# C#Test

## 第三部分 面向对象的C#

### 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

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 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;//乘客数

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

: base(wheels, weight)

{

\_wheels = wheels;

\_weight = weight;

\_Passengers = Passengers;

}

//public void Speak() //不加new编译会发出警告

new public void Speak()

{

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

}

public int wheels

{

get { return \_wheels; }

}

public double weight

{

get { return \_weight; }

}

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}", c1.wheels);

Console.WriteLine("Car.weight={0}", c1.weight);

Console.WriteLine("Car.Passengers={0}", c1.Passengers);

}

}

}

##### 14.1.3 base保留字

using System;

using classS = zh.test.classS;

namespace zh.test.classS

{

class classA

{

public void F()

{

Console.WriteLine("this is 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

{

public void G()

{

base[1] = "abcd";

string x= base[1];

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派生类通过重载改变虚方法

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;

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

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()

{

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;//乘客数

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

: base(wheels, weight)

{

\_wheels = wheels;

\_weight = weight;

\_Passengers = Passengers;

}

public override void Speak()

{

Console.WriteLine("this is 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()

{

Console.WriteLine("this is 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);

v1.Speak();

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

v1 = c1;

v1.Speak();

c1.Speak();

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

v1 = t1;

v1.Speak();

t1.Speak();

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

Console.WriteLine("Car.wheels={0}", c1.wheels);

Console.WriteLine("Car.weight={0}", c1.weight);

Console.WriteLine("Car.Passengers={0}", c1.Passengers);

}

}

}

#### 14.3抽象和密封

##### 14.3.1抽象类

using System;

using nsClass = zh.test.Class;

namespace zh.test.Class

{

abstract class A

{

public abstract void F();

}

abstract class B : A

{

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

}

}

namespace zh.test

{

class C : nsClass.B

{

public override void F()

{Console.WriteLine("C实现抽象方法A.F()! ");}

}

class MyTestClass

{

static void Main()

{

C c = new C();

c.F();

c.G();

}

}

}

#### 14.4继承中关于属性的一些问题

#### 14.5小结

# 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

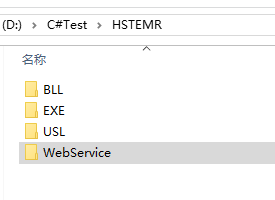
### N04.微软C#完全手册

### N05.C#高级编程(第7版)

## 创建解决方案

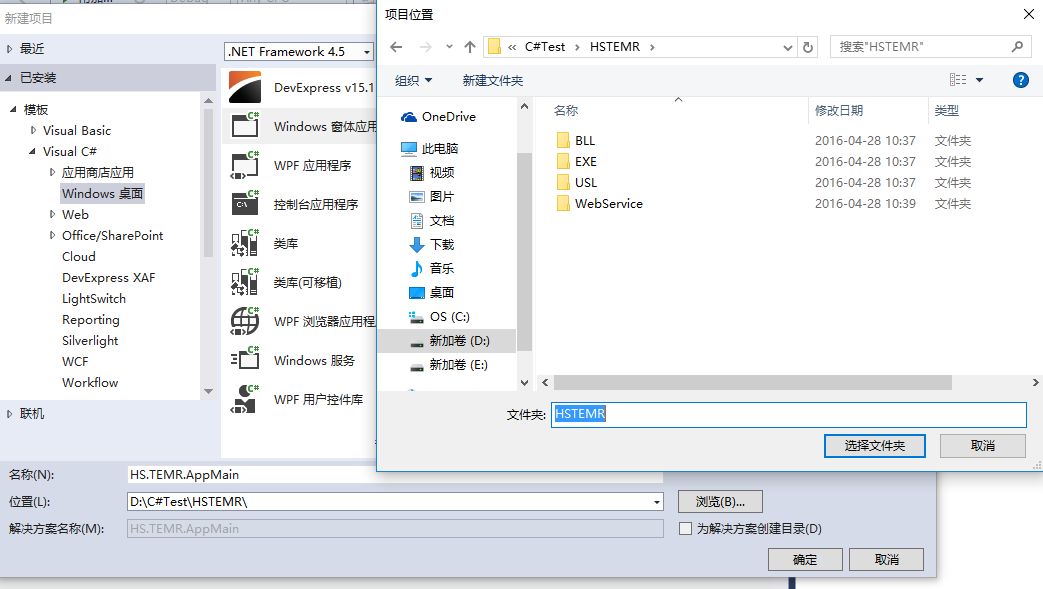
### 创建文件目录

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



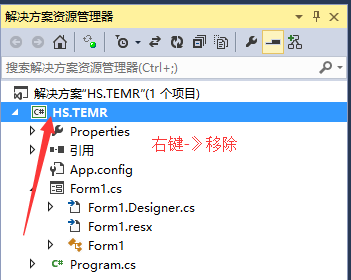
### 新建解决方案

HS.TEMR【文件-》新建-》解决方案】

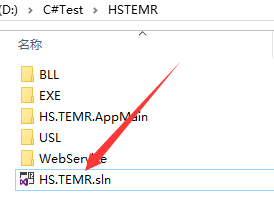


### 移除项目

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

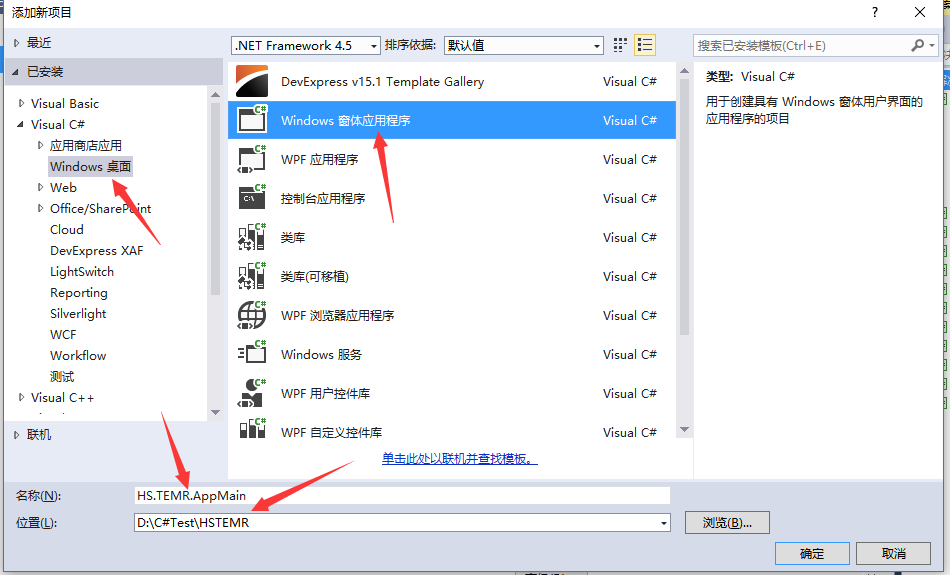
主窗体通常保存到根目录、

HS.TEMR.BILL.\*逻辑层：保存到

HS.TEMR.USL.\*展示层 保存到BLL

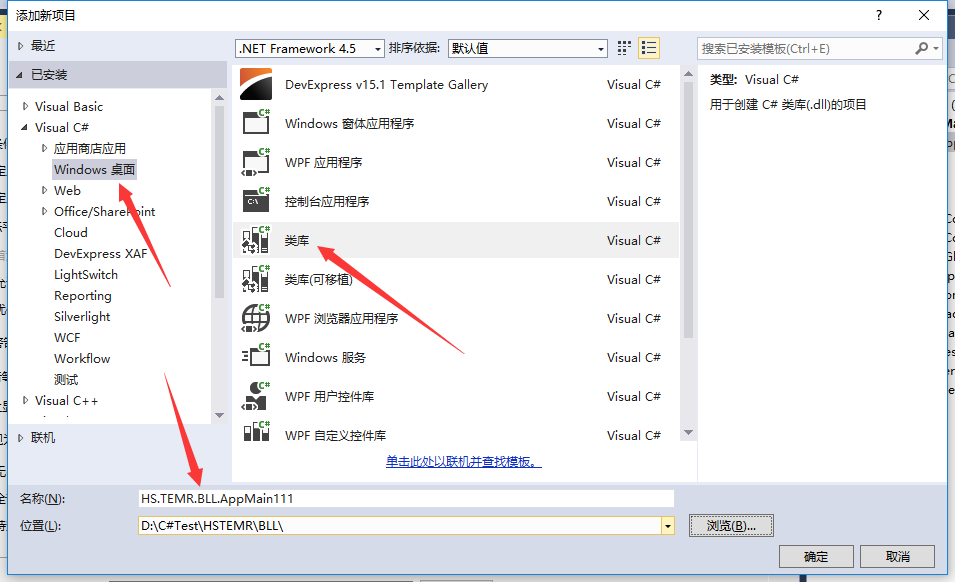
添加项目HS.TEMR.AppMain

【文件->添加->添加新项目】



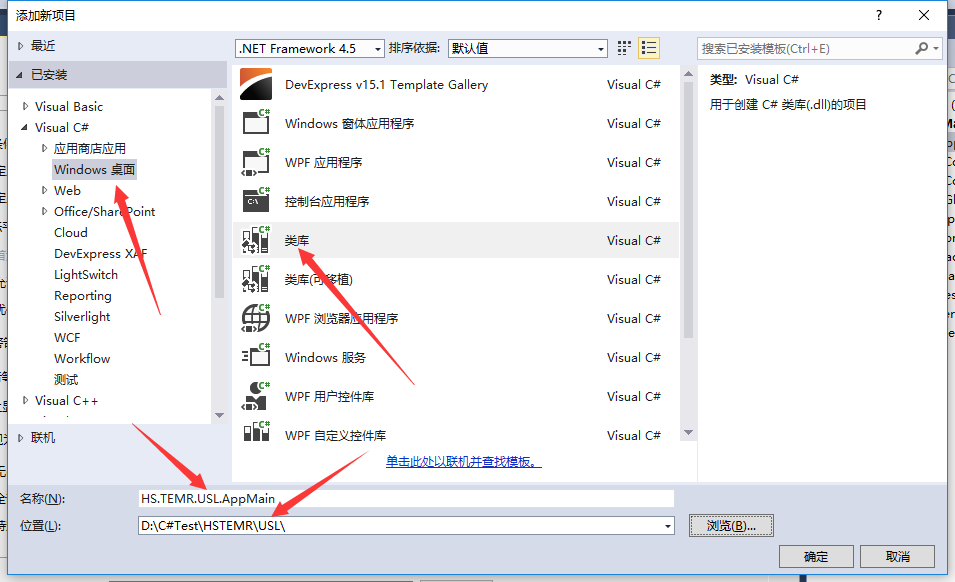
#### 添加类库HS.TEMR.BLL.AppMain

【文件->添加->添加新项目】



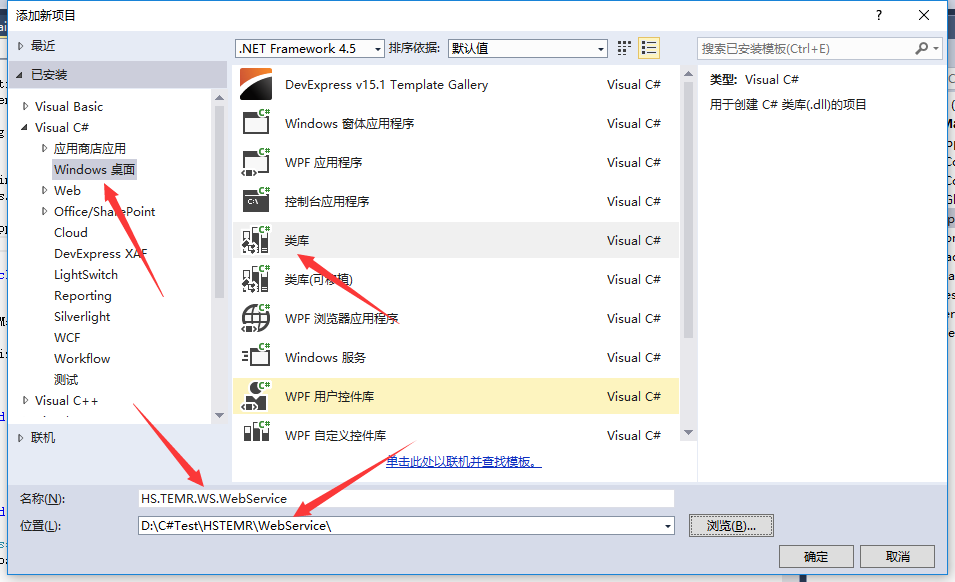
#### 添加类库HS.TEMR.USL.AppMain

【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】

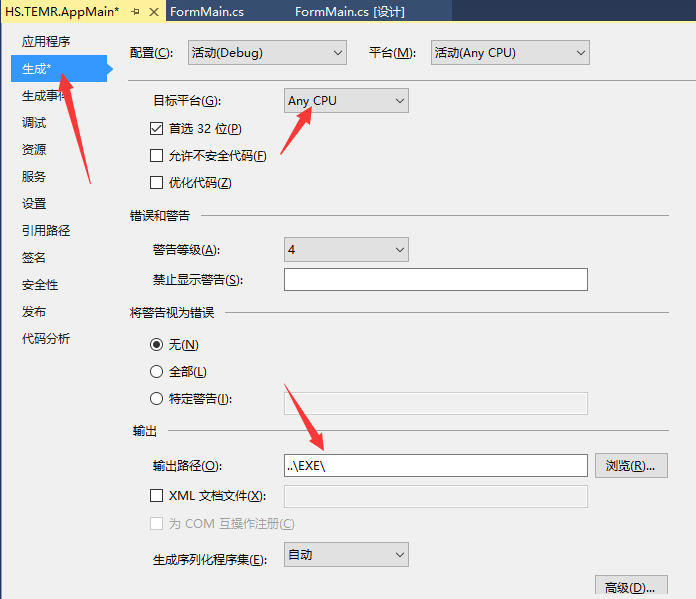


### 设置项目属性

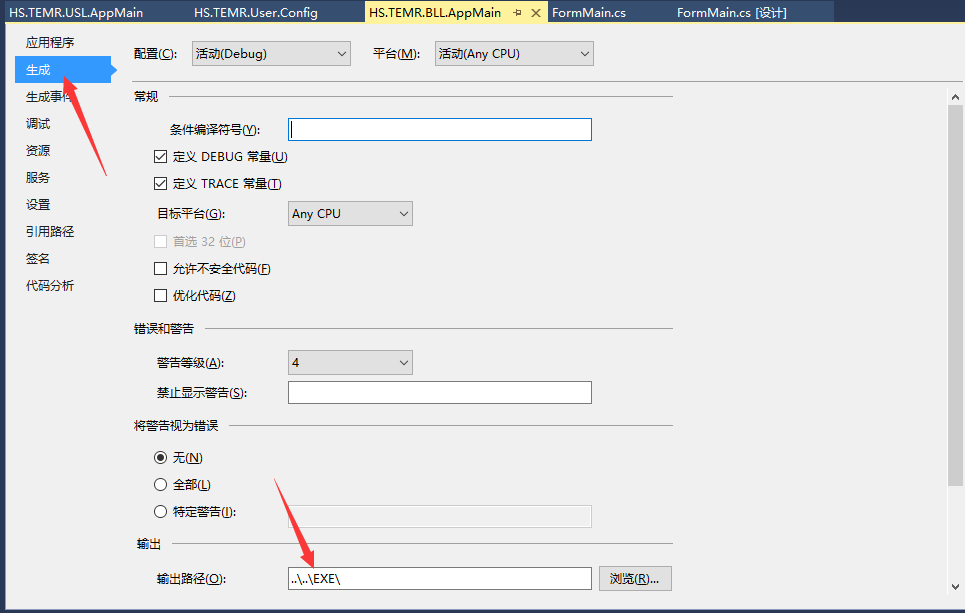
HS.TEMR.WS.WebService输出到..\bin\

HS.TEMR.AppMain.. 输出到..\..\exe\

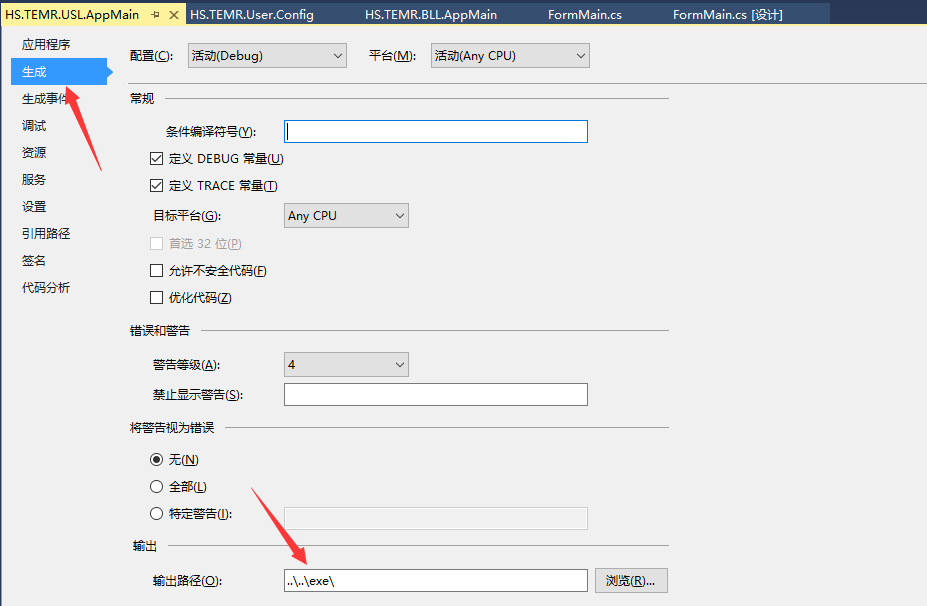
设置HS.TEMR.AppMain属性



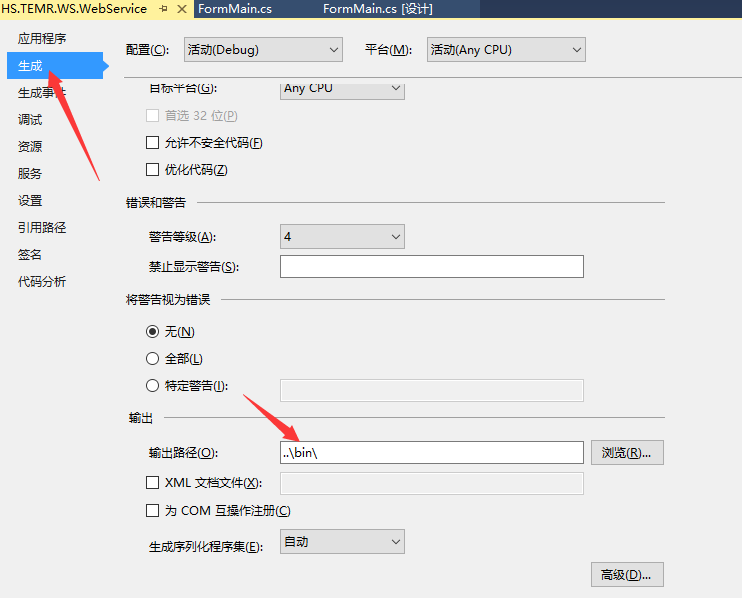
#### 设置HS.TEMR.BLL.AppMain属性



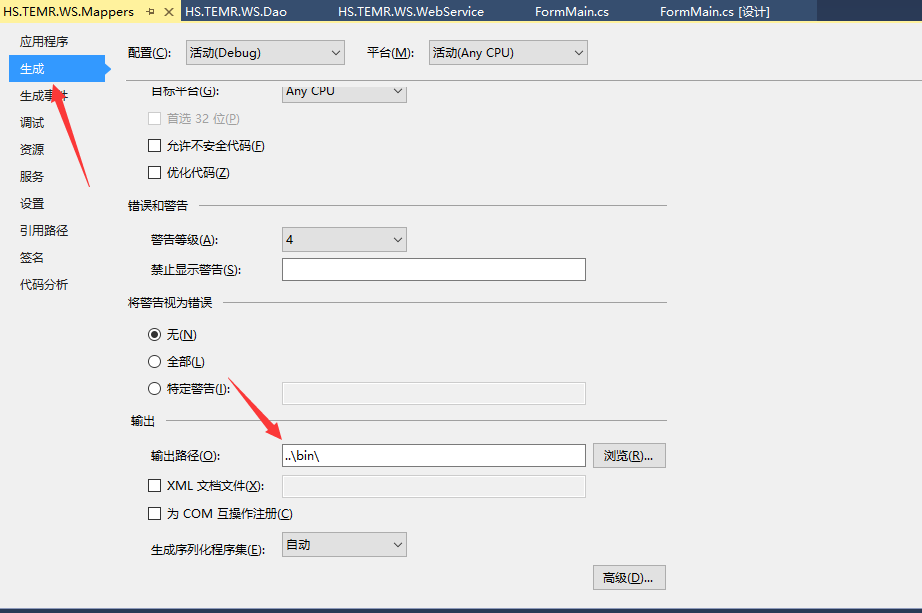
#### 设置HS.TEMR.USL.AppMain属性



#### 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

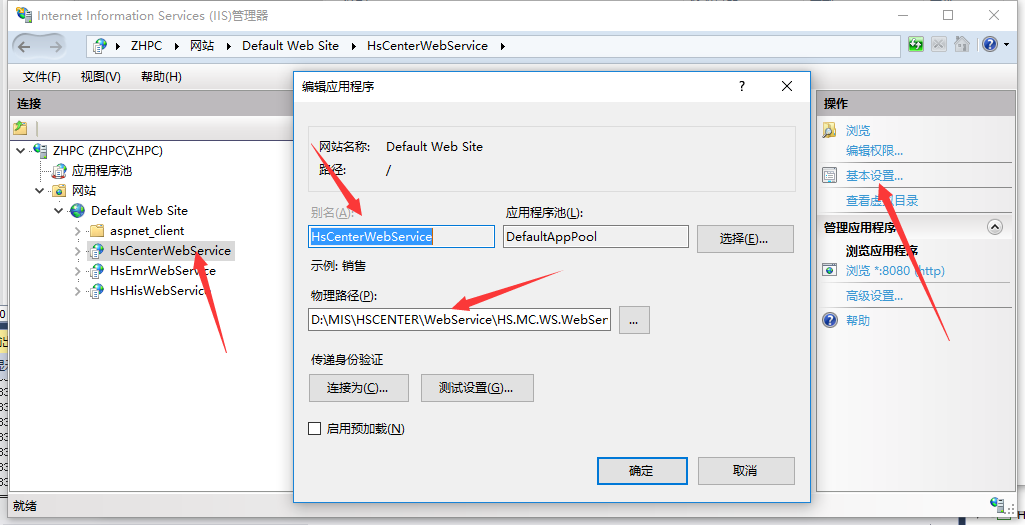
控制面板-》管理工具-》Internet Information Services (IIS)管理器

### 常用IIS配置

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

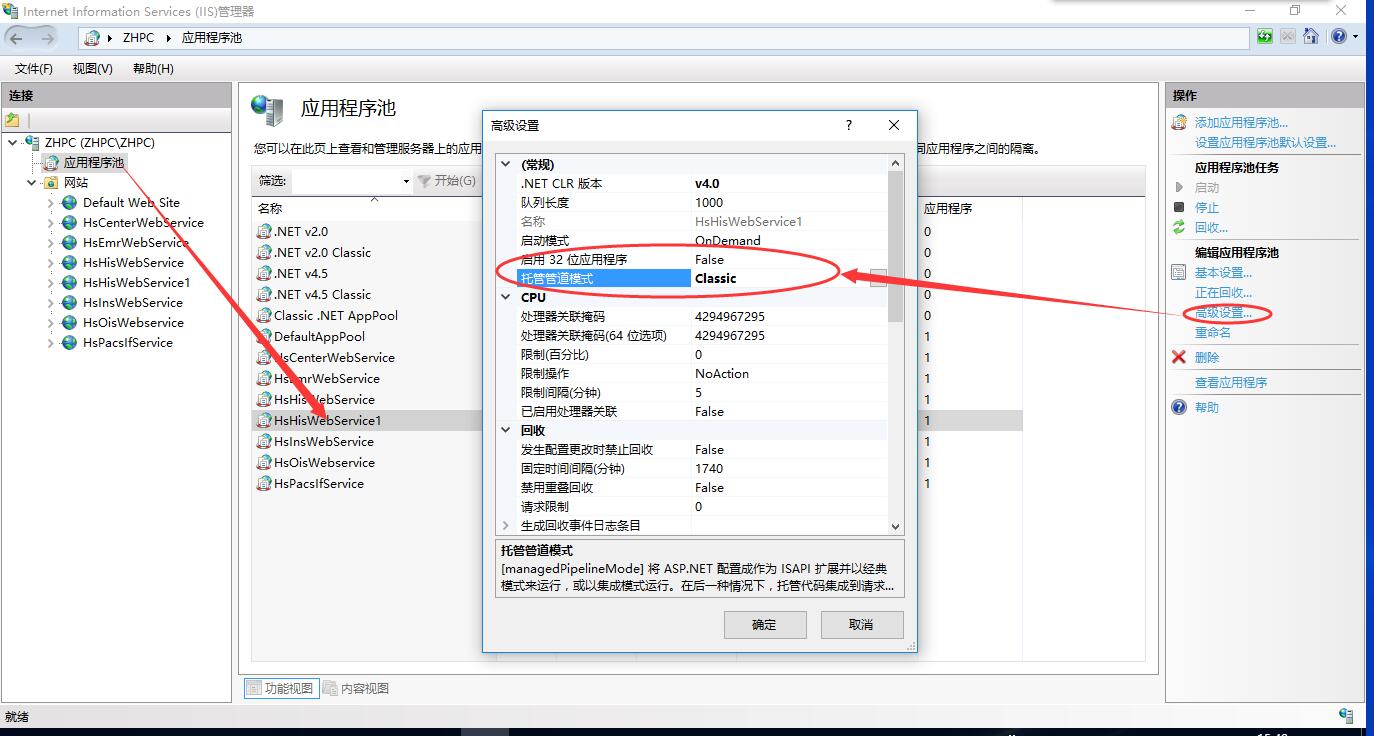
2、HsEmrWebService D:\MIS\HSEMR\WebService\HS.EMR.WS.WebService

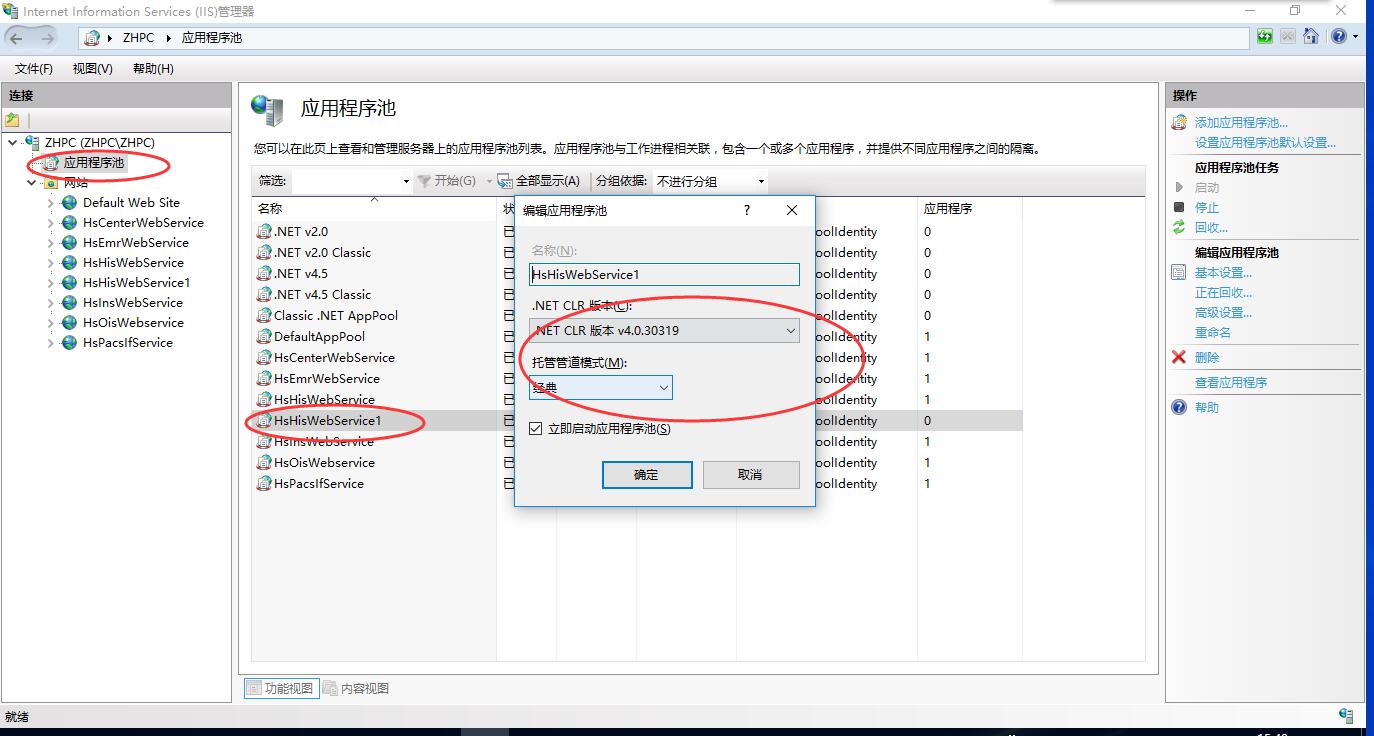
3、HsHisWebService D:\MIS\HSHIS5\HS.HIS.WebService\HS.HIS.WS.WebService



### 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)

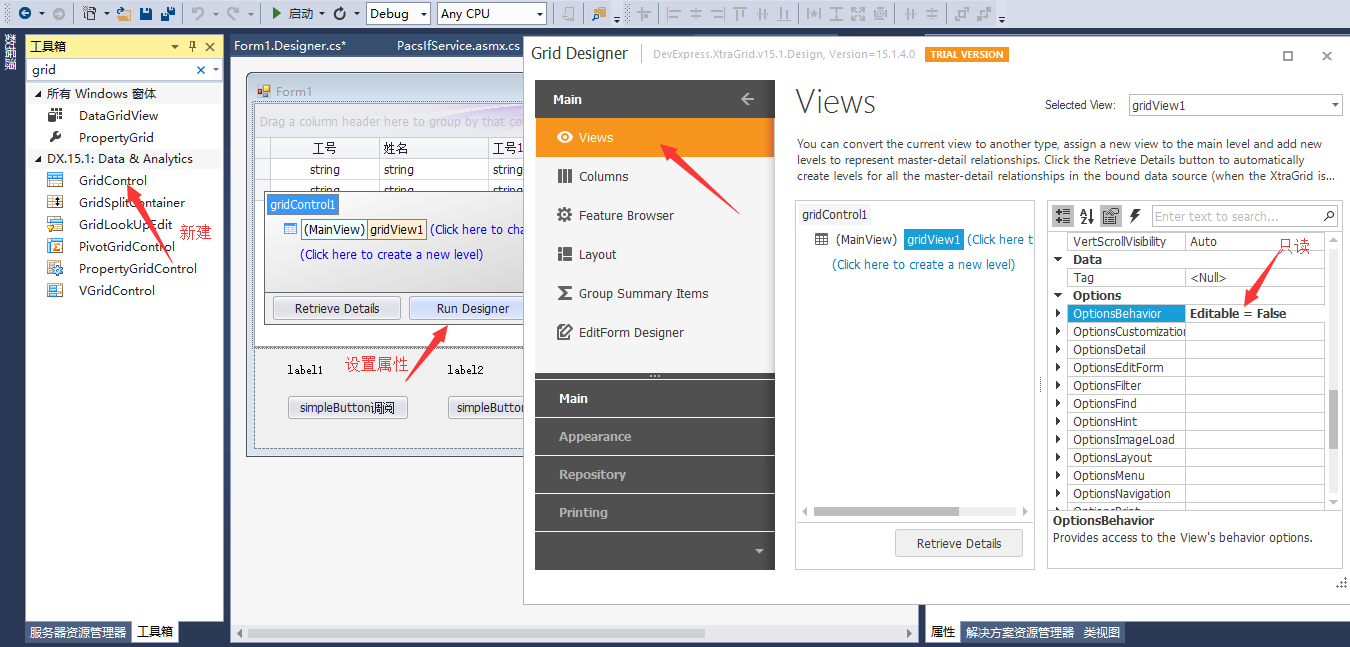
{

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

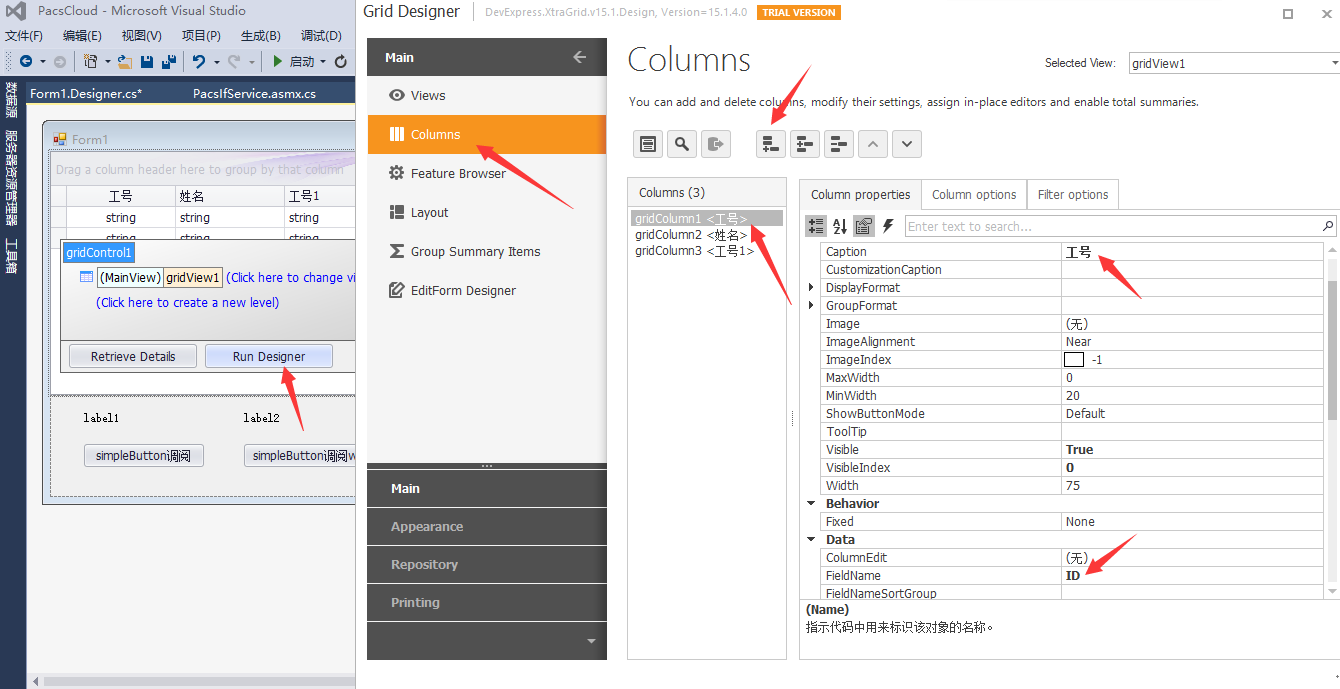
label2.Text = this.gridView1.GetFocusedDataRow()["DEPTID"].ToString();//指定列值

}

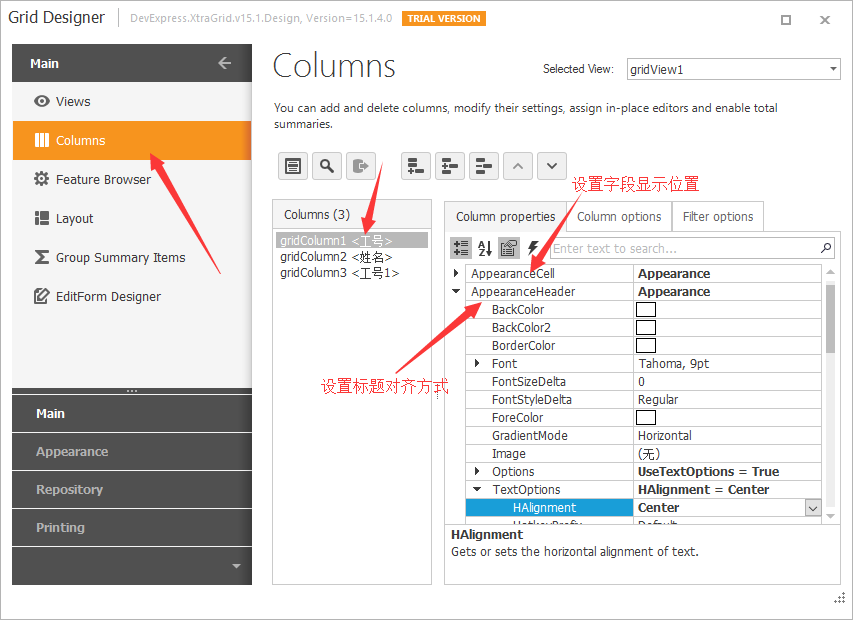
### 添加、设置为只读



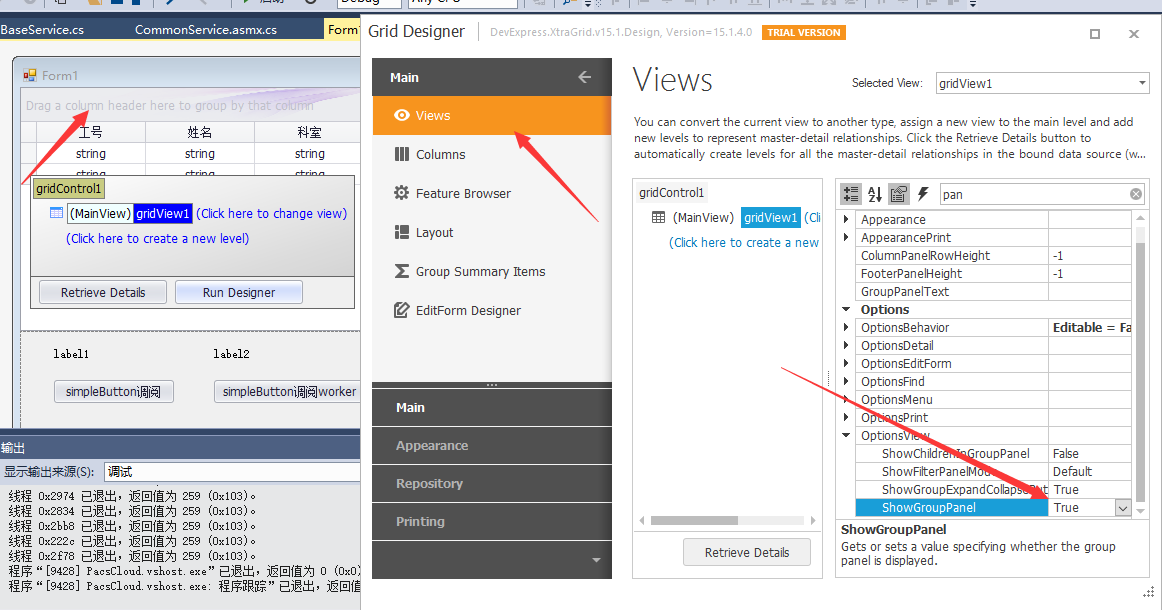
### 绑定字段



### 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

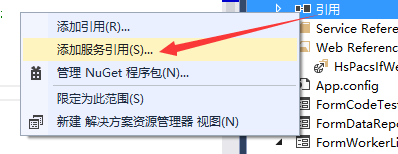
url: <http://222.216.218.90:8090/DTC_WebservicesXML/service/WebServices?wsdl>

方法：sjxz

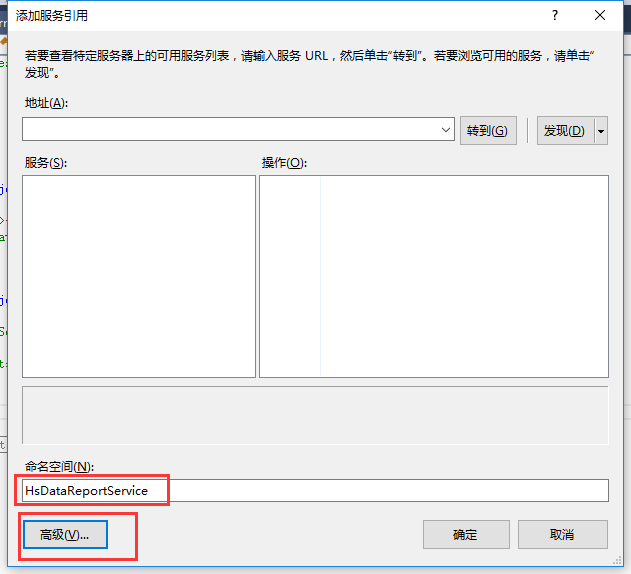
入参：<cx><id>22540</id><jystr>icd10</jystr></cx>

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>

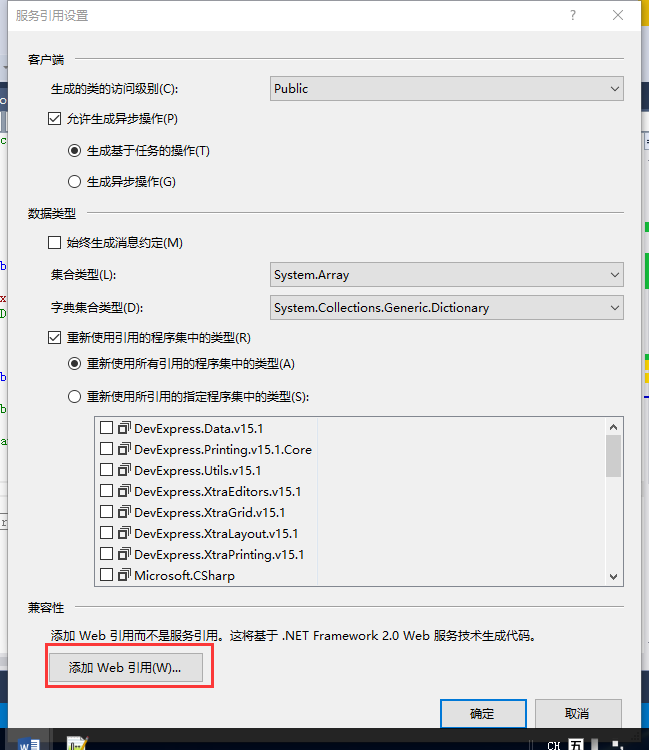
### 1、添加web引用



### 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

cd /d E:\TEST\Ctest

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

csc test.cs

test.exe

### Test.cs:

using System;

namespace gosoa.com

{

class MyFirstClass

{

static void Main()

{

Console.WriteLine("Hello world !");

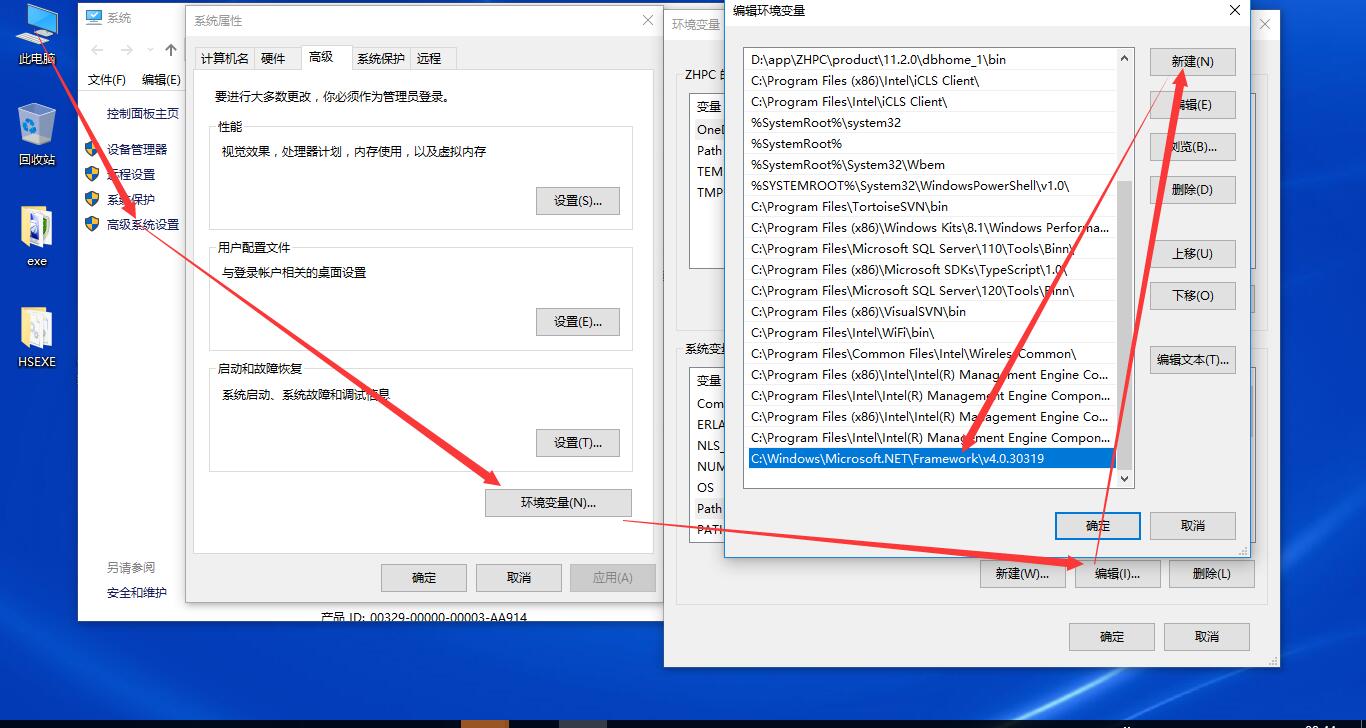
}

}

}

### “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);

}

}

}

### 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

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));

}

}

}

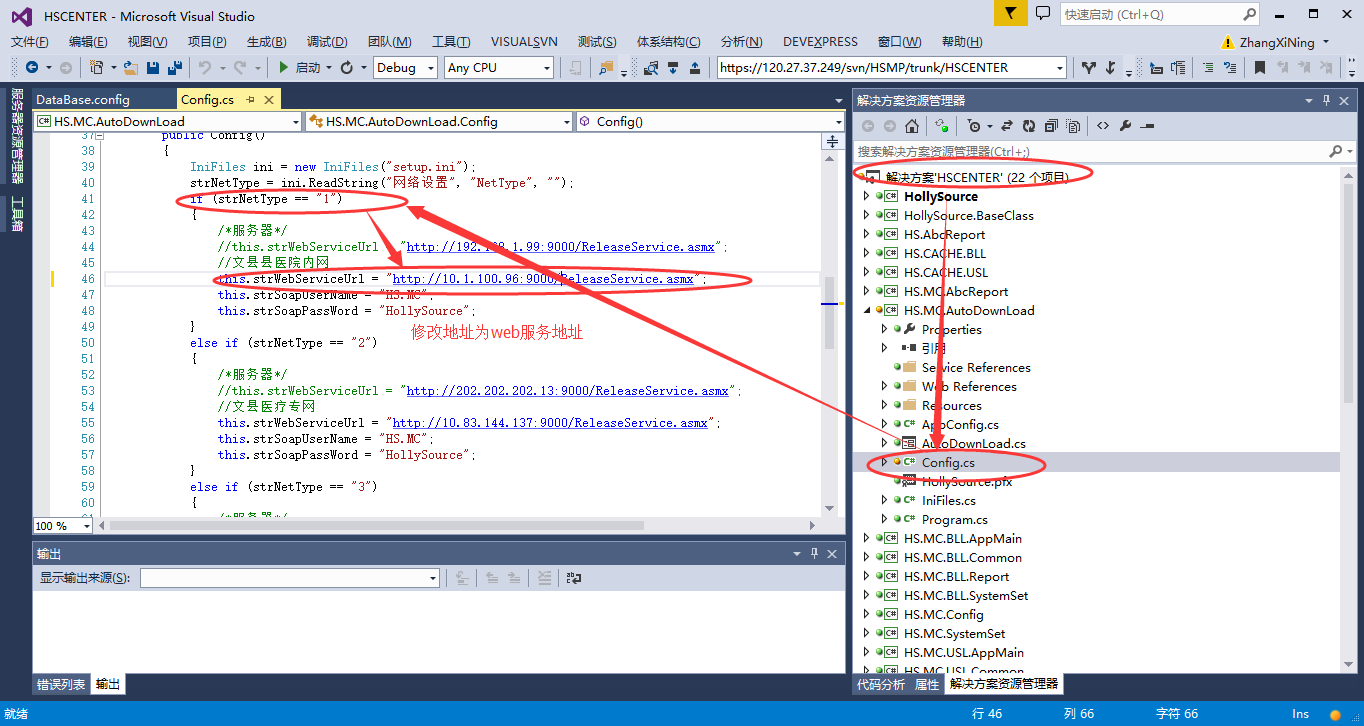
## 生成安装文件

### 所需文件

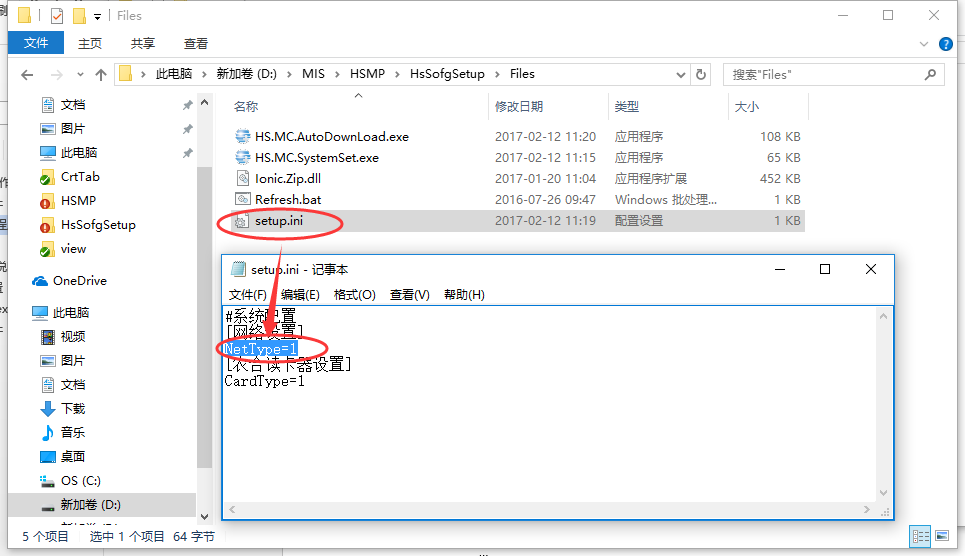
### 文件路径：

D:\MIS\HSMP\HSEXE

#### 1、HS.MC.AutoDownLoad.exe



#### 2、setup.ini



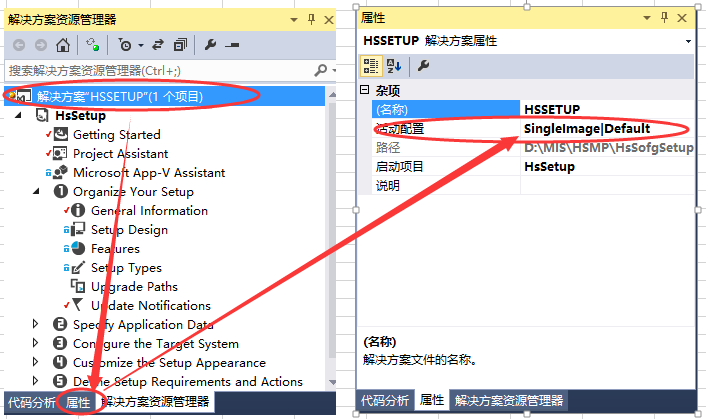
#### 3、Refresh.bat

#### 4、HS.MC.SystemSet.exe

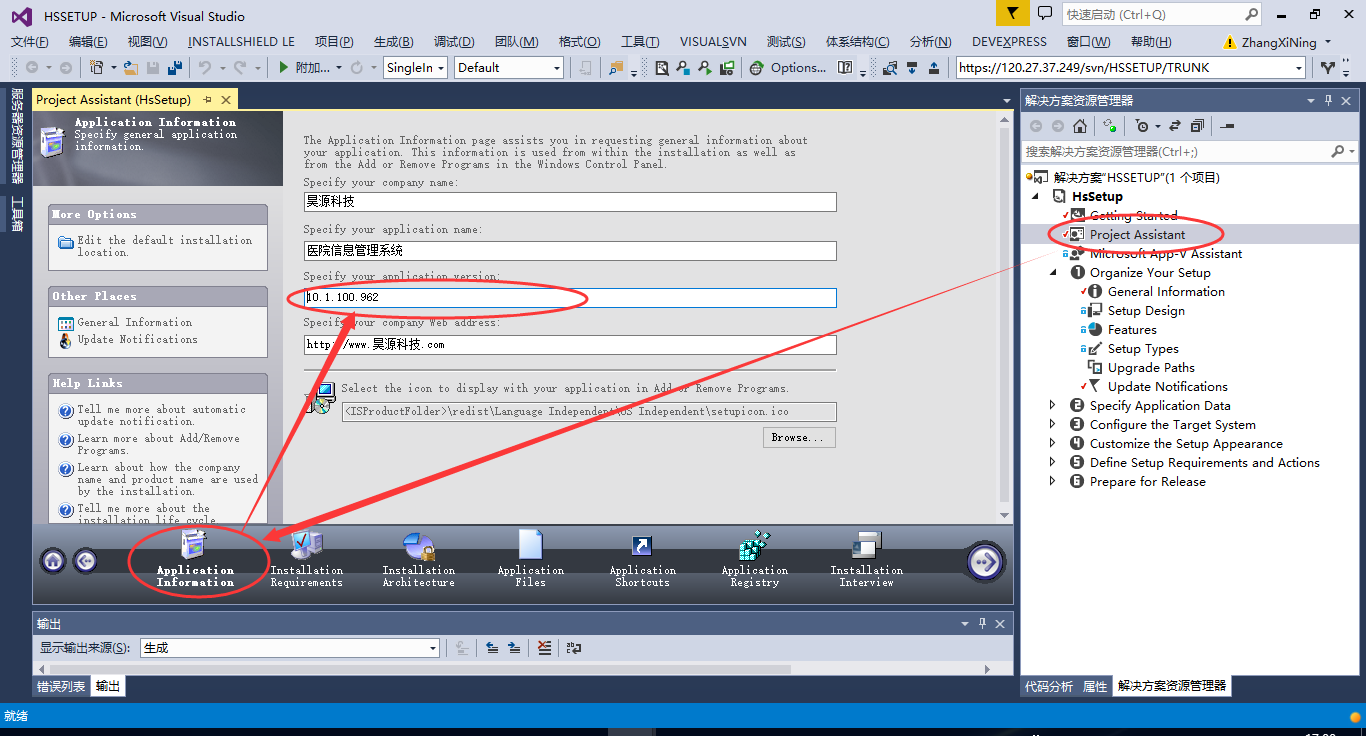
#### 5、Ionic.Zip.dll

### 相关设置

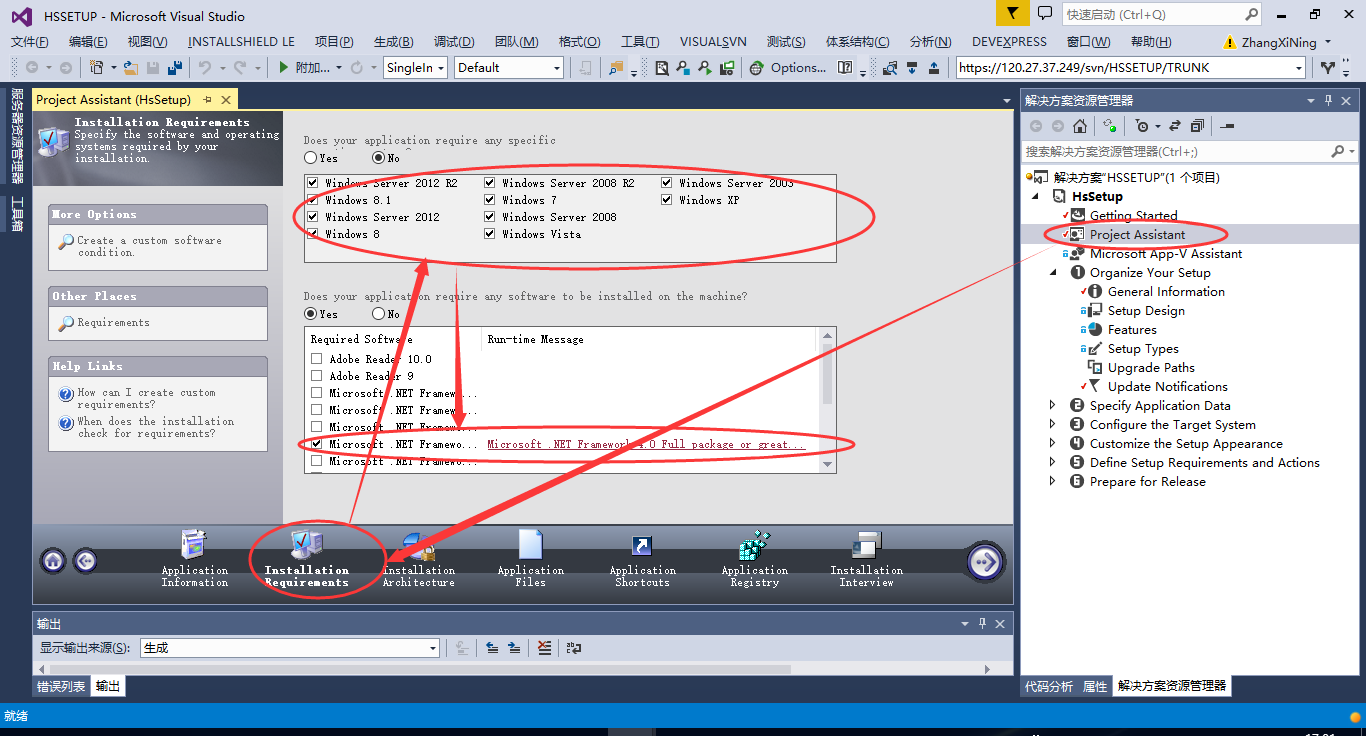
#### 输出类型：exe



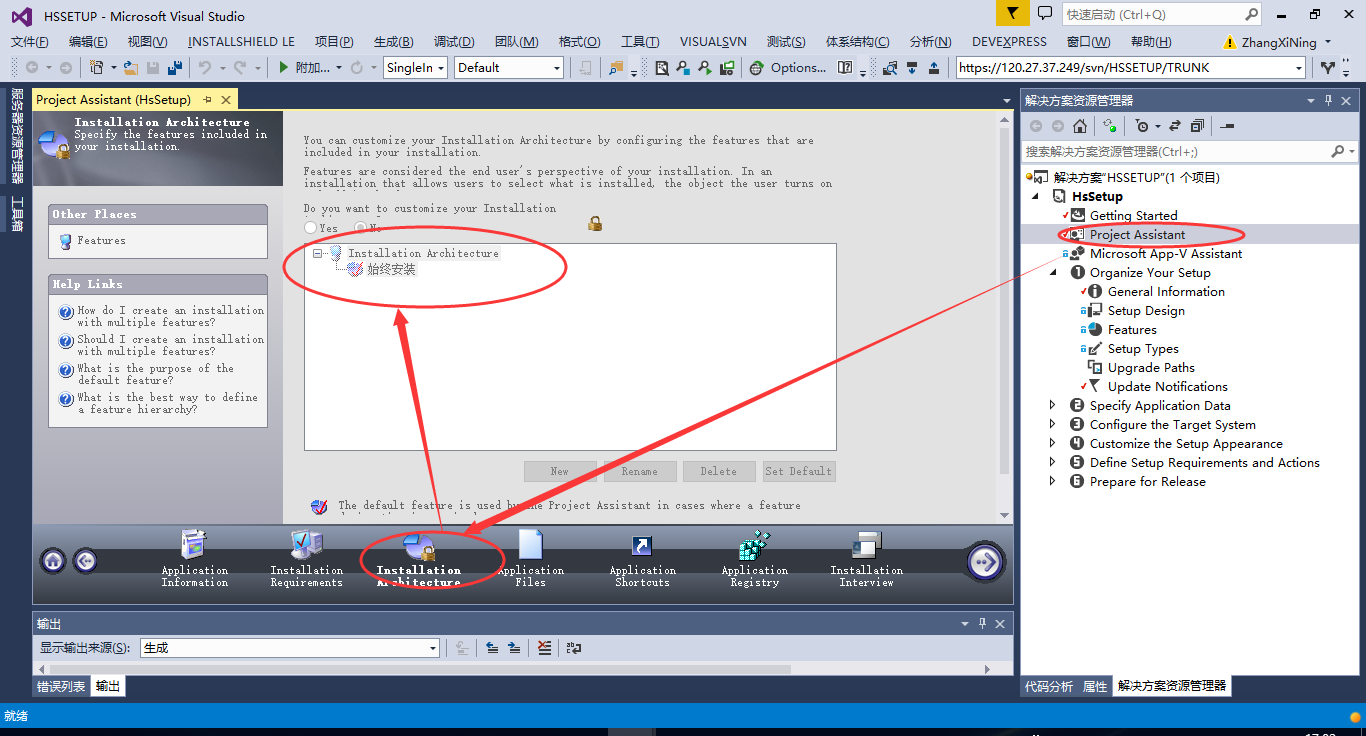
#### 设置1、版本号=web服务ip



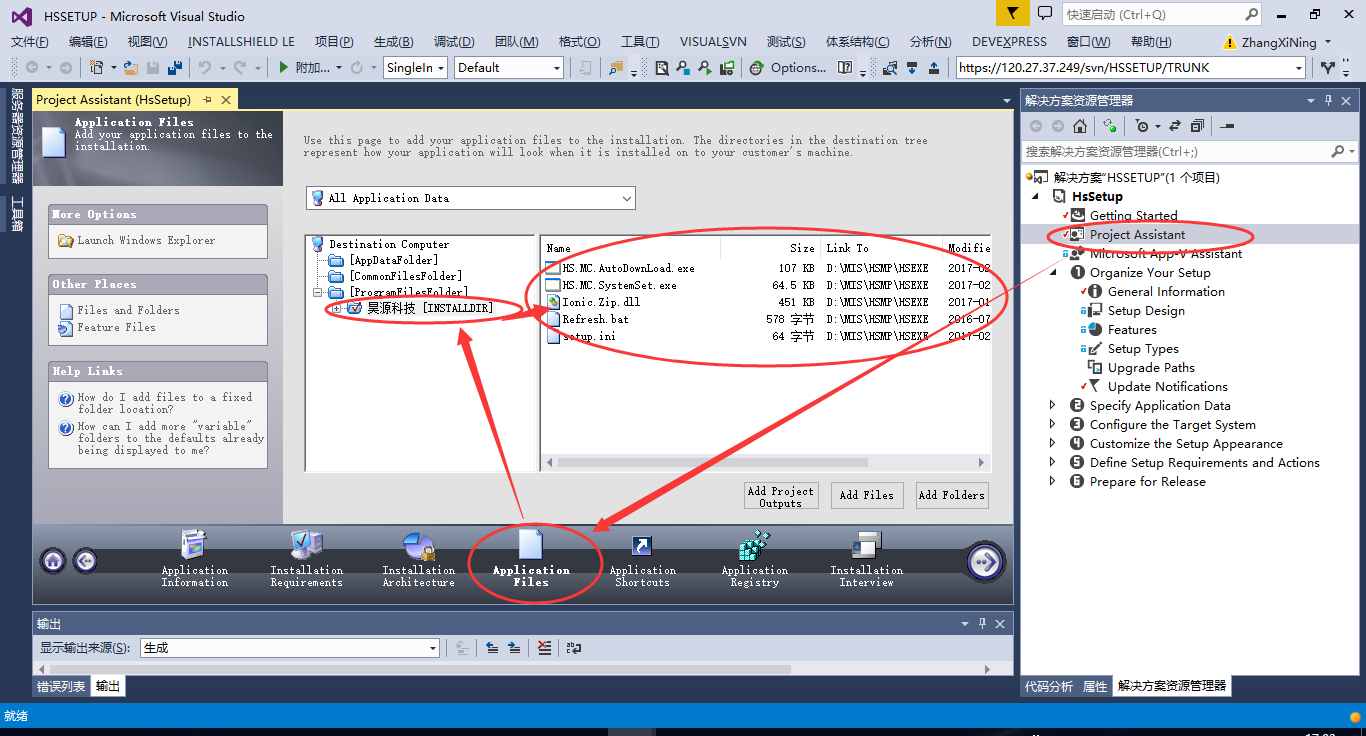
#### 设置2、



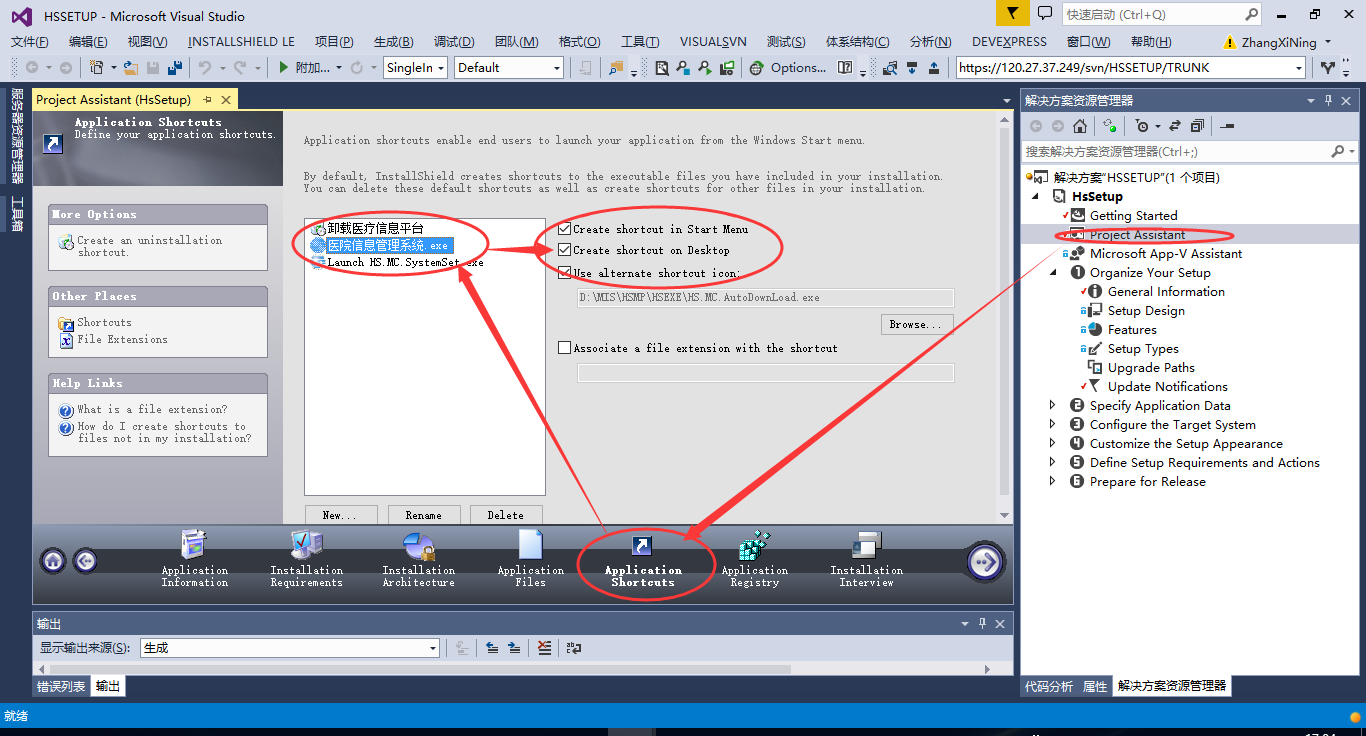
#### 设置3、



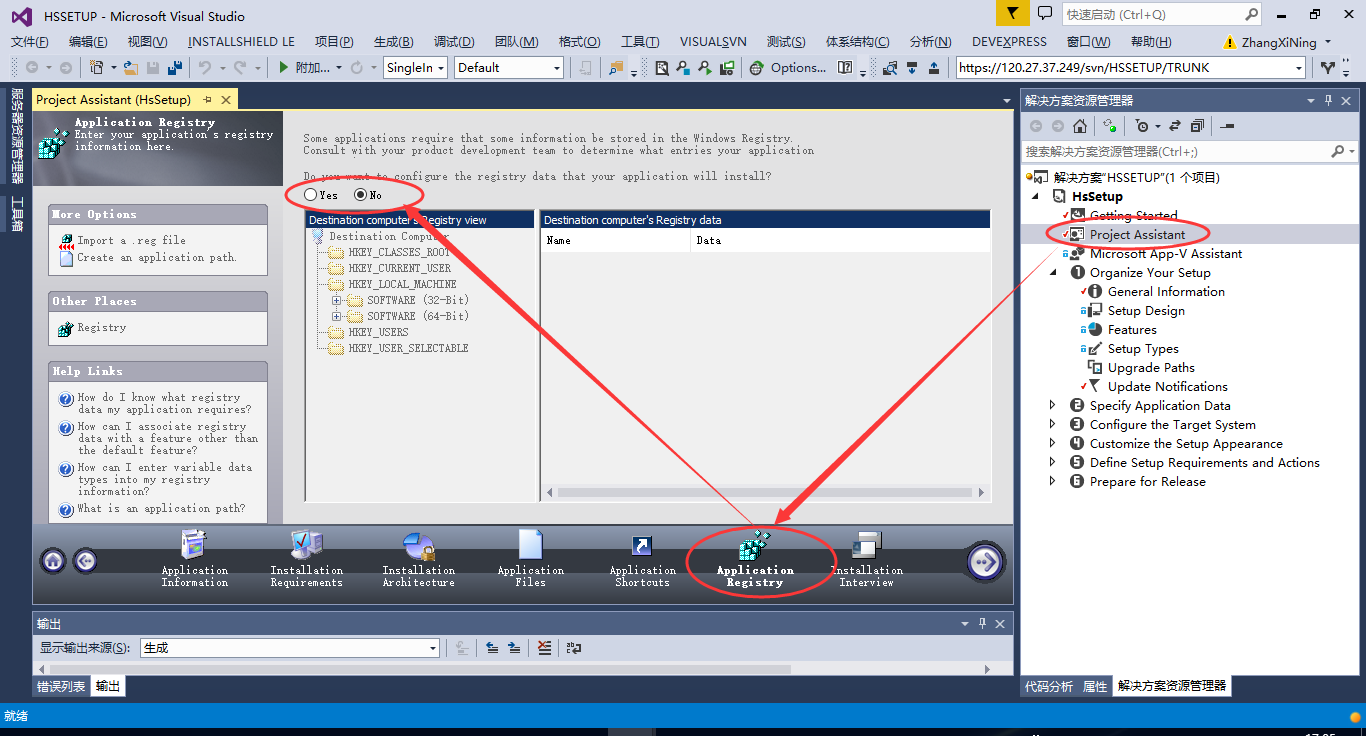
#### 设置4、所需文件



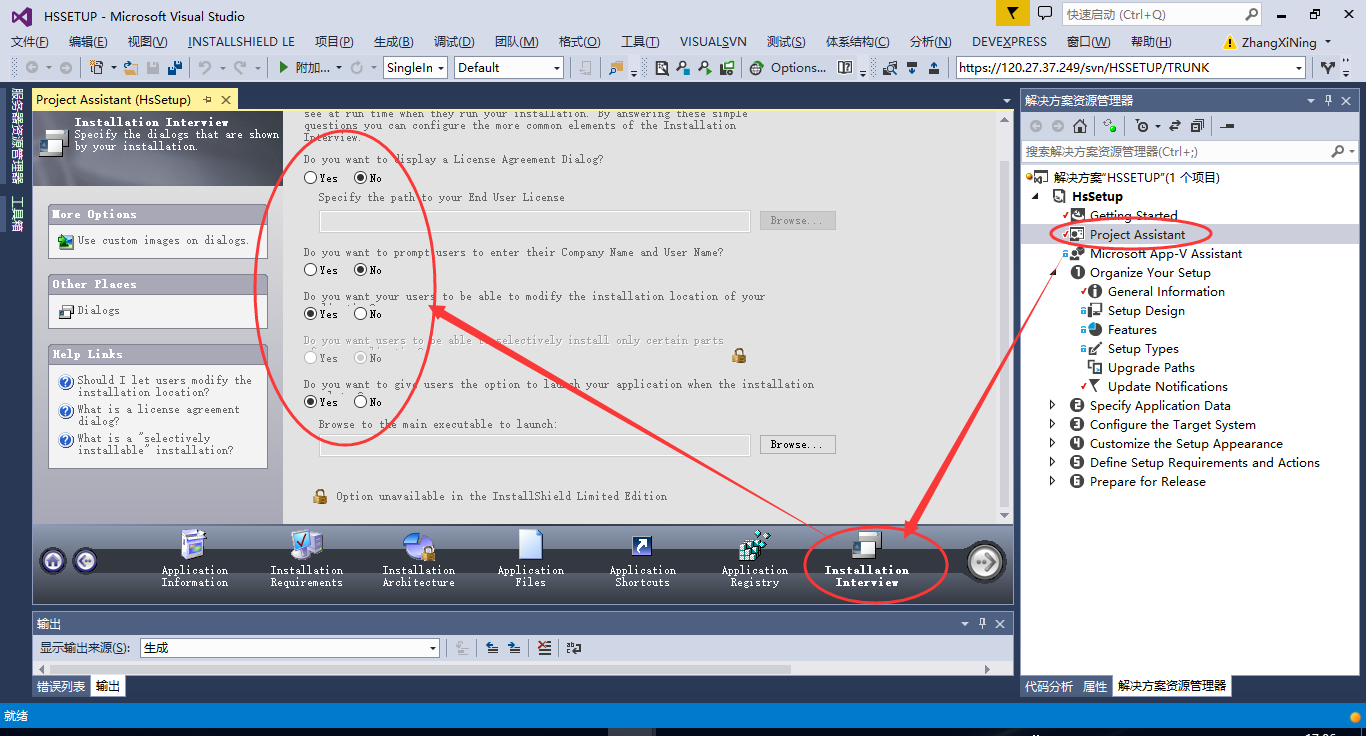
#### 设置5、快捷方式\菜单



#### 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)

## 其它