# 类的编程

# 内容简介

类是对一系列相同性质对象的抽象，是对象共同特征的描述。类也是面向对象的基本单位，是一种包含数据成员、方法成员和嵌套类型的数据结构

在本章中将讲述如何定义以及使用类字段、类属性和类方法。

# 本章学习要点

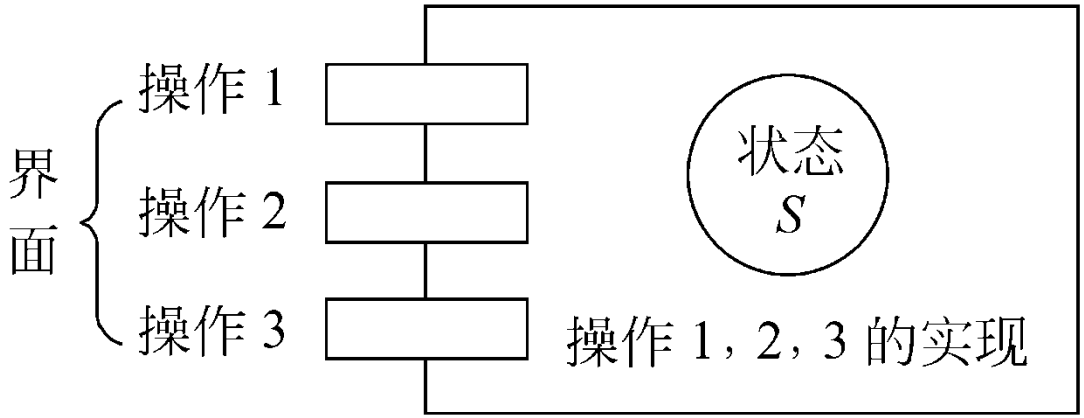
- 了解面向对象内容
- 理解并掌握类字段的使用
- 熟练掌握属性使用方法
- 熟练掌握静态方法
- 了解析构方法
- 熟练掌握方法的重载

# 6.1 面向对象简介

客观世界中的实体通常都既具有静态的属性，又具有动态的行为。面向对象方法学中的对象是由描述该对象属性的数据以及可以对这些数据施加的所有操作封装在一起构成的统一体。

对象可以作的操作表示它的动态行为，在面向对象分析和面向对象设计中，通常把对象的操作称为服务或方法。

## 对象的形象表示



## 对象的特点

(1) 以数据为中心。操作围绕对其数据所需要做的处理来设置，不设置与这些数据无关的操作，而且操作的结果往往与当时所处的状态 ( 数据的值 ) 有关。

(2) 对象是主动的。它与传统的数据有本质不同，不是被动地等待对它进行处理，相反，它是进行处理的主体。为了完成某个操作，不能从外部直接加工它的私有数据，而是必须通过它的公有接口向对象发消息，请求它执行它的某个操作，处理它的私有数据。

(3) 实现了数据封装。

对象好像是一只黑盒子，它的私有数据完全被封装在盒子内部，对外是隐藏的、不可见的。

为了使用对象内部的私有数据，只需知道数据的取值范围（值域）和可以对该数据施加的操作，根本无须知道数据的具体结构以及实现操作的算法。这也就是抽象数据类型的概念。

(4) 本质上具有并行性。

对象是描述其内部状态的数据及可以对这些数据施加的全部操作的集合。不同对象各自独立地处理自身的数据，彼此通过发消息传递信息完成通信。因此，本质上具有并行工作的属性。

## 类 (class)

现实世界中存在的客观事物有些是彼此相似的，例如，张三、李四、王五……虽说每个人职业、性格、爱好、特长等等各有不同，但是，他们的基本特征是相似的，都是黄皮肤、黑头发、黑眼睛，于是人们把他们统称为“中国人”。

人类习惯于把有相似特征的事物归为一类，分类是人类认识客观世界的基本方法。

## 实例 (instance)

实例就是由某个特定的类所描述的一个具体的对象。类在现实世界中并不能真正存在。

类是建立对象时使用的“样板”，按照这个样板所建立的一个个具体的对象，就是类的实际例子，通常称为实例。

当使用“对象”这个术语时，既可以指一个具体的对象，也可以泛指一般的对象，但是，当使用“实例”这个术语时，必然是指一个具体的对象。

## 消息 (message)

消息就是要求某个对象执行在定义它的那个类中所定义的某个操作的规格说明。通常，一个消息由下述 3 部分组成：

- 接收消息的对象
- 消息名
- 变元（零个或多个）

## 方法 (method)

方法就是对象所能执行的操作，也就是类中所定义的服务。方法描述了对象执行操作的算法，响应消息的方法。

**属性 (attribute)**

属性就是类中所定义的数据，它是对客观世界实体所具有的性质的抽象。

类的每个实例都有自己特有的属性值。

**封装 (encapsulation)**

从字面上理解，所谓封装就是把某个事物包起来，使外界不知道该事物的具体内容。

**继承 (inheritance)**

广义地说，继承是指能够直接获得已有的性质和特征，而不必重复定义它们。在面向对象的软件技术中，继承是指：子类自动地共享基类中定义的数据和方法。

继承具有传递性。因此，属于某类的对象除了具有该类所描述的性质外，还具有该类上层全部基类描述的一切性质。

**多态性 (polymorphism)**

在类等级的不同层次中可以共享 ( 公用 ) 一个行为 ( 方法 ) 的名字，然而不同层次中的每个类却各自按自己的需要来实现这个行为。当对象接收到发送给它的消息时，根据该对象所属于的类动态选用在该类中定义的实现算法。

【例】在一般类“几何图形”中定义了一个服务“绘图”，但并不确定执行时到底画一个什么样的图形。

特殊类“椭圆”和“矩形”都继承了“几何图形”类的“绘图”服务，但功能却不同。

当系统的其余部分请求画出任何一种图形时，消息中给出的服务名同样都是“绘图”，但“椭圆”和“矩形”类的对象接收到这个消息时却各自执行不同的绘图方法。

## 重载 (overloading)

有两种重载：

- 运算符重载是指同一个运算符可以施加于不同类型的操作数上面。在编译时根据被操作数的类型，决定使用该算符的哪种语义。

  2+3 ， 4.1+5.5

- 函数重载是指在同一作用域内的若干个参数特征不同的函数可以使用相同的函数名字。在编译时根据函数变元的个数和类型，决定到底使用函数的哪个实现代码。

  Compare( str1,str2)

  Compare( str1,str2,boolean)

重载进一步提高了面向对象系统的灵活性和可读性。

# 6.2 类

对象的抽象是类，类的具体化就是对象，即类的实例是对象。

从定义上来说类是一个数据结构，类包含有数据成员和功能成员。

# 6.2.1 类声明语法

在 C# 中类是使用 class 关键字定义，定义类的语法格式如下所示：

**修饰符 class 类名 [ 基类或者接口 ]**
**{ 类成员 }**

类的修饰符可以是 public 、 sealed 、 abstract 、 protected 、 internal 以及 private 等。基类和接口属于可选部分。

**例**：名为 Test 的类，包含字段、方法、属性以及构造方法。

```
public class Test
   {
      public string username;   // 字段
      public Test()    // 构造函数
      {    Console.Write(“ 默认构造函数 ”);
           username = " 无名氏 ";              }
      public void print() // 方法
      {   Console.Write(" 类中的方法 ");
}
      public string validName     // 属性
      {   get { return username; }
          set {
             if( value!=“000") username = value;
          }
    }
```

```
class Program
   {
      static void Main(string[] args)
      {
         Test t1 = new Test();
         Test t2 = new Test();
         t1.print();
         t1.validName = " 张三 ";
         t2.validName = "000";
         Console.WriteLine(t1.username +
                                t2.username);
         Console.Read();
    }
}
```

# 6.2.2 类字段

在 C# 中类的成员变量也可以被称为字段。类变量与其他变量的定义方式一样，不同的是类变量在类内部定义。声明类字段语法格式如下所示：

**修饰符　字段类型　字段名**

修饰符可以是下列任意一种：

- private　只能在本类中访问该对象字段
- public　可以在任何地方访问该对象字段
- static　该修饰符将一个字段成员指定为静态成员。如果一个字段成员是静态成员，则它将同常量成员一样，属于类本身。如果外界访问该对象字段则需使用 " 类名 . 成员名 " 方式访问
- read only　该字段成员的值只能读，不能写。除了赋予初始值外，在程序的任何一个部分将无法更改这个字段成员的值
- const　将字段声明为常量类型，和 read only 声明的字段相似
- protected　能在子类或者基类中访问该对象字段
- internal　能在本项目中访问该对象字段

```
class student
 {
      public  string stuname;
      readonly  string  stusex=" 男 ";
      const  int  stuscore=80;
      private  double  stuheight;
 }
```

# 6.2.3 属性

属性可以说是字段的延伸。

属性有两个核心的代码块分别是 get 访问器和 set 访问器：

◆设置属性的值时会执行 set 代码块

◆取属性的值时访问 get 代码块

属性和字段的访问方式是相同的。与字段不同的是属性不作为变量类分类，所以不能将属性作为 ref 参数和 out 参数传递。

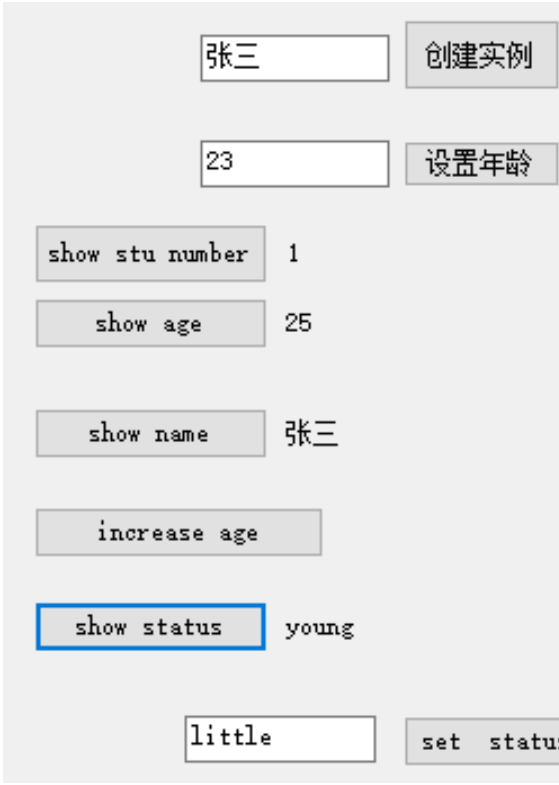
下面我们定义一个 sea 类，在该类中定义一个 Water 属性，示例代码如下所示：

```
public class sea
 {
    private string water;
    public string Water
    {
       get
       {
          return water;
       }
       set
       {
          water = value;
       }
    }
 }
```

```
public class Student
  {  public static int num = 0; // 计数器
     public int age;
     public String name;
     public Student(String n)   // 构造函数
      {  name = n;
         age = 25;
         num++;
      }
     public void  inc() // 方法
      {   age++;   }
     public string status    // 属性
      { get
        {if (age>20 & age<30) return
"young";
            else if (age >= 30) return "old";
            else return "little";
        }
        set
        {   if (value == "little") age = 15;
            if (value == "young") age = 25;
            if (value == "old") age = 75;
        }
     }
  }
```

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    public void Form1_Load(object sender, EventArgs e)
}

    Student  s;
    private void button1_Click(object sender, EventArgs e)
     {    s = new Student(textBox1.Text);       }
    private void button3_Click(object sender, EventArgs e)
     {   label2.Text = Convert.ToString( s.age);       }
    private void button4_Click(object sender, EventArgs e)
     {   label3.Text = s.name;     }
    private void button5_Click(object sender, EventArgs e)
     {   s.age++;      }
    private void button6_Click(object sender, EventArgs e)
     {   label4.Text = s.status;       }
    private void button2_Click(object sender, EventArgs e)
     {   s.age = Convert.ToInt32(textBox2.Text); }
    private void button7_Click(object sender, EventArgs e)
     {   s.status = textBox3.Text;      }
}
```



# 6.3 方法

C# 中类的方法分为：无参方法、有参方法；静态方法和非静态方法（实例方法）。

- 6.3.1 无参方法
- 6.3.2 有参方法
- 6.3.3 重载方法
- 6.3.4 Main() 方法

# 6.3.1 无参方法

无参方法也即是没有传递任何参数的方法，在 C# 中无参方法的定义包含有修饰符、返回类型、方法名和方法体，无参方法语法格式如下所示：

```
修饰符 返回类型 方法名 () { 方法体 }
```

修饰符可以是 public 、 private 、 internal 等，返回类型有两种：一种有返回值，另一种无返回值。如有返回值则方法体中必须使用 return 。

**例：** 无参方法返回一个 int 类型的值，代码如下：

```
public int getint()
{
    return 12;
}
```

如果没有返回值则返回类型为 void ：

```
public void print()
{
   Console.Write(" 该方法没有返回值 ");
}
```

# 6.3.2 有参方法

**修饰符　返回类型　方法名 ( 参数 1 ，参数 2 ，参数 3.... 参数 n)　{ 方法体 }**

值。参数之间用逗号隔开。调用方法时，数据类型、顺序、数量要完全一致。

**例**：判断传递参数之和是奇数还是偶数，在主程序中调用该方法。

```
public partial class Form2 : Form
   {
      public Form2()
       {    InitializeComponent();     }
      private void button1_Click(object sender, EventArgs e)
       {   Judge  j = new  Judge();
          textBox1.Text = j.result  ( Convert.ToInt32(textBox1.Text),
                                      Convert.ToInt32(textBox2.Text));
   }  }
public class Judge
   {   public Judge()  { }   // 构造函数
      public String result(int x, int y)
      {   int z = x + y;
         return (z % 2 == 0) ? "even" : "odd";
   }  }
```

# 6.3.3 重载方法

两个或者两个以上具有相同方法名称、不同参数个数、不同参数类型的方法称为重载。
编译器会根据使用的参数的数量、类型和顺序决定调用哪个方法。
**例**：一个通过重载计算矩形面积的方法。

```
public class More
{  public int getInt { get; set; }
   public float getDouble { get; set; }
   public int getarea(int height,int width)
   {  getInt = height * width;
      return getInt;
   }
   public float getarea(float height1, float width1)
   {  getDouble = height1 * width1;
      return getDouble;
   }
}
static void Main(string[] args)
{  More m = new More();
   int area = m.getarea(5, 6);
   Console.Write(" 矩形的面积： "+area+"\n");
   float area1 = m.getarea((float)5.6,(float)7.8);
   Console.Write(" 矩形的面积： "+area1+"\n");
}
```

# 6.3.4 Main() 方法

Main() 方法是程序的入口点，将在那里创建对象和调用其他方法，程序控制在该方法中开始和结束。该方法在类或结构的内部声明。 Main() 方法必须为静态方法，不应为公共方法。

Main() 方法有两种返回类型： void 和 int 。

声明 Main() 方法时可以使用参数，也可以没有参数，如果使用参数时必须使用 string[] 作为该方法的参数。

Main() 方法有以下几种形式：

- static void Main()
- static int Main()
- static void Main(string[] args)
- static int Main(string[] args)

一般情况下 Main() 方法没有修饰符，即使包含访问修饰符，也会在运行时忽略它们。 C# 的惯例是在 Main() 方法中忽略访问修饰符。

# 6.4 构造方法

**构造方法是特殊的方法：**

1.是最先调用的方法

2.构造方法的名称与类名相同

3.没有返回类型

```
public class MyClass
 {     public  MyClass(string x)
          {   Console.WriteLine(“A new instance: ”+x);     }
 }
```

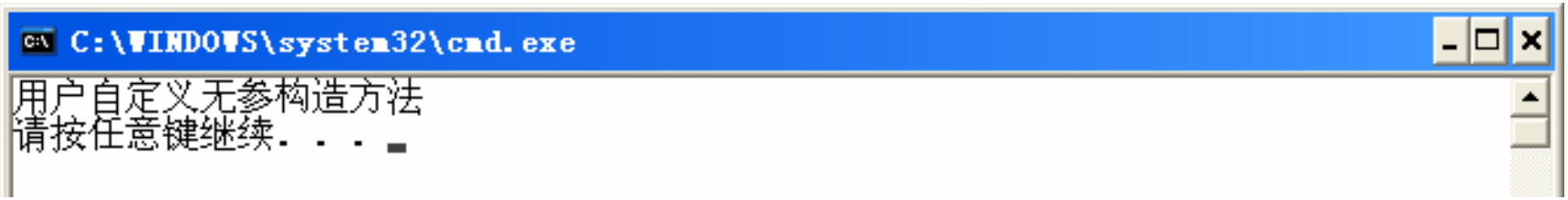
# 6.4.1 无参构造方法

声明无参构造方法语法格式如下所示：

```
class GS
{
  public GS()
  {
    Console.Write(" 用户自定义无参构造方法 ");
  }
}
```

在 Main() 方法中初始化 GS 类时会调用无参的构造方法，示例代码如下所示：

```
static void Main(string[] args)
{
  GS gs = new GS();
}
```



# 6.4.2 有参构造方法

**例**：定义两个有参构造方法：一个方法返回学生的成绩，另一个方法返回学生的个人信息。

```
class student
{
  public student (string stuno, int score)
  {  string  str = "";
     if (score >= 85 && score <= 100)
        str = “ 学生成绩 \n” + “ 学号 :” + stuno + “  ” + “ 成绩 ：” + score+“ 等级 ：”
+" 优 ";
     else if (score >= 70)
        str = “ 学生成绩 \n” + “ 学号 :” + stuno + “  ” + “ 成绩 ：” + score + “ 等级 ：” +
 " 良 ";
     else if (score >= 60)
        str = " 学生成绩 \n" + " 学号 :" + stuno + "   " + " 成绩 ： " + score + " 等级 ： " +
" 及格 ";
    else
        str = " 学生成绩 \n" + " 学号 :" + stuno + "   " + " 成绩 ： " + score + " 等级 ： " +
" 差 ";
     Console.WriteLine(str);
   }
   public student (string name, string sex, int age)
   {  Console.Write(" 个人信息 \n"+" 姓名 ："+name+"   "+" 性别 ："+sex+"   "+" 年
```

**定义构造方法的规则：**

1.每个类至少有一个构造方法。

2.如果没有显式定义任何构造方法，则编译系统会自动在后台产生一个无参数的默认构造方法，它不执行任何代码。

3.如果定义了带参数的构造方法，则编译器就不自动提供默认的构造方法。

**构造函数的重载**

可以为几个构造方法提供不同的参数表来重载构造方法。当发出“ new 类名（参数）”命令时，根据参数表决定采用哪个构造方法。

# 6.5 静态方法

静态方法与非静态方法非常相似，只是需要在返回类型前加上 static 关键字。静态方法不属于类的某一个具体的实例，而是属于类本身。对静态方法不需要创建一个类的实例，只需通过类名便可以调用静态方法。

```
public class stuinfo
{   public static string getStuName { get; set; }
    public static string getStuNo { get; set; }
    public static int getStuAge { get; set; }
    public stuinfo() { }
    public stuinfo(string name, string no, int age)
    {
        getStuName = name;     getStuNo = no;     getStuAge = age;
    }
    public static string  getInfo() {
        return “ 姓名：” + getStuName + “  编号：” + getStuNo + “  年龄：” + getStuAge
+ “ 性格： " + getStr(); }
    public static string  getStr() {
        return "\n 静若处子 动若疯兔 \n";}
}
```

Main() 方法的代码：

```
static void Main(string[] args)
{  stuinfo su = new stuinfo(" 张明 ", "s1001", 18);
   string info=stuinfo.getInfo();
   Console.Write(info);  }
```

# 6.6 析构方法

析构方法与构造方法相反，当对象脱离其作用域时（例如对象所在的方法已调用完毕），系统自动执行析构方法。声明析构方法时必须以析构方法的类命名：

```
~类名 () { 方法体 }
```

```
namespace ConsoleApplication2
{
   class Program
   { static void Main(string[] args)
      { MyClass m = new MyClass(66, 88);
        Console.WriteLine(" 在主程序中 ");
        Console.ReadLine();
      }
   class MyClass
   { public MyClass(int x, int y)
        { Console.WriteLine("x={0}  y={1}", x, y); }
     ~MyClass()
        { Console.Write(" 类被破坏时调用析构函数 : {0}", this);
          Console.ReadLine();
        }
}  }
```

```
x=66  y=88
主程序

类被破坏时调用析构函数: ConsoleApplication2.Program+MyClass
```