万水计算机编程技术与应用系列

C#语言与程序设计

赵青松　卿瑞　等编著

中国水利水电出版社

内 容 提 要

C#是一种以 C 和 C++为基础的目标指向语言。该语言适用于微软公开的基于“.NET”平台的 XML 基础应用开发业务。

本书可以作为 C#语言的一本入门和提高书籍。本书在介绍基本的 C#语法规则的基础之上，结合了大量的实例，使得广大读者更易于理解和掌握。

凡是有志于程序开发的人员都可以使用此书，它既能作为入门的教材，又可作为方便使用的工具书。

图书在版编目（CIP）**数据**

C#**语言与程序设计**／**赵青松等编著**．—**北京**：**中国水利水电出版社**，2001.6

（**万水计算机编程技术与应用系列**）

ISBN 7-5084-0644-3

Ⅰ.C…　Ⅱ.**赵**…　Ⅲ.C **语言－程序设计**　Ⅳ.TP312

中国版本图书馆 CIP **数据核字**（2001）**第** 027705 **号**

书　　名	C#**语言与程序设计**
作　　者	**赵青松　卿瑞　等编著**
出版、发行	**中国水利水电出版社**（**北京市三里河路** 6 **号**　100044） **网址**：www.waterpub.com.cn E-mail：mchannel@public3.bta.net.cn（**万水**） sale@waterpub.com.cn **电话**：（010）68359286（**万水**）、63202266（**总机**）、68331835（**发行部**）
经　　售	**全国各地新华书店**
排　　版	**北京万水电子信息有限公司**
印　　刷	**北京市天竺颖华印刷厂**
规　　格	787×1092 **毫米**　16 **开本**　17.50 **印张**　380 **千字**
版　　次	2001 **年** 6 **月第一版**　2001 **年** 6 **月北京第一次印刷**
印　　数	0001—6000 **册**
定　　价	28.00 **元**

凡购买我社图书，如有缺页、倒页、脱页的，本社发行部负责调换

前　言

在过去的 20 年中，C 和 C++已经是开发商用软件和企业软件时使用最广泛的编程语言之一。这两种语言为开发者提供了大量细致灵活的控制，这种灵活性是以生产的成本为代价的。就拿 VB 来与之比较，C 和 C++应用程序相对来说需要较长的开发时间。由于复杂性和较长的周期，许多 C 和 C++程序员已经在寻找一种能在功能和生产力之间提供更好均衡的编程语言。有几种编程语言是通过牺牲 C 和 C++程序员常常需要的灵活性来提高生产力的。这样的解决方案对开发者的约束太多，并且通用性很差。它们与先前存在的系统很难相互操作，并且它们与当前的 Web 设计方法不能很好地吻合。

对于 C 和 C++程序员来说，理想的解决方法应该是快速的开发与访问所有潜在平台的能力相结合。开发环境应该完全与新的 Web 标准同步并容易与现存的应用系统集成，同时，C 和 C++程序员还希望能够在必要的时候在底层编写代码。

微软为了解决上述问题，提出了一种叫 C# 的语言（英文中读作：C sharp）。C#是一种现代的、面向对象的语言，它使开发人员能够在微软新的 .NET 平台上快速建立广泛的应用，其提供的工具和服务能充分发掘系统的计算和通讯能力。

本书共分为 10 章：第一章讲述 C#的产生背景及其主要特点；第二章讲述了 C#的数据类型和变量；第三章讲述了表达式和操作符语义；第四章介绍了 C#中的各种语句；第五章讲述了数组和结构两种数据类型；第六章介绍了重要的类类型；第七章是关于 C#中的属性、代表和事件；第八章介绍了接口、名空间和版本内容；第九章是关于条件编译和文档注释的内容；第十章讲述了安全机制和不安全代码。

全书在讲述 C#基本的语法规则的同时，还提供了大量的程序实例。读者可以结合实例来学习 C#的基本概念，并在此基础之上设计自己的 C#程序。

本书编写人员为赵青松、卿瑞、孙志刚、吴坤、谢林等，另外也感谢那些在此书写作过程中给予大力支持的各位专家教授，以及关心和支持 C#语言的各位同志。

由于作者水平有限和 C#语言处于发展初期，书中疏漏之处，在所难免，希望广大读者批评指正。

编者

2001 年 2 月

目　　录

第一章　C#语言概述

1.1　C#语言出现的历史背景

在过去的 20 年中，C 和 C++已经是开发商用软件和企业软件时使用最广泛的编程语言之一。这两种语言为开发者提供了大量细致灵活的控制，这种灵活性是以生产的成本为代价的。就拿 VB 来与之比较，C 和 C++应用程序相对来说需要较长的开发时间。由于复杂性和较长的周期，许多 C 和 C++程序员已经在寻找一种能在功能和生产力之间提供更好均衡的编程语言。

有几种编程语言是通过牺牲 C 和 C++程序员常常需要的灵活性来提高生产力的。这样的解决方案对开发者的约束太多（例如：通过省略低级代码控制）并且通用性很差。它们与先前存在的系统很难相互操作，并且它们与当前的 Web 设计方法不能很好地吻合。

对 C 和 C++程序员来说，理想的解决方法应该是快速地开发与访问所有潜在平台的能力相结合。开发环境应该完全与新的 Web 标准同步并容易与现存的应用系统集成，同时，C 和 C++程序员还希望能够在必要的时候在底层编写代码。

微软为了解决上述问题，提出了一种叫 C#的语言（英文中读作：C sharp）。C#是一种现代的、面向对象的语言，它使开发人员能够在微软新的.NET 平台上快速建立广泛的应用，其提供的工具和服务能充分发掘系统的计算和通讯能力。

C#语言将作为 Microsoft Visual Studio 7.0 的一部分而推出，同时这个开发环境还支持 Visual Basic、Visual C++和脚本语言 VBScript、JScript。微软的 Next Generation Windows Services（NGWS）平台提供了一个执行引擎和一个丰富的类库来支持上述语言的解释和执行。对于 C#开发人员来说，这就意味着即使 C#是一种新的语言，它也可以使用 Visual Basic、Visual C++使用的类库，而自身并不用包含类库。

因为其优良的面向对象设计，在构建从高级业务对象到系统级应用的各种不同组件时，C#是一个首要的选择。使用简易的 C# 语言构造，组件可以被转换为 Web 服务，从而允许从运行在任何操作系统上的任何语言中跨越 Internet 调用它们。

不仅如此，C#的设计为 C++程序员带来了快速的开发能力，而不用牺牲 C++已有的功能和控制能力。通过这种继承，C#高度地保持了与 C 和 C++的一致。开发者只要熟悉 C 和 C++语言就可以快速地掌握 C#并写出更多的 C#应用程序。

1.2 C#语言的特点

1.2.1 生产力和安全

新的 Web 经济要求所有商业比从前更快地响应竞争的威胁。要求开发者在更短的时间内生产发行更多的程序版本，而不是单一纪念物似的版本。C#的设计考虑到了这些因素，它的设计帮助开发者用更少的代码行和更少的出错机会做更多的事情。

一、遵循新的 Web 设计标准

新的应用程序开发模型意味着越来越多的解决方案需要使用新的 Web 标准，如：HTML、XML 和 SOAP（Simple Object Access Protocol）。现存的开发工具都是在 Internet 之前或之初开发的。因此，它们总是不能很好地适应新的 Web 技术。

C#程序员能用一个扩展的框架来构建微软.NET 平台上的应用。C#包括内建的支持使任何组件转换为一个能在 Internet 上运行任何平台上任何应用中被调用的服务。

对程序员来说，更出色的是这个 Web 服务框架能使现存的 Web 服务看起来像本地的 C#对象，因此允许开发者权衡利用它们已有的面向对象的编程技能与现存的 Web 服务。

有更多的特点使得 C#成为主要的 Internet 开发工具。例如，XML 是新出现的在 Internet 中传递结构化数据的方法。这种数据集通常很小。为了提高性能，C#允许 XML 数据被直接映射到一个结构数据类型而不是一个类。在数据量不大时，这是一种更有效的处理方式。

二、消除重要的编程错误

即便是专家级的 C++程序员都避免不了最简单的错误——如忘记初始化一个变量。这些简单的错误常常导致不可预见的问题，它们长时间隐蔽，不易发现。一旦程序投入生产运行后，要排除哪怕是最简单的编程错误，都要付出昂贵的代价。

C#现代的设计排除了大多数普通的 C++编程错误。例如：

- 垃圾清理减轻了程序员自己管理内存的负担。
- 在 C#中的变量是自动被环境初始化的。
- 变量类型是安全的。

三、依赖内建的转换支持降低开发成本

更新软件组件是一个容易出错的任务。修订代码版本无意中可能会改动程序的语义。为了帮助程序员解决这个问题，C#语言中包含了转换支持。例如，与 C++和 Java 不同，方法重载必须显式进行，这有助于防止代码错误和保留转换的灵活性。有关的特点还有本地的接口和接口继承支持。这些特点进化并发展了复杂的框架。

总之，这些特点使一个工程的后继版本的开发方法更加强健，同时为成功的应用降低了整体的开发成本。

1.2.2　功能、表现和灵活性

随着公司制作商业计划的水平越来越高，抽象的商业过程与实际软件实现之间的紧密连接已成一种必然趋势。但大多数语言工具没有一种简单易行的方式来链接业务逻辑和代码。例如，今天开发者可能使用代码注释来说明哪个类构成主要的抽象业务对象。

C#语言允许分类，扩展能应用于任何对象的元数据。一个工程的构建者能够定义特定领域的属性并将它们应用到任何语言的类元素，接口等等。然后开发者编程测试每个元素的属性。这就简化了自动化工具的编写，而自动化工具将保证每个类或者接口被正确表示为特定抽象商业对象，或简化了创建基于对象特定属性的过程。紧密联结定制元数据和程序代码有助于加强意料的程序行为与实际的实现之间的联系。

可管理、类型安全环境对大多数企业应用是适合的。但是现实世界中的经验告诉我们，有些应用延续需要“本地”代码，其原因要么是因为性能问题，要么是因为与现存的应用程序接口问题。这样的情况强迫开发者使用 C++来开发，即便他们喜欢使用生产性更好的开发环境。

C#通过下面的方式解决了这个问题：包含组件对象模型（COM）的本地支持和基于 Windows API 的支持。

在 C#中，每一个对象自动地成为一个 COM 对象。开发者不再需要显式地实现 IUnknown 和其他 COM 接口，而是将它们内置了。同样，C#编程能在本地使用现存的 COM 对象，与它们用什么语言创建的无关。

对于需要这种特性的开发者来说，C#包含了一个特别的特性：那就是一个程序可以调用任何本地 API。在一个特别标记的代码块中，允许开发者使用指针和传统的 C/C++特点，如自己管理内存和指针算法。

相对于其他开发环境，这是一个巨大的优点。它意味着 C# 程序员能建立他们自己现存的大量 C 和 C++代码，而不用丢弃它们。

COM 支持和本地 API 访问支持，目的是向开发者提供重要的能力和控制，不用离开 C# 环境。

具体来说则包括如下一些特点：

现代性：你在学习 C#上所付出的努力将是一项巨大的投资，因为 C#被设计成为开发.NET 应用的首选语言。你会发现许多在 C++中必须由你自己来实现或者干脆没有的特征，都成为基础 C#的一部分了。

小数类型对于企业级编程语言来说是很受欢迎的一个附加类型。你可以使用一个新的 DECIMAL 数据类型来进行货币计算。如果你不喜欢这个简单类型，可以很容易地为你的应用特别定制一个新的类型。

在 C#中，指针不再是你的编程武器了，所以你也就不必负责整个内存的管理了。.NET 平台环境提供了自动内存管理机制负责你的 C#程序的内存管理工作。

对于 C++程序员来说，异常处理是 C#的一个主要特性，但这决不是什么新鲜事。然而，和 C++不同的是，异常处理是跨语言的（运行环境的另一特点）。在 C#之前，你必须对付离奇的 HRESULT，现在不必要了，稳健的基于异常处理的错误处理机制能做好这些。

面向对象：C#作为一门新的语言理所当然的支持面向对象的所有关键概念：封装、继承和多态性。整个 C#的类模型是建立在.NET 虚拟对象系统（VOS, Virtual Object System）之上的。对象模型是基础构架的一部分，而不是编程语言的一部分。

在面向对象的程序设计中，没有全局函数、变量或者常数。每样东西必须被封装在一个类中，或者作为一个实例成员（通过类的一个实例对象来访问），或者作为一个静态成员（通过类型来访问），这会使你的 C#代码具有更好的可读性，并且减少了发生命名冲突的可能性。

在类中定义的方法缺省情况下不是虚拟的（不能被派生类所覆盖），这一做法的要点是可以去掉另一个产生错误的来源——对方法的错误覆盖。如果一个方法可以被覆盖，那么它必须有一个显式 virtual 修饰符。这样不但可以减少虚函数表的长度，还能保证正确的版本处理行为。

对于 C++程序员而言，已经熟悉了使用访问权限修饰符为类的成员指定不同的访问级别。C#也支持私有（private）、保护（protected）和公共（public）访问权限修饰符，而且增加了第四个：internal。

在实际的程序设计中，很多情况下只需要从一个类派生，从多个基类派生所带来的问题比这种做法所能解决的问题要更多，所以 C#只允许一个基类，但是完全可以用接口来实现同样的多重继承的功能。

类型安全性：在 C++中，定义了指针之后，你可以自由的把它指向任何一个类型，包括做一些相当愚昧的事情，比如将一个整型指针指向一个双精度型指针，只要内存支持这一操作，它就会凑合着工作，而往往是这种情况导致了程序的错误。

为了避免上述问题，C#实施了最严格的类型安全来保护它自身及垃圾收集器。在 C#中，必须遵守关于变量的一些规定：

不能使用未初始化的变量。对于对象的成员变量，编译器负责把它们置零。局部变量需要你自己处理。如果使用了未初始化的变量，编译器就会提醒你。这样做的好处是：你可以避免使用了一个未被初始化的变量，从而带来的严重后果。

C#不支持不安全的指向。不能将整数指向引用类型，并且 C#会时时验证指向的有效性。

边界检查是 C#的一部分。当数组实际上只有 n-1 个元素时，不可能使用超过它的元素。

算术运算可能溢出结果数据类型的范围。C#允许在应用级或者语句级检查这种操作中的溢出，当溢出发生时会抛出一个异常。

C#中传送的引用参数是类型安全的。

版本技术：在过去的几年中，几乎所有的程序员都和 DDL 打过交道，也备受折磨。产生这个问题的原因是因为许多计算机上安装了同一 DDL 的不同版本。有时老的 DDL 应用在新的 DDL 版本上还侥幸可以运行，但大多数情况下，它们都会崩溃。版本处理技术是当前面临

的一个重要问题。

在 C#中，考虑到了这种要求，它将尽其所能支持这种版本处理功能。虽然 C#自己并不能保证提供正确的版本处理结果，但它为程序员提供了这种版本处理的可能性。有了这种支持，开发者可以确保当他开发的类库升级时，会与已有的客户应用保持二进制兼容。

兼容性：C#不是存在于一个封闭的世界里。它允许你通过遵守重要的.NET 公用语言规范 CLS（Common Language Specification）访问不同的 API。CLS 定义了遵守这些规则的语言间相互操作标准。为了增强 CLS 的兼容性，C#编译器会检查所有公开输出项所遵守的条件，发现不符合规则的就会报错。

当然你也可能希望能够访问老的 COM 对象，.NET 平台提供了对 COM 的透明访问。另外 C#还支持 OLE 自动化，并且不要考虑细节问题。

最后，C#允许与 C 风格的 API 进行相互操作。动态链接库（DDL）的任何入口点（以风格给出）都可以在应用程序中进行访问。这种访问本地 API 的特性叫做平台调用服务。

简单性：C#同 C++相比，其简单性就是一个特点，它虽然增加了许多新特性，但同时舍弃了一些老的东西，对 C#的总体简单性做出了贡献。

没有指针是 C#的一个显著特性。在缺省情况下，你使用一种可控制的代码进行工作，此时，一些不安全的操作，如直接内存操作，将是不允许的。在 C++中，有::，．和－>操作符，分别用于名空间、成员和引用操作，在 C#中，去掉了别的操作符，只支持一个“.”，程序员现在需要理解的一切就是名字嵌套的概念了。

在 C#中不再需要记住那些源于不同处理器结构的神秘类型了，包括可变长的整数类型。C#通过提供了一个统一的类型系统解决了这个问题。这个类型系统使每种类型都可以被看作是一个对象。与别的语言不同的是，把单个类型看作对象并不会增加执行上的困难。

灵活性：虽然 C#代码缺省模式是安全模式，但也可以声明某些类或者仅仅是类的某些方法为非安全的，这一声明使你能够使用指针、结构和静态分配的数组。安全和不安全代码两者都在可操作空间中运行。这就意味着从安全代码调用不安全代码不会有什么问题。

1.3　一个“Hello，world”程序

这个最短的 C#版本应用程序如下，把它存起来，文件名为 helloworld.cs，以便使你能按照说明，完成诸如编译应用程序等其他余下来的步骤。

最简单的“Hello World”程序:

```
1: class HelloWorld
2: {
3:  public static void Main()
4:  {
5:   System.Console.WriteLine("Hello World");
```

```
6:  }
7: }
```

在C#中，代码块（语句组）由大括弧{和}括住。所以，甚至你以前没有C++的经验，你也可以说出Main（）方法就是HelloWorld类语句的一部分，因为类被括在所定义的大括弧中。C#应用程序（可执行）的入口点就是static Main方法，它必须包含在一个类中。仅有一个类能使用该标志定义，除非你告诉编译器它应使用哪一个 Main 方法（否则，会产生一个编译错误）。和 C++相比，Main 的第一个字母是大写的 M，而不是你曾经使用过的小写字母。在这个方法中，你的程序开始并结束。方法中可以调用其他方法——如这个例子中，用于输出文本——或者创建对象并激活该方法。正如你所看到的，Main 方法返回一个 void 类型。

```
public static void Main()
```

尽管看到这些语句时，C++程序员肯定会觉得似曾相识，但是其他程序员并不如此。首先，public 的访问标志告诉我们这个方法可以被任何程序访问，这是它被调用的必要条件。其次，static 意味着没有先创建类的实例也可以调用方法——你所要做的就是用类名调用方法。

```
HelloWorld.Main();
```

但是，不赞成在Main方法中执行这行代码，因为递归会导致堆栈溢出。

另一个重要的方面是返回类型。对于方法Main，可选择void（意味着根本就没有返回值），或用int为整型结果（应用程序返回的错误级别）。因此，两种可能的Main方法为：

```
public static void Main()
public static int Main()
```

C++程序员同样会知道后面我要提到的——可以传给应用程序的命令行参数数组。如：

```
public static void Main(string[] args)
```

现在并不想详细地说明如何访问参数，但事先给C++程序员一个警告：和C++相比，应用程序路径不是这个数组的一部分。仅仅那些参数包含在这个数组中。在对Main方法并不简短的介绍之后，让我们把注意力集中到惟一真正的代码行——这行代码在屏幕上显示“Hello World”。

```
System.Console.WriteLine("Hello World");
```

假如不是由于有了System，大家会马上猜到WriteLine是Console 对象的一个静态方法。那么System代表什么呢？它是包含Console对象的名字空间（范围），实际上并不是每次都在Console对象前加上名字空间的前缀，你可以像所示范的那样，在应用程序中引入名字空间。

在应用程序中引入名字空间：

```
1: using System;
2:
3: class HelloWorld
```

```
4: {
5:  public static void Main()
6:  {
7:   Console.WriteLine("Hello World");
8:  }
9: }
```

所有你要做的就是给 System 名字空间加一个 using 指令。在这之后，不再需要规定名字空间，就可以使用它们的方法和属性了。NGWS 框架体系中有很多的名字空间，这里只对巨大的名字空间池中的少数几个对象进行探讨。

由于 NGWS Runtime 支持所有的编译器（VB、C++和 C#），你不必买一个单独的开发工具来把应用程序编译成 IL（中间语言）。但是，如果你从没有用过命令行编译器编译过应用程序（仅懂得编译名，而没有熟记），它还是你的首要选择。

打开命令提示符并切换到保存 helloworld.cs 的目录。敲入以下命令：

```
csc helloworld.cs
```

helloworld.cs 被编译并链接成 hellworld.exe。因为源码没有错误（那当然!），C#编译器没有出错提示，在整个编译过程没有丝毫停顿。

现在你已经准备好运行第一个真正用 C#编写的应用程序了。简单地在命令行上敲入 helloworld，输出结果为“Hello World”。

尽管这本书只介绍了 C#编程的概念而不介绍用户接口编程，但需要让你迅速学会简单的屏幕输入和输出方法——相应于 C 的 scanf 和 printf，或者 C++的 cin 和 cout。这里不能提供 VB 相应的函数，因为屏幕访问不是该核心语言的一部分。你只需要能够读用户的输入并提示一些信息给用户。程序说明如何读一个用户请求的名字输入，并显示一条已定制好的“Hello”信息。

从控制台读输入信息：

```
1: using System;
2:
3: class InputOutput
4: {
5:  public static void Main()
6:  {
7:   Console.Write("Please enter your name: ");
8:   string strName = Console.ReadLine();
9:   Console.WriteLine("Hello " + strName);
10:  }
11: }
```

第 7 行使用 Console 对象的一个新方法用于提示文本信息给用户，它就是 Write 方法。它与 WriteLine 不同的地方在于它输出时不换行。使用这种方法以便用户可以在信息提示的同一行输入名字。在用户输入他的名字后（并按回车键），ReadLine 方法读入了一个字符串变量。名字字符串连接到常量字符串“Hello”，并用我们早已熟悉的 WriteLine 方法显示出来。

你几乎已学完了 NGWS 框架必要的输入和输出功能。但是，你还需要为用户显示多个值。为用户写一个格式串:

```
1: using System;
2:
3: class InputOutput
4: {
5:  public static void Main()
6:  {
7:   Console.Write("Please enter your name: ");
8:   string strName = Console.ReadLine();
9:   Console.WriteLine("Hello {0}", strName);
10:  }
11: }
```

第 9 行包含了使用格式串的 Console.WriteLine 语句。格式串例子如下：

```
"Hello {0}"
```

{0}代替 WriteLine 方法的参数表中紧随格式串后的第一个变量。你可以用该技术格式化多于三个的变量。

```
Console.WriteLine("Hello {0} {1}, from {2}",
strFirstname, strLastname, strCity);
```

当然，并不仅限于只使用字符串变量。你可以使用任何类型，这些类型在后面的 C#类型中有讨论。

当写代码时，你应为代码写注释条文，解释实现的内容、变更史等。尽管你注释中提供的信息（如果有的话）是给你写的，但是你还是必须遵守写 C#注释的方法。

```
1: using System;
2:
3: class HelloWorld
4: {
5:  public static void Main()
6:  {
7:   // 这是单行注释
8:   /* 这种注释
```

```
9:   跨越多行 */
10:   Console.WriteLine(/*"Hello World"*/);
11:  }
12: }
```

“//”符号用于单行注释。你可以用“//”注释当前所在行，或是跟在一个代码语句的后面：

```
int nMyVar = 10; // 这里是注释
```

所有在“//”后面的被认为是一条注释；所以，你可以同样用它们来注释一整行或一行源代码的部分。这种注释方式同 C++中介绍的相似。如果你的注释跨越多行，必须使用“/* */”的字符组合。这种方式在 C 中有效。除了单行注释外，这种方式在 C++和 C#也同样有效。因 C、C++和 C#都使用这种多行注释方式，所以它们也使用相同的终结符。请看下列代码行：

```
/* Console.WriteLine("Hello World"); */
```

使用“/* */”简单地注释一整行，现在假定这一行是很长代码的一部分，而且决定要暂时禁用一个程序块：

```
/*
...
/* Console.WriteLine("Hello World"); */
...
*/
```

这个结构所存在的问题为：“Hello World”那一行后面的“*/”终止了始于第一行的“/*”的注释，余下的代码对编译器有效，你将看到一些有趣的出错信息。至少最后的“*/”被标志为归属错误。这里只不过想提醒一下，让你了解这种错误。

1.4　命令行参数

下面的例子讲述怎样访问命令行参数，以及访问它们的两种办法。

例 1

本例显示出，怎样打印出命令行参数。

```
// CommandLine\cmdline1.cs
using System;
public class CommandLine
{
   public static void Main(string[] args)
  {
    Console.WriteLine("Number of command line parameters = {0}", args.Length);
```