# 参考资料

中文手册

http://shouce.jb51.net/net/index.html

# myGit

git@github.com:zhtool/MyBat.git

# C#Test

## 第三部分 面向对象的C#

### 修饰符

#### private 私有的

#### public 公有的

#### protected 受保护的

#### partial 部分的

#### internal 内部的

#### extern 外部的

#### static 静态的

#### virtual 虚拟的

#### override 重写

#### abstract 抽象的

#### sealed 密封的

### 第十章 类

#### 10.1 类的声明

using System;

namespace zh.test

{

class A { }

class MyTestClass

{

static void OutputLine(string str)

{

System.Console.WriteLine(str);

}

static void Main()

{

A a = new A();

OutputLine("New Class A");

}

}

}

#### 10.2 类的成员

类 (class) 是最基础的 C# 类型。类是一个数据结构，将状态（字段）和操作（方法和其他函数成员）组合在一个单元中。

##### 10.2.1类的成员的访问修饰符用法

###### 例10-1

using System;

namespace zh.test

{

class ClassA

{

public int i;

protected int j;

private int k;

public void F()

{

i = 11;//正确，允许访问自身成员

j = 12;//正确，允许访问自身成员

k = 13;//正确，允许访问自身成员

}

}

class ClassB

{

public int i;

protected int j;

private int k;

public void F()

{

i = 21;//正确，允许访问自身成员

j = 22;//正确，允许访问自身成员

k = 23;//正确，允许访问自身成员

ClassA cA = new ClassA();

cA.i = 21;//正确，允许访问cA的公有成员

//cA.j = 22;//错误，不允许访问cA的保护成员

//cA.k = 23;//错误，不允许访问cA的私有成员

System.Console.WriteLine("ClassB.F()");

}

}

class ClassC:ClassA

{

private int m=34;

public new void F() //用new隐藏 ClassA.F()

{

int i = 31;//正确，允许访问自身成员

ClassA cA = new ClassA();

cA.i = 11;//正确，允许访问cA的公有成员

//cA.j = 32;//错误，父类的保护成员，限定符必须是子类

j = 32;//正确，允许访问继承来的保护成员

//cA.k = 33;//错误，不允许访问cA的私有成员

System.Console.WriteLine("ClassC.F() cA.i={0}", cA.i);

System.Console.WriteLine("ClassC.F() cC.i={0}", i);

System.Console.WriteLine("ClassC.F() cC.m={0}", m);

}

~ClassC()

{

System.Console.WriteLine("~ClassC()");

}

}

class MyTestClass

{

static void Main()

{

ClassB b = new ClassB();

b.F();

ClassC c = new ClassC();

c.F();

}

}

}

##### 10.2.2 this保留字

###### 例10-2

This保留字公限于在构造函数、类的方法和类的实例中使用。

using System;

namespace zh.test

{

class classA

{

int x = 5;

public void OutputLine()

{

System.Console.WriteLine("x={0}", x);

System.Console.WriteLine("this.x={0}", this.x);

}

}

class MyTestClass

{

static void Main()

{

classA a = new classA();

a.OutputLine();

}

}

}

##### 10.2.3 静态成员属于类所有，非静态成员属于类的实例-对象。

###### 例10-5 static

using System;

namespace zh.test

{

class classA

{

public int x ;

public static int y;

void F()

{

x = 1;//等价于 this.x=1

y = 1;//等价于 classA.x=1

}

static void G()

{

//x = 1;//错误，不能访问 this.x

y = 1;//等价于 classA.x=1

}

}

class MyTestClass

{

static void Main()

{

classA a = new classA();

a.x = 1;//正确

classA.y = 1;//正确，可以按类访问静态变量

//classA.x = 1;//错误，不能按类访问非静态变量

//a.y = 1;//错误，不能在类的实例中访问静态变量

System.Console.WriteLine("a.x={0}", a.x);

System.Console.WriteLine("classA.y={0}", classA.y);

}

}

}

##### 10.2.4 成员常量

###### 例10-5.1

using System;

namespace zh.test

{

class classA

{

public const int x = 10, y = 20, z = 30;

static classA()

{

System.Console.WriteLine("x={0}", x);

System.Console.WriteLine("y={0}", y);

System.Console.WriteLine("z={0}", x);

}

}

class MyTestClass

{

static void Main()

{

classA a = new classA();

}

}

}

#### 10.3构造和析构函数

##### 10.3.1构造函数

using System;

namespace zh.test

{

class classA

{

private int x, y, z;

public classA()

{

x = 11;

y = 21;

z = 31;

}

public classA(int inX,int inY,int inZ)

{

x = inX;

y = inY;

z = inZ;

}

public void Show(string inClassName)

{

System.Console.WriteLine("{0} x={1}",inClassName, x);

System.Console.WriteLine("{0} y={1}", inClassName, y);

System.Console.WriteLine("{0} z={1}", inClassName, z);

}

~classA()

{

System.Console.WriteLine("~classA()");

}

}

class MyTestClass

{

static void Main()

{

classA a = new classA();

a.Show("classA()");

classA b = new classA(1, 2, 3);

b.Show("classA(1, 2, 3)");

}

}

}

##### 10.3.3析构函数

using System;

namespace zh.test

{

class A { }

class MyTestClass

{

static void OutputLine(string str)

{

System.Console.WriteLine(str);

}

static void Main()

{

A a = new A();

OutputLine("New Class A");

}

~MyTestClass()

{

System.Console.WriteLine("~MyTestClass()");

}

}

}

### 第十一章 方法

#### 11.1 方法的声明

##### 例11.1 返回值

using System;

namespace zh.test

{

class MyTestClass

{

static int F(int inX,int inY)

{

if (inX > inY)

return inX;

else

return inY;

}

static void Main()

{

System.Console.WriteLine("Max of 6 and 8 is:{0}",F(6,8));

}

}

}

#### 11.2 方法中的参数

##### 11.2.1 值参数

###### 例11.2 实参作为拷贝传递给方法，实参什不会变化

using System;

namespace zh.test

{

class MyTestClass

{

static void F(int x,int y)

{

int temp = x;

y = x;

x = temp;

}

static void Main()

{

int i = 1, j = 2;

F(i, j);

System.Console.WriteLine("i={0},j={1}",i,j);

}

}

}

##### 11.2.3 引用参数

###### 例11.3引用型参数把实际值的地址传递给方法

using System;

namespace zh.test

{

class MyTestClass

{

static void F(ref int x,ref int y)

{

int temp = x;

x = y;

y = temp;

}

static void Main()

{

int i = 1, j = 2;

F(ref i,ref j);

System.Console.WriteLine("运行结果： i={0},j={1}",i,j);

}

}

}

多个变量指向同一内在地址

using System;

namespace zh.test

{

class MyTestClass

{

static string s;

static void F(ref string x,ref string y)

{

string temp = x;

y = x;

x = temp;

s = "ssss";

System.Console.WriteLine("x={0},y={1},s={2}", x, y,s);

}

static void Main()

{

F(ref s, ref s);

}

}

}

##### 11.2.3 输出参数

输出型参数用于传递方法返回的数据

using System;

namespace zh.test

{

class MyTestClass

{

static string s;

static void F(out string x)

{

x = "OutString";

}

static void Main()

{

F(out s);

System.Console.WriteLine("s={0}", s);

}

}

}

##### 11.2.4 数组型参数

using System;

namespace zh.test

{

class classA

{

public static void F( params int[] args)

{

System.Console.WriteLine("args.Length={0}", args.Length);

foreach (int i in args)

System.Console.WriteLine("{0} ", i);

Console.WriteLine();

}

}

class MyTestClass

{

static void Main()

{

int[] a = { 1, 2, 3 };

classA.F( a);

classA.F(new int[] {10, 20, 30, 40});

//classA.F(10, 20, 30, 40);//简写

classA.F(new int[]{});

//classA.F();//简写

}

}

}

#### 11.3 静态和非静态方法

静态方法不属于类的某一个具体的实例。静态方法只能访问类中的静态成员

##### 例11-5

using System;

namespace zh.test

{

class classA

{

int x;

static int y;

public static void F()

{

//x = 1;//错误，不允许访问

y = 2;//正确，允许访问

//System.Console.WriteLine("x={0}", x);

System.Console.WriteLine("y={0}", y);

Console.WriteLine();

}

public void G()

{

x = 1;//错误，不允许访问

y = 2;//正确，允许访问

System.Console.WriteLine("x={0}", x);

System.Console.WriteLine("y={0}", y);

}

}

class MyTestClass

{

static void Main()

{

classA.F();

classA a = new classA();

a.G();

}

}

}

#### 11.4. 方法的重载

##### 例11-9

using System;

namespace zh.test

{

class Student

{

public string name;

public int age;

public double weight;

public Student(string inName, int inAge, double inWeight)

{

name = inName;

age = inAge;

weight = inWeight;

}

public int max(int x,int y)

{

return x > y ? x : y;

}

public double max(double x, double y)

{

return x > y ? x : y;

}

}

class MyTestClass

{

static void Main()

{

Student s1 = new Student("student1", 20, 71.1);

Student s2 = new Student("student2", 21, 72.2);

Console.WriteLine("{0} and {1} max age is {2}", s1.name, s2.name, s1.max(s1.age, s2.age));

Console.WriteLine("{0} and {1} max weight is {2}", s1.name, s2.name, s1.max(s1.weight, s2.weight));

}

}

}

#### 11.5 操作符重载

##### 11.5.2 使用成员方法重载操作符

###### 例11-10 一元操作符重载

using System;

namespace zh.test

{

class Player

{

public int NeiLi;

public int TiLi;

public int JingYan;

public int NeiLi\_r;

public int TiLi\_r;

public Player()

{

NeiLi = 10;

TiLi = 50;

JingYan = 0;

NeiLi\_r = 10;

TiLi\_r = 50;

}

public static Player operator ++ (Player p)

{

p.NeiLi = p.NeiLi\*2;

p.TiLi = p.TiLi\*2;

p.NeiLi\_r = p.NeiLi;

p.TiLi\_r = p.TiLi;

return p;

}

public void Show()

{

System.Console.WriteLine("TiLi:{0}",TiLi);

System.Console.WriteLine("JingYan:{0}", JingYan);

System.Console.WriteLine("NeiLi:{0}", NeiLi);

System.Console.WriteLine("TiLi\_full:{0}", TiLi\_r);

System.Console.WriteLine("NeiLi\_full:{0}", NeiLi\_r);

}

}

class MyTestClass

{

static void Main()

{

Player p = new Player();

p.Show();

p++;

Console.WriteLine("Now upgrading....");

Console.WriteLine();

p.Show();

p++;

Console.WriteLine("Now upgrading....");

Console.WriteLine();

p.Show();

}

}

}

###### 例11-11 二元操作符重载

using System;

namespace zh.test

{

class DKR

{

public int x,y,z;

public DKR(int ix,int iy,int iz)

{

x = ix;

y = iy;

z = iz;

}

public static DKR operator +(DKR d1,DKR d2)

{

DKR dkr = new DKR(0,0,0);

dkr.x = d1.x+d2.x;

dkr.y = d1.y + d2.y;

dkr.z = d1.z + d2.z;

return dkr;

}

}

class MyTestClass

{

static void Main()

{

DKR d1 = new DKR(3, 2, 1);

DKR d2 = new DKR(0, 6, 5);

DKR d3 = d1 + d2;

System.Console.WriteLine("d3 is {0},{1},{2}", d3.x, d3.y, d3.z);

}

}

}

### 第十二章 域和属性

#### 12.1 域

##### 12.1.1

using System;

namespace zh.test

{

class classA

{

//域

public int x;

protected int y;

private int z;

public classA(int ix,int iy,int iz)

{

x = ix;

y = iy;

z = iz;

System.Console.WriteLine("x={0},y={1},z={2}", x,y,z);

}

}

class MyTestClass

{

static void Main()

{

classA d1 = new classA(3, 2, 1);

}

}

}

##### 12.1.2静态域和非静态域

###### 例12-1

using System;

namespace zh.test

{

class classA

{

//域

static int count;

int number;

public classA()

{

count = count + 1;

number = count;

}

public void Show()

{

System.Console.WriteLine("object{0}:count={1}",number,count);

}

}

class MyTestClass

{

static void Main()

{

classA a = new classA();

a.Show();

Console.WriteLine("-------------------------------");

classA b = new classA();

a.Show();

b.Show();

Console.WriteLine("-------------------------------");

classA c = new classA();

a.Show();

b.Show();

c.Show();

}

}

}

##### 12.1.3 只读域

###### 例12-2

using System;

using TR = zh.test.ReadOnly;

namespace zh.test.ReadOnly

{

class classA

{

public static readonly double PI = 3.14159;

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

Console.WriteLine("{0}.PI={1}", "classA", TR.classA.PI);

}

}

}

##### 12.1.4 域的初始化

using System;

using TR = zh.test.ReadOnly;

namespace zh.test.ReadOnly

{

class classA

{

public static readonly int x = 1;

public static readonly int y = x + 1;

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

Console.WriteLine("x={0} y={1}", TR.classA.x,TR.classA.y);

}

}

}

x=1 y=2

-----------------

using System;

using TR = zh.test.ReadOnly;

namespace zh.test.ReadOnly

{

class classA

{

public static readonly int x;

public static readonly int y;

static classA()

{

x = 21;

y = 22;

Console.WriteLine("x={0} y={1}", x, y);

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

TR.classA a = new TR.classA();

Console.WriteLine("--------------");

Console.WriteLine("x={0} y={1}", TR.classA.x, TR.classA.y);

}

}

}

x=21 y=22

--------------------

x=21 y=22

#### 12.2 属性

##### 12.2.1 声明

##### 12.2.2 访问属性的值

###### 例12-3

using System;

using nsFile = zh.test.Files;

namespace zh.test.Files

{

class File

{

private string \_FileName;

public string FileName

{

set{ if (\_FileName != value){\_FileName = value;}}

get{return \_FileName; }

}

public File()

{

\_FileName = "长城";

}

public File(string sFileName)

{

\_FileName = sFileName;

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsFile.File f = new nsFile.File();

Console.WriteLine("File.FileName={0} ", f.FileName);

f.FileName = "长城1";

Console.WriteLine("File.FileName={0} ", f.FileName);

Console.WriteLine("--------------");

nsFile.File f1 = new nsFile.File("英雄");

Console.WriteLine("File.FileName={0} ", f1.FileName);

f1.FileName = "英雄1";

Console.WriteLine("File.FileName={0} ", f1.FileName);

}

}

}

###### 例12-4

using System;

using nsCustomer = zh.test.customers;

namespace zh.test.customers

{

/// <summary>

/// 记录酒店入住客人信息

/// </summary>

class Customer

{

public enum sex { men, women };

private string \_Name;

public string Name

{

get { return \_Name; }

}

//private sex \_Sex;

//public sex Sex

//{

// get { return \_Sex; }

//}

private string \_Sex;

public string Sex

{

get { return \_Sex; }

}

private string \_RoomNo;//房间号

public string RoomNo

{

get { return \_RoomNo; }

set { if (\_RoomNo != value) { \_RoomNo = value; } }

}

private int \_Day;//入住天数

public int Day

{

get { return \_Day; }

set { if (\_Day != value) { \_Day = value; } }

}

public Customer(string sName, string sSex, string sRoomNo, int iDay)

{

\_Name = sName;

\_Sex = sSex;

\_RoomNo = sRoomNo;

\_Day = iDay;

}

public void setCustomer(string sName, string sSex, string sRoomNo, int iDay)

{

\_Name = sName;

\_Sex = sSex;

RoomNo = sRoomNo;

Day = iDay;

//\_RoomNo = sRoomNo;

//\_Day = iDay;

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsCustomer.Customer c = new nsCustomer.Customer("张三丰", "men", "1111", 3);

Console.WriteLine("姓名：{0};性别：{1};房间号：{2};天数：{3} ", c.Name, c.Sex, c.RoomNo, c.Day);

Console.WriteLine("--------------");

c.RoomNo = "1121";

c.Day = 4;

Console.WriteLine("姓名：{0};性别：{1};房间号：{2};天数：{3} ", c.Name, c.Sex, c.RoomNo, c.Day);

Console.WriteLine("--------------");

c.setCustomer("郭啸天", "men", "1131", 5);

Console.WriteLine("姓名：{0};性别：{1};房间号：{2};天数：{3} ", c.Name, c.Sex, c.RoomNo, c.Day);

}

}

}

姓名：张三丰;性别：men;房间号：1111;天数：3

--------------

姓名：张三丰;性别：men;房间号：1121;天数：4

--------------

姓名：郭啸天;性别：men;房间号：1131;天数：5

#### 12.3 小结

### 第十三章 事件和索引指示器

#### 13.1 事件\*？

#### 13.2 索引指示器

##### 例13-4

using System;

using nsTeams = zh.test.Teams;

namespace zh.test.Teams

{

class Team

{

string[] \_Name = new string[8];

public string this[int iIndex]

{

get { return \_Name[iIndex]; }

set { \_Name[iIndex] = value; }

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsTeams.Team t1 = new nsTeams.Team();

for (int i = 0; i < 6; i++)

{

t1[i] = "a" + i.ToString();

Console.WriteLine("t[{0}]={1} ", i,t1[i]);

}

}

}

}

#### 13.3 小结

### 第十四章 继承

#### 14.1继承机制

##### 14.1.1概述

###### 例14-1

参考例14-2

##### 14.1.2覆盖

###### 例14-2

//例14-2

using System;

using nsVehicles = zh.test.Vehicles;

namespace zh.test.Vehicles

{

class Vehicle //定义洗车类

{

private int \_Wheels;//公用成员：轮子个数

protected double \_Weight;//保护成员：重量

public int Wheels { get { return \_Wheels; } }

public double Weight { get { return \_Weight; } }

public Vehicle() { ;}

public Vehicle(int wheels, double weight)

{

\_Wheels = wheels;

\_Weight = weight;

}

public void Speak()

{

Console.WriteLine("this is Vehicle.Speak()! ");

}

}

}

namespace zh.test

{

class Car : nsVehicles.Vehicle //定义轿车类，从汽车类继承

{

int \_Passengers;//私有成员:乘客数

public Car(int wheels, double weight, int Passengers)

: base(wheels, weight)//用base关键字为基类成员赋值

{

\_Passengers = Passengers;

}

new public void Speak()//重载基类的方法

{

Console.WriteLine("this is Car.Speak()! ");

}

public int Passengers

{

get { return \_Passengers; }

}

}

class MyTestClass

{

static void Main()

{

Car c1 = new Car(4, 1600.01, 5);

c1.Speak();

Console.WriteLine("Car.wheels={0}\nCar.weight={1}\nCar.Passengers={2}", c1.Wheels, c1.Weight, c1.Passengers);

}

}

}

##### 14.1.3 base保留字

//使用Base关键字访问基类成员

using System;

using classS = zh.test.classS;

namespace zh.test.classS

{

class classA

{

public void F()

{

Console.WriteLine("classA.F()! ");

}

string[] \_Name = new string[8];

public string this[int iIndex]

{

get { return \_Name[iIndex]; }

set { \_Name[iIndex] = value; }

}

}

}

namespace zh.test

{

class classB : classS.classA //A派生类B

{

public void G()

{

base[1] = "abcd";//利用A的索引指示器赋值

string x = base[1];//调用A的索引指示器

base.F();

System.Console.WriteLine("classA.\_Name[1]={0}", x);

}

}

class MyTestClass

{

static void Main()

{

classB b = new classB();

b.G();

}

}

}

#### 14.2多态性

##### 14.2.1C#的多态性

编译时多态性、运行时多态性

##### 14.2.2虚方法

###### 例14-3

//例14-3

using System;

using classS = zh.test.classS;

namespace zh.test.classS

{

class classA

{

public void F() { Console.WriteLine("this is classA.F()! "); }

public virtual void G() { Console.WriteLine("this is classA.G()! "); }

}

}

namespace zh.test

{

class classB : classS.classA

{

new public void F() { Console.WriteLine("this is classB.F()! "); }

public override void G() { Console.WriteLine("this is classB.G()! "); }

}

class MyTestClass

{

static void Main()

{

classB b = new classB();

classS.classA a = b;

b.F(); b.G();a.F(); a.G();

}

}

}

##### 14.2.3在派生类中对虚方法进行重载

###### 例14-4

###### 基类先后指代不同派生类的实例，调用不同版本，实现多态性

//例14-4

using System;

using nsVehicles = zh.test.Vehicles;

namespace zh.test.Vehicles

{

class Vehicle

{

public int \_wheels;//公用成员：轮子个数

protected double \_weight;//保护成员：重量

public Vehicle() { ;}

public Vehicle(int wheels, double weight)

{

\_wheels = wheels;

\_weight = weight;

}

public virtual void Speak()//定义虚方法

{

Console.WriteLine("Vehicle.Speak()! ");

}

}

}

namespace zh.test

{

class Car : nsVehicles.Vehicle //定义轿车类，从汽车类继承

{

int \_Passengers;//乘客数

public Car(int wheels, double weight, int Passengers)

: base(wheels, weight)

{

\_wheels = wheels;

\_weight = weight;

\_Passengers = Passengers;

}

public override void Speak()//重载基类的虚方法

{

Console.WriteLine("Car.Speak()! ");

}

public int wheels { get { return \_wheels; } }

public double weight { get { return \_weight; } }

public int Passengers { get { return \_Passengers; } }

}

class Truck : nsVehicles.Vehicle //定义卡车类，从汽车类继承

{

int \_Passengers;//乘客数

int \_Load;//载重量：吨

public Truck(int wheels, double weight, int Passengers, int load)

: base(wheels, weight)

{

\_wheels = wheels;

\_weight = weight;

\_Passengers = Passengers;

\_Load = load;

}

public override void Speak()//重载基类的虚方法

{

Console.WriteLine("Truck.Speak()! ");

}

public int wheels { get { return \_wheels; } }

public double weight { get { return \_weight; } }

public int Passengers { get { return \_Passengers; } }

public int Load { get { return \_Load; } }

}

class MyTestClass

{

static void Main()

{

nsVehicles.Vehicle v1 = new nsVehicles.Vehicle(11, 1100.01);

Car c1 = new Car(4, 2200.01, 5);

Truck t1 = new Truck(12, 3300.01, 3, 12);

Console.WriteLine("-------v1------- ");

v1.Speak();

v1 = c1;

Console.WriteLine("-------v1 = c1------- ");

v1.Speak();

c1.Speak();

v1 = t1;

Console.WriteLine("-------v1 = t1------- ");

v1.Speak();

t1.Speak();

}

}

}

#### 14.3抽象和密封

##### 14.3.1抽象类

//例14.3.1抽象类

using System;

using nsClass = zh.test.Class;

namespace zh.test.Class

{

abstract class A//定义抽象类A

{

public abstract void F();//定义抽象方法F

}

abstract class B : A//定义抽象类B，继承抽象类A

{

//未重载抽象方法A.F

public void G() { Console.WriteLine("B.G()!"); }

}

}

namespace zh.test

{

class C : nsClass.B

{

public override void F()//重载抽象方法A.F

{ Console.WriteLine("C重载抽象方法A.F"); }

}

class MyTestClass

{

static void Main()

{

C c = new C();

c.F();

c.G();

}

}

}

###### 例14-5

//例14-5

using System;

using nsVehicles = zh.test.Vehicles;

namespace zh.test.Vehicles

{

abstract class Vehicle//定义抽象类

{

public int \_wheels;//公用成员：轮子个数

protected double \_weight;//保护成员：重量

public Vehicle(int wheels, double weight)

{

\_wheels = wheels;

\_weight = weight;

}

public virtual void Speak() //定义虚方法Speak

{

Console.WriteLine("this is Vehicle.Speak()! ");

}

}

}

namespace zh.test

{

class Car : nsVehicles.Vehicle //定义轿Car类，从抽象类Vehicle继承

{

int \_Passengers;//乘客数

public Car(int wheels, double weight, int Passengers)

: base(wheels, weight)//用base关键字为基类赋值

{

\_Passengers = Passengers;

}

public override void Speak()//重载虚方法Speak

{

Console.WriteLine("Car.Speak()! ");

Console.WriteLine("Car:wheels={0};weight={1};Passengers={2};", this.wheels, this.weight, this.Passengers);

}

public int wheels { get { return \_wheels; } }

public double weight { get { return \_weight; } }

public int Passengers { get { return \_Passengers; } }

}

class Truck : nsVehicles.Vehicle //定义轿Truck类，从抽象类Vehicle继承

{

int \_Passengers;//乘客数

int \_Load;//载重量：吨

public Truck(int wheels, double weight, int Passengers, int load)

: base(wheels, weight)

{

\_Passengers = Passengers;

\_Load = load;

}

public override void Speak()//重载虚方法Speak

{

Console.WriteLine("Truck.Speak()! ");

Console.WriteLine("Truck: wheels={0};weight={1};Passengers={2};Load={3};", this.wheels,this.weight,this.Passengers,this.Load);

}

public int wheels { get { return \_wheels; } }

public double weight { get { return \_weight; } }

public int Passengers { get { return \_Passengers; } }

public int Load { get { return \_Load; } }

}

class MyTestClass

{

static void Main()

{

Car c1 = new Car(4, 2200.01, 5);

Truck t1 = new Truck(12, 3300.01, 3, 12);

Console.WriteLine("-------------- ");

c1.Speak();

Console.WriteLine("-------------- ");

t1.Speak();

Console.WriteLine("-------------- ");

}

}

}

##### 14.3.2抽象方法

###### 例14-6

//例14-6

using System;

using nsVehicles = zh.test.Vehicles;

namespace zh.test.Vehicles

{

abstract class Vehicle

{

public int \_wheels;//公用成员：轮子个数

protected double \_weight;//保护成员：重量

public Vehicle(int wheels, double weight)

{

\_wheels = wheels;

\_weight = weight;

}

public abstract void Speak();//创建抽象方法

}

}

namespace zh.test

{

class Car : nsVehicles.Vehicle //定义轿车类，从汽车类继承

{

int \_Passengers;//乘客数

public Car(int wheels, double weight, int Passengers)

: base(wheels, weight)

{

\_Passengers = Passengers;

}

public override void Speak()//重载基类中的抽象方法Speak()

{

Console.WriteLine("Car.Speak()! ");

Console.WriteLine("Car: wheels={0};weight={1};Passengers={2};", wheels, weight, Passengers);

}

public int wheels { get { return \_wheels; } }

public double weight { get { return \_weight; } }

public int Passengers { get { return \_Passengers; } }

}

class Truck : nsVehicles.Vehicle //定义卡车类，从汽车类继承

{

int \_Passengers;//乘客数

int \_Load;//载重量：吨

public Truck(int wheels, double weight, int Passengers, int load)

: base(wheels, weight)

{

\_Passengers = Passengers;

\_Load = load;

}

public override void Speak()

{

Console.WriteLine("Truck.Speak()! ");

Console.WriteLine("Truck: wheels={0};weight={1};Passengers={2};Load={3};",wheels, weight, Passengers, Load);

}

public int wheels { get { return \_wheels; } }

public double weight { get { return \_weight; } }

public int Passengers { get { return \_Passengers; } }

public int Load { get { return \_Load; } }

}

class MyTestClass

{

static void Main()

{

//抽象类不允许创建实例，如下所示

//nsVehicles.Vehicle v1 = new nsVehicles.Vehicle(12,3300.03);

Car c1 = new Car(4, 2200.01, 5);

Truck t1 = new Truck(12, 3300.01, 3, 12);

Console.WriteLine("-------------- ");

c1.Speak();

Console.WriteLine("-------------- ");

t1.Speak();

}

}

}

###### 使用抽象方法重载基类的虚方法，实现“拦截”功能

//例14.3.2

using System;

using nsClass = zh.test.Class;

namespace zh.test.Class

{

class A

{

public virtual void F()//定义虚方法

{Console.WriteLine("A.F()!");}

}

abstract class B : A

{

public abstract override void F();//定义抽象方法

}

}

namespace zh.test

{

class C : nsClass.B

{

public override void F()//重载基类的抽象方法

{ Console.WriteLine("C.F()! "); }

}

class MyTestClass

{

static void Main()

{

C c = new C();

c.F();

nsClass.A a = new nsClass.A();

a.F();

}

}

}

##### 14.3.3密封类

###### 密封类不允许有派生类

例14.3.3

//例14.3.3

using System;

using nsClass = zh.test.Class;

namespace zh.test.Class

{

abstract class A

{

public abstract void F();//定义抽象方法

}

sealed class B : A//定义密封类B，继承自抽象类A

{

public override void F()

{

Console.WriteLine("重载抽象方法A.F()");

}

}

//class C : nsClass.B{;}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsClass.B b = new nsClass.B();

b.F()

Console.WriteLine("\"class C : nsClass.B\" 无法从密封类型\"zh.test.Class.B\"派生");

}

}

}

##### 14.3.4密封方法

###### 例14-7

//例14-7

using System;

using classS = zh.test.classS;

namespace zh.test.classS

{

class classA

{

public virtual void F() { Console.WriteLine("A.F()"); }

public virtual void G() { Console.WriteLine("A.G()"); }

}

class classB : classA

{

sealed override public void F() { Console.WriteLine("密封方法B.F()重载虚方法A.F(),B.F()不可被重载"); }

override public void G() { Console.WriteLine("B.G()重载虚方法A.G()"); }

}

}

namespace zh.test

{

class classC : classS.classB

{

override public void G() { Console.WriteLine("C.G()重载B.G()"); }

}

class MyTestClass

{

static void Main()

{

classS.classB b = new classS.classB();

Console.WriteLine("------B的方法-------");

b.F(); b.G();

classC c = new classC();

Console.WriteLine("------C的方法-------");

c.F(); c.G();

}

}

}

#### 14.4继承中关于属性的一些问题

###### 例14-18、19

//例14-18、19

using System;

using nsPeoples = zh.test.Peoples;

namespace zh.test.Peoples

{

public enum sex{woman, man};

//因为定义了抽象属性Card，类必须定义为抽象的

abstract public class People

{

private string \_Name = "张三丰";

public virtual string Name //虚属性

{get { return \_Name; }}

private string \_Sex = "men";

public virtual string Sex //虚属性

{get { return \_Sex; } }

protected string \_Card;

public abstract string Card //抽象属性

{get;set;}

}

class Customer : People

{

string \_No;

int \_Day;

public Customer(string sCard, string sNo, int iDay)

{

Card = sCard;

\_No = sNo;

\_Day = iDay;

}

public string No

{

get { return \_No; }

set { if (\_No != value) { \_No = value; } }

}

public int Day

{

get { return \_Day; }

set { if (\_Day != value) { \_Day = value; } }

}

public override string Name //重载基类的虚属性

{

get { return base.Name; }//访问基类中的访问器

}

public override string Sex

{get { return base.Sex; } }

public override string Card

{

get { return \_Card; }

set { \_Card = value; }

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsPeoples.Customer c = new nsPeoples.Customer("Card123", "No1234", 4);

Console.WriteLine("c.Name={0};c.Sex={1};c.Card={2};c.No={3};c.Day={4}", c.Name, c.Sex, c.Card, c.No, c.Day);

Console.WriteLine("---------");

}

}

}

#### 14.5小结

using System;

using nsClass = zh.test.Class;

namespace zh.test.Class

{

abstract class A

{

public virtual int X

{

get { return 0; }

}

int \_Y;

public virtual int Y

{

get { return \_Y; }

set { \_Y = value; }

}

public abstract int Z { get; set; }

}

class B : A

{

int \_Z;

public override int X{get{return base.X+1;}}

public override int Y

{

set { base.Y = base.Y<0?0:value; }

}

public override int Z

{

get { return \_Z; }

set { \_Z = value; }

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsClass.B b = new nsClass.B();

b.Y = 2;

b.Z = 3;

Console.WriteLine("class B : b.X={0};B.Y={1};B.Z={2}",b.X,b.Y,b.Z);

}

}

}

## 深入了解C#

### 第十五章 接口

#### 15.1组件编程技术

##### 15.1.1应用程序的体系结构

###### 表示层:向用户提供数据，展现用户接口

###### 商业层：实施商业逻辑，为表示层提供的服务

###### 数据访问层:提供数据访问服务，包括检索和存储

##### 15.1.2组件component

##### 15.1.3组件化程序设计

##### 15.1.4

#### 15.2接口定义

##### 15.2.1声明

##### 15.2.2接口继承

###### 例15-1

//例15-1

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IControl

{

void Paint();

}

interface ITextBox:IControl

{

void SetText(String sText);

}

interface IListBox : IControl

{

void SetItems(String[] sItems);

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

Console.WriteLine("接口继承示例");

}

}

}

#### 15.3接口的成员

##### 15.3.1接口成员定义

//例15.3.1

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

//接口成员:索引指示器、事件E、方法F、属性P

interface IExample

{

string this[int index] { get; set; }

event EventHandler E;

void F(int value);

string P { get; set; }

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

Console.WriteLine("接口继承示例");

}

}

}

##### 15.3.2对接口成员的访问

###### 例15-2

//例15-2

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

//接口成员:索引指示器、事件E、方法F、属性P

interface ISequcence

{

int Count { get; set; }

}

interface IRing

{

void Count(int i);

}

interface IIRingSequcence : ISequcence, IRing{ }

}

namespace zh.test

{

class C

{

void Test(nsIF.IIRingSequcence rs)

{

//rs.Count(1); 错误，Count有二义性

//rs.Count=1; 错误，Count有二义性

((nsIF.ISequcence)rs).Count = 1;

((nsIF.IRing)rs).Count(1);//正确调用IRing.Count

}

}

class MyTestClass

{

static void Main()

{

Console.WriteLine("接口继承示例");

}

}

}

###### 例15-3

//例15-3

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

//接口成员:索引指示器、事件E、方法F、属性P

interface IInteger

{

void Add(int i);

}

interface IDouble

{

void Add(double d);

}

interface INumber : IInteger, IDouble { }

}

namespace zh.test

{

class C

{

void Test(nsIF.INumber n)

{

n.Add(1); //错误，需要显示指派

n.Add(1.0); //正确

((nsIF.IInteger)n).Add(1);

((nsIF.IDouble)n).Add(1.0);

}

}

class MyTestClass

{

static void Main()

{

C c = new C();

Console.WriteLine("接口继承示例");

}

}

}

//例15-3.1

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IBase

{

void F(int i);

}

interface ILeft : IBase

{

new void F(int i);

}

interface IRight : IBase

{

void G(int i);

}

interface IDerived : ILeft, IRight { }

}

namespace zh.test

{

class C

{

void Test(nsIF.IDerived d)

{

d.F(1); //调用ILeft.F

((nsIF.IBase)d).F(1); //调用IBase.F

((nsIF.ILeft)d).F(1); //调用ILeft.F

((nsIF.IRight)d).F(1);//调用IBase.F

}

}

class MyTestClass

{

static void Main()

{

C c = new C();

Console.WriteLine("接口继承示例");

}

}

}

##### 15.3.3接口成员的全权名

###### 例15-1中Paint的全权名是：nsIF.IControl.IControl.Paint

#### 15.4接口的实现

##### 51.4.1类对接口的实现

###### 例15-4

//例15-4

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface ISequcence

{

void Add();

}

interface IRing

{

int Insert(object obj);

}

class IRingISequcence : ISequcence, IRing

{

public void Add() { ;}

public int Insert(object obj) { return 1;}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

Console.WriteLine("接口继承示例");

}

}

}

例15-4.1类实现了所有父接口的的成员

//例15-4.1

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IControl

{

void Paint();

}

interface ITextBox : IControl

{

void SetText(String sText);

}

class TextBox:ITextBox

{

public void Paint(){;}

public void SetText(String sText) { ;}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

Console.WriteLine("接口继承示例");

}

}

}

例15-4.2一个类可以实现多个接口

//例15-4.2

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IControl

{

void Paint();

}

interface IDataBound

{

void Bind(int b);

}

public class Control:IControl

{

public void Paint()

{ Console.WriteLine("Control >> IControl.Paint()"); }

}

public class EditBox:Control,IControl,IDataBound

{

public new void Paint()

{ Console.WriteLine("EditBox >> IControl.Paint()"); }

public void Bind(int b)

{ Console.WriteLine("EditBox >> IDataBound.Bind()"); }

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.Control c = new nsIF.Control();

nsIF.EditBox e = new nsIF.EditBox();

c.Paint();

e.Paint();

e.Bind(1);

}

}

}

例15-4.3显式接口成员

//例15-4.3显式接口成员

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IControl

{

void Paint();

}

interface IDataBound

{

void Bind(int b);

}

public class EditBox:IControl,IDataBound

{

void IControl.Paint()//显式接口成员执行体

{ Console.WriteLine("EditBox >> IControl.Paint()"); }

void IDataBound.Bind(int b)//显式接口成员执行体

{ Console.WriteLine("EditBox >> IDataBound.Bind()"); }

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.EditBox e = new nsIF.EditBox();

//e.Paint();

//显式接口成员只能通过接口调用

nsIF.IControl Control = e;

Control.Paint();

nsIF.IDataBound DataBound = e;

DataBound.Bind(1);

}

}

}

##### 15.4.2显式接口成员执行体

##### 15.4.3接口映射

在类中定位接口成员的实现称之为映射(interface mapping)

###### 例15.4.3-1接口映射

//例15.4.3-1派生类使用父类的成员实现父接口的成员

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface Interface1

{

void F();

}

class Class1

{

public void F() { Console.WriteLine("Class1.F"); }

public void G() { Console.WriteLine("Class1.G"); }

}

class Class2 : Class1, Interface1

{

new public void G()

{

Console.WriteLine("Class2.G");

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.Class2 c = new nsIF.Class2();

c.F();//Class2使用Class1的成员方法F来实现接口Interface1的成员方法F

c.G();

}

}

}

###### 例15.4.3-2派生类的一个成员实现多个接口的同名成员

//例15.4.3-2派生类的一个成员实现多个接口的同名成员

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface Interface1

{

void F();

}

interface Interface2

{

void F();

}

class Class1 : Interface1, Interface2

{

public void F()//成员F实现接口Interface1和Interface2的成员F

{

Console.WriteLine("Class1.F");

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.Class1 c = new nsIF.Class1();

c.F();

}

}

}

###### 例15.4.3-3 显式实现上例

//例15.4.3-3 显式实现上例

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface Interface1{void F();}

interface Interface2{void F();}

class Class1 : Interface1, Interface2

{

public void F(){ Console.WriteLine("Class1.F");}

//显式接口成员执行体

void Interface1.F(){Console.WriteLine("Interface1.F");}

void Interface2.F(){Console.WriteLine("Interface2.F");}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.Class1 c = new nsIF.Class1();c.F();

nsIF.Interface1 F1 = c;F1.F();

nsIF.Interface2 F2 = c;F2.F();

}

}

}

###### 例15.4.3-4接口IF2覆盖了接口IF1的成员，类C继承了IF2并实现IF1和IF2的成员

//例15.4.3-4

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface Interface1

{

void F();

}

interface Interface2 : Interface1

{

new void F();

}

class Class1 : Interface2

{

public void F()

{

Console.WriteLine("Class1.F");

}

//显式接口成员执行体

void Interface1.F()

{

Console.WriteLine("Interface1.F");

}

void Interface2.F()

{

Console.WriteLine("Interface2.F");

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.Class1 c = new nsIF.Class1();

c.F();

nsIF.Interface1 F1 = c;

F1.F();

nsIF.Interface2 F2 = c;

F2.F();

}

}

}

###### 例15.4.3-5多个接口继承了同一个父接口F1，F的成员只允许被实现一次

//例15.4.3-5

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IF

{

void F();

}

interface Interface1 : IF

{

void G(int i);

}

interface Interface2 : IF

{

void H(string s);

}

class Class1 : IF,Interface1, Interface2

{

public void F()

{

Console.WriteLine("Class1.F");

}

//显式接口成员执行体

void IF.F()//被继承三次，只允许实现一次

{

Console.WriteLine("IF.F");

}

void Interface1.G(int i)

{

Console.WriteLine("Interface1.F");

}

void Interface2.H(string s)

{

Console.WriteLine("Interface2.H");

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.Class1 c = new nsIF.Class1();

Console.WriteLine("----Class1----");

c.F();

nsIF.IF F0 = c;

Console.WriteLine("----IF----");

F0.F();

nsIF.Interface1 F1 = c;

Console.WriteLine("----Interface1----");

F1.F();

F1.G(1);

nsIF.Interface2 F2 = c;

Console.WriteLine("----Interface2----");

F2.F();

F2.H("str");

}

}

}

##### 15.4.4接口实现的继承机制

###### 例15.4.4-1类如果不显式地重新实现接口，派生类就无法改变基类中继承来的接口映射

//例15.4.4-1

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IControl

{

void F();

}

class Control:IControl

{

public void F()

{

Console.WriteLine("Control.F");

}

}

class TextBox:Control

{

//此处覆盖了Control.F，但Control.F对IControl.F的映射没有改变

new public void F()

{

Console.WriteLine("TextBox.F");

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.Control c = new nsIF.Control();

nsIF.TextBox t = new nsIF.TextBox();

nsIF.IControl ic = c;

nsIF.IControl it = t;

Console.WriteLine("----Control----");

c.F();

ic.F();

Console.WriteLine("----TextBox----");

t.F();

it.F();

}

}

}

###### 例15.4.4-2IControl.F被映射到虚方法，派生类可以重载虚方法并改变IControl.F的实现

//例15.4.4-2

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IControl

{

void F();

}

class Control:IControl

{

//IControl.F被映射到虚方法，派生类可以重载虚方法并改变IControl.F的实现

public virtual void F()

{

Console.WriteLine("Control.F");

}

}

class TextBox:Control

{

//重载Control.F，Control.F的映射发生改变

public override void F()

{

Console.WriteLine("TextBox.F");

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.Control c = new nsIF.Control();

nsIF.TextBox t = new nsIF.TextBox();

nsIF.IControl ic = c;

nsIF.IControl it = t;

Console.WriteLine("----Control----");

c.F();

ic.F();

Console.WriteLine("----TextBox----");

t.F();

it.F();

}

}

}

##### 15.4.5接口重实现

###### 例15.4.5-1派生类可以对基类中已经定义的接口成员进行重载，叫做接口重实现

//例15.4.5-1

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IControl

{

void F();

}

class Control:IControl

{

void IControl.F()

{

Console.WriteLine("Control.F");

}

}

class MyControl : Control, IControl

{

public void F()

{

Console.WriteLine("MyControl.F");

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.Control c = new nsIF.Control();

nsIF.MyControl m = new nsIF.MyControl();

nsIF.IControl ic = c;

nsIF.IControl im = m;

Console.WriteLine("----Control----");

//c.F();

ic.F();

Console.WriteLine("----TextBox----");

m.F();

im.F();

}

}

}

###### 例15.4.5-2继承而来的公有成员声明和继承而来的显式接口成员的声明参与到接口映射过程

//例15.4.5-2

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IMethods

{

void F();

void G();

void H();

void I();

}

class Base : IMethods

{

void IMethods.F() { Console.WriteLine("Base.F"); }

void IMethods.G() { Console.WriteLine("Base.G"); }

void IMethods.H() { Console.WriteLine("Base.H"); }

void IMethods.I() { Console.WriteLine("Base.I"); }

}

class Derived : Base, IMethods

{

public void F()

{

Console.WriteLine("Derived.F");

}

void IMethods.H() { Console.WriteLine("Derived.H"); }

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.Base b = new nsIF.Base();

nsIF.Derived d = new nsIF.Derived();

nsIF.IMethods ib = b;

nsIF.IMethods id = d;

Console.WriteLine("----Base----");

ib.F();ib.G();ib.H();ib.I();

Console.WriteLine("----Derived----");

d.F();id.F();id.G();id.H();id.I();

}

}

}

###### 例15.4.5-3类在重实现一个接口时，同时隐式地实现了该接口的所有父接口

//例15.4.5-3

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IBase

{

void F();

}

interface IDerived:IBase

{

void G();

}

class C:IDerived

{

void IBase.F() { Console.WriteLine("C.F"); }

void IDerived.G() { Console.WriteLine("C.G"); }

}

class D :C, IDerived

{

public void F() { Console.WriteLine("D.F"); }

public void G() { Console.WriteLine("D.G"); }

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.C c = new nsIF.C();

nsIF.D d = new nsIF.D();

nsIF.IDerived ic = c;

nsIF.IDerived id = d;

Console.WriteLine("----Base----");

ic.F();ic.G();

Console.WriteLine("----Derived----");

d.F(); d.G(); id.F(); id.G();

}

}

}

#### 15.5抽象类和接口

##### 例15.5-1抽象类允许将接口的方法映射到抽象的成员方法

//例15.5-1

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IMethods

{

void F();

void G();

}

abstract class C : IMethods

{

public abstract void F();

public abstract void G();

}

class Derived : C

{

public override void F()

{

Console.WriteLine("Derived.F");

}

public override void G()

{

Console.WriteLine("Derived.G");

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

nsIF.Derived d = new nsIF.Derived();

nsIF.IMethods id = d;

d.F(); d.G(); id.F(); id.G();

}

}

}

##### 例15.5-2显式说明的接口成员不能是抽象的，但它允许调用抽象方法

//例15.5-2

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IMethods

{

void F();

void G();

}

}

namespace zh.test

{

abstract class C : nsIF.IMethods

{

void nsIF.IMethods.F() { FF(); }

void nsIF.IMethods.G() { GG(); }

public abstract void FF();

public abstract void GG();

}

class Derived : C

{

public override void FF()

{

Console.WriteLine("Derived.FF");

}

public override void GG()

{

Console.WriteLine("Derived.GG");

}

}

class E:Derived{ }

class MyTestClass

{

static void Main()

{

E e = new E();

nsIF.IMethods ie = e;

Console.WriteLine("------E--------");

e.FF(); e.GG(); ie.F(); ie.G();

//Derived d = new Derived();

//nsIF.IMethods id = d;

//Console.WriteLine("------Derived--------");

//d.FF(); d.GG(); id.F(); id.G();

}

}

}

#### 15.6小结

##### 例15.6-1关于字典程序的小例子

//例15.6-1

using System;

using nsIF = zh.test.IF;

namespace zh.test.IF

{

interface IDictonary

{

void LoadLibrary(string sFileName);

void FreeLibrary(string sFileName);

void RestoreLibrary(string sFileName);

bool InsertWord(string sWord,out string sResult);

bool LookupWord(string sWord, string sResultWord, out string sResult);

void DeleteWord(string sWord);

}

}

namespace zh.test

{

class Dict

{

public void LoadLibrary(string sFileName) { Console.WriteLine("Dict.LoadLibrary"); }

public void FreeLibrary(string sFileName) { Console.WriteLine("Dict.FreeLibrary"); }

public void RestoreLibrary(string sFileName) { Console.WriteLine("Dict.RestoreLibrary"); }

public bool InsertWord(string sWord, out string sOutResult)

{

sOutResult = "Dict.InsertWord->sOutResult";

return true;

}

public bool LookupWord(string sWord, string sResultWord, out string sOutResult)

{

sOutResult = "Dict.LookupWord->sOutResult";

return true;

}

public void DeleteWord(string sWord) { Console.WriteLine("Dict.DeleteWord"); }

}

class NewDict : Dict, nsIF.IDictonary

{

void LoadLibrary(string sFileName) { Console.WriteLine("Dict.LoadLibrary1"); }

void FreeLibrary(string sFileName) { Console.WriteLine("Dict.FreeLibrary1"); }

public void RestoreLibrary(string sFileName) { Console.WriteLine("Dict.RestoreLibrary1"); }

public bool InsertWord(string sWord, out string sOutResult)

{

sOutResult = "Dict.InsertWord->sOutResult1";

return true;

}

public bool LookupWord(string sWord, string sResultWord, out string sOutResult)

{

sOutResult = "Dict.LookupWord->sOutResult1";

return true;

}

public void DeleteWord(string sWord) { Console.WriteLine("Dict.DeleteWord1"); }

}

class E : Dict { }

class N : NewDict { }

class MyTestClass

{

static void Main()

{

E e = new E();

Console.WriteLine("------Dict--------");

e.LoadLibrary("FileName");

e.FreeLibrary("FileName");

e.RestoreLibrary("FileName");

string str;

bool bBool = e.InsertWord("Str",out str);

Console.WriteLine(str);

bBool = e.LookupWord("str1","str2",out str);

Console.WriteLine(str);

e.DeleteWord("FileName");

Console.WriteLine("------NewDict--------");

N n = new N();

n.LoadLibrary("FileName");

n.FreeLibrary("FileName");

n.RestoreLibrary("FileName");

bBool = n.InsertWord("Str", out str);

Console.WriteLine(str);

bBool = n.LookupWord("str1", "str2", out str);

Console.WriteLine(str);

n.DeleteWord("FileName");

}

}

}

### 第十六章 组织应用程序

#### 16.1基本概念

##### 16.1.1动态链接库

##### 16.1.2编译单元

##### 16.1.3名字空间和装配

#### 16.2使用名字空间

##### 16.2.1名字空间的声明

###### 例16.2.1-1使用非嵌套语法实现名字空间的嵌套声明

//例16.2.1-1

using System;

using N1;

namespace N1

{

namespace N2

{

class A

{

public void F() { Console.WriteLine("A.F"); }

}

class B { }

}

}

//以下注释部分等价于以上部分

//namespace N1.N2

//{

// class A { }

// class B { }

//}

namespace zh.test

{

class N : N1.N2.A { }

class MyTestClass

{

static void Main()

{

N a = new N();

a.F();

}

}

}

##### 16.2.2成员与类型声明

##### 16.2.3改写”Well Come”程序

演示名字空间的基本用法，将"Welcome"程序分为库(WelcomeMsg.cs)和控制台(WelcomeApp.cs)两部分。库(WelcomeMsg.dll)用来提供显示消息，控制台程序WelcomeApp.exe执行显示。

###### 控制台编译命令如下

@echo off

rem if exist .\test.exe del .\test.exe

rem ---0、直接产生test.exe

rem csc test.cs

rem ---1-1、产生test.dll类库

csc /target:library WelcomeMessage.cs

rem ---1-2、产生test.exe应用程序

csc /reference:WelcomeMessage.dll WelcomeApp.cs

WelcomeApp.exe

pause

###### 例16-0原有”Welcome”程序

//例16-0

using System;

using nsCode = zh.Code.WelcomeMessage;

namespace zh.Code.WelcomeMessage

{

public class WelcomeMessage

{

string \_Msg;

public WelcomeMessage()

{ \_Msg = "WelCome!"; }

public string Msg

{

get { return \_Msg; }

set { \_Msg = value; }

}

}

}

namespace zh.test

{

class WelcomeApp

{

static void Main()

{

nsCode.WelcomeMessage w = new nsCode.WelcomeMessage();

System.Console.WriteLine(w.Msg);

System.Console.WriteLine("请输入你的名字！");

string sInput= System.Console.ReadLine();

w.Msg = "Welcome " + sInput;

System.Console.WriteLine(w.Msg);

}

}

}

###### 例16-1 文件名WelcomeMsg.cs

//例16-1产生WelcomeMessage.dll

using System;

namespace zh.Code.WelcomeMsg

{

public class WelcomeMessage

{

string \_Msg;

public WelcomeMessage()

{ \_Msg = "WelCome!"; }

public string Msg

{

get { return \_Msg; }

set { \_Msg = value; }

}

public void Show()

{System.Console.WriteLine(Msg); }

public void Show(string sStr)

{ System.Console.WriteLine(Msg+" "+sStr); }

}

}

###### 例16-2 文件名WelcomeApp.cs

//例16-2 本例产生控制台程序WelcomeApp.exe

using System;

using zh.Code.WelcomeMsg;

namespace zh.Code.test

{

public class WelcomeApp

{

static void Main()

{

WelcomeMessage w = new WelcomeMessage();

w.Show();

w.Msg = "Hello ";

System.Console.WriteLine("请输入你的名字！");

w.Show(Console.ReadLine());

}

}

}

#### 16.3使用指示符

##### 16.3.1别名使用指示符

###### 例16.3.1-1

//例16.3.1-1

using System;

using nsCode = zh.Code.WelcomeMsg;

using cWelcomeMessage = zh.Code.WelcomeMsg.WelcomeMessage;

namespace zh.Code.WelcomeMsg

{

public class WelcomeMessage

{

string \_Msg;

public WelcomeMessage()

{ \_Msg = "WelCome!"; }

public void Show() { Console.WriteLine(\_Msg); }

}

}

namespace zh.test

{

class WelcomeMsgs:cWelcomeMessage{}

class WelcomeApp

{

static void Main()

{

nsCode.WelcomeMessage w = new nsCode.WelcomeMessage();

w.Show();

WelcomeMsgs w1 = new WelcomeMsgs();

w1.Show();

}

}

}

##### 16.3.2名字空间使用指示符

##### 16.3.3

#### 16.4程序示例

##### 例16-3.1 MyShape.cs

//MyShape.cs //用于定义图形，作为其它图形的基类

using System;

namespace zh.MyShape

{

public class Shape

{

//虚方法，用于图形绘制

public virtual void Draw(){}

//虚方法，用于计算图形面积

public virtual int GetArea()

{ return 0;}

}

}

##### 例16-3.2 Rect.cs

//Rect.cs //定义矩形和正方形

using System;

namespace zh.MyShape

{

//定义矩形类

public class Rectangle:Shape

{

protected int \_a;

protected int \_b;//矩形边长

public Rectangle(int a,int b )

{

\_a = a;

\_b = b;

}

//重载虚方法，计算矩形面积

public override int GetArea()

{

return \_a \* \_b;

}

//重载虚方法，绘制矩形

public override void Draw()

{

Console.WriteLine("\*\*\*\*\*\*");

Console.WriteLine("\* \*");

Console.WriteLine("\* \*");

Console.WriteLine("\*\*\*\*\*\*");

}

}

//定义正方形类

public class Square:Rectangle

{

public Square(int a) : base(a, a) { }

//重载，绘制正方形

public override void Draw()

{

Console.WriteLine("\*\*\*\*\*\*");

Console.WriteLine("\* \*");

Console.WriteLine("\* \*");

Console.WriteLine("\* \*");

Console.WriteLine("\* \*");

Console.WriteLine("\*\*\*\*\*\*");

}

}

}

##### 例16-3.3 Triangle.cs

//Triangle.cs //定义三角形

using System;

namespace zh.MyShape

{

//定义普通三角形类，作为其它三角形的基类

public class Triangle:Shape

{

protected int \_a;

protected int \_b;

protected int \_c;

public Triangle(int a, int b,int c)

{

\_a = a;

\_b = b;

\_c = c;

}

//重载虚方法，计算矩形面积

public override int GetArea()

{

int s = (\_a + \_b + \_c) / 2;

int area = (int)(Math.Sqrt(s \* (s - \_a) \* (s - \_b) \* (s - \_c)));

return area;

}

}

//定义直角三角形

public class RectTriangle:Triangle

{

new protected int \_a;

new protected int \_b;

public RectTriangle(int a,int b):base(a,b,(int)(Math.Sqrt(a\*b +b\*b)))

{

\_a=a;

\_b=b;

}

public override int GetArea()

{

return (int)(\_a \* \_b / 2);

}

public override void Draw()

{

Console.WriteLine("\*");

Console.WriteLine("\* \*");

Console.WriteLine("\* \*");

Console.WriteLine("\* \*");

Console.WriteLine("\*\*\*\*\*\*\*\*");

}

}

//定义等腰直角三角形

public class RectEqualTriangle : RectTriangle

{

new protected int \_a;

public RectEqualTriangle(int a)

: base(a, a)

{

\_a = a;

}

public override int GetArea()

{

return (int)(\_a \* \_a / 2);

}

public override void Draw()

{

Console.WriteLine("---RectEqualTriangle---");

Console.WriteLine("\*");

Console.WriteLine("\* \*");

Console.WriteLine("\* \*");

Console.WriteLine("\* \*");

Console.WriteLine("\*\*\*\*\*\*\*\*");

}

}

}

##### 例16-3.4 MyMessage.cs

//MyMessage.cs //定义程序显示

using System;

namespace zh.MyMessage

{

//定义矩形类

public class Message

{

public void Begin()

{

Console.WriteLine("---Message---");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("\* \* \* \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\* \* \* \*\*\*\*\*\*\*\*\*");

Console.WriteLine("\* \* \* \* \* \*");

Console.WriteLine("\* \* \* \* \* \*");

Console.WriteLine("\* \* \* \* \* \*");

Console.WriteLine("\* \* \* \* \* \*");

Console.WriteLine("\* \* \* \* \*");

Console.WriteLine("\* \* \* \*");

Console.WriteLine("\* \* \* \*");

Console.WriteLine("\* \* \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

public bool Ask()

{

Console.WriteLine("Press 0 to exit the game");

Console.WriteLine("Press any other key to continue the game");

Console.WriteLine();

int c = Console.Read();

if (c == 48)

return false;

return true;

}

}

}

##### 例16-3.5 Client.cs

//Client.cs 客户程序

using System;

using zh.MyShape;

using zh.MyMessage;

namespace zh.Client

{

//定义矩形类

class ClientTest

{

public static void Main()

{

int \_Score = 1000;//总分

int \_Win;//每一局赢取的分数

int \_Choice;//随机获得的图形号

int \_Bet;//每一局的下注

string \_s;

Shape sp = new Shape();

Random ran = new Random();

Message msg = new Message();

msg.Begin();

while(true)

{

if (!msg.Ask())

break;

Console.WriteLine("Your Score:{0}",\_Score);

Console.WriteLine("Enter Your bet");

\_s = Console.ReadLine();

//如果押注的输入不正确，进行异常处理，并默认下注100分

try

{

\_Bet = int.Parse(\_s);

}

catch (Exception)

{

\_Bet = 100;

}

if (\_Bet < \_Score)

\_Score -= \_Bet;

else

{

\_Bet = \_Score;

\_Score = 0;

}

Console.WriteLine("Remain Score:{0}",\_Score);

\_Win = 0;

for (int i = 0; i < 3; i++)

{

\_Choice = ran.Next() % 4;//随机发生器

switch (\_Choice)

{

case 0:

sp = new RectTriangle(5, 4);

goto end;

case 1:

sp = new RectEqualTriangle(5);

goto end;

case 2:

sp = new Rectangle(5, 4);

goto end;

case 3:

sp = new Square(5);

goto end;

default:

break;

}

end:

//利用多态性，计算得分

sp.Draw();

\_Win += sp.GetArea() \* (i + 1) \* \_Bet / 100;

Console.WriteLine("Your win:{0}", \_Win);

}

\_Score += \_Win;

Console.WriteLine("Your Score:{0}", \_Score);

if (\_Score<100)

{

Console.WriteLine("Your remain Score is not enough to play");

break;

}

}

}

}

}

##### 例16-3.6编译脚本

LoadShape.bat

@echo off

rem if exist .\WelcomeApp.exe del .\WelcomeApp.exe

rem if exist .\WelcomeMessage.dll del .\WelcomeMessage.dll

rem ---0、直接产生test.exe

rem csc test.cs

rem ---1-1、产生 MyShape.dll、MyMessage.dll 类库

csc /target:library /out:MyShape.dll MyShape.cs Rect.cs Triangle.cs

csc /target:library /out:MyMessage.dll MyMessage.cs

rem ---1-2、产生Client.exe应用程序

csc /reference:MyShape.dll;MyMessage.dll Client.cs

Client.exe

pause

#### 16.5小结

### 第十七章 文件操作

### 第十八章 高级话题

# C#Note

## 参考文档

### N01.C#文档中文版(微软)

### N02.C#基础教程\_温故而知新

### N03.微软C# 语言规范

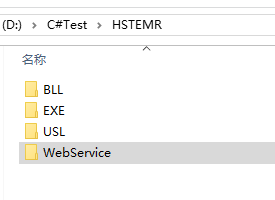
### N04.微软C#完全手册

### N05.C#高级编程(第7版)

## 创建解决方案

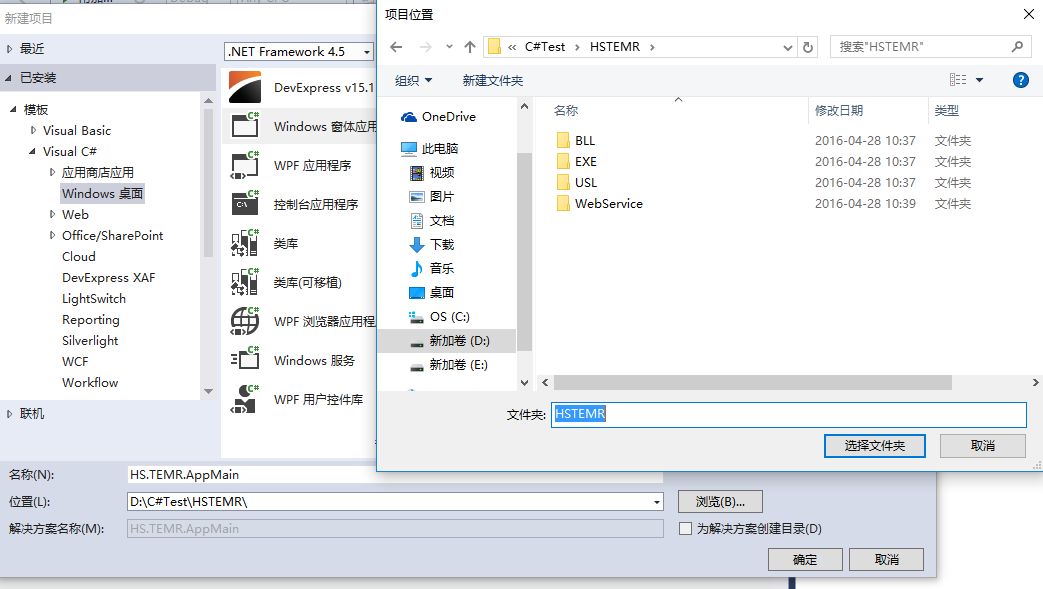
### 创建文件目录

D:\C#Test\HSTEMR包括以下 BLL、EXE、USL、WebService 文件夹



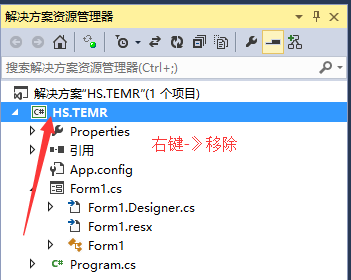
### 新建解决方案

HS.TEMR【文件-》新建-》解决方案】

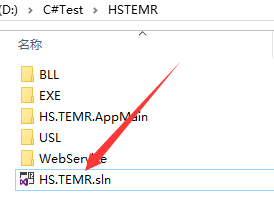


### 移除项目

生成空解决方案



#### 剪切HS.TEMR.sln到主目录HSEMR



### 添加新项目

主要包括 主窗体、BLL、USL、WebService、User\*

AppMain类型：Window桌面-》Windows桌面应用程序

其它类型：Window桌面-》类库

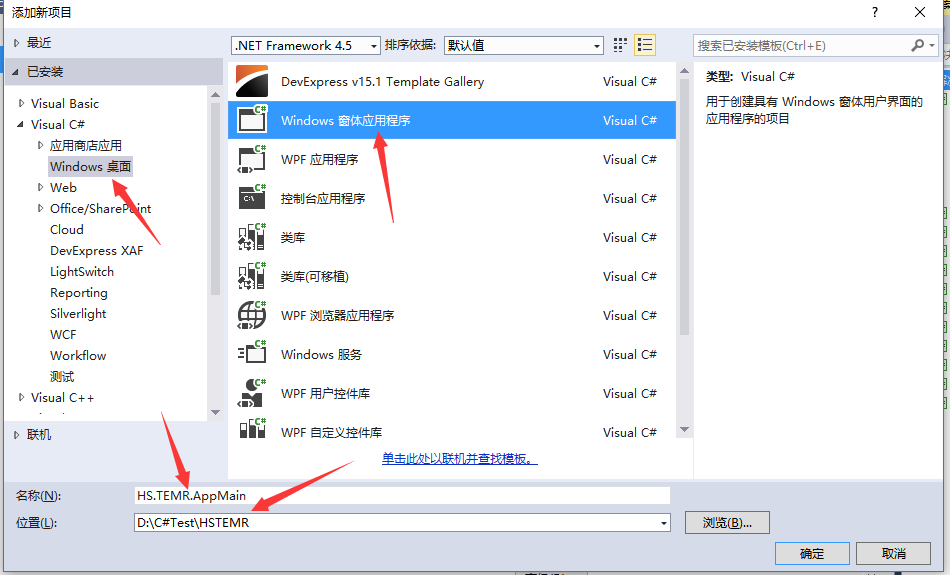
主窗体通常保存到根目录、

HS.TEMR.BILL.\*逻辑层：保存到

HS.TEMR.USL.\*展示层 保存到BLL

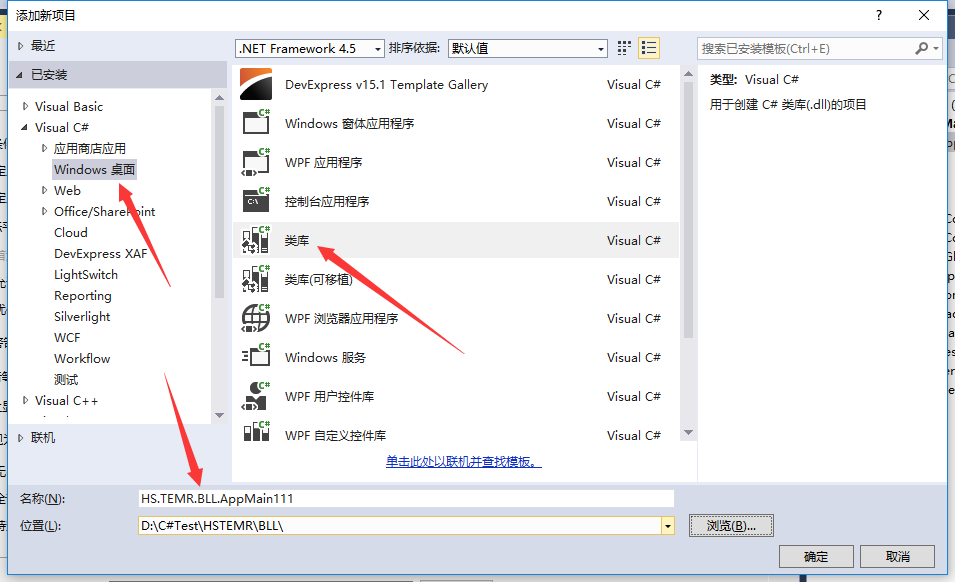
添加项目HS.TEMR.AppMain

【文件->添加->添加新项目】



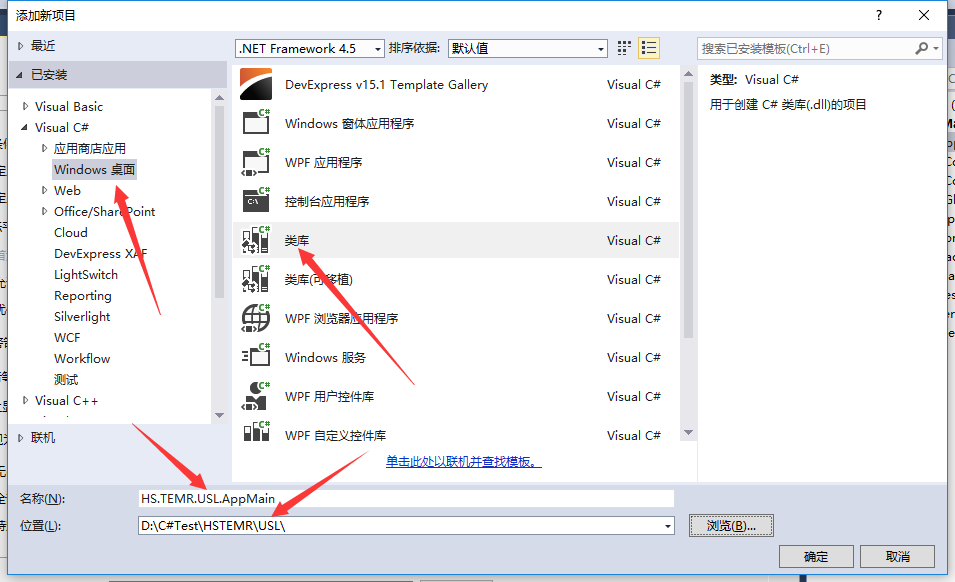
#### 添加类库HS.TEMR.BLL.AppMain

【文件->添加->添加新项目】



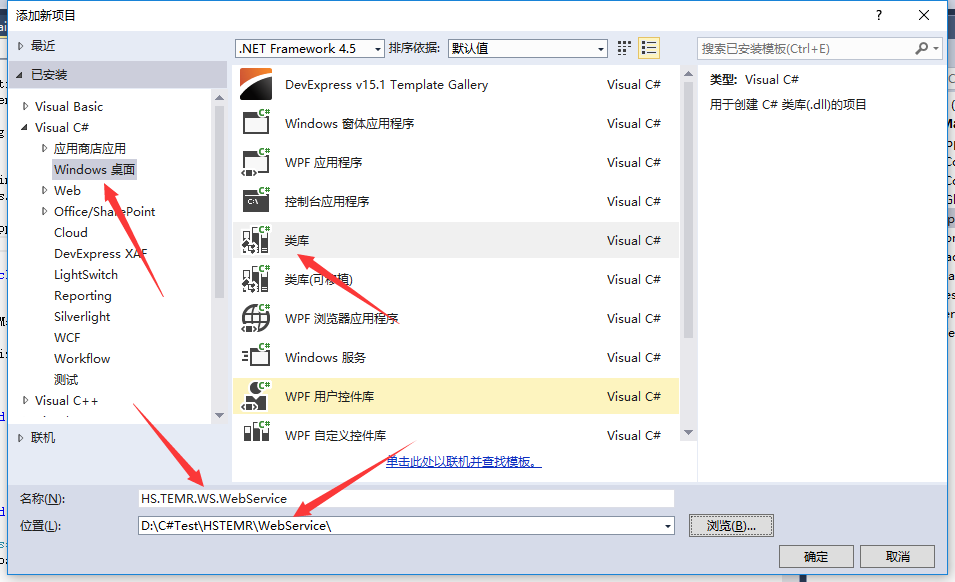
#### 添加类库HS.TEMR.USL.AppMain

【文件->添加->添加新项目】



#### 添加类库HS.TEMR.WS.WebService

【文件->添加->添加新项目】

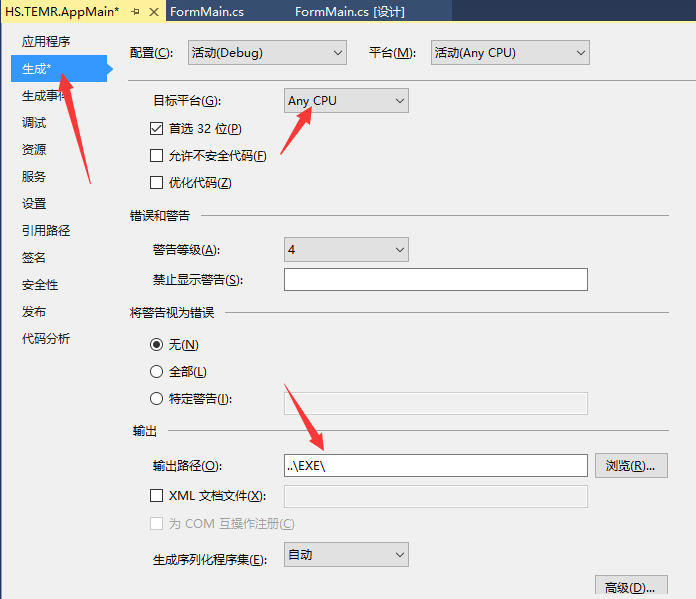


### 设置项目属性

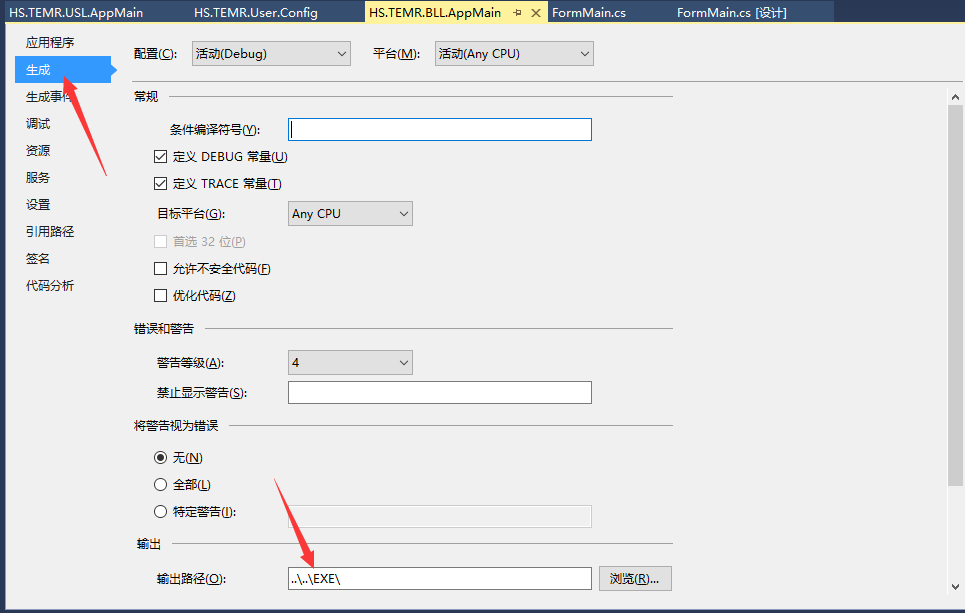
HS.TEMR.WS.WebService输出到..\bin\

HS.TEMR.AppMain.. 输出到..\..\exe\

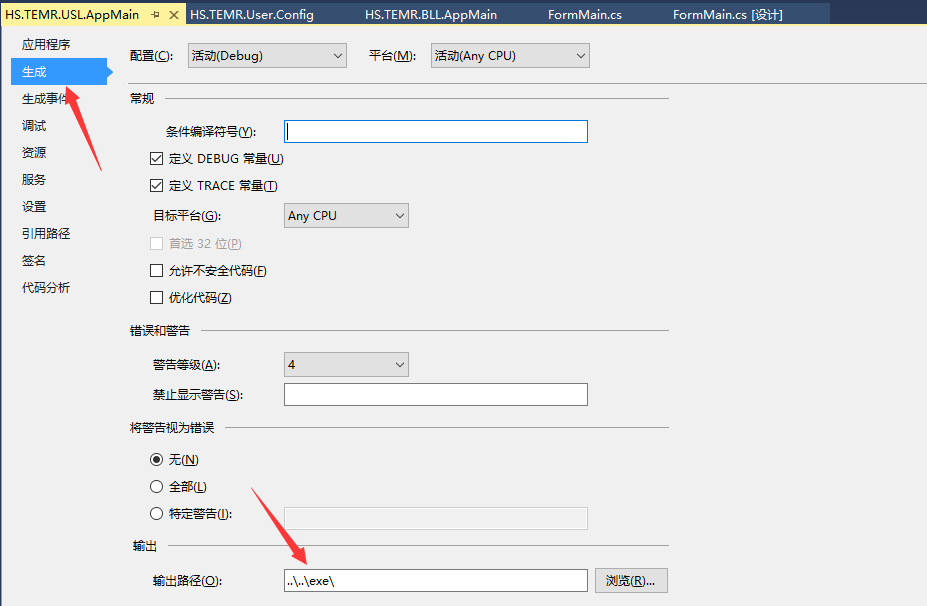
设置HS.TEMR.AppMain属性



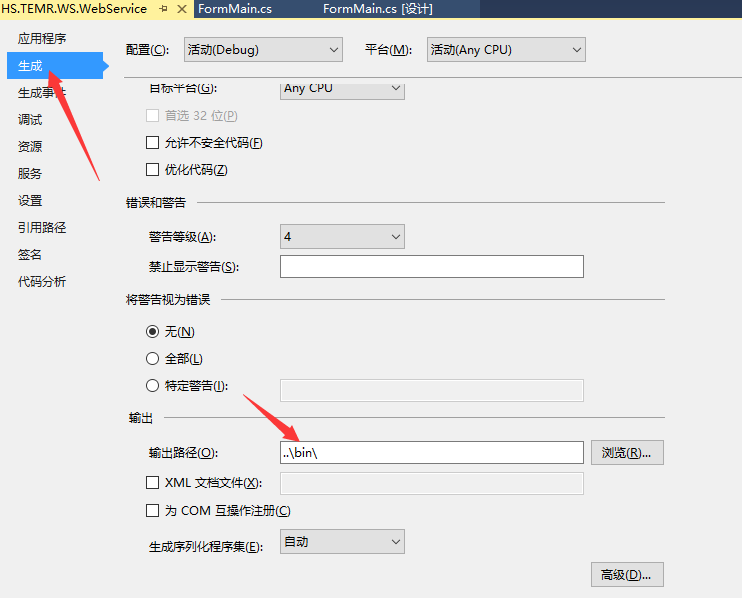
#### 设置HS.TEMR.BLL.AppMain属性



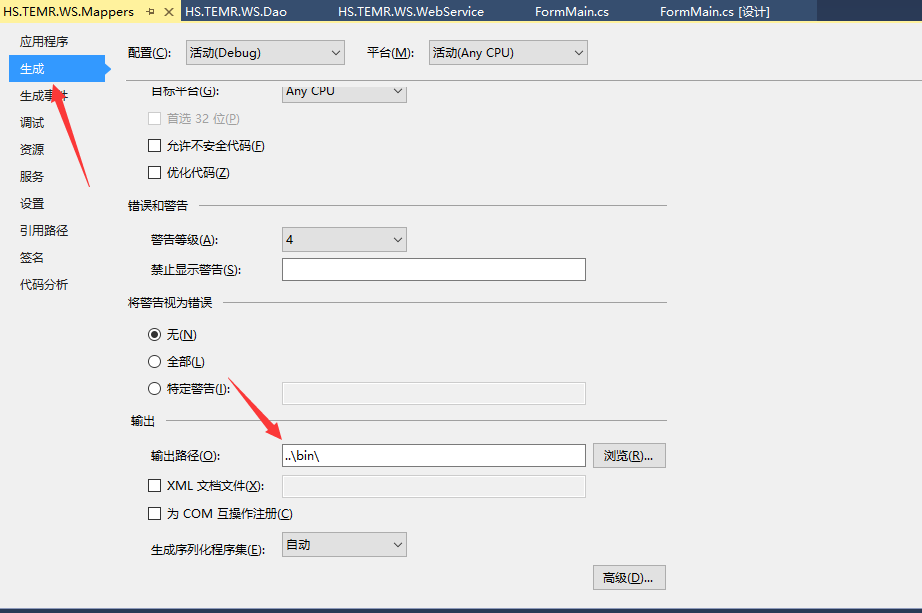
#### 设置HS.TEMR.USL.AppMain属性



#### 设置HS.TEMR.WS.WebService属性



#### 其它..WS.\*属性设置



## 窗体居中

private void FormAppMain\_Load(object sender, EventArgs e)

{

//this.StartPosition = CenterScreen

this.Width = 1003;

this.Height = 551;

this.Top = (Screen.PrimaryScreen.Bounds.Height - this.Height)/2;

this.Left = (Screen.PrimaryScreen.Bounds.Width - this.Width) / 2;

}

## 退出系统

### FormClosing

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

DialogResult result = MessageBox.Show("将要关闭系统，确定吗？", "提示信息", MessageBoxButtons.OKCancel, MessageBoxIcon.Question);

e.Cancel = (result == DialogResult.Cancel);

}

### ButtonClick

private void simpleButton关闭\_Click\_1(object sender, EventArgs e)

{

this.Close();

}

## 循环遍历文档

/// <summary> 遍历文件

/// 遍历文件日志目录，生成日志列表

/// </summary>

private void TraversalFile()

{

this.listBox1.Items.Clear();

String sPath = System.Environment.CurrentDirectory + "\\" + textBoxPath.Text + "\\log";

//判断指定路径的文件夹是否存在

if (Directory.Exists(@sPath))

{

DirectoryInfo theFolder = new DirectoryInfo(@sPath);

DirectoryInfo[] dirInfo = theFolder.GetDirectories();

FileInfo[] fileInfo = theFolder.GetFiles();

foreach (FileInfo NextFile in fileInfo)  //遍历文件

this.listBox1.Items.Add(NextFile.Name);

}

else

MessageBox.Show("指定的路径不存在：" + sPath);

}

## 读取文档内容

/// <summary> 读取指定文档

/// 读取指定文档内容，显示到textbox

/// </summary>

private void ReadLog(string sFileName)

{

String sFile = System.Environment.CurrentDirectory + "\\"+ textBoxPath.Text +"\\log\\"+sFileName;

textBox1.Text = sFile;

if (File.Exists(@sFile))

{

StreamReader sr = new StreamReader(sFile, Encoding.Default);

String line;

while ((line = sr.ReadLine()) != null)

{

textBox1.Text += Environment.NewLine + line.ToString();

}

sr.Close();

}

else

MessageBox.Show("指定的文件不存在:" + sFile);

}

## IIS设置

控制面板\所有控制面板项\程序和功能=>启用/关闭Win功能

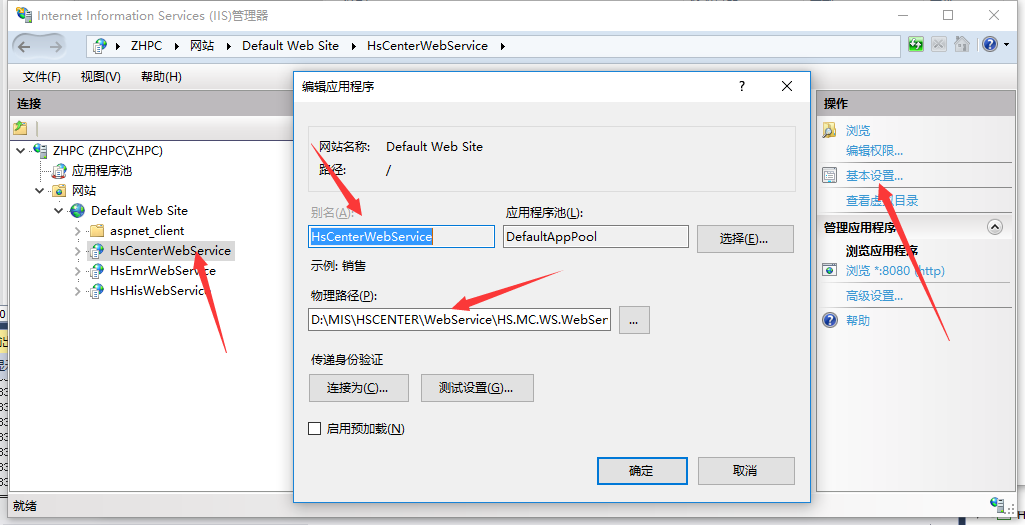
控制面板-》管理工具-》Internet Information Services (IIS)管理器

### 常用IIS配置

1、HsCenterWebService D:\MIS\HSCENTER\WebService\HS.MC.WS.WebService

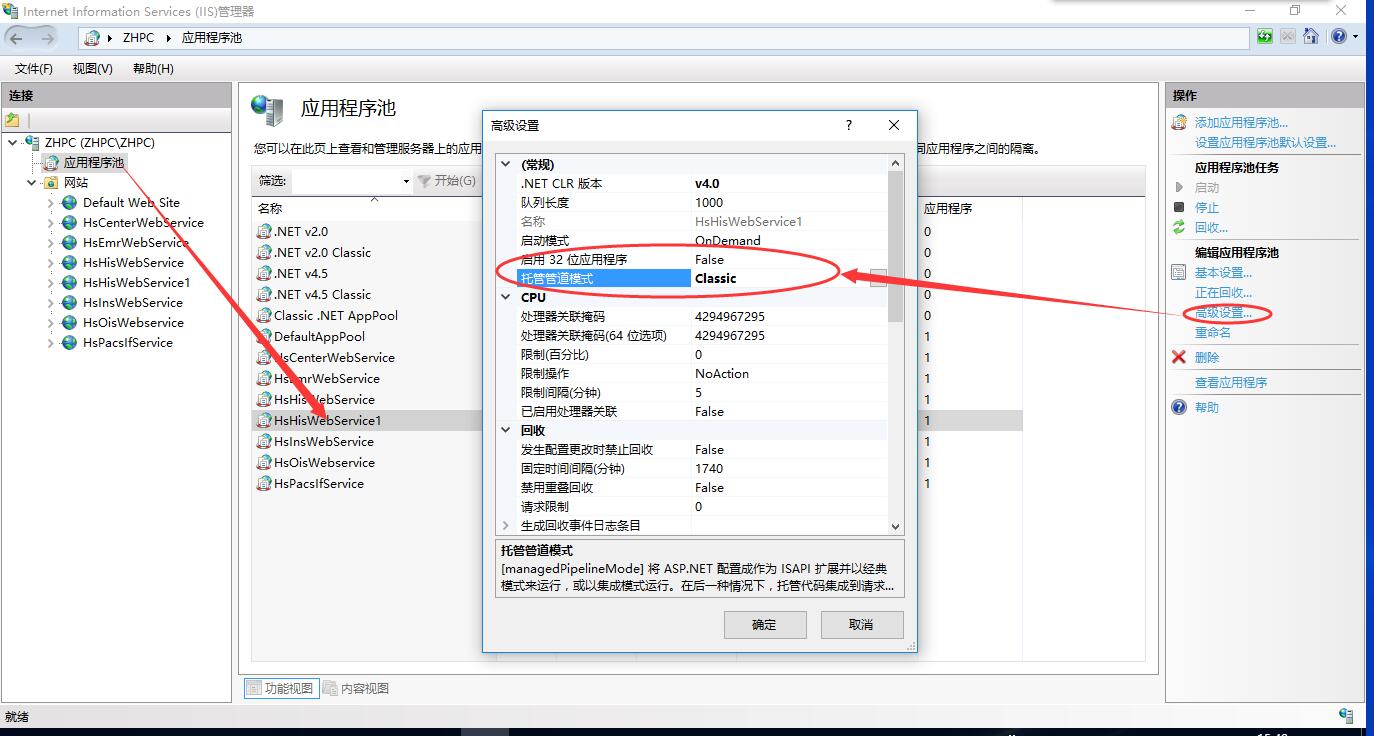
2、HsEmrWebService D:\MIS\HSEMR\WebService\HS.EMR.WS.WebService

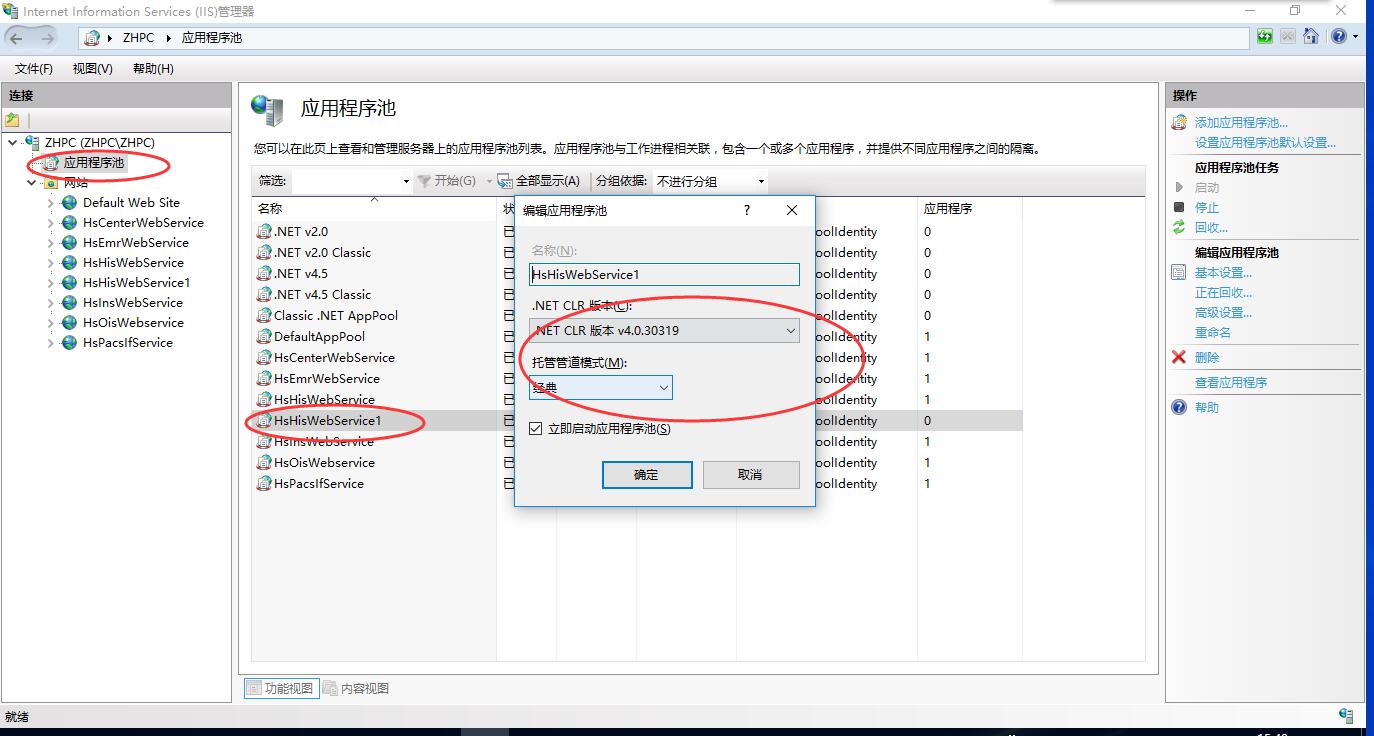
3、HsHisWebService D:\MIS\HSHIS5\HS.HIS.WebService\HS.HIS.WS.WebService



### 新建IIS【Default Web Site->右键->添加应用程序】

#### 





## gridControl 用法

### 常用属性

gridView1.OptionsBehavior.Editable = false;//设为只读

gridView1.OptionsView.ShowGroupPanel = false;//不显示分组面板

gridView1.Columns[0].Caption = "工号";//列名称

gridView1.Columns[0].FieldName = "ID";//列字段

gridView1.Columns[0].AppearanceCell.TextOptions.HAlignment = CenterToParent;//字段居中

gridView1.Columns[0].AppearanceHeader.TextOptions.HAlignment = CenterToParent;//标题居中

### gridView1关联DataSet

private void simpleButton调阅worker\_Click\_1(object sender, EventArgs e)

{

pacs.CommonService service = new pacs.CommonService();

pacs.MySoapHeader header = new pacs.MySoapHeader();

header.UserName = "hsemr";

header.PassWord = "OSBMz5dNMfltPDifRHJw+MEzAPZHKI";

header.HospId = "H1";

header.DeptId = "0000001528";

header.WorkerId = "001001";

service.MySoapHeaderValue = header;

string ds = service.GetTestWorkerXml(ref this.outRet, ref this.outMsg);

MessageBox.Show(this.outRet.ToString() + "\_" + this.outMsg);

if (this.outRet == 0)

{

textBox2.Text = ds;

}

//得到Worker信息

using (DataSet \_ds = service.GetTestWorkerForTreeList(ref outRet, ref outMsg))

{

if (outRet == 0)

{

this.gridControl1.DataSource = \_ds.Tables[0];

}

}

}

### gridView1读取字段值

private void gridView1\_Click(object sender, EventArgs e)

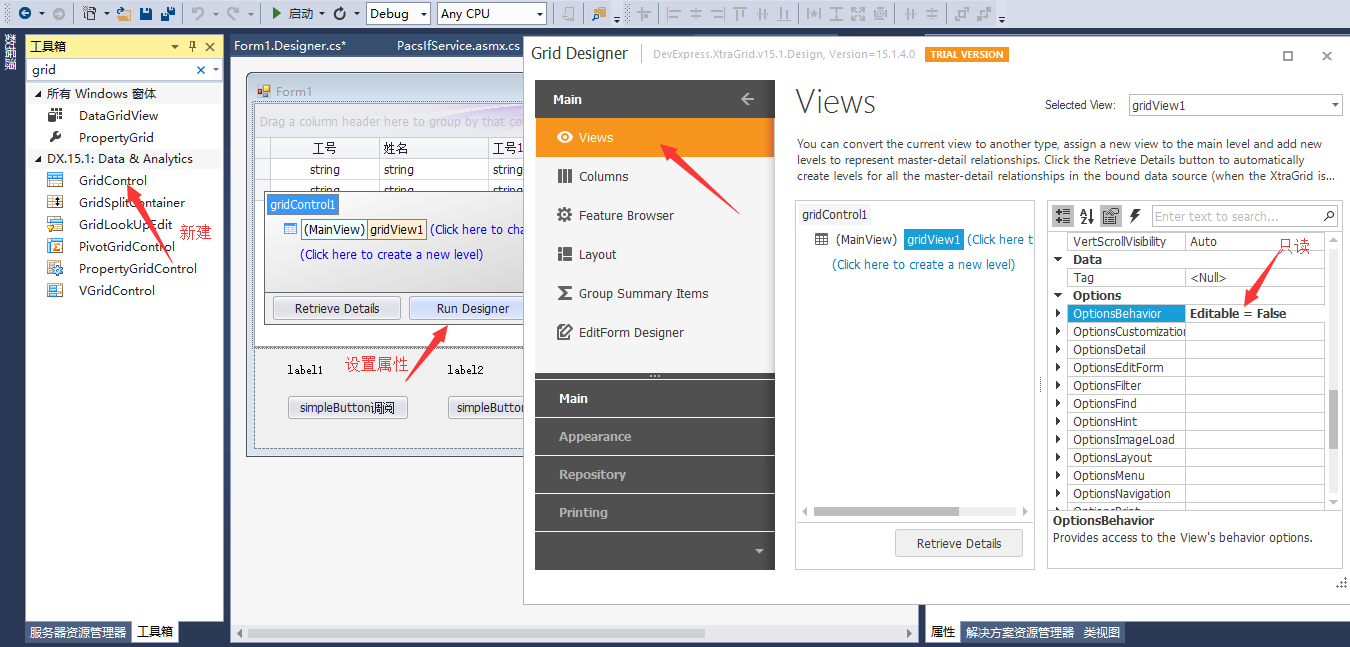
{

label1.Text = this.gridView1.GetFocusedDisplayText();//所选列的值

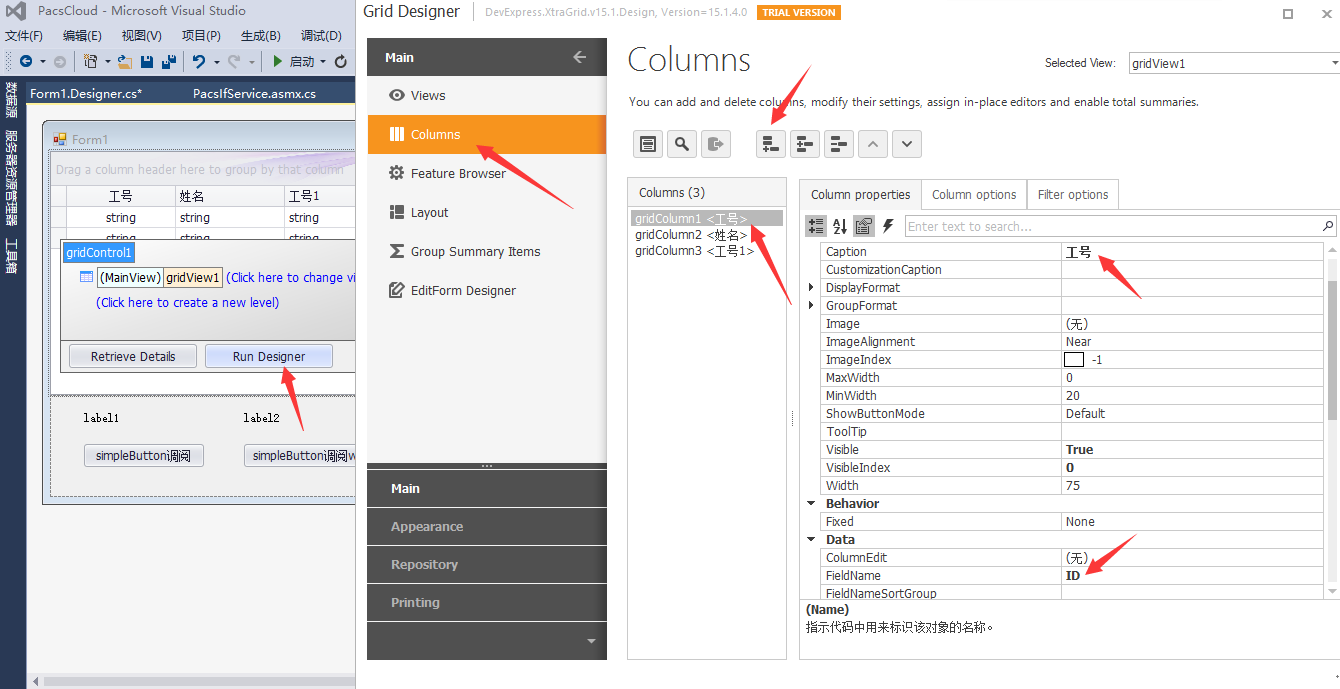
label2.Text = this.gridView1.GetFocusedDataRow()["DEPTID"].ToString();//指定列值

}

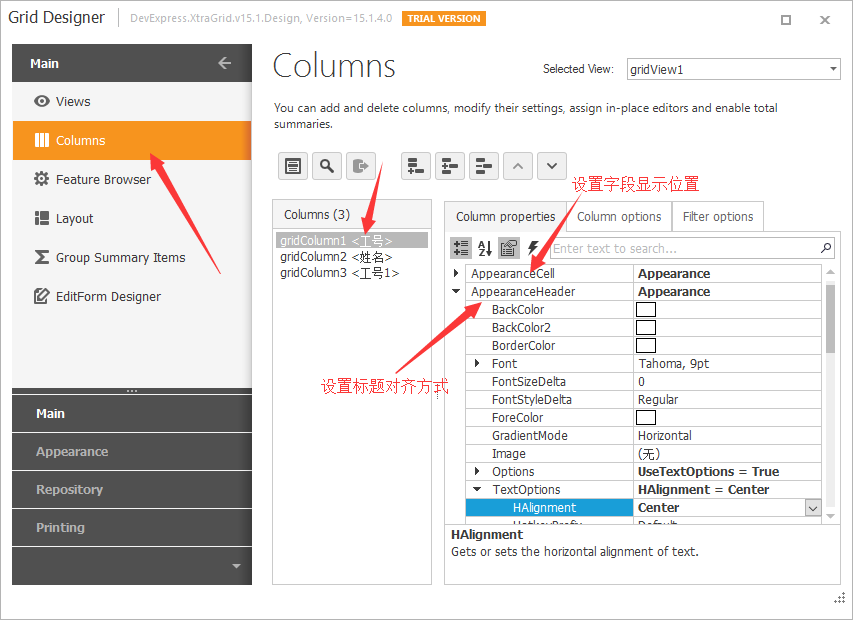
### 添加、设置为只读



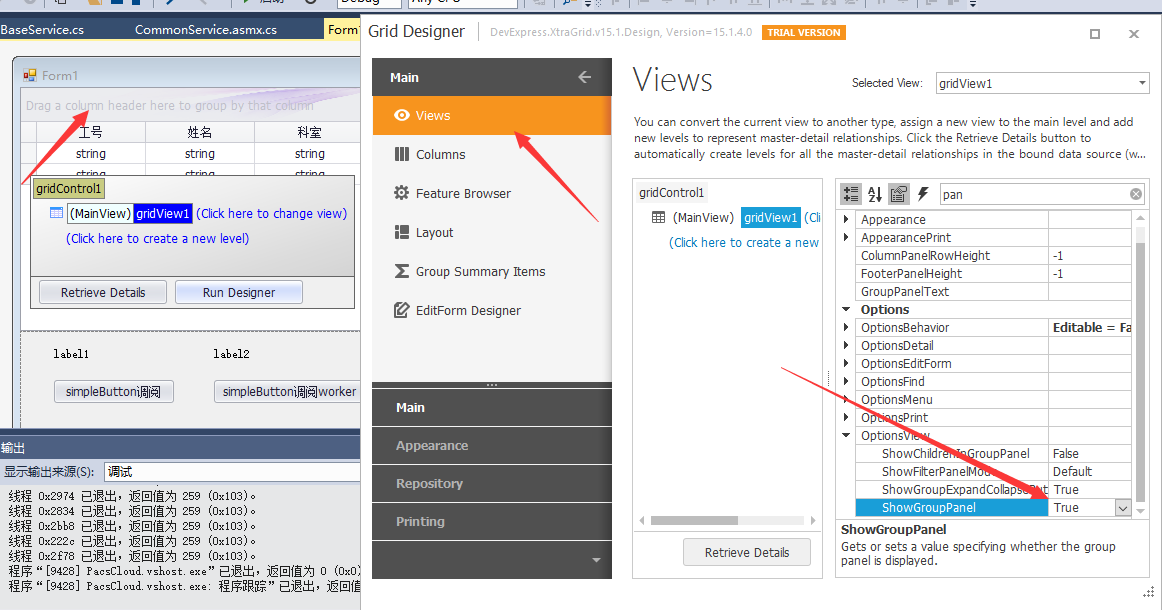
### 绑定字段



### 字段设置标题、内容对齐方式



### GridView显示隐藏分组栏



## 添加外部web引用

### 0、文档资料

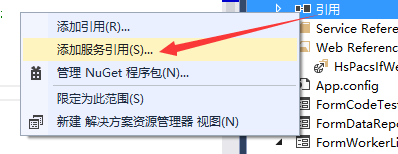
url: <http://222.216.218.90:8090/DTC_WebservicesXML/service/WebServices?wsdl>

方法：sjxz

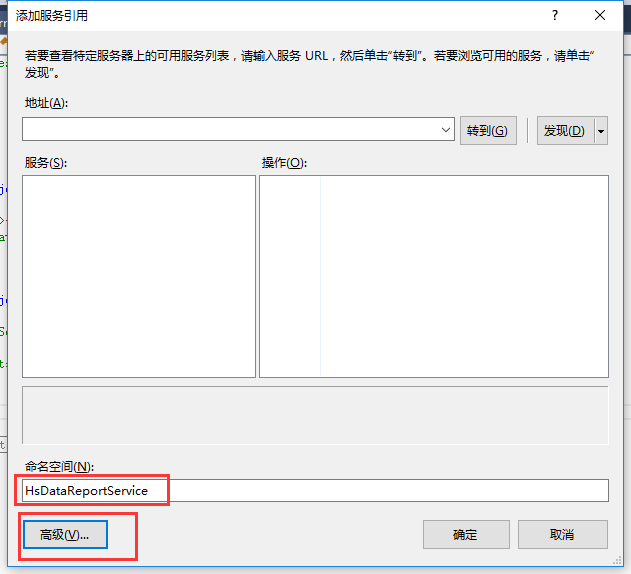
入参：<cx><id>22540</id><jystr>icd10</jystr></cx>

出参：<ID>22540</ID><ICDBM>N25.900</ICDBM><BZMS>肾小管功能损害所致疾病</BZMS><ID>22541</ID><ICDBM>N27.000</ICDBM><BZMS>单侧小肾</BZMS><ID>22542</ID><ICDBM>N28.000</ICDBM><BZMS>肾缺血和肾梗死</BZMS></Record>

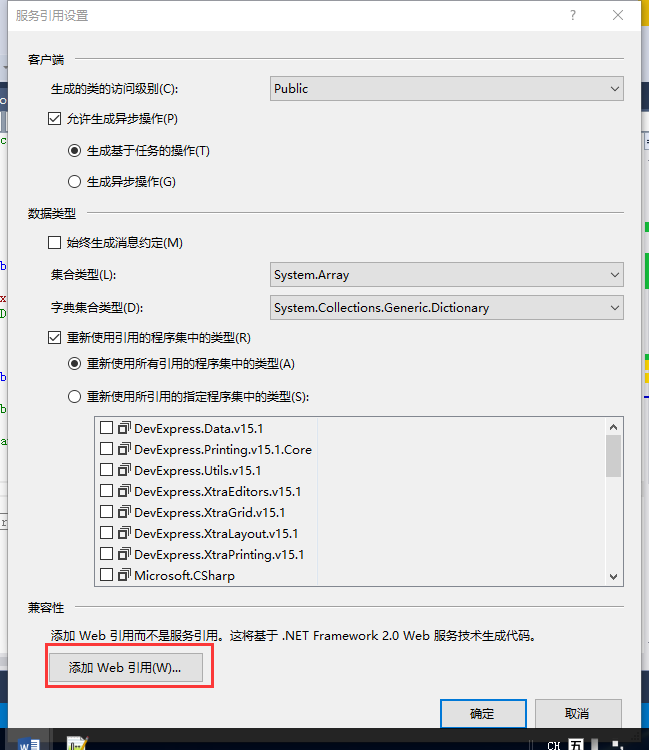
### 1、添加web引用



### 2、选择“高级”



### 3、继续



### 4、继续



### 5、代码调用

private void button2\_Click(object sender, EventArgs e)

{

button2.Enabled = false;

HsDataReportService.WebServices DataReport = new HsDataReportService.WebServices();

this.textBox1.Text = "<cx><id>22540</id><jystr>icd10</jystr></cx>";

this.textBox2.Text = DataReport.sjxz(this.textBox1.Text);

button2.Enabled = true;

}

## [获取方法所在的命名空间 类名 方法名](http://www.cnblogs.com/runliuv/archive/2012/08/06/2625310.html)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Diagnostics;

using System.Reflection;

namespace GetMethodNameSpace

{

class Program

{

public static string GetMethodInfo()

{

string str = "";

//取得当前方法命名空间

str += "命名空间名:" + System.Reflection.MethodBase.GetCurrentMethod().DeclaringType.Namespace + "\n";

//取得当前方法类全名

str += "类名:" + System.Reflection.MethodBase.GetCurrentMethod().DeclaringType.FullName + "\n";

//取得当前方法名

str += "方法名:" + System.Reflection.MethodBase.GetCurrentMethod().Name + "\n";

str += "\n";

StackTrace ss = new StackTrace(true);

MethodBase mb = ss.GetFrame(1).GetMethod();

//取得父方法命名空间

str += mb.DeclaringType.Namespace + "\n";

//取得父方法类名

str += mb.DeclaringType.Name + "\n";

//取得父方法类全名

str += mb.DeclaringType.FullName + "\n";

//取得父方法名

str += mb.Name + "\n";

return str;

}

public static void Main()

{

Console.WriteLine(GetMethodInfo());

Console.ReadKey();

}

}

}

## 输出语句

Console.WriteLine(s);

## 输入语句

string str=Console.ReadLine();

## C#命令行编译器 Csc \*.cs

cmd

@echo off

if exist .\test.exe del .\test.exe

rem ---以下命令产生test.dll类库

rem csc /target:library test.cs

rem ---以下命令产生test.exe应用程序

rem csc /reference:test.dll test.cs

rem ---直接产生test.exe

rem csc test.cs

rem >log.log

test.exe

rem >>log.log

if exist .\test.exe del .\test.exe

pause

### Test.cs:

using System;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

Console.WriteLine("Hello world !");

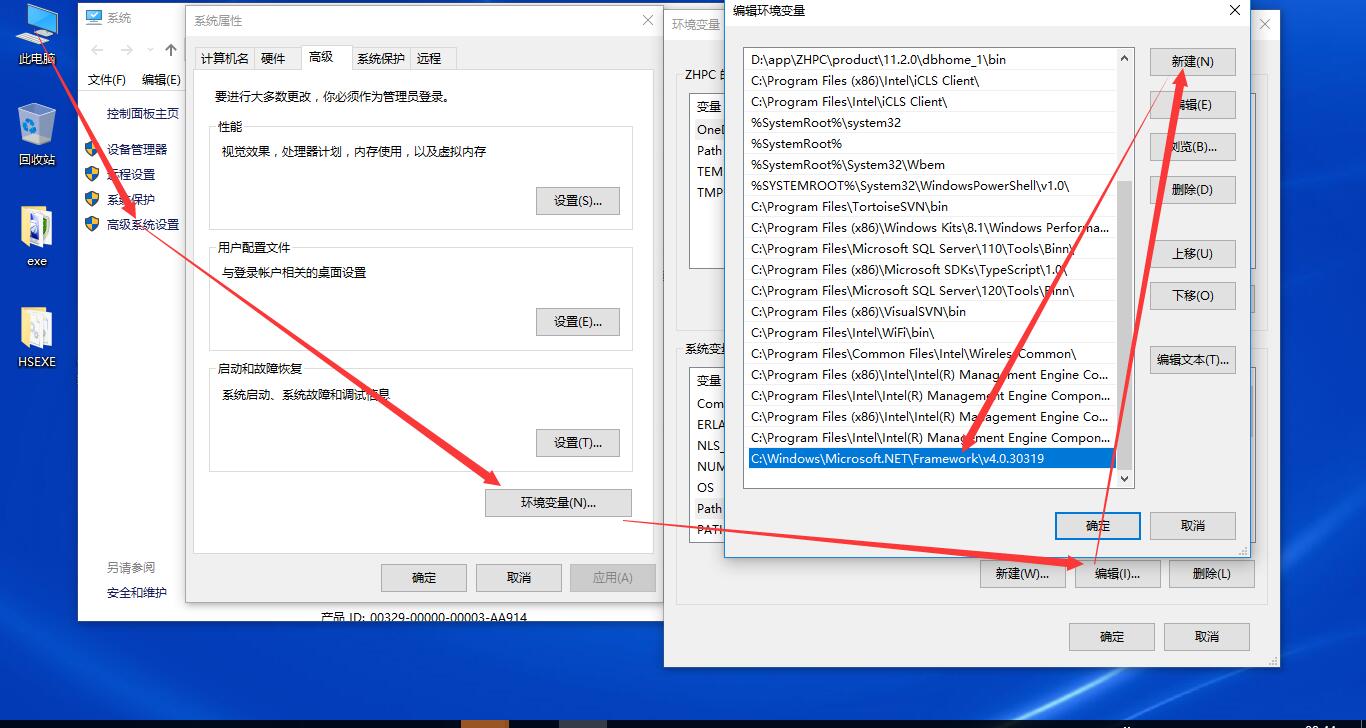
}

}

}

### “csc不是内部或外部命令“错误处理

环境:win10 vs2013 C:\Windows\Microsoft.NET\Framework64\v4.0.30319



## 占位符{}

using System;

namespace hs.test

{

class MyFirstClass

{

static void Main()

{

int h=1, j=3, k = 5;

Console.WriteLine("h的值：{0},j的值：{1},k的值：{2}" , h.ToString(), j.ToString(), k.ToString());

}

}

}

## 介绍

### 数据类型

#### 变量-类型推断

using System;

namespace zh.test

{

class MyTestClass

{

static void Main()

{

var Name = "Tom";

var Age = 25;

var IsRabbit = true;

Type NameType = Name.GetType();

Type AgeType = Age.GetType();

Type IsRabbitType = IsRabbit.GetType();

Console.WriteLine("Name is type {0}",NameType.ToString());

Console.WriteLine("Age is type {0}", AgeType.ToString());

Console.WriteLine("IsRabbit is type " + IsRabbitType.ToString());

Console.WriteLine("Well come!!");

//System.Console.ReadLine();

return;

}

}

}

#### 变量的对比或比较==

using System;

namespace zh.test

{

class MyTestClass

{

static void Main()

{

String s = "Test";

String t = String.Copy(s);

Console.WriteLine(s==t);

Console.WriteLine((object)s == (object)t);

Console.WriteLine(s.GetType().ToString());

Console.WriteLine(t.GetType().ToString());

return;

}

}

}

#### 类型转换

隐式转换

using System;

namespace zh.test

{

class MyTestClass

{

static void Main()

{

int i = 123;

long L = i;

Console.WriteLine("(long){0}={1}",i,L);

return;

}

}

}

显式转换

using System;

namespace zh.test

{

class MyTestClass

{

static void Main()

{

long L = System.Int64.MaxValue;

int i = (int)L;

Console.WriteLine("(int){0}={1}",L,i);

return;

}

}

}

#### 数组类型

using System;

namespace zh.test

{

class MyTestClass

{

static void Main()

{

int[] a1 = new int[] { 1, 2, 3 };

int[,] a2 = new int[,] { { 1, 2, 3 }, { 4, 5, 6 } };

int[, ,] a3 = new int[10, 20, 30];

int[][] j2 = new int[3][];

j2[0] = new int[] { 1,2,3};

j2[1] = new int[] { 1,2,3,4,5,6};

j2[2] = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

int[] arr = new int[5];

for (int i = 0; i < arr.Length; i++)

arr[i] = i \* i;

for (int i = 0; i < arr.Length; i++)

Console.WriteLine("arr[{0}]={1}", i,arr[i]);

return;

}

}

}

#### 装箱拆箱

using System;

namespace zh.test

{

class MyTestClass

{

static void Main()

{

Console.WriteLine("3.ToString={0}", 3.ToString());

int i = 123;

Console.WriteLine("i.ToString={0}", i.ToString());

object o = i;//boxing

Console.WriteLine("o.ToString={0}", o.ToString());

int j = (int)o;//unboxing

Console.WriteLine("j.ToString={0}", j.ToString());

return;

}

}

}

#### Char

using System;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

string s;

char c = 'a';

Console.WriteLine(c);

c = '\x0032';//'\x'十六进制转译符

Console.WriteLine(c);

c = '\u0032';//'\u'unicode表示法

Console.WriteLine(c);

}

}

}

using System;

using System.Text;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

string ss="abcd";

char[] cc = ss.ToCharArray();

//string 转换成 Char[]

Console.WriteLine("输出char[] cc");

Console.WriteLine(cc);

//Char[] 转换成string

string s = new string(cc);

Console.WriteLine("输出string s:" + s);

//byte[] 与 string 之间的装换

byte[] bb = Encoding.UTF8.GetBytes(ss);

Console.WriteLine("输出byte bb");

Console.WriteLine(bb.ToString());

s = Encoding.UTF8.GetString(bb);

Console.WriteLine("输出s:" + s);

string strOr = "OR";

string result = string.Concat(new string[] { " A ", strOr, " B ", Environment.NewLine, " C ", strOr, " D " });

Console.WriteLine("输出result:" + result);

}

}

}

#### 泛型List<int>

List<int> ListInt = new List<int>()

using System;

using System.Text;

using System.Collections;

using System.Collections.Generic;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

string ss="abcd";

char[] cc = ss.ToCharArray();

//string 转换成 Char[]

Console.WriteLine("输出char[] cc");

Console.WriteLine(cc);

//Char[] 转换成string

string s = new string(cc);

Console.WriteLine("输出string s:" + s);

//byte[] 与 string 之间的装换

byte[] bb = Encoding.UTF8.GetBytes(ss);

Console.WriteLine("输出byte bb");

Console.WriteLine(bb.ToString());

s = Encoding.UTF8.GetString(bb);

Console.WriteLine("输出s:" + s);

string strOr = "OR";

string result = string.Concat(new string[] { " A ", strOr, " B ", Environment.NewLine, " C ", strOr, " D " });

Console.WriteLine("输出result:" + result);

List<int> ListInt = new List<int>();

Console.WriteLine("输出List<int> ListInt:");

for (int i = 0; i < 4; i++)

{

ListInt.Add(i);

//Console.WriteLine(ListInt[i]);

}

ListInt[1] = 10;

ListInt.Add(456);

ListInt.Add(457);

ListInt.RemoveAt(2);

foreach (int i in ListInt)

{

Console.WriteLine(i);

}

}

}

}

### 变量和参数

#### 值类型的入参

using System;

namespace zh.test

{

class MyTestClass

{

static void F(int p)

{

Console.WriteLine("p={0}", p);

p++;

}

static void Main()

{

int i = 123;

Console.WriteLine("pro={0}", i);

F(i);

Console.WriteLine("post={0}", i);

return;

}

}

}

#### 引用类型的入参ref

using System;

namespace zh.test

{

class MyTestClass

{

static void F(ref int a,ref int b)

{

int t = a;

a = b;

b = t;

}

static void Main()

{

int x = 20;

int y = 30;

Console.WriteLine("pro:x={0},y={1}", x,y);

F(ref x,ref y);

Console.WriteLine("post:x={0},y={1}", x, y);

return;

}

}

}

#### 出参out

using System;

namespace zh.test

{

class MyTestClass

{

static void F( int a, int b,out int result)

{

result = a \* b;

}

static void Main()

{

for (int i = 0; i < 10; i++)

for (int j = 0; j < 10; j++)

{

int r;

F(i, j, out r);

Console.WriteLine("{0}\*{1}={2}",i,j,r);

}

return;

}

}

}

#### 参量参数params

using System;

namespace zh.test

{

class MyTestClass

{

static void F( int a, params int[]args)

{

Console.WriteLine("No:{0} of arguments: {1}",a, args.Length);

for (int i = 0; i < args.Length; i++)

Console.WriteLine("\targs[{0}] = {1}", i, args[i]);

}

static void Main()

{

F(1,1);

F(2,1,2);

F(3,1,2,3);

F(4,new int[] { 1, 2, 3, 4 });

return;

}

}

}

### 表达式

#### 一元操作符

#### 二元操作符

#### 三元操作符

#### 基本的

(x) x.y f(x) a[x] x++ x-- new

typeof sizeof checked unchecked

#### 一元的

+ - ! ~ ++x --x (T)x

#### 乘法的

\* / %

#### 加法的

+ -

#### 移位

<< >>

#### 关系

< > <= >= is

#### 等式

== !=

#### 逻辑与

&

#### 逻辑异或

^

#### 逻辑或

|

#### 条件与

&&

#### 条件或

||

#### 条件的

?:

#### 赋值

= \*= /= %= += -= <<= >>= &= ^= |=

### 声明

参考C#文档中文版(微软).pdf类型-》声明

### 类

#### 类示例

using System;

namespace zh.test

{

class MyClass

{

public MyClass()

{

Console.WriteLine("Constructor");

}

public MyClass(int value)

{

MyField = value;

Console.WriteLine("Constructor");

}

~MyClass()

{

Console.WriteLine("Destructor");

}

public const int MyConstant = 12;

public int MyField = 34;

public void MyMethod()

{

Console.WriteLine("MyClass.MyMethod");

}

public int MyProperty

{

get{return MyField; }

set{ MyField = value; }

}

public int this[int index]

{

get{return 0;}

set{Console.WriteLine("this[{0}] was set to {1}", index, value);}

}

public event EventHandler MyEvent;

public static MyClass operator +(MyClass a, MyClass b)

{

return new MyClass(a.MyField + b.MyField);

}

internal class MyNestedClass

{ }

}

class MyTestClass

{

static void Main()

{

// Constructor usage

MyClass a = new MyClass();

MyClass b = new MyClass(123);

// Constant usage

Console.WriteLine("MyClass.MyConstant = {0}", MyClass.MyConstant);

// Field usage

a.MyField++;

Console.WriteLine("a.MyField = {0}", a.MyField);

// Method usage

a.MyMethod();

// Property usage

a.MyProperty++;

Console.WriteLine("a.MyProperty = {0}", a.MyProperty);

// Indexer usage

a[3] = a[1] = a[2];

Console.WriteLine("a[3] = {0}", a[3]);

// Event usage

a.MyEvent += new EventHandler(MyHandler);

// Overloaded operator usage

MyClass c = a + b;

}

static void MyHandler(object sender, EventArgs e)

{

Console.WriteLine("Test.MyHandler");

}

internal class MyNestedClass

{ }

}

}

#### 常数

using System;

namespace zh.test

{

class Constants

{

public const int A = 1;

public const int B = A + 1;

}

class MyTestClass

{

static void Main()

{

Console.WriteLine("A = {0}, B = {1}", Constants.A, Constants.B);

}

}

}

#### 域

using System;

namespace zh.test

{

class Color

{

internal ushort redPart;

internal ushort bluePart;

internal ushort greenPart;

public Color(ushort red, ushort blue, ushort green)

{

redPart = red;

bluePart = blue;

greenPart = green;

}

public static readonly Color Red = new Color(0xFF, 0, 0);

public static readonly Color Blue = new Color(0, 0xFF, 0);

public static readonly Color Green = new Color(0, 0, 0xFF);

public static readonly Color White = new Color(0, 0, 0);

public static readonly Color Black = new Color(0xFF, 0xFF, 0xFF);

}

class MyTestClass

{

static void Main()

{

Console.WriteLine("Red.redPart = {0}", Color.Red.redPart);

Console.WriteLine("Red.bluePart = {0}", Color.Red.bluePart);

}

}

}

#### 方法

using System;

namespace zh.test

{

class MyTestClass

{

static void F()

{

Console.WriteLine("F()");

}

static void F(object o)

{

Console.WriteLine("F(object)");

}

static void F(int value)

{

Console.WriteLine("F(int)");

}

static void F(int a, int b)

{

Console.WriteLine("F(int, int)");

}

static void F(int[] values)

{

Console.WriteLine("F(int[])");

}

static void Main()

{

F();

F(1);

F((object)1);

F(1, 2);

F(new int[] { 1, 2, 3 });

}

}

}

#### 属性

using System;

namespace zh.test

{

public class Button

{

private String \_Caption;

public String Caption

{

get { return \_Caption; }

set { \_Caption = value;}

}

}

class MyTestClass

{

static void Main()

{

Button b = new Button();

b.Caption = "ABC"; //set

b.Caption += "EDF";//get & set

String s = b.Caption;//get

System.Console.WriteLine("s={0}", s);

}

}

}

#### 事件\*?

using System;

namespace zh.test

{

public delegate void EventHandler(object sender, EventArgs e);

public class Button

{

public event EventHandler Click;

public void Reset()

{

Click = null;

}

}

public class Form1

{

public Form1()

{

// Add Button1\_Click as an event handler for Button1’s Click event

Button1.Click += new EventHandler(Button1\_Click);

}

Button Button1 = new Button();

public void Button1\_Click(object sender, EventArgs e)

{

Console.WriteLine("Button1 was clicked!");

}

public void Disconnect()

{

Button1.Click -= new EventHandler(Button1\_Click);

}

}

class MyTestClass

{

static void Main()

{

Form1 f1 = new Form1();

f1.Button1\_Click(null, null);

}

}

}

#### 操作符

using System;

namespace zh.test

{

public struct Digit

{

byte value;

public Digit(byte value)

{

if (value < 0 || value > 9) throw new ArgumentException();

this.value = value;

}

public Digit(int value) : this((byte)value) { }

public static implicit operator byte(Digit d)

{

return d.value;

}

public static explicit operator Digit(byte b)

{

return new Digit(b);

}

public static Digit operator +(Digit a, Digit b)

{

return new Digit(a.value + b.value);

}

public static Digit operator -(Digit a, Digit b)

{

return new Digit(a.value - b.value);

}

public static bool operator ==(Digit a, Digit b)

{

return a.value == b.value;

}

public static bool operator !=(Digit a, Digit b)

{

return a.value != b.value;

}

public override bool Equals(object value)

{

return this == (Digit)value;

}

public override int GetHashCode()

{

return value.GetHashCode();

}

public override string ToString()

{

return value.ToString();

}

}

class MyTestClass

{

static void Main()

{

Digit a = (Digit)5;

Digit b = (Digit)3;

Digit plus = a + b;

Digit minus = a - b;

bool equals = (a == b);

Console.WriteLine("{0} + {1} = {2}", a, b, plus);

Console.WriteLine("{0} - {1} = {2}", a, b, minus);

Console.WriteLine("{0} == {1} = {2}", a, b, equals);

}

}

}

#### 索引

#### 构造函数

实例构造函数是实现对类中实例进行初始化的行为的成员

using System;

namespace zh.test

{

class Point

{

public double x, y;

public Point()

{

this.x = 0;

this.y = 0;

}

public Point(double x, double y)

{

this.x = x;

this.y = y;

}

~Point()

{

Console.WriteLine("Destructed {0}", this);

}

public static double Distance(Point a, Point b)

{

double xdiff = a.x - b.x;

double ydiff = a.y - b.y;

return Math.Sqrt(xdiff \* xdiff + ydiff \* ydiff);

}

public override string ToString()

{

return string.Format("({0}, {1})", x, y);

}

}

class MyTestClass

{

static void Main()

{

Point a = new Point();

Point b = new Point(3, 4);

double d = Point.Distance(a, b);

Console.WriteLine("Distance from {0} to {1} is {2}", a, b, d);

}

}

}

#### 析构函数

参考上例

#### 静态构造函数

#### 继承

##### 1、类支持单继承，object类型是所有类的基类。

using System;

namespace zh.test

{

class A

{

public void F() { Console.WriteLine("A.F"); }

}

class B : A

{

public void G() { Console.WriteLine("B.G"); }

}

class MyTestClass

{

static void Main()

{

B b = new B();

b.F(); // Inherited from A

b.G(); // Introduced in B

A a = b; // Treat a B as an A

a.F();

}

}

}

##### 2、派生类中重写父类的方法

using System;

namespace zh.test

{

class A

{

public virtual void F() { Console.WriteLine("A.F"); }

}

class B : A

{

public override void F()

{

base.F();

Console.WriteLine("B.F");

}

}

class MyTestClass

{

static void Main()

{

B b = new B();

b.F();

A a = b;

a.F();

}

}

}

##### 3、抽象类：抽象类可以指定抽象函数－非抽象派生类必须实现的成员

using System;

namespace zh.test

{

abstract class A

{

public abstract void F();

}

class B : A

{

public override void F() { Console.WriteLine("B.F"); }

}

class MyTestClass

{

static void Main()

{

B b = new B();

b.F();

A a = b;

a.F();

}

}

}

### 结构

结构是数值类型而不是引用类型，不支持继承，其数值存储在“堆栈中”；

其它方面与类相似。

#### 1.用类实现

using System;

namespace zh.test

{

class Point

{

public int x, y;

public Point(int x, int y)

{

this.x = x;

this.y = y;

}

}

class MyTestClass

{

static void Main()

{

Point[] points = new Point[10];

for (int i = 0; i < points.Length; i++)

points[i] = new Point(i, i \* i);

Console.WriteLine("points[{0}].x={1}", points.Length - 1, points[points.Length-1].x);

Console.WriteLine("points[{0}].y={1}", points.Length - 1, points[points.Length - 1].y);

}

}

}

#### 2.用结构实现

using System;

namespace zh.test

{

struct Point

{

public int x, y;

public Point(int x, int y)

{

this.x = x;

this.y = y;

}

}

class MyTestClass

{

static void Main()

{

Point[] points = new Point[10];

for (int i = 0; i < points.Length; i++)

points[i] = new Point(i, i \* i);

Console.WriteLine("points[{0}].x={1}", points.Length - 1, points[points.Length-1].x);

Console.WriteLine("points[{0}].y={1}", points.Length - 1, points[points.Length - 1].y);

}

}

}

#### 3.结构型数组

using System;

using zh.code;

namespace zh.code

{

public enum eBankName : int { LZbank, NYbank, ZGbank }//银行类型:本地银行、跨区银行

public enum eBankType : int { Local, TransRegional }//银行类型:本地银行、跨区银行

public struct BankInfo

{

public string BankCode;

public eBankName BankName;

public eBankType BankType;

public BankInfo(string sBankCode, eBankName sBankName, eBankType sBankType)

{

this.BankCode = sBankCode;

this.BankName = sBankName;

this.BankType = sBankType;

}

}

public class Bank

{

BankInfo[] \_BankInfo = new BankInfo[3];

public Bank()

{

\_BankInfo[0] = new BankInfo("L1001", eBankName.LZbank, eBankType.Local);

\_BankInfo[1] = new BankInfo("T1001", eBankName.NYbank, eBankType.TransRegional);

\_BankInfo[2] = new BankInfo("T1002", eBankName.ZGbank, eBankType.TransRegional);

}

public BankInfo[] get\_BankInfo()

{

return \_BankInfo;

}

public BankInfo get\_BankInfo(int index)

{

return \_BankInfo[index];

}

}

}

namespace zh.Test

{

class WelcomeApp

{

static void Main()

{

Bank b = new Bank();

Console.WriteLine("------get\_BankInfo()-----");

BankInfo[] bk = b.get\_BankInfo();

for (int i = 0; i < bk.Length; i++)

{

Console.WriteLine("Bank[{0}].BankCode={1}",i, bk[i].BankCode);

Console.WriteLine("Bank[{0}].BankName={1}",i, bk[i].BankName);

Console.WriteLine("Bank[{0}].BankType={1}",i, bk[i].BankType);

Console.WriteLine("-----------");

}

Console.WriteLine("------get\_BankInfo(int)-----");

BankInfo bk1;

for (int i = 0; i < 3; i++)

{

bk1 =b.get\_BankInfo(i);

Console.WriteLine("Bank[{0}].BankCode={1}", i, bk1.BankCode);

Console.WriteLine("Bank[{0}].BankName={1}", i, bk1.BankName);

Console.WriteLine("Bank[{0}].BankType={1}", i, bk1.BankType);

Console.WriteLine("-----------");

}

}

}

}

### 接口\*？

接口定义了一个连接。一个类或这结构必须根据它的连接来实现接口。接口可以把方法、属性、索引和事件作为成员。

### 代表

using System;

namespace zh.test

{

delegate void SimpleDelegate();

class MyTestClass

{

static void F()

{

System.Console.WriteLine("F()");

}

static void Main()

{

SimpleDelegate d = new SimpleDelegate(F);

d();

}

}

}

using System;

namespace zh.test

{

delegate void SimpleDelegate(int InCount);

class MyTestClass

{

static void F(int InCount)

{

System.Console.WriteLine("F({0})", InCount);

}

static void MultiCall(SimpleDelegate d, int count)

{

for (int i = 0; i < count; i++)

d(i+1);

}

static void Main()

{

SimpleDelegate d = new SimpleDelegate(F);

d(1);

MultiCall(d, 3);

}

}

}

### 联合

using System;

namespace zh.test

{

public enum Color

{

Red,

Blue,

Green,

Other

}

public class Shape

{

public void Fill(Color color)

{

String s = "";

switch (color)

{

case Color.Red:

s = "Color.Red";

break;

case Color.Blue:

s = "Color.Blue";

break;

case Color.Green:

s = "Color.Green";

break;

default:

s = "Color.Other";

break;

}

System.Console.WriteLine("F({0})", s);

}

}

class MyTestClass

{

static void Main()

{

Shape shape = new Shape();

System.Console.WriteLine("请输入颜色：r->Red; b->Blue; g->Green");

string s = System.Console.ReadLine();

Color color ;

switch (s)

{

case "r": color = Color.Red; break;

case "b": color = Color.Blue; break;

case "g": color = Color.Green; break;

default: color = Color.Other; break;

}

shape.Fill(color);

}

}

}

### 名称空间和集合

using System;

using Test2= zh.test2;

namespace zh.test2

{

public enum Color

{

Red,

Blue,

Green,

Other

}

public class Shape

{

public void Fill(Color color)

{

String s = "";

switch (color)

{

case Color.Red:

s = "Color.Red";

break;

case Color.Blue:

s = "Color.Blue";

break;

case Color.Green:

s = "Color.Green";

break;

default:

s = "Color.Other";

break;

}

System.Console.WriteLine("F({0})", s);

}

}

}

namespace zh.test

{

class MyTestClass

{

static void Main()

{

Test2.Shape shape = new Test2.Shape();

System.Console.WriteLine("请输入颜色：r->Red; b->Blue; g->Green");

string s = System.Console.ReadLine();

Test2.Color color;

switch (s)

{

case "r": color = Test2.Color.Red; break;

case "b": color = Test2.Color.Blue; break;

case "g": color = Test2.Color.Green; break;

default: color = Test2.Color.Other; break;

}

shape.Fill(color);

}

}

}

### 版本

### 特征

## 语法结构

### 预处理声明

#define定义一个标识符。#undef“反定义”

### 语法分析-》句法分析-》数据符号[转义]

using System;

namespace zh.test

{

class MyTestClass

{

static void Main()

{

string a = "hello, world"; // hello, world

string b = @"hello, world"; // hello, world

string c = "hello \t world"; // hello world

string d = @"hello \t world"; // hello \t world

string e = "Joe said \"Hello\" to me"; // Joe said "Hello"

string f = @"Joe said ""Hello"" to me"; // Joe said "Hello"

string g = "\\\\server\\share\\file.txt"; // \\server\share\file.txt

string h = @"\\server\share\file.txt"; // \\server\share\file.txt

string i = "one\ntwo\nthree";

string j = @"one

two

three";

Console.WriteLine("a 输出:{0}",a);

Console.WriteLine("b 输出:{0}", b);

Console.WriteLine("c 输出:{0}", c);

Console.WriteLine("d 输出:{0}", d);

Console.WriteLine("e 输出:{0}", e);

Console.WriteLine("f 输出:{0}", f);

Console.WriteLine("g 输出:{0}", g);

Console.WriteLine("h 输出:{0}", h);

Console.WriteLine("i 输出:{0}", i);

Console.WriteLine("j 输出:{0}", j);

}

}

}

## 条件语句

### if

using System;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

string str = Console.ReadLine();

if (str == "a")

{

Console.WriteLine("满足条件1输入了" + str);

}

else if (str == "b")

{

Console.WriteLine("满足条件2输入了" + str);

}

else

{

Console.WriteLine("满足其它条件输入了" + str);

}

}

}

}

### swich

#### swich1

using System;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

string str = Console.ReadLine();

switch (str)

{

case "a":

Console.WriteLine("满足条件1输入了" + str);

break;

case "b":

Console.WriteLine("满足条件2输入了" + str);

break;

default:

Console.WriteLine("满足其它条件输入了" + str);

break;

}

}

}

}

#### Swich2

using System;

namespace hs.test

{

class MyFirstClass

{

static void Main()

{

//Console.WriteLine("请输入字母或客串：" );

//string str = Console.ReadLine();

//switch (str)

//{

// case "A": Console.WriteLine("输入了字母:" + str); break;

// case "B": Console.WriteLine("输入了字母:" + str); break;

// case "C": Console.WriteLine("输入了字母:" + str); break;

// default:

// Console.WriteLine("输入了其它字母:" + str);

// break;

//}

Console.WriteLine("请输入数字x [ 0< x <101 ]");

int i = 0;

try

{

i = int.Parse(Console.ReadLine());

if (i < 0 || i > 100)

{

Console.WriteLine("超出了正常范围，取值范围 0<x<101");

return;

}

i = i / 10;

int y = 0;

switch (i)

{

case 6: y = 2; break;

case 7: y = 3; break;

case 8: y = 3; break;

case 9: y = 4; break;

case 10: y = 4; break;

default: y = 1; break;

}

Console.WriteLine("等级:" + (y.ToString()));

}

catch (Exception e)

{

Console.WriteLine("运行时发生错误:" + e.ToString());

}

}

}

}

## 循环

### 1、for 循环

using System;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

for (int i = 0; i < 6; i++)

{

Console.WriteLine(i);

}

}

}

}

### 2、while 循环

using System;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

int i = 2;

while(i<7)

{

Console.WriteLine(i);

i++;

}

}

}

}

### 3、do…while 循环

using System;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

int i = 0;

do

{

Console.WriteLine(i);

i++;

}

while (i < 7);

}

}

}

### 4、foreach 循环

using System;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

string[] strArr = { "aaa", "bbb", "ccc", "cn" };

foreach(string temp in strArr)

{

Console.WriteLine(temp);

}

}

}

}

## 跳转语句

### Break

退出某个case语句或退出某个循环

### continue

跳出本次循环，接着执行下次循环

using System;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

string[] strArr = { "aaa", "bbb", "ccc", "cn" };

foreach(string temp in strArr)

{

if (temp=="bbb")

{

continue;

//break;

//return;

}

Console.WriteLine(temp);

}

}

}

}

### return 语句

用于退出类，或者方法，或者返回方法的返回值

## 枚举

//例4.1.4-1

using System;

using zh.Test.Code;

using nseEnum = zh.Test.Code.WeekDay;

namespace zh.Test.Code

{

public enum WeekDay:int { Monday=1,Tuesday ,Wednesday , Thursday,Friday ,Saturday , Sunday }

public class Enums

{

public void Show(WeekDay iWeekDay)

{

Console.WriteLine("Value={0};Name={1}", (int)iWeekDay, iWeekDay);

}

public void ShowWeekDay(WeekDay iWeekDay)

{

switch (iWeekDay)

{

case WeekDay.Monday:

Console.WriteLine("工作安排:{0}", iWeekDay);

break;

case WeekDay.Tuesday:

Console.WriteLine("工作安排:{0}", iWeekDay);

break;

case WeekDay.Wednesday:

Console.WriteLine("工作安排:{0}", iWeekDay);

break;

case WeekDay.Thursday:

Console.WriteLine("工作安排:{0}", iWeekDay);

break;

case WeekDay.Friday:

Console.WriteLine("工作安排:{0}", iWeekDay);

break;

case WeekDay.Saturday:

Console.WriteLine("工作安排:{0}", iWeekDay);

break;

default:

Console.WriteLine("无工作安排:{0}", iWeekDay);

break;

}

}

}

}

namespace zh.Test

{

class WelcomeApp

{

static void Main()

{

Enums c = new Enums();

c.Show(WeekDay.Tuesday);

c.Show(nseEnum.Wednesday);

c.Show(zh.Test.Code.WeekDay.Friday);

Console.WriteLine("------------");

c.ShowWeekDay(WeekDay.Monday);

c.ShowWeekDay(WeekDay.Sunday);

}

}

}

using System;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

//int userAage = (int)user.fatherAge;

//Console.WriteLine(userAage);

user userTemp = user.gread;

Console.WriteLine(userTemp.ToString());

}

public enum user

{

Age=18,gread=2,fatherAge=60

}

}

}

## 二、数组

int [] userCount ;

int [] userCount=new int [20];

## 三、命名空间

using System;

using gosoa1 = gosoa.com;

using gosoa2 = gosoa.com.cn;

namespace gosoa.com

{

public class MyFirstClass

{

public string getUrl()

{

return "gosoa.com";

}

static void Main()

{

//gosoa.com.MyFirstClass urlClassNew1 = new gosoa.com.MyFirstClass();

//gosoa.com.cn.MyFirstClass urlClassNew2 = new gosoa.com.cn.MyFirstClass();

gosoa1.MyFirstClass urlClassNew1 = new gosoa1.MyFirstClass();

gosoa2.MyFirstClass urlClassNew2 = new gosoa2.MyFirstClass();

string url1 = urlClassNew1.getUrl();

string url2 = urlClassNew2.getUrl();

Console.WriteLine(url1);

Console.WriteLine(url2);

}

}

}

namespace gosoa.com.cn

{

public class MyFirstClass

{

public string getUrl()

{

return "gosoa.com.cn";

}

}

}

## 类

### 类的概述

using System;

namespace gosoa.com.cn

{

public class Doctor

{

//两个域

private string \_Name;

private byte \_Age;

//两个属性

public string Name

{

get { return this.\_Name; }

set { this.\_Name = value; }

}

public byte Age

{

get { return this.\_Age; }

set { this.\_Age = value; }

}

//两个方法

public string doSth()

{

return "这是一个医生类！";

}

public static string doAnth()

{

return "这是医生类中的一个静态方法";

}

//两个构造函数

public Doctor() { }

public Doctor(string name, byte age)

{

this.\_name = Name;

this.\_age = Age;

}

}

public class OneDoctor

{

static void Main()

{

//实例化类

Doctor dc = new Doctor();

dc.Name = "李四";

dc.Age = 25;

Doctor dc2 = new Doctor("张三", 35);

Console.WriteLine(dc.Name);

Console.WriteLine(dc.Age);

Console.WriteLine(dc2.Name);

Console.WriteLine(dc2.Age);

Console.WriteLine(dc.doSth());

//静态方法无须实例化，可直接调用

Console.WriteLine(Doctor.doAnth());

}

}

}

## 方法

### 1、方法概述

静态方法和非静态方法，静态方法无须实例化，可直接调用。

### 2、方法的参数

### 3、Ref参数

using System;

namespace gosoa.com.cn

{

public class OneDoctor2

{

static void FunctionTest(int[] arr,ref int x)

{

arr[0] = 100;

x = 10;

}

static void Main()

{

int[] arrTemp = { 0, 1, 2, 3, 4 };

int y = 30;

Console.WriteLine(arrTemp[0]);

Console.WriteLine(y);

FunctionTest(arrTemp,ref y);

Console.WriteLine(arrTemp[0]);

Console.WriteLine(y);

}

}

}

本例的输出结果是 0，30，100，30 因为数组是引用类型，在调用方法前后，引用类型的修改会保留下 来，而值类型的修改不会保留下来。

### 4、out参数

using System;

namespace gosoa.com.cn

{

public class OneDoctor2

{

static void FunctionTest(out int x)

{

x = 200;

}

static void Main()

{

int y;

FunctionTest(out y);

Console.WriteLine(y);

}

}

}

### 5、方法重载

using System;

namespace gosoa.com.cn

{

public class test

{

static int FunctionTest(int x)

{

return x + 100;

}

static string FunctionTest(string str)

{

return str;

}

static int FunctionTest(int x, int y)

{

return x + y;

}

static void Main()

{

Console.WriteLine(FunctionTest(10));

Console.WriteLine(FunctionTest("gosoa.com.cn"));

Console.WriteLine(FunctionTest(5, 20));

}

}

}

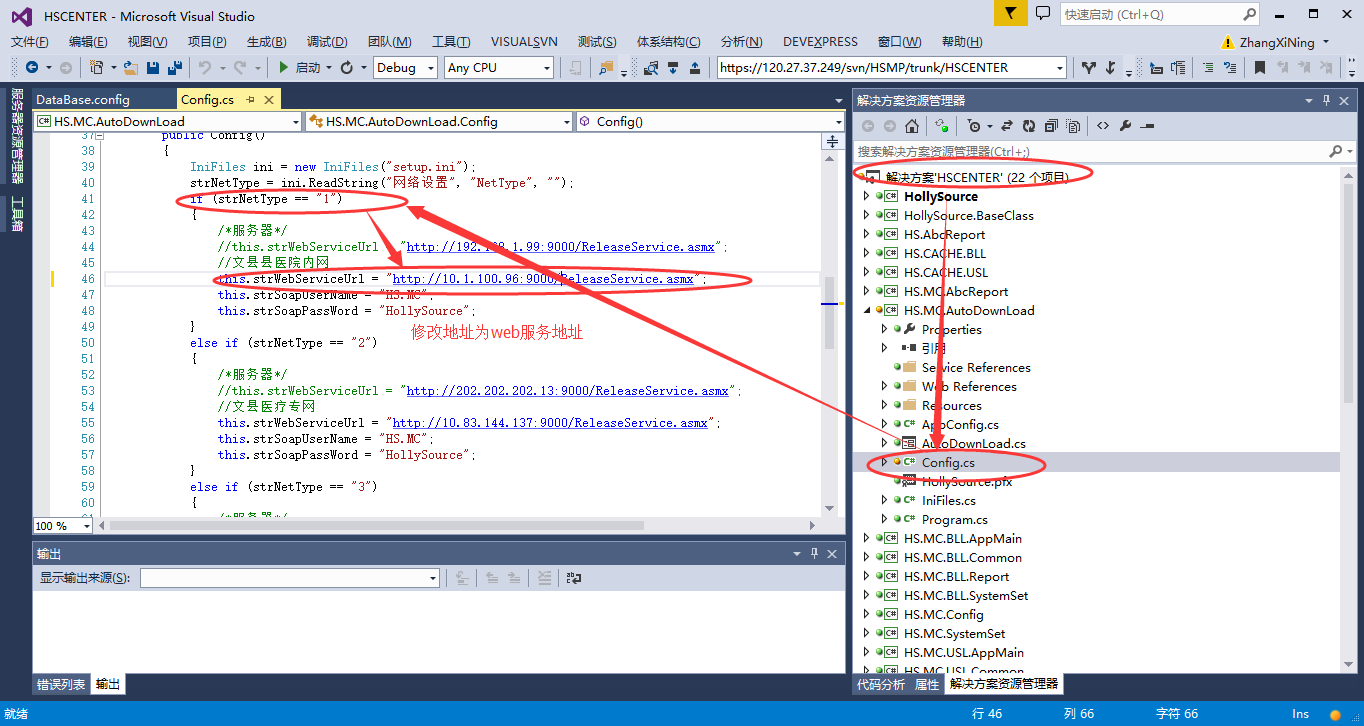
## 生成安装文件

### 所需文件

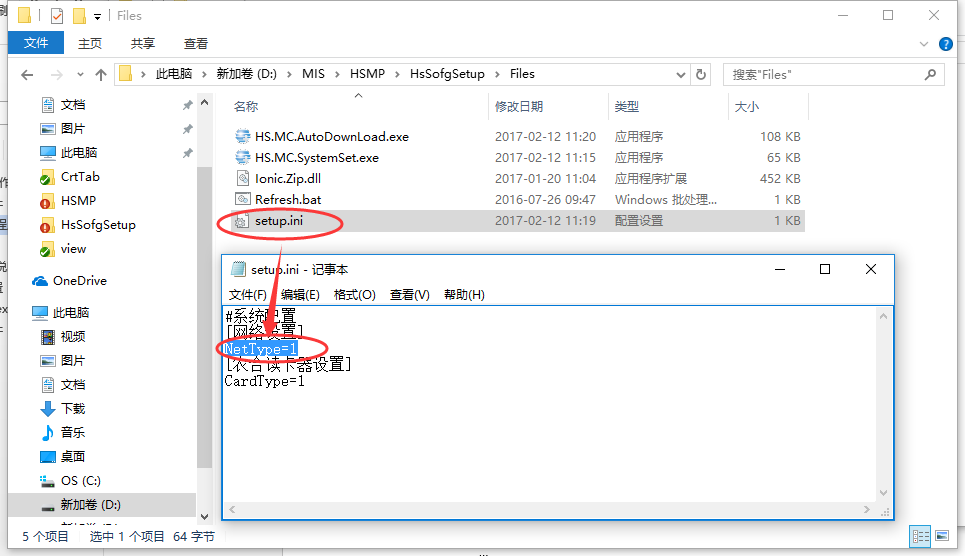
### 文件路径：

D:\MIS\HSMP\HSEXE

#### 1、HS.MC.AutoDownLoad.exe



#### 2、setup.ini



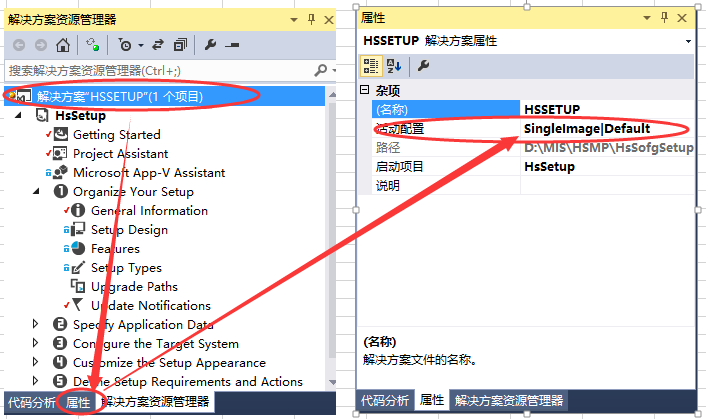
#### 3、Refresh.bat

#### 4、HS.MC.SystemSet.exe

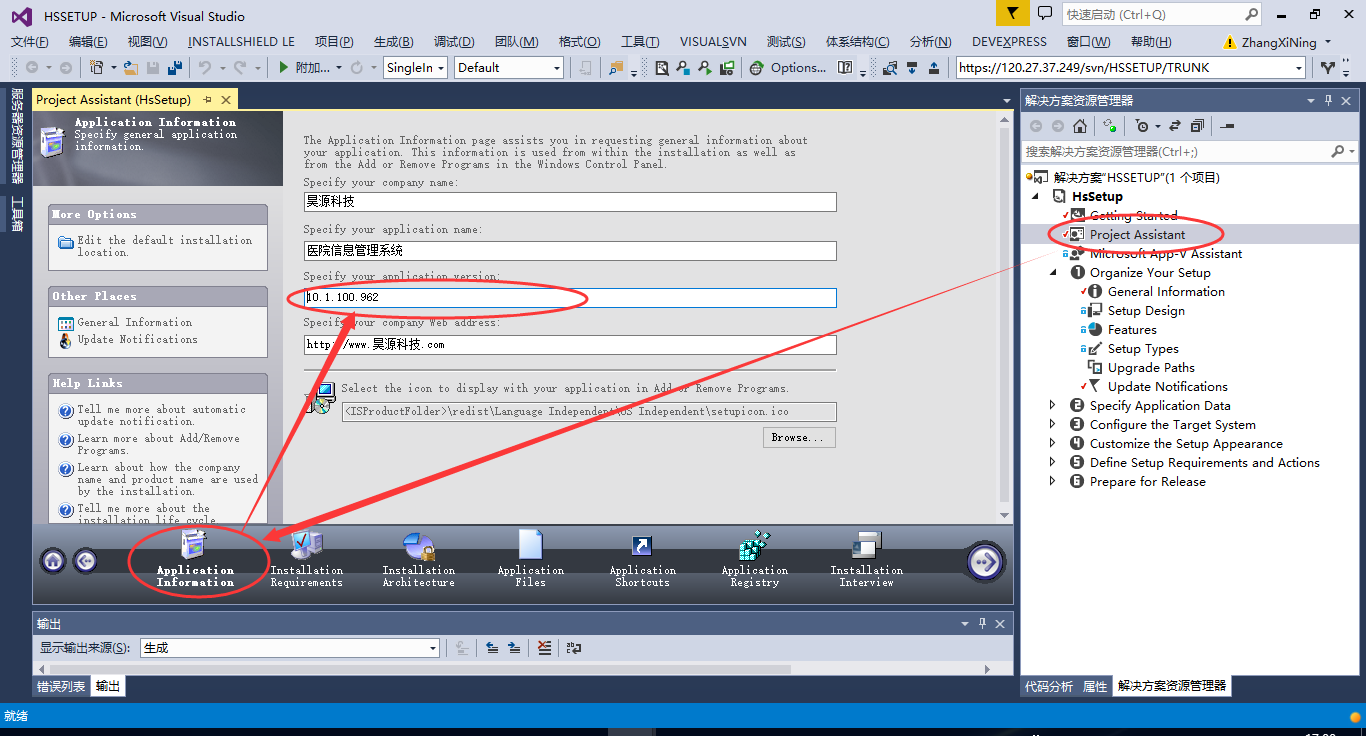
#### 5、Ionic.Zip.dll

### 相关设置

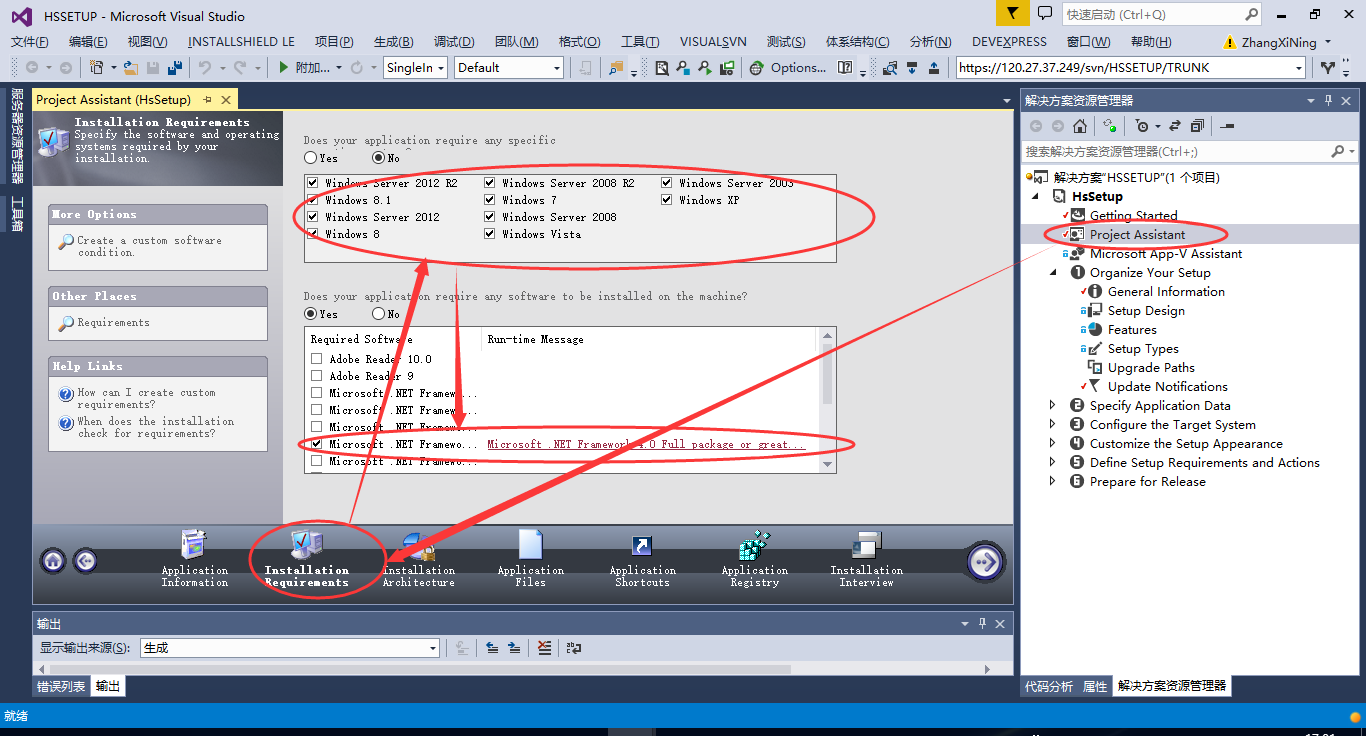
#### 输出类型：exe



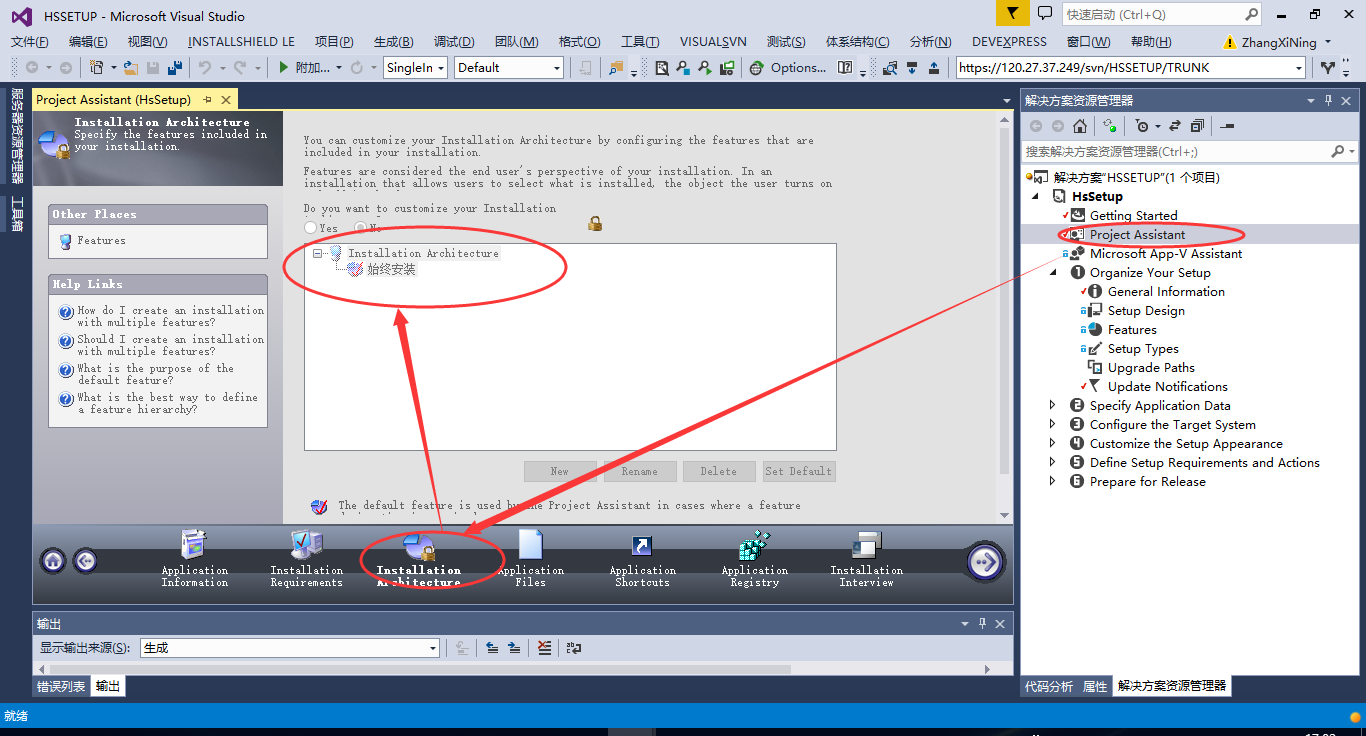
#### 设置1、版本号=web服务ip



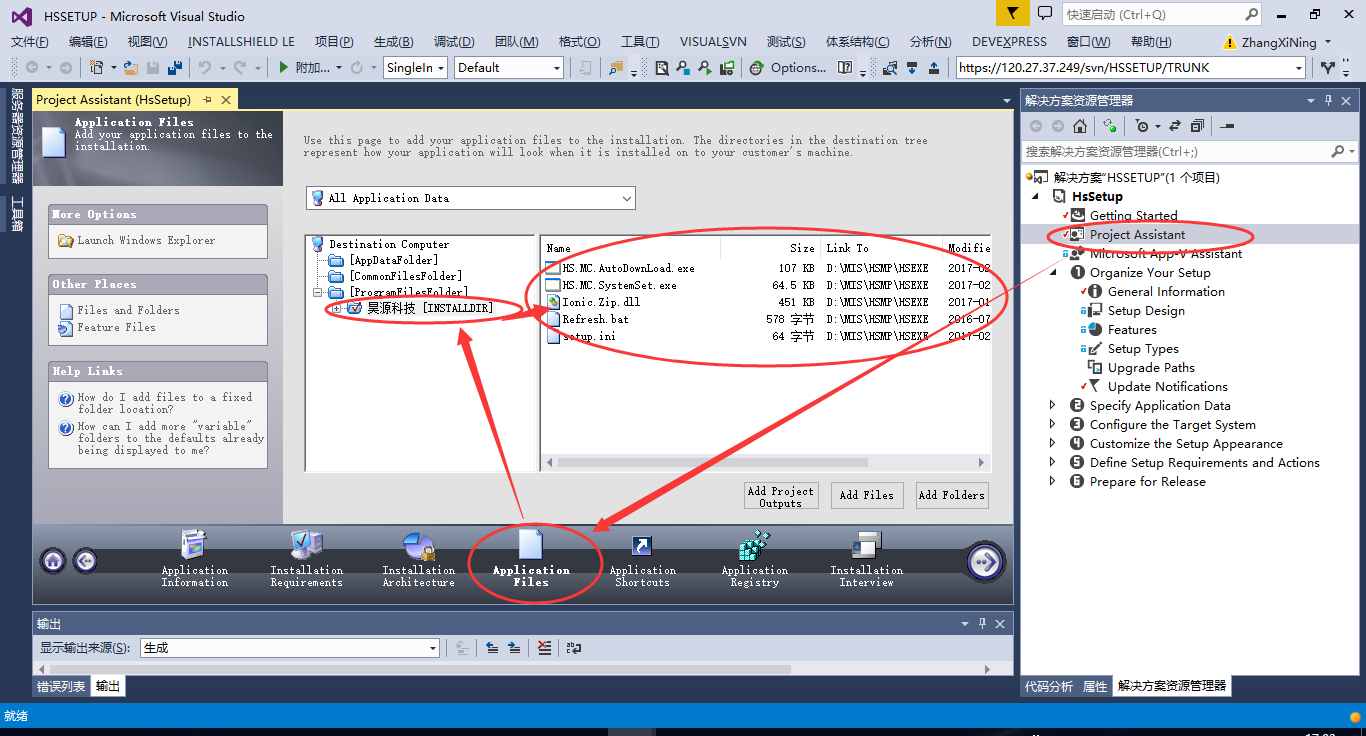
#### 设置2、



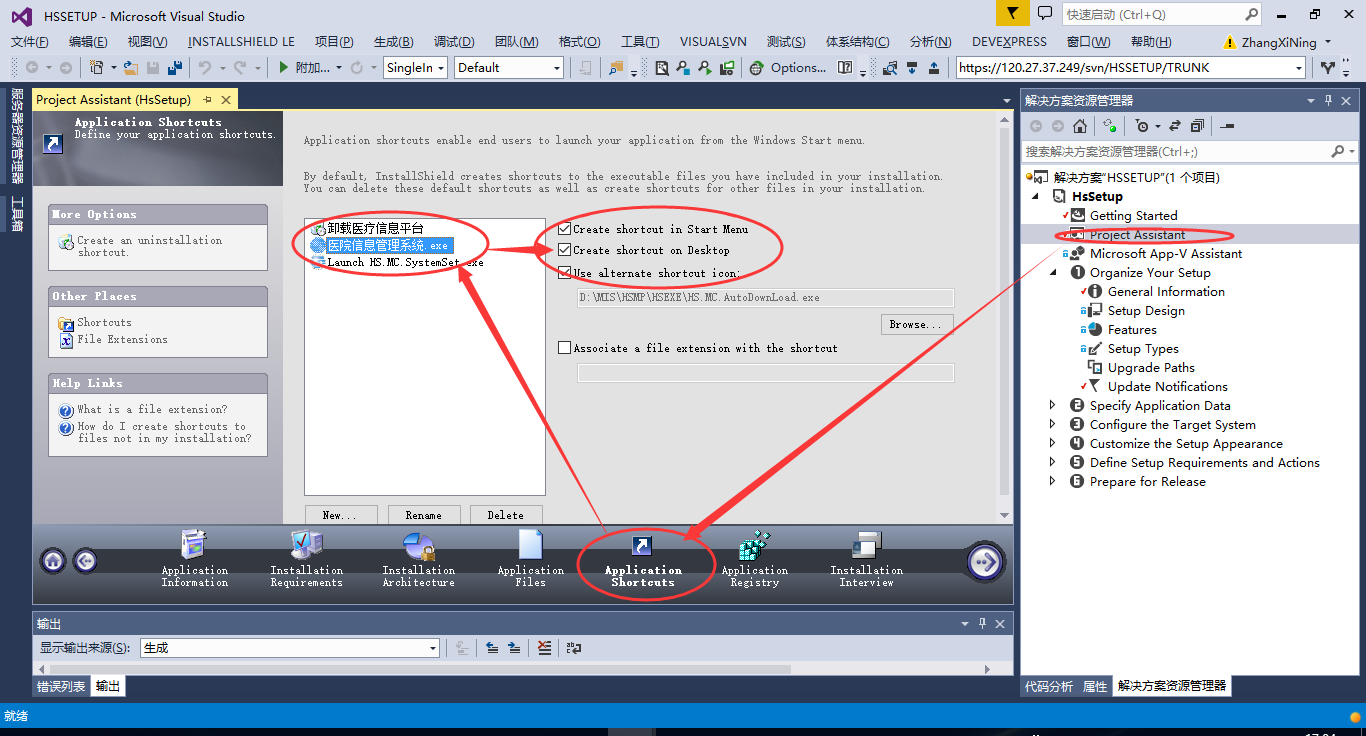
#### 设置3、



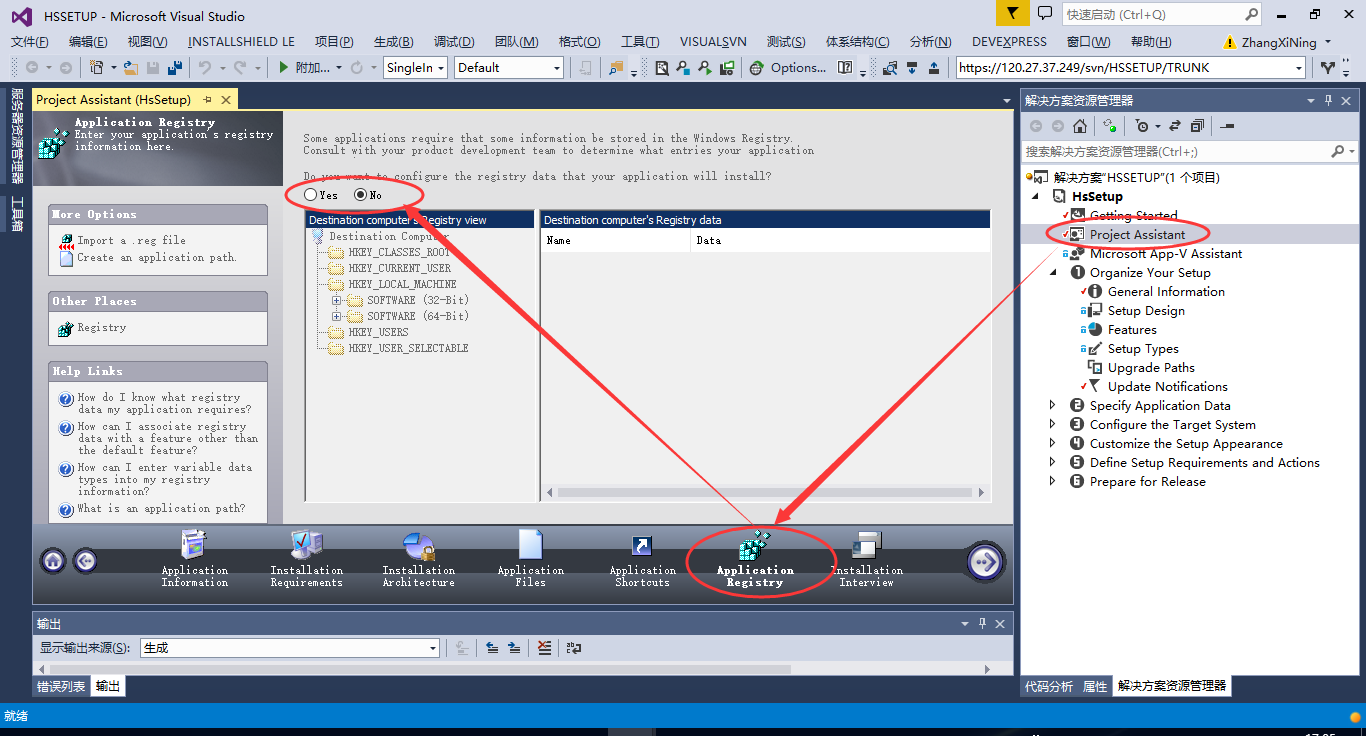
#### 设置4、所需文件



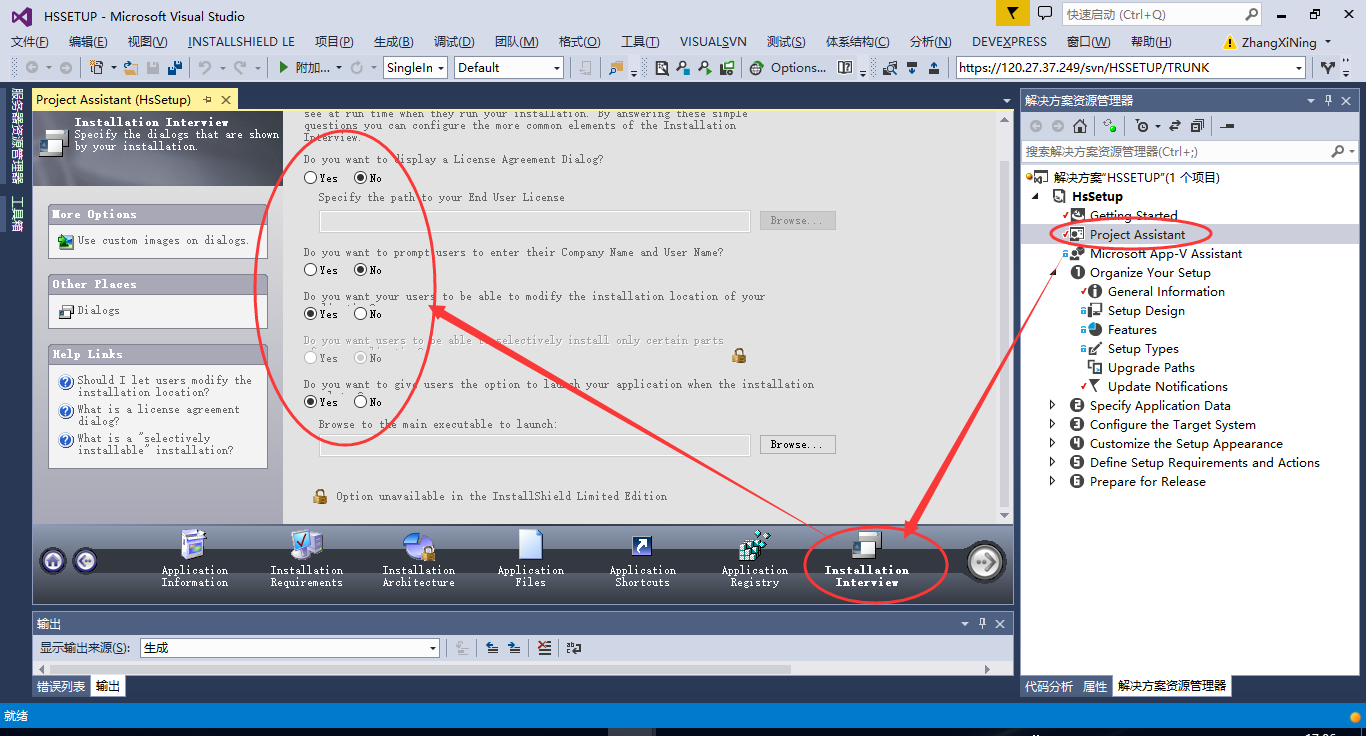
#### 设置5、快捷方式\菜单



#### 设置6、注册表设置



#### 设置7、



### 生成文件

#### 1、删除.\HsSetup\Express

#### 2、生成方案

#### 3、输出路径：

. \HsSetup\Express\SingleImage\DiskImages\DISK1

## HS.MC.AutoDownLoad\Config.cs -> Web服务器->ODP

## Web服务器ip='192.168.1.168'

1、D:\MIS\HSCENTER\HS.MC.AutoDownLoad\Config.cs

this.strWebServiceUrl = "http://192.168.1.168:9000/ReleaseService.asmx";

2、D:\MIS\HSCENTER\WebService\HS.MC.WS.Config\DataBase.config

<add key="Host" value="192.168.1.168"/>

3、hscenter.app\_config\_

webservice\_url = 'http://192.168.1.168:9003'

1、D:\MIS\HSMP\HSCENTER\WebService\HS.MC.WS.Config\DataBase.config

修改 <add key="Host" value="localhost"/>//数据服务器地址

2、D:\MIS\HSMP\HSCENTER\HS.MC.AutoDownLoad\Config.cs

修改 this.strWebServiceUrl = "http://192.168.0.170:9000/ReleaseService.asmx"; //web服务器地址

hssoftv6.0编译注意事项：

1、目标数据库地址：

HS.XXX.WS.Config->DataBase.config

<add key="Host" value="localhost"/>

<add key="Host" value="188.188.1.1"/>

2、首次访问：web服务地址：

hscenter.app\_config 是发布服务的地址

如:D:\MIS\HSMP\HSCENTER\HS.MC.AutoDownLoad\Config.cs

this.strWebServiceUrl = "http://188.188.0.1:9000/ReleaseService.asmx";

3、最终访问：web服务地址

hscenter.app\_config.url

## RabbitMQ安装

## ODP安装卸载

经过反复测试：

如果oracle数据库和iis在同一个机器上，并且机器是64位操作系统，1：如果oracle是64位的，

那么odp.net安装64位的；2：如果oracle是32位的，必须安装32位的odp.net；

卸载odp.net过程：

1.停止iis；

2.进入：c:\odp.net,执行：unconfigure.bat all myhome(configure all component)，uninstall.bat all C:\odp.net myhome(install all components)；

3.删除c:\odp.net目录；

4.重装匹配的odp.net；

## 安装：

## cmd

## d:

## cd /d D:\ODAC112040Xcopy\_64bit

## install.bat all C:\odp.net myhome(install all components)

## cd /d C:\odp.net

## configure.bat all myhome(configure all component)

## conn

# ODP链接ORACLE

//https://www.cnblogs.com/yjmyzz/p/3400999.html

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Oracle.ManagedDataAccess.Client;

namespace ZH.TEST.WS.FormMain

{

public partial class FormMain : Form

{

public FormMain()

{

InitializeComponent();

}

private void connOra\_tsm\_Click(object sender, EventArgs e)

{

Programs pg = new Programs();

OracleConnection conn = null;

try

{

conn = pg.OpenConn();

var cmd = conn.CreateCommand();

cmd.CommandText = "select e.empno,e.ename,e.deptno from scott.emp e where e.empno = '7369'";

cmd.CommandType = CommandType.Text;

var reader = cmd.ExecuteReader();

while (reader.Read())

{

MessageBox.Show(string.Format("ename:{0},deptno:{1}", reader["ename"], reader["deptno"]));

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

finally

{

pg.CloseConn(conn);

}

Console.Read();

}

}

public class Programs

{

public OracleConnection OpenConn()

{

OracleConnection conn = new OracleConnection();

conn.ConnectionString = "Data Source=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost )(PORT=1521))(CONNECT\_DATA=(SERVICE\_NAME=his)));Persist Security Info=True;User ID=hospital;Password=source;";

conn.Open();

return conn;

}

public void CloseConn(OracleConnection conn)

{

if (conn == null) { return; }

try

{

if (conn.State != ConnectionState.Closed)

{

conn.Close();

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

finally

{

conn.Dispose();

}

}

}

}

# 生成parameterMap

**select** '<procedure id="P$' || ua.object\_name || '" parameterMap ="PM$' ||  
 ua.object\_name || '">' s,  
 1 seq,  
 0 sequence  
 **from** user\_arguments ua  
 **where** ua.package\_name = :in\_package\_name  
 **and** ua.object\_name = :in\_object\_name  
 **and** rownum = 1  
**union**  
**select** **user** || '.' || ua.package\_name || '.' || ua.object\_name s,  
 2 seq,  
 0 sequence  
 **from** user\_arguments ua  
 **where** ua.package\_name = :in\_package\_name  
 **and** ua.object\_name = :in\_object\_name  
 **and** rownum = 1  
**union**  
**select** '</procedure>' s, 3 seq, 0 sequence  
 **from** dual  
**union**  
**select** '<parameterMap class="Hashtable" id="PM$' || ua.object\_name || '">' s,  
 4 seq,  
 ua.sequence  
 **from** user\_arguments ua  
 **where** ua.package\_name = :in\_package\_name  
 **and** ua.object\_name = :in\_object\_name  
 **and** rownum = 1  
**union**  
**select** '<parameter column="' || ua.argument\_name || '" direction="' ||  
 decode(ua.in\_out, 'IN', 'Input', 'Output') || '"' ||  
 decode(ua.in\_out, 'IN', '', 'size="100"') || ' property="' ||  
 lower(ua.argument\_name) || '"/>' s,  
 5 seq,  
 ua.sequence  
 **from** user\_arguments ua  
 **where** ua.package\_name = :in\_package\_name  
 **and** ua.object\_name = :in\_object\_name  
**union**  
**select** '</parameterMap>' s, 6 seq, 999 sequence  
 **from** dual  
 **order** **by** seq, sequence

# 其它