C#程序设计及应用

唐大仕
dstang2000@263.net
北京大学

第3章 面向对象的C#语言

北京大学 唐大仕
dstang2000@263.net
http://www.dstang.com

本章内容

- 类、字段、方法
- 属性、索引
- 类的继承
- 修饰符
- 接口
- 结构与枚举
- 面向对象：继承、封装、多态
- UML类图简介

3.1 类 字段 方法

现实中的事物抽象为类

- 类(class)最基本的要素是
  - 字段(field): 变量
  - 方法(method)： 函数

定义类中的字段和方法

```
class Person {
        public string name;
        public int age;
        public void SayHello()
        {
                Console.WriteLine("Hello!  My name is " + name );
        }
        public string GetInfo()
        {
                return "Name: " + name + ", Age: " + age;
        }
}
```

构造方法（constructor）

- 构造方法的主要作用是完成对象的初始化工作
- (1)构造方法的方法名与类名相同。
- (2)构造方法没有返回类型，也不能写void。

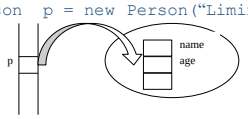
```
public Person( string n, int a ){
        name = n;
        age = a;
}
```


默认(default)构造方法

- 如果用户没有定义任何构造方法，
- 则系统会自动产生一个
- public Person() {}

对象的创建

- 构造方法不能显式地直接调用，而是用new来调用。
- `Person  p = new Person("Liming", 20 );`



对象的使用

- `Person  p = new Person("Liming", 20 );`
- Console.WriteLine(p.name) ;
- p.SayHello();

析构方法

```
class Person {
        ……
        ~ Person() {
                ……
        }
}
```

- 由于C#自动进行对象的释放，所以用户一般不定义析构方法

方法的重载（overloading）

```
public void SayHello(){
        Console.WriteLine("Hello!  My name is " + name );
}

public void SayHello( Person another ){
        Console.WriteLine("Hello," + another.name
                + "! My name is " + name );
}
```

方法的签名：方法名及参数个数及类型构成（参数名不算）

OverloadingTest.cs

使用this

this指这个对象本身，常用于：

（1）访问这个对象的字段及方法(VS会智能提示）

（2）区分字段与局部变量

```
public Person( int age, string name ){
        this.age = age; this.name = name;
}
```

(3) 用于构造方法调用另一个构造方法，注意其位置

```
public Person(  ) : this( 0, "" )
{           // 构造方法的其他语句；  }
```


应用示例：银行系统

- 系统中有几类对象？Account, Bank, ATM
- 每个类中有什么字段、方法？

注：可在vs中，添加“类关系图”（类图）

3.2 属性 索引

使用属性、索引的示例

- 使用属性 button1.Text
  - button1.Text = “说你好”;
    - 含义相当于button1.SetText(“说你好”);
  - string s = button1.Text;
    - 含义相当于s = button1.GetText();
- 使用属性 string s= “abcde”;
  - 求出长度： s.Length
    - 含义上相当于s.GetLength();
- 使用索引 string s=“abcde”;
  - 求出第0个字符： s[0]
    - 含义上相当于s.Get(0)

属性（property)的书写

```
private string _name;
public string Name
{
        get
        {
                return _name;
        }
        set
        {
                _name = value;
        }
}
```


在C#3以上版中可简写为

- public string Name { set; get; }

对属性进行访问

- Person p = new Person();
- p.Name = “Li Ming”;
- Console.WriteLine(p.Name);

- 编译器产生的方法是：
- void set_Name(string value);
- string get_Name();

PersonProperty.cs

属性与字段的比较

- 由于属性实际上是方法，
- 所以属性可以具有优点
  - 可以只读或只写： 只有get或set
  - 可以进行有效性检查：if…
  - 可以是计算得到的数据：

```
public string Info{
    get{return "Name:" + Name + ",Age:" + Age;}
}
```

  - 可以定义抽象属性

索引器(Indexer)

```
修饰符 类型名 this [ 参数列表 ]
{
    set
    {
    }
    get
    {
    }
}
```


使用索引

- 对象名[参数]

- 编译器自动产生两个方法，以供调用：
- T get_Item(P);
- void set_Item(P, T value);

IndexerRecord.cs

属性与索引的比较

属 性	索 引 器
通过名称标识	通过参数列表进行标识
通过简单名称来访问	通过[]运算符来访问
可以用static修饰	不能用static修饰
属性的get访问器没有参数	索引的get访问器具有与索引相同的参数列表
属性的set访问器包含隐式value参数	除了value参数外，索引的set访问器还具有与索引相同的参数列表

IndexerBitArray.cs

3.3 类的继承

使用继承的示例

- 我们定义的窗体
  - public class Form1 : System.Windows.Forms.Form
  - 神奇的冒号

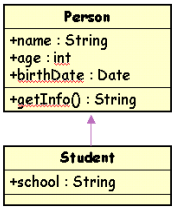
继承(inheritance)

- 子类subclass、父类baseclass
- C#中采用单继承
- 所有的类都是通过直接或间接地继承
- object(即System.Object)得到的。

```
class SubClass : BaseClass {
.....
}
```


示例

```
class  Student  : Person {
        //.....
}
```



继承

- 子类自动地从父类那里继承所有的
  - 字段、方法、属性、索引器等成员作为自己的成员。
- 除了继承父类的成员外，子类还可以
  - 添加新的成员，
  - 隐藏或修改父类的成员。

字段的继承、添加与隐藏

```
class  A{
  public int a;
}
class  B : A
{
  new public int a;
}
```


方法的继承、添加

- 方法的继承（自动）
- 方法的添加（多定义一些方法）

与父类同名的方法

- 一是定义同名、但参数列表（签名）与父类不同的方法，这称为对父类方法的重载（Overloading）
- 二是定义同名且参数列表也与父类相同的方法，这称为新增加一种方法 ，用new表示
- 三是定义同名且参数列表也与父类相同的方法，而且父类的方法用了abstract或virtual进行了修饰，子类的同名方法用了override进行了修饰，这称为虚方法的覆盖（Overriding）。

InheritFieldMethod.cs

使用base

```
void sayHello(){
        base.sayHello();
        Console.WriteLine( "My school is " + school );
}

Student(string name, int age, string school )： base( name, age )
{
        this.school = school;
}
```


父类与子类的转换

```
Person  p1 = new Person();
Person  p2 = new Student();
Student  s1 = new Student();
Student  s2 = new Student();
p1 = s1; //可以，因为Person类型的变量可以引用Student对象
s2 = p1; //不行，因为会产生编译错误
s2 = (Student) p1; // 编译时可以通过，运行时则会出现类型不能转换的异常
s2 = (Student) p2; //正确，因为p2引用的正好是Student对象实例
```


as运算符

- 如果不能转换，则值为null
  - Student s3 = p1 as Student; //结果s3为null
  - Student s4 = p2 as Student; //s4被赋值
- 与强制类型转换的差别
  - as只能针对引用型变量
  - 如果不能转换，as运算不会引起异常，只是值为null

is运算符

- if(p is Person)
- 判断一个对象是不是某个类(及其子类)的实例

typeof()运算符

- 获得其运行时的类型
  - Type t = typeof(变量);
  - Type t = typeof(类名);

属性、索引的继承

- 属性、索引也是可以继承的

3.4 修饰符

访问控制符

访问控制符	同类中	相同程序集的子类	相同程序集的非子类	不同程序集的子类	不同程序集的非子类
public	Yes	Yes	Yes	Yes	Yes
protected internal	Yes	Yes	Yes	Yes	
protected	Yes	Yes		Yes	
internal	Yes	Yes	Yes		
private	Yes				

static

- static的字段、方法、属性是属于整个类的
  - static方法中，不能访问实例变量
  - 调用static方法时，直接用类名访问
    - Console.Write(...); Math.Sqrt(...);
    - Convert.ToDateTime(…); DateTime.Parse
    - String.Copy(a);String.Format(“{0}”,x)
  - static变量可以用来表示“全局变量”
- 在c#2.0中，类名也可以用static来修饰

static构造方法

```
class Person {
        static long totalNum;
        static Person() {
                totalNum = (long)52e8;
                Console.WriteLine("人类总人口" + totalNu
m );
        }
}
```

Static构造方法只会调用一次，但其调用时间是不确定的。

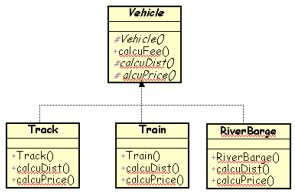
const及readonly

- const相当于静态常量
  - 如Math.PI
- readonly相当于不可改量，只能赋一次值
  - 如String.Empty
  - 在构造方法中赋值，或者在声明时就赋值
- 注：
  - const 只能用于基本类型及string
  - readonly只能修饰字段，而const还可以修饰局部变量

sealed及abstract

- sealed类，不可继承(也有利于编译优化)
  - 如**String Console Math Convert Graphics Font**
- abstract类，不可实例化（new）
  - 如Array, **RandomNumberGenerator**
- abstract的方法体，不用{}，用;
  - abstract 类型 方法名(参数列表);
  - abstract 类型 属性名{get;set;}

抽象类表示了其子类的属性



AbstractShapeTest.cs

小结

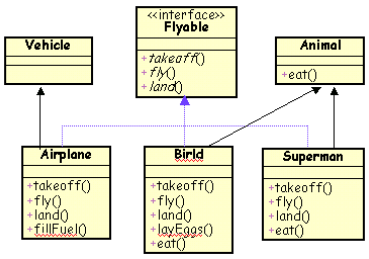
- public/private/internal/protected 是访问控制符
- static 属于类的而非实例的
- const 常量　readonly只读量
- sealed 不可继承的　abstract 抽象的

3.5 接口

接口(interface)

- 接口实际上是一个约定
  - 如：ICloneable, IComparable
- 接口是抽象成员的集合
  - ICloneable含有方法 clone()
  - IComparable含有方法 compare()
- 接口是一个引用类型，比抽象类更抽象

帮助实现多重继承



接口的用处

- 实现不相关类的相同行为
  - 不需要考虑这些类之间的层次关系
- 通过接口可以了解对象的交互界面，而不需了解对象所对应的类
  - 例如：
  - **public sealed class String : IComparable, ICloneable, IConvertible, IEnumerable**

定义一个接口

```
public interface IStringList
{
  void Add(string s);
  int Count { get; }
  string this[int index] { get; set; }
}
```

注：public abstract 这两个关键词不写出来

实现接口

```
class 类名 ： [父类，] 接口， 接口， ……，接口
{
    public 方法（ ）{……}
}
```


显式接口成员实现

- 在实现多个接口时，如果不同的接口有同名的方法，
- 为了消除歧义，需要在
- 方法名前写接口名
  - void IWindow.Close () {……}
- 调用时，只能用接口调用
  - ((IWindow) f).Close();

总之

- 接口是一种约定
- 接口中有多个抽象方法
- 接口能实现多继承

- 面向接口进行编程

3.6 结构及枚举

结构struct

结构常用来表示较简单的多个分量（字段）

如： Point, Color, Size, DateTime

Int32

public struct Int32 : IComparable, IFormattable, IConvertible, IComparable<int>, IEquatable<int>

可以有方法、属性等其他成员

例

```
struct Point
{
  public double x, y;
  public Point(int x, int y) {
        this.x = x;
        this.y = y;
  }
  public double R(){
        return Math.Sqrt(x*x+y*y);
  }
}
```


定义struct要注意

- struct是值类型
  - 结构不能包含无参数构造方法
    - 如 Point p; s.X=100;
  - 每个字段在定义时，不能给初始值
  - 构造方法中，必须对每个字段进行赋值
- struct 是sealed的，不能被继承

StructPoint.cs StructNew.cs

使用结构要注意

- struct是值类型
  - 实例化时，使用new，但与引用型变量的内存是不同的
    - Point p = new Point(100,80);
  - 值类型变量在赋值时，实行的是字段的copy
    - Point p1, p2;
    - p1=p2;

枚举 (enum)

枚举实际上是有意义的整数

如FontStyle, GraphicsUnit, KnownColor, DockStyle, DialogResult

EnumColor.cs

定义枚举

声明自己的属性

enum MyColor　　(注：后者可以跟一个：int）

{

Red,

Green=1,

Blue=2

}

使用枚举

- 赋值及比较
  - MyColor c = MyColor.Red;
  - if(c==MyColor.Red)
  - switch(c){ case MyColor.Red: ... }
- 与字符串的转换
  - Console.WriteLine(c.ToString());
  - c = (MyColor) Enum.Parse(typeof(MyColor), "Red");

总之

- 结构主要用来表示多个分量
- 枚举主要用来表示符号化的常量
- 它们都是值类型

3.7 面向对象编程

面向对象OO

- Object Oriented方法的三大特点
  - 继承 inheritance
    - 子类继承父类的成员，还可增加、调用、隐藏
    - 提高软件模块的可重用性和可扩充性
  - 封装 encapsulation
    - 使用接口，而不关心具体的类
    - 使用属性，而将字段设为private
  - 多态 polymorphism
    - 相同的方法，不同的参数
    - 自动调用子类相应的方法（虚方法调用，以后讲））

UML类图简介

- UML，统一建模语言
- 有类图、状态图、时序图等多种图形

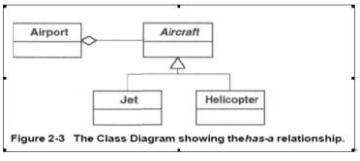


Figure 2-3 The Class Diagram showing the *has-a* relationship.

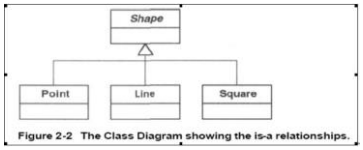


Figure 2-2 The Class Diagram showing the is-a relationships.

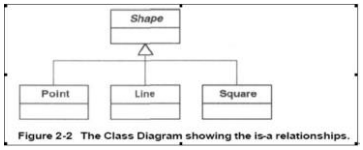


Figure 2-2 The Class Diagram showing the is-a relationships.

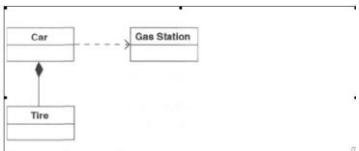


Figure 2-4 The Class Diagram showing composition and the uses relationship.

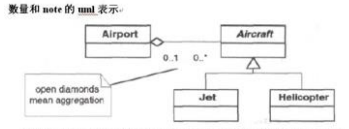


Figure 2-6 The cardinality of the Airport-Aircraft relationship.

VS中的类关系图

- 在项目上，点右键，"查看类关系图"
- 添加新项，其他项，类关系图
  - 然后将相应的类文件拖动到该“类关系图”中

UML工具

- IBM Rational XDE for .NET
- Borland Together
- Microsoft Visio

- 正向工程，由UML图自动产生代码
- 反向工程，由代码自动产生UML图

小结

- 类class、接口interface
- 结构struct、枚举enum
- 类的成员：字段、方法、属性、索引
- 修饰词
  - public protected internal private
  - static　const　readonly
  - abstract sealed virtual override new
- OO与UML

练习

- 参见讲义及ch03目录

进一步阅读

- 书稿《3 类和接口.doc》
- C#语言规范
- http://www.uml.org.cn

问题与讨论

dstang2000@263.net

编程提示

- 数字要有意义，不能天上掉下来
- 常用手段
  - 使用变量　deltx = 100
  - 使用常量　const int MAX_LEN = 1000
  - 使用系统常量　Math.PI
  - 使用枚举