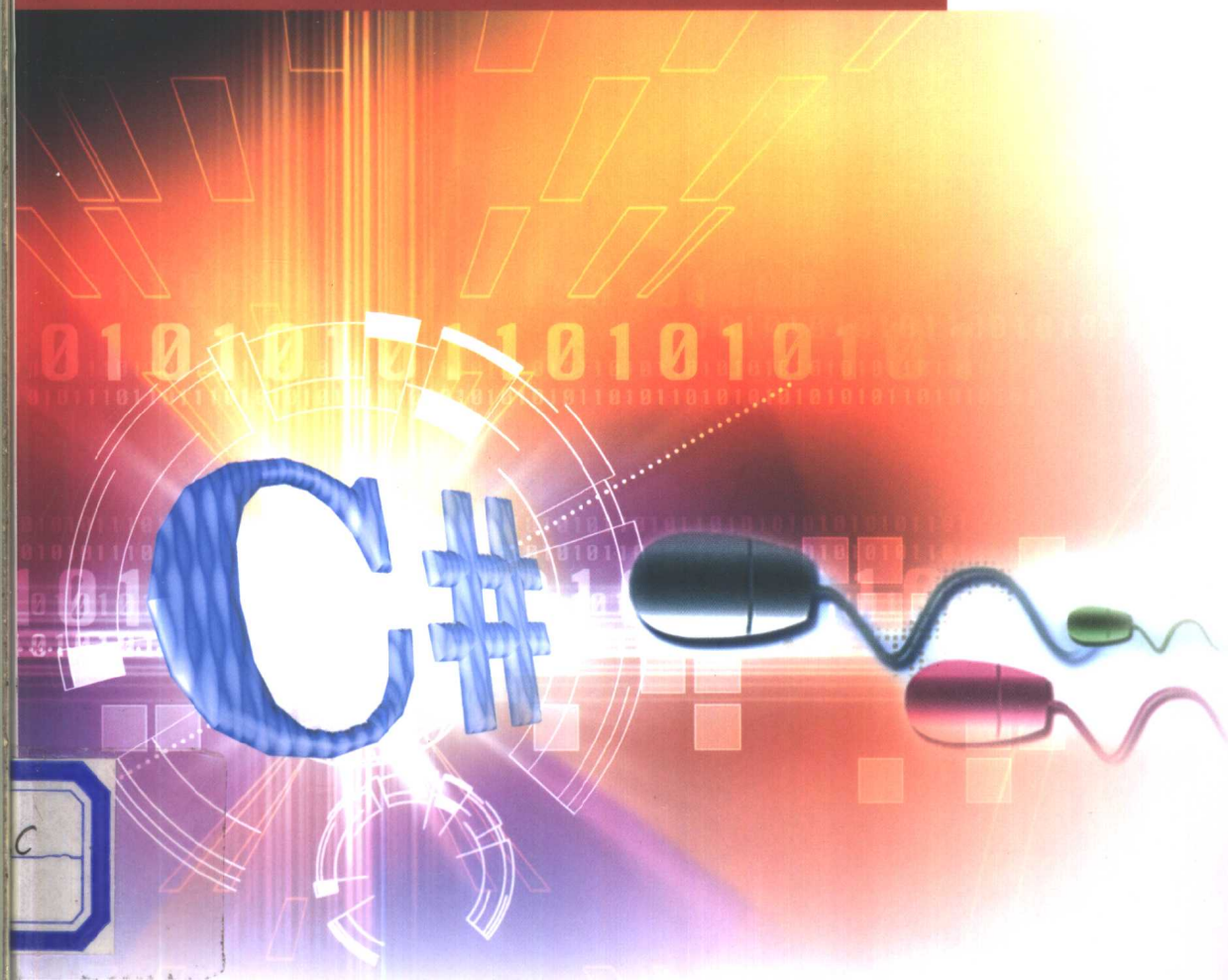


Visual C#.NET

入门与进阶

冉林仓 尹建民 编著



清华大学出版社

Visual C# .NET 入门与进阶

冉林仓 尹建民 编著

清华大学出版社

内 容 简 介

本书是《Visual Basic.NET 入门进阶》、《Visual C++ .NET 入门进阶》两书的姊妹篇。作者在 1 年多的时间里,收集、整理、加工、调试了大量 Visual C# 应用程序,并汇集成书。本书共计 17 章,120 多个示例程序,这些程序例子涵盖了 Visual C# 从语言到用户界面和系统编程的方方面面。主要包括:C# 快速入门、C# 控件的创建、按钮控件的创建、列表视图框和下拉列表框的创建、文本编辑框的创建、列表视图和树视图控件的创建、菜单和工具栏控件的创建、选择控件的创建、Tab 和属性页的创建、杂项控件的创建、多媒体和 GDI+ 程序的开发、数据库和 ADO.NET 的应用、杂类编程、COM 互操作性编程、网络编程、系统编程、XML 和 Web Service 等内容。

本书面向 C# 初、中级读者,以大量详实的实例和讲解介绍了使用 C# 如何创建漂亮的 Windows 应用程序界面,以及它在 COM、API、数据库、XML、WebService 方面的应用,本书以帮助读者实际应用为宗旨,力图使读者达到学以致用目的。

版权所有,翻印必究。

本书封面贴有清华大学出版社激光防伪标签,无标签者不得销售。

图书在版编目(CIP)数据

Visual C#.NET 入门与进阶/冉林仓,尹建民编著.北京:清华大学出版社,2003
ISBN 7-302-06281-1

I.V... II.①冉...②尹... III.C 语言—程序设计 IV.TP312

中国版本图书馆 CIP 数据核字(2003)第 005101 号

出 版 者: 清华大学出版社(北京清华大学学研大厦,邮编 100084)

<http://www.tup.tsinghua.edu.cn>

责任编辑: 宋 韬

印 刷 者: 北京市清华园胶印厂

发 行 者: 新华书店总店北京发行所

开 本: 787×1092 1/16 **印张:** 24.75 **字数:** 566 千字

版 次: 2003 年 2 月第 1 版 2003 年 2 月第 1 次印刷

书 号: ISBN 7-302-06281-1/TP·4749

印 数: 0001~5000

定 价: 40.00 元

前 言

本书汇集了 Visual C# 问世以来来自全世界 Visual C# 和 Visual C++ 程序员无偿奉献的源程序,既有短短几行切中要害的关键代码,又有完整应用的实现。这些代码无论对于 Visual C# 的初学者还是老手,以及从事其他语言编程的程序员如 Visual Basic.NET、Delphi.NET、托管 C++, 都具有很高的参考价值。读者通过使用这些代码,可以少走一些弯路,从而加快项目的开发进度。不过在使用这些程序的时候,读者一定要尊重程序原创作者的劳动成果,引用时保留程序的出处和作者的姓名。

本书共分 17 章,从 C# 快速入门讲起,使熟悉 C++ 的用户对 C# 有一个初步的了解,从而快速进入 C# 世界。然后通过几个简单的基本实例介绍了如何在 Visual C# .NET 环境下创建和使用自己的控件。接着又以大量的控件示例,介绍了它在创建复杂应用程序界面方面的应用,以及它在数据库、多媒体、图形操作、网络编程、组件编程方面的应用。这些例子生动详实、深入浅出,能够满足各个层次读者朋友的需要,并能从根本上提高读者的编程和应用水平。

本书的代码主要来自 www.codeproject.com、www.codeguru.com、msdn.microsoft.com 以及 www.vcdj.com 等网站。这些代码可能还存在一些 Bug,读者遇到一些问题不妨访问这些网站,查询代码是否更新。也可以直接和源码的作者进行交流。

书中为了查阅方便,大部分的程序都给出了完整的源码,对于个别特复杂的功能实现,由于代码过长,仅给出使用代码的方法。详细的实现代码可以参考上述网站或者从作者主页 [Http://www.microran2002.com/](http://www.microran2002.com/) 下载。

本书主要由冉林仓、尹建民编写,另外参加本书编写的还有薛年喜、曹爱东、曹东兴、郝钢、刘元成、王献州、朱江伟、李书领、贾金木、朱枫、李记增、台双良、李升奇、谢希军、胡天国、罗培荣、崔勇、易攀峰、樊立新、贾百成、赵大茗、于海、贾新萍、彭珍瑞、张利萍、徐辉、孙长华、陈永运等人,在此,对于他们的辛勤的劳动表示衷心的感谢,同时对于那些无偿奉献源码的作者,表示深深的敬意。

作 者

2003 年 1 月

目 录

第 1 章 C# 快速入门	(1)
1.1 C# 简介	(1)
1.1.1 简单性(Simplicity)——C# 与 C++ 简单性的比较	(2)
1.1.2 一致性(Consistency)——C# 与 C++ 一致性的比较	(3)
1.1.3 全新化——C# 相对于 C++、Java 全新在哪里	(5)
1.1.4 面向对象(Object Oriented)——C# 对 C++ 的扬弃	(6)
1.1.5 类型安全——C# 数组的应用	(7)
1.1.6 可伸缩性(Scalability)——C# 对编译器的改进	(9)
1.1.7 版本支持(Version Support)——名称空间的应用	(9)
1.1.8 兼容性(Compatibility)——动态链接库和 COM 的应用	(10)
1.1.9 灵活性(Flexibility)——C# 对本机代码的支持	(11)
1.2 从 C++ 到 C#	(12)
1.2.1 本机环境到托管环境的移植	(12)
1.2.2 常见的语法陷阱	(13)
1.2.3 引用类型和值类型	(13)
1.2.4 结构	(13)
1.2.5 对象的派生——类成员函数的重载举例	(13)
1.2.6 引用参数和输出参数的比较	(15)
1.2.7 new 的使用——对象初始化举例	(15)
1.2.8 属性(Property)的应用举例	(16)
1.2.9 数组	(17)
1.2.10 索引(Index)的应用举例	(18)
1.2.11 接口	(20)
1.2.12 IEnumerable 接口的应用举例	(20)
1.2.13 基类类库的使用——异步 IO 应用举例	(22)
第 2 章 C# 控件的创建	(25)
2.1 创建控件	(25)
2.1.1 创建的具体步骤及代码	(25)
2.1.2 控件的使用和测试	(28)
2.2 使用 .NET SDK 创建一个时钟控件	(28)
2.3 渐变色控件的创建	(33)
第 3 章 按钮控件的创建	(38)
3.1 AutoRepeat 按钮的实现	(38)
3.2 使用 .NET SDK 创建位图按钮	(41)
3.3 LED 风格的按钮的实现(一)	(42)

3.4	LED 风格的按钮的实现(二)	(44)
第 4 章	列表框和下拉列表框的创建	(49)
4.1	大小自动变化的列表框的创建	(49)
4.2	Visual Studio.NET 风格的平面下拉列表框的创建	(51)
4.3	C# 颜色列表框的创建	(55)
4.4	驱动器下拉列表框的创建	(57)
4.5	平面的驱动器下拉列表框的创建	(64)
4.6	多列列表框的创建	(65)
4.7	图标列表框的创建	(67)
4.8	自绘的下拉列表框的创建	(70)
4.9	服务器下拉列表框的创建	(71)
4.10	URL 下拉列表框的创建	(73)
4.11	带有提示文本框的列表框的创建	(74)
第 5 章	文本编辑框控件的创建	(77)
5.1	实现有效检查功能的文本框	(77)
5.2	具有掩码的文本框控件的实现	(80)
5.3	背景颜色高亮显示的 RichTextBox 控件的创建	(83)
第 6 章	列表视图和树视图的创建	(85)
6.1	C# 排序列表视图的创建	(85)
6.2	自绘树显示列表的实现	(88)
6.3	基于 TreeView 的属性树控件的实现	(92)
6.4	XP 风格的收缩面板控件的实现	(93)
第 7 章	菜单和工具栏控件的创建	(95)
7.1	Visual Studio.NET 风格的工具栏的实现	(95)
7.2	C# ReBar 控件的实现	(97)
7.3	ToolBar 接口扩展举例	(98)
7.4	Visual Studio.NET 风格的菜单的实现	(101)
7.5	状态栏信息的接口扩展举例	(104)
7.6	可拖动和缩放的停泊控件的实现	(106)
7.7	.NET 风格的 SideTab 控件的实现	(116)
第 8 章	选择控件的创建	(117)
8.1	Visual Studio.NET 风格的颜色拾取控件的实现	(117)
8.2	颜色拾取控件的实现	(118)
8.3	RangeBar 控件的实现	(121)
8.4	文件夹浏览组件的实现	(123)
8.5	浏览文件夹对话框应用举例	(126)
8.6	使用 Form 控件创建文件夹选择对话框	(129)
第 9 章	Tab 和属性页的创建	(133)
9.1	Windows XP 外观的 Tab Page 控件的创建	(133)

9.2 可拖放的 Tab 控件的实现	(137)
第 10 章 杂项控件的创建	(142)
10.1 增强进度条控件的创建	(142)
10.2 Outlook Bar 控件的创建	(144)
10.3 具有平面外观和活动状态的平面控件的创建	(146)
10.4 仅绘制标题的 GroupBox 控件的实现	(150)
10.5 带有进度条的 DataGrid 控件的实现	(151)
10.6 支持气球提示的任务状态栏控件的创建	(154)
10.7 Delphi 风格的事件列表控件的实现	(163)
10.8 语言国际化的属性网格控件的实现	(166)
10.9 进度对话框应用举例	(171)
10.10 以 ActiveX 控件方式使用 Windows Form	(173)
10.11 工具栏提示的添加举例	(177)
10.12 使用不同的透明度实现异形窗口	(177)
10.13 “换肤”技术的实现	(178)
10.14 Timer 组件的应用举例	(179)
10.15 高精度定时的实现	(180)
10.16 改变 DataGrid 单元的颜色	(182)
10.17 气球窗口提示框的实现	(184)
第 11 章 多媒体和 GDI+ 程序的开发	(187)
11.1 使用 Windows 媒体播放器实现声音特效	(187)
11.2 使用 C# 实现的 DirectShow 媒体播放器	(190)
11.3 避免 C# 绘图时的屏幕闪烁	(195)
11.4 克服 Graphics.MeasureString 的局限性	(198)
11.5 使用 Reflection 产生完整的颜色列表图	(199)
11.6 图像格式的转换举例	(200)
11.7 使用 GDI+ 实现像素的 Alpha 融合	(201)
11.8 在控制台应用程序中输出彩色文字	(207)
第 12 章 数据库和 ADO.NET 的应用	(210)
12.1 基于 SQL Server 的相册查看器的实现	(210)
12.2 使用 COM 互操作实现 ADO 数据存取	(214)
12.3 使用 C# 实现读取、插入、修改和删除数据库	(217)
12.4 Access 数据库编辑器的实现	(220)
第 13 章 杂类编程	(223)
13.1 拖放操作和剪贴板查看器的实现	(223)
13.2 .NET 的剪贴板操作举例	(226)
13.3 工作者线程应用举例	(229)
13.4 使用事件实现线程的同步	(232)
13.5 在 C# 中使用指针	(237)

13.6	枚举和结构的应用举例	(240)
13.7	事件和事件处理的应用	(243)
13.8	继承和多态的应用举例	(246)
第 14 章	COM 互操作性编程	(251)
14.1	使用 MSHTML 高级寄主(Host)接口	(251)
14.2	在 C# 程序中应用 WebBrowser 控件	(253)
14.3	在托管环境下使用 MSAgent 控件	(255)
14.4	读写 INI 的文件	(259)
14.5	C# 实现 Shell 扩展举例	(260)
14.6	命令提示的 Explorer Bar 的实现	(266)
14.7	本机动态链接库的后期绑定应用举例	(284)
14.8	在 C# 程序中捕获 IE 的实例	(287)
14.9	使用 C# 创建 Excel 电子表格	(289)
第 15 章	网络编程	(291)
15.1	FTP 组件的实现	(291)
15.2	网络监听程序的实现	(294)
15.3	SNTP 客户端的实现	(300)
15.4	基于 HTTP 协议的在线拼写检查的实现	(302)
15.5	异步的 Socket 通信的实现	(304)
15.6	基于 TCP 协议客户端/服务器编程举例	(310)
15.7	基于 Pop3 协议收取 E-mail 的实现	(312)
15.8	使用 SMTP 协议发送邮件	(315)
15.9	MAPI 函数的使用	(317)
15.10	创建自己的 Web 服务器	(319)
15.11	实时的 TCP/IP 协议的实现	(327)
15.12	使用 C# 发送有附件的 E-mail	(332)
15.13	获得主机的 IP 地址	(333)
第 16 章	系统编程	(335)
16.1	读写二进制文件	(335)
16.2	监视文件系统的修改	(336)
16.3	汇编语言在 C# 中的应用举例	(337)
16.4	C# 管理 Windows NT 服务	(341)
16.5	使用 SHGetFileInfo 取得文件或文件夹图标	(345)
16.6	C# 实现屏幕保护程序	(347)
16.7	注册编辑表的存取举例	(349)
16.8	文件分割程序的实现	(352)
16.9	从头创建 Windows NT 服务	(356)
第 17 章	XML 和 Web Service	(362)
17.1	XML 目录列表创建举例	(362)

17.2	HTML 解析器的实现	(366)
17.3	从 XML 资源文件中存取二进制数据	(370)
17.4	XML 客户端提供者的实现	(373)
17.5	INI 文件到 XML 文件转换的实现	(373)
17.6	从头创建 C# Web Service 程序	(377)
17.7	Google Web Service 客户端程序的实现	(381)

第 1 章 C# 快速入门

1.1 C# 简介

计算机技术普及以来,大约每隔 10 年,开发人员就必须花费时间和精力去学习一种新的语言。早在 20 世纪 80 年代,编程人员为了适应 Unix 的开发,必须掌握 C 语言的编程;到了 90 年代,整个 PC 世界成了 Windows 的天下,编程人员不得不又去适应 C++。进入 21 世纪,在未来的 10 年中,将是微软公司 .NET 的世界,编程人员为了适应 .NET 框架的编程,将不得不去面对 C# (拼作 CSharp)。C# 是未来 10 年最重要的开发工具,它源于 C++,但是它全面超越了 C++,同时它比 Java、Visual Basic、Delphi 出现的晚,作为一种全新的开发工具,它汲取了这些语言的精华,成为一种非常优秀的快速应用开发工具。

很多开发人员都希望有一种语言能够像 Visual Basic 那样简单,容易编写和维护;同时他们也希望这种语言能够提供 C++ 强大的功能和灵活性。对于这些开发者而言,C# 是他们最好的选择。C# 是微软公司在 2000 年 6 月推出的全新开发工具,它提供了 C++ 强大的功能和 Visual Basic 的简单性。同时 C# 还具有以下一些优点:类型安全、自动垃圾回收、类型声明简化、版本和伸缩性支持。这些优点使得项目的开发速度大大加快,而且变得异常容易,特别是对于 COM+ 和 Web Service 的开发。

也许还有不少开发人员在抱怨,C# 是一个基于 .NET 框架的开发工具,它和以往的开发工具是不兼容的,也就是说它的运行必须需要 .NET 框架支持,而迄今为止还没有一个操作系统预先安装了 .NET 框架,为了运行 .NET 应用程序,客户端必须安装一个接近 120 MB 的 .NET 框架环境。这的确是一个现实,不过所幸的是,微软公司已经把发展的重心移向了 .NET,微软公司不再提供 Windows 98 的销售(.NET 框架运行需要 Windows 2000 版本以上的操作系统支持)和售后服务。Windows XP 将是微软公司未来的主流操作系统,在已经发布的 Windows XP 的 Service Pack 1 中,.NET 框架将可以作为可选项目安装,Windows XP 的服务器版 Windows .NET Server 和 Windows XP 的升级版本 Longhorn,以及后面 Black Comb 操作系统将把 .NET 框架作为标准组件安装。而且 Borland 公司也已经发布了他们基于 .NET 框架的 Delphi 版本。.NET 框架环境可以从微软公司的网站免费下载,而且按照微软公司官方所言,未来的 Windows 环境将基于 .NET,所以开发人员无须担心他们的产品是否被市场所认可和接受。

下面分 9 个部分,分别对 C# 的特点进行描述,以使用户对 C# 有一个宏观上的认识。

1.1.1 简单性 (Simplicity)——C# 与 C++ 简单性的比较

使用 C++ 最大的烦恼莫过于操作符的使用,它一会儿要求使用“→”,一会儿又使用“::”,还有使用“.”操作符。为了正确的使用操作符,用户不得不去跟踪每一个类变量的声明。在 C# 中就不同了,C# 没有那么多类成员操作符,它只有一个“.”操作符,无论是一个名称空间、一个类、一个结构、枚举变量、引用等,都使用“.”操作符,没有其他操作符。

使用 C++ 和 C 的另外一个难点是数据类型的精确声明,在 C++ 中用户不得不指出字符串是 Unicode 或是 Ansi 类型字符串,在 C# 中,一个 Unicode 字符本身就是一个 Char 类型的字符,它没有 wchar_t 字符,也没有 unsigned char、signed char;64 位整数类型对应的是 long 类型,在 C# 中没有 _int64。

使用 C++ 和 C 还有一个难点就是把整数作为布尔类型来使用,这样往往会导致很多赋值错误,特别是使用“=”和“==”操作符的时候。在 C# 中,一个布尔类型只能为 true 或者 false,而不能把它转换成其他类型,也就是说整数类型或者引用的对象不能被检测为 true 或者 false。它只能和零值进行比较,这样也避免了许多混淆,使程序的可读性大大增强。如下面例 1-1 所示。

【例 1-1】 在 C++ 中,可以使用下面代码。

```
int i;  
if(i){  
    ...  
}
```

而在 C# 中,只能使用下面语句代替。

```
int i;  
if(i != 0){  
    ...  
}
```

另外一个比较好的改进是对 switch 语句的使用。在 C++ 中代码的运行会通过 case 语句执行,除非它遇到一个 break 语句。如下面例 1-2 所示。

【例 1-2】 switch 在 C++ 中应用。

```
switch(i)  
{  
    case 1:  
        FunctionA();  
    case 2:  
        FunctionB();  
}
```

在 C++ 代码中,如果 i 为 1,那么程序将先后执行 FunctionA 和 FunctionB,如果不想让它执行 FunctionB 函数,则必须在 FunctionA 函数调用之后执行 break 语句。

而在 C# 语法中,它和 Visual Basic 基本上是一致的,当 i 为 1 时,程序并不会执行 FunctionB,它隐含执行了一个 break 语句。如果用户希望代码执行了 FunctionA 之后接着执行 FunctionB,那么它就必须执行一个 goto case 2,该语句如下例 1-3 所示。

[例 1-3] 在 C# 中的 case 语句。

```
switch(i)
{
case 1:
    FunctionA();
    goto case 2;
case 2:
    FunctionB();
}
```

1.1.2 一致性 (Consistency)——C# 与 C++ 一致性的比较

在 C# 语言中,无论数据类型是一个类、一个结构、一个数组或者一个基本元素,类型系统都将把它视为一个对象(Object),这些对象被包含到一个名称空间之中,它不再需要 C++ 中的包含文件,例如,下面例 1-4 的 C++ 声明。

[例 1-4] C++ 的包含语句。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

在 C# 中将使用 using 语句引用一个特定的名称空间,通过它用户可以存取该空间所有的类和对象,如下面代码所示。

```
using system;
```

有了这个包含语句,在使用这个名称空间的类和对象的时候,不必书写名称空间的名字,可以直接使用这个名称空间下面的类和对象。如例 1-5 代码所示。

[例 1-5] 名称空间的使用。

```
using system;
Console.WriteLine("Hi,Hello World! ");
```

和

```
using system;
System.Console.WriteLine("Hi,Hello World! ");
```

是一样的。

一个完整基于控制台的 Hello World 的应用程序如例 1-6 所示。

[例 1-6] C# 的 HelloWorld 程序。

```
using system;
```

```
class helloWorld{  
    Public static int Main(String[] args)  
    Console.WriteLine(S" Hi,Hello World! ");  
    return 0;  
}
```

熟悉 Java 语言的用户,很快就会发现,这段 C# 代码和 Java 语言有很多相似之处,程序没有脱离类存在的全局变量和全局函数,C++ 的全局函数或者全局变量必须作为一个类成员形式存在,程序的静态入口函数为 Main,一个应用程序只能有一个 Main 函数,而且大小写敏感。

如果用户需要重用这个类,可以把代码变换成如例 1-7 所示的样子。

[例 1-7] C# 的代码重用。

```
using system;  
namespace CSharpTutor  
{  
    class helloWorld{  
        Public static int SayHi(String[] args)  
        Console.WriteLine(S" Hi,Hello World! ");  
        return 0;  
    }  
}
```

然后用户可以把这个文件编译成为一个动态链接库文件,使用这个动态链接库时,可以把它包含到其他项目的程序中。使用的方法如例 1-8 所示。

[例 1-8] 使用动态链接库。

```
using system;  
using CSharpTutor;  
  
class Caller{  
    Public static int Main(String[] args)  
        helloWorld.SayHi();  
    return 0;  
}
```

最后需要指出的是,如果用户编码时在两个名称空间中有两个类的名称是一样的,C# 允许用户分别为它们定义一个别名,而不必书写类型的全部名称,即可以省去名称空间的书写。例如,下面的例 1-9,假定创建了一个类 NS1.NS2.ClassA。

[例 1-9] 重名类的使用。

```
namespace NS1.NS2{  
    class ClassA{  
    }
```

```
}
```

然后再从上面的 ClassA 派生一个新类。

```
Namespace NS3|  
    class ClassB:NS1.NS2.ClassA|  
    |  
|
```

显然这个构造的名字声明显得特别长,为了简化代码的书写,可以为 NS1.NS2.ClassA 起一个别名 A,如下面代码例 1-10 所示。

[例 1-10] 别名的使用。

```
Namespace NS3|  
    using A = NS1.NS2.ClassA;  
    class ClassB:A|  
    |  
|
```

显然这个简化后的书写方式方便很多了,当然用户也可以为 NS1.NS2 起一个别名。代码如下:

```
Namespace NS3|  
    using A = NS1.NS2;  
    class ClassB:A.ClassA|  
    |  
|
```

1.1.3 全新化——C# 相对于 C++、Java 全新在哪里

作为脱胎于 C++ 的一种语言,C# 和 Java 一样对 C++ 进行了改进,其中最激动人心之处莫过于垃圾自动回收了。作为一个编程的常识,用户都知道当某一个变量不再引用时,应该清理它占用的内存。在 C++ 中对于一些局部变量或者系统分配的变量,系统会自动回收它占用的内存,例如,基于堆分配的变量。但是系统不会自行释放用户自己分配的内存,由于用户忘记释放内存造成的内存泄漏,在一定程度上会给程序的稳定性造成隐患。C# 能够自动回收用户分配的内存,用户可以大胆分配内存,而不用关心它的释放,而且 C# 实现了类型安全,确保应用程序的稳定性。当然类型安全也使代码可读性大大增强,更有利于团队的开发。

C# 比 C++ 有更丰富的错误处理模型,在程序编写的过程中可能会遇到大量的未检查的 HRESULTs,也就是说程序中有大量的函数运行后返回了一个结果值,但是程序并没有对运行的结果进行有效的检查处理,这样任何一个调用的失败,都会造成一连串的连锁反应,程序会弹出一个丑陋的错误对话框,以示错误。为了改进这一状况,C# 提供了对包括 throw、try...Catch、try...finally 关键词的内在支持。在 C++ 中,也有类似的关键词,不过它们都是通过宏来实现的。

作为一种现代化的语言,往往需要它能够更加贴近实际的现实生活。例如,基于财务方面和基于时间的数据类型,尽管实现起来十分简单,但是还是有很多语言忽视了这方面的需求。C# 和 SQL 语言提供了对 decimal、string 数据类型的内在支持,而且它还允许在用户创建和现有类型一样高效的原始类型。

C# 提供了更加现代的调试方法,在以往的 C++ 程序中,用户往往经常需要使用 #ifdef 和 #endif 关键字。这样一部分代码只能运行在调试阶段。用户最终可以生成调试版本和发行版本。发行版本的一些函数调用将执行一些空操作。C# 提供的条件关键字可以根据定义的标记决定程序的流向。例如,以上面例 1-7 CSharpTutor 名称空间为例,这里向它加入一个条件编译语句,如下例所示。

[例 1-11] 条件编译的使用。

```
using system;
namespace CSharpTutor
{
    class helloWorld{
        [condition(" DEBUG" )]
        Public static void SayHi(String[] args)
        {
            Console.WriteLine(" Hi,Hello World! ");
            return ;
        }
    }
}
```

注意:条件函数只能是 void 类型,就像上面书写的那样,客户端程序需要加上 #define DEBUG。

```
using system;
using CSharpTutor;
#define DEBUG
class Caller{
    Public static int Main(String[] args)
    {
        helloWorld.SayHi();
        return 0;
    }
}
```

这种方法比使用 #ifdef 语句更直观,而且可以按照两个编译方式生成不同的代码。

1.1.4 面向对象 (Object Oriented)——C# 对 C++ 的扬弃

C# 是一个面向对象的语言,但是它抛弃了 C++ 多重继承,却提供了对 COM+ 虚拟对象系统的内在支持,封装、多态、继承也被保留了下来。

C# 没有全局函数、全局变量、常量的概念,不过可以使用创建类的静态成员来代替它们,这样一方面增强了代码的可读性,同时也避免了许多命名冲突。

说起命名冲突,很容易使人想起函数的重载和虚函数。默认情况下 C# 的方法不是

虚函数,除非用户在方法的前面加上 `virtual` 修饰符。这样,几乎不可能因为意外而重载一个方法,很容易提供一个正确的版本,而且 `VTable` 的增长并不快。C# 中的类成员可以定义为 `private`、`protected`、`public` 或者 `internal`。用户可以对它们的封装实现完全的控制。

在 C# 中,方法和操作符都可以被重载,使用的语法也比 C++ 容易的多,用户不能重载一个全局的操作符函数,重载只能限于局部范围内。下面的例 1-12 就是一个函数重载的例子。

[例 1-12] 函数的重载。

```
interface ITest
{
    void f();
    void f(int x);
    void f(ref int x);
    void f(out int x);
    int f(int x, int y);
    int f(string s);
}
```

1.1.5 类型安全——C# 数组的应用

类型安全可以提高程序的健壮性,例如,所有动态分配的对象和数组都被初始化为 0,对于一些局部变量,尽管 C# 不会自动对它们实现自动的初始化,但是如果用户在使用这些变量之前,没有对其进行初始化赋值,在编译时就会产生警告错误。当使用一个数组时,同样系统会自动实现范围检查防止数组越界,而且同 C++ 和 C 语言不一样的是,在 C# 中用户不能存取没有分配的内存。

在 C# 中,用户不能创建一个无效的引用,所有的 `cast` 操作都必须是安全的,用户不能在一个 `integer` 和 `reference` 类型之间进行 `cast` 转换。C# 的自动垃圾回收能够确保程序代码不会出现内存泄漏,与此相关的是数据溢出检查,数值运算和转化如果超出了目标变量的表示范围都将是不允许的。除非用户出于某种原因想让某一个变量溢出,这时可以显式地禁止这种溢出检查。

前面说过,C# 和 C++ 中支持的数据类型是不一样的,例如,在 C# 中 `char` 类型是 16 位的,而在 C++ 中则为 8 位。一些有用的数据类型如 `string` 和 `decimal` 也被集成到语言中。C# 与 C++ 之间另外一个最大的不同就是数组的使用。

C# 的数组是一种托管类型,该类型是引用而不是数值,这样能够实现垃圾回收。声明数组的方法有多种,包括多维数组 (`rectangular`) 和数组的数组 (`jagged`)。注意,下面的例子中方括号出现在类型的后面,而不像某些语言,出现在变量名的后面。

[例 1-13] 数组的声明。

```
int[] intArray;           //简单数组
int[, ] intArray;         //三维数组
int[][] intArray;         //一个数组的数组即 jagged 数组
```



```
int [,,][,] intArray;           //复合数组
```

数组实际上也是对象,当用户第一次声明数组时,数组并没有确切的大小,正因为这个原因,用户必须在声明之后对它进行创建,例如,一个大小为 5 的简单数组可以采用下面的方法创建。

```
int [] intArray = new int[5];
```

如果用户执行两次同样的操作,数组大小会自动重新分配,例如:

```
int [] intArray;  
intArray = new int[5];  
intArray = new int[15];
```

这样 intArray 数组中的元素个数就达到了 15 个,实例化一个二维数组的方法可以采用下面的代码。

```
int [,] intArray = new int [3,4];
```

实例化一个 jagged 数组稍微麻烦些,用户可以这样声明一个 jagged 数组 int [3][4], 语句如下:

```
int [][] intArray = new int[3][];  
for(int a = 0; a < intArray.Length; a++) {  
    intArray[a] = new intArray[4];  
}
```

当然用户还可以在一行之内实现数组的创建和初始化,语句如下:

```
int [] intArray = new int[5]{1,2,3,4,5};  
string[] strArray = new string[3]{"BeiJing", "ShangHai", "GuangZhou"};  
int [,] intArray = new [3,2] { {1,2}, {3,4}, {5,6} };  
int [][] intArray = new [][] { new int[] {2,3,4}, new int [] {5,6,7} };
```

用户也可以省去 new 操作符,甚至可以直接使用隐含的大小初始化数组。如下面的代码所示。

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

在 C# 中数组被视为一个对象,而且也是作为对象来处理的,不是作为可标识地址的字节流来处理的。数组能够实现自动垃圾回收,使用完数组之后,用户不用考虑数组元素的释放。所有的数组都是基于 C# 的 System.Array 类创建的,用户可以把它看成一个集合对象,可以使用它的 Length 属性或者 GetLength 方法,对数组的元素进行遍历。System.Array 本身还提供了复制、排序和检索方法。下面的代码即可实现数组的遍历输出。

```
int[] intArray = {2,4,5,6,7,8,9};  
for(int i = 0; i < intArray.Length; i++) {  
    System.Console.WriteLine(intArray[i]);  
}
```

C# 也有一个类似于 Visual Basic 的 foreach 操作符,通过它可以实现数组的遍历。