C＃基础（一）（温故而知新）

一、C#与.NET 的关系

C＃本身是一门语言，他是用于生成面向.NET 环境的代码，但其并不是.NET 的一部分。换言之，C#编写 的代码总是运行在.NET Framework 中。而且，在很多时候，C＃的特定功能依赖于.NET。比如，在 C＃中声 明一个 int 类型，实际上是.NET 中 System.Int32 类的一个实例。

.NET 的核心是 CLR(公共语言运行时)。在 CLR 中运行的代码，我们成为“托管代码”。

我们写的代码在运行之前，会经过两个编译阶段。

1、把源代码编译为 IL（Microsoft 中间语言）

2、CLR 把 IL 编译为平台专用的代码。

在这里，我们来简单说说“托管代码”。

IL 和 Java 的字节代码的理念基本是一样的，都是一种低级语言，都用数字表示，可以非常快速度的转 化为机器代码，这样执行效率会高很多，同时这也就是我们经常说的“平台无关性”的实现很 重要的一个环节。VB.NET C# J# 的源码都可以被编译成相同的 IL, 也就实现了平台无关。换言之，只 要.NET Framework 可以运行在任何设备或系统上，VB.NET C# J# 编写的代码都可以运行，而且都可以运 行在任何设备或者系统上。

VB.NET C# J# 的源码都可以被编译成相同的 IL，这样不同语言编写的不同类都可以很容易的相互操作 ，也就是所说的，语言的互操作性。

似乎这个 IL（Microsoft 中间语言）很好很强大，很神很奇特。我们接下来了解下这个 IL。

首先，简要的说说它的主要特征：

A、面向对象和使用接口

B、值类型和引用类型之间的巨大差别

C、强数据类型

D、使用异常来处理错误。

E、使用属性（attribute）
接下来具体说说这些特征。

（一）、面向对象和使用接口

面向对象，在此不做多的阐述了。主要在这里说说接口。关于接口，有一个很重要的理念：提供一个 契约，实现给定接口的类，必须提供该接口的所有方法和属性的实现。举个例子说明吧，有个接口 IUser ，其有 age,sex,userName 属性，有 setUname(),getAge()方法。有个类，UserClass 继承自 IUser。这时， UserClass 就必须实现 IUser 的所有方法 setUname(),getAge()和所有属性 age,sex,userName。

（二）、值类型和引用类型的巨大差异。

对于值类型，变量直接保存其数据，而对于引用类型，变量仅仅保存数据的地址。值类型一般存储在 堆栈中，引用类型一般存储在托管堆中。

（三）、强数据类型

所谓强数据类型，是指所有的变量都明确的被标记为某个特定的数据类型。比如 string 或者 int 等。IL 不允许对模糊数据类型执行操作。在 javascript 中的 var 变量就是模糊类型数据。

不过在现在的,.NET 3.5 中也加入了 var ，也就是模糊数据类型。

（四）、使用异常来处理错误。

C＃中以 try{} catch{}finally{}代码块来处理。这个具体在后面再谈。

（五）、属性的使用。

用户定义的 Attribute 和其对应的数据类型或方法的元数据放在一起，这对于文档说明很有用，它们和 反射技术一起使用，执行特定的任务。

在这里提及到的元数据以及反射，在后面我们将会详细讲述到。

C＃基础（二）（温故而知新）

二、程序集

程序集（Assembly）是包涵编译好的,面向.NET Framework 的代码的逻辑单元。程序集是完全自我描 述性的，也是一个逻辑单元。程序集的一个重要特点是其包含的元数据描述了对应代码中定义的类型和方 法。程序集也包含了自身的元数据。

程序集也有私有程序集和共享程序集之分。这些在后面我们将会学习到。

三。、反射。

因为程序集存储了元数据，包括在程序集中定义的所有类型和方法，所以可以编程访问这些元数据。 这种技术就成为反射。这种方法很有趣，我们在后面详细介绍。

四、命名空间

命名空间是.NET 避免类名冲突的一种方式。在大型开发中，往往都是团队开发，假如 A 公司开发了一个 user 类，而 B 公司也开发了一个 user 类，如果两个类要一起工作，就很可能产生冲突，这时候，命名空间 便能很好的解决问题。A 公司的 user 类的命名空间可以命名为 companyA，而 B 公司的 user 类可以命名为 companyB，这样即使两个类在一起工作也不会产生冲突，在实例化类的时候，我们采用 命名空间.类名 这样的方式来实例化的。即：companyA.user 这样的方式。

C＃基础（三）（变量的声明，变量的作用域以及常量）

从这一节开始，我们开始复习 C＃基础知识，涉及变量声明，变量的初始化，作用域，C#的预定义数据 类型，C#中的循环和条件语句，枚举，命名空间，Main()方法，C#标识符和关键字，C#编码的规范和约定 ，等等。

首先，我们还是从非常经典的“Hello word !”开始。

我们新建一个.txt 文件，并且更改为 test.cs 。然后键入以下内容。

```
using System;
namespace gosoa.com
{
        class MyFirstClass
        {
                static void Main()
                {
                        Console.WriteLine("Hello world !");
                }
        }
}
```

然后 打开 C#命令行编译器，编译这个文件。也就是在 C#命令行中输入

csc test.cs

编译成功后会在存放该文件的目录，生成一个 test.exe 文件。我们继续在 C#命令行中输入 test.exe 就 会运行 test.exe 文件，输出 Hello World !（说明：这里的 C＃命令行在 开始－程序－Microsoft .NET Framework SDK v2.0－SDK 命令提示）

在上个例子中，我们简单说明几个该注意的事项，首先，第一句 using System; 是用来引入 System 基 类。和 java 的 import 相似，这是 C#的基类，C#的所有工作都依赖于该基类。第二句 namespace gosoa.com 是我们之前提到的命名空间，命名空间为 gosoa.com,当然，你可以命名为任意名称。但，为了避免冲突， 我们一般采用自己公司的域名做为命名空间。第三句，class MyFirstClass 是声明一个名称为 MyFirstClass 的类。第四句，static void

Main() 是程序的主方法，注意，这里的 Main() 方法首字母大写喔。第五句，Console.WriteLine ("Hello world !"); 就是输出 Hello world ，

Console.WriteLine 就是基类中的一个方法。

在 C#中，和其他很多语言(C,JAVA 等)一样，句子末尾以分号”;”结束，程序代码都放在{}一对大括号 中。

一、变量

1.1 变量的声明

C#中变量的声明我们以实例来说明，例如，int i; 该句声明了一个 int (整型) 变量 i 。再如，string str; 这句声明了一个 string(字符串类型)的

变量 str 。

1.2 变量初始化

C#编译器需要每个变量在有了初始值之后才能使用该变量。

在 C#变量初始化时有两点需要注意，

A 、变量是类或者结构中的字段，如果没有显式的初始化，在默认状态下，创建这些变量时，其初始 值就是 0；例如下面的代码：

```
using System;
namespace gosoa.com
{
        class MyFirstClass
        {
                static int y;
                static void Main()
                {
                        Console.WriteLine(y);
                }
        }
}
```

我们在类中声明了一个变量 y，然后输出该变量，编译并运行后我们会看到输出的结果是 0 。

B、方法中的变量，必须显式的初始化，否则在使用该变量的时候会出错。如下面的代码：在编译的时 候就会报错。我们需要把 int y ; 显式的初始化

，才会通过编译。比如我们初始化 y 的值为 10 ，即 int y=10; 便会通过编译。

using System;

```
namespace gosoa.com
{
	class MyFirstClass
	{
		static void Main()
		{
			int y;
			Console.WriteLine(y);
		}
	}
}
```

1.3　　变量的作用域

变量的作用域是指可以使用该变量的代码区域。一般情况下，确定作用域有如下规则。

A、只要变量所属的类在某个作用域内，其字段（也叫做 成员变量）也在该作用域中。

B、局部变量存在于声明该变量的块语句或方法结束的大括号之前的作用域。

C、在 for、while 循环中声明的变量，只存在于该循环体内。

在变量使用中，可能产生命名冲突的情况，首先，我们来看下局部变量的作用域冲突。如下代码示例 ：

```
using System;
namespace gosoa.com
{
	class MyFirstClass
	{
		static void Main()
		{
			for(int i=0;i<10;i++)
			{
				Console.WriteLine(i);
			}
			for(int i=0;i<20;i++)
			{
				Console.WriteLine(i);
			}
		}
	}
}
```

两个循环中都使用了 i ，但都可以正常输出，因为每个 i 的作用域都在其对应的两个循环体内。

再看下例代码：

```
using System;
namespace gosoa.com
{
        class MyFirstClass
        {
                static void Main()
                {
int j=5;
                        for(int i=0;i<10;i++)
                        {
                                int j=20;
                                Console.WriteLine(i+j);
                        }
                }
        }
}
```

这段代码编译就会出错，因为第一个 j 在作用域是整个 Main()方法，这样，其在循环体内也是有效的 。于是，在循环体内定义一个同名的 j 时，就会报错了。

我们再看如下示例代码，

```
using System;
namespace gosoa.com
{
        class MyFirstClass
        {
                int j=30;
                static void Main()
                {
                        int j=20;
                        int i=5;
                        Console.WriteLine(i+j);
                }
        }
}
```

在这段代码中，第一个 j 的作用域是整个类，也就是类的字段，第二个 j 的声明会替代第一

个 j，所以该 程序会输出 25.

1.4 常量

在声明变量时，在变量前面加上 const 关键字就可以把该变量指定为一个常量。

在这里需要注意几点，

A 常量必须在声明的时候就初始化，而且其赋值后就不能再更改了。

B 常量总是静态(static)的，不必在声明常量时添加 static 关键字。

C＃基础（四）（C#预定义值类型和引用类型）

一、预定义类型。

1、值类型和引用类型

C#中的数据类型，可以分为值类型和引用类型，值类型存储在堆栈上，而引用类型存储在托管堆上。

如下代码示例，

```
int i=10;

int j=i;
```

i 和 j 的值都是 10，并且在内存中会有两个地方存储 10.

再看下面的代码

```
Vector x=new Vector();
x.Value=20;
Vector y=x;
Console.WriteLine(y.Value);
y.Value=50;
Console.WriteLine(x.Value);
```

Vector 是一个引用类型，引用类型在使用的时候需要 new 来实例化一个。这段代码执行后，只有一个 Vector 对象，x,y 都指向包含该对象的内存地址。因为 x，y 存储的都是对象的引用，所以当 y 改变的时候， x 也会改变。所以该程序输出的结果是 20 和 50.

如果变量是一个引用，就可以把其值设置为 null，表示不指向任何对象。

2、CTS 类型。

C#的预定义类型并没有内置于语言中，而是内置于.NET Framework 中，比如声明一个 int 类型时，实际 上是.NET 结构 System.Int32 的一个实例。这说明，可以把所有的基本数据类型看作是支持某些方法的类。\

3、预定义的值类型

A、整型

sbyte ，8 位有符号的整数，范围从 -128 到 127.

byte ，8 位无符号的整数，范围从 0 到 255.

short，16 位有符号的整数，范围从 -32768 到 32767

ushort ，16 位无符号的整数，范围从 0 到 65535

int，32 位有符号的整数，范围从-2147483648 到 2147483647

uint，32 位无符号的整数，范围从 0 到 4294967295

long，64 位有符号的整数，范围从-2 的 31 次方到 2 的 31 次方减 1

ulong，64 位无符号的整数，范围从 0 到 2 的 64 次方减 1

B、浮点类型。

float ，32 位单精度浮点数。

double，64 位双精度浮点数。

如果代码对某个非整数值，如 12.3 硬编码，则编译器一般假定该变量是 double，如果想指定其为 float ，则可以在后面加上字符 f。

C、decimal 类型。

该类型是一种财务专用数据类型，是 128 位高精度十进制表示法。

要把数据指定为 decimal 类型的，只需在数字后面加上 M(或者 m)

A、 bool 类型。

C#的 bool 类型包含 true 和 false。

B、 字符类型。

也就是 char 类型，表示一个 16 位的 unicode 字符。

char 类型的字面量是采用 单引号 括起来的。而不是双引号。双引号括起来的是字符串类型的。

4、预定义引用类型。

A、object 类型。

这是 C#的基类，所有的类都派生自它。所以，可以使用 object 引用绑定任何子类型的对象，object 类 型执行许多基本的一般用途的方法，如 Equals() GetHashCode(),GetType()等，我们需要针对某些方法 进行“重写”，这在后面我们将会学习到。

B、string 类型。

注意，string 类型是属于引用类型。我们来看下面一段代码，在修改一个字符串的时候，实际上是创 建了一个新的字符串，而并非修改了原来在字符串。我们来看一个示例：

```
using System;
using System.Windows;
namespace gosoa.com
{
    class MyFirstClass
    {
        static void Main()
        {
            string str1="GoSoA.com.cn";
            string str2=str1;
            Console.WriteLine("str1="+str1);
            Console.WriteLine("str2="+str2);
            str1="www.GoSoA.com.cn";
            Console.WriteLine("str1="+str1);
            Console.WriteLine("str2="+str2);
        }
    }
}
```

在这个示例中会输出

str1="GoSoA.com.cn";

str2="GoSoA.com.cn";

str1="www.GoSoA.com.cn";

str2="GoSoA.com.cn";

这和我们所预期的引用类型正好相反，为什么呢？

因为当我们用“GoSoA.com.cn”来初始化 str1 的时候，就在堆上分配了一个

string 对象，当初始化 str2 的时候，也指向了这个对象。当 str1 改变的时候，并不是修改了

原有的对 象，而是新创建了一个对象，但 str2 还是指向原来的对象，所以，str2 的值并未改变。

C＃基础（五）（C#条件，循环和判断）

一、条件语句

1、if 语句

我们来看个示例

```
using System;
using System.Windows;
namespace gosoa.com.cn
{
	class MyFirstClass
	{
		static void Main()
		{
			string str=Console.ReadLine();
			if(str=="GoSoA")
			{
				Console.WriteLine("www.GoSoA.com.cn");
			}else if(str=="163")
			{
				Console.WriteLine("www.163.com");
			}else if(str=="sina")
			{
				Console.WriteLine("www.sina.com");
			}else
			{
				Console.WriteLine("www.cnblogs.com");
			}
		}
	}
}
```

在上面的代码示例中，我们可以看见，if else 的用法。很简单的哈。

2、switch 语句

假如，需要判断的条件很多，我们可以视情况而定，考虑使用 switch 语句。

我们看下面的示例，

using System;

```
using System.Windows;
namespace gosoa.com.cn
{
	class MyFirstClass
	{
		static void Main()
		{
			string str=Console.ReadLine();
			switch(str)
			{
				case "gosoa":
					Console.WriteLine("www.gosoa.com.cn");
					break;
				case "163":
					Console.WriteLine("www.163.com");
					break;
				default:
					Console.WriteLine("www.cnblogs.com");
					break;
			}
		}
	}
}
```

在这里，我们需要注意，case 后的值，必须是常量，不可以是变量。而且 case 结尾是冒号，每个 case 后都必须跟有一个 break 。句中 default 是在所有的 case 语句都失败的情况下执行的操作。

我们再看个例子，

```
using System;
using System.Windows;
namespace gosoa.com.cn
{
	class MyFirstClass
	{
		static void Main()
		{
			string str=Console.ReadLine();
			switch(str)
			{
				case "sina":
				case "google":
				case "gosoa":
```

```
                Console.WriteLine("www.gosoa.com.cn");
                break;
            case "163":
                Console.WriteLine("www.163.com");
                break;
            default:
                Console.WriteLine("www.cnblogs.com");
                break;
        }
    }
  }
}
```

在这段示例代码中，前两个 case 后都没有执行语句，这样是允许的，但该段代码会输出 www.gosoa.com.cn，因为，在 case 没有执行语句的时候，会依次执行下去，直到第一个有执行语句的 case 。所以，就输出了 www.gosoa.com.cn。

在这里，还有个 goto 语句，但，我们不提倡使用，在这里就不说了。如果您有兴趣，可以去 google 下。^_^。

二、循环

1、for 循环。

我们还是以示例来讲解。

```
using System;
using System.Windows;
namespace gosoa.com.cn
{
    class MyFirstClass
    {
        static void Main()
        {
            for(int i=0;i<100;i++)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```

我们来看示例中的，for(int i=0;i<100;i++) 这段代码，这就是 for 循环。其执行过程是这样的：首先，初始化一个变量 i=0，然后判断，i 是否小于 100，如果成立，则执行 Console.WriteLine(i)

这句， 执行完后在 i++ （i++的意思是，i=i+1）。现在 i 的值变成了 2，接着判断，i 是否小于 100，再接着执行 。。。。直到 i 小于 100 不成立，就会退出循环。

2、while 循环

while 循环，我们也以例子来学习。

```
using System;
using System.Windows;
namespace gosoa.com.cn
{
        class MyFirstClass
        {
                static void Main()
                {
                        int i=0;
                        while(i<100)
                        {
                                Console.WriteLine(i);
                                i++;
                        }
                }
        }
}
```

While 的执行过程是这样的，第一步就直接判断，i 是否小于 100，如果成立，则执行 Console.WriteLine(i) 和 i++ 语句。接着进行判断，一直到 i 小于 100 不成立。

3、do…while 循环

我们依然来看示例代码。

```
using System;
using System.Windows;
namespace gosoa.com.cn
{
        class MyFirstClass
        {
                static void Main()
                {
                        int i=200;
                        do
                        {
                                Console.WriteLine(i);
```

```
                    i++;
                }
                while(i<100);
            }
        }
}
```

该段程序，输出的结果是 200.因为 do while 的执行过程是 先执行再判断，所以，就先输出了 200.

4、foreach 循环

我们看示例

```
using System;
using System.Windows;
namespace gosoa.com.cn
{
        class MyFirstClass
        {
            static void Main()
            {
                string [] strArr={"www","gosoa","com","cn"};
                foreach(string temp in strArr)
                {
                    Console.WriteLine(temp);
                }
            }
        }
}
```

首先我们定义了一个 strArr 字符串数组。foreach(string temp in strArr) 就是来遍历该数组，并 且输出数组中的每一个项。该示例输出的结果是 www, gosoa ,com ,cn 三列。

三、跳转语句

1、break 这个关键字我们在上个示例中见到过了，是用来退出某个 case 语句的。实际上，break 也可 以退出 for foreach while 等循环。

2、continue ，和 break 类似，只是 break 直接跳出了迅，而 continue 不会跳出循环，只是该次循环不 执行，直接执行下次循环。我们看一段示例。

```
using System;
using System.Windows;
```

```
namespace gosoa.com.cn
{
        class MyFirstClass
        {
                static void Main()
                {
                        string [] strArr={"www","gosoa","com","cn"};
                        foreach(string temp in strArr)
                        {
                                if(temp=="www")
                                {
                                        continue;
                                }
                                Console.WriteLine(temp);
                        }
                }
        }
}
```

该示例输出的结果是 gosoa ,com ,cn 三列。和上个示例的不同之处，就是少了 www。

3、return 语句。

该语句一般用于退出类，或者方法的。如果方法有返回类型，则 return 语句必须返回这个类型的值。 如果没有返回值，就直接 return 就可以了。

C#基础（六）（枚举，数组，命名空间）

一、枚举。

枚举是用户定义的整数类型。在声明一个枚举类型时，需要指定该枚举可以包含的一组可以接受的实 例值。

我们看个示例。

```
using System;
using System.Windows;
namespace gosoa.com.cn
{
    class MyFirstClass
    {
        static void Main()
        {
                int userAage=(int)user.fatherAge;
                Console.WriteLine(userAage);
         }
         public enum user
        {
            Age=18, //年龄
            gread=2, //年级
            fatherAge=65 //父亲的年龄
         }
    }
}
```

在示例中，会输出 65. public enum user 就是声明了一个 user 的枚举类型。Age,gread,fatherAge 就是它的三个选项。分别赋有三个特定的值。int userAage=(int)user.fatherAge; 这是定义了一个 userAage 并且给其赋值为 user.fatherAge，也就是 65.

我们再来看个小示例

```
static void Main()
        {
            user userTemp=user.gread;
            Console.WriteLine(userTemp.ToString());
         }
```

在这里示例中，输出的是 gread 字符串。为什么呢？因为枚举在后台会实例化为派生于

System.Enum 的结构，这表示可以对其进行调用方法，执行一些操作。上面的例子就是一个说明。

二、数组。

我们先来声明一个整型数组。int [] userCount ;

int [] 这就表示了一个整型数组，userCount 是数组名称。

我们再看一种声明方式： int [] userCount=new int [20];

这个例子声明了一个大小为 20 的整型数组。

注意，所有的数组都是引用类型。

数组，我们就在这里简单的介绍这么一点，在后面我们和集合一起详细的学习数组。

三、命名空间。

我们先来看个示例

```
using System;
namespace gosoa.com.cn
{
    public class MyFirstClass
    {
        public    string getUrl()
        {
            return "gosoa.com.cn";
        }
        static void Main()
        {
                www.gosoa.com.cn.MyFirstClass    urlClassNew=new
www.gosoa.com.cn.MyFirstClass ();
            string url=urlClassNew.getUrl();
            MyFirstClass MyFirstClassNew =new MyFirstClass();
             string url2=MyFirstClassNew.getUrl();
            Console.WriteLine(url);
             Console.WriteLine(url2);
          }
    }
}
namespace www.gosoa.com.cn
{
    public class MyFirstClass
```

```
    {
        public string getUrl()
        {
            return "www.gosoa.com.cn";
        }
    }
}
```

在这个示例中，有两个 namespace 。 注意，两个 namespace 中的类名称是一样的，而且都有个 getUrl() 方法。但在第一个 类的 Main()方法中，我们调用这两个方法的时候，并没有报错，正是因为两 个类分别位于两个不同的命名空间中，从而避免了类名的冲突问题。

最后输出的结果是 www.gosoa.com.cn 和 gosoa.com.cn

在这个例子中，gosoa.MyFirstClass urlClassNew=new gosoa.MyFirstClass(); 是用来实例化一个 gosoa.MyFirstClass 这个类的。我们要访问类，就需要使用 命名空间+”.”+类名 这样的方 式访问。

如果命名空间名称很长，就会很显得冗长，于是，我们可以采用 using 语句简化。

我们看到，在每个类前面第一句总是 using System; 这是因为所有的 c＃的许多类都包含在 System 命名空间中。

我们也可以给命名空间起个别名。比如，在上个例子中，我们可以使用

using gosoa=www.gosoa.com.cn ； 这样来引入 命名空间。

C#高级（一）面向对象

对于面向对象，有 N 多大师写过 N 多文字。我也不敢妄言。就简单说说自己的认识。

我们先来看看“对象”，什么是对象？我们在现实生活中所能看到的一切都可以称为对象 。比如，企业、医院、宠物、植物、人。。。等等。在面向对象编程中，对象往往被当作一个类，类有属 性和行为。我们以医院为例来说，医院有医生，有护士等，在医院可以做 CT，B 超等。在面向对象编程中 ，我们定义一个 hospital 类，其有属性（医生，护士）和行为（做 CT,做 B 超）。这就是一个简单的面向 对象编程。

其实，我们不光可以把医院做为一个类，我们还可以把医生做为一个类，医生有其属性（年龄，姓名 ，性别等）和其行为（检查病人，开处方，做手术等）。甚至我们可以把儿科医生定义为一个类，同样， 骨科，外科，内科等医生，我们都可以分别定义为一个类，但这些类都具有同样的属性（年龄，姓名，性 别等）甚至行为，难道每个类我们都要依次定义这些属性吗？答案是否定的。

我们可以定义一个医生类，定义这些共有属性和行为。儿科医生，骨科医生等都继承自这个医生类， 这样就具有了医生类的所有属性和行为。这在面向对象编程中被称为继承。

我们再来看一个例子。

人，有白人，黑人，黄种人等。人有年龄，性别，会吃饭。狗有白毛的狗，有黑色的狗等，狗有年龄 ，性别，会吃饭，咬人。

在这种情况下，我们可以把人定义为一个基类（有属性 年龄，性别和行为吃饭等），黑人，白人，黄 种人都继承自“人”这个类。我们再“狗”定义为一个类（有属性 年龄，性别和行为吃饭，咬人等），白狗，黑狗都继承于这个“狗”类。

在这里，我们可以看到，狗所具有的一些属性在“人”这个类里面已经具有了，我们没有 必要重新在“狗”类中定义的，“狗”类中只需要定义其独特的行为或属性，比如 ：咬人。而黑狗，白狗等继承自“人”和“狗”这两个类。这样就不存在重复定义 的问题了。

但在 C#和 Java 中都不支持多重继承。那怎么办呢？这时候就诞生了一个“接口”的概念。一个类可以实现多个接口。在上例中，可以定义一个“动物”的接口（人也是一种动物嘛），在接口中定义人和狗共有的年龄，性别，吃饭等属性和行为。再定义一个“人”这样的基类，并实现这个接口，然后白人，黑人等类再继承“人”这个基类就可以了。狗也同理。

呵呵。就简单的介绍这么些吧。可能说的不清楚，大家去 google 一下，有很多介绍面向对象的文字。

C#高级（二）类

一、类的概述

类，是创建对象的模板，每个对象都包含数据，并且提供了处理和访问数据的方法。换言之，类，定 义了每个对象，也就是“实例”包含什么数据和功能。

比如我们定义一个“医生”类，并且实例化一个。我们看下面的代码：

```
using System;
namespace gosoa.com.cn
{
public class Doctor
{
public Doctor(){}
public Doctor(string name,byte age)
{
this._name=name;
this._age=age;
}
private string _name;
private byte _age;
public string Name
{
get{return this._name;}
set{this._name=value;}
}
public byte Age
{
get{return this._age;}
set{this._age=value; }
}
public string doSth()
{
return "我会给人治病喔～～";
}
public static string doAnth()
{
return "执行的另一个静态方法";
}
}
public class OneDoctor
{
```

```
static void Main()
{
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());
Console.WriteLine(Doctor.doAnth());
}
}
}
```

在这个例子中，public class Doctor 便是声明了一个类。_name 和_age 是其两个属性。doSth()是其 的一个方法（即对象的行为）。 Doctor dc=new Doctor() 用来实例化了一个 Doctor 类，也就类似实例化 了一个对象，产生了一个新医生。Doctor dc2=new Doctor("张三",35);是实例化的另外一个类，也就是 另外一个医生。

在 Doctor 类中，public Doctor(){} public Doctor(string name,byte age) 这两个方法叫做 构造函数。是用来初始化类的，在每个类被实例化的时候，会自动调用。

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

这段代码是用来设置和获取类的属性的。也就类似 java 中的 getName 和 setName 方法。只是在 C#中 这变得更容易了。

注意一点：类是存储在托管堆上的引用类型。

二、方法

1、 方法概述

方法和 C 语言中的 函数共享同一个理念。一直以来，我们在用的 Main()方法就是个例子。还有上例中 public string doSth() 也是一个方法。其中，public 是 类的修饰符，string 是方法的返回值，也可以 没有返回值，即 void ，doSth 是方法名称。()括号必须有，在括号中可以有参数，如 Doctor 类的构造函 数 public Doctor(string name,byte age) 就有两个参数。方法体则必须用一对{}括起来。

方法的调用，则需要先实例化类，然后调用类的某个方法。上例中 Doctor dc=new Doctor(); 来实例化 了类，然后 dc.doSth() 就是调用了 Doctor 类的方法。

如果方法是静态的，即 static，则不需要实例化类，直接使用 类名.方法名 就可以调用了。如上例 中 Console.WriteLine(Doctor.doAnth()); 即是直接调用了静态的 doAnth 方法。

2、方法的参数

参数可以通过引用或者值传递 给方法。具体有什么区别呢？

我们来看个例子。

```
using System;
namespace gosoa.com.cn
{
public class OneDoctor
{
static void FunctionTest(int [] arr, 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, 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。

3、ref 参数。

我们把 上例中的方法修改为 static void FunctionTest(int [] arr, ref int x) 这样，调用的时 候也加上 ref 即：functionTest(arrTemp, ref y); 执行后的结果就是 0，30，100，10。

ref 关键字是强迫参数通过引用传递。

注意：在调用有 ref 参数的方法时，必须将参数要传递的参数提前初始化。但在调用 out 参数的方法时 ，就不必提前初始化。

4、out 参数

在上例中，我们稍作修改。

```
static void FunctionTest(out int x)
{
x=100;
}
static void Main()
{
int y;
FunctionTest(out y);
Console.WriteLine(y);
}
```

在 Maim()函数中调用 FunctionTest 之前，y 并没有初始化。但其输出结果确实 100；因为这样属于引用 传递，值的修改会被保留下来。

5、方法的重载

所谓重载就是指 方法名相同，而参数不同（参数类型，参数个数）看下面一个例子

```
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));
}
}
}
```

在这里例子中，有三个方法 functionTest 其参数都不一样。在调用的时候，系统会根据传递的参数自 动选择调用哪个方法的。这就是方法的重载。

在这里注意，重载的条件是，必须参数类型不同，或者参数个数不同。

C#高级（三）构造函数

构造函数是和类名相同的类的一个方法，如果没有显式的声明，在系统会在编译的时候，自动生成一 个不带参数的，不执行任何动作的构造函数。

但如果显式的声明了构造函数，系统就不会自动生成了。如果声明的构造函数是有参数的构造函数， 我们在实例化类的时候，就必须以该构造函数而实例化类。看下面的代码:

```
using System;
namespace gosoa.com.cn
{
public class test
{
public int num;
public test (int i)
{
this.num=i+5;
}
static void Main()
{
test classOne=new test(10);
int x=classOne.num;
Console.WriteLine(x);
}
}
}
```

如上代码，在实例化类的时候，test classOne=new test(10); 传递了一个参数。如果我们 test classOne=new test();这样来实例化类，就会报错了。因为我们显式的声明了一个带参的构造方法，new test() 这样实例化的时候，调用的是无参的构造函数，但类中却没有无参的构造函数。

我们再来看一下静态构造函数。

在 C# 中我们可以给类定义一个无参的静态构造函数（注意，必须是无参的），只要创建类的对象，该 方法就会执行。该函数只执行一次，并且在代码引用类之前执行。

一般，在类中有一些静态字段或者属性，需要在第一次使用类之前从外部数据源初始化这些静态字段 和属性，这时，我们就采用静态构造函数的方式来解决。

静态构造函数没有访问修饰符，其他 C#代码也不调用它，在加载类时，总是由.NET 运行库调用它。一 个类只能有一个静态构造函数。

注意，无参的实例构造函数可以和静态构造函数在类中共存。因为静态构造函数是在加载类的时候执 行的，而实例构造函数是在创建实例时执行的，两者并不冲突。

我们看下面的例子

```
using System;
namespace gosoa.com.cn
{
public class test
{
static test()
{
Console.WriteLine("www.gosoa.com.cn");
}
public test ()
{
}
static void Main()
{
test classOne=new test();
}
}
}
```

该程序运行的结果是 www.gosoa.com.cn 在类的对象创建的时候，静态构造函数已经运行了。

我们再来看一个例子

```
using System;
namespace gosoa.com.cn
{
public class test
{
private string domain;
private string url;
public test (string dom,string url)
{
this.domain=dom;
this.url=url;
}
public test(string dom)
{
this.domain=dom;
```

```
this.url="gosoa.com.cn";
}
static void Main()
{
test classOne=new test("gosoa");
Console.WriteLine(classOne.url);
}
}
}
```

在上例中，有两个构造函数，有可能两个构造函数需要初始化同一个字段，这种情况，C#中有个特殊 的语言，称为“构造函数初始化器”可以实现。看下面代码

```
using System;
namespace gosoa.com.cn
{
public class test
{
private string domain;
private string url;
public test (string dom,string url)
{
this.domain=dom;
this.url=url;
}
public test(string dom) : this (dom,"www.gosoa.com.cn")
{
}
static void Main()
{
test classOne=new test("gosoa");
Console.WriteLine(classOne.url);
}
}
}
```

如上实例，就是采用了 构造函数初始化器。注意，构造函数初始化器在构造函数之前执行。

2、只读字段(readonly)。

只读字段比常量灵活的多，常量(const)字段必须在声明之初就初始化，但 readonly 字段甚至可以进行 一些运算再确定其值。

注意，可以在构造函数中对只读字段赋值，但不能在其他地方赋值。

C＃高级（四）结构，部分类和 Object 类

一、结构

结构和类几乎一样，在定义的时候只需要把 class 写成 struct 。为结构定义函数和为类定义函数完全 一样。

那什么时候用结构呢？在一些时候，我们仅需要一个小的数据结构。如果用类的话性能是有损失的， 而采用结构就比较合适。

注意，结构是值类型，而类是引用类型；结构不支持继承；结构在编译时，编译器总会提供一个无参 的构造函数；在结构中不允许定义无参的构造函数。

二、部分类

有时候，可能多个人在访问同一个类，我们可能把同一个类，接口或者结构放在不同的文件中，只需 要在类 接口 结构前面加上 partial 关键字。

比如下面两段不同的源码分别位于不同的文件中，当编译的时候，系统会自动创建一个 testOne 类，合 并两个文件中的方法。

```
partial class testOne{
                public void MethodOne(){}
}
        partial class testOne{
                public void MethodTwo(){}
}
```

三、Object 类。

所有的.NET 类都派生于 System.Object。

既然都派生自 System.Object，这样其一些特定的方法我们就可以直接使用。比如最常简的 ToString() 方法，是获取对象的字符串表示的一种便捷方式。

C#高级（五）继承,派生类，派生类的构造方法

一、基本概念

首先我们来了解两个基本概念：实现继承和接口继承

（1）、实现继承，表示一个类派生于一个基类型，并拥有该基类型的所有成员字段和函数。

（2）、接口继承，表示一个类型只继承了函数的签名，没有任何实现的代码。在需要指定该类型具有某些可用的特性时，最好使用这种继承。

注意，在 C#中，不支持多重继承，但一个类却可以实现多个接口。同样，结构总是派生于 System.ValueType，他们还可以派生于任意多个接口。

二、实现继承。

(1)、我们先来看个例子。

```
using System;namespace gosoa.com.cn
{
    public class baseClass
        {           public string getUrl()
                    {
                    return "www.gosoa.com.cn";
                    }
        }
        public class test : baseClass
        {
            static void Main()
            {
                test classOne=new test();
                Console.WriteLine(classOne.getUrl());
            }
        }
}
```

在上面的例子中，public class test : baseClass 就是声明了类 test 继承自 baseClass。这样，在类 test 中也就具有了父类的方法，getUrl() 。在上例中输出结果是 www.gosoa.com.cn

（2）、虚方法

把一个基类中的方法声明为 virtual ，则该函数可以在任何派生类中重写了。

在 C#中，函数默认下不是虚拟的，需要显式的声明。但在 java 中，所有函数都是虚拟的。C＃中，派生类的函数重写另一个函数时，要使用 override 关键字显式的声明。如果声明了 override 函数，但在基类中如果没有可以重写的函数，编译器就会报错了。

注意，成员字段和静态函数都不能声明为 virtual ，因为这个概念只对类中的实例函数成员有意义。

我们来看个例子。

```
using System;namespace gosoa.com.cn
{
    public class baseClass
        {
            public virtual    string getUrl()
            {
                return "cnblogs.com";
             }
        }
        public class baseClass2:baseClass
        {
            public override    string getUrl()
            {
                return "gosoa.com.cn";
             }
        }
        public class test : baseClass2
        {
            public override    string getUrl()
            {
                        return "www.gosoa.com.cn";
             }
            static void Main()
            {
                test classOne=new test();
                Console.WriteLine(classOne.getUrl());
            }
        }
    }
    本例的输出结果是  www.gosoa.com.cn
```

(3)、隐藏方法。

在上例中，baseClass2 类的 getUrl()方法，如果没有 override 关键字，则 baseClass2 类的 getUrl()方法就会隐藏基类的 getUrl()方法。在编译的时候，系统会给予警告。

（4）、调用函数的基本版本

还是看上一个例子，我们稍作修改，如下，

```
public class test : baseClass2
    {
            public override    string getUrl()
            {
                    string url="http://";
                    url+=base.getUrl();
                    return url;
             }
            static void Main()
            {
                test classOne=new test();
                Console.WriteLine(classOne.getUrl());
            }
    }
```

我们来看 url+=base.getUrl();这一句，base 就是调用基类的意思，所以，本例的输出结果是

http://gosoa.com.cn

(5)、抽象类和抽象方法。

C#允许把类声明为 abstract ，抽象类不能实例化，抽象方法不没有执行代码。

我觉得抽象类和抽象方法没有什么用，一般我们用接口就可以了。搞不太明白 C#中这个抽象类和抽象方法到底想用来干什么。

（6）、密封类和密封方法。

如果把类声明为 sealed 即标明该类不可以被继承，如果是方法，则方法不可以被重写。

（7）、派生类的构造方法。

在派生类中，构造方法是依次从基类中执行，最后到派生类本身的构造函数。

我们来看下面的例子：

```
using System;namespace gosoa.com.cn
{
    public class userBase
        {
            private string username;
            public    userBase()
            {
                Console.WriteLine( "I'm good men");
             }
            public    userBase(string username)
            {
                this.username=username;
             }
        }
        public class oneMen : userBase
        {
            public    oneMen()
            {
                Console.WriteLine( "Yes , I'm very good !");
            }
            public    oneMen(string username):base(username)
            {
                Console.WriteLine( username +" is a    good    men!");
            }
            public    oneMen(string username,string hisWebSite):base(username)
           {
                Console.WriteLine( username +"'s webSite is    "+hisWebSite);
            }
            public static void Main()
            {
                oneMen classOne=new oneMen();
                //oneMen classTwo=new oneMen("pan");
                //oneMen classThree=new oneMen("pan","www.gosoa.com.cn");
            }
        }
    }
```

我们先声明了一个 userBase 类，其有个私有成员变量，还有两个构造函数。oneMen 类派生自 userBase 类。并且 oneMen 类有其自己的三个构造函数。public oneMen(string username):base(username)这个构造函数继承了基类中的构造函数，间接的给基类中的私有字段赋值了。public oneMen(string username,string hisWebSite):base(username) 这个构造函数也继承了基类的构造函数，在 Main()函数中我们依次通过三种方式实例化 oneMen 类的时候，依次输出的结果是

```
oneMen()：

I'm good men

Yes , I'm very good !

oneMen("pan")：

pan is a    good    men!

oneMen("pan","www.gosoa.com.cn")：

pan’s 's webSite is www.gosoa.com.cn
```

希望通过这个例子让大家了解到派生类的构造函数。

C#高级（六）接口,接口继承

接口我们在前面也已经有所提及。接口的命名传统上都以大写 I 开头。

我们假设这样一种情况，一个系统有很多用户，我们可以查询某个用户是否存在，并且可以修改用户 的密码。但有可能某天我们的数据库从 mysql 升级成为 sqlserver 。在这种情况下，我们看下面一个例 子。

```
using System;namespace gosoa.com.cn
{
      public interface IUserOperation

    {
            bool userExites(string username);
              bool updateUserPwd(string newPwd,string oldPwd);
    }
      public class SqlUserOperation: IUserOperation

      {
                  //这里我们假使 uname oldPwd 是通过 sql 在数据库中查到的。具体查询，这
里就不说了。

            string uname="pan";
            public string oldPwd="gosoa.com.cn";
            public bool userExites(string username)
              {
                  if(username==uname)
                  {
                        return true;

                  }
                  Else
                  {
                     return false;
                  }

            }
          public bool updateUserPwd(string newPwd,string oldPwd)
            {
                  if(newPwd==oldPwd)
                  {
                     return true;
                    }
```

```
                Else
                {
                                return false;
                 }
            }

}
    public class MainClass
     {
                static void Main(string [] args)

                 {
                 string      newPwd =Console.ReadLine();
                 string      username =Console.ReadLine();
                 SqlUserOperation one=new SqlUserOperation();
                   IUserOperation tow=new    SqlUserOperation();
            if(tow.userExites(username))
            {
                   Console.WriteLine("用户存在");
            }else
              {
                     Console.WriteLine("用户不存在");
            }
            if(tow.updateUserPwd(newPwd,one.oldPwd))
            {
                 Console.WriteLine("密码修改成功");
            }
            else
            {
                 Console.WriteLine("密码修改失败");
              }

        }
   }
      /*      //我们可能某天需要用 mysql  数据库了。这时候的具体实现又有所不同了。
public class MysqlUserOperation: IUserOperation       {       }

*/}注意，实现接口的类，必须实现类的全部成员。否则会报错喔。
```

我们来看这一句 IUserOperation tow=new SqlUserOperation(); 该句把引用变量声明为 IUserOperation 的引用方式，这表示，他们可以指向实现这个接口的任何类的实例。

同时，接口也可以彼此继承，但要注意，实现接口的类必须实现接口以及接口的父接口的所有方法。

C#高级（七）类型强制转换,拆箱,装箱

一、装箱和拆箱

装箱就是 将值类型转换为引用类型。拆箱就是 将引用类型转换为值类型。

比如我们非常常用的 .ToString() 方法，就是典型的一个装箱的过程。

再如下面的例子

```
int i=10;

object y=(object) i ; //这就是装箱

int x=(int)y;    //这是拆箱。
```

二、对象的相等比较

在 C#中，有四种比较相等的方法。

1、ReferenceEquals() 该方法是一个静态方法，用来判断两个引用是否指向同一个实例。也就是是否 指向同一个内存地址，如果是，则返回 true ，否则返回 false。

2、虚拟的 Equals() 方法。因为是虚拟的方法，所以可以重写。这样它既可以用来比较对象，也可以 比较值。

3、静态的 Equals() 方法，这与虚拟的 Equals()方法作用相同，只是带有两个参数，并对其进行比较 。这个方法可以处理两个对象中有一个是 null 的情况，当有一个是 null 的时候，就会抛出异常。

4、比较运算符 == 我们最好把这种比较看做是严格值比较和严格引用比较之间的中间选项。但注意 ，通过这样的方式比较字符串的时候，而不是引用。

C#高级（八）委托

一、委托

当我们需要把方法做为参数传递给其他方法的时候，就需要使用委托。

因为有时候，我们要操作的对象，不是针对数据进行的，而是针对某个方法进行的操作。

我们还是来以代码入手

```
using System;
namespace gosoa.com.cn
{
	public class test
	{
		public delegate string GetAString();
		public static void Main()
		{
			int x=10;
			GetAString firstString=new GetAString(x.ToString);
			Console.WriteLine(firstString());
			//上句和下面这句类似。
			//Console.WriteLine(x.ToString());
		}
	}
}
```

在上例中，public delegate string GetAString(); 就是声明了一个委托(delegate)，其语法和方法的定义类似，只是没有方法体，前面要加上关键字 delegate 。定义一个委托，基本上是定义一个新类， 所以，可以在任何定义类的地方，定义委托。

注意，在 C#中，委托总是自带一个有参数的构造函数，这就是为什么在上例中，GetAString firstString=new GetAString(x.ToString); 通过这句初始化一个新的 delegate 的时候，给传递了一个 x.ToString 方法。但，在定义 delegate 的时候，却没有定义参数。

在看另一个例子之前，我们先来了解下匿名方法。

匿名方法的使用，我们看个例子

```
using System;
namespace gosoa.com.cn
{
	public class test
```

```
    {
        delegate string GetUrl(string val);
        static void Main(string [] args)
        {
            string domin="www.gosoa.com.cn";
            GetUrl url=delegate(string    param)
            {
                param="http://"+param;
                return param;
            };
            Console.WriteLine(url(domin));
        }
    }
}
```

在本例中，GetUrl url=delegate(string param) 在这里实例化一个 delegate 的时候，采用了匿名的 方法。本例输出的结果是 http://www.gosoa.com.cn

接下来我们再看一个委托的例子。

```
using System;
namespace gosoa.com.cn
{
    class NumberOpthion
    {
        public static double numOne(double x)
        {
            return x*2;
        }
        public static double numTwo(double x)
        {
            return x*x;
        }
    }

    public class delegateTest
    {

        delegate double DoubleOpration(double x);
        static void printNumber(DoubleOpration dp,double x)
        {
            double result=dp(x);
            Console.WriteLine(
                "value is {0}, result of DoubleOpration is {1}:",x,result
```

```
                );
            }

            static void Main()
            {
                DoubleOpration doption =new DoubleOpration(NumberOpthion.numOne);
                printNumber(doption,1.5);
                doption =new DoubleOpration(NumberOpthion. numTwo);
                printNumber(doption,3.2);

            }
        }
}
```

首先我们定义了一个 NumberOpthion 类。用来对数字进行*2 和 2 次方运算。接着，我们定义了一个委托 delegate double DoubleOpration(double x)。下面，我们定义了 printNumber(DoubleOpration dp,double x) 这样一个方法，其中一个参数就是委托。最后我们 DoubleOpration doption =new DoubleOpration(NumberOpthion.numOne);实例化了一个委托，并调用了 printNumber 方法。最后的输出 结果是

Value is 0.5 result of DoubleOpration is 3;

Value is 3.2 result of DoubleOpration is 10.24;

在上例中，我们如果采用匿名方法，代码就会如下：

```
using System;
namespace gosoa.com.cn
{
        public class delegateTest
        {
            delegate double DoubleOpration(double x);
            static void printNumber(DoubleOpration dp,double x)
            {
                double result=dp(x);
                Console.WriteLine(
                    "value is {0}, result of DoubleOpration is {1}:",x,result
                );
            }
            static void Main()
            {
                DoubleOpration doptionOne =delegate(double x){return x*2;};
                DoubleOpration doptionTwo =delegate(double x){return x*x;};
                printNumber(doptionOne,1.5);
```

```
                printNumber(doptionTwo,3.2);
            }
        }
}
```

委托，还有一种情况，是多播委托。这个在以后我们应用到的时候，会学习到。

C#高级（九）C#数据结构,集合

一、集合的基本概念

在.NET 中，对于数据结构的支持，即把许多类似的对象组合起来。最简单的数据结构就是数组。

集合表示一组可以通过遍历每个元素来访问的的一组对象，特别是可以使用 foreach 循环来访问他们。 对象如果可以提供相关对象的引用，就是一个集合。称为

使用 foreach 循环是集合的主要目的，集合没有提供其他特性。

二、数组列表。

数组列表类似数组，但数组列表是可以增大的。数组在规定的大小后，就不可以再增加了，但数组列 表可以。

比如 ArrayList arrayListTest=new ArraryList(10); 该句创建了一个大小为 10 的 ArraryList 对象，当我们再为其添加第 11 项时，其容量会自动扩大 1 倍，也就变成了 20，而原来的对象会被添加上垃圾收集 器的标记。

为其添加对象的方法是 .Add()

比如：arrayListTest.Add(“gosoa.com.cn”);

我们来完整的看个例子。

```
using System;using System.Collections;namespace gosoa.com.cn
{
        class Test
        {
            static void Main()
            {
                ArrayList arrayTest = new ArrayList(4);
                arrayTest.Add("www.");
                arrayTest.Add("gosoa.");
                arrayTest.Add("com.");
               arrayTest.Add("cn");
                foreach(string item in arrayTest)
                {
                        Console.Write(item);
                }
                Console.WriteLine("\n"+arrayTest.Capacity.ToString());
```

```
                arrayTest.Add("url");
                Console.WriteLine(arrayTest.Capacity.ToString());
            }
        }
}
```