# C#基础学习笔记（八）

## 一、引子

### 1、面向过程-----> 面向对象

* 面向过程：面向的是完成这件事儿的过程，强调的是完成这件事儿的动作。

比如说：把大象塞进冰箱里（过程思维）

1、打开冰箱门  
2、把大象塞进去，亲下大象的屁股  
3、关闭冰箱门

张三瘦小 矮  
张三踩着小板凳打开冰箱门  
张三找李四帮忙把大象塞进冰箱里，孙全踩着板凳去亲。  
张三踩着板凳关闭冰箱门

李四 190cm 非常大力气  
1、李四自己就能打开冰箱门  
2、李四自己将大象塞进冰箱里，李四可以自己亲一下。  
3、李四自己关闭冰箱门

如果我们用面向过程的思想来解决这件事儿，当执行这件事的人的不同的时候，我们需要为每个不同的人量身定做解决事情的方法。

* 面向对象：找个对象帮你做事儿。

把大象塞进冰箱里（面向对象思维）  
我们把冰箱作为对象：   
1、冰箱门可以被打开  
2、大象可以被塞进冰箱里  
3、冰箱门可以被关闭

张三  
张三 1  
张三 2  
张三 3

李四  
李四 1  
李四 2  
李四 3

面向对象：意在写出一个通用的代码，屏蔽差异。

## 二、面向对象

### 1、世界是由什么组成的？

* 数学家——数字
* 摄影家.——图片
* 画家——颜色
* 程序员——对象
* 万物皆对象

### 2、如何找到对象呢？

关门  
面向过程：关门  
张三 一脚把门踹紧了  
李四 轻轻的把门带上了  
王五 门没关严，留了个尾巴

面向对象：关门（一般是被动的那个）  
门可以被关闭

* 试着描述张三和学生李四的特征和行为

属性：值

姓名：张三  
性别：男  
身高：180cm  
体重：70kg  
年龄：22岁

方法（行为）

性格活泼，乐观  
打篮球

姓名：李四  
性别：男  
身高：180cm  
体重：70KG  
年龄：23岁  
身体健康  
打羽毛球

我们在代码中描述一个对象，通过描述这个对象的属性和方法  
对象必须是看得见摸得着的

属性——对象具有的各种特征  
每个对象的每个属性都拥有特定值  
例如:李四和张三的年龄不一样.  
属性:姓名,性别,年龄  
属性的值:  
方法: 行为,动作.执行的操作.  
方法：  
张三的行为打篮球  
李四的行为打羽毛球

* 灯：属性和方法

属性：  
外形：长的  
亮度：500W  
颜色：白色  
牌子：XX  
方法：发光

* 电风扇：属性和方法

属性：  
外形：三个扇叶  
颜色：白色  
品牌：XX  
方法：转动，扇风

* **找出下列对象共性**

共性就是对象共同的属性和方法

张三(一个学生)\杨老师\邻居售货员张阿姨\李四的爸爸\李四的妈妈

* 有个共同属性是都有名字，性别等，共有的行为是吃喝。也有着各自的特点
* 注意都是人不是共同的属性，属性是对象具有的各种特征

门口停的奔驰S320汽车/老师开的QQ汽车/杨老师的解放牌大货车/牛老师开的宝马自行车

* 共同属性是都有品牌，共有的行为是都可以载人。也有特点
* 我们把这些具有相同属性和相同方法的对象进行进一步的封装，抽象出来**类**这个概念。
* 例如上面的第一个例子抽象出来人类，第二个是车类
* 类就是个模子，**确定了对象应该具有的属性和方法。**
* 类是对象的类型
* 对象是根据类创建出来的。

类就是一个盖大楼的图纸 对象就是盖出来的大楼。

### 3、类

语法：

|  |
| --- |
| [public] class 类名  {  字段;  属性;  方法;  } |

写好了一个类之后，我们需要创建这个类的对象，那么，我们管创建这个类的对象过程称之为类的实例化。  
使用关键字 new**.**

当我们创建好一个类的对象后，需要给这个对象的每个属性去赋值，我们管这个过程称之为对象的初始化。

类的实例化,用关键字new  
– 语法: 类 实例名 = new 类();  
  
类的成员的访问:  
 实例名.属性 实例名.方法名();

字段、方法、属性（后面讲）都可以叫做类的成员Member，它们都需要定义访问级别。访问级别的用处在于控制成员在哪些地方可以被访问，这样达到面向对象中“封装”的目的.

访问修饰符:public private (internal protected)

this:表示当前这个类的对象。  
类是不占内存的，而对象是占内存的。

### 4、属性

属性的作用就是保护字段、对字段的赋值和取值进行限定。

属性的本质就是两个方法，一个叫get()一个叫set()。  
既有get()也有set()我们诚之为可读可写属性。  
只有get()没有set()我们称之为只读属性  
没有get()只有set()我们称之为只写属性

|  |
| --- |
| class Program  {  static void Main(string[] args)  {  //系统的类，这里也是string类型的s变量  string s;  //自定义类，这里相当于声明了一个person类型的lisi变量  //这里运行lisi=null还没有开空间  Person lisi;  //创建Person类的对象  //这里就不是null了，已经开了空间了  Person zhangsan = new Person();  zhangsan.Name = "";  ////这样写-23仍然会显示-23，为了防止非法值的设定，需要属性的出现，并且要有判断  zhangsan.Age = -23;  //zhangsan.Age = 23;  //zhangsan.Gender = '男';  zhangsan.Gender = '同';  zhangsan.sports();  Console.ReadKey();  }  } |

|  |
| --- |
| public class Person  {  //字段  //待在家里去掉public  string \_name;  //属性  //这样就可以不用直接给字段进行赋值，而通过属性进行赋值  //属性这起到中间商的作用  public string Name  {  //在这个过程中属性不存值  //取值时候调用，当你输出属性的值得时候 会执行get的方法  get  {  //if (\_name != "张三")  //{  // return "张三";  //}  return \_name;  }  //赋值时候调用，当你给属性赋值的时候 会首先执行set方法  set  {  //if (value != "张三")  //{  // value = "张三";  //}  \_name = value;  }  }  int \_age;  public int Age  {  get { return \_age; }  //赋值是在program里边赋值的  //value实际就是传过来的值，所以\_age就是前面穿过来的值  set  {  //加入限制的条件  if (value < 0 || value > 100)  {  value = 0;  }  \_age = value;  }  }  char \_gender;  public char Gender  {  //取传过来的值的时候判断输入的东西是不是男或者女  //要不是默认返回的就是男，下面方法输出的就是男  get  {  if (\_gender != '男' && \_gender != '女')  {  return \_gender = '男';  }  return \_gender;  }  set { \_gender = value; }  }  //方法  //现在不加static  public void sports()  {  ////在这里就是输出属性，所以这里  //Console.WriteLine("我叫{0}，我今年{1}岁了，我是{2}生", this.\_name, this.\_age, this.\_gender);  //写成这样  Console.WriteLine("我叫{0}，我今年{1}岁了，我是{2}生", this.Name, this.Age, this.Gender);  }  } |

Field字段  
Method方法  
Property属性

字段就是要待在家里 属性才是在外边和其他东西打交道的。

字段用public修饰的问题…用private

属性的定义.get;set;

属性是为了保护与之相对应的字段的.保证对字段的读取和赋值符合要求.

属性可分为:读写、只读、只写。

允许外部访问的变量一定要声明为属性。

**属性和public字段的区别是什么？**

调用set方法为一个属性设值，然后用get方法读取出来的值一定是set进去的值吗？

属性可以对设值、取值的过程进行非法值控制，比如年龄禁止设值负数，而字段则不能进行这样的设置。虽然一般情况下get读取的值就是set设置的值，但是可以让get读取的值不是set设置的值的，极端的例子。

Public Age{get{return 100;}set{}}。

加分的补充回答：用reflector反编译可以看出，属性内部本质上就是set*\*\*\*、get*\*\*\*方法，

### 5、访问修饰符

public：公开的公共的，在哪都能访问。  
private：私有的，只能在当前类的内部进行访问，出了这个类就访问不到了。

**1、简述private、 protected、public、internal 修饰符的访问权限。**

private : 私有成员, 在类的内部才可以访问。

protected : 保护成员，该类内部和继承类中可以访问。

public : 公共成员，完全公开，没有访问限制。

internal: 当前程序集内可以访问。

### 6、静态和非静态的区别

1)、在非静态类中，既可以有实例成员，也可以有静态成员。

2)、在调用实例成员的时候，需要使用对象名.实例成员;

3)、在调用静态成员的时候，需要使用类名.静态成员名;

总结：静态成员必须使用类名去调用，而实例成员使用对象名调用。

**注意：**  
静态函数中，只能访问静态成员，不允许访问实例成员。  
实例函数中，既可以使用静态成员，也可以使用实例成员。  
静态类中只允许有静态成员，不允许出现实例成员。

**使用：**  
1)、如果你想要你的类当做一个"工具类"去使用，这个时候可以考虑将类写成静态的。  
2)、静态类在整个项目中资源共享。  
只有在程序全部结束之后，静态类才会释放资源。

堆 栈 静态存储区域

释放资源。GC Garbage Collection垃圾回收器

|  |
| --- |
| class Program  {  static void Main(string[] args)  {  //这里因为他们里边的实例成员要用对象去调用  //调用实例成员  Person p = new Person();  //实例方法  p.M1();  //静态方法  Person.M2();    //静态类无法创建对象，在调用静态成员的时候需要用类名调用，而不能使用对象名去调用  //所以这个对象对于静态类没有什么意义  //Student s = new Student();  Console.ReadKey();  }  } |

|  |
| --- |
| class Person  {  private static string \_name;  public static string Name  {  get  {  //静态要加Person.但是这里提示可以省去  return Person.\_name;  }  set  {  Person.\_name = value;  }  }  private char \_gender;  public char Gender  {  get  {  return \_gender;  }  set  {  \_gender = value;  }  }  public void M1()  {  Console.WriteLine("我是非静态的方法");  }  public static void M2()  {    Console.WriteLine("我是一个静态方法");  }  } |

|  |
| --- |
| public static class Student  {  private static string \_name;  public static string Name  {  get  {  return \_name;  }  set  {  \_name = value;  }  }  public static void M1()  {  Console.WriteLine("Hello World");  }  //不能再静态类中出现实例成员  //public int \_age;  } |

**类的练习**：

1. 定义一个学生类,有六个属性,分别为姓名、性别、年龄、语文、数学、英语成绩。（有一定延伸）

|  |
| --- |
| class Program  {  static void Main(string[] args)  {  //这样写太麻烦了，可以用到构造函数  //Student zsstudent = new Student();  //zsstudent.Name = "张三";  //zsstudent.Age = 23;  //zsstudent.Gender = '无';  //zsstudent.Chinese = 100;  //zsstudent.Math = 100;  //zsstudent.English = 100;  Student zsstudent = new Student("张三", 23, '无', 100, 100, 100);  zsstudent.SayHello();  zsstudent.ShowScore();  Student lsstudent = new Student("李四", 20, '女', 50, 50, 50);  //lsstudent.Name = "李四";  //lsstudent.Age = 20;  //lsstudent.Gender = '女';  //lsstudent.Chinese = 50;  //lsstudent.Math = 50;  //lsstudent.English = 50;  lsstudent.SayHello();  lsstudent.ShowScore();  Console.ReadKey();  }  } |

|  |
| --- |
| public class Student  {  //字段，属性，方法，构造函数  private string \_name;  private char \_gender;  private int \_age;  private int \_chinese;  private int \_math;  private int \_english;  public string Name  {  get  {  return \_name;  }  set  {  \_name = value;  }  }  public char Gender  {  get  {  if (\_gender != '男' && \_gender != '女')  {  return \_gender = '男';  }  return \_gender;  }  set  {  \_gender = value;  }  }  public int Age  {  get  {  return \_age;  }  set  {  if (value < 0 || value > 100)  {  value = 0;  }  \_age = value;  }  }  public int Chinese  {  get  {  return \_chinese;  }  set  {  \_chinese = value;  }  }  public int Math  {  get  {  return \_math;  }  set  {  \_math = value;  }  }  public int English  {  get  {  return \_english;  }  set  {  \_english = value;  }  }  //类本身就有一个默认的无参的构造函数  public Student(string name,int age,char gender,int chinese,int math,int english)  {  //在pro里执行完创造对象之后就会跳到这里  //Console.WriteLine("你看我什么时候被调用");  //在这里可以对对象的属性进行赋值  Name = name;  Age = age;  Gender = gender;  Chinese = chinese;  Math = math;  English = english;  }  public Student(string name,int chinese, int math, int english):this(name,0,'g',chinese,math,english)  {  ////每次都要写有很麻烦，用this，让他去找最全的构造函数  //Name = name;  //Chinese = chinese;  //Math = math;  //English = english;  }  //构造函数的重载  public Student(string name, int age, char gender)  {  //Name = name;  //Age = age;  //Gender = gender;  }  //析够函数 注意和构造函数区别  //帮助我们释放资源  //是当我们想让他马上释放资源才使用的  ~Student()  {  }  public void SayHello()  {  Console.WriteLine("我叫{0}，我今年{1}岁了，我是{2}生", Name, Age, Gender);  }  public void ShowScore()  {  Console.WriteLine("我叫{0}，我的总成绩{1}，平均成绩{2}", Name, Chinese + Math + English, (Chinese + Math + English) / 3);  }  } |

### 7、构造函数

作用：帮助我们初始化对象(给对象的每个属性依次的赋值)  
构造函数是一个特殊的方法：  
1)、构造函数没有返回值，连void也不能写。  
2)、构造函数的名称必须跟类名一样。

创建对象的时候会执行构造函数  
构造函数是可以有重载的。

类当中会有一个默认的无参数的构造函数，当你写一个新的构造函数之后，不管是有参数的还是  
无参数的，那个默认的无参数的构造函数都被干掉了。

* 构造方法用来创建对象，并且可以在构造函数中对对象进行初始化。
* 构造函数是用来创建对象的特殊方法，方法名和类名一样，没有返回值，连void都不用。
* 构造函数可以有参数，new对象的时候传递函数参数即可
* 如果不指定构造函数，则类有一个默认的无参构造函数。如果指定了构造函数，则不再有默认的无参构造函数，如果需要无参构造函数，则需要自己来写。
* 构造函数可以重载，也就是有多个参数不同的构造函数。
* 使用构造方法能解决：

1)、在赋初值时，重复的书写对象名

2)、假如我们有一个属性，不允许用户随意改动。我们一般把这个属性定义为只读类型的属性。

那么这个只读类型的属性就不能在实例化后对他赋值了，那么我们如何对他初始化呢？我们可以通过构造方来进行初始化。

我们定义好一个类,如果没有写构造方法,那么编译器就会自动在这个类中给我们添加一个没有参数的构造方法。

一旦我们写了一个构造方法,那么编译器就不会再给我们添加这个没有参数构造方法了。

### 8、new关键字

Person zsPerson=new Person();  
new帮助我们做了3件事儿：  
1)、在内存中开辟一块空间  
2)、在开辟的空间中创建对象  
3)、调用对象的构造函数进行初始化对象

### 9、this关键字

1)、代表当前类的对象  
2)、在类当中显示的调用本类的构造函数 :this

### 10、析够函数(\*) ~

* 不能在结构中定义析构函数。只能对类使用析构函数。
* 一个类只能有一个析构函数。
* 无法继承或重载析构函数。
* 无法调用析构函数。它们是被自动调用的。
* 析构函数既没有修饰符，也没有参数。

**练习**：

写一个Ticket类,有一个距离属性(本属性只读,在构造方法中赋值),不能为负数,有一个价格属性,价格属性只读,并且根据距离distance计算价格Price (1元/公里):

–0-100公里 票价不打折

–101-200公里 总额打9.5折

–201-300公里 总额打9折

–300公里以上 总额打8折

有一个方法,可以显示这张票的信息.

|  |
| --- |
| class Program  {  static void Main(string[] args)  {  Ticket t = new Ticket(150);  t.ShowTicket();  Console.ReadKey();  }  } |

|  |
| --- |
| public class Ticket  {  //有一个距离属性(本属性只读, 在构造方法中赋值),不能为负数,有一个价格属性,价格属性只读,  //且根据距离distance计算价格Price(1元/公里):  //–0-100公里 票价不打折  //–101-200公里 总额打9.5折  //–201-300公里 总额打9折  //–300公里以上 总额打8折  private double \_distance;  private double \_price;  public double Distance  {  //只读  get  {  return \_distance;  }  }  //–0-100公里 票价不打折  //–101-200公里 总额打9.5折  //–201-300公里 总额打9折  //–300公里以上 总额打8折  public double Price  {  //在这就可以处理这些  get  {  if (\_distance > 0 && \_distance <= 100)  {  return \_distance \* 1.0;  }  else if (\_distance >= 101 && \_distance <= 200)  {  return \_distance \* 0.95;  }  else if (\_distance >= 201 && \_distance <= 300)  {  return \_distance \* 0.9;  }  else  {  return \_distance \* 0.8;  }  }  }  //构造函数  public Ticket(double distance)  {  if (distance < 0)  {  distance = 0;  }  \_distance = distance;  }  public void ShowTicket()  {  Console.WriteLine("{0}公里需要{1}元", Distance, Price);  }  } |