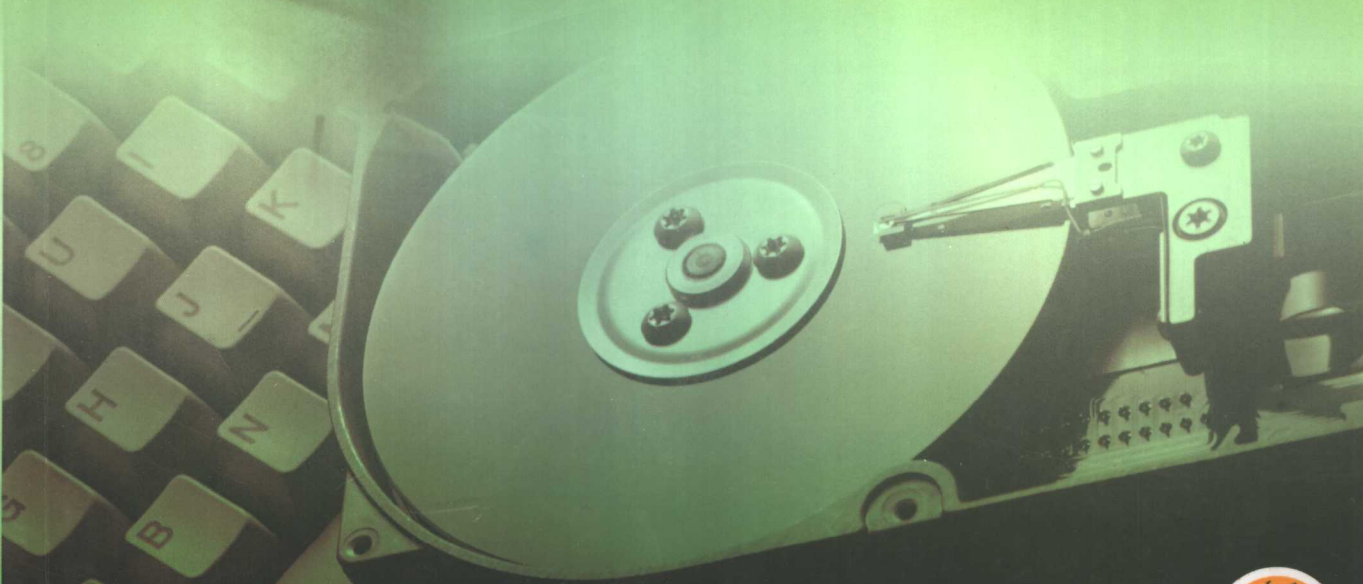


pliant

millennium

C# 编程思想

嘉木工作室 编著



 机械工业出版社
CHINA MACHINE PRESS



C#编程思想

嘉木工作室 编著



机械工业出版社

该 C# 开发语言是包含在 Visual Studio.NET 框架中的最新应用程序开发语言。目前, 随着网络应用程序的广泛开发与应用需求, C# 已经成为开发基于计算和通信的最流行的语言。本书从基础入门, 结合 Web 开发的特点, 详细介绍 C# 开发语言的语法, 并针对这些语法提供了丰富的例程, 以充分发挥 C# 语言的开发优势。

本书适合从事网络程序设计的程序员、大中专院校相关专业师生和培训学校学生。

图书在版编目 (CIP) 数据

C# 编程思想/嘉木工作室编著. —北京: 机械工业出版社, 2003.3
ISBN 7-111-11860-X

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

中国版本图书馆 CIP 数据核字 (2003) 第 025283 号

机械工业出版社 (北京市百万庄大街 22 号 邮政编码 100037)

策 划: 边 萌 责任编辑: 边 萌

封面设计: 张 静 责任印制: 闫 焱

北京交通印务实业公司印刷·新华书店北京发行所发行

2003 年 5 月第 1 版第 1 次印刷

787mm×1092mm×1/16·22 印张·554 千字

0 001—4000 册

定价: 36.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

本社购书热线电话 (010) 68993821、88379646

封面无防伪标均为盗版

前 言

在过去的二十多年内，C 和 C++已经成为在软件开发中应用最广泛的开发语言。但是 C 和 C++的灵活性牺牲了它们的开发效率，如果和其他开发语言相比，相同功能的 C/C++软件通常需要更长的开发周期。正是由于 C/C++开发的复杂性和需要较长的开发周期，许多 C/C++开发人员都在寻找一种可以在功能和开发效率间提供更多平衡的开发语言。作为快速创建和集成 XML Web 服务和应用程序的单一综合工具，Visual Studio .NET 在改善操作的同时极大地提高了开发效率。其中的 Visual C# .NET（C#的发音为 C Sharp）是 Microsoft 推出的新型语言，它可提高 C 和 C++ 开发人员的效率。Visual C# .NET 对具有属性、方法、索引器、特性、版本和事件的组件提供一流的支持，并同时为 .NET 平台提供强有力的和有效的支持。

C#是一种先进的、面向对象的语言，通过 C#可以让开发人员快速建立大范围的基于 MS 网络平台的应用，并且提供大量的开发工具和服务，帮助开发人员开发基于计算和通信的各种应用。由于 C#是一种面向对象的开发语言，所以将 C#可以大范围的适用于高层商业应用和底层系统的开发。即使是通过简单的 C#构造也可以将各种组件方便地转变为基于 Web 的应用，并且能够通过 Internet 被各种系统或其他开发语言所开发的应用调用。

即使抛开上面所提到的优点，C#也可以为 C/C++开发人员提供快速的开发手段而不需要牺牲任何 C/C++语言的特点/优点。从继承角度来看，C#在更高层次上重新实现了 C/C++，熟悉 C/C++开发的人员可以很快的转变为 C#开发人员。C#的优点包括：

- 高的开发效率与安全性；
- 与 Web 开发相结合；
- 减小开发中的错误；
- 提供内置的版本支持来减少开发费用；
- 功能强，易于表现，灵活；
- 更好的结合商业应用中的流程与软件实现；
- 可扩展的协作能力；
- 允许有限制的使用指针。

总之，使用 C#能够利用方便快捷的 Microsoft 网络平台建立各种应用和建立能够在网络间相互调用的 Web 服务。从开发语言的角度来讲，C#可以更好地帮助开发人员避免错误，提高工作效率，并具有 C/C++的强大功能。

该书结构合理、内容翔实、循序渐进，适合各个层次的读者学习参考。由于时间仓促，本书编著过程中难免存在错误和纰漏，恳请广大读者批评指正。

编 者

目 录

前言

第 1 章 C#简介.....	1
1.1 开始 (Start)	1
1.2 类型 (Types)	2
1.2.1 预定义类型 (Predefined type)	4
1.2.2 转换 (Conversion)	6
1.2.3 数组类型 (Array)	7
1.2.4 类型系统的统一(Type System Unification)	9
1.3 变量与参数 (Variables And Parameters)	10
1.4 自动内存管理 (Automatic Memory Management)	14
1.5 表达式 (Expressions)	17
1.6 语句 (Statements)	17
1.7 类 (Class)	20
1.7.1 常量 (Constants)	22
1.7.2 域 (Fields)	23
1.7.3 方法 (Methods)	24
1.7.4 属性 (Properties)	26
1.7.5 事件 (Event)	26
1.7.6 操作符 (Operators)	28
1.7.7 索引 (Index)	29
1.7.8 实例构造函数 (Instance Constructors)	31
1.7.9 析构函数 (Destructors)	32
1.7.10 静态构造函数 (Static Constructors)	32
1.7.11 继承 (Inheritance)	33
1.8 结构 (Structs)	34
1.9 接口 (Interfaces)	35
1.10 委托 (Delegates)	37
1.11 枚举 (Enums)	38
1.12 名字空间与汇编 (Namespaces And Assemblies)	39
1.13 版本 (Versioning)	41
1.14 属性 (Attributes)	44
第 2 章 词法结构	46
2.1 翻译阶段 (Phases of Translation)	46
2.2 语法符号 (Grammar Notation)	46
2.3 词法分析 (Lexical Analysis)	47

2.3.1 输入 (Input)	47
2.3.2 字符输入 (Input Characters)	47
2.3.3 线终端函数 (Line Terminators)	48
2.3.4 注释 (Comments)	48
2.3.5 空格 (White Space)	49
2.4 标识符 (Tokens)	49
2.4.1 单码字符转义序列 (Unicode Character Escape Sequences)	50
2.4.2 标识符 (Identifiers)	51
2.4.3 关键字 (Keywords)	52
2.4.4 字母 (Literals)	53
2.4.5 操作符与标点符号 (Operators And Punctuators)	58
2.5 预处理指令 (Pre-processing Directive)	58
2.5.1 预处理标识符 (Pre-processing Identifiers)	59
2.5.2 预处理表达式 (Preprocessing expressions)	59
2.5.3 预处理声明 (Pre-processing Declarations)	60
2.5.4 #if, #elif, #else, #endif	61
2.5.5 #error 和 #warning	64
2.5.6 #region 和 #endregion	65
2.5.7 #line	65
第3章 基本概念	66
3.1 程序开始 (Program Startup)	66
3.2 程序结束 (Program Termination)	66
3.3 声明 (Declarations)	67
3.4 元素 (Members)	69
3.4.1 名字空间元素 (Namespace Members)	69
3.4.2 结构成员 (Struct Members)	70
3.4.3 枚举成员 (Enumeration Members)	70
3.4.4 类项 (Class Member)	70
3.4.5 接口成员 (Interface Members)	71
3.4.6 数组项 (Array Members)	71
3.4.7 委托成员 (Delegate Members)	71
3.5 成员访问 (Member Access)	71
3.5.1 声明的可访问性 (Declared Accessibility)	71
3.5.2 可访问域 (Accessibility Domains)	72
3.5.3 访问保护 (Protected Access)	74
3.5.4 访问约束 (Accessibility Constraints)	75
3.6 签名和重载 (Signatures And Overloading)	76
3.7 范围 (Scopes)	77
3.7.1 名字的范围 (Name Scopes)	77

3.7.2 名字隐藏 (Name Hiding)	79
3.8 名字空间和类型名字 (Namespace And Type Names)	82
第4章 类型	84
4.1 值类型 (Value Types)	84
4.1.1 默认构造函数 (Default Constructors)	85
4.1.2 结构类型 (Struct Types)	86
4.1.3 简单类型 (Simple Types)	86
4.1.4 整类型 (Integral Types)	87
4.1.5 浮点数类型 (Floating Point Types)	88
4.1.6 十进制类型 (The Decimal Type)	89
4.1.7 布尔类型 (The Bool Type)	90
4.1.8 枚举类型 (Enumeration Types)	90
4.2 引用类型 (Reference Types)	90
4.2.1 类类型 (Class Types)	91
4.2.2 对象类型 (The Object Type)	91
4.2.3 字符串类型 (The String Type)	91
4.2.4 接口类型 (Interface Types)	92
4.2.5 数组类型 (Array Type)	92
4.2.6 委托类型 (Delegate Type)	92
4.3 封箱和非封箱 (Boxing and Unboxing)	92
4.3.1 封箱转换 (Boxing Conversions)	92
4.3.2 非封箱转换 (Unboxing Conversions)	94
第5章 变量	95
5.1 变量分类 (Variable Categories)	95
5.1.1 静态变量 (Static Variables)	95
5.1.2 实例变量 (Instance Variables)	96
5.1.3 数组成员 (Array Elements)	96
5.1.4 值参数 (Value Parameters)	96
5.1.5 引用参数 (Reference Parameters)	96
5.1.6 输出参数 (Output Parameters)	97
5.1.7 局部变量 (Local Variables)	97
5.2 默认值 (Default Values)	97
5.3 明确赋值 (Definite Assignment)	98
5.3.1 初始赋值变量 (Initially Assigned Variables)	100
5.3.2 初始未赋值变量 (Initially Unassigned Variables)	100
5.4 变量引用 (Variable References)	100
第6章 转换	101

6.1 隐式转换 (Implicit Conversions)	101
6.1.1 一致性转换 (Identity Conversion)	101
6.1.2 隐式数值转换 (Implicit Numeric Conversions)	101
6.1.3 隐式枚举转换 (Implicit Enumeration Conversions)	102
6.1.4 隐式参照转换 (Implicit Reference Conversions)	102
6.1.5 封箱转换 (Boxing Conversions)	102
6.1.6 隐式常量表达式转换 (Implicit Constant Expression Conversions)	102
6.1.7 用户自定义隐式转换 (User-defined Implicit Conversions)	103
6.2 显式转换 (Explicit Conversions)	103
6.2.1 显式数值转换 (Explicit Numeric Conversions)	103
6.2.2 显式枚举转换 (Explicit Enum Conversions)	104
6.2.3 显式引用转换 (Explicit Reference Conversions)	105
6.2.4 非封箱转换 (Unboxing Conversions)	105
6.2.5 用户自定义显式转换 (User-Defined Explicit Conversions)	105
6.3 标准转换 (Standard Conversions)	106
6.3.1 标准隐式转换 (Standard Implicit Conversions)	106
6.3.2 标准显式转换 (Standard Explicit Conversions)	106
6.4 用户自定义转换 (User-Defined Conversions)	106
6.4.1 被允许的用户自定义转换 (Permitted User-Defined Conversions)	106
6.4.2 用户自定义转换求值 (Evaluation of User-Defined Conversions)	106
6.4.3 用户自定义隐式转换 (User-Defined Implicit Conversions)	107
6.4.4 用户自定义显式转换 (User-Defined Explicit Conversions)	108
第7章 表达式	109
7.1 表达式分类 (Expression Classifications)	109
7.1.1 分类 (Classifications)	109
7.1.2 表达式的值 (Values of Expressions)	110
7.2 操作符 (Operators)	110
7.2.1 操作符优先与结合性 (Operator Precedence And Associativity)	110
7.2.2 操作符重载 (Operator Overloading)	111
7.2.3 一元操作符重载分解 (Unary Operator Overload Resolution)	112
7.2.4 二进制操作符重载分解 (Binary Operator Overload Resolution)	112
7.2.5 候选用户自定义操作符 (Candidate User-Defined Operators)	113
7.2.6 数提升 (Numeric Promotions)	113
7.3 成员查找 (Member Lookup)	114
7.4 函数成员 (Function Members)	115
7.4.1 自变量列表 (Argument Lists)	116
7.4.2 重载分解 (Overload Resolution)	119
7.4.3 函数成员引用 (Function Member Invocation)	121
7.5 原始表达式 (Primary Expressions)	122

7.5.1	字母 (Literals)	122
7.5.2	简化名 (Simple Names)	122
7.5.3	括弧表达式 (Parenthesized Expressions)	124
7.5.4	成员访问 (Member Access)	124
7.5.5	引用表达式 (Invocation Expressions)	126
7.5.6	成员访问 (Element Access)	128
7.5.7	This 访问 (This Access)	130
7.5.8	基本访问 (Base Access)	130
7.5.9	后缀增量和减量操作符 (Postfix Increment And Decrement Operators)	131
7.5.10	New 操作符 (New Operator)	132
7.5.11	Typeof 操作符 (Type of Operator)	136
7.5.12	检查的和未检查操作符 (Checked And Unchecked Operators)	137
7.6	一元表达式 (Unary Expression)	139
7.6.1	一元加操作符 (Unary Plus Operator)	140
7.6.2	一元减操作符 (Unary Minus Operator)	140
7.6.3	逻辑非操作符 (Logical Negation Operator)	141
7.6.4	按位求补码操作符 (Bitwise Complement Operator)	141
7.6.5	前缀增量和减量操作符 (Prefix Increment And Decrement Operators)	142
7.6.6	CAST 表达式 (Cast Expressions)	142
7.7	算术运算符 (Arithmetic Operators)	143
7.7.1	乘法运算操作符 (Multiplication Operator)	143
7.7.2	除法运算操作符 (Division operator)	144
7.7.3	求余数操作符 (Remainder Operator)	145
7.7.4	加法操作符 (Addition Operator)	146
7.7.5	减法操作符 (Subtraction Operator)	148
7.8	转换操作符 (Shift Operators)	149
7.9	关系操作符 (Relational Operators)	150
7.9.1	整数比较操作符 (Integer Comparison Operators)	151
7.9.2	浮点数比较操作符 (Floating-Point Comparison Operators)	152
7.9.3	十进制比较操作符 (Decimal Comparison Operators)	153
7.9.4	布尔等操作符 (Boolean Equality Operators)	153
7.9.5	枚举比较操作符 (Enumeration Comparison Operators)	153
7.9.6	引用类型相等操作符 (Reference type Equality Operators)	154
7.9.7	字符串相等操作符 (String Equality Operators)	155
7.9.8	委托相等操作符 (Delegate Equality Operators)	156
7.9.9	is 操作符 (The is Operator)	156
7.9.10	as 操作符 (The as Operator)	157
7.10	逻辑操作符 (Logical Operators)	157
7.10.1	整数逻辑操作符 (Integer Logical Operators)	158

7.10.2	枚举逻辑操作符 (Enumeration Logical Operators)	158
7.10.3	布尔逻辑操作符 (Boolean Logical Operators)	159
7.11	附加条件逻辑操作符 (Conditional Logical Operators)	159
7.11.1	布尔条件逻辑操作符 (Boolean Conditional Logical Operators)	159
7.11.2	自定义条件逻辑操作符 (User-Defined Conditional Logical Operators)	160
7.12	条件操作符 (Conditional Operator)	160
7.13	赋值操作符 (Assignment Operators)	161
7.13.1	简单赋值 (Simple Assignment)	161
7.13.2	复合赋值 (Compound Assignment)	164
7.14	表达式 (Expression)	165
7.15	常量表达式 (Constant Expressions)	165
7.16	布尔表达式 (Boolean Expressions)	166
第 8 章	语句	167
8.1	结束点和可到达性 (End Points And Reachability)	167
8.2	块 (Blocks)	169
8.3	空语句 (The Empty Statement)	170
8.4	标号语句 (Labeled Statements)	170
8.5	声明语句 (Declaration Statements)	171
8.5.1	局部变量声明 (Local Variable Declarations)	171
8.5.2	局部常量声明 (Local Constant Declarations)	172
8.6	表达式语句 (Expression Statements)	173
8.7	选择语句 (Selection Statements)	173
8.7.1	if 语句 (The If Statement)	173
8.7.2	swith 语句 (The Switch Statement)	174
8.8	iteration 语句 (Iteration Statements)	178
8.8.1	while 语句 (The While Statement)	178
8.8.2	do 语句 (The Do Statement)	179
8.8.3	for 语句 (The For Statement)	179
8.8.4	for each 语句 (The For Each Statement)	180
8.9	jump 语句 (Jump Statements)	182
8.9.1	break 语句 (The Break Statement)	183
8.9.2	continue 语句 (The Continue Statement)	183
8.9.3	goto 语句 (The Goto Statement)	183
8.9.4	return 语句 (The Return Statement)	184
8.9.5	throw 语句 (The Throw Statement)	185
8.10	try 语句 (The Try Statement)	186
8.11	hecked 和 unchecked 语句 (The Checked And Unchecked Statements)	189
8.12	lock 语句 (The Lock Statement)	189
8.13	using 语句 (The Using Statement)	190

第 9 章 名字空间	192
9.1 编译单元 (Compilation Units)	192
9.2 名字空间声明 (Namespace Declarations)	192
9.3 使用指令 (Using Directives)	194
9.3.1 别名指令使用 (Using Alias Directives)	194
9.3.2 名字空间指令使用 (Using Namespace Directives)	197
9.4 名字空间成员 (Namespace Members)	199
9.5 类型声明 (Type Declarations)	200
第 10 章 类	201
10.1 类声明 (Class Declarations)	201
10.1.1 类修改函数 (Class Modifiers)	201
10.1.2 类基本说明 (Class Base Specification)	202
10.1.3 类主体 (Class Body)	204
10.2 类成员 (Class Member)	204
10.2.1 继承 (Inheritance)	205
10.2.2 new 修改函数 (The New Modifier)	206
10.2.3 访问修改函数 (Access Modifiers)	206
10.2.4 constituent 类型 (Constituent Types)	206
10.2.5 静态和实例成员 (Static And Instance Members)	206
10.3 常量 (Constants)	207
10.4 域 (Fields)	209
10.4.1 静态和实例域 (Static And Instance Fields)	210
10.4.2 readonly 域 (Readonly Fields)	211
10.4.3 域初始化 (Field Initialization)	212
10.4.4 变量初始化 (Variable Initializers)	213
10.5 方法 (Methods)	214
10.5.1 方法参数 (Method Parameters)	216
10.5.2 静态和实例方法 (Static And Instance Methods)	222
10.5.3 虚拟方法 (Virtual Methods)	222
10.5.4 重载方法 (Override Methods)	224
10.5.5 封装方法 (Sealed Methods)	226
10.5.6 抽象方法 (Abstract Methods)	227
10.5.7 外部方法 (External Methods)	228
10.5.8 方法主体 (Method Body)	229
10.5.9 方法重载 (Method Overloading)	230
10.6 属性 (Properties)	230
10.6.1 静态属性 (Static Properties)	231
10.6.2 访问函数 (Accessors)	231

10.6.3 虚拟、封装、重载和抽象访问函数 (Virtual, Sealed, Override And Abstract Accessors)	237
10.7 事件 (Events)	238
10.7.1 事件访问函数 (Event Accessors)	241
10.7.2 静态事件 (Static Events)	243
10.8 索引 (Indexers)	243
10.9 操作符 (Operators)	246
10.9.1 一元操作符 (Unary Operators)	247
10.9.2 二元操作符 (Binary Operators)	248
10.9.3 转换操作符 (Conversion Operators)	248
10.10 实例构造函数 (Instance Constructors)	249
10.10.1 构造函数初始化 (Constructor Initializers)	250
10.10.2 实例变量初始化函数 (Instance Variable Initializers)	251
10.10.3 构造函数执行 (Constructor Execution)	251
10.10.4 默认构造函数 (Default Constructors)	253
10.10.5 局部构造函数 (Private Constructors)	254
10.10.6 可选的构造函数参数 (Optional Constructor Parameters)	254
10.11 静态构造函数 (Static Constructors)	255
10.12 析构函数 (Destructors)	258
第 11 章 结构	259
11.1 结构声明 (Struct Declarations)	259
11.1.1 结构修改函数 (Struct Modifiers)	259
11.1.2 结构接口 (Struct Interfaces)	260
11.1.3 结构主体 (Struct Body)	260
11.2 结构成员 (Struct Members)	260
11.3 类和结构差异 (Class And Struct Differences)	260
11.3.1 值语义 (Value Semantics)	260
11.3.2 继承 (Inheritance)	261
11.3.3 赋值 (Assignment)	261
11.3.4 默认值 (Default Values)	262
11.3.5 封箱和非封箱 (Boxing And Unboxing)	262
11.3.6 this 的意思 (Meaning Of This)	263
11.3.7 域初始化 (Field Initializers)	263
11.3.8 构造函数 (Constructors)	263
11.3.9 析构函数 (Destructors)	263
11.4 结构实例 (Struct Examples)	263
11.4.1 数据库整数类型 (Database Integer Type)	263
11.4.2 布尔型数据库类型 (Database Boolean Type)	265
第 12 章 数组	268

12.1	数组类型 (Array Types)	268
12.2	数组建立 (Array Creation)	269
12.3	数组成员访问 (Array Element Access)	269
12.4	数组成员 (Array Members)	269
12.5	数组方差 (Array Covariance)	269
12.6	数组初始化函数 (Array Initializers)	270
第 13 章 接口		272
13.1	接口声明 (Interface declarations)	272
13.1.1	接口修改函数 (Interface Modifiers)	272
13.1.2	基本接口 (Base Interfaces)	273
13.1.3	接口主体 (Interface Body)	273
13.2	接口成员 (Interface Members)	274
13.2.1	接口方法 (Interface Methods)	275
13.2.3	接口事件 (Interface Events)	275
13.2.4	接口索引 (Interface Indexers)	275
13.2.5	接口成员访问 (Interface Member Access)	276
13.3	全权接口成员名字 (Fully Qualified Interface Member Names)	278
13.4	接口执行 (Interface Implementations)	278
13.4.1	显式接口成员执行 (Explicit Interface Member Implementations)	279
13.4.2	接口映射 (Interface Mapping)	281
13.4.3	接口执行继承 (Interface Implementation Inheritance)	284
13.4.4	接口再执行 (Interface Re-Implementation)	286
13.4.5	抽象类和接口 (Abstract Classes And Interfaces)	288
第 14 章 枚举		289
14.1	枚举声明 (Enum Declarations)	289
14.2	枚举修改函数 (Enum Modifiers)	290
14.3	枚举成员 (Enum Members)	290
14.4	Enum 值和操作 (Enum Values And Operations)	292
第 15 章 委托		293
15.1	委托声明 (Delegate Declarations)	293
15.2	委托实例 (Delegate Instantiation)	294
15.3	多 cast 委托 (Multi-Cast Delegates)	294
15.4	委托引用 (Delegate Invocation)	294
第 16 章 异常		295
16.1	异常原因 (Causes Of Exceptions)	295
16.2	System.Exception 类 (The System.Exception Class)	295

16.3 怎样处理异常 (How Exceptions Are Handled)	295
16.4 常用的异常类 (Common Exception Classes)	296
第 17 章 属性	297
17.1 类属性 (Attribute Classes)	297
17.1.1 AttributeUsage 属性 (The Attributeusage Attribute)	297
17.1.2 位置和命名参数 (Positional And Named Parameters)	298
17.1.3 属性参数类型 (Attribute Parameter Types)	299
17.2 属性说明 (Attribute Specification)	299
17.3 属性实例 (Attribute Instances)	302
17.3.1 属性的编译 (Compilation Of An Attribute)	302
17.3.2 属性实例的运行期恢复 (Run-time Retrieval Of An Attribute Instance)	303
17.4 保留属性 (Reserved Attribute)	303
17.4.1 AttributeUsage 属性 (The AttributeUsage Attribute)	303
17.4.2 Conditional 属性 (The Conditional Attribute)	304
17.4.3 Obsolete 属性 (The Obsolete Attribute)	307
附录 A 不安全代码	308
附录 B 互操作性	323
参考文献	336

第1章 C# 简介

C#是在 C 和 C++基础上发展起来的一门简单、现代、对象定位、类型安全的程序语言。C#（读作“C sharp”）植根于 C 和 C++语言家族，且与 C 和 C++程序非常接近，C#旨在把 System 的多产性与 C++尚未完善的功能结合起来。

C#是 Microsoft Visual Studio.NET 的一部分。除 C#外，Visual Studio 还支持 Visual Basic、Visual C++，以及撰稿语言 VBScript 和 Jscript。所有这些语言都可访问 Microsoft.NET 平台，后者包括一个普通的执行工具和一个丰富的类库。Microsoft.NET 定义一个普通语言的子集（CLS）及一种混合语言，这种混合语言确保 CLS 所属的语言和类库之间能够通用地操作，且不留下任何痕迹。虽然 C#是一门新的语言，但对于 C#开发者来说，它完全可以访问正时兴的工具如 Visual Basic 和 Visual C++所用的丰富的类库。C#本身没有类库。

本章以下几部分将对这门语言的基本特征加以介绍。以后的几章则对其规则及异常现象进行具体的说明，有时也涉及其运算方式。本章的虽然不十分详细，但却简短清晰。其目的是为了向读者简要介绍这门语言，以便读者能够编写简单的程序及对以后几章的阅读。

1.1 开始 (Start)

典型的“hello,world”程序的编写方法如下：

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

C#程序的源代码一般储存在一或多个带有扩展名.CS 的文本文件如 hello.cs 中。使用 Visual Studio 提供的命令行编译函数，这个程序编译时可带有命令行指令：

```
CSC    hello.cs
```

该指令产生一个 hello.exe 的可执行程序。这个程序输出：

```
Hello,world
```

此程序严格的检测过程如下：

- 指令 Using System 引用一个由 Microsoft.NET Frame Work 类库提供的名字空间 System。这个名字空间含有方法 Main 中的类 Console。名字空间提供类库中不同元素的等级含义。指令“using”使得作为名字空间成员的类型非权名使用成为可

能。在程序“hello,world”中使用的 `Console.WriteLine` 就是 `System.Console.WriteLine` 的编写形式。

- 方法 `Main` 是类 `hello` 的成员，它具有 `static` 修改函数。因此，它是类 `hello` 而不是该类实例的一个方法。
- 程序的主要入口——即被调用开始执行程序的方法——总是静态方法 `Main`。
- “hello,world”通过使用一个类库进行输出。这种语言本身没有类库，而是使用一个也被其他语言如 `Visual Basic` 和 `Visual C++` 使用的普通的类库。C 和 C++ 的开发者对于那些在程序“hello,world”中出现的语句很感兴趣。
- 该程序不使用 `Main` 的总方法。总体水平不支持方法和变量；这样的元素总是包含在类型声明过程中（如类和结构的声明过程）。
- 该程序既不使用操作符“`::`”，也不使用“`->`”。“`::`”实际上不是一个操作符，而操作符“`->`”只在一小部分程序中使用。分隔符“`.`”被用在复合名字如：`Console.WriteLine` 中。
- 该程序没有前声明。也没有必要使用前声明，因为声明顺序无关紧要。
- 该程序不用 `#include` 引进程序文本。程序的从属关系是根据符号而不是根据文本确定的。此系统消除了用不同语言所编写的程序之间的障碍。例如，类 `console` 也可用其他语言编写。

1.2 类型 (Types)

C#支持两种类型：Value types 和 reference types。值类型包括简单类型（如 `char`，`int` 和 `float`）、枚举类型和结构类型。引用类型包括类类型、接口类型、委托类型和数组类型。

值类型与引用类型的不同点在于，值类型的变量直接包括它们的数据，而引用类型的变量则把引用储存到对象中。引用类型的两个变量可以引用同一个对象。这样，对一个变量的操作就可能影响另一个变量所引用的对象。值类型的每一个变量都具有它们自己的数据拷贝，因此对一个变量的操作不可能影响另一个变量。下面的例子就表明了这种差异。

```
using System;
class Class1
{
    public int Value = 0;
}
class Test
{
    static void Main() {
        int val1 = 0;
        int val2 = val1;
        val2 = 123;
        Class1 ref1 = new Class1();
        Class1 ref2 = ref1;
```



```
        ref2.Value = 123;
        Console.WriteLine("Values: {0}, {1}", val1, val2);
        Console.WriteLine("Refs: {0}, {1}", ref1.Value, ref2.Value);
    }
}
```

该程序输出:

```
Values: 0, 123
Refs: 123, 123
```

对局部变量 `val1` 的赋值不影响局部变量 `val2`，因为两个局部变量属于同一个值类型（即类型 `int`），而值类型的每一个局部变量都单独储存。在常量中，赋值表达式 `ref2.value=123;` 既影响 `ref1` 所引用的对象，也影响 `ref2` 所引用的对象。以下几行则另当别论:

```
Console.WriteLine("Values: {0}, {1}", val1, val2);
Console.WriteLine("Refs: {0}, {1}", ref1.Value, ref2.Value);
```

因为它们表示 `Console.WriteLine` 的一些字符串的格式化过程，这个过程所需要的自变量数其实是可变的。第一个自变量是一个字符串，它可能含有数个座位如 `{0}` 和 `{1}`。每一个座位表示一系列自变量，如 `{0}` 表示第二个自变量，`{1}` 表示第三个自变量等等。在输出结果被传送到控制台之前，每一个自变量值都被其相应自变量格式化的值取代。

开发者可以通过枚举或结构声明定义新的值类型，也可以通过类、接口以及委托声明定义新的引用类型。

下面的例子:

```
using System;
public enum Color
{
    Red, Blue, Green
}
public struct Point
{
    public int x, y;
}
public interface IBase
{
    void F();
}
public interface IDerived: IBase
{
```