



The C# Programming Language

Third Edition

C#程序设计语言

(原书第3版)

Anders Hejlsberg

Mads Torgersen

(美) Scott Wiltamuth 著

Peter Golde

顾雁宏 徐旭铭 译



机械工业出版社
China Machine Press

开发人员专业技术丛书

The C# Programming Language

Third Edition

C#程序设计语言

(原书第3版)

Anders Hejlsberg

Mads Torgersen

(美) Scott Wiltamuth 著

Peter Golde

Peter Golde

顾雁宏 徐旭铭 译



机械工业出版社
China Machine Press

C#语言结合了快速应用开发语言的高效和C/C++语言的强大。现在C# 3.0又加入了函数式编程技术和语言集成查询(Language INtegrated Query, LINQ)。本书正是C# 3.0的权威技术指南。

这一版由C#的缔造者Anders Hejlsberg和他的同事Mads Torgersen、Scott Wiltamuth和Peter Golde编写, 全部内容都更新到了C# 3.0版。本书提供了C#语言完整的规格说明, 以及大量的描述、参考资料、范例代码, 和来自9位卓越的C#大师的详细注解。

这些注解(这一版新的特色)所达到的深度和广度是很难在其他书中找到的。本书的正文介绍了C#的概念, 而这些恰到好处的注解则解释了为什么这些特性是重要的, 应该怎么使用它们, 它们和其他语言的关系是什么, 以及它们是如何进化而来的。

对任何希望深入理解C#的程序员来说, 本书都是绝对不容错过的参考经典。

Simplified Chinese edition copyright © 2009 by Pearson Education Asia Limited and China Machine Press.

Original English language title The C# Programming Language, 3E: (ISBN 978-0-321-56299-9) by Anders Hejlsberg, Mads Torgersen, Scott Wiltamuth, Peter Golde, Copyright © 2009.

All rights reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Addison-Wesley.

本书封面贴有Pearson Education (培生教育出版集团) 激光防伪标签, 无标签者不得销售。

版权所有, 侵权必究

本书法律顾问 北京市展达律师事务所

本书版权登记号: 图字: 01-2009-4560

图书在版编目(CIP)数据

C#程序设计语言(原书第3版) / (美) 海杰尔斯伯格(Hejlsberg, A.) 等著; 顾雁宏等译.
—北京: 机械工业出版社, 2009.9

(开发人员专业技术丛书)

书名原文: C# Programming Language, Third Edition.

ISBN 978-7-111-28261-7

I. C… II. ①海… ②顾… III. C语言—程序设计 IV. TP312

中国版本图书馆CIP数据核字(2009)第161240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: 陈佳媛

北京京师印务有限公司印刷

2010年1月第1版第1次印刷

186mm × 240mm · 35.25印张

标准书号: ISBN 978-7-111-28261-7

定价: 79.00元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991; 88361066

购书热线: (010) 68326294; 88379649; 68995259

投稿热线: (010) 88379604

读者信箱: hzjsj@hzbook.com

序

自2000年夏.NET发布以来已经有8个年头了。对我来说，当时.NET最重要的两点就是结合了托管代码的本地执行和用于程序之间通信的XML消息机制。不过那个时候我还没有意识到C#会变得那么重要。

C#从一开始就是程序员们理解和使用.NET的主要手段。如果你问一个普通的.NET程序员一个值类型和一个引用类型的区别是什么，通常的回答都是“结构和类的区别”，而非“是否是从System.ValueType继承而来的类型”。为什么？因为我们都是用语言，而不是通过API来和运行时（更重要的是，其他人）交流想法和意图的。

如果没有一门出色的语言，一个平台要想成功是不可想象的。C#最初就为人们如何看待.NET打下了坚实的基础。随着.NET的不断发展，C#的重要性也与日俱增，诸如迭代器和真正的闭包（也叫匿名方法）都是用C#编译器实现的纯语言特性，而不是平台特性。C# 3.0的发布更意味着C#成为了.NET不断创新的主角，它引入了标准化的查询操作符，简洁的lambda表达式、扩展方法，以及在运行时访问表达式树的能力——这些都是通过语言和编译器实现的。

说到C#就一定会提到它的缔造者Anders Hejlsberg。我非常荣幸地在C# 3.0设计阶段连续好几个月参与了C#的设计会议，Anders的工作让我大开眼界。他那种深谙程序员喜欢什么和不喜欢什么的天赋实在是一流——同时他又能和设计团队紧密合作，并最终获得最佳的设计方案。

特别是在C# 3.0上，Anders在从函数式语言社区获取灵感并将它们带给广大群众的过程中展现出无与伦比的能力。要知道这绝对不是一件容易的事情。Guy Steele曾经在谈论Java时说到：“我们没打算要吸引Lisp程序员，我们的目标是C++程序员。我们成功地把他们从转向Lisp的路上吸引了过来。”当我看到C# 3.0的时候，我就知道C#已经至少获得了一名C++程序员（就是我自己）的青睐。

即使C#很出色，但是人们还是需要一份用自然语言（这里是英文）和一些范式（BNF）写成的文档来帮助他们抓住要点，以及任何消除晦涩的地方。而你手中的这本书正是这样的一份文档。据我的经验，我敢说每个.NET程序员在读这本书的时候都至少会有一次“啊，原来如此”的感叹，它能让你的水平更上一层楼。

享受它吧。

Don Box
2008年7月

作者简介

Anders Hejlsberg是编程界的传奇人物。他是C#语言的架构师，同时也是微软技术专家。他在1996年加入微软，之前13年的职业生涯则是在Borland度过，他曾经是Delphi和Turbo Pascal的首席架构师。

Mads Torgersen是微软的资深程序经理。作为C#的程序经理，他负责召开C#语言的设计会议及维护C#语言的规范。在2005年加入微软之前，Mads是奥尔胡斯大学的副教授，主要教授和研究面向对象编程语言。在那里，他领导的小组设计实现了Java的泛型通配符。

Scott Wiltamuth是Visual Studio的合作程序经理。他在微软做过很多面向程序员的项目，包括Visual Basic、VBScript、Jscript、Visual J++和Visual C#。Scott是C#语言的设计师之一，他拥有斯坦福大学计算机科学硕士学位。

Peter Golde在离开微软之前是微软C#编译器的首席程序员。他作为微软在ECMA委员会（这个委员会负责了C#的标准化工作）的主要代表，领导实现了编译器并参与了语言的设计。

注解者简介

Brad Abrams是CLR和.NET架构团队在微软的创始人之一。他从1998年起就参与了.NET架构的设计，现在他是微软UI框架和服务团队的首席程序经理。在Brad的博客<http://blogs.msdn.com/BradA>上可以读到 he 最新的想法。

Joseph Albahari是《C# 3.0 in a Nutshell》(O'Reilly, 2007)、《C# 3.0 Pocket Reference》(O'Reilly, 2008)和《LINQ Pocket Reference》(O'Reilly, 2008)的合著者。他在保健、教育和电信领域里作为资深程序员和软件架构师有17年的经验，他还是LINQPad的作者，这是一个可以用LINQ交互查询数据库的工具。

Krzysztof Cwalina是.NET架构团队的程序经理。他最初在微软负责了Framework 1.0的API设计。现在他主要负责开发、引导和推行.NET架构的设计思想。他是《Framework Design Guidelines》(Addison-Wesley, 2005)的作者之一。他的博客是<http://blogs.msdn.com/kcwalina>。

Don Box是微软杰出的工程师，他开发的声明式语言和工具大大简化了应用程序和服务的开发，同时他还参与设计了语言、框架和第一手的用户体验来帮助人们把他们对软件的想法和要求变成可执行的代码。Don在软件开发领域还是一名广受尊敬的作家，负责编辑了微软.NET开发系列丛书(Addison-Wesley)，他还参与编辑了《C++ Report》，《Microsoft Systems Journal (MSJ)》和《MSDN Magazine》。

Jesse Liberty (“Silverlight专家”)是微软的高级程序经理，还是好几本畅销书的作者，包括了O'Reilly Media即将出版的《Programming Silverlight 2》和《Programming C# 3.0》。Jesse有20多年的编程经历。他的个人主页是www.JesseLiberty.com。

Eric Lippert是微软C#编译器团队的资深程序员。他参与设计和实现了Visual Basic、VBScript、Jscript、C#和Visual Studio Tools for Office。他的博客上全是这方面的资料，网址是<http://blogs.msdn.com/EricLippert>。

Fritz Onion是.NET程序员培训公司Pluralsight的创办人之一。Fritz领衔了Pluralsight的Web开发课程，在全球教授ASP.NET、Ajax和Silverlight。他是广受好评的《Essential ASP.NET》和《Essential ASP.NET 2.0》(都是由Addison-Wesley出版)的作者。同时他还为很多专业的程序员杂志撰写文章。他的博客是<http://pluralsight.com/fritz>。

Vladimir Reshetnikov是微软Visual C#的MVP。他有超过6年的软件开发经验，以及4年Microsoft.NET和C#的经验。

Chris Sells是微软Connected Systems分部的程序经理。Chris编写了好几本书，包括了《Programming WPF》(O'Reilly, 2007)、《Windows Forms 2.0 Programming》(Addison-Wesley, 2006)和《ATL Internal》(Addison-Wesley, 1999)。闲暇的时候，他还主持了很多研讨会，并且在微软内部的产品讨论组里也相当活跃。要想了解Chris和他的项目，请访问www.sellsbrothers.com。

Bill Wagner是SRT Solutions的创始人、微软的区域主管和一名C# MVP。他是《Effective C#》(Addison-Wesley, 2005)和《More Effective C#》(Addison-Wesley, 2009)的作者，《Visual Studio Magazine》的专栏作家，同时还是一位C#开发者中心的贡献者。

前言

C#项目始于10年之前的1998年12月，当初的目标是要为全新的（还未命名的）.NET平台创造一种简单、现代化、面对对象和类型安全的编程语言。一路走来，C#也算是历经坎坷。现在这门语言已经有超过一百万的程序员，发布了3个版本，其中每一版都加入许多重大的新特性。

而本书也一样出到了第3版。作为一本C#语言完整的技术规范，第3版和之前的两个版本有很多不同的地方。其中最显著的当然就是它覆盖了所有C# 3.0的新特性，包括对象和集合初始化、匿名类型、lambda表达式、查询表达式和局部方法。绝大多数这些特性都是为了支持一种更加函数式和声明式的编程风格，具体来讲，就是语言集成查询（LINQ），它提供了一种统一的方式来查询不同数据源的数据。而LINQ又大量构建在一些C# 2.0引入的特性之上，比如泛型、迭代器和局部类型。

第3版中的另一个变化是所有的规范说明都重新组织了。在第2版里，C# 2.0引入的特性和原本C# 1.0中的特性是分开描述的。要是再加上一套新特性的话，这个方法就没用了——读者需要从3个不同的地方获取相关的信息，本书的实用性将会大打折扣。所以，本书会按照主题来组织内容，3个语言版本中的特性被放在一起介绍。

最后一个和之前版本区别的地方就是本书包含了大量的注解。我们很幸运地请到了一些世界级的C#和.NET专家，在书中各处以注解的形式为我们提供了出色的指导、背景和观点。我们很高兴这些注解和本书的核心内容相辅相成，让C#的这些特性跃然纸上。

创造C#语言是很多人共同努力的结果。C# 1.0的设计团队由Anders Hejlsberg、Scott Wiltamuth、Peter Golde、Peter Sollich和Eric Gunnerson组成。而C# 2.0团队的成员有Anders Hejlsberg、Peter Golde、Peter Hallam、Shon Katzenberger、Todd Proebsting和Anson Horton。此外，C#和.NET公共语言运行时（Common Language Runtime）中泛型的设计和实现是基于微软研究院里Don Syme和Andrew Kennedy构建的“Gyro”原型之上。C# 3.0则是由Anders Hejlsberg、Peter Hallam、Shon Katzenberger、Dinesh Kulkarni、Erik Meijer、Mads Torgersen和Matt Warren负责设计。

要感谢所有影响了C#设计的人是不可能的，尽管如此我们还是要感谢你们。闭门造车是不会有好设计的，所以来自我们庞大和热情的程序员社区的意见和建议都是无价的。

C#是（并将继续是）我们工作过的最有挑战性和最刺激的项目之一。希望你们用得开心，我们也做得高兴。

Anders Hejlsberg

Mads Torgersen

Scott Wiltamuth

华盛顿西雅图

2008年7月

目 录

序	
作者简介	
注解者简介	
前言	
第1章 介绍	1
1.1 Hello, World	2
1.2 程序结构	3
1.3 类型和变量	5
1.4 表达式	7
1.5 语句	9
1.6 类和对象	13
1.6.1 成员	13
1.6.2 访问控制	14
1.6.3 类型参数	14
1.6.4 基类	15
1.6.5 字段	16
1.6.6 方法	17
1.6.7 其他函数成员	25
1.7 结构	30
1.8 数组	32
1.9 接口	34
1.10 枚举	35
1.11 委托	37
1.12 特性	40
第2章 词法结构	42
2.1 程序	42
2.2 文法	42
2.2.1 文法表示法	42
2.2.2 词法文法	43
2.2.3 语法规文法	43
2.3 词法分析	43
2.3.1 行终结符	44
2.3.2 注释	45
2.3.3 空白符	46
2.4 标记	46
2.4.1 Unicode字符转义序列	46
2.4.2 标识符	47
2.4.3 关键字	49
2.4.4 字量	50
2.4.5 操作符和标点符号	56
2.5 预处理指令	56
2.5.1 条件编译符号	57
2.5.2 预处理表达式	58
2.5.3 声明指令	58
2.5.4 条件编译指令	59
2.5.5 诊断指令	62
2.5.6 区域指令	63
2.5.7 行指令	63
2.5.8 编译指示指令	64
第3章 基本概念	66
3.1 应用程序起始	66
3.2 应用程序终止	67
3.3 声明	67
3.4 成员	70
3.4.1 命名空间成员	70
3.4.2 结构成员	70
3.4.3 枚举成员	71
3.4.4 类成员	71
3.4.5 接口成员	71
3.4.6 数组成员	71
3.4.7 委托成员	71
3.5 成员访问	71
3.5.1 声明可访问性	72

3.5.2 可访问域	73	4.5 类型参数	107
3.5.3 实例成员的保护访问	75	4.6 表达式树类型	108
3.5.4 访问限制	77	第5章 变量	110
3.6 签名和重载	78	5.1 变量类别	110
3.7 作用域	79	5.1.1 静态变量	110
3.7.1 名字隐藏	82	5.1.2 实例变量	110
3.8 命名空间和类型名称	84	5.1.3 数组元素	111
3.8.1 完全限定名	86	5.1.4 值参数	111
3.9 自动化内存管理	87	5.1.5 引用参数	111
3.10 执行顺序	91	5.1.6 输出参数	112
第4章 类型	92	5.1.7 局部变量	112
4.1 值类型	92	5.2 默认值	113
4.1.1 System.ValueType类型	93	5.3 明确赋值	114
4.1.2 默认构造函数	93	5.3.1 初始赋值的变量	114
4.1.3 结构类型	94	5.3.2 未赋初值的变量	115
4.1.4 简单类型	94	5.3.3 确定明确赋值的精确规则	115
4.1.5 整数类型	95	5.4 变量引用	125
4.1.6 浮点数类型	96	5.5 变量引用的原子性	125
4.1.7 decimal类型	97	第6章 转换	126
4.1.8 bool类型	98	6.1 隐式转换	126
4.1.9 枚举类型	99	6.1.1 标识转换	127
4.1.10 可空值类型	99	6.1.2 隐式数字转换	127
4.2 引用类型	100	6.1.3 隐式枚举转换	127
4.2.1 类类型	100	6.1.4 隐式可空值转换	128
4.2.2 Object类型	101	6.1.5 Null字量转换	128
4.2.3 String类型	101	6.1.6 隐式引用转换	128
4.2.4 接口类型	101	6.1.7 装箱转换	129
4.2.5 数组类型	101	6.1.8 隐式常量表达式转换	130
4.2.6 委托类型	101	6.1.9 带类型参数的隐式转换	130
4.3 装箱和拆箱	102	6.1.10 自定义隐式转换	130
4.3.1 装箱转换	102	6.1.11 匿名函数转换和方法组转换	130
4.3.2 拆箱转换	103	6.2 显式转换	130
4.4 构造类型	104	6.2.1 显式数字转换	131
4.4.1 类型实参	105	6.2.2 显式枚举转换	132
4.4.2 开放式和封闭式类型	105	6.2.3 显式可空值转换	133
4.4.3 绑定和未绑定类型	106	6.2.4 显式引用转换	133
4.4.4 满足限制	106	6.2.5 拆箱转换	134

6.2.6 带类型参数的显式转换	134	7.5.2 简单名字	170
6.2.7 自定义显式转换	135	7.5.3 括号表达式	172
6.3 标准转换	136	7.5.4 成员访问	173
6.3.1 标准隐式转换	136	7.5.5 调用表达式	175
6.3.2 标准显式转换	136	7.5.6 元素访问	180
6.4 自定义转换	136	7.5.7 this访问	182
6.4.1 允许的自定义转换	136	7.5.8 base访问	182
6.4.2 提升转换操作符	137	7.5.9 后缀递增和递减操作符	183
6.4.3 自定义转换的计算	137	7.5.10 new操作符	184
6.4.4 自定义隐式转换	138	7.5.11 typeof操作符	195
6.4.5 自定义显式转换	139	7.5.12 checked和unchecked操作符	197
6.5 匿名函数转换	140	7.5.13 默认值表达式	199
6.5.1 匿名函数到委托类型转换的计算	140	7.5.14 匿名方法表达式	200
6.5.2 匿名函数到表达式树类型转换的 计算	141	7.6 一元操作符	200
6.5.3 实现举例	141	7.6.1 一元加号操作符	200
6.6 方法组转换	144	7.6.2 一元减号操作符	200
第7章 表达式	147	7.6.3 逻辑否操作符	201
7.1 表达式分类	147	7.6.4 按位求补操作符	201
7.1.1 表达式的值	148	7.6.5 前缀递增和递减操作符	202
7.2 操作符	148	7.6.6 转换表达式	203
7.2.1 操作符优先级和结合性	149	7.7 算术操作符	204
7.2.2 操作符重载	150	7.7.1 乘法操作符	204
7.2.3 一元操作符重载决策	151	7.7.2 除法操作符	205
7.2.4 二元操作符重载决策	151	7.7.3 求余操作符	206
7.2.5 候选自定义操作符	152	7.7.4 加法操作符	207
7.2.6 数字提升	152	7.7.5 减法操作符	209
7.2.7 提升操作符	154	7.8 移位操作符	211
7.3 成员查找	154	7.9 关系和类型测试操作符	212
7.3.1 基础类型	156	7.9.1 整数比较操作符	212
7.4 函数成员	156	7.9.2 浮点数比较操作符	213
7.4.1 参数列表	158	7.9.3 小数比较操作符	214
7.4.2 类型推导	160	7.9.4 布尔值相等操作符	214
7.4.3 重载决策	165	7.9.5 枚举比较操作符	215
7.4.4 函数成员调用	168	7.9.6 引用类型相等操作符	215
7.5 基础表达式	169	7.9.7 字符串相等操作符	217
7.5.1 字量	170	7.9.8 委托相等操作符	217
		7.9.9 相等操作符和null	218

7.9.10 is操作符	218	8.5.2 局部常量声明	257
7.9.11 as操作符	218	8.6 表达式语句	258
7.10 逻辑操作符	220	8.7 选择语句	258
7.10.1 整数逻辑操作符	220	8.7.1 if语句	258
7.10.2 枚举逻辑操作符	220	8.7.2 switch语句	259
7.10.3 布尔值逻辑操作符	221	8.8 迭代语句	263
7.10.4 可空值布尔逻辑操作符	221	8.8.1 while语句	263
7.11 条件逻辑操作符	221	8.8.2 do语句	264
7.11.1 布尔条件逻辑操作符	222	8.8.3 for语句	264
7.11.2 自定义条件逻辑操作符	222	8.8.4 foreach语句	266
7.12 Null拼接操作符	223	8.9 跳转语句	269
7.13 条件操作符	224	8.9.1 break语句	270
7.14 匿名函数表达式	225	8.9.2 continue语句	270
7.14.1 匿名函数签名	227	8.9.3 goto语句	271
7.14.2 匿名函数主体	227	8.9.4 return语句	272
7.14.3 重载决策	228	8.9.5 throw语句	273
7.14.4 外部变量	229	8.10 try语句	274
7.14.5 匿名函数表达式的计算	232	8.11 checked和unchecked语句	277
7.15 查询表达式	232	8.12 lock语句	277
7.15.1 查询表达式里的歧义	234	8.13 using语句	279
7.15.2 查询表达式翻译	234	8.14 yield语句	281
7.15.3 查询表达式模式	242	第9章 命名空间	284
7.16 赋值操作符	244	9.1 编译单元	284
7.16.1 简单赋值	244	9.2 命名空间声明	284
7.16.2 组合赋值	247	9.3 Extern别名	286
7.16.3 事件赋值	248	9.4 using指令	286
7.17 表达式	248	9.4.1 using别名指令	287
7.18 常量表达式	248	9.4.2 using命名空间指令	290
7.19 布尔表达式	249	9.5 命名空间成员	292
第8章 语句	251	9.6 类型声明	292
8.1 终点和可及性	251	9.7 命名空间别名限定符	293
8.2 块	253	9.7.1 别名的唯一性	294
8.2.1 语句列表	253	第10章 类	295
8.3 空语句	254	10.1 类声明	295
8.4 标签语句	254	10.1.1 类修饰符	295
8.5 声明语句	255	10.1.2 partial修饰符	297
8.5.1 局部变量声明	255	10.1.3 类型形参	297

10.1.4 类基础规范	298	10.6.8 局部方法	349
10.1.5 类型形参限制	300	10.6.9 扩展方法	349
10.1.6 类主体	305	10.6.10 方法主体	350
10.2 局部类型	305	10.6.11 方法重载	351
10.2.1 特性	306	10.7 属性	351
10.2.2 修饰符	306	10.7.1 静态属性和实例属性	352
10.2.3 类型形参和限制	306	10.7.2 访问器	352
10.2.4 基类	307	10.7.3 自动实现的属性	358
10.2.5 基础接口	307	10.7.4 可访问性	358
10.2.6 成员	308	10.7.5 虚拟、密封、覆写和抽象访问器	360
10.2.7 局部方法	308	10.8 事件	362
10.2.8 名字绑定	311	10.8.1 类似字段的事件	364
10.3 类成员	312	10.8.2 事件访问器	365
10.3.1 实例类型	313	10.8.3 静态事件和实例事件	367
10.3.2 构造类型的成员	313	10.8.4 虚拟、密封、覆写和抽象访问器	367
10.3.3 继承	315	10.9 索引	367
10.3.4 new修饰符	315	10.9.1 索引重载	371
10.3.5 访问修饰符	316	10.10 操作符	371
10.3.6 组成类型	316	10.10.1 一元操作符	372
10.3.7 静态成员和实例成员	316	10.10.2 二元操作符	373
10.3.8 嵌套类型	317	10.10.3 转换操作符	374
10.3.9 保留成员名	321	10.11 实例构造函数	376
10.4 常量	323	10.11.1 构造函数初始化语句	377
10.5 字段	325	10.11.2 实例字段初始化语句	379
10.5.1 静态字段和实例字段	326	10.11.3 构造函数的执行	379
10.5.2 只读字段	327	10.11.4 默认构造函数	381
10.5.3 易失字段	329	10.11.5 私有构造函数	381
10.5.4 字段初始化	330	10.11.6 可选的实例构造函数参数	382
10.5.5 字段初始化语句	330	10.12 静态构造函数	383
10.6 方法	333	10.13 析构函数	385
10.6.1 方法形参	335	10.14 迭代器	387
10.6.2 静态和实例方法	341	10.14.1 计数接口	388
10.6.3 虚拟方法	341	10.14.2 枚举接口	388
10.6.4 覆写方法	343	10.14.3 Yield类型	388
10.6.5 密封方法	346	10.14.4 计数对象	388
10.6.6 抽象方法	347	10.14.5 枚举对象	390
10.6.7 外部方法	348	10.14.6 实现举例	391

第11章 结构	399	13.2.1 接口方法	420
11.1 结构声明	399	13.2.2 接口属性	420
11.1.1 结构修饰符	399	13.2.3 接口事件	421
11.1.2 partial修饰符	400	13.2.4 接口索引	421
11.1.3 结构接口	400	13.2.5 接口成员访问	421
11.1.4 结构主体	400	13.3 完全限定接口成员名	423
11.2 结构成员	400	13.4 接口实现	424
11.3 类和结构的区别	401	13.4.1 显式接口成员实现	425
11.3.1 值语义	401	13.4.2 实现接口的唯一性	427
11.3.2 继承	402	13.4.3 泛型方法的实现	428
11.3.3 赋值	402	13.4.4 接口映射	429
11.3.4 默认值	402	13.4.5 接口实现继承	432
11.3.5 装箱和拆箱	403	13.4.6 重新实现接口	433
11.3.6 this的含义	405	13.4.7 抽象类和接口	435
11.3.7 字段初始化语句	405	第14章 枚举	437
11.3.8 构造函数	406	14.1 枚举声明	437
11.3.9 析构函数	407	14.2 枚举修饰符	438
11.3.10 静态构造函数	407	14.3 枚举成员	438
11.4 结构举例	407	14.4 System.Enum类型	440
11.4.1 数据库整数类型	407	14.5 枚举值和操作	440
11.4.2 数据库布尔类型	409	第15章 委托	442
第12章 数组	412	15.1 委托声明	442
12.1 数组类型	412	15.2 委托兼容性	444
12.1.1 System.Array类型	413	15.3 委托实例化	445
12.1.2 数组和泛型IList接口	413	15.4 委托调用	445
12.2 数组创建	414	第16章 异常	448
12.3 数组元素访问	414	16.1 异常产生的原因	448
12.4 数组成员	414	16.2 System.Exception类	448
12.5 数组协变	414	16.3 异常是如何处理的	449
12.6 数组初始化语句	415	16.4 常见的异常类	449
第13章 接口	417	第17章 特性	450
13.1 接口声明	417	17.1 特性类	450
13.1.1 接口修饰符	417	17.1.1 特性的用法	450
13.1.2 partial修饰符	418	17.1.2 位置和已命名参数	452
13.1.3 基础接口	418	17.1.3 特性形参类型	453
13.1.4 接口主体	419	17.2 特性规范	453
13.2 接口成员	419	17.3 特性实例	458

17.3.1 特性的编译	459	18.5.2 指针成员访问	475
17.3.2 在运行时获取一个特性实例	459	18.5.3 指针元素访问	476
17.4 保留特性	459	18.5.4 取地址操作符	476
17.4.1 AttributeUsage特性	459	18.5.5 指针递增和递减	477
17.4.2 Conditional特性	460	18.5.6 指针算术	477
17.4.3 Obsolete特性	464	18.5.7 指针比较	478
17.5 用于互操作的特性	465	18.5.8 sizeof操作符	479
17.5.1 和COM以及Win32组件互操作	465	18.6 fixed语句	479
17.5.2 和其他.NET语言互操作	465	18.7 定长缓冲区	483
第18章 不安全的代码	466	18.7.1 定长缓冲区声明	483
18.1 不安全的上下文	466	18.7.2 表达式里的定长缓冲区	485
18.2 指针类型	469	18.7.3 明确赋值检查	486
18.3 固定变量和可移动变量	471	18.8 栈分配	486
18.4 指针转换	472	18.9 动态内存分配	487
18.4.1 指针数组	473	附录A 文档注释	490
18.5 表达式里的指针	474	附录B 文法	512
18.5.1 指针间接寻址	474	附录C 参考资料	549

第 1 章 介 绍

C#（发音同“see sharp”）是一门简单、现代化、面对对象和类型安全的编程语言。C#属于C语言家族，任何C、C++或Java程序员都不会觉得它很陌生。C#在ECMA International的标准是ECMA-334，在ISO/IEC上的标准是ISO/IEC 23270。微软.NET Framework上的C#编译器的实现同时遵循了这两个标准。

C#不只是一门面对对象的语言，它还包含了对面向组件（component-oriented）编程的支持。现代的软件设计越来越依赖于自包含和自描述的功能包形式的软件组件。这类组件的重要之处在于它们提供了一个带有属性（property）、方法（method）和事件（event）的编程模型，拥有提供了关于组件声明信息的特性（attribute），另外它们还包含了自身的文档。C#在语言级别上直接支持了这些概念，令C#可以很自然地创建和使用软件组件。

C#还提供了一些特性来帮助构建健壮、耐用的应用程序：垃圾收集（Garbage Collection）会自动回收不再使用的对象所占用的内存；异常处理（exception handling）提供了一种结构化且可扩展的方式来检测错误和恢复；而语言的类型安全（type-safe）设计则可以防止读取未初始化的变量、数组越界或进行未检查的类型转换。

C#拥有统一的类型系统（unified type system）。所有的C#类型，包括int和double这样的基础类型，都是从根类型object继承而来。所以，所有的类型都具有一些通用的操作，任何类型的值都可以通过一致的方式进行保存、传递和操作。此外，C#还支持用户自定义引用类型和值类型，允许动态分配对象和轻型结构的内联（in-line）存储。

为了保证C#程序和类库能以兼容的方式向前发展，C#在设计过程中非常注意版本控制（versioning）。很多编程语言都对这一点重视不够，所以当新版本的依赖库被引入时，用这些语言编写的程序都无谓地失灵了。C#设计中的很多方面都直接受到了版本控制考量的影响，包括区分virtual和override修饰符、方法重载的规则，以及对接口成员显式声明的支持。

在本章的剩余部分里，我们将介绍C#语言的重要特性。稍后的章节会更细致（更精确）地讲解一些规则和例外，不过这里先尽量做一个简明的介绍。这让读者可以先对语言有一个初步的认识，以便及早开始编程和理解稍后的章节。

CHRIS SELLS 我绝对同意“现代化、面对对象和类型安全”的论述，不过C#已经不再是一门简单的语言了。但是，随着C# 2.0加入的泛型和匿名委托，以及C# 3.0引入的LINQ特性，其程序本身正在变得更加简单易读，维护起来也更加容易——这些正是任何编程语言的梦想。

ERIC LIPPERT C#还越来越像一门函数式编程语言。诸如类型推导（type inference）、lambda表达式和一元查询推导式（monadic query comprehension）这样的特性让传统的面对对象程序员可以利用函数式编程的思想来增加语言的表达能力。

1.1 Hello, World

介绍编程语言时最经典的自然就是“Hello, World”程序了。这里是C#版：

```
using System;

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

C#的源文件通常是以.cs结尾。假设这个“Hello, World”程序被保存为hello.cs，那么就可以在命令行下用微软C#编译器这样编译程序

```
csc hello.cs
```

它会生成一个可执行文件hello.exe。这个程序执行的输出为

```
Hello, World
```

“Hello, World”程序的开始通过using指令引用了System命名空间。命名空间提供了一种层次化的方式来组织C#程序和类库。它可以包含类型和其他命名空间，例如，System命名空间包含了诸如在程序中引用的Console类的多种类型；以及其他一些命名空间，比如IO和Collections。通过using指令引用某个命名空间时，你就可以不加前缀地使用这个命名空间里的类型。在这个程序里的using指令让程序可以把System.Console.WriteLine简写为Console.WriteLine。

“Hello, World”程序声明的Hello类只有一个成员方法Main。Main方法在声明时要带上static修饰符。实例方法可以通过关键字this来引用自身，而静态方法在使用时无需引用某个特定对象。按照惯例，静态方法Main是程序的入口点。

程序的输出是由命名空间System里的Console类的WriteLine方法产生的。在默认情况下，微软的C#编译器会自动引用这个由.NET框架类库提供的类。注意C#本身并没有一个单独的运行库，.NET框架就是C#的运行库。

BRAD ABRAMS 请注意：有趣的是这里Console.WriteLine()其实是Console.Out.WriteLine的简写。Console.Out是一个属性，它返回了一个设计用来专门向控制台输出的System.IO.TextWriter基类的实现。上面的例子写成这样也是正确的：

```
using System;
class Hello
{
    static void Main() {
        Console.Out.WriteLine("Hello, World");
    }
}
```

在框架设计的初期，我们就注意到了在C#语言规格说明中的这节给语言所带来的复杂性。我们选择了方便地重载Console来让“Hello, World”更加容易编写。事实上，今天你会发现你几乎用不着去调用Console.Out.WriteLine()了，大家也都很喜欢这个决定。

1.2 程序结构

在C#中关键的组织概念是程序、命名空间、类型、成员和汇编代码。C#的程序由一个或多个源文件组成。程序声明了类型，而类型包含了成员。类型可以被组织到命名空间里。类和接口都是类型的例子。变量、方法、属性和事件则是成员的例子。当编译C#程序的时候，它们通常都会被打包到汇编代码里去。它们的文件扩展名可是.exe或是.dll，分别取决于它们实现的是应用程序还是类库。

下面这个例子

```
using System;

namespace Acme.Collections
{
    public class Stack
    {
        Entry top;

        public void Push(object data) {
            top = new Entry(top, data);
        }

        public object Pop() {
            if (top == null) throw new InvalidOperationException();
            object result = top.data;
            top = top.next;
            return result;
        }

        class Entry
        {
            public Entry next;
            public object data;

            public Entry(Entry next, object data) {
                this.next = next;
                this.data = data;
            }
        }
    }
}
```

在命名空间Acme.Collections里声明了一个Stack类，所以这个类完整的名字就是Acme.Collections.Stack。它包含了几个成员：一个变量top，两个方法Push和Pop，还有一个嵌套类Entry。Entry类又包含了3个成员：一个next变量，一个data变量，一个构造函数。假设这个例子的源代码被保存在文件acme.cs里，那么命令

```
csc /t:library acme.cs
```

会把它编译成一个类库（没有Main入口点的代码）并生成一个汇编文件acme.dll。

汇编文件包含了中间语言（Intermediate Language, IL）指令形式的可执行代码，以及元数据（metadata）形式的链接信息。在执行之前，IL代码会被.NET 公共语言运行库的即时编译