C# 开发人员手册

Pure C#

[美] William Robison 著

邱仲潘 等译

電子工業出版社
Publishing House of Electronics 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。

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. ①罗... ②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

责任编辑：徐　申
印 刷 者：北京牛山世兴印刷厂
出版发行：电子工业出版社　www.phei.com.cn
　　北京市海淀区万寿路173信箱　邮编：100036
经　　销：各地新华书店
开　　本：787 × 1092　1/16　　印张：18.5　字数：462千字
版　　次：2002年4月第1版　　2002年4月第1次印刷
定　　价：29.00元

目　录

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

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
➥simple program.\n");
12:         }
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
	(Typename)a	显式类型转换（如(int)f）
乘法运算符	*, /	乘、除
	%	求模
加法运算符	+, -	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： `boolExp?trueAction():falseAction()`

（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，可以在需要时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，惟一例外是数组。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，Vertex struct 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 Vertex struct 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 Vertex struct 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
protected internal	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用。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

实现派生类重载（override）这个方法所需的管道，但不要求override。abstract修饰符不带方法体，表示方法要在子类中重载。本章稍后“继承”一节将详细介绍这些修饰符。

方法取0个或多个参数，每个参数可以用相应修饰符标识为数值、引用或输出参数。不带修饰符的参数是数值参数，包含调用程序调用这个方法时提供的数值拷贝。尽管方法可能修改数值参数，但这个改变只影响拷贝，调用程序的数值保持不变。

引用参数用修饰符ref表示，在声明和调用中分别如下：

```
...
    //declare a method with a reference parameter
    public void DoSomething(ref int a ){}
...
    //invoke the method with myVar
    mcVar.DoSomething(ref int myVar );
...
```

调用方法时并不复制引用参数，函数体中进行的修改会影响调用程序的数值拷贝。由于不了解这一点可能造成意外副作用，因此这个语言要求在调用函数时（而不只是在声明时）也包括ref修饰符。这样，就可以主动表示知道其含义。必须先将传入引用参数的变量初始化，然后再调用方法。

输出参数与引用参数正好相反。引用参数可能由方法修改，但却有许多好处。另一方面，输出参数要由方法初始化。输出参数提供一个方法，可以从方法传出多个值，而不只其返回值。惟一的缺点是必须初始化每个输出参数。在调用方法的代码中，以输出参数形式传递的变量在返回值之后主动赋值。可以用out修饰符将一个方法标识为输出参数。

方法可以重载，以提供用户代码中的不同访问方法，只要声明多个同名方法即可。每个这种方法要有不同参数表（方法名和参数个数与类型构成方法签名，即signature）。修饰符和方法类型在解析使用的方法时不考虑，而只考虑签名。运行环境根据调用程序提供的参数表和可用的方法签名，确定应使用的方法，它将激活其签名匹配于调用程序提供的参数的方法。C#不允许带相同参数表及相同签名的同名方法。

构造函数与析构函数

可以定义两种特殊方法，即构造函数与析构函数。构造函数在生成实例时调用，在访问任何其他成员之前调用，而析构函数在删除实例时调用。

类可以提供类和实例构造函数。类构造函数用static修饰符标识，在使用任何静态字段和生成任何类实例之前执行。类构造函数可以初始化任何非const静态成员，实例构造函数初始化实例字段。

一个类可以用不同参数表重载实例构造函数，和任何其他方法重载一样，采用相同的解决与惟一性规则。编程人员通常实现多个构造函数，使类用户可以提供生成实例所需的信息量，类可以对其他值提供合理默认值。程序清单1.3显示了重载构造函数的用法。

程序清单1.3 使用不断具体化的构造函数

```
class Simple
{
    private int myA;
    private string myS;
    public const int defaultA = 0;
    public const string defaultS = "S";

    public SimpleStart() : this(defaultA, defaultS)    {}
    public SimpleStart( int a ) : this(a, defaultS) {}
    public SimpleStart( int a, string s )
    {
        myA = a;
        myS = s;
    }

} // end class Simple
```

程序清单1.3中的3个构造函数允许用户指定0个、1个或2个参数。如果需要，也可以实现一个构造函数，只取字符串参数。使用这种方法时，类用户得到所要的灵活性，同时又能保证类在确定状态中实例化，实际初始化代码只放在一个地方。但要注意，与C++不同的是，不能直接调用另一构造函数，而只能通过构造函数头中的初始化列表调用。

定制运算符

可以在类中声明定制运算符，这些定制运算符是另一种特殊方法，可以根据代码中的具体操作调整运算符。这在编写数学代码时很有用（经典例子是实现向量和矩阵类）。和方法一样，对于这种已定义运算符的重新声明称为重载。

大多数运算符根据操作数为一个或两个而分为一元运算符和二元运算符。例如，在代码x=-y中，=运算符是个二元运算符，而-（否定）运算符则是一元运算符（只对变量y进行操作）。表1.4列出了C#中可重载的运算符。

表1.4 C#中可重载的运算符

运算符	类型	正常语法
+, -	一元运算符	算术正负
++, --	一元运算符	递增和递减（注意C#中不能重载各个前缀与后缀递增和递减运算符）
!	一元运算符	逻辑逆
~	一元运算符	二进制逆
true , false	一元运算符	应用程序相关，要求逆
+, -	二元运算符	加、减
*, /	二元运算符	乘、除
%	二元运算符	求模
<<, >>	二元运算符	位左移、位右移
<, >, <=, >=	二元运算符	小于、大于、小于等于、大于等于，要求逆