TURING 图灵程序设计丛书 微软技术系列

Apress®

Accelerated C# 2008

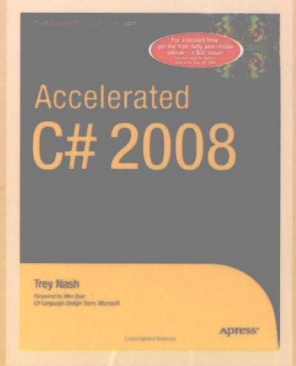
C#捷径教程

[美] Trey Nash 著

刘新军 译

涵盖
C# 3.0
新特性

- 好评如潮的C#实战图书
- 汲取.NET技术精髓的捷径
- 专章讲述习惯用法与设计模式



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书 微软技术系列

Accelerated C# 2008

C#捷径教程

[美] Trey Nash 著
刘新军 译

人民邮电出版社
北 京

图书在版编目（CIP）数据

C# 捷径教程 /（美）纳什（Nash, T.）著；刘新军译．
—北京：人民邮电出版社，2009.2
（图灵程序设计丛书）
ISBN 978-7-115-19258-5

Ⅰ. C… Ⅱ. ①纳…②刘… Ⅲ. C 语言－程序设计－教材
Ⅳ. TP312

中国版本图书馆CIP数据核字（2008）第184146号

内 容 提 要

C# 3.0 提供了很多强大的特性，通过使用 lambda 表达式、扩展方法和语言集成查询（LINQ），方便地引入了函数式编程，使 C# 程序员如虎添翼。本书通过许多精彩的示例介绍了每个特性，深入浅出地讲解了 C# 语言的核心概念，以及如何聪明地应用 C# 的习惯用法和面向对象的设计模式来挖掘 C# 和 CLR 的能力。

本书适合有一定编程经验的程序员阅读。

图灵程序设计丛书

C# 捷径教程

◆ 著 [美] Trey Nash
译 刘新军
责任编辑 傅志红
执行编辑 陈兴璐

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京铭成印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：28.75
字数：770千字 2009年2月第1版
印数：1－3000册 2009年2月北京第1次印刷

著作权合同登记号 图字：01-2007-5617号
ISBN 978-7-115-19258-5/TP

定价：59.00元

读者服务热线：(010)88593802 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

以此纪念我的祖父

Weldon W. Nash 爵士

1912 年 12 月 19 日—2007 年 4 月 29 日

感谢 Svetlana

如此信任我

译 者 序

因为在学校微软技术俱乐部活动的缘故，C# Beta版一经问世我就从微软高校关系部拿到了试用版，并且在微软的支持下俱乐部的一帮人一起做了几个小项目。2003年10月，微软全球副总裁Jawad Khaki来北邮做讲座，之后有人去索取签名，一个哥们居然拿着一本《Java编程思想》上去。结果Jawad大笔一挥在扉页上写下了几个大字“C# is a better Java”。这句话正是我当时对C#的第一印象，可能也正反映了微软对C#的期望和当时业界对C#的评价。

当然微软的雄心远不止于此，.NET才是它的战略发展方向，C#只是推动这个发展的一个独门武器。几年来，微软投入了大量的资源，把这款武器打造得越来越锋利。随着泛型、匿名方法、迭代器、分部类型/分部方法、LINQ、匿名类型、对象/集合初始化器、扩展方法、lambda表达式等一系列特性的逐步引入，C#俨然成为了特性最丰富的编程语言。

丰富的特性带来了强大的功能和高效的开发效率，但与之相伴的副作用之一就是加陡了学习曲线。一件再好的武器，如果你不能透彻掌握其奥妙玄机，后果将轻则浪费宝物，重则误伤自己。因此，除了会使手中的利器，找到一位好师父、觅得一本好宝典同样非常重要。

目前市面上有很多关于C#的图书，也有很多诸如《24小时学会XXX》、《轻松掌握XXX》的图书。刚开始，我对出版社定下的本书中文书名——《C#捷径教程》颇有微词。不过， 仔细一想，“捷径”确实最贴切原书名中Accelerated的本意。学习原本没有捷径可走，但要在有限的时间内取得最大的成效，还是有策略和方法的。本书的“捷径”体现在以下两个方面。

1. 以相对短的篇幅浓缩了C#的几乎所有的精华，高屋建瓴，取精用弘。本书没有纠缠于繁琐的语法细节（有任何C/C++/Java语言背景的人都可以触类旁通），也没有涉及华丽的IDE操作与界面设计内容（可以通过Tutorial或Help更方便地获得），而是直接切入C#语言本身的精髓，让你最直接地感受并掌握最激动人心的特性。

2. 以简洁的语言与丰富的示例来讲解各个特性，一针见血，酣畅淋漓。作者是位资深的程序员，因此本书的行文风格都是基于程序员的角度和思维方式，使你在阅读的过程中有一种心领神会的默契之感。

当然，翻译的过程没有捷径可走。本书的面世是许多人帮助与支持的结果。

感谢编辑陈兴璐小姐。这是第二次与她合作。她一如既往地友善耐心、认真负责，保证了本书的按时按质完成。

感谢杨军、崔晓川、刘光强、李辛鹤、徐进在翻译过程中给予我的的帮助，他们的专业水平和中肯建议，使翻译润色不少。与他们的讨论交流，也让我受益匪浅。

感谢小顾同学，忍受了我长达半年深居简出的宅男生活:)。

由于时间仓促，水平有限，不足之处望各位同仁不吝赐教。

希望你在《C#捷径教程》中找到学习C#的捷径！

刘新军

2008年11月于北京

序

编程是令人愉快的。它是一个惊险刺激的历程，充满着令人烦恼的问题和解决问题之后带来的愉悦感受。在这个过程中，开发者透过编程语言这个镜头去观察遇到的问题，思考其解决方案。因此，开发者对语言的熟悉非常重要。

开发者面对的问题是千奇百怪的，从医疗应用到保险软件，从多媒体应用到数据挖掘工具。C#紧随开发者所面对的问题的演进而演进。问题变得越来越复杂，为此语言也相应地变得更简单更强大。

C#诞生之初是为了描述运行在多种执行环境的可重用组件。在这个阶段，它确立了其自己作为描述组件和系统架构的重要语言的地位，同时也继承了C风格语言的优点并发扬光大。它的主要贡献之一是实现了一个完全面向对象类型系统，把原始类型和复杂类型的概念与垃圾回收统一在一起。

C#的第1版是一个重大成就，而第2版的出现代表了一个决定性的时刻。随着泛型的引入，它的类型系统变得更加丰富。C#还开始提供迭代器、匿名方法等更方便简单和更优雅设计的特性。使用这些特性可以开发出更灵活更强大的框架。

C#的第3版开辟了一片新天地。它把代码和数据之间的界限变得模糊，引入了说明式的查询语法，给程序员带来了函数式编程特性。这些新增的特性使程序员能够容易地解决在处理数据过程中所遇到的困难，不管这些数据是来自内存、数据库还是Web服务器。程序员会发现这些特性给编程重新带来了乐趣。

在本书中，Trey Nash对C#语言的解说令人耳目一新。他不仅自己精通C#，还能够引导读者通过学习好好掌握C#。他用许多精彩的示例来介绍每个特性的来龙去脉，演示常用和最佳实践，使学习的过程变得轻松愉快。我相信，读者阅读本书后，肯定能写出更好的代码。

Wes Dyer
微软公司C#编译器和语言设计组

前　言

对熟悉其他面向对象语言的人来说，Visual C# .NET（C#）学习起来相对容易。熟悉Visual Basic 6.0的人想学一门面向对象语言，也会发现C#很容易上手。然而，尽管C#和.NET框架为创建简单应用提供了一条捷径，但为了开发复杂、健壮和容错的C#应用，你还是需要掌握很多的信息并理解怎样正确地使用它们。本书将教授你需要掌握的知识，并解释如何最好地运用这些知识来快速获得真正的C#专业技能。

学会习惯用法和设计模式对培养和应用专业技能有不可估量的作用，本书将展示怎样使用它们来创建高效、健壮、容错和异常安全（exception safe）的应用程序。虽然Java和C++程序员对于其中的许多模式都比较熟悉，但有一些是.NET和公共语言运行库（CLR）独有的。本书后面的章节会展示如何应用这些必不可少的习惯用法和设计模式，无缝地把C#应用程序与.NET运行库集成，重点放在C# 3.0的新功能上。

设计模式记录的是许多程序员在应用程序设计中反复发现的最佳实践。事实上，.NET框架本身就实现了许多众所周知的设计模式。同样，在过去的.NET框架的三个版本和C#的两个版本中，还发现了许多新的习惯用法和最佳实践。你会看到本书详细描述了这些实践。另外，值得注意的是，宝贵的技术工具库也在不断革新。

随着C# 3.0的到来，可以使用lambda表达式、扩展方法和语言集成查询（Language Integrated Query，LINQ）方便地引入函数式编程。lambda表达式可以方便地在某个点声明和实例化函数委托（function delegate）。另外，有了lambda表达式，创建functional就是小菜一碟。functional是以函数作为参数并返回另一个函数作为返回值的函数。即使你之前可以在C#里面实现函数式编程（虽然还是有点困难），但C# 3.0里面的新语言特性提供了一个新的环境，在这里函数式编程和典型的命令式编程可以繁荣共存。LINQ允许使用这种语言的语法来表示数据查询操作（这本质上也是functional）。一旦知道了LINQ的工作原理，你就会意识到它所做的远远不止简单的数据查询，还可以用它来实现复杂的函数式编程。

.NET和CLR提供了一个独特和稳定的跨平台执行环境。C#是这个平台的首选语言。但是你会发现本书探讨的技术也适用于任何针对.NET运行时的语言。对于那些有丰富C++经验，熟悉C++规范形式（canonical form）、异常安全、资源获得即初始化（Resource Acquisition Is Initialization，RAII）、const正确性（const correctness）等概念的读者，本书说明了如何把这些概念应用于C#。对于那些有多年技术积累的Java或Visual Basic程序员，可通过本书掌握如何把这些技术有效地应用于C#。

总之，要成为一名C#专家，并不需要花几年的功夫去摸着石头过河，只需要学习正确的知识并以正确的方式来使用它们。这正是我写作本书的动机。

关于本书

本书假设你已经具备了一些面向对象语言的基本知识，比如用过C++、Java或Visual Basic .NET。

因为C#语法起源于C++和Java，所以我不会花很多时间来介绍C#语法，重点讨论的是与C++或Java有显著区别的地方。如果你已经对C#有所了解，那么可以略读甚至跳过第1章~第3章。

第1章大致给出了一个简单的C#应用，并描述了C#编程环境与C++编程环境之间的基本区别。

第2章是第1章的具体展开，快速说明了C#应用程序运行的托管环境。接着介绍了程序集（assembly），它是C#代码文件编译成的基本构建块。还讲解了元数据是如何使程序集成为自我描述的。

第3章深入讲解了C#的语法。这里介绍了CLR中的两种基本类型：值类型和引用类型。还介绍了命名空间，以及如何用它来从逻辑上划分应用里的类型和功能。

第4~13章深入介绍了如何在C#程序和设计里运用习惯用法、设计模式和最佳实践。这本书尽量以逻辑顺序来安排这些章节，但还是会在其中某一章不可避免地引用后面一章涉及的技术或主题。

第4章详细讲解了如何在C#里定义类型。可以在本章学到关于CLR里值类型和引用类型的更多知识。本章还简单涉及了CLR和C#对本地接口的支持。你会发现C#里类型继承的工作原理，也会看到每个对象如何派生自System.Object。本章还包括了相当多关于托管环境以及如何在其中定义有用类型所必需的信息。本章介绍了其中很多的主题，并在随后的几章中更详细地讨论。

第5章详细介绍了接口及其在C#语言中扮演的角色。接口定义了类型可以选择实现的功能性契约。你可掌握类型实现接口的各种方法，以及接口方法被调用的时候，运行时如何选择调用哪个方法。

第6章详细介绍了将内建的C#操作符应用于自定义的类型时如何赋予其定制的功能。你将了解如何可靠地重载操作符，因为不是所有的为CLR编译代码的托管语言都能使用重载的操作符。

第7章展示了C#和CLR的异常处理功能。虽然与C++的语法相似，但创建异常安全和异常中立的代码还是颇有难度的，甚至比在本地C++中创建异常安全代码还要难一点。你会看到创建容错的、异常安全的代码根本不需要try、catch或finally指令。本章还描述了.NET 2.0运行时中新增的一些功能，允许你创建比用.NET 1.1创建的代码的容错性更好的代码。

第8章描述了字符串如何作为CLR的头等（first-class）类型及如何在C#中有效地使用它们。本章的大部分篇幅集中在.NET框架中各种类型的字符串格式化功能，以及如何让自己定义的类型通过实现IFormattable接口来具有类似的行为。另外，还介绍了.NET框架的全球化功能以及如何为.NET框架目前还不知道的文化和地区创建定制的CultureInfo。

第9章介绍了C#中的各种数组和容器类型。你可以创建两种类型的多维数组，可以在使用容器工具类的同时创建自己的容器类型。你会看到使用C# 2.0引入的新迭代器语法定义前向、反向和双向迭代器，这样你自己的容器类型就能与foreach语句配合工作了。

第10章展示C#里的回调机制。历史上，所有可行的框架都提供了某种机制来实现回调。C#更进了一步，把回调包装到了称为委托的可调用对象中。C# 2.0提供了一种简短的语法来创建委托，这就是*匿名方法*。匿名方法与函数式编程里的lambda函数类似。另外，还会看到如何在委托之上构建框架来提供发布/订阅事件通知机制，允许你在设计中把事件的源头与消费者解耦。

第11章介绍C# 2.0和CLR中加入的也许是最激动人心的特性。那些熟悉C++模板的读者会发现泛型似曾相识，虽然二者存在一些本质的区别。使用泛型可以提供一个功能的壳，可以在运行时在这个壳内定义更具体的类型。泛型用于容器类型的时候最有用，它与之前.NET版本里的容器相比效率更高。

第12章介绍在C#托管虚拟执行环境中创建多线程应用需要做的工作。如果你熟悉本地Win32环境的线程，就会注意到二者的显著区别。托管环境提供了更多的基础设施使多线程更加容易使用。可以

看到委托如何通过“我欠你（I Owe You，IOU）”模式提供一个优秀的网关通往进程的线程池。可以说，同步是使多线程并发运行的最重要的概念。本章介绍了可以在应用中使用的各种同步工具。

第13章论述定义新类型的最佳实践，以及如何实现这些类型，从而可以自然地使用它们，也避免类型的使用者不经意地误用它们。这些概念在其他章有所涉及，但本章作了详细论述。本章最后给出了一个使用C#定义新类型时应该考虑的问题检查单。

第14章介绍C# 3.0引入的一个新特性。由于可以像实例方法那样在它们所扩展的类型上调用扩展方法，因此扩展方法看起来像类型契约的扩展。但扩展方法的作用不止于此，本章向你展示C#中扩展方法将如何打开函数式编程之门。

第15章介绍C# 3.0里引入的另一个新特性LINQ。你可以用lambda表达式以一种简洁和描述性的语法来声明并实例化委托。虽然前面提到的匿名方法也能达到同样的目的，但匿名方法更加啰嗦而且语法上没有这么清新流畅。在C# 3.0里可以把lambda表达式转换成表达式树。也就是说，C#语言具有内建的功能来把代码转换成数据结构。这个能力本身就很有用，但如果与LINQ结合起来使用功效更大。lambda表达式与扩展方法的结合把函数式编程完整地引入了C#。

第16章介绍C# 3.0引入的最强大的新特性LINQ。通过C# 3.0里新增的面向LINQ的关键字来使用LINQ表达式，可以把数据查询无缝地集成到代码中。LINQ在C#命令式编程与数据查询的函数式编程之间架起了一座桥梁。LINQ表达式可以用来操作正常的对象，也可以用来操作来自SQL数据库、数据集和XML等多种来源的数据。

致谢

写书是一个漫长而艰巨的过程，在这个过程中我得到了来自朋友和家人的巨大支持。我为之深深感动。如果没有他们的支持，这个过程会更加困难，而且不会取得这样的成效。

我想特别感谢下面这些人，他们对本书的第1版作出了巨大贡献。他们是（排名不分先后）：David Weller, Stephen Toub, Rex Jaeschke, Vladimir Levin, Jerry Maresca, Chris Pels, Christopher T. McNabb, Brad Wilson, Peter Partch, Paul Stubbs, Rufus Littlefield, Tomas Restrepo, John Lambert, Joan Murray, Sheri Cain, Jessica D'Amico, Karen Gettman, Jim Huddleston, Richard Dal Porto, Gary Cornell, Brad Abrams, Ellie Fountain, Nicole Abramowitz以及所有Apress的员工和Shelley Nash。

在第2版的写作过程中，我想感谢以下这些人的帮助和支持。他们是（排名不分先后）：Shawn Wildermuth, Sofia Marchant, Jim Compton, Dominic Shakeshaft, Wes Dyer, Kelly Winquist和Laura Cheu。

如果我还有遗漏，完全是我的失误，绝不是故意的。如果没有你们的支持，我绝对不能完成本书。

谢谢你们！

目　录

第1章 C# 预 览

因为本书的目标读者是有经验的面向对象的开发人员，假定你已经对.NET运行时有了一些了解。Don Box的《.NET本质论第一卷：公共语言运行库》是关于.NET运行时的一本好书，如果想进一步了解.NET运行时，可以参阅此书。另外，了解C#和C++之间的异同也很重要，因此本章从比较二者的区别开始。之后用一个基本的“Hello World”例子来加以说明。如果你有.NET应用程序的开发经验，可以跳过本章。但是，你可能还是想要看看1.4节的内容。

1.1 C#和 C++的区别

C#是一种强类型的面向对象语言，其代码看起来和C++以及Java类似。C#语言设计者的这个决定使C++开发者可在已有的知识结构基础上更高效地应用C#语言。从语法上来讲，C#和C++有些差异，但它们之间大部分的差别是来自语义和行为上的，这是由执行它们的运行时环境的不同带来的。

1.1.1 C#

C#源代码编译成托管代码。我们知道，托管代码是一种中间语言，它介于高级语言（C#）和最低级的语言（汇编语言或机器码）之间。运行的时候，公共语言运行库（Common Language Runtime，CLR）用即时（Just In Time，JIT）编译来动态编译托管代码。就如任何工程产物一样，这种技术有利也有弊。一个显著的弊端就是运行时编译的效率不高。这个过程不同于Perl、JScript等脚本语言所用的解释（interpreting）方式。JIT编译器并不会在每个函数或方法每次调用的时候都编译它们，而只是在第一次调用的时候把托管代码编译成运行平台的本地机器码。因为中间语言的内存占用量比较少，JIT编译的一个显著优势就是应用程序的工作集（working set）减少了。在应用程序执行过程中，只有必要的代码用JIT编译。例如，如果应用程序里面有打印的代码，但用户不打印文档，这段代码就是不需要的，因而JIT编译器不会编译它。另外，CLR在运行时可以动态地优化程序的执行。例如，CLR可以通过重新调整内存中编译后的代码，用某种方式来减少内存管理中的页错误，这都是在运行时做的。考虑到所有这些优势，你就会发现对大部分应用程序来说，是利大于弊的。

注解 事实上，可以选择用原始的中间语言来编写程序，然后在IL汇编程序（ILASM）中调试。但是，这样做很浪费时间。高级语言几乎总是可以提供通过IL代码所获得的全部功能。

1.1.2 C++

与C#不同，C++代码总是编译成本地代码。本地代码是针对编译程序的处理器的机器码。为了方

便，我们约定这里讨论的是本地编译的C++代码，而不是通过C++/CLI实现的托管C++。如果希望本地C++应用程序运行在不同的平台，比如32位平台和64位平台，那么必须分别单独编译。通常，本地的二进制输出不是跨平台兼容的。

而CLR构建的基础公共语言基础架构（Common Language Infrastructure，CLI）是一个国际标准[①]，所以中间语言是跨平台兼容的。这个标准正迅速被大力推动，也正在Microsoft Windows之外的平台上得到实现。

注解　建议你看看Mono团队所取得的成果，他们正在其他平台创建开源的虚拟执行系统（VES）。[②]

CLI定义了托管代码的可移植的执行（Portable Executable，PE）文件格式。因此可以在Windows平台上编译一个C#程序，这个输出在Windows和Linux平台都可以执行而不需要重新编译，因为甚至文件格式都是标准的[③]。这个级别的可移植性非常便利，这是COM/DCOM设计者以前梦寐以求的，但由于种种原因，它没有在这个层次取得跨越异构平台的成功[④]。失败的主要原因之一就是，COM对描述类型及其依赖关系缺乏有足够表现力的、可扩展的机制。CLI通过引入元数据轻而易举地解决了这个问题。我们将在第2章介绍元数据。

1.1.3　CLR 垃圾回收

CLR中的关键工具之一是垃圾回收器（garbage collector，GC）。GC让你从分配和释放内存的负担中解脱出来，而这些内存管理工作是很多软件错误的根源。然而GC并没有为你解除所有的资源处理负担，从第4章可以看到这一点。例如，文件句柄作为一个资源必须在使用后释放，就像内存一样。GC只直接管理内存资源，其他的比如数据库连接和文件句柄，可以用一个终结器（finalizer）（将在第13章介绍）在GC通知你对象将要被摧毁的时候来释放。但是，一个更好的办法是用Disposable模式来完成这个任务，我们将在第4章和第13章介绍。

注解　CLR间接引用所有引用类型的对象，和C++里面的引用和指针类似，但C#没有指针的语法定义。在C#中定义一个引用类型的时候，事实上预留了一个与类型关联的存储位置，这个位置在堆或者栈上保存着对象的引用。当把一个变量的对象引用复制到另一个变量时，会得到引用到同一个对象的两个变量。所有的引用类型实例都位于托管堆上。CLR管理这些对象的位置，如果需要移动，它会更新那些指向被移动对象的引用去指向新的位置。CLR里面也有值类型，它们的实例存活在栈上或作为托管堆上对象的一个域。它们的用法有很多限制和区别。通常在需要轻量级的结构来管理相关数据的时候需要用到它们。值类型在对非可变（immutable）的数据块建模的时候也有用，第4章将更详细地讨论这个问题。

---

① 可以在www.ecma-international.org 找到CLI标准文档Ecma-335。另外，Ecma-334是C#语言的标准文档。

② 可以在www.mono-project.com网站上找到关于Mono项目的内容。

③ 当然，目标平台上要安装好程序所需要的所有库文件。有了.NET标准库，很快就要成为现实了。例如，浏览一下www.go-mono.com/docs/，就会看到Mono项目的库文件覆盖有多广。

④ 要了解这段惨痛的历史，我推荐阅读Don Box和Chris Sell的《.NET本质论第一卷：公共语言运行库》[这个标题让我们相信第二卷随时会出版，我们希望它别像Mel Brooks的《帝国时代：第一部》那样。（Mel Brooks是美国著名导演；《帝国时代》只出了第一部，没有第二部）]

用C#可以快速开发应用，而不必处理如C++环境里面那么繁琐的细节。同时，C#是一种让C++或Java开发者感到熟悉的语言。

1.2 C#程序示例

我们来进一步看看C#程序是什么样子。首先看一下大家都喜闻乐见的“Hello World!”程序。用C#编写的控制台版本是这个样子的：

```
class EntryPoint {
    static void Main() {
        System.Console.WriteLine( "Hello World!" );
    }
}
```

注意这个C#程序的结构。它声明了一个类型（名为EntryPoint的类）和类的一个成员（名为Main的方法）。这和C++不同，C++是在头文件里面定义类型，然后在一个单独的编译单元定义它，通常用a.cpp之类的文件名。另外，元数据（元数据描述了模块里面的所有数据，由C#编译器透明生成）使C++里面的前置声明和包含（inclusion）不再需要。实际上，C#中甚至不存在前置声明。

C++程序员会发现对静态的Main方法比较熟悉，只是名字的首字母变成了大写。所有的程序都需要一个入口点，对C#而言就是静态Main方法。当然还有更多的区别，例如，Main方法在类里面声明。在C#里，所有的方法都必须在类型定义里面声明，而没有C++里面的静态、自由函数。Main方法的返回值可以是int或void，这个取决于你的需要。在这个例子中，Main没有参数，但如果需要访问命令行参数，Main方法可以声明一个参数（一个字符串数组）来访问它们。

注解 如果应用程序含有多个含静态Main方法的类型，则可以通过/main编译开关来选择一个。

你可能注意到WriteLine调用看起来比较繁琐。必须用类名Console来限定这个方法名字，同时还必须指定Console所在的命名空间（这里是System）。.NET（C#也一样）支持命名空间来避免巨大的全局命名空间内名字的冲突。当然，全限定名（fully qualified name），包括命名空间，是不必每次输入的，C#提供了using指令来解决这个问题。using与Java里的import和C++里的using namespace类似。因此可以对上面的程序稍加修改，得到代码清单1-1。

代码清单1-1 hello_world.cs

```
using System;

class EntryPoint {
    static void Main() {
        Console.WriteLine( "Hello World!" );
    }
}
```

有了using System指令，可以在调用Console.WriteLine的时候省略System命名空间。

可以在一个Windows 命令行窗口执行下面的命令来编译这个例子：

```
csc.exe /r:mscorlib.dll /target:exe hello_world.cs
```

我们来仔细看看这个命令行做了什么：