

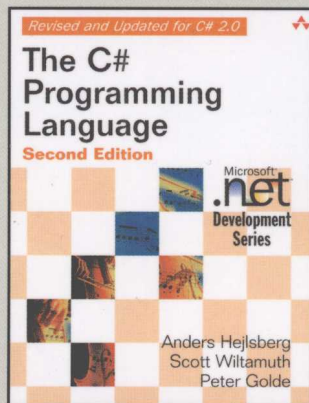


The C# Programming Language Second Edition

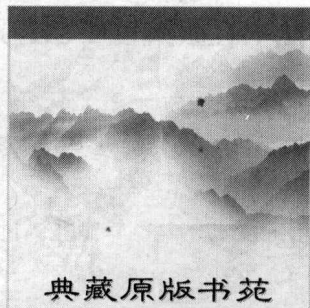
[美] Anders Hejlsberg Scott Wiltamuth Peter Golde 著

C# 编程语言

(第 2 版)(英文版)



C# 语言创始人权威著作



C#编程语言

(第2版)(英文版)

Anders Hejlsberg

[美] Scott Wiltamuth 著

Peter Golde

人民邮电出版社

北 京

图书在版编目 (CIP) 数据

C#编程语言: 第2版. 英文 / (美) 海吉斯博格 (Hejlsberg, A.) 等著.

—北京: 人民邮电出版社, 2007.8

(典藏原版书苑)

ISBN 978-7-115-16228-1

I. C... II. 海... III. C 语言—程序设计—英文 IV. TP312

中国版本图书馆 CIP 数据核字 (2007) 第 067561 号

版 权 声 明

Original edition, entitled C# Programming Language, 2nd Edition, 0321334434 by Hejlsberg Anders, Wiltamuth Scott, Golde Peter, published by Pearson Education, Inc, publishing as Addison Wesley Professional, Copyright © 2006 by Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

China edition published by PEARSON EDUCATION ASIA LTD., and POSTS & TELECOMMUNICATIONS PRESS Copyright © 2007.

This edition is manufactured in the People's Republic of China, and is authorized for sale only in People's Republic of China excluding Hong Kong, Macau and Taiwan.

仅限于中华人民共和国境内 (不包括中国香港、澳门特别行政区和中国台湾地区) 销售。

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。

典藏原版书苑

C#编程语言 (第2版) (英文版)

◆ 著 [美] Anders Hejlsberg Scott Wiltamuth Peter Golde

责任编辑 付 飞

◆ 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

新华书店总店北京发行所经销

◆ 开本: 800×1000 1/16

印张: 44.75

字数: 963 千字

印数: 1—3 000 册

2007 年 8 月第 1 版

2007 年 8 月北京第 1 次印刷

著作权合同登记号 图字: 01-2007-1479 号

ISBN 978-7-115-16228-1/TP

定价: 95.00 元

读者服务热线: (010) 67132705 印装质量热线: (010) 67129223

内容提要

本书以通俗易懂的语言、精辟丰富的实例，从对 C# 的简介开始，全面讲解了 C# 编程语言规范以及各个层面的特性，并且提供了 C# 设计小组的代码示例。本书第 2 版针对 C# 2.0 进行了全面升级，是 C# 权威参考书籍。本书第一部分以 C# 语言概述开篇，阐明 C# 语言的概念，接下来对随 Visual Studio .NET 2002 和 2003 发布的 C# 1.0 做了细致完整的技术讲解，内容包括 C# 的词法结构、类型、变量、表达式、语句、名字空间、例外、属性和不安全代码等主题。第二部分描述 C# 2.0 的众多特性，包括 Generics、匿名方法 (Anonymous Methods)、迭代子 (Iterators)、不完整类型 (Partial Type) 和空类型 (Nullable Type)。第 2 版在第 1 版的基础上作了大量的增补和改进，并附有参考资料列表和详尽的索引，能使读者更有效地检索正文并快速找到最感兴趣的主体。

本书作者均为微软 C# 开发团队的核心成员，第一作者更是被誉为编程界的神话，“跟 Anders 学 C#” 已成全世界读者的普遍共识。本书向读者提供权威、详尽的指导，是 C# 程序员必备的参考书。

前 言

C#项目是从7年多以前，也就是1998年的12月开始的，其目标是创造一种简单、现代、面向对象并且类型安全的新编程语言，也就是现在的.NET平台。从那时起，C#经历了漫长的历程，现在已经有数十万程序员在使用它，ECMA和ISO/IEC都为它建立了标准，第2版包括几种主要的新特性，其开发工作已经完成。

这是一本C#编程语言的详尽参考书。全书共分三部分：第一部分“C#1.0”，包括第1章~第18章，对随Visual Studio .NET 2002/2003发布的C# 1.0做了细致完整的技术讲解；第二部分“C#2.0”，包括第19章~第25章，描述C# 2.0的新增特性，包括Generics、匿名方法(Anonymous Methods)、迭代子(Iterators)、不完整类型(Partial Type)和空类型(Nullable Type)等；第三部分“附录”，描述了文档注释(documentation comments)，总结了C#2.0的词法和语法(lexical and syntactic grammars)等内容。

很多人参与了C#语言的创造。C# 1.0的设计团队成员有Anders Hejlsberg、Peter Golde、Peter Hallam、Shon Katzenberger、Todd Proebsting和Anson Horton。此外，C#中generic的设计与执行和.NET通用语言运行时(Common Language Runtime)基于微软研究院的Don Syme和Andrew Kennedy建立的“Gyro”原型。本书的第2版最终由Mads Torgersen编辑而成。

在这里无法向所有为C#设计做出贡献的人们表达谢意。虽然如此，我们还是要在这里感谢他们。没有一个好的设计能够凭空产生，我们从广大热情的开发人员那里得到了持续不断的反馈，这些信息是无价的。

C#一直并且还将是我们参与的最具有挑战性、最激动人心的项目之一。我们希望您享受使用C#的过程，就像我们创造它时一样。

Anders Hejlsberg

Scott Wiltamuth

2006年5月于西雅图



Contents

PART I C# 1.0 1

1 Introduction 3

- 1.1 Hello World 4
- 1.2 Program Structure 5
- 1.3 Types and Variables 7
- 1.4 Expressions 11
- 1.5 Statements 14
- 1.6 Classes and Objects 18
- 1.7 Structs 34
- 1.8 Arrays 35
- 1.9 Interfaces 37
- 1.10 Enums 39
- 1.11 Delegates 40
- 1.12 Attributes 42

2 Lexical Structure 45

- 2.1 Programs 45
- 2.2 Grammars 45
- 2.3 Lexical Analysis 47
- 2.4 Tokens 51
- 2.5 Preprocessing Directives 61

3	Basic Concepts	73
3.1	Application Startup	73
3.2	Application Termination	74
3.3	Declarations	75
3.4	Members	77
3.5	Member Access	79
3.6	Signatures and Overloading	86
3.7	Scopes	87
3.8	Namespace and Type Names	93
3.9	Automatic Memory Management	95
3.10	Execution Order	99
4	Types	101
4.1	Value Types	101
4.2	Reference Types	110
4.3	Boxing and Unboxing	112
5	Variables	115
5.1	Variable Categories	115
5.2	Default Values	119
5.3	Definite Assignment	119
5.4	Variable References	133
5.5	Atomicity of Variable References	133
6	Conversions	135
6.1	Implicit Conversions	135
6.2	Explicit Conversions	138
6.3	Standard Conversions	142
6.4	User-Defined Conversions	143
7	Expressions	147
7.1	Expression Classifications	147
7.2	Operators	149
7.3	Member Lookup	156
7.4	Function Members	157

7.5	Primary Expressions	170
7.6	Unary Operators	193
7.7	Arithmetic Operators	198
7.8	Shift Operators	207
7.9	Relational and Type-Testing Operators	209
7.10	Logical Operators	216
7.11	Conditional Logical Operators	218
7.12	Conditional Operator	220
7.13	Assignment Operators	221
7.14	Expression	226
7.15	Constant Expressions	226
7.16	Boolean Expressions	228
8	Statements	229
8.1	End Points and Reachability	230
8.2	Blocks	232
8.3	The Empty Statement	233
8.4	Labeled Statements	233
8.5	Declaration Statements	234
8.6	Expression Statements	236
8.7	Selection Statements	237
8.8	Iteration Statements	243
8.9	Jump Statements	248
8.10	The try Statement	255
8.11	The checked and unchecked Statements	258
8.12	The lock Statement	259
8.13	The using Statement	260
9	Namespaces	263
9.1	Compilation Units	263
9.2	Namespace Declarations	264
9.3	Using Directives	265
9.4	Namespace Members	271
9.5	Type Declarations	271

10 Classes 273

- 10.1 Class Declarations 273
- 10.2 Class Members 277
- 10.3 Constants 287
- 10.4 Fields 290
- 10.5 Methods 299
- 10.6 Properties 317
- 10.7 Events 327
- 10.8 Indexers 333
- 10.9 Operators 338
- 10.10 Instance Constructors 343
- 10.11 Static Constructors 349
- 10.12 Destructors 352

11 Structs 355

- 11.1 Struct Declarations 355
- 11.2 Struct Members 356
- 11.3 Class and Struct Differences 357
- 11.4 Struct Examples 363

12 Arrays 367

- 12.1 Array Types 367
- 12.2 Array Creation 369
- 12.3 Array Element Access 369
- 12.4 Array Members 369
- 12.5 Array Covariance 369
- 12.6 Array Initializers 370

13 Interfaces 373

- 13.1 Interface Declarations 373
- 13.2 Interface Members 375
- 13.3 Fully Qualified Interface Member Names 380
- 13.4 Interface Implementations 380

14 Enums 393

- 14.1 Enum Declarations 393
- 14.2 Enum Modifiers 394
- 14.3 Enum Members 394
- 14.4 The System.Enum Type 397
- 14.5 Enum Values and Operations 397

15 Delegates 399

- 15.1 Delegate Declarations 399
- 15.2 Delegate Instantiation 402
- 15.3 Delegate Invocation 403

16 Exceptions 407

- 16.1 Causes of Exceptions 407
- 16.2 The System.Exception Class 408
- 16.3 How Exceptions Are Handled 408
- 16.4 Common Exception Classes 409

17 Attributes 411

- 17.1 Attribute Classes 411
- 17.2 Attribute Specification 414
- 17.3 Attribute Instances 420
- 17.4 Reserved Attributes 422
- 17.5 Attributes for Interoperation 428

18 Unsafe Code 429

- 18.1 Unsafe Contexts 429
- 18.2 Pointer Types 433
- 18.3 Fixed and Moveable Variables 436
- 18.4 Pointer Conversions 437
- 18.5 Pointers in Expressions 438
- 18.6 The fixed Statement 446
- 18.7 Stack Allocation 450
- 18.8 Dynamic Memory Allocation 451

PART II C# 2.0 455

19 Introduction to C# 2.0 457

- 19.1 Generics 458
- 19.2 Anonymous Methods 463
- 19.3 Iterators 467
- 19.4 Partial Types 471
- 19.5 Nullable Types 472

20 Generics 477

- 20.1 Generic Class Declarations 477
- 20.2 Generic Struct Declarations 488
- 20.3 Generic Interface Declarations 488
- 20.4 Generic Delegate Declarations 490
- 20.5 Constructed Types 491
- 20.6 Generic Methods 498
- 20.7 Constraints 506
- 20.8 Expressions and Statements 517
- 20.9 Revised Lookup Rules 521
- 20.10 Right-Shift Grammar Changes 533

21 Anonymous Methods 535

- 21.1 Anonymous Method Expressions 535
- 21.2 Anonymous Method Signatures 535
- 21.3 Anonymous Method Conversions 536
- 21.4 Anonymous Method Blocks 537
- 21.5 Outer Variables 538
- 21.6 Anonymous Method Evaluation 541
- 21.7 Delegate Instance Equality 542
- 21.8 Definite Assignment 543
- 21.9 Method Group Conversions 544
- 21.10 Delegate Creation Expressions 546
- 21.11 Implementation Example 546

22 Iterators 551

- 22.1 Iterator Blocks 551
- 22.2 Enumerator Objects 552
- 22.3 Enumerable Objects 555
- 22.4 The yield Statement 556
- 22.5 Implementation Example 558

23 Partial Types 567

- 23.1 Partial Declarations 567
- 23.2 Name Binding 571

24 Nullable Types 573

- 24.1 Nullable Types 573
- 24.2 Conversions 574
- 24.3 Expressions 580

25 Other Features 587

- 25.1 Property Accessor Accessibility 587
- 25.2 Static Classes 590
- 25.3 Namespace Alias Qualifiers 592
- 25.4 Extern Aliases 596
- 25.5 Pragma Directives 600
- 25.6 Default Value Expression 601
- 25.7 Conditional Attribute Classes 602
- 25.8 Fixed Size Buffers 603

PART III Appendixes 609**A Documentation Comments 611**

- A.1 Introduction 611
- A.2 Recommended Tags 613
- A.3 Processing the Documentation File 623
- A.4 An Example 629

B Grammar 635

B.1 Lexical Grammar 635

B.2 Syntactic Grammar 644

B.3 Grammar Extensions for Unsafe Code 671

Index 675



Part I

C# 1.0

1. Introduction

C# (pronounced “See Sharp”) is a simple, modern, object-oriented, and type-safe programming language. C# has its roots in the C family of languages and will be immediately familiar to C, C++, and Java programmers. C# is standardized by ECMA International as the *ECMA-334* standard and by ISO/IEC as the *ISO/IEC 23270* standard. Microsoft’s C# compiler for the .NET Framework is a conforming implementation of both of these standards.

C# is an object-oriented language, but C# further includes support for *component-oriented* programming. Contemporary software design increasingly relies on software components in the form of self-contained and self-describing packages of functionality. Key to such components is that they present a programming model with properties, methods, and events; they have attributes that provide declarative information about the component; and they incorporate their own documentation. C# provides language constructs to directly support these concepts, making C# a very natural language in which to create and use software components.

Several C# features aid in the construction of robust and durable applications: *Garbage collection* automatically reclaims memory occupied by unused objects; *exception handling* provides a structured and extensible approach to error detection and recovery; and the *type-safe* design of the language makes it impossible to have uninitialized variables, to index arrays beyond their bounds, or to perform unchecked type casts.

C# has a *unified type system*. All C# types, including primitive types such as `int` and `double`, inherit from a single root object type. Thus, all types share a set of common operations, and values of any type can be stored, transported, and operated upon in a consistent manner. Furthermore, C# supports both user-defined reference types and value types, allowing dynamic allocation of objects as well as in-line storage of lightweight structures.

To ensure that C# programs and libraries can evolve over time in a compatible manner, much emphasis has been placed on *versioning* in C#’s design. Many programming languages pay little attention to this issue, and, as a result, programs written in those languages break more often than necessary when newer versions of dependent libraries are introduced. Aspects of C#’s design that were directly influenced by versioning considerations include the separate `virtual` and `override` modifiers, the rules for method overload resolution, and support for explicit interface member declarations.

The rest of this chapter describes the essential features of the C# language. Although later chapters describe rules and exceptions in a detail-oriented and sometimes mathematical manner, this chapter strives for clarity and brevity at the expense of completeness. The intent is to provide the reader with an introduction to the language that will facilitate the writing of early programs and the reading of later chapters.

1.1 Hello World

The “Hello, World” program is traditionally used to introduce a programming language. Here it is in C#:

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

C# source files typically have the file extension `.cs`. Assuming that the “Hello, World” program is stored in the file `hello.cs`, the program can be compiled with the Microsoft C# compiler using the command line

```
csc hello.cs
```

which produces an executable assembly named `hello.exe`. The output produced by this application when it is run is

```
Hello, World
```

The “Hello, World” program starts with a `using` directive that references the `System` namespace. Namespaces provide a hierarchical means of organizing C# programs and libraries. Namespaces contain types and other namespaces—for example, the `System` namespace contains a number of types, such as the `Console` class referenced in the program, and a number of other namespaces, such as `IO` and `Collections`. A `using` directive that references a given namespace enables unqualified use of the types that are members of that namespace. Because of the `using` directive, the program can use `Console.WriteLine` as shorthand for `System.Console.WriteLine`.

The `Hello` class declared by the “Hello, World” program has a single member, the method named `Main`. The `Main` method is declared with the `static` modifier. Unlike instance methods, which reference a particular object instance using the keyword `this`, static methods operate without reference to a particular object. By convention, a static method named `Main` serves as the entry point of a program.