普通高等教育“十三五”规划教材

（计算机专业群）

基于C#的可视化编程基础

主　编　张蕾蕾　黄　健
副主编　和智横　梁文博　董　咪　贾明珠

中国水利水电出版社
www.waterpub.com.cn

普通高等教育“十三五”规划教材（计算机专业群）

基于C#的可视化编程基础

主　编　张蕾蕾　黄　健

副主编　和智横　梁文博　董　咪　贾明珠

中国水利水电出版社
www.waterpub.com.cn

·北京·

内 容 提 要

C#是目前最为流行的程序设计语言之一。本书以 Microsoft Visual Studio 2013 为平台，以培养高等工程技术应用型人才为目标，以工程应用为背景，深入浅出地讲解了 C#的可视化开发基础知识；在内容取材上，力求简明精练，以够用为度；在讲述方法上，既注重基本内容、基本方法的介绍，力求通俗易懂，又强调理论与实际融会贯通，通过大量的实用例程突出本书的实用性。

全书共分 7 章，其中第 1 章至第 4 章介绍 C#基本语法、基本数据类型、循环控制语句、类与对象、集合、命名空间、Windows 窗体、菜单和菜单组件、Windows 窗体的美化、WinForm 应用程序常用控件、Windows 应用程序高级控件、容器、对话框设计、界面布局、第三方组件库；第 5 章和第 6 章介绍 SQLite 开发基础和网络编程基础；第 7 章是一个综合范例——餐厅管理系统的设计。所有知识点均结合具体实例进行介绍，涉及的程序代码都给出了详细的注释，通过实例与代码设计有机结合使读者轻松领会 C#应用程序开发的精髓，快速提高开发技能。

本书可作为高等学校计算机或工科非计算机专业的程序设计教材，也可供从事软件开发的爱好者参考。

本书配有免费电子教案，读者可以从中国水利水电出版社网站以及万水书苑下载，网址为：http://www.waterpub.com.cn/softdown/或 http://www.wsbookshow.com。

图书在版编目（CIP）数据

基于C#的可视化编程基础 / 张蕾蕾，黄健主编. -- 北京 : 中国水利水电出版社，2019.3
普通高等教育“十三五”规划教材. 计算机专业群
ISBN 978-7-5170-7533-2

Ⅰ. ①基… Ⅱ. ①张… ②黄… Ⅲ. ①C语言－程序设计－高等学校－教材 Ⅳ. ①TP312.8

中国版本图书馆CIP数据核字(2019)第051196号

策划编辑：石永峰　责任编辑：张玉玲　加工编辑：吕　慧　封面设计：李　佳

书　名	普通高等教育“十三五”规划教材（计算机专业群） 基于 C#的可视化编程基础 JIYU C# DE KESHIHUA BIANCHENG JICHU
作　者	主　编　张蕾蕾　黄　健 副主编　和智横　梁文博　董　咪　贾明珠
出版发行	中国水利水电出版社 （北京市海淀区玉渊潭南路 1 号 D 座　100038） 网址：www.waterpub.com.cn E-mail：mchannel@263.net（万水） sales@waterpub.com.cn 电话：（010）68367658（营销中心）、82562819（万水）
经　售	全国各地新华书店和相关出版物销售网点
排　版	北京万水电子信息有限公司
印　刷	三河航远印刷有限公司
规　格	184mm×260mm　16 开本　13.25 印张　324 千字
版　次	2019 年 3 月第 1 版　2019 年 3 月第 1 次印刷
印　数	0001—3000 册
定　价	35.00 元

前　言

Visual Studio 2013 是微软公司推出的新一代可视化开发平台。作为创建企业规模的 Web 应用程序和高性能的桌面应用程序所推出的.NET 框架构建，该平台在很多方面进行了很大改进。C#是 Visual Studio 2013 开发平台上最主流的开发语言。

C#语法结构简单，在很多方面都与 C 和 C++极其相似。C#是一种完全面向对象的程序设计语言，具备面向对象的封装、继承、多态等基本特征。随着在实际中的广泛应用，C#引起了广大计算机应用开发者的学习兴趣，兴起了学习和使用 C#的热潮。随着组件对象的不断进步和 Internet 应用的不断普及，高校有必要将 C#作为程序设计的入门语言。本书正是在这一背景下编写的，适合各高校选作程序设计的教材。

本书是作者结合多年教学经验并依据应用实践编写而成的，全面系统地介绍了 C#程序设计的基础知识。依照读者的认知规律，全书分为 7 章。其中第 1 章至第 4 章介绍 C#基本语法、基本数据类型、循环控制语句、类与对象、集合、命名空间、Windows 窗体、菜单和菜单组件、Windows 窗体的美化、WinForm 应用程序常用控件、Windows 应用程序高级控件、容器、对话框设计、界面布局、第三方组件；第 5 章介绍 SQLite 开发基础，掌握此数据库用法，基本上可以操作当前主流关系型数据库；第 6 章介绍网络编程基础；第 7 章是一个综合范例——餐厅管理系统的设计，读者可全面学习 C#的可视化应用设计。

本书概念清晰、层次分明、逻辑性强，内容选材上由浅入深、循序渐进、实例丰富经典，而且每章后均配有丰富的习题，供读者练习与自测。

本书的重点是 C#程序设计基础方法，并对网络与数据库开发及实现提供必要的基本知识。本书是为计算机专业的学生和从事计算机软件开发的技术人员编写的，也适合非计算机专业学生使用，尤其适合 C#初学者作为入门教材使用。

本书由西安邮电大学理学院张蕾蕾任第一主编并统稿，西安科技大学通信学院黄健任第二主编并负责内容审核和部分章节编写，西安科技大学通信学院和智横、梁文博、董咪、贾明珠任副主编。编写分工如下：张蕾蕾编写第 1 章至第 4 章，梁文博、和智横、董咪、贾明珠编写第 5 章和第 6 章，黄健编写第 7 章。

在编写本书过程中，我们得到了许多专家和同仁的热情帮助与大力支持，在此一并表示感谢。

由于作者水平有限，书中疏漏甚至错误之处在所难免，恳请读者批评指正。

编　者

2019 年 1 月

目　　录

第 1 章　C#基础

1.1　基本语法

1.1.1　C#概述

C#是微软公司推出的一种语法简洁、类型安全的面向对象编程语言，开发人员可以通过它编写在.NET Framework 上运行的各种安全可靠的应用程序。近几年 C#的使用人数呈现上升趋势，这也说明了 C#语言的简单、现代、面向对象和类型安全等特点正在被更多人所认同。C#具有以下突出特点：

- 语法简洁。不允许直接操作内存，去掉了指针操作。
- 彻底的面向对象设计。C#具有面向对象语言所应有的一切特性：封装、继承和多态。
- 与 Web 紧密结合。C#支持绝大多数的 Web 标准，如 HTML、XML、SOAP 等。
- 强大的安全机制。可以消除软件开发中常见的错误，.NET 提供的垃圾回收器能够帮助开发者有效地管理内存资源。
- 兼容性。因为 C#遵循.NET 的公共语言规范，从而能够保证与其他语言开发的组件兼容。
- 灵活的版本处理技术。因为 C#语言本身内置了版本控制功能，因此开发人员能更加容易地开发和维护应用程序。
- 完善的错误、异常处理机制。C#提供了完善的错误异常处理机制，使程序在交付使用时能够更加健壮。

1.1.2　编写第一个 C#程序

我们使用 Visual Studio 和 C#语言来编写第一个程序，程序在控制台上显示字符串“Hello World!”。

【例 1.1】创建一个控制台应用程序，使用 WriteLine 方法输出“Hello World!”字符串。

代码如下：

```
static void Main(string[] args)                    //在 main 方法下编写代码输出数据
{
    Console.WriteLine("Hello World!");             //输出“Hello World!”
    Console.ReadLine();
}
```

程序运行结果如下：

```
Hello World!
```

1.2 基本数据类型

1.2.1 值类型

值类型变量直接存储其数据值，主要有整数类型、浮点类型、布尔类型等。值类型变量在堆栈中进行分配，因此效率很高。使用值类型的主要目的是提高性能。值类型具有以下特征：

- 值类型变量都存储在堆栈中。
- 访问值类型变量时，一般都是直接访问其实例。
- 每个值类型变量都有自己的数据副本，因此对一个值类型变量的操作不会影响其变量。
- 复制值类型变量时，复制的是变量的值，而不是变量的地址。
- 值类型变量不能为null，必须具有一个确定的值。

下面详细介绍值类型中包含的几种数据类型。

1. 整数类型

整数类型代表一种没有小数点的整数数值，在C#中内置的整数类型如表1-1所示。

表1-1 C#内置的数据类型

类型	说明	范围
sbyte	8位有符号整数	-128～127
short	16位有符号整数	-32768～32767
int	32位有符号整数	-2147483648～2147483647
long	64位有符号整数	-9223372036854775808～9223372036854775807
byte	8位无符号整数	0～255
ushort	16位无符号整数	0～65535
uint	32位无符号整数	0～4294967295
ulong	64位无符号整数	0～18446744073709551615

【例1.2】创建一个控制台应用程序，在其中声明一个int类型的变量a并初始化为10、一个byte类型的变量b并初始化为24，最后输出。

代码如下：

```
static void Main(string[] args)
{
    int a = 10;                          //声明一个 int 类型的变量 a
    byte b = 24;                         //声明一个 byte 类型的变量 b
    Console.WriteLine("a={0}", a);       //输出 int 类型的变量 a
    Console.WriteLine("b={0}", b);       //输出 byte 类型的变量 b
    Console.ReadLine();
}
```

程序运行结果如下：

```
a=10
b=24
```

此时，如果将 byte 类型的变量 b 赋值为 266，重新编译程序就会出现错误提示。主要原因是 byte 类型的变量是 8 位无符号整数，它的范围为 0～255，266 已经超出了 byte 类型的范围，所以编译程序会出现错误提示。

2. 浮点类型

浮点类型变量主要用于处理含有小数的数值数据。浮点类型主要包含 float 和 double 两种数值类型。表 1-2 列出了这两种数值类型的描述信息。

表 1-2　浮点类型及描述

类型	说明	范围
float	精确到 7 位数	1.5×10^-45～3.4×10^38
double	精确到 15～16 位数	50×10^-324～1.7×10^308

【例 1.3】下面的代码就是将数值强制指定为 float 类型和 double 类型。

```
float myFlo = 1.23F;                //使用 F 强制指定为 float 类型
float myflo = 9.23f;                //使用 f 强制指定为 float 类型
double myDou = 11D;                 //使用 D 强制指定为 double 类型
double mydou = 92d;                 //使用 d 强制指定为 double 类型
```

3. 布尔类型

布尔类型主要是用来表示 true/false 值。一个布尔类型的变量，其值只能是 true 或者 false，不能将其他的值指定给布尔类型变量。布尔类型变量不能与其他类型进行转换。

1.2.2　引用类型

引用类型是构建 C#应用程序的主要对象类型数据。在应用程序执行过程中，预先定义的对象类型以 new 创建对象实例并存储在堆栈中。堆栈是一种由系统弹性配置的内存空间，没有特定大小及存活时间，因此可以弹性地运用于对象的访问。引用类型具有如下特征：

- 必须在托管堆中为引用类型变量分配内存。
- 必须使用关键字 new 来创建引用类型变量。
- 在托管堆中分配的每个对象都有与之相关联的附加成员，这些成员必须被初始化。
- 引用类型变量是由垃圾回收机制来管理的。
- 多个引用类型变量可以引用同一对象，这种情形下，对一个变量的操作会影响另一个变量所引用的同一对象。
- 引用类型被赋值前的值都是 null。

所有被称为“类”的都是引用类型，主要包括类、接口、数组和委托。下面通过一个实例来演示如何使用引用类型。

【例 1.4】创建一个控制台应用程序，在其中创建一个类 C，在该类中建立一个字段 value 并初始化为 0，然后在程序的其他位置通过 new 创建对该类的引用类型变量，最后输出。

代码如下：

```
class Programe
{
    class C
    {
        public int value = 0;
    }
    static void Main(string[] args)
    {
        int v1 = 0;                          //声明变量 v1 并初始化为 0
        int v2 = v1;                         //声明变量 v2 并将 v1 赋值给 v2
        v2 = 99;                             //重新将变量 v2 赋值为 99
        C r1 = new C();                      //使用 new 关键字创建引用对象
        C r2 = r1;                           //使 r1 等于 r2
        r2.value = 112;                      //设置变量 r2 的 value 值
        Console.WriteLine("value:{0},{1}", v1, v2);        //输出变量 v1 和 v2
        //输出引用类型对象的 value 值
        Console.WriteLine("refs:{0},{1}", r1.value, r2.value);
        Console.ReadLine();
    }
}
```

程序运行结果如下：

```
value:0,99
refs:112,112
```

1.2.3 枚举类型

枚举类型是一种独特的值类型，用于声明一组具有相同性质的常量。编写与日期相关的应用程序时，经常需要使用年、月、日、星期等数据，可以将这些数据组织成多个不同名称的枚举类型。使用枚举类型可以增加程序的可读性和可维护性。同时，枚举类型可以避免类型错误。在定义枚举类型时，如果不对其进行赋值，默认情况下，第一个枚举数的值为 0，后面每一个枚举数的值依次递增 1。

在 C#中使用关键字 enum 来声明枚举，形式如下：

```
enum 枚举名
{
    list1 = value1,
    list2 = value2,
    list3 = value3,
    ...
    listN = valueN,
}
```

其中，大括号中的内容为枚举值列表，每个枚举值均对应一个枚举值名称，value1～valueN 为整数数据类型，list1～listN 为枚举值的标识名称。

1.3 循环与跳转语句

1.3.1 循环语句

循环语句主要用于重复执行嵌入语句。在 C#中，常见的循环语句有 while 语句、do...while 语句、for 语句和 foreach 语句。

1. while 语句

while 语句用于根据条件值执行一条语句零次或多次，当每次 while 语句中的代码执行完毕时，将重新查看是否符合条件值，若符合则再次执行相同的程序代码，否则跳出 while 语句，执行其他程序代码。while 语句的基本格式如下：

```
while(【布尔表达式】)
{
    【语句块】
}
```

while 语句的执行顺序如下：

（1）计算【布尔表达式】的值。

（2）如果【布尔表达式】的值为 true，程序执行【语句块】。执行完毕重新计算【布尔表达式】的值是否为 true。

（3）如果【布尔表达式】的值为 false，则控制将转移到 while 语句的结尾。

下面通过实例演示如何使用 while 语句。

【例 1.5】创建一个控制台应用程序，声明一个 int 类型的数组并初始化，然后通过 while 语句循环输出数组中的所有成员。

代码如下：

```
static void Main(string[] args)
{
    int[] num = new int[6]{1,2,3,4,5,6};          //声明一个 int 类型的数组并初始化
    int s = 0;                                    //声明一个 int 类型的变量 s 并初始化
    while(s<6)                                    //调用 while 语句，当 s 小于 6 时执行
    {
        Console.WriteLine("num[{0}]的值为{1}",s,num[s]);
        s++;                                      //s 自增 1
    }
    Console.ReadLine();
}
```

程序运行结果为：

```
num[0]的值为 1
num[1]的值为 2
num[2]的值为 3
num[3]的值为 4
num[4]的值为 5
num[5]的值为 6
```

在 while 语句的嵌入语句块中，break 语句可用于将控制转到 while 语句的结束点，而 continue 语句可用于将控制直接转到下一次循环。

2. do...while 语句

do...while 语句与 while 语句相似，它的判断条件在循环后。do...while 循环会在计算循环表达式之前执行一次，其基本形式如下：

```
do
{
    【语句块】
}while(【布尔表达式】);
```

do...while 语句的执行顺序如下：

（1）程序首先执行【语句块】。

（2）当程序到达【语句块】的结束点时，计算【布尔表达式】的值，如果【布尔表达式】的值是 true，程序跳到 do...while 语句的开头，否则，结束循环。

【例 1.6】创建一个控制台应用程序，声明一个 bool 类型的变量 term 并初始化为 false，再声明一个 int 类型的数组并初始化，然后调用 do...while 语句，通过 for 语句循环输出数组中的值。

代码如下：

```
static void Main(string[] args)
{
    bool term = false;                          //声明一个 bool 类型的变量 term 并初始化为 false
    int [] myArray = new int[5]{0,1,2,3,4};     //声明一个 int 类型的数组并初始化
    do
    {
        for(int i=0;i<myArray.Length;i++)       //调用 for 语句输出数组中的所有数据
        {
            Console.WriteLine(myArray[i]);      //输出数组中的数据
        }
    }while(term);                               //设置 do...while 语句的条件
    Console.ReadLine();
}
```

程序运行结果为：

```
0
1
2
3
4
```

从代码中可以看出，bool 类型变量 term 被初始化为 false，但是 do...while 依然执行了一次 for 循环，将数组中的值输出。由此可以说明，do...while 语句至少要执行代码一次，无论最后的条件是 true 还是 false。

3. for 语句

for 语句用于计算一个初始化序列，然后当某个条件为 true 时，重复执行嵌套语句并计算一个迭代表达式序列；如果为 false，则终止循环，退出 for 循环。for 循环语句的基本形式如下：

```
for(【初始化表达式】;【条件表达式】;【迭代表达式】)
{
    【语句块】
}
```

【初始化表达式】由一个局部变量或者一个由逗号分隔的表达式列表组成。【初始化表达式】声明的局部变量的作用域，是从变量的声明开始，一直到嵌入语句的结尾。【条件表达式】必须是一个布尔表达式。【迭代表达式】必须包含一个用逗号分隔的表达式列表。

for 语句执行的顺序如下：

（1）如果有【初始表达式】，则按变量初始值设定项或语句表达式的书写顺序指定它们，此步骤只执行一次。

（2）如果存在【条件表达式】，则计算它。

（3）如果不存在【条件表达式】，则程序将转移到嵌入语句。如果程序到达了嵌入语句的结束点，按顺序计算 for 迭代表达式，然后从上一个步骤中 for 条件的计算开始，执行另一次迭代。

for 循环是循环语句中最常用的一种，它实现了一种规定次数、逐次反复的功能，但是由于代码编写方式不同，所以也可能实现其他循环的功能。

【例 1.7】创建一个控制台应用程序，首先声明一个 int 类型的数组，然后初始化数组，最后使用 for 循环语句遍历数组，并将数组中的值输出。

代码如下：

```
static void Main(string[] args)
{
    int[] myArray = new int[5]{0,1,2,3,4};
    for(int i=0;i<myArray.Length;i++)
    {
        Console.WriteLine("myArray[{0}]的值是：{1}",i,myArray[i]);
    }
    Console.ReadLine();
}
```

程序运行结果为：

```
myArray[0]的值是：0
myArray[1]的值是：1
myArray[2]的值是：2
myArray[3]的值是：3
myArray[4]的值是：4
```

4. foreach 语句

foreach 语句用于枚举一个集合的元素，并对该集合中的每一个元素执行一次嵌入语句，但是 foreach 语句不能用于更改集合内容，以避免产生不可预知的错误。foreach 语句的基本形式如下：

```
foreach(【类型】【迭代变量名】 in 【集合类型表达式】)
{
    【语句块】
}
```

其中，【类型】和【迭代变量名】用于声明迭代变量，迭代变量相当于一个范围覆盖整个语句块的局部变量。在 foreach 语句执行期间，迭代变量表示当前正在为其执行迭代的集合元素。【集合类型表达式】必须有一个从该集合的元素类型到迭代变量的类型的显式转换，如果【集合类型表达式】的值为 null，则会出现异常。

【例 1.8】创建一个控制台应用程序，实例化一个 ArrayList 数组，向数组中添加值，然后通过使用 foreach 语句遍历整个数组，并输出数组中的值。

代码如下：

```
static void Main(string[] args)
{
    ArrayList list = new ArrayList();                    //实例化 ArrayList 类
    list.Add("Hello");                                   //使用 Add 方法向对象中添加数据
    list.Add("World");                                   //使用 Add 方法向对象中添加数据
    foreach(string Words in list)
    {
        Console.WriteLine(Words);                        //输出 ArrayList 对象中的所有数据
    }
    Console.ReadLine();
}
```

程序运行结果如下：

```
Hello
World
```

1.3.2 跳转语句

跳转语句主要用于无条件的转移控制。跳转语句会将控制跳转到某个位置，这个位置就称为跳转语句的目标。如果跳转语句出现在一个语句块内，而跳转语句的目标却在该语句块之外，则称该跳转语句退出该语句块。跳转语句主要有 break 语句、continue 语句、goto 语句和 return 语句。

1. break 语句

break 语句只能应用在 switch、while、do...while、for、foreach 语句中，break 语句包含在这几种语句中，否则会出现编译错误。当多条 switch、while、do...while、for、foreach 语句相互嵌套时，break 语句只应用于最里层的语句，如果要穿越多个嵌套层，则必须使用 goto 语句。

【例 1.9】创建一个控制台应用程序，声明一个 int 类型的变量 i，用于获取当前日期的返回值，然后通过使用 switch 语句根据变量 i 输出当前日期是星期几。

代码如下：

```
static void Main(string[] args)
{
    int i = Convert.ToInt32(DateTime.Today.DayOfWeek);    //获取当前日期的数值
    switch(i)
    {                                                     //调用 switch 语句
        case 1:Console.WriteLine("今天是星期一");break;
        case 2:Console.WriteLine("今天是星期二");break;
        case 3:Console.WriteLine("今天是星期三");break;
```

```
            case 4:Console.WriteLine("今天是星期四");break;
            case 5:Console.WriteLine("今天是星期五");break;
            case 6:Console.WriteLine("今天是星期六");break;
            case 7:Console.WriteLine("今天是星期日");break;
        }
        Console.ReadLine();
    }
```

2. continue 语句

continue 语句只能应用在 while、do...while、for、foreach 语句中，用来忽略循环语句块内位于它后面的代码而直接开始一次新的循环。当多个 while、do...while、for、foreach 语句相互嵌套时，continue 语句使直接包含它的循环语句开始一次新的循环。

【例 1.10】创建一个控制台应用程序，使用两个 for 语句进行嵌套循环。在内层 for 语句中，使用 continue 语句，实现当 int 类型变量 j 为偶数时不输出，重新开始内层的 for 循环，只输出 0～20 内的所有奇数。

代码如下：

```
static void Main(string[] args)
{
    for(int i=0;i<2;i++)
    {                                                    //调用 for 循环
        Console.WriteLine("\n 第{0}次循环：",i);          //输出提示第几次循环
        for(int j=0;j<20;j++)
        {                                                //调用 for 循环
            if(j%2 == 0)                                 //调用 if 语句判断 j 是否为偶数
            continue;                                    //若为偶数，继续下一次循环
            Console.Write(j+" ");
        }
        Console.WriteLine();
    }
    Console.ReadLine();
}
```

程序运行结果为：

```
第 0 次循环：1 3 5 7 9 11 13 15 17 19
第 1 次循环：1 3 5 7 9 11 13 15 17 19
```

从程序的运行结果可以看出，当 int 类型的变量 j 为偶数时，使用 continue 语句，忽略它后面的代码而重新执行内层的 for 循环。这期间并没有影响外部的 for 循环，程序依然执行。

3. goto 语句

goto 语句用于将控制转移到由标签标记的语句。goto 语句可以被应用在 switch 语句中的 case 标签、default 标签，以及标记语句所声明的标签。goto 语句的 3 种形式如下：

```
goto【标签】
goto case【参数表达式】
goto default
```

goto【标签】语句的目标是给定标签的标记语句；goto case【参数表达式】语句的目标是它所在的 switch 语句中的某个语句列表，此列表包含一个具有给定常数值的 case 标签；goto

default 语句的目标是它所在的 switch 语句中的 default 标签。

【例 1.11】创建一个控制台应用程序，通过 goto 语句实现程序跳转到指定语句。

```
static void Main(string[] args)
{
    Console.WriteLine("请输入要查找的文字：");          //输出提示信息
    string inputStr = Console.ReadLine();              //获取输入值
    string[] myStr = new string[2];                    //创建数组
    myStr[0] = "Hello";                                //向数组中添加元素
    myStr[1] = "World";
    for(int i=0;i<myStr.Length;i++)
    {
        if(myStr[i].Equals(inputStr))
        {                                              //判断是否存在输入的字符串
            goto Found;                                //调用 goto 语句跳转到 Found
        }
    }
    Console.WriteLine("您查找的{0}不存在！",inputStr);  //输出信息
    goto Finish;                                       //调用 goto 语句跳转到 Finish
    Found:
    Console.WriteLine("您查找的{0}存在！",inputStr);
    Finish:
    Console.WriteLine("查找完毕！");                    //输出信息，提示查找完毕
    Console.ReadLine();
}
```

4. return 语句

return 语句用于退出类的方法，是控制返回方法的调用者。如果方法有返回类型，return 语句必须返回这个类型的值；如果方法没有返回类型，应使用没有表达式的 return 语句。

【例 1.12】创建一个控制台应用程序，建立一个返回类型为 string 类型的方法，利用 return 语句返回一个 string 类型的值，然后在 main 方法中调用这个自定义的方法并输出这个方法的返回值。

代码如下：

```
static string MyStr(string str)                    //创建一个 string 类型的方法
{
    string outStr = "您输入的数据是："+str;         //声明一个字符串变量并为其赋值
    return outStr;                                 //使用 return 语句返回字符串变量
}
static void Main(string[] args)
{
    Console.WriteLine("请您输入内容：");            //输出提示信息
    string inputStr = Console.ReadLine();          //获取输入信息
    Console.WriteLine(MyStr(inputStr));            //调用 MyStr 方法并将结果显示出来
    Console.ReadLine();
}
```

程序运行结果为：

```
您输入的数据是：Hello World
```

1.4　类与对象

1.4.1　类

1. 类的概念

类是对象概念在面向对象编程语言中的反映，是相同对象的集合。类描述了一系列在概念上有相同含义的对象，并为这些对象统一定义了编程语言上的属性和方法。如水果就可以看成一个类，苹果、葡萄都是该类的子类。苹果的生产地、名称、价格相当于该类的属性，苹果的种植方法相当于类的方法。简而言之，类是 C#中功能最为强大的数据类型，类也定义了数据类型的数据和行为。

2. 类的声明

C#中，类是使用关键字 class 来声明的，语法如下：

```
类修饰符 class 类名
{
}
```

【例 1.13】以汽车为例声明一个类。

代码如下：

```
public class Car
{
    public int number;          //编号
    public string color;        //颜色
    public string brand;        //厂家
}
```

public 是类的修饰符，下面介绍几种常见的修饰符。

- public：不限制对该类的访问。
- protected：只能从其所在类和所在类的子类进行访问。
- internal：只有其所在类才能访问。
- private：只有.NET 中的应用程序或库才能访问。
- abstract：抽象类，不允许建立类的实例。
- sealed：密封类，不允许被继承。

3. 对象的声明和实例化

对象是具有数据、行为和标识的编程结构，是面向对象应用程序的一个组成部分。这个组成部分封装了部分应用程序。这部分程序可以是一个过程、一些数据或一些更抽象的实体。

对象包含变量成员和方法类型，它所包含的变量组成了存储在对象中的数据，而其包含的方法可以访问对象的变量，略为复杂的对象可能不包含任何数据，而只包含方法，并使用方法表示一个过程。

C#中的对象是把类实例化，这表示创建一个类的实例，“类的实例”和对象表示相同的含