azon 网上书店热销图书
ns 出 版 社 独 家 授 权

网络前沿技术系列
Network Advanced Technology Series

.NET 编程先锋 C#

Presenting C#

(美) Christoph Wille/著
袁 萌/译

C# 相对于 C++ 来说，不愧为一代高效、简化、现代的面向对象的程序设计语言，可谓青出于蓝胜于蓝。相信 C# 将会成为您的首选。

——微软公司 C# 产品部经理 Tony Goodhew

SAMS

中国青年出版社

网络前沿技术系列
Network Advanced Technology Series

.NET 编程先锋 C#

Presenting C#

(美) Christoph Wille/著
袁 萌/译

中国青年出版社
CHINA YOUTH PRESS

（京）新登字083号

策　　划：胡守文
　　王修文
　　郭　光
责任编辑：江　颖
责任校对：肖新民

书　　名：《.NET编程先锋C#》
编　　著：*Christoph Wille*
翻　　译：袁　萌
出版发行：中国青年出版社
　　地址：北京市东四十二条21号　邮政编码：100708
　　电话：(010) 64069368　传真：(010) 64053266
印　　刷：高唐印刷有限公司
开　　本：16开
版　　次：2001年1月北京第1版
印　　次：2001年1月第1次印刷
印　　数：1-5000
定　　价：19.00元

译 者 的 话

在计算机图书出版领域享有盛誉的著名出版社 Sams 新近出版的《Presenting C#》（中译名为《.NET 编程先锋 C#》），是一本很有深度且颇受关注的书，涉及到当代现代计算机编程语言设计的许多方面。对于具有一定的 C、C++计算机语言基础的软件工作者、大学相关专业的师生以及关心软件业的现状及其未来的各方人士，本书都非常具有阅读和收藏的价值。作者 Christoph Wille 是参与微软 C#程序语言早期版本开发工作的少数几个人之一，精通于计算机语言设计，曾有许多知名的专著。

C#是在 C、C++基础上发展起来的，是具有当代风格、面向对象的一种跨语言平台(Cross-Language)的程序设计语言。由 C#的编译器产生的输出代码，称为“受操控代码”（Managed Code），“受操控代码”并不能直接在目标机器上运行，只是形成了一种跨越传统计算机语言的所谓“中间语言 IL”（Intermediate Language）文件。同时，C#编译器又在此文件中的各种数据类型前面直接插入了标示其相应属性的一种所谓“元数据”（Metadata），形成了一种可移植、可执行的所谓“PE(Portable & Executable)文件”。公用语言运行环境（CLR，Common Language Runtime）按照一定的操控规范，对中间语言 IL 代码再进行即时编译优化处理，产生最终的目标机上的本地码（Native Code），直至被高速执行而获得最终结果。CLR 对 IL 代码的操作控制过程，可用一种“虚拟对象系统 VOS”(Virtual Object System)来模拟，使其满足一定的“虚拟执行规范 VES”（Virtual Execution Specification），实现了跨越各种计算机语言的软件集成，从而最终构成了微软.NET 的基础架构。C#是迈向.NET 时代的新一代计算机语言的明星，是面向网络应用，实现跨越不同语言——“从软件到软件”目标——的首选语言。

在 C#中，由于元数据技术的引入和应用，使紧随其后的相关数据对象的处理得到即时的“操控”，从而顺利实现了对各种数据类型“版本差异”和电子签名的即时控制，也加强了系统安全性的检测，进一步提高了系统的稳定性和安全性。由此，我们可以预见到 C#语言将得以广泛应用，软件变服务的所谓.NET 时代即将来临，让我们一起迎接新型网络软件业的未来吧！

中国青年出版社慧眼独具，在C#刚刚推出之时，就引进本书的版权，对中国信息技术的发展无疑具有十分重要的积极意义。感谢本书策划编辑江颖对翻译工作的帮助和支持！

参加本书翻译工作的有：袁萌、沈更新、王宏洲、袁丹、张式广、李梦翔、赖真刚、徐杰。袁萌负责全面审校工作，周冰梅协助。罗隆玉完成了文字录入工作。由于时间紧迫，译文难免有误译之处，请指正。

袁　萌

2000年12月18日

前　言

《.NET 编程先锋 C#》欢迎你！C#是一种运行于公用语言运行环境（CLR，Common Language Runtime）和.NET 的平台上的企业级程序设计语言（读作 C Sharp）。希望本书能引导你乘上这趟快车，迅速掌握这种全新的语言。

公用语言运行环境（CLR）不只是操纵控制代码的运行，而且它可以使编程变得很轻松。C#编译器专为这个运行环境产生可操控码。你从中会得到跨语言的集成、跨语言异常处理、安全机制增强、配制和版本处理技术以及任意调试和解析（profiling）代码等多项支持。

C#是 CLR 中的首选语言。支持.NET 的大多数框架都是用 C#写，因此 C#的编译器是绑定在.NET 平台上最经得起检验、优化程度最好的编译器。C#语言不仅从 C++中借用了许多强大的功能，同时又体现了现代编程思想，增加了类型安全性，这使得 C#成为提供企业解决方案的首选语言。

本书面向的读者

如果你的编程经验不是特别的丰富，那么本书就不怎么适合你。本书是要让程序员在已有 C++或 C 程序设计的基础上更进一步，掌握如何使用 C#提供的强大功能进行开发。本书主要是为那些已经具有编程语言如 C 或 C++，VB，Java 等的使用经验的程序员而写的。

如果你有使用 C++的背景，那么掌握 C#不是件难事，不过即使你没有 C++的基础，如果你精通别的编程语言，本书也会让你跟上编程技术发展的前沿。如果你有一点 COM 编程的知识那最好不过，但这并不是必需的。

本书的组织结构

全书分为 12 章，下面将摘要介绍每章讲述的内容

- 第 1 章，“C#概述”——介绍 C# 的主要特点，以及 C# 为什么我们要学习 C#。
- 第 2 章，“理论基础——公用语言运行环境”——讲述公用语言运行环境（CLR）如何为你的 C# 代码提供运行基础。
- 第 3 章，“第一个 C#应用程序”——本章将创建你的第一个 C#应用程序（猜到是什么了吗），它就是“Helle World”程序。

- 第 4 章 “C#类型”——介绍你能在应用程序中使用的各种数据类型。你将看到值类型和引用类型之间的差异，装箱和拆箱机制的工作原理。
- 第 5 章，“类”——你将接触到 C#的核心（从中你会感到其强大的功能），那就是使用类进行面向对象的编程。你将在此章学到许多关于构造器、析构器、方法、属性、索引器和事件方面的知识。
- 第 6 章，“控制语句”——执行应用程序中的流控制，学习 C#提供的各种条件语句和循环语句。
- 第 7 章，“异常处理”——你将学到通过设置恰当的异常处理，使你的应用程序在公用语言运行环境中循规蹈矩地运行。
- 第 8 章，“用 C#中开发组件”——用 C#中开发可被不同语言使用的组件，因为你的代码是基于 CLR 的。
- 第 9 章，“配置和实施”——此章讲述如何在 C#中进行条件编译，如何从 C#的源代码产生文档。另外本章还介绍.NET 平台的版本处理技术。
- 第 10 章，“和未操控码交互”——如何从 C#内部使用未操控码，未操控码如何与你的 C#组件交互。
- 第 11 章，“调试 C#代码”——如何使用 SDK 提供的调试工具来发现、修正 C#应用程序中的细微错误。
- 第 12 章，“安全”——探讨.NET 平台的安全概念。介绍代码访问级安全性和基于角色的安全性。

学习本书需要的工具

从本书的观点看，你所需要的是一个.NET 软件开发工具包（SDK）。虽然最低限度的配置环境只需要有公用语言运行环境和 C#编译器，但在探索如此激动人心的新技术的功能时，能在机器中装上文档和所有的 SDK 工具（包括调试器）无疑是一个好想法。

本书不要求你的机器中已经安装任何 Visual Studio 7 的工具。我的惟一一点建议是：在编辑 C#源文件时，最好有一个像科班出身的程序员所使用的能显示行号的编辑器。

目 录

第1章　C# 概述

- 为什么需要另一种编程语言？

欢迎来到C# 的世界！本章将带你游历 C# 世界，并回答一些问题，比如：为什么你要使用 C#，C# 和 C++ 的主要区别是什么，以及 C# 为何使开发工作变得更容易和更有趣。

1.1 为什么需要C#

需要回答的有意义的问题是：在你已经能用 C++或 Visual Basic 来开发企业应用的情况下，为什么还需要学习另一种编程语言？从市场角度考虑，C#将成为企业级计算机领域开发.NET 应用的首选语言。本章将用一些论据来支持这一论点，展示 C#的许多特点，并激发你的学习兴趣。

作为编程语言，C#是从 C 和 C++发展而来的；它是现代的、简单的、完全面向对象的，而且是类型安全的。如果你是一个 C/C++程序员，你的学习曲线将会很平坦。许多 C#语句是直接从你所喜爱的语言里借来的，包括表达式和操作符。乍看上去，一个 C#程序很像一个 C++程序。

关于 C#的重要一点是，它是一种现代编程语言。在类、名字空间、方法重载和异常处理等方面，C#简化和现代化了 C++。C#去掉了 C++中的许多复杂性，使其变得更加易于使用，不易出错。

对易于使用的贡献是去掉了 C++的一些特性：没有宏，没有模板，没有了多重继承。以上这几项特性所造成的问题要比其带来的益处还要多，尤其是对于企业开发者来说。

带来更多方便于使用的新增特点有：严格的类型安全性、版本处理技术、垃圾收集等，所有这些特性都是为了开发面向组件的软件。虽然你没有了在 C++中所具有的全部能力，但你的开发将变得快速而富有成果。

在我走得太快和全面介绍太多的特性之前，我要先停住全面介绍一下基于后继章节要点之上的 C#的各种特点：

- 简单性
- 现代性
- 面向对象
- 类型安全性
- 版本处理技术
- 兼容性
- 灵活性

1.1.1 简单性

你一定不会认为 C++具有简单易学的特点，C#就不是这样。这种编程语言的首要目标就是简单性，它通过增加许多特性，同时舍弃了一些特性，对 C#的总体简单性做出了贡献。

没有指针是C#的一个显著特性。在缺省情况下，你使用一种可操控的（managed）代码进行工作，此时，一些不安全的操作，如直接的内存操作，将是不允许的。我认为没有任何一个C++程序员敢说他从来没有用指针存取过不属于他自己的内存。

和指针的“故事”（drama）紧密相关的是操作符的“疯狂”(madness)使用。在C++中，你拥有像::，.，和->一样的操作符，分别用于名字空间、成员和引用等操作。对于一个初学者，这些操作符会成为他学习中的困难日子。C#去掉了别的操作符，只支持一个“.”（即“dot”）,程序员现在需要理解的一切就只有名字嵌套的概念了。

你不再需要记住那些源于不同处理器结构的神秘类型，包括可变长的整数类型。C#通过提供一个统一的类型系统去掉了这些问题。这个类型系统使你可以将每种类型看作一个对象，不管它是初始数据类型还是发育完全的类。与别的语言不同的是，把单个类型当作对象看待并不会增加执行上的难度，因为它提供了一个叫做装箱(boxing)和拆箱(unboxing)的机制，关于这个机制以后再详细讲解。基本的一点是，这个技术仅在需要的情况下才让一个对象去访问单个类型。

老练的程序员一开始可能会不喜欢这样做，但是现在整型和布尔数据类型是完全不同的类型了。这意味着if判别式的结果只能是布尔数据类型，如果是别的类型则编译器会报错。那种搞混了比较和赋值运算的错误不会再发生了。

C#也去掉了多年来在C++中蔓延的冗余问题，比如const 和#define、各种的字符类型等等。常用的形式在C#中被保留下来，而别的冗余形式从C#语言中被清除出去了。

1.1.2 现代性

你在学习C#上所付出的努力是一项巨大的投资，因为C#是被设计成为开发.NET应用的首选语言。你会发现许多在C++中必须由你自己来实现或者干脆没有的特征，都成为基础C#语言实现的一个部分了。

金融类型对于企业级编程语言来说是很受欢迎的一个附加类型。你可以使用一个新的decimal数据类型进行货币计算。如果你不喜欢这个简单类型，可以很容易地为你的应用特别精制一个新的类型。

我已经说过指针不再是你的编程武器了，所以当你得知不用再负责整个内存的管理时你不应该感到惊讶。.NET平台运行环境提供了一个垃圾收集器负责你的C#程序的内存管理工作。由于内存和你的应用是可以被操控的，所以要切记强化类型安全以便保证应用的稳定性。

对于C++程序员来说，异常处理是C#的一个主要特性，这绝不是什么新鲜事。然而，和C++中不同，异常处理是跨语言的（运行环境的另一特点）。在C#之前，你必须对付离

奇的 HRESULTs——现在已经没必要了，因为稳健的基于异常的错误处理机制能做好这些。

安全性是现代应用的头等要求，C#不会让你对此袖手旁观：它提供了一种声明某种权能的元数据（metadata）语法和构建.NET 安全模型的许可条件。元数据是公用语言运行环境的关键概念，下一章将深入揭示其中的奥秘。

1.1.3 面向对象

没有人会认为一个新语言竟会不支持面向对象的特性。理所当然，C#也支持面向对象的所有关键概念：封装、继承和多态性。整个C#的类模型是建立在.NET虚拟对象系统（VOS，Virtual Object System）之上的，VOS 将在下章讲述。对象模型是基础架构(infrastructure)的一部分而不再是编程语言的一部分。

你可能从一开始就会注意到的一件事情是：没有全局函数、变量或常数。每样东西必须被封装在一个类中，或者作为一个实例成员（通过类的一个实例对象来访问），或者作为一个静态成员（通过类型来访问），这会使你的 C#代码具有更好的可读性，并且减少了发生命名冲突的可能性。

在类中定义的方法缺省情况下不是虚拟的（不能被派生类所覆盖），这一做法的要点是可以去掉另一个产生错误的来源——对方法的误覆盖。如果一个方法可以被覆盖，它必须有一个显式 Virtual 修饰符。这样不但可以减小虚函数表的长度，还能保证正确的版本处理行为。

当熟悉了用 C++的类编程之后，你就知道可以用访问修饰符为类的成员指定不同的访问级别。C#也支持 private，protected 和 public 访问修饰符，而且增加了第四个：internal。关于这些访问修饰符的细节将在第 5 章“类”中介绍。

你们中有多少人曾经创建过一个从多个基类派生的类？（ATL 的程序员,你的票不算数），在多数情况下，你只需要从一个类派生，从多个基类派生所带来的问题比这种做法所能解决的问题要更多。这就是为什么 C#只允许一个基类的原因。如果你觉得需要多重继承，你可以执行接口。

一个可能产生的问题是：既然在 C#中没有指针，那么怎样去模拟指针功能呢？这个问题的答案是 delegates（代表），它提供了.NET 事件模型的基础架构。至于具体如何实现，我将在第 5 章对其进行详细介绍。

1.1.4 类型安全性

我再一次拿指针来举例。当你在 C++中定义了一个指针后，你可以自由地把它指向任意一个类型，包括做一些相当愚昧的事，比如将一个 int*(整型指针)指向（双精度型指针）。

只要内存支持这一操作，它就会凑合着工作。这当然不是你所设想的企业级编程语言的类型安全性。

因为上述问题，C#实施了最严格的类型安全来保护它自身及其垃圾收集器。因此，在C#中你必须遵守关于变量的一些规定：

- 不能使用未初始化变量。对于对象的成员变量，编译器负责将它们置零。局部变量你应自己负责。如果使用了未经初始化的变量，编译器会提醒你。这样做的好处是：你可以摆脱因使用未初始化变量得到一个可笑结果的错误。
- C#不支持不安全的指向。不能将整数指向引用类型（比如对象），当进行下行指向时，C#会验证指向的有效性（就是说，导出对象确实是从你要将它下行指向的类型中导出的）。
- 边界检查是C#的一部分。当数组实际上只有n－1个元素时，不可能使用它“额外”的数组元素n，这使重写未经分配的内存成为不可能。
- 算术运算可能溢出结果数据类型的范围。C#允许在应用级或者语句级检查这类操作中的溢出，使用溢出检查，当溢出发生时会出现一个异常。
- C#中传递的引用参数是类型安全的。

1.1.5 版本处理技术

在过去的几年中，几乎所有的程序员都至少一次地和已成为所谓的的“DLL地狱”打过交道，产生这个问题是因为许多计算机上安装了同一DLL的不同版本。有时，老的应用在较新版的DLL有幸还可以运行，但大多数情况下，它们崩溃了。版本处理技术是当今一个真正很痛苦的问题。

在第8章“用C#开发组件”中你将看到，CLR对你写的应用的版本处理技术支持。C#尽其所能支持这种版本处理功能。虽然C#自己并不能保证提供正确的版本处理结果，但它为程序员提供了这种版本处理的可能性。有了这个适当的支持，开发者可以确保当他开发的类库升级时，会与已有的客户应用保持二进制兼容。

1.1.6 兼容性

C#不是存在于一个封闭的世界里。它允许你通过遵守重要的.NET 公用语言规范(CLS Common Language Specification) 访问不同的API。 CLS定义了遵守这一标准规范的语言间相互操作标准。为了加强 CLS 的兼容性,C#编译器检查所有公开输出项所遵守的条件，发现不符合规范的情况就会报错。

当然，你也希望能够访问的老COM对象。.NET平台提供了对COM的透明访问，而