件是你希望学习和研究使用 C# 语言编写应用程序的新技能。当然了，既然你手里拿着这本书，你显然是有这种愿望的。

本书的组织结构

本书被仔细地组织为三个在逻辑上相关的部分：每个部分由几章组成，每一章分别讨论 C# 或 .NET开发的一个方面。

第一部分：C# 类基本原理

在第一部分中，我们提供了使用 C# 定义和使用类和基本类成员的基础知识。但是，因为我的一个主要目标是提供非常实在的内容，所以我们没有谈论 Microsoft 的想法或我为什么认为 .NET 很出色等话题。程序员希望看到代码，而不是听我们大发议论。第 1 章直接带领你开始编写应用程序、编译库（DLL）和 .NET 模块以及创建多文件配件。你甚至将学习如何查看全局配件缓存并且使用 ILDASM 工具查看编译器产生的配件代码（MSIL）。第 2 章简单地概述了通用类型系统，第 3 章和第 4 章讲解 C# 类和结构定义的基础知识，以及如何定义和调用方法。这几章和书中其他章一样采用先易后难的组织形式，最后你会学到常量和只读字段的差异、类型初始化器的用途以及如何使用 new 和 override 关键字重定义虚拟方法。

第 5 章将这三个主题合在一起讨论，因为这些功能的目标都是使你能够为客户编写更直观的类接口。在讨论基础知识之后，你将通过许多底层 MSIL 来了解灵巧字段（特性）和灵巧数组（索引器）在幕后是如何实现的。第 6 章介绍我最喜欢的一个 C# 开发功能——属性。属性使你能够为类型定义文本性的标注，以后可以通过反射这些标注来判断类型的某些运行时特征。我们讨论了与属性相关的所有问题，包括属性的工作方式、如何在运行时查询一个类型是否具有属性以及如何获得属性的值。我们还讨论了预定义属性 Conditional、Obsolete、CLSCompliant、DllImport、StructLayout 以及各种配件属性。本书的第一部分以第 7 章结束，这一章讨论了如何声明和实现接口以及如何通过 is 和 as 操作符查询接口实现等主题。我们还讨论了如何显式地限定接口的成员名称、与接口继承相关的许多问题以及如何合并接口。

第二部分：编写代码

第一部分主要讲解如何定义和使用类和结构；第二部分讲述的都是与编程任务相关的内容。第 8 章和第 9 章分别讨论了如何使用表达式和操作符以及如何控制程序流。第 10 章是新添加的一章，它讨论的两个功能在使用任何语言进行编程时都是最重要的问题。第 11 章也是新增的，它集中讨论如何使用 .NET 框架提供的流类和文件系统类在磁盘上读写数据。你会用到该章中讨论的所有内容，包括字符串和二进制读取器以及使用 BinaryFormatter、SoapFormatter 和 XMLSerializer 对数据进行串行化。

我总是认为当今的程序设计书籍不应该将异常处理问题留到最后一章讨论，在 .NET 环境中更是如此。在 .NET 环境中异常处理用于跨语言界限传递错误信息，甚至 COM 错误也被自动映射为异常。第 12 章讲解了异常处理语法的基本知识、如何使用预定义的系统异常类以及如何定义自己的异常类。

对于我个人来说，只要一种语言功能能使我的类使用起来更直观，我就会喜欢它，而操作符重载功能正是如此。第 13 章不只讨论了操作符重载的语法和规则，还给出了两个适当进行操作符重载的现实例子：对发票进行总计和创建一个 RGB 类（它的操作符简化了在 for 循环中产生渐变值的过程）。接下来，我们在同一章中讨论了 C# 特有的功能：用户定义的转换。简单地说，用户定义的转换使你能够在结构或类上声明转换方式，这样结构或类就可以被转换为其他结构、类或基本的 C# 类型。

接下来是第 14 章。与接口和反射一样，这个主题也是对于 .NET 初学者最重要的一个主

题。在这一章中，将学习如何定义和创建委托和多点委托，以及如何使用事件实现公布/预订（即观察者）模式。第二部分以另一个很酷的新章结束。这一章讨论 C # 特有的从 C # 源代码注释产生 XML 文档的功能。当然，这个功能的最大优点是它基于 XML，这意味着它是完全开放的和可定制的（.NET 框架的许多功能也具有这种性质）。

第三部分：高级 C #

我写这一部分时心情最舒畅，因为它包含了一些我喜欢的主题。第 16 章介绍了数值处理和 Math 类。第 17 章讲解了如何使用 IEnumerable 和 IEnumerator 接口允许客户枚举你的类的数据。这一章中的高级主题包括创建有版本的枚举器、在允许枚举的同时保护数据以及实现可变接口。第 18 章讲解如何在应用程序中加入多线程和异步编程功能。单单一章是无法完全覆盖多线程这个主题的，但是我们还是讨论了开始使用线程所需的所有基本知识。在学习了如何创建、管理、调度和中止线程之后，将学习如何与线程进行数据交换。然后，我们将会看到几种串行化对线程方法的访问的办法（其目的是使方法成为线程安全的）。最后，将讲解如何使用委托启动异步方法，并且在方法返回之前阻塞其他操作，或者让此方法在完成时自动调用一个回调方法。第 19 章讨论了如何使用反射查询元数据。

第 20 章详细说明了内存管理和固定的知识，讨论的主题包括 .NET 垃圾收集器（GC）、Dispose 方案、IDisposable 接口、弱引用和编写不安全（未托管）的代码。接下来讨论 COM 互操作性问题，但是这个主题的内容太丰富以致于我使用了两章的篇幅。第 21 章讨论如何从 .NET（C #）应用程序使用传统的 COM 组件；第 22 章讨论如何从 Microsoft Visual C ++ 和 Microsoft Visual Basic 客户（用 COM 编写）使用 .NET 组件（用 C # 编写）。这两章远远超出了初学者的能力范围，而且要求读者具有一定的 COM 经验。第三部分的最后一章（第 23 章）相当详细地讨论了安全程序设计问题，讨论的主题包括代码签名、密码服务、代码访问安全、基于角色的安全和隔离的存储。不要错过这一章。

附录：MSIL 指令表

因为 MSIL 代码清单是本书的重要部分，所以本书的附录给出了一个完整的表格，它详细记录了每个 MSIL 指令、它的操作码、参数、说明和堆栈转换。堆栈转换使你能够看到执行指令之前和之后堆栈上与被调用的指令相关的内容。

本书中使用的惯例

与所有程序员一样，我也有自己喜欢的术语和编程风格。所以，下面对你将在书中看到的几个术语加以说明，并且解释几个惯例：

- **“problem domain”** 这个术语是我在多年前使用 Coad/Yourdon 面向对象的分析和设计方法论时学到的。“problem domain”是一个一般性术语，它是指要解决的问题（我们将这个术语译为“所涉及的问题”等——译者注）。
- **“consumer”（使用者）和“client”（客户）** 这两个术语可以互换，它们代表使用类或类型的任何代码。
- **“server”（服务器）** 这个术语用于表示客户或使用者使用的一段代码（通常是一个类）。

- **“argument”和“parameter”** 与大多数程序员一样，我将这两个术语互换使用，都是指传递给方法的值（我们将这两个术语都译为“参数”——译者注）。
- **“MSIL opcode”（MSIL操作码）和“MSIL instruction”（MSIL指令）** 尽管这两个术语从技术上讲不完全相同，但在本书中，它们可以互换。
- **“method prototype”（方法原型）和“method signature”（方法识别标志）** 在本书中这两个术语可互换使用。
- **分号** 我有时用分号结束类定义，有时不这么做。与C++不同，C#编译器不检查类定义末尾的分号，所以你可以自由选择加不加分号。我开始使用分号只是因为我相信这有助于将同一个源代码文件中定义的多个类分隔开（只是在视觉效果上）。但这是个人选择，而且据我所知没有任何设计原则专门规定了这个问题。
- **代码的换行** 由于书的版心所限，源代码不得不加一些奇怪的换行。这很遗憾，但却是不可避免的。当你看到格式奇怪的代码行时请记住这一情况。

编译演示应用程序

本书的一个目标是使读者无需使用Microsoft Visual Studio .NET就可以编译所有的演示应用程序。这样，读者只要有C#编译器就能够编译和运行这些演示程序。但是，因为Visual Studio .NET是流行的C#应用程序编写工具，所以我们使用它创建大多数演示工程。最终结果是，你既可以使用Visual Studio .NET编译这些工程，也可以按照第1章的说明使用命令行编译器。

在这方面有几点需要注意，还有几处例外：

- **所有Visual Studio工程** 当使用Visual Studio .NET创建C#控制台应用程序时，产生两个源代码文件。第一个是主源代码文件Class1.cs。你将在第1章中学习如何将这个文件编译为应用程序。产生的第二个文件称为AssemblyInfo.cs。这个文件通常用于定义配件级属性。第1章介绍了配件，第6章讨论了属性。如果你使用命令行编译器，而且希望将Class1.cs和AssemblyInfo.cs文件编译为一个应用程序，那么只需使用以下的C#编译器开关（其中的appname.exe是你给产生的应用程序起的名字）：

```
csc /out:<appname.exe> class1.cs assemblyinfo.cs
```

- **第1章** HelloWorldVS工程是这一章中使用Visual Studio .NET创建的惟一工程。
- **第2章** 这一章是对通用类型系统的简短概述，所以它不包含演示应用程序。
- **第19章** Visual Studio .NET不支持创建用于组合成多文件配件的C# .NET模块。因此，CommProtocol演示应用程序只能从命令行编译。为了简化这个任务，我提供了一个简单的make文件（buildall.cmd），它对这个演示程序所需的所有组件进行编译。另外，为了简单起见，ILGeneration演示应用程序也只能从命令行编译。这个演示程序也有一个用于编译它的组件的buildall.cmd文件。
- **第21章** 这一章的主题是在C#应用程序中使用传统的COM组件，所以此章的几个演示应用程序是使用Visual Basic和Visual C++编译的。你需要使用Visual Studio编译这些工程。但是我有意使用了Visual C++ 6和Visual Basic 6。这样，如果你安装了Visual Studio 6和.NET Framework SDK（而没有安装Visual Studio .NET），那么仍然可以使用这些演示程序。另外，如果使用Visual Studio .NET打开这些文件，你只会得到一个消息，

问你是否希望将这些工程转换为新格式。回答 yes 就可以正常运行它们了。

- **第 22 章** 这一章的主题是从未托管代码使用 .NET 组件。与第 21 章一样，某些演示程序是使用 Visual C++ 6 和 Visual Basic 6 编写的，所以也有前面所说明的问题。

关于附带的光盘

本书附带了一张光盘。如果你的 Windows 上启用了 AutoRun 功能，那么在将光盘放入光驱时你会看到一个快闪屏，它提供了一些安装选项。手工启动这个屏幕的办法是从光盘的根目录运行 StartCD。StartCD 程序提供了到光盘上包含的 eBook（电子书）的链接、本书示例文件的安装程序和到光盘上包含的 Microsoft .NET Framework SDK 的链接。

本书的示例程序位于 BookFiles \ Code 文件夹。你可以在光盘上查看这些示例，也可以使用 StartCD 中的安装程序将它们安装到硬盘上。

注意 如果你无法浏览 Samples 文件夹中的文件，那么可能是因为你使用的光盘驱动程序比较旧，不支持长文件名。如果是这种情况，必须通过运行 Setup 程序将示例文件安装到硬盘上，才能浏览这些文件。

Web 上的示例代码

也可以从本书的网站（www.microsoft.com/MSPress/books/5861.asp）获得示例代码。在这个页面上还有代码升级和到相关书籍的链接。

系统需求

为了最充分地利用本书，我强烈建议你在阅读每一章的同时运行示例应用程序。为此需要 .NET Framework SDK（其中包含 C# 编译器）。注意，如果你有 Visual Studio .NET，Framework SDK 就包含在其中。

另一个重要需求是：.NET 代码可以在 Windows 98 和更高版本上执行，但是只有以下操作系统支持 Framework SDK：

- Microsoft Windows NT 4.0
- Microsoft Windows 2000（建议使用 Service Pack 2）
- Microsoft Windows XP（运行 Microsoft ASP.NET 需要 Microsoft Windows XP Professional）

反馈

我并不认为“因为我写书，我就是专家”。我只是一个幸运地得到了写书机会的普通人。因此，我要向其他人学习而且实际上我也乐意这么做。如果你有关于本书的任何问题或者只是想与人谈谈 C# 和 .NET，那么可以通过我的网站 www.TheCodeChannel.com 与我联系。我还在这个网站上提供了书籍的勘误表、补充章节（通常包含由于时间限制而没有放进本书的资料）和我使用 C# 编写的 .NET 应用程序示例。

目　录

第二部分 编写代码