

代码详尽的实用手册

# C#

## 开发人员手册

[美] William Robison 著

邱仲潘 等译

Pure C#

SAMS



电子工业出版社  
Publishing House of Electronics Industry  
www.phei.com.cn

# C# 开发人员手册

PureC#

[美] William Robison 著

邱仲潘 等译

電子工業出版社

Publishing House of Electronic Industry

北京·Beijing

## 内容简介

这是介绍.NET平台中最新利器——C#语言的精彩著作，C#与C++、Java语言相似，但内部构造大不相同。本书首先简明扼要地介绍了C#语言本身，包括其基本的内部构造，然后翔实细致地介绍了C#语言参考和基类库（C#运行环境）中常用的组件，读者在编程时可以使用本书找到所要答案。最后，在附录中介绍了C#语言的语法元素和类库。作者假设读者已经知道如何用其他一些语言进行编程，掌握了部分基本概念，因此着力介绍C#中的新特性。本书是编程人员的宝贵参考资料，可以作为学习C#的教材，也可以作为编程时的参考手册，许多代码还可以在程序中直接借用。

Authorized translation from the English language edition published by Sams Publishing. Copyright © 2002. 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 the Publisher.

Simplified Chinese language edition published by Publishing House of Electronics Industry. Copyright © 2002.

本书中文简体版专有翻译出版版权由Pearson教育集团所属的Sams Publishing授予电子工业出版社。其原文版权及中文翻译出版版权受法律保护。未经许可，不得以任何形式或手段复制或抄袭本书内容。

版权贸易合同登记号 图字：01-2001-4964

图书在版编目（CIP）数据

C#开发人员手册 / (美)罗宾森(Robison, w.)等著;邱仲潘译.

-北京:电子工业出版社,2002.4

书名原文:Pure C#

ISBN 7-5053-7563-6

I.C...II. 罗... 邱... III.C语言-程序设计-技术手册 IV.TP312-62

中国版本图书馆CIP数据核字(2002)第021511号

## 译后记

微软公司的每个举动都代表着当今IT行业的新方向，.NET是盖茨亲自捉刀的杰作，更引起了巨大的波澜，C#语言是Microsoft公司为未来建立的NET平台技术系列的组件。Microsoft公司建立的技术环境无疑是强大而前瞻性的，而这一切的基础正是C#编程语言。我们很高兴把本书翻译出来，献给广大读者。

翻译本书时，厦门正是寒冬初到，但我们在白鹭飞起的地方，怀着巨大的热情，挥毫泼墨，细推慢敲，终于拿出自己略感满意的手稿，厦门市电脑学会理事长李棠秋教授亲自审阅了部分章节，张荣、李青、钟铨光、王凌飞、刘文琼、刘云昌、刘昌和、严明英、赖华龙、陈凌峰、陈纯颖、周阳生、邹能东、李耀平、彭振庆等朋友也在翻译整理和录排方面提供了诸多帮助。套用一句老话，这是集体智慧的结晶，在此对各位深表感谢。也盼望广大读者不吝赐教，让我们精益求精，更上一层楼。最后，感谢何大曾先生在北京期间提供的用车和其他方便。

# 目录

第一部分 基本概念 .....	1
第1章 语言元素 .....	2
1.1 应用程序结构 .....	2
1.2 类型与声明 .....	3
1.2.1 内置数值类型 .....	3
1.2.2 类类型 .....	7
1.3 接口类型 .....	15
1.4 管理控制流 .....	17
1.4.1 正常执行 .....	17
1.4.2 代理 .....	21
1.4.3 异常 .....	22
1.5 继承 .....	29
1.6 不安全代码 .....	32
1.6.1 调用外部函数 .....	32
1.6.2 编写不安全代码 .....	32
1.7 预处理器指令 .....	34
1.8 小结 .....	37
第2章 处理应用程序 .....	38
2.1 中间语言和通用语言运行时 .....	38
2.1.1 中间语言 .....	38
2.1.2 通用语言运行时间 .....	38
2.2 执行程序、汇编与组件 .....	39
2.2.1 汇编 .....	40
2.2.2 融合 .....	40
2.2.3 组件 .....	40
2.3 组件与汇编属性 .....	41
2.4 开发工具 .....	43
2.4.1 用csc编译C# .....	43
2.4.2 用nmake管理编译 .....	45
2.4.3 用sn和al建立汇编 .....	49
2.4.4 用gacutil管理汇编 .....	52

---

2.5 .NET中的调试 .....	53
2.5.1 DbgCLR调试 .....	53
2.5.2 浏览编译汇编内部 .....	55
2.6小结 .....	57
<b>第3章 基类库 .....</b>	<b>58</b>
3.1 体系结构与配置文件 .....	58
3.2 字符串与正则表达式 .....	59
3.3集合 .....	63
3.4序列化 .....	66
3.5输入与输出 .....	68
3.6网络通信 .....	72
3.6.1套接字 .....	72
3.6.2套接字通信 .....	73
3.6.3网络帮助类 .....	77
3.7小结 .....	79
<b>第4章 变量与类型 .....</b>	<b>80</b>
4.1简单数据 .....	80
4.1.1实例化与使用 .....	80
4.1.2字符串与字符串转换 .....	81
4.1.3转换类型 .....	84
4.2类 .....	86
4.3接口 .....	89
4.4结构 .....	91
4.5枚举类型 .....	91
4.6小结 .....	93
<b>第二部分 技术参考 .....</b>	<b>95</b>
<b>第5章 类与组件 .....</b>	<b>96</b>
5.1 定义实体与类 .....	96
5.2方法 .....	97
5.3属性 .....	101
5.4名字空间 .....	108
5.5小结 .....	110
<b>第6章 C#与内存管理 .....</b>	<b>111</b>
6.1 .NET框架中的内存管理 .....	111

---

6.1.1 Disposable .....	113
6.1.2 最后化器 .....	116
6.1.3 使用弱引用 .....	121
6.2 C#中的内存使用 .....	122
6.2.1 fixed与using语句 .....	122
6.2.2 有效内存管理 .....	123
6.3 小结 .....	123
<b>第7章 高级应用程序控制 .....</b>	<b>124</b>
7.1 线程 .....	124
7.2 同步 .....	128
7.3 代理 .....	132
7.4 事件 .....	136
7.5 小结 .....	138
<b>第8章 不安全代码 .....</b>	<b>139</b>
8.1 指针 .....	139
8.1.1 指针问题 .....	139
8.1.2 解决方案 .....	139
8.1.3 PlatformInvoke与内存使用 .....	140
8.2 不安全情境 .....	144
8.3 不安全语言元素 .....	145
8.4 不安全代码与内存管理 .....	147
8.5 小结 .....	148
<b>第9章 使用元数据和映射 .....</b>	<b>149</b>
9.1 使用属性 .....	149
9.2 创建定制属性 .....	152
9.3 映射与动态关联 .....	154
9.3.1 静态关联元素的映射 .....	154
9.3.2 动态装入与关联 .....	155
9.4 小结 .....	161
<b>第10章 配置组件与应用程序 .....</b>	<b>162</b>
10.1 配置汇编 .....	162
10.1.1 配置级别 .....	162
10.1.2 管理配置文件 .....	162
10.2 管理资源 .....	165
10.2.1 使用文化中立资源 .....	165
10.2.2 使用文化特定资源 .....	167

10.3小结 .....	171
<b>第 11 章 使用 SDK .....</b>	<b>172</b>
11.1 编译与链接 .....	172
11.1.1 基本编译步骤 .....	172
11.1.2 集成COM+ .....	177
11.2 调试与检查 .....	181
11.3 部署方案 .....	182
11.4小结 .....	183
<b>第三部分 附录 .....</b>	<b>185</b>
附录A C#语言 .....	186
附录B 关键类型速查手册 .....	224

# 第一部分

## 基本概念

C# 是 C++ 语言的修订与扩展，但即使 C++ 编程高手也要学习许多东西才能灵活使用 C# 编写应用程序。本书第一部分介绍了使用 C# 编程需掌握的基础知识。第 1 章将介绍 C# 语言的语法及如何用 C# 构造基本元素，第 2 章将介绍 .NET Framework SDK 中编译与链接程序和库的工具，第 3 章将介绍 .NET 平台所带的应用程序运行环境类库。

- 第 1 章 语言元素
- 第 2 章 处理应用程序
- 第 3 章 基类库
- 第 4 章 变量与类型

# 第1章 语言元素

C# 是 C++ 语言的修订与扩展，因此通过演示来学习大部分内容就很容易。但是，对 C# 语言的基本元素做一个简单而全面的论述却很难。本章将总结这个语言的语法，介绍 C# 与当前语言的差别，从而为后面的章节打下基础。

Microsoft 公司把 C# 描述为“简单、现代、面向对象和类型安全的”语言，是一种提供高生产率、使开发人员可以充分利用框架功能的工具。C# 开发小组基本达到了这些目标，提供了可以和任何类似技术相媲美的强大语言。

## 1.1 应用程序结构

C# 使用 C++ 式语法。首先看看基本语言流，程序清单 1.1 显示了如何使用简单 C# 程序打印一个文本消息。

程序清单1.1 初步认识C#

```
1: using System;
2:
3:     /// <summary>
4:     /// Demonstrates simplest C# structure.
5:     /// </summary>
6:     class SimpleStart
7:     {
8:         static void Main(string[] args)
9:         {
10:             // send some text to the screen
11:             Console.WriteLine("This is a pretty
12: simple program.\n");
13:         }
```

C# 代码由使用分号分开的语句流组成，一些语句可以包含另一些语句，用大括号组合起来。程序清单1.1中的class SimpleStart {}语句就是这种语句。通常，语句之间的空白符是无关紧要的，但关键字、标识符和其他类似语言元素中却不能有空白符。例如，下列语句是有效的：

```
int x =4444;
string y ="This is a string.";
```

而下列语句是无效的：

```
int x =44 44;           //whitespace in token is Bad Thing
string y ="This is
                    a string."; //newline in string constant is Bad Thing,too.
```

从上例中可以看出，可以用双斜杠 ( // ) 将说明语句放进代码中，也可以用传统C语言形式的注释 ( /\*comment \*/ )，或像对XML文档一样使用三斜杠。

语句可以是声明式的，如程序清单1.1第6行的class声明，建立程序结构元素；也可以是命令式的，在运行时执行某个索引，例如程序清单1.1第11行的Console.WriteLine(...)语句。

C#用名字空间(namespace)在应用程序中组织符号定义。代码中引用的任何项目都要在引用它的名字空间中声明，在用using语句标识的名字空间中声明或用完全限定标识符引用它。对于后两种情况，名字空间可以在程序中，也可以通过.NET平台的融合过程汇编到程序中。由于这个例子在第11行使用System.Console对象，因此程序第1行在全局汇编缓冲区中从CLR组件链接System名字空间。

C#与C++的不同之处在于，所有变量、函数和其他实例声明都要放在类定义中，不存在全局常量、转发函数定义之类的结构。C#也没有头文件，因为.NET环境中不需要头文件。

注释：这里说的没有全局变量与声明并不准确，实际上，只是不能使用无范围全局变量。由于类能够声明静态成员，因此仍然可以声明常量与一般实用程序，通过代码提供。运行环境中到处都有常量与函数的静态声明，但这些声明放在类声明中，避免出现名称冲突。

## 1.2 类型与声明

C#有两种类型：数值类型与引用类型。引用类型(Reference)是对象值，在堆中分配。数值类型(Value types)优化整数、浮点数之类的简单类型，将其存放在堆栈中，不需要对象初始化与删除。但是，利用制箱(boxing)过程可以像使用对象一样使用所有数值类型。

### 1.2.1 内置数值类型

数值类型大部分是系统的原子性基本类型，只有两个例外。数值类型在堆中分配，加速生成、访问与处理。表1.1列出了目前定义的C#数值类型。

表1.1 C#数值类型

类型关键字	数值	运行类型
sbyte	带符号8位int值	SByte
byte	无符号8位int值	Byte
short	带符号16位int值	Int16
ushort	无符号16位int值	UInt16
int	带符号32位int值	Int32
uint	无符号32位int值	UInt32
long	带符号64位int值	Int64
ulong	带符号64位int值	UInt64
float	32位浮点数值	Single
double	64位浮点数值	Double

(续表)

类型关键字	数值	运行类型
decimal	128 位浮点数值	Object(Decimal)
bool	布尔值	Boolean
char	通配符	Char
enum	用户定义	Int32
struct	用户定义	不定,默认为 Int32

### 固有运算符

表 1.2 列出了适用于所有数值类型的 C# 运算符,运算符的优先顺序按从高到低顺序排列,只要熟悉 C++,就很容易理解 C# 运算符。当然,C# 中没有指针运算符 (\*, ->) 和类范围运算符 (::)。C# 只用点号运算符 (.) 进行成员选择,不管成员是在数值类型中、引用中,还是属于类静态成员。尽管这样可能损失一些清晰性,但可以减少错误,特别是对于不太熟悉这个语言的编程人员。

表 1.2 C#运算符

类型	运算符	作用
基本运算符	.	成员选择 (如 myObj.member)
	[]	数组或索引下标
	↓	函数调用 (如 MyFunc(aParam))
	a++, a--	后递增 / 递减
	new	分配
	typeof	运行类型判断
	(un)checked	开 / 关边界检查
一元运算符	+, -	绝对符号
	!	布尔非
	~	位非
	++a, --a	前递增 / 递减
乘法运算符	(Type)name)a	显式类型转换 (如(int)f)
	*/	乘、除
加法运算符	%	求模
	+, -	加、减
移位运算符	<<, >>	位左移、位右移
条件运算符	<, >, <=, >=	小于、大于、小于等于、大于等于
	is	运行类型判断
	as	安全类型转换
等于运算符	==, !=	测试相等性
位与运算符	&	两个操作数具有一致的位设置时,结果位设置
位异或运算符	^	两个操作数的位设置不一致时,结果位设置
位或运算符		有一个操作数的对应为设置时,结果位设置
布尔与运算符	&&	两个操作数为真时,结果为真
布尔或运算符		有一个操作数为真时,结果为真
布尔选择运算符	?:	根据条件或相等性表达式选择一个或另一个表达式:
		<pre>boolExp?trueAction(): falseAction()</pre>

(续表)

类型	运算符	作用
赋值运算符	=	将右边值 (RHS) 赋予左边值 (LHS)
	*=,/=	将 LHS 乘 (除) 以 RHS 并将结果赋予 LHS
	% =	求模和赋值
	+=, -=	加 (减) 和赋值
	<<=, >>=	左 (右) 移位和赋值
	&=, ^=,  =	位操作和赋值

C# 中的许多对象都可以用运算符操作,但有些是没有意义的。例如,位移不适用于字符串。此外,可以在需要在代码中定义自己的运算符。由于 C# 是个强类型语言,因此运算中的所有变量都要符合多态原则中的匹配。换句话说,如果将一个对象赋予另一个对象如下:

```
a = b;
```

则 b 或者与 a 类型相同,或者要定义隐式转换,能将 b 类型对象转换成 a 类型对象。

### 使用变量

声明与使用变量很简单。声明类型实例时,使用类型名加声明的变量名,类型名后面还可以加上初始化器:

```
int myIntVar =5;
```

也可以用方括号 (C# 下标分隔符) 声明数组:

```
int [] myArrayVal =new int [] {1,2,3,4 };
```

上述例子在声明的同时将变量初始化。尽管上述代码中包括 new typename 部分,但声明并初始化数组时也可以省略这个部分:

```
int [] myArray ={1,2,3,4,5 };
```

不一定要在声明时初始化数组,但如果没有初始化器,则后面应在使用变量之前将其初始化,例如:

```
int myIntVar;
int[] myIntArr;;
...
myIntVar =new int(5);           //creates the int with value 5
myIntArr =new int [25 ];       //creates the array and zeroes elements
```

本例中,使用 new 运算符来生成适当类型的新实例(一个 int 和一个包含 25 个元素的 int 数组)。

无论选择何种初始化方式,都要在使用变量之前将其初始化。编译器要检查代码路径,在执行路径可能使用未初始化值时不允许编译这个代码。Microsoft 公司将这种强制称为“主动初始化”(positive initialization),几乎完全放弃了隐式将变量初始化为 0 的习惯做法。这里,唯一例外是数组。建立新数组时,每个元素用默认值(对数值类型)或 null (对引用类型)进行初始化。

## struct与enum

struct与enum是数值类型中较难理解的部分。枚举类型用enum关键字声明,主要是为了便于对常量值命名。但是,使用枚举类型并不是十分方便。例如下面的声明语句:

```
enum Direction {Up,Down,Left,Right }
```

这条语句声明整型标志Up, Down, Left与Right,可以使代码更清晰。对上述语句,可以声明和使用枚举的值:

```
Direction steerDirection =Direction.Up;
```

另一个用户定义数值类型是struct。struct是个小型低开销类型,将一小组相关信息组织起来。struct是个过渡对象,关于数值类型和引用类型之间,可以优化不需要引用类型支持的对象。下列声明定义了一个struct类型:

```
public struct Vertex
{
    public int x,y,z;
    public Vertex(int newX,int newY,int newZ )
    {
        x =newX;
        y =newY;
        z =newZ;
    }
}
```

本例中,Vertexstruct声明三个成员和一个构造函数。这个struct的实例作为数值类型进行处理,存放在堆栈中。但是,与其他数值类型不同的是,可以不用new运算符声明新实例,只要声明变量就可以了。复杂引用类型通常可以提供构造函数,像上例中一样,构造函数可以在实例化struct变量时调用。

提示:构造函数是特殊类型的方法,在生成类实例时自动调用。本章稍后将详细介绍构造函数。

下面介绍实例化Vertexstruct的两个方法:

```
Vertex v1(1,1,1 ); //initializes using constructor
Vertex v2;         //initializes using default
```

第一个声明v1调用声明中提供的构造函数,并用提供的数值初始化struct的字段。第二个声明利用编译器生成默认构造函数(即不带参数的构造函数),将结构中的所有字段清零。在这两个声明中,变量在声明之后立即初始化。另外,生成的默认构造函数不能重载,如果对struct声明不带参数的构造函数,则编译器会产生错误。

访问struct成员的语法很简单,而且总是相同,就是用变量名、点号加成员名。例如,要访问Vertexstruct实例变量myVar的成员值,可以用语法myVar.x, myVar.y与myVar.z。

前面曾介绍过,通过制箱机制可以把数值类型当作对象。逻辑上,数值类型实际上也是从System.Object类派生的对象。但是,除非用户请求生成,否则不生成访问其对象性所需的其他

结构,只保存其数值。但是,每个简单数值类型都有相应的类类型。使用类方法时(如对 int 调用 ToString(),取得其字符串表示),C#对变量制箱,生成相应类类型的实例,用基类型的数值将对象实例初始化,然后对对象变量调用相应方法。利用这种对象包装的“即时”实例化,与纯粹基于对象的方法相比,程序开销大大减少,只在使用对象时才生成对象。

## 1.2.2 类类型

引用类型通常指 C# 中的类,是项目类型定义,可以在程序中像对象一样实例化。事实上,大部分文档中可以把“类型”一词换成“类”,根本不会改变含义。这是因为在 .NET 和 C# 中,几乎一切都是某种类。因此,下面先介绍类声明。

类声明可以定义如下:

属性

成员隐藏修饰符(只对 new 嵌套类)

有效性修饰符(public protected private internal)

继承修饰符(sealed或abstract)

类型名

基类与字段

成员变量(在 C# 中称为字段)

常量

属性

事件

运算符

索引

构造函数与析构函数

成员函数(在 C# 中称为方法)

本节没有列出声明的正式定义(请参见附录 A“C# 语言”),只是介绍了一个类声明的范例,在以后的内容中将引用程序清单 1.2。

### 程序清单 1.2 类声明举例

```
1: [Obsolete("Use something else")]
2: sealed class MyClass : Object, IDisposable
3: {
4:     private int myField = 5;           // private initialized variable
5:     private int[] myArray;           // private array variable
6:
7:     public const int myConst = 30;    // public constant
8:
9:     protected int Multiply( int param ) // protected method
10:    {
11:        return myField * param;
```

```
12:     }
13:
14:     public void Dispose()                // public method
15:     {
16:         myArray = null;
17:         GC.SuppressFinalize(this);
18:     }
19:
20:     public MyClass()                    // constructor
21:     {
22:         myArray = new int[25];
23:     }
24:
25:     ~MyClass()                          // destructor
26:     {
27:         if ( myArray != null )
28:             myArray = null;
29:     }
30:     public int Field                    // public property
31:     {
32:         set { myField = value; }
33:         get { return myField; }
34:     }
35:
36:     public int this[int ind]            // public indexer
37:     {
38:         get { return myArray[ind]; }
39:         set { myArray[ind] = value; }
40:     }
41:
42: }
```

## 类声明

类声明依次包括属性、修饰符、关键字class、类型名、基类列表和类体。关键字class、类型名和类体是必需的，而其他元素是可选的，根据声明的类型采用。下面一一介绍了程序清单1.2中所使用的元素。

第1行使用类属性。属性是声明中的可定制修饰符，通常针对紧接在后面的项目，放在方括号中（这里使用Obsolete属性标识不宜再用的项目）。关于属性的详细内容请参见第9章“使用元数据和映射”。

第2行是开始类声明。类声明包含关键字class加类名。本例中用sealed关键字修饰类，表示这个类不能继承。也可以用abstract修饰符，表示类要派生子类才能使用。显然，同一个类不能同时使用这两个修饰符。

表示继承和接口实现时，在类名后面加上冒号，然后是0个或1个基类与0个或多个接口

名的列表,中间用逗号分隔。本例中,MyClass继承Object并实现IDisposable接口(所有类型从Object隐式派生,但这样就足以演示语法了)。与Java不同的是,这里不提供显式implements关键字。

从一个类派生另一个类时,它继承父类的所有成员。实现接口则要求用相同语法实现接口定义的成员,这与COM中基于合约的模型相似,但COM的核心只允许接口中有成员函数,而C#接口则可以包含任何方法、属性、索引和事件的组合。C#还允许接口多重继承,虽然实现类中的多重继承是脆弱的,但C++标准模板库(STL, Standard Template Library)之类的库充分体现了多重继承在接口规范中的价值。

其余类声明包含可以提供类的大多数成员例子。在每个成员声明语句中,都是先用一个访问修饰符指定哪些代码可以使用这个成员。表1.3列出了C#访问修饰符。

表1.3 C#访问修饰符

访问修饰符	访问性	适用对象
public	任何地方	class或struct
protected	类及其子类	class
private	本类	class或struct
internal	本项目	class或struct
protectedInternal	本类及其子类	class或struct

如果不对成员指定访问修饰符,则成员默认为私有的。

### 数值字段

程序清单1.2中的第4行显示了如何声明简单数值字段。其中包括了字段初始值,演示其语法,对于普通数值,初始化器是可选的。但是,首次使用变量之前,必须先将其初始化。

如果没有存储修饰符,则字段是实例字段;即对类的每个实例建立字段副本。也可以用static修饰符建立类字段,即类的所有实例共享一个副本;或使用readonly,即不允许在初始化之后改变数值。

第5行定义未初始化数组。和简单数值一样,声明中允许初始化,不过是可选的。和平常一样,数组首次使用之前,必须先初始化,因此这里在第22行的类构造函数中将其初始化。

第7行用const修饰符声明一个常量字段,这声明一个数值(通常是public),从记录pi和普朗克常量之类的不变值。需要为const字段提供初始化器,因为这种字段不允许在初始化之后改变数值。

### 方法

程序清单1.2第9行~第18行声明两个方法。Multiply()是保护方法,因此只在当前类及其子类中有效。Dispose()是公开的,因此可以在任何可访问的代码中使用。事实上,Dispose()方法的实现是IDisposable接口要求的,这个方法满足类声明中建立的约定。

一般方法声明与大多数C/C++派生语言采用相同形式;基本声明包括返回类型、方法名、参数表和作为方法体的代码块,当调用程序调用这个方法时将执行代码。

方法修饰符包括static, virtual, abstract与override。static修饰符声明方法为类范围,即不与任何特定实例相关联,它可以通过类类型引用调用,但不能访问任何实例成员。virtual方法