# C#基础

## 使用记事本开发步骤

一般做.NET开发，我们都是用集成开发环境VS。实际上和java一样，我们也是可以直接使用记事本等任何文本编辑工具进行C#的开发。下面简述一下如何用技术本编写C#程序。

配置环境变量

首先电脑上必须有.NET framework。安装.NET framework的时候会一起安装C#编译器。如果直接安装过VS的，也会一起安装这些东西。

打开cmd窗口，输入csc命令，确定是否安装了csc编译器。有可能，明明已经安装了，但还是有“ 不是内部或外部命令”的提示，这就需要手动把CSC编译器添加到环境变量。

Path：C:\Windows\Microsoft.NET\Framework\v4.0.30319

记事本中写代码并保存后缀为cs的文件, “.cs”是C#源文件的后缀名。

**using** System;//和java导包差不多，和java包**package**差不多，它这里叫命名空间

**namespace** ddd{ //工程名

class Program{

static void Main(string[] args) {

//如果没有上面导包操作变成：System. Console.WriteLine("你好啊");

Console.WriteLine("你好啊");

Console.ReadLine();//获取键盘的输入（后进行回车）和java的System.in一样

}

}

}

在cmd窗口中，首先cd到源文件所在的路径。如：我的源文件在G盘根目录：

C:\Users\wwl>g:

G:\>csc d.cs

就会编译成后缀exe源文件，进行双击运行

## Visual studio工具开发

Ctrl +k+c：注释 CTRL+K+U 取消注释

Ctrl +k+c：整理代码

Ctrl+j 快速智能提示。Java ：Ctrl+/

快捷写出常用的代码：如cw：Console.WriteLine();cw后按tab两次

在VS即时窗口输入：&s1和&s2查看内存地址可以看到：栈中内存地址不一样，但是堆中地址是一样的；

string s1=”曾丽方”

string s2=”曾丽方”

Console.ReadKey();

## 值类型和引用类型

区别：

1、值类型和引用类型在内存上存储的地方不一样。

2、在传递值类型和传递引用类型的时候，传递的方式不一样。

值类型我们称之为值传递，引用类型我们称之为引用传递。

我们学的值类型和引用类型：

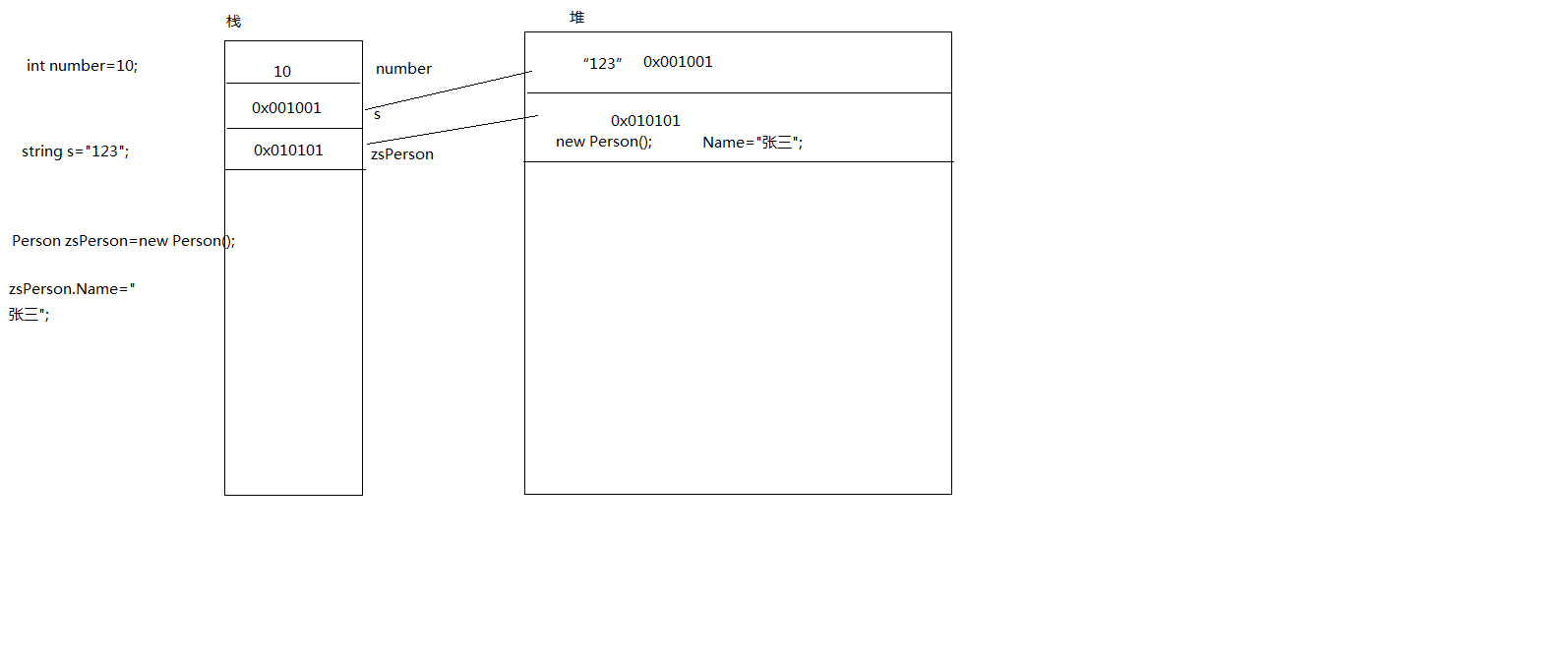
值类型：int、double、bool、char、decimal、struct、enum

引用类型：string、自定义类、数组

存储：

值类型的值是存储在内存的**栈**当中。

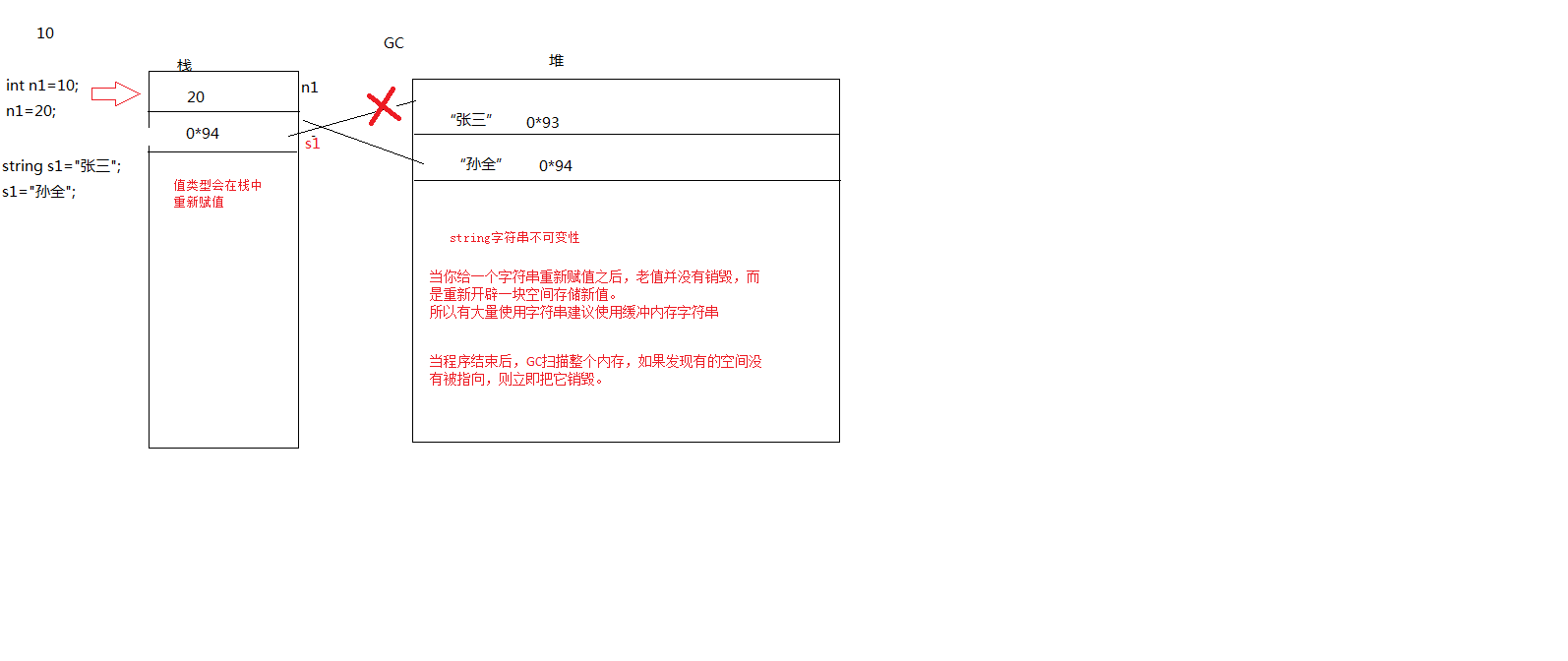
引用类型的值是存储在内存的**堆**中。



字符串的不可变性

当你给一个字符串重新赋值之后，老值并没有销毁，而是重新开辟一块空间存储新值。

当程序结束后，GC扫描整个内存，如果发现有的空间没有被指向，则立即把它销毁。

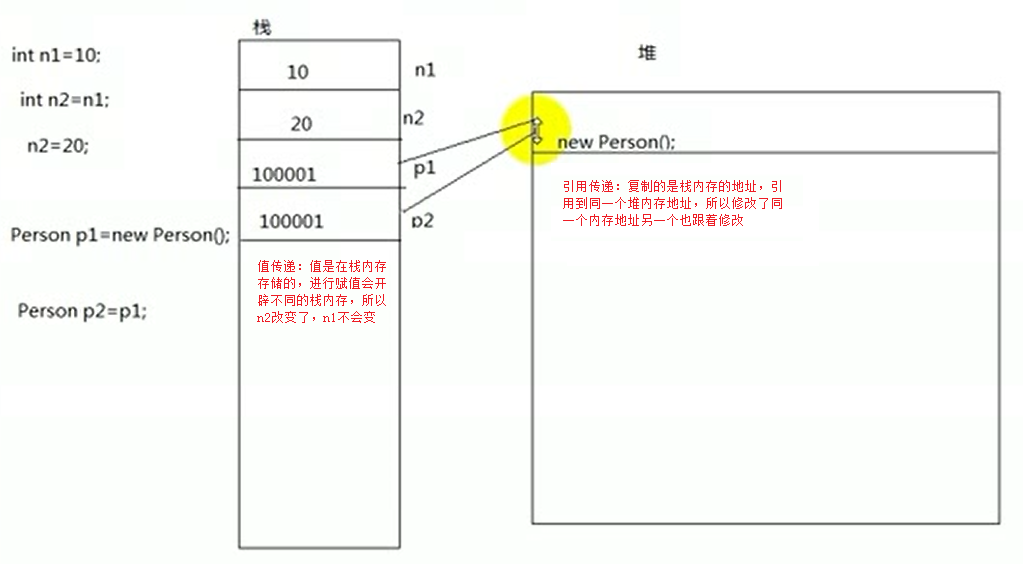


值传递和引用传递

值类型在赋值的时候，传递的是这个值得本身。

引用类型在赋值的时候，传递的是对这个对象的引用。

留意C#才有的ref和out



## 占位符

+号的作用

1)、连接：当+号两边有一边是字符串的时候，+号就起到连接的作用。

2)、相加：两边是数字的时候

占位符

使用方法：先挖个坑，再填个坑。

使用占位符需要注意的地方：

1、你挖了几个坑，就应该填几个坑，如果你多填了，没效果。

如果你少填了，抛异常。

2、输出顺序：按照挖坑的顺序输出。

int n1 = 10;

int n2 = 20;

int n3 = 30;

Console.WriteLine("占位符{1}，第二个数字是{0}，第三个数字是{2}", n1, n2,n3);

Console.WriteLine("第一个数字是:" + n1 + "，第二个数字是：" + n2 + ",第三个数字是：" + n3);

Console.ReadKey();

## 转义符

转义符指的就是一个'\'+一个特殊的字符，组成了一个具有特殊意义的字符。

\n:表示换行

\":表示一个英文半角的双引号

\t:表示一个tab键的空格

\b:表示一个退格键，放到字符串的两边没有效果。

\r\n:windows操作系统不认识\n,只认识\r\n

\\:表示一个\

@符号

1、取消\在字符串中的转义作用，使其单纯的表示为一个'\'

2、将字符串按照编辑的原格式输出

## Out参数

java没有这样的方法，形参传出值

Out：只出不进：变量值进入会先清空，ref：又进又出

如果你在一个方法中，返回多个相同类型的值的时候，可以考虑返回一个数组。

但是，如果返回多个不同类型的值的时候，返回数组就不行了，那么这个时候，

我们可以考虑使用out参数。

out参数就侧重于在一个方法中可以返回多个不同类型的值。

static void Main(string[] args) {

int c=9;

gaibian(out c);

Console.WriteLine(c);

Console.ReadKey();

}

public static void gaibian(out int c) {

Console.WriteLine(c);//这代码会报错:使用了未赋值的参数c

c = 100;

}

## ref参数

java没有这样的方法，形参传出值

能够将一个变量带入一个方法中进行改变，改变完成后，再将改变后的值带出方法。

ref参数要求在方法外必须为其赋值，而方法内可以不赋值。

static void Main(string[] args) {

int c=9;

gaibian(ref c);

Console.WriteLine(c);//打印100

Console.ReadKey();

}

public static void gaibian(ref int c) {

Console.WriteLine(c);//打印：9，

c = 100;

}

## 委托

当一个方法需要另一个方法的时候，可以把另一个方法当做参数传进来

例子：有ABC三个方法，C方法需要B、C方法，使用委托操作

namespace 基础知识测试项目{

//第一步：关键字delegate在class类上定义一个和要委托的方法签名（返回值和参数要一致）

public delegate void D(string name);

class Test{

//第二步：在方法中放入委托参数D ,方法内调用委托d(name);

public void C(string name,D d) { C方法

d(name);

}

public void A(string name) { A方法

Console.WriteLine("A 传入的参数是："+name1);

}

public void B(string name2) { B方法

Console.WriteLine("B 传入的参数是：" + name2);

}

public static void Main(string[] ags) {

Test t = new Test();

// 第三步 调用

1、委托的整体写法.：new委托对象传入需要传参的方法名t.A

D d = new D(t.A); //放入A类方法

t.C("小明",d); //调用C方法

2、委托的简写：直接调用方法C传入需要传的参数

t. C("小亮",t.B);

Console.ReadKey();

}

}

}

## 编码

string s1 = Encoding.Default.GetString(参数的byte[]);//用系统默认的编码进行解析

string s = Encoding.UTF8.GetString(参数的byte[]);//用utf8进行解码文件

## 循环

### Foreach

效率比普通的for循环高

和java增强型for循环差不多，唯一区别变量可以写var，也可以写string等

foreach (var d in s.Keys)：循环键值对操作看hashtable

**var**：和js一样，全局变量，C#也可以这样进行变量，唯一不好的就是一定要赋值

## 进程

我们可以把计算机中每一个运行的应用程序都当做是一个进程。

而一个进程又是由多个线程组成的。

需要导入命名空间System.Diagnostics.Process类

用处：打开windows应用程序和查看windows所有进程等系统

//获得当前程序中所有正在运行的进程

Process[] pros = Process.GetProcesses();

foreach (var item in pros) {

//杀掉windows所有进程

item.Kill();

//获取windows所有的进程信息

Console.WriteLine(item);

}

//通过进程打开一些应用程序

//Process.Start("calc");//打开计算器

//Process.Start("mspaint");//画图工具

//Process.Start("notepad");//记事本

//Process.Start("iexplore", "http://www.baidu.com");//通过浏览器打开百度网页

//通过一个进程打开指定的文件

ProcessStartInfo psi = new ProcessStartInfo(@"C:\Users\SpringRain\Desktop\1.exe");

//第一：创建进程对象

Process p = new Process();

p.StartInfo = psi;//打开一个实例

p.Start();

// p.star

### 线程

System.Threading.Thread 和java的差不多

多线程可以让一个程序“同时”处理多个事情。

后台运行程序，提高程序的运行效率，也不会使主界面出现无响应的情况

**产生一个线程的4步骤：**

1、 编写产生线程所要执行的方法

2、 引用System.Threading命名空间

3、实例化Thread类，并传入一个指向线程所要运行方法的委托。（这时候这个线程已经产生，但是还没有运行）

4、调用Thread实例的Start方法，标记该线程可以被CPU执行了，但具体执行时间由CPU决定

**前台线程**：只有所有的前台线程都关闭才能完成程序关闭。

**后台线程**：只要所有的前台线程结束，后台线程自动结束。，主线程也属于前台线程

Start()启动线程(告诉CPU 我可以被执行了，具体什么时候执行，由CPU决定)

Abort()终止线程 终止完成之后不能再Start()

Thread.Sleep(1) 静态方法，可以使当前线程停止一段时间运行

Name线程名

Thread.CurrentThread获得当前的线程引用

前提引用单线程的问题：单线程运行下面的例子会造成窗体假死用不了，因为：主线程都在运行逻辑上的操作了。

例子：

private void button1\_Click(object sender, EventArgs e) {

//创建一个线程去执行这个方法

th = new Thread(Test);

//标记这个线程准备就绪了，可以随时被执行。具体什么时候执行这个线程，由cpu决定

//第一个问题：将线程设置为后台线程，不设置后台线程关闭主线程（主窗体）还是在运行中

th.IsBackground = true;

th.Start();

th.Abort();//演示不能进行关了再开启，不然报错

th.Start();

}

private void Test(){

for (int i = 0; i < 10000; i++) {

//Console.WriteLine(i);

//第二个问题：这里的textBox1是主线的，要进行取消跨线程限制，不然报错

textBox1.Text = i.ToString();

}

}

private void Form1\_Load(object sender, EventArgs e) {

//一运行窗体：取消跨线程的限制访问，设置为false不让检查

Control.CheckForIllegalCrossThreadCalls = false;

}

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

//第三个问题：当你点击关闭窗体的时候，判断新线程是否为null，没有判断的话有时会因为运行不了关闭资源的操作，造成报错

if (th != null) {

//结束这个线程

th.Abort();

}

}

第四个问题：如果要传参要在Start（“123”）传参，类型必须Object类型

摇奖机例子

bool b = false;

private void button1\_Click(object sender, EventArgs e) {

if (b == false) {

b = true;

button1.Text = "停止";

Thread th = new Thread(PlayGame);

th.IsBackground = true;

th.Name = "新线程";

th.Start();

}else//b==true{

b = false;

button1.Text = "开始";

}

}

private void PlayGame(){

Random r = new Random();

while (b) {

label1.Text = r.Next(0, 10).ToString();

label2.Text = r.Next(0, 10).ToString();

label3.Text = r.Next(0, 10).ToString();

}

}

private void Form1\_Load(object sender, EventArgs e) {

Control.CheckForIllegalCrossThreadCalls = false;

}

}

## 面向对象

### 类

语法：

[public] class 类名:不写修饰符默认是Internal

{

字段;

属性;

方法;

}

**实例化：**写好了一个类之后，我们需要创建这个类的对象，

那么，我们管创建这个类的对象过程称之为类的实例化。使用关键字 new.

**初始化**：当我们创建好一个类的对象后，需要给这个对象的每个属性去赋值。

我们管这个过程称之为对象的初始化。

**this**:表示当前这个类的对象。

类是不占内存的，而对象是占内存的。

命名空间（java包package）

等于是java进行导包，c#没有默认的：像java的lang类不用导包

导包(命名空间)方式：

自动导包：

1）、用鼠标去点

2）、快捷键：alt+shift+F10

静态和非静态的区别：（java没有静态类就是不能再class中static，只有内部静态类）

1)、在非静态类中：既可以有实例成员，也可以有静态成员。

2)、在调用实例成员的时候，需要使用对象名.实例成员;

在调用静态成员的时候，需要使用类名.静态成员名;

总结：静态成员必须使用类名去调用，而实例成员使用对象名调用。

静态函数中，只能访问静态成员，不允许访问实例成员。

实例函数中，既可以使用静态成员，也可以使用实例成员。

静态类中只允许有静态成员，不允许出现实例成员。

使用：

1)、如果你想要你的类当做一个"工具类"去使用，这个时候可以考虑将类写成静态的。

2)、静态类在整个项目中资源共享。

只有在程序全部结束之后，静态类才会释放资源。

堆 栈 静态存储区域（java说法是方法区）

Field字段

Method方法

Property属性

属性

属性的作用就是保护字段、对字段的赋值和取值进行限定。

属性的本质就是两个方法，一个叫get()一个叫set()。

既有get()也有set()我们诚之为可读可写属性。

只有get()没有set()我们称之为只读属性

没有get()只有set()我们称之为只写属性

**自动属性写法**

Public int Age{

get；

set；

}

**自动属性写法相当于下面写法成员变量设置set、get:编译后会自动增加成员变量**

Private int age;

Public string Age{

get{return age}

set{age=value}

不好的地方：不可以进行再set种进行限制判断

访问修饰符

public：公开的公共的，在哪都能访问。

private：私有的，只能在当前类的内部进行访问，出了这个类就访问不到了。

Protected：受保护的，可以在当前类的内部以及该类的子类中访问

Internal：只能在当前程序集中访问，在同一项目中，internal和public的权限一样的

Proteted internal：

**特别注意：在C#中如果类内部成员（方法、属性、字段等）不写任何修饰符则默认为private；  
如果类（class）的访问修饰符什么也不写默认是internal。**

**1)、能够修饰类的访问修饰符只有两个：public、internal。**

**2)、可访问性不一致。**

**子类的访问权限不能高于父类的访问权限，会暴漏父类的成员。（经测试和java刚好相反）**

释放资源。**GC Garbage Collection垃圾回收器**

构造函数

作用：帮助我们初始化对象(给对象的每个属性依次的赋值)

构造函数是一个特殊的方法：

1)、构造函数没有返回值，连void也不能写。

2)、构造函数的名称必须跟类名一样。

创建对象的时候会执行构造函数

构造函数是可以有重载的。

\*\*\*

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

析构函数

**当程序结束的时候 析构函数才执行：**帮助我们释放资源，GC Garbage Collection （垃圾回器）有时候没有给我释放资源，我又希望立即释放，就可以使用此函数了

写法：

~User(){ //注意有一个 ~

Console.ReadKey();

}

new关键字

Person zsPerson=new Person();

new帮助我们做了3件事儿：

1)、在内存中开辟一块空间

2)、在开辟的空间中创建对象

3)、调用对象的构造函数进行初始化对象

this关键字

1)、代表当前类的对象

2)、在类当中显示的调用本类的构造函数 :this

#### 部分类

就是在一个工程中可以有两个相同的名称类：关键字partial

Public partial class A{}

它们本质上就是一个类，私有的可以访问

#### 密封类

不能被继承的类，可以继承别的类 关键字：sealed

Public sealed class P{}

#### 枚举类型

enum：关键字，声明枚举的关键字

枚举名：要符合Pascal命名规范

将枚举声明到命名空间的下面，类的外面，表示这个命名空间下，所有的类都可以使用这个枚举。

枚举就是一个变量类型 ，int--double string decimal.

只是枚举声明、赋值、使用的方式跟那些普通的变量类型不一样。

我们可以将一个枚举类型的变量跟int类型和string类型互相转换。

枚举类型默认是跟int类型相互兼容的，所以可以通过强制类型转换的语法互相转换。

当转换一个枚举中没有的值的时候，不会抛异常，而是直接将数字显示出来。

枚举同样也可以跟string类型互相转换，如果将枚举类型转换成string类型，则直接调用ToString().

如果将字符串转换成枚举类型则需要下面这样一行代码：

(要转换的枚举类型)Enum.Parse(typeof(要转换的枚举类型),"要转换的字符串");

如果转换的字符串是数字，则就算枚举中没有，也会不会抛异常。

如果转换的字符串是文本，如果枚举中没有，则会抛出异常。

namespace \_02枚举{

[public] enum 枚举名{

值1,

值2,

值3,

........

}

//声明了一个枚举 Gender

public enum Gender{

男,

女

}

class Program{

static void Main(string[] args) {

//变量类型 变量名=值;

int n = 10;

Gender gender = Gender.男;

//性别

//char gender = '男';

//string s1 = "female";

//string ss1 = "爷们";

}

}

}

#### 接口

[public] interface I..able

{

成员;

}

实现接口跟继承一样都是冒号：

接口中的成员不能加“访问修饰符”，接口中的成员访问修饰符为public,不能修改。

（默认为public）接口中的成员不能有任何实现（“光说不做”，只是定义了一组未实现的成员）。接口中只能有方法、属性、索引器、事件，不能有“字段”和构造函数。

接口与接口之间可以继承，并且可以多继承。

接口并不能去继承一个类，而类可以继承接口 （接口只能继承于接口，而类既可以继承接口，也可以继承类）实现接口的子类必须实现该接口的全部成员。一个类可以同时继承一个类并实现多个接口，如果一个子类同时继承了父类A，并实现了接口IA,那么语法上A必须写在IA的前面。

class MyClass:A,IA，因为类是单继承的。

显示实现接口的目的：解决方法的重名问题

什么时候显示的去实现接口：

当继承的方法接口中的方法和参数一摸一样的时候，要是用显示的实现接口

namespace \_14\_显示实现接口{

class Program{

static void Main(string[] args) {

//显示实现接口就是为了解决方法的重名问题

IFlyable fly = new Bird();

fly.Fly();

Bird bird = new Bird();

bird.Fly();//创建自己对象调用独有的方法Fly

Console.ReadKey();

}

}

public class Bird : IFlyable{

public void Fly(){

Console.WriteLine("我是本类的Fly方法");

}

/// 显示实现接口：跟本类的Fly方法同名

void IFlyable.Fly(){

Console.WriteLine("我是实现接口的Fly方法");

}

}

public interface IFlyable{

void Fly();

}

}

当一个抽象类实现接口的时候，需要子类去实现接口。

### 继承

和java不同点

new关键字：（java只有创建对象作用）

1)、创建对象

2)、隐藏从父类那里继承过来的同名成员。

隐藏的后果就是子类调用不到父类的成员。

public void new A（）{} 父类的成员方法和子类方法一致这样写不出现警告

问题：子类有没有继承父类的构造函数？

答：子类并没有继承父类的构造函数，但是子类会默认的调用父类无参数的构造函数来创建父类对象，让子类可以使用父类中的成员。

在子类中显示的调用父类的构造函数，使用关键字:base()（java中的super）

构造方法中调用方法：和java不一样

调用父类无参数的操作

public A():base()//和java不一样

{

}

纠结很久的例子：

**继承的特性**

1、继承的单根性：一个子类只能有一个父类。

2、继承的传递性：爷爷的也一样可以继承

查看类图

右键工程名---🡪视图---🡪查看类图可以看继承关系

//子类

class A:Extends{

public A():base(){

}

public void Teacher(){

Console.WriteLine("{0}在教学生跳舞",base.UserName);

//父类

class Extends{

string userName;

int age;

string gender;

public string UserName{

get { return userName; }

set { userName = value; }

}

public int Age{

get { return age; }

set { age = value; }

}

public string Gender{

get { return gender; }

set { gender = value; }

}

调用

class Program{

static void Main(string[] args) {

Extends t = new Extends();

t.UserName="曾丽方";通过创建父类设置值，子类创建对象调用方法获取的值是null（java也一样.super去取值也没有，所以取值必须通过构造函数来取值）

A a = new A();

a.UserName = "曾丽方懒猪头";

a.Teacher();

Console.ReadKey();

}

}

总结：通过父类new的对象来进行赋值，是没有用的，因为：子类创建对象：隐式的对父类也进行new对象，此时的父类不是自己之前new的父类对象

### 多态

不同点：重写父类方法需要加关键字override，有同名的需用new隐藏父类同名方法

抽象类通过多态接收子类对象，获取实例化就可以调本类不是抽象的方法了

1)重写:C#用override关键字重写父类同名且被virtual或abstract关键字修饰的方法。而JAVA则不必。

2)隐藏:C#可以用new关键字隐藏父类的同名方法，而JAVA则不存在这个概念。

概念:让一个对象能够表现出多种的状态(类型)

不同点：多了一个虚方法：使用关键字 virtual、override

实现多态的3种手段：1、虚方法 2、抽象类 3、接口

1)、虚方法

步骤：

将父类的方法标记为虚方法 ，使用关键字 virtual，这个函数可以被子类重新写一个遍。

重写：需要在父类添加virtual，子类方法中需要添加override，（java父类方法和子类方法同名直接重写）

2)、抽象类

当父类中的方法不知道如何去实现的时候，可以考虑将父类写成抽象类，将方法写成抽象方法。

例子：

## 可变参数与自动装箱和拆箱

将实参列表中跟可变参数数组类型一致的元素都当做数组的元素去处理。

params可变参数必须是形参列表中的最后一个元素。

和java不同：它是关键字params 加上 数组，其它都相同

namespace ConsoleSuanFa {

class Person {

public void wantEat(params String[] wants) {

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

Console.WriteLine(wants[i]);

}

}

public static void Main(String[] args) {

Person p = new Person();

p.wantEat("香蕉","扒拉","苹果","朗爬");

p.wantEat("吃个鸟,不让你吃");

2、装箱、拆箱

装箱：就是将值类型转换为引用类型。

拆箱：将引用类型转换为值类型。

看两种类型是否发生了装箱或者拆箱，要看，这两种类型是否存在继承关系。

## 里氏转换

1)、子类可以赋值给父类：如果有一个地方需要一个父类作为参数，我们可以给一个子类代替

B b = new B();//子类

A a = b;

例子：

string s=string.Join("|",new string[]{"1","2"});//第2个参数属于object子类，

Console.WriteLine(s);//打印 1 | 2

2）、如果父类中装的是子类对象，那么可以将这个父类强转为子类对象

转换判断：

Is ：表示类型转换，如果能够转换成功，则返回一个true，否则false

as：表示类型转换，如果能够转换则返回对应的对象，否则返回一个null；

A a = new B();

if(a is C){

B b = (B)a;

b.Bb();

Console.WriteLine("成功");

}

else

{

Console.WriteLine("失败");

}

B b2 = a as B;

if (b2 != null)

{

b2.Bb();

}

Console.ReadKey();

}

3）创建一个父类数组，可以把子类放到这个父类数组里

（多态为什么不能调用子类的呢，因为变量是父类，它是父类对象，只有自己的方法，进行强制类型转换后用子类对象接收就可以接收了。）

## 序列化和反序列化

using System.Runtime.Serialization.Formatters.Binary;

用来把实体类对象，保存到硬盘

namespace \_05序列化和反序列化

{

class Program

{

static void Main(string[] args)

{

//要将p这个对象 传输给对方电脑

//Person p = new Person();

//p.Name = "张三";

//p.Age = 19;

//p.Gender = '男';

//using (FileStream fsWrite = new FileStream(@"C:\Users\ 111.txt", FileMode.OpenOrCreate, FileAccess.Write))

//{

// //开始序列化对象

// BinaryFormatter bf = new BinaryFormatter();

// bf.Serialize(fsWrite, p);

//}

//Console.WriteLine("序列化成功");

//Console.ReadKey();

//接收对方发送过来的二进制 反序列化成对象

Person p;

using (FileStream fsRead = new FileStream(@"C:\Users\SpringRain\Desktop\111.txt", FileMode.OpenOrCreate, FileAccess.Read))

{

//序列化对象

BinaryFormatter bf = new BinaryFormatter();

p = (Person)bf.Deserialize(fsRead);

}

Console.WriteLine(p.Name);

Console.WriteLine(p.Age);

Console.WriteLine(p.Gender);

Console.ReadKey();

}

}

[Serializable] 它是这样写的，也是这个类

public class Person

{

private string \_name;

public string Name{

get { return \_name; }

set { \_name = value; }

}

private char \_gender;

public char Gender {

get { return \_gender; }

set { \_gender = value; }

}

private int \_age;

public int Age{

get { return \_age; }

set { \_age = value; }

}

}

}

# C#.net常用类

## Object

支持 .NET Framework 类层次结构中的所有类，并为派生类提供低级别服务。 这是 .NET Framework 中所有类的最终基类；它是类型层次结构的根。

命名空间: System

程序集: mscorlib（位于 mscorlib.dll）

继承层次结构

System.Object 所有类、结构、枚举和委托。

构造函数

**Object()**初始化 Object 类的新实例。

方法

**1)Equals(Object)**确定指定的对象是否等于当前对象。

**2)Equals(Object, Object)** 确定指定的对象实例是否被视为相等。

**3)Finalize()** 在垃圾回收将某一对象回收前允许该对象尝试释放资源并执行其他清理操作。

**4)GetHashCode()** 作为默认哈希函数。

**5)GetType()**获取当前实例的 Type。

**6)MemberwiseClone()**创建当前 Object 的浅表副本。

**7)ReferenceEquals(Object, Object)**确定指定的 Object 实例是否是相同的实例。

**8)ToString()**返回表示当前对象的字符串。

## String

命名空间: System

程序集: mscorlib（位于 mscorlib.dll）

继承层次结构

System.Object

  System.String

构造函数

**String(Char\*)**将 String 类的新实例初始化为由指向 Unicode 字符数组的指定指针指示的值。

**String(Char\*, Int32, Int32)**将 String 类的新实例初始化为由指向 Unicode 字符数组的指定指针指示的值、该数组内的起始字符位置和一个长度指示的值。

**String(Char, Int32)**将 String 类的新实例初始化为由重复指定次数的指定 Unicode 字符指示的值。

**String(Char[])** 将 String 类的新实例初始化为由 Unicode 字符数组指示的值。

**String(Char[], Int32, Int32)**将 String 类的新实例初始化为由 Unicode 字符数组、该数组内的起始字符位置和一个长度指示的值。

**String(SByte\*)**将 String 类的新实例初始化为由指向 8 位有符号整数数组的指针指示的值。

**String(SByte\*, Int32, Int32)**将 String 类的新实例初始化为由指向 8 位有符号整数数组的指定指针、该数组内的起始位置和一个长度指示的值。

**String(SByte\*, Int32, Int32, Encoding)**将 String 的新实例初始化为由指向 8 位有符号整数数组的指定指针、该数组内的起始位置、长度以及 Encoding 对象指示的值。

属性

**Chars[Int32]** 获取当前 Char 对象中位于指定位置的 String 对象。

**Length** 获取当前 String 对象中的字符数。

方法

2)、我们可以讲字符串看做是char类型的一个只读数组。

例子：

//可以讲string类型 看做是char类型的一个只读数组

string s = "abcdefg";

// s[0] = 'b';不能这样做 因为是只读的

//首先将字符串转换为char类型的数组

char[] chs = s.ToCharArray();

chs[0] = 'b';

//将字符数组转换为我们的字符串

s = new string(chs);

//既然可以将string看做char类型的只读数组，所以我可以通过下标去访问字符串中的某一个元素

Console.WriteLine(s[0]);//这种写法，不知道java可不可以

Console.WriteLine(s);

Console.ReadKey();

//当然最简单的就是直接赋值，改变了，

s = "bbcdefg";

ToCharArray(); 将字符串转换为char数组

new string(char[] chs): 能够将char数组转换为字符串

老师还对比了StringBUilder缓冲字符串，

Stopwatch sw=new Stopwatch（）；//创建了一个计数器，用来记录程序运行的时间

Sw.start//开始计时，sw.stop();结束计时

为什么慢那么多呢：字符串创建100000个，要不断的开辟空间来放入值，而springBuilder不用开辟空间，append(i)是直接添加在同一个内存地址加数据

结论：使用大量字符串操作，直接用此缓冲字符串可以，它们方法都一样的

4、字符串提供的各种方法

1)、Length：获得当前字符串中字符的个数

2)、ToUpper():将字符转换成大写形式

3)、ToLower():将字符串转换成小写形式

4)、Equals(lessonTwo,StringComparison.OrdinalIgnoreCase):比较两个字符串，可以忽略大小写

5)、Split()：分割字符串，返回字符串类型的数组。

6)、Substring()：解决字符串。在截取的时候包含要截取的那个位置。

7)、IndexOf():判断某个字符串在字符串中第一次出现的位置，如果没有返回-1、值类型和引用类型在内存上存储的地方不一样。

8)、LastIndexOf()：判断某个字符串在字符串中最后一次出现的位置，如果没有同样返回-1

9)、StartsWith():判断以....开始

10)、EndsWith():判断以...结束

11)、Replace():将字符串中某个字符串替换成一个新的字符串

12)、Contains():判断某个字符串是否包含指定的字符串

13)、Trim():去掉字符串中前后的空格

14)、TrimEnd()：去掉字符串中结尾的空格

15)、TrimStart()：去掉字符串中前面的空格

16)、string.IsNullOrEmpty():判断一个字符串是否为空或者为null

17)、string.Join()：将数组按照指定的字符串连接，返回一个字符串。

## 集合

集合：很多数据的一个集合

数组：长度不可变、类型单一

集合的好处：长度可以任意改变 类型随便

### ArrayList

命名空间（java包package）

using System.Collections;

集合的长度问题：

每次集合中实际包含的元素个数(count)超过了可以包含的元素的个数(capcity)的时候，

集合就会向内存中申请多开辟一倍的空间，来保证集合的长度一直够用。

如：刚开始添加了1一个元素，开辟4个，超过4个开辟8个，超过8个开辟16个

常用方法

**ArrayList list = new ArrayList();**

**添加：**添加单个元素

list.Add(true);

list.Add(1);

list.Add("张三");

添加集合元素

list.AddRange(new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 });

list.AddRange(list);//也可以添加自身

//list.Insert(1, "插入的");在指定的位置插入一个元素

**删除：**

//list.Clear();清空所有元素

//list.Remove(true);删除单个元素 写谁就删谁

//list.RemoveAt(0);根据下标去删除元素

//list.RemoveRange(0, 3);根据下标去移除一定范围的元素

**其它**

// list.Sort();//升序排列：需要自然元素

//list.Reverse();反转

//list.InsertRange(0, new string[] { "张三", "李四" });在指定的位置插入一个集合

**判断**

//bool b = list.Contains(1);判断是否包含某个指定的元素

list.Add("颜世伟");

if (!list.Contains("颜世伟")){

list.Add("颜世伟");

}else{

Console.WriteLine("已经有这个屌丝啦");

}

for (int i = 0; i < list.Count; i++){

Console.WriteLine(list[i]);

}

Console.ReadKey();

### HashTable

键：不能重复，值：可以重复

它就好比是java没有加泛型的map

常用方法

Hashtable s = new Hashtable();

**增加**

s.Add(1, "张三");

s.Add(6, true);

s.Add("ni","你好");

s.Add(false,"错误的");

s["这样也可以增加"] = "新来的";

**修改,强制更改**

s["ni"] = "把键为ni，修改掉";

**判断**

if (s.ContainsKey("abc"))//判断是否有此键

{

}

**删除**

s.Clear();//移除所有的值

s.Remove("ni");//删除键为ni的值

**查看**

Console.WriteLine(s[1]);//获取key的值为1的值

Console.WriteLine(s["ni"]);//获取key的值为ni的值

Console.WriteLine(s[false]);//打印值：错误的

//用foreach循环遍历

foreach (var d in s.Keys)//var可以说是全局变量，和js一样，使用此变量必须赋值，

{

// foreach (var c in s) Console.WriteLine(c);打印：命名空间：System.Collections.DictionaryEntry

Console.WriteLine("键："+d+"值："+s[d]);

老师讲了一个简体转繁体转换例子

把简体字设为：键，把繁体字设为：值

private const String Jian = "啊阿埃挨哎唉哀皑癌蔼矮艾碍爱隘鞍氨安俺按”

private const String Fan = "啊阿埃挨哎唉哀皚癌藹矮艾礙愛隘鞍氨咹俺按”

static void Main(string[] args)

{

Hashtable ht = new Hashtable();

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

{

ht.Add(Jian[i], Fan[i]);

}

Console.WriteLine("请随便输入");

string input = Console.ReadLine();//张三abc李四

//遍历用户输入的字符串 将每一个字符都转换成繁体字

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

{

if (ht.ContainsKey(input[i]))

{

Console.Write(ht[input[i]]);

}

else

{

Console.Write(input[i]);

}

}

Console.ReadKey();

//将用户输入的简体字 转换成繁体字

}

### 泛型集合

好坑爹：还有泛型集合类 重点

需要导入命名空间using System.Collections.Generic;

#### List<>

创建泛型集合对象

List<string> list = new List<string>();

和ArrayList一样的方法

不一样的

List泛型集合可以转换为数组

ToArray()

#### Dictionary<>

重要：对应Hashtable（也就是java的泛型map）

遍历可以使用下列方式了

Dictionary<int, string> dic = new Dictionary<int, string>();

dic.Add(1, "张三");

dic.Add(2, "李四");

dic.Add(3, "王五");

dic[1] = "新来的";

//可以变成这样循环输出键值两个值了

foreach (KeyValuePair<int,string> kv in dic)

{

Console.WriteLine("{0}---{1}",kv.Key,kv.Value);

}

## IO流

需要导入using System.IO;

### 文件夹操作

#### （路径）Path

它是静态类（java没有此类）

string str=@"D:\n\lv\t.txt";//@是让\不要转义

//获取文件名

Console.WriteLine(Path.GetFileName(str));//打印t.txt,原理就是spring方法配合封装

//获取文件名不包含扩展名：txt

Console.WriteLine(Path.GetFileNameWithoutExtension(str));//打印t

//获取扩展名：txt

Console.WriteLine(Path.GetExtension(str));//打印.txt

//获取文件所在的路径：也就是没有文件名

Console.WriteLine(Path.GetDirectoryName(str));//打印：D:\n\lv\

//获取全路径

Console.WriteLine(Path.GetFullPath(str));//打印：D:\n\lv\t.txt

//连接字符串作为路径

Console.WriteLine(Path.Combine(@"c:\a\", "b.txt"));//打印：c:\a\b.txt

Console.WriteLine(Path.GetFullPath(str));

Console.ReadKey();

#### （文件夹）Directory

Directory静态类

创建文件夹

Directory.CreateDirectory(@"C:\a");

Console.WriteLine("创建成功");

Console.ReadKey();

删除文件夹

参数加true表示非空也删除

Directory.Delete(@"C:\a",true);

Console.WriteLine("删除成功");

Console.ReadKey();

剪切文件夹

Directory.Move(@"c:\a", @"C:\Users\SpringRain\Desktop\new");

Console.WriteLine("剪切成功");

Console.ReadKey();

获得指定文件夹下所有文件的全路径

string[] path = Directory.GetFiles(@"C:\Users\SpringRain\Desktop\Picture","\*.jpg");

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

Console.WriteLine(path[i]);

}

Console.ReadKey();

获得所有文件夹下文件的全路径

string[] path = Directory.GetDirectories(@"C:\Users\SpringRain\Desktop\new");

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

Console.WriteLine(path[i]);

}

Console.ReadKey();

判断指定的文件夹是否存在

if (Directory.Exists(@"C:\a\b")){

for (int i = 0; i < 100; i++){

Directory.CreateDirectory(@"C:\a\b\" + i);

}

}

Console.WriteLine("OK");

Console.ReadKey();

#### （文件）File

它是一个静态类，全是静态方法，java要new

创建、拷贝、复制、删除

创建一个文件 File.Create(@"C:\Users\SpringRain\Desktop\new.txt");

File.Create(@"C:\Users\SpringRain\Desktop\new.txt");

Console.WriteLine("创建成功");

Console.ReadKey();

删除一个文件 File.Delete //此删除不是到回收站

File.Delete(@"C:\Users\SpringRain\Desktop\new.txt");

Console.WriteLine("删除成功");

Console.ReadKey();

复制一个文件 File.Copy

//File.Copy(@"C:\Users \code.txt", @"C:\Users\SpringRain\Desktop\new.txt");

//Console.WriteLine("复制成功");

//Console.ReadKey();

剪切

File.Move(@"C:\Users \code.txt", @"C:\Users\ newnew.txt");

Console.WriteLine("剪切成功");

Console.ReadKey();

##### 读写操作

不是重点可以不看

**读：只能读小文件:** **File.ReadAllBytes(@"d:\n1.txt");**

读取字节操作

//读取文件获取字节数组

byte[] buffer = File.ReadAllBytes(@"d:\n1.txt");

////将字节数组中的每一个元素都要按照我们指定的编码格式解码成字符串

////UTF-8 GB2312 GBK ASCII Unicode

string s1 = Encoding.Default.GetString(buffer);//用系统默认的编码进行解析

string s = Encoding.UTF8.GetString(buffer);//用utf8进行解码文件

Console.WriteLine(s);

Console.ReadKey();

读取字符操作

1)一行一行读取文件，返回string数组

//string[] contents = File.ReadAllLines(@"C:\Users\SpringRain\Desktop\抽象类特点.txt", Encoding.Default);

//foreach (string item in contents)

//{

// Console.WriteLine(item);

//}

//Console.ReadKey();

2)读取全部，返回string

//string str = File.ReadAllText("抽象类特点.txt", Encoding.Default);

//Console.WriteLine(str);

//Console.ReadKey();

写

字节

没有这个文件的话会给你创建一个,有的话 会给你覆盖掉

string str="今天天气好晴朗处处好风光";

需要将字符串转换成字节数组

byte[] buffer= Encoding.Default.GetBytes(str);

File.WriteAllBytes(@"C:\Users\SpringRain\Desktop\new.txt", buffer);

Console.WriteLine("写入成功");

Console.ReadKey();

字符

一行一行写入，

File.WriteAllLines(@"C:\Users\ new.txt", new string[] { "aoe", "ewu" });

Console.WriteLine("OK");

Console.ReadKey();

一次性写入

File.WriteAllText(@"C:\Users\ \new.txt", "张三李四王五赵六");

Console.WriteLine("OK");

Console.ReadKey();

总结：上面两个字符方法会覆盖内容，Append追加内容不是清空后再加

File.AppendAllText(@"C:\Users \new.txt", "看我有木有把你覆盖掉");

Console.WriteLine("OK");

Console.ReadKey();

File.AppendAllLines(path，spring[]);

File.AppendText(文件path);//把目标文件编码改为utf-8

### 三个重要流

#### FileStream

重要必须掌握 字节流

读 、写字节操作都是同一个类FileStream

使用FileStream来读取数据,参数：

Path：读取的路径

FileMode静态类：(针对这个文件要做什么操作)，

FileAccess静态类：(操作文本内的数据做什么操作)

读的操作 FileMode.OpenOrCreate FileAccess.Read

FileStream fsRead = new FileStream(@"C:\Users\SpringRain\Desktop\new.txt",

FileMode.OpenOrCreate, FileAccess.Read);

//设置最多读5M

byte[] buffer = new byte[1024 \* 1024 \* 5];

//3.8M 5M

//返回本次实际读取到的有效字节数

int r = fsRead.Read(buffer, 0, buffer.Length);

//将字节数组中每一个元素按照指定的编码格式解码成字符串

string s = Encoding.UTF8.GetString(buffer, 0, r);

//关闭流

fsRead.Close();

//释放流所占用的资源

fsRead.Dispose();

Console.WriteLine(s);

Console.ReadKey();

写的操作：

//using写在里面帮我们释放占用的资源，就不用关闭流和释放流操作了

//using (FileStream fsWrite = new FileStream(@"C:\Users\SpringRain\Desktop\new.txt", FileMode.OpenOrCreate, FileAccess.Write)) {

string str = "看我游牧把你覆盖掉";//只是覆盖这些字节的，不是清空完内容再写入

byte[] buffer = Encoding.UTF8.GetBytes(str);

fsWrite.Write(buffer, 0, buffer.Length);

}

Console.WriteLine("写入OK");

Console.ReadKey();

例子：大文件复制,老师的麻烦，还是java中的好用

和java差不多的：路径的分隔符随便位置都可以的，2、没有操作关闭流操作（和java一样）

FileStream f = new FileStream(@"C:\ \JAVAJDKAPI17.zip", FileMode.OpenOrCreate, FileAccess.Read);

FileStream w = new FileStream("e://Windchill-master.zip", FileMode.OpenOrCreate, FileAccess.Write);

byte[] b = new byte[1024\*1024\*10];

int i = 0;

while ((i=f.Read(b, 0, b.Length)) != 0) {

w.Write(b,0,i);

}

老师的超级麻烦都不想看了

namespace \_07使用文件流来实现多媒体文件的复制{

class Program{

static void Main(string[] args) {

//思路：就是先将要复制的多媒体文件读取出来，然后再写入到你指定的位置

string source = @"C:\Users\SpringRain\Desktop\1、复习.wmv";

string target = @"C:\Users\SpringRain\Desktop\new.wmv";

CopyFile(source, target);

Console.WriteLine("复制成功");

Console.ReadKey();

}

public static void CopyFile(string soucre, string target){

//1、我们创建一个负责读取的流

using (FileStream fsRead = new FileStream(soucre, FileMode.Open, FileAccess.Read)) {

//2、创建一个负责写入的流

using (FileStream fsWrite = new FileStream(target, FileMode.OpenOrCreate, FileAccess.Write)) {

byte[] buffer = new byte[1024 \* 1024 \* 5];

//因为文件可能会比较大，所以我们在读取的时候 应该通过一个循环去读取

while (true) {

//返回本次读取实际读取到的字节数

int r = fsRead.Read(buffer, 0, buffer.Length);

//如果返回一个0，也就意味什么都没有读取到，读取完了

if (r == 0) {

break;

}

fsWrite.Write(buffer, 0, r);

}

#### StreamReader

主要作用：读取文本文件

使用StreamReader来读取一个文本文件

using (StreamReader sr = new StreamReader(@"C:\Users\抽象类特点.txt",Encoding.Default)) {

while (!sr.EndOfStream) {

Console.WriteLine(sr.ReadLine());

}

}

注意using写法：就不用进行关闭流的操作了

#### StreamWriter

使用StreamWriter来写入一个文本文件

//加true不覆盖内容

using (StreamWriter sw = new StreamWriter(@"C:\Users \newnew.txt",true)) {

sw.Write("看我有木有把你覆盖掉");

}

Console.WriteLine("OK");

Console.ReadKey();+

## Socket

### Socket服务端

private void button1\_Click(object sender, EventArgs e) {

//集合保存ip和客户端连接的socket

Dictionary<string, Socket> sendMap = new Dictionary<string, Socket>();

try{

//当点击开始监听的时候 在服务器端创建一个负责监听IP地址跟端口号的Socket

Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,

ProtocolType.Tcp);

IPAddress ip = IPAddress.Any;//获得随机的监听IP

//创建端口号对象，参数：监听ip地址 ，端口（从textBox获取字符串进行转换）

IPEndPoint point = new IPEndPoint(ip, Convert.ToInt32(textPort.Text));

//监听

socket.Bind(point);//绑定监听端口

ShowMsg("监听成功");

socket.Listen(10);//表示时间点内只能进10人

Thread t = new Thread(Connection);

t.IsBackground = true;

t.Start(socket);

} catch{}

}

/// 接收客户端窗体进行设置

private void ShowMsg(string p) {

//追加，不要覆盖，让其换行

textShow.AppendText(p + "\r\n");

}

void Connection(Object o){

try{

//第一个问题：要把连接写到另一个线程：因为一但点击开始监听：窗体又成假死状态

Socket socket = o as Socket;

Socket sockeSend;

//第二个问题：如果不设置为循环会出问题的，只能连接一个人，再进行连接会报错

while(true){

//等待客户端的连接

sockeSend = socket.Accept();

//把连接的客户端IP和socket对象存储到数组

sendMap.Add(sockeSend.RemoteEndPoint.ToString(),sockeSend);

//把连接的客户端Ip和socket对象存储到下拉框

comboBox1.Items.Add(sockeSend.RemoteEndPoint.ToString());

//获取对方的ip：

ShowMsg(sockeSend.RemoteEndPoint.ToString() + ":" + "连接成功");

//第三个问题：这里还有不断接受客户端传过来的信息一个循环要写在另一个线程，不写到另一个线程：窗体又要成 假死状态，因为是死循环，

Thread t = new Thread(ConnetionMessiag);

t.IsBackground = true;

t.Start(sockeSend);

}

}

catch{ }

}

/// 主窗体一加载进行窗体跨线程限制取消

private void Form1\_Load(object sender, EventArgs e) {

// 第四个问题:必须把窗体跨线程限制取消掉

Control.CheckForIllegalCrossThreadCalls = false;

}

//设置成变量属性让发送按钮能接收：用来发送消息至客户端

Socket sockeSend;

/// 服务器不停的接收客户端发过来的信息

void ConnetionMessiag(Object o) {

try{

sockeSend = o as Socket;

while(true){

byte[] b = new byte[1024 \* 1024 \* 3];

int i = sockeSend.Receive(b);//为0证明远程客户端已经关掉了，你发空格也是有一个字节的

string str = Encoding.UTF8.GetString(b, 0, i);

//第五个问题：客户端中一旦关掉，就会出现循环发送空信息，进行判断

if (i==0) {

break;

}

ShowMsg(sockeSend.RemoteEndPoint + ":" + str);

}

}catch { }

//第六个问题进行try是因为程序都会出现各式各样的问题如断电等因素，这样做不弹出错误提示语，让用户认为应用没有程序问题，catch空什么也不处理,其实微软也是这样做的！

}

private void textSend\_Click(object sender, EventArgs e) {

try{

string tm = textMessage.Text;

byte[] b = System.Text.Encoding.UTF8.GetBytes(tm);

//通过Map保存的K：Ip，找到对应的Socketsend

sendMap[comboBox1.Text].Send(b);

}catch { }

}

}

### Clent客户端

Socket s;

private void button1\_Click(object sender, EventArgs e) {

try{

//创建负责通信的Socket

s = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

IPAddress ip = IPAddress.Parse(textBoxIp.Text);

IPEndPoint point = new IPEndPoint(ip, Convert.ToInt32(textPort.Text));

s.Connect(point);

Thread t = new Thread(Reception);

t.IsBackground = true;

t.Start(s);

}

catch { }

}

/// <summary>

/// 设置显示页面信息

/// </summary>

/// <param name="s"></param>

void ShowMsg(string s)

{

textMessage.AppendText(s+"\r\n");

}

/// <summary>

/// 客户端发送消息

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button3\_Click(object sender, EventArgs e)

{

try {

string str= textSend.Text.Trim();

byte[] b = Encoding.UTF8.GetBytes(str);

s.Send(b);

}

catch { }

}

/// <summary>

/// 客户端接收数据

/// </summary>

private void Reception(Object o)

{

try

{

while (true)

{

Socket socket = o as Socket;

byte[] b = new byte[1024 \* 1024 \* 3];

int r = s.Receive(b);

if (r == 0)

{

break;

}

string str = Encoding.UTF8.GetString(b, 0, r);

ShowMsg(s.RemoteEndPoint + ":" + str);

}

}

catch { }

}

private void Form1\_Load(object sender, EventArgs e)

{

Control.CheckForIllegalCrossThreadCalls = false;

}

## 时间类

### DateTime

DateTime.Now.ToString:获取当前时间

DateTime.Now.Hour：获取小时部分

DateTime.Now.Second:获取秒

## 播放音乐类

此类只能播放wav的音乐

SoundPlayer sp= new SoundPlayer();

//路径path

Sp.SoundLocation=@”d:\1.wav”;

Sp.Play();

# GUID和MD5加密

**GUID：**产生一个不会重复的编号

Console.WriteLine(Guid.NewGuid().ToString());

打印：fe3e7d92-4a03-44ce-9b4d-8b2e1fb18ed2

MD5加密

//创建MD5对象

MD5 md5 = MD5.Create();

//开始加密

//需要将字符处转换成字节数组

byte[] buffer = Encoding.GetEncoding("GBK").GetBytes(str);

//返回一个加密好的字节数组

byte[] MD5Buffer = md5.ComputeHash(buffer);

//将字节数组转换成字符串

//字节数组---字符串

//将字节数组中每个元素按照指定的编码格式解析成字符串

//直接将数组ToString();

//将字节数组中的每个元素ToString()

// return Encoding.GetEncoding("GBK").GetString(MD5Buffer);

// 189 273 345 我爱你

// 189 273 345

string strNew = "";

for (int i = 0; i < MD5Buffer.Length; i++){

strNew += MD5Buffer[i].ToString("x2");

}

return strNew;

}

# Winform

1、winform应用程序是一种智能客户端技术，我们可以使用winform应用程序

帮助我们获得信息或者传输信息等。

2、属性

Name：在后台要获得前台的控件对象，需要使用Name属性。

visible：指示一个控件是否可见。

Enabled：指示一个控件是否可用。

3、事件：发生一件事情。

注册事件：双击控件注册的都是控件默认被选中的那个事件。

触发事件:click等

4、

在Main函数当中创建的窗体对象，我们称之为这个窗体应用程序的主窗体。

也就意味着，当你将主窗体关闭后，整个应用程序都关闭了。

5、TextBox控件

WordWrap:指示文本框是否换行。

PasswordChar：让文本框显示一个单一的字符

ScollBars：是否显示滚动条

事件：TextChanged 当文本框中的内容发生改变的时候触发这个事件。

6、Timer

在指定的时间间隔内做一件指定的事情。

7、单选和多选

checked：指示这个控件是否处于选中状态。

默认情况下，在一个窗体中，所有的单选按钮只允许选中一个，可以使用groupbox进行分组。

8、MDI窗体的设计

1、首先确定一个父窗体。 将IsMdiContainer设置为true。

2、创建子窗体，并且设置他们的父窗体。

## 事件

Timer：每隔指定时间运行同一件事情的事件，工具箱🡪组件🡪timer

使用：属性把默认enabled改为true，这意思是让此事件生效，然后双击写代码了

## 事件的传值

基于两个窗体

第一步 创建事件对象EventHandler，

public event EventHandler evt;

第二步 注册事件：传入方法（方法中必须有两个形参：object sender, EventArgs e）

evt = evt + new EventHandler(f2.setTest);

第三步 判断此事件是否为null，再进行操作evt(this,m);

if (evt !=null) {

evt(this,e);

f2.ShowDialog();

evt = null;

}else{

MessageBox.Show("请先注册事件！");

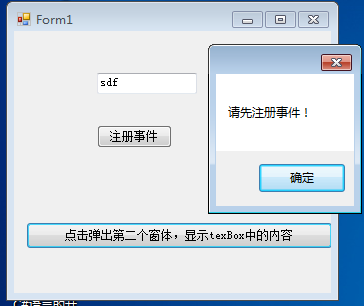
}

第四步 如果有值要传入：需要创建类继承EventHandler类：evt(this,m),m是继承了EcentHandler的对象

private void button1\_Click(object sender, EventArgs e)

有一个叫做EventHandler 的家伙，他会告诉你（主程序），有一些事情发生了：这个事情是谁导致的呢？是某个object类型对象导致的，它用Source或Sender来表示。这个事情是什么事呢？e的内容就是事情的内容了。

例子：



窗体类：form1

public Form1(){

InitializeComponent();

}

//新建事件对象 EventHandler

public event EventHandler evt;

//创建第二个窗体对象

Form2 f2 = new Form2();

//创建自己写的类继承了EventArgs对象:用于传数据

MyEventArgs m = new MyEventArgs();

private void button1\_Click(object sender, EventArgs e) {

//注册事件：具体的方法（这里是窗体2中自己写的方法setTest）

evt = evt + new EventHandler(f2.setTest);

}

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

// textBox1.Text赋值给自己的类有属性text，

m.Text = textBox1.Text;

if (evt !=null) {

//事件源（form1窗体类），事件（自己写的类继承了EventArgs）

evt(this,m);

//使用此方法显示窗体，不然第二次点击出错

f2.ShowDialog();

evt = null;

}else{

MessageBox.Show("请先注册事件！");

}

新建一个类继承事件源EventArgs：主要用来封装数据，因为evt(this,m)中的m需要的参数是：EventArgs类型的。

public class MyEventArgs:EventArgs{

private string text;

public string Text{

get { return text; }

set { text = value; }

}

}

窗体2：form2

public partial class Form2 : Form{

public Form2(){

InitializeComponent();

}

//自己写的方法，形参事件源，事件内容

public void setTest(object sender, EventArgs e) {

MyEventArgs c= e as MyEventArgs;

textBox1.Text = c.Text;

}

## 例子

### 1、爱与不爱

点击爱与不爱的列子：移到不爱触发事件让它随机在屏幕范围内随机变动，让你点击不了

点击到爱button显示信息：

用到事件：

MouseEnter：在鼠标进入控件的可见部分发生

MouseClick：点击事件

用到的类：

Random：随机类

Point：提供有序的x和y坐标，在二维平面上定义一个点

ClientSize： ClientSize这个类是可以随着用户拉大而变化，可以获取长和kuan

方法：button2.Location获取或设置button坐标

事件中的方法：

private void button1\_Click(object sender, EventArgs e) {

MessageBox.Show("我也爱你思密达！！！");

}

private void button2\_MouseEnter(object sender, EventArgs e) {

//获得按钮的宽

int butt=this.button2.Width;

//获得按钮的高

int height= this.button2.Height;

//获得窗体的宽，ClientSize这个类是可以随着用户拉大而变化

int cw= this.ClientSize.Width;

//获得窗体的长

int ch = this.ClientSize.Height;

int rWidth = cw - butt;//减去按钮的宽度

int rhegth = ch - height;

Random r = new Random();

button2.Location = new Point(r.Next(0, rWidth + 1), r.Next(0, rhegth + 1));

}

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

【消息盒子】 MessageBox.Show("还是被你电机到了");

this.Close();

}

### 2、输入内容在label显示

把textBox【文本盒子】的内容通过事件：值改变，赋值到label中

控件：

textBox：允许用户输入文本、密码

label1：提供信息说明文字

事件：

Changed：更改text属性的值是引发的事件

当文本框中的内容发生改变的时候 将值赋值给Label

private void txtWords\_TextChanged(object sender, EventArgs e) {

lblText.Text = txtWords.Text;// lblText、txtWords：属性name的值

}

### 3、跑马灯

1、显示五角星在跑

2、显示系统时间

3、闹铃功能：播放wav格式的音乐

工具箱中的组件：timer：指定时间触发一次时间

控件：label

事件：

每隔指定时间运行同一件事情的事件，工具箱组件timer

类：

DateTime：事件类

SoundPlayer：播放wav格式的类

运用到了spring的截取subspring功能截取第一个，把他追加到字符串最后面

private void timer1\_Tick(object sender, EventArgs e) {

//MessageBox.Show("你中病毒啦~~~！！！关不掉了吧");

label1.Text = label1.Text.Substring(1) + label1.Text.Substring(0, 1);

}

每隔一秒钟就把当前的时间赋值给label

private void timer2\_Tick(object sender, EventArgs e) {

lblTme.Text = DateTime.Now.ToString();

//15:32分播放音乐叫我起床

if (DateTime.Now.Hour == 15 && DateTime.Now.Minute == 34 && DateTime.Now.Second == 50){

//播放音乐

SoundPlayer sp = new SoundPlayer();

sp.SoundLocation = @"C:\Users\SpringRain\Desktop\1.wav";

sp.Play();

}

当窗体加载的时候 将当前系统的时间赋值给我的Label

private void Form1\_Load(object sender, EventArgs e) {

lblTme.Text = DateTime.Now.ToString();

}

### 4、简单记事儿本应用程序

1)、在程序加载的时候，取消文本框的自动换行，以及让两个按钮和文本框隐藏

2)、点击登陆，判断是否登陆成功

3)、自动换行功能

4)、保存文本到一个指定目录下。

### WebBrowser浏览器控件

模仿浏览器在uri中输入网址点击前往进入页面

事件：click

控件：WebBrowser：工具箱—>公共控件—>最后一个

Button、textBox

private void button1\_Click(object sender, EventArgs e) {

string s = textBox1.Text;

Uri u = new Uri("http://" + s);

webBrowser1.Url = u;

### 下拉框comboBox操作

事件：click

控件：comboBox：工具箱—>公共控件

private void button1\_Click(object sender, EventArgs e) {

//通过代码获得集合增加comboBox内容

comboBox2.Items.Add("张三");

comboBox2.Items.Add("李四");

}

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

comboBox2.Items.Clear();

}

日期选择器

功能：选择第一个下拉框（年），获得下一个（月）再选择月，获得下一个下垃框（日）

控件：comboBox

事件：load,SelectedIndexChang

使用到的方法：通过窗体加载使用DateTime.Now.Year获得年份

难度：第三个下垃框（日）较难：只有2月才跟年有关系（闰年，平年），

private void Form1\_Load(object sender, EventArgs e) {

程序加载的时候 将年份添加到下拉框中

获得当前的年份

int year = DateTime.Now.Year;

for (int i = year; i >= 1949; i--){

cboYear.Items.Add(i + "年");

}

}

当年份发生改变的时候 加载月份

private void cboYear\_SelectedIndexChanged(object sender, EventArgs e) {

//添加之前应该清空之前的内容

cboMonth.Items.Clear();

for (int i = 1; i <= 12; i++){

cboMonth.Items.Add(i + "月");

}

}

当月份发生改变的时候 加载天

private void cboMonth\_SelectedIndexChanged(object sender, EventArgs e) {

cboDays.Items.Clear();

int day = 0;//定义一个变量来存储天数

//获得月份cboMonth.SelectedItem.ToString()这个方法才能拿到选择的值：12月，切掉“月”

string strMonth = cboMonth.SelectedItem.ToString().Split(new char[] { '月' }, StringSplitOptions.RemoveEmptyEntries)[0];//此方法获得数组

string strYear = cboYear.SelectedItem.ToString().Split(new char[] { '年' }, StringSplitOptions.RemoveEmptyEntries)[0];

//转换string为int

int year = Convert.ToInt32(strYear);

int month = Convert.ToInt32(strMonth);

switch (month) {

case 1:

case 3:

case 5:

case 7:

case 8:

case 10:

case 12: day = 31;

break;

case 2:

//判断是否闰年

if ((year % 400 == 0) || (year % 4 == 0 && year % 100 != 0)) {

day = 29;

}else{

day = 28;

}

break;

//30天的月份

default: day = 30;

break;

}

for (int i = 1; i <= day; i++){

cboDays.Items.Add(i + "日");

}

## ListBox列表控件

功能：常用来列表数据的：如播放器音乐列表

**控件：**ListBox

**事件：load**

**使用到的方法：**

**难度：**

private void Form1\_Load(object sender, EventArgs e) {

listBox1.Items.Add(1);

listBox1.Items.Add(1);

listBox1.Items.Add(1);

listBox1.Items.Add(1);

listBox1.Items.Add(1);

}

实现点击显示照片

控件：

事件：doubleClick：双击事件

使用到的方法：

//用来存储图片文件的全路径

List<string> list = new List<string>();

private void Form1\_Load(object sender, EventArgs e) {

//直接把路径放到下面就可以了，不用再声明list，老师告诉你这样也可以做

string[] path = Directory.GetFiles(@"C:\Users\SpringRain\Desktop\Picture", "\*.jpg");

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

string fileName = Path.GetFileName(path[i]);

listBox1.Items.Add(fileName);

//将图片的全路径添加到List泛型集合中

list.Add(path[i]);

//listBox1.Items.Add(path[i]);

}

}

/// 双击播放图片

private void listBox1\_DoubleClick(object sender, EventArgs e) {

pictureBox1.Image = Image.FromFile(list[listBox1.SelectedIndex]);

}

}

双击播放音乐和上面原理一模一样的

## 对话框

对话框类都是dialog后缀的

**功能：模仿windows文本：打开对话框中的text文件：**OpenFileDialog

private void button1\_Click(object sender, EventArgs e) {

//点击弹出对话框

OpenFileDialog ofd = new OpenFileDialog();

//设置对话框的标题

ofd.Title = "请选择要打开的文本文件哟亲 O(∩\_∩)O~";

//设置对话框可以多选

ofd.Multiselect = true;

//设置对话框的初始目录,就打开文本出来的目录

ofd.InitialDirectory = @"C:\Users\SpringRain\Desktop";

//设置对话框的文件类型，就是设置刷选类型

ofd.Filter = "文本文件|\*.txt|媒体文件|\*.wmv|图片文件|\*.jpg|所有文件|\*.\*";

//展示对话框，要放到最后

ofd.ShowDialog();

//获得在打开对话框中选中文件的路径，还有一个FileNames多选

string path = ofd.FileName;

//选择了”取消”判断，就是没有path是空

if (path == "")

{

return;

}

using (FileStream fsRead = new FileStream(path, FileMode.OpenOrCreate, FileAccess.Read))

{

byte[] buffer = new byte[1024 \* 1024 \* 5];

//实际读取到的字节数

int r = fsRead.Read(buffer, 0, buffer.Length);

textBox1.Text = Encoding.Default.GetString(buffer, 0, r);

}

}

**保存文件对话框：**SaveFileDialog

private void button1\_Click(object sender, EventArgs e) {

SaveFileDialog sfd = new SaveFileDialog();

sfd.Title = "请选择要保存的路径";

sfd.InitialDirectory = @"C:\Users\SpringRain\Desktop";

sfd.Filter = "文本文件|\*.txt|所有文件|\*.\*";

sfd.ShowDialog();

//获得保存文件的路径

string path = sfd.FileName;

if (path == "")

{

return;

}

using (FileStream fsWrite = new FileStream(path, FileMode.OpenOrCreate, FileAccess.Write)) {

byte[] buffer = Encoding.Default.GetBytes(textBox1.Text);

fsWrite.Write(buffer, 0, buffer.Length);

}

MessageBox.Show("保存成功");

}

字体颜色对话框

/// 字体对话框FontDialog

private void button1\_Click(object sender, EventArgs e) {

FontDialog fd = new FontDialog();

fd.ShowDialog();//设置打开

textBox1.Font = fd.Font;

}

颜色对话框：FontDialog

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

ColorDialog cd = new ColorDialog();

cd.ShowDialog();

textBox1.ForeColor = cd.Color;

}

}

## Panel容器

允许对控件集合分组

点击显示和隐藏操作

private void button1\_Click(object sender, EventArgs e) {

panel1.Visible = true;

}

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

panel1.Visible = false;

}

# 连接数据库

连接mysql

第一步 引用驱动：MySql.Data.dll，导入命名空间：MySql.Data.MySqlClient

mysql官网下载：mysql-connector-net-6.3.8.msi此包安装，在目录中就有驱动

MySQL\MySQL Connector Net 6.3.8\Assemblies\v4.0

第二步 连接数据库url

string l= "server=127.0.0.1;user id=root;password=123456;database=wwl";

第三步 打开数据库

m.Open();

第四步 准备sql语句和传送sql语句对象

string sql = "select \* from user";

MySqlCommand mc = new MySqlCommand(sql,m);

第五步 创建对象：CRUD

增删改对象ExecuteNonQuery【执行非查询】

Int insert=mc.ExecuteNonQuery()

查询对象：MySqlDataReader获取数据有两种写法：第一种：md["id"] 第二种：md.GetString("name")

MySqlDataReader md = mc.ExecuteReader();

if(md.HasRows){ //判断有无值

while (md.Read()){//判断有无下一行

int i= (int)md["id"];//强制转换

int i2= Convert.ToInt32(md["id"]);//使用Convert对象转换

string str = md["name"].ToString();

int i= md.GetInt32("id");

string str = md.GetString("name");

Console.WriteLine(i+":"+str);

返回一条数据： executeScalar

}

防止sql语句注入：

1、语句中参数不用：UserName=user.username，而是变成：@name（自己随便取名）

Select count(\*) from UserLogin where UserName=@name and UserPwd=@自己随便取值

2、使用方法：mc.parameters

MySqlCommand mc = new MySqlCommand(sql,m);

mc.parameters.AddWithValue(“@name”,”wenwenliang”)//账号

mc.parameters.AddWithValue(“@自己随便取值”,”123456”)//密码

mc.Parameters.AddRange(放入数组)//此方法放入数组

## 临时数据库

Ado.net(断开式数据库)

导入命名空间；using System.Data;

代码

static void Main(string[] args)

{

//创建数据库

DataSet d = new DataSet("wwl");

//创建表

DataTable dt = new DataTable("user");

//创建列

DataColumn dc1 = new DataColumn("id",typeof(int));//第一个列

DataColumn dc2 = new DataColumn("name", typeof(string));//第二列

//把列添加到表中

dt.Columns.Add(dc1);

dt.Columns.Add(dc2);

//把表到数据库中

d.Tables.Add(dt);

dt.Rows.Add(1, "你好");

dt.Rows.Add(2,"不好");

//遍历

foreach(DataTable table in d.Tables){

foreach(DataRow row in table .Rows){

Console.WriteLine(row[0]+":"+row[1]);

}

Console.ReadKey();

}

}

# 设计模式

## 工厂模式

所谓工厂模式就是生产对象，看你需要什么对象，给你什么对象

namespace \_03简单工厂设计模式{

class Program {

static void Main(string[] args) {

Console.WriteLine("请输入您想要的笔记本品牌");

string brand = Console.ReadLine();

NoteBook nb = GetNoteBook(brand);

nb.SayHello();

Console.ReadKey();

}

/// 简单工厂的核心 根据用户的输入创建对象赋值给父类

public static NoteBook GetNoteBook(string brand) {

NoteBook nb = null;

switch (brand) {

case "Lenovo": nb = new Lenovo();

break;

case "IBM": nb = new IBM();

break;

case "Acer": nb = new Acer();

break;

case "Dell": nb = new Dell();

break;

}

return nb;

}

}

public abstract class NoteBook{

public abstract void SayHello();

}

public class Lenovo : NoteBook {

public override void SayHello(){

Console.WriteLine("我是联想笔记本，你联想也别想");

}

}

public class Acer : NoteBook{

public override void SayHello(){

Console.WriteLine("我是鸿基笔记本");

}

}

public class Dell : NoteBook {

public override void SayHello(){

Console.WriteLine("我是戴尔笔记本");

}

}

public class IBM : NoteBook{

public override void SayHello(){

Console.WriteLine("我是IBM笔记本");

}

# 时间类

获取当前时间

DateTime dt = new DateTime();

Console.WriteLine(dt);//打印：0001/1/1 0:00:00

DateTime dt2= DateTime.Now;//获取当前时间

Console.WriteLine(dt2.ToString("d"));//转换格式为 C#默认的格式：d（短日期格式） 如 2016/5/9 D(长日期格式) 2016年5月9日

C# 中字符串转换成日期

我们在处理字符串日期格式常用DateTime.Pares() 但是这个形式的转换是相当有限的，有些C#是会不懂你写入的日期格式的如20031231。那么类似 "20100101" 或者其它形式的字符串如何转换成日期型呢?

一、拼接字符串的形式

DateTime dt=Convert.ToDateTime("20100101".Substring(0,4)+"-"+"20100101".Substring(4,2)+"-"+"20071107".Substring(6,2));

二、Convert.ToDateTime(string)

string格式有要求，必须是yyyy-MM-dd hh:mm:ss

三、Convert.ToDateTime(string, IFormatProvider)

DateTime dt;

需要导包：using System.Globalization;

DateTimeFormatInfo dtFormat = new System.Globalization.DateTimeFormatInfo();

dtFormat.ShortDatePattern = "yyyy/MM/dd";

dt = Convert.ToDateTime("2011/05/26", dtFormat);

四、DateTime.ParseExact()

string dateString = "20110526";

DateTime dt = DateTime.ParseExact(dateString, "yyyyMMdd", System.Globalization.CultureInfo.CurrentCulture);

//或者

DateTime dt = DateTime.ParseExact(dateString, "yyyyMMdd", System.Globalization.CultureInfo.InvariantCulture);

五、DateTime.TryParse(string,out datetime)

1、更多时候，会采用DateTime.TryParse(string,out datetime)方法，因为此方法有安全机制，当string内容不正确时，可以返回日期的最小值MinValue。并且可以通过返回的bool值判断转化是否成功。而DateTime.ParseExact()需要按特定的格式来转换，对格式的要求比较严，如果string中不是日期内容，而量类似“asdfasd”的字符串，则会出错。

2、用DateTime.TryParse(string,out datetime)转换后，得到的datetime可以用 datetime.ToString("ddd, MMM. dd")来转换为特殊需求的格式，比较灵活方便。

DateTime dt = new DateTime();

DateTime.TryParse("2018/9/12",out dt);

Console.WriteLine( dt.ToString("yyy/MM/dd"));

# C#和java不同点

## 继承

重写父类同名方法：C#需要在子类中和父类相同的方法上写上关键字：override

使用多态才会显现出区别

子类（class修饰）的访问权限不能高于父类的访问权限，会暴漏父类的成员。（经测试和java刚好相反）

Java重写父类方法的时候是可以把protected修饰的方法改为public，而C#不可以

public class Test{

public static void Main(String[] args){

A a=new B(); //多态调用

Console.Write(a.getVal()); // 输出:A，没有使用关键字但父类和子类都是相同的方法名，调用的是父类的方法（java是子类方法）

a=new C();

Console.Write(a.getVal()); // 输出:C 实例是C，使用了关键字override

}

}

class A {

// 要允许子类重写，方法必须标识为virtual 或 abstract

public virtual String getVal(){

return "A";

}

}

class B : A {

// 隐藏父类方法 用new标识，不标识时会出现编译警告，不影响使用

public new String getVal(){

return "B";

}

}

class C : A {

// 重写方法 必须用override标识

public override String getVal(){

return "C";

}

}

# Java和C#基本类库的区别

java.lang

|  |  |
| --- | --- |
| java | .net |
| Boolean | System.Boolean |
| Byte | System. Byte |
| Character | System.Char |
| Class | System.Type |
| Double | System.Double |
| Float | System.Single |
| Integer | System.Int32 |
| Long | System.Int64 |
| Math | System.Math |
| Object | System.Object |
| Process | System. Diagnostics .Process |
| Runtime | System. Diagnostics .Process |
| Short | System.Int16 |
| StrictMath | System.Math |
| Strng | System.Strng |
| StringBuffer | System. Text .StringBuffer |
| Thread | System. Threading .Thread |
| ThreadLocal | System.LocalDataStoreSlot |

java.lang.reflect

|  |  |
| --- | --- |
| java | .net |
| AccessibleObject | System.reflection.MemberInfo |
| Array | System. Array |
| Constructor | System.reflection.ConstructorInfo |
| Field | System.reflection.FieldInfo |
| Method | System.reflection.MethodInfo |
| Modifier | System.reflection.MethodAttributes |
| ReflectdPermission | System. Security . Permissions.ReflectdPermission |

java.lang.ref

|  |  |
| --- | --- |
| java | .net |
| WeakReference | System.WeakReference |

java.util包

java.util

|  |  |
| --- | --- |
| java | .net |
| AbstractColletion | System.Collections.CollectionsBase |
| ArrayList | System.Collections.ArrayList |
| Arrays | System.Arrays |
| BitSet | System.Collections.BitArray |
| Calendar | System.Globalization.Calendar |
| Currency | System.Globalization.RegionInfo |
| Date | System.Datetime |
| Dictionary | System.Collections.DictionaryBase |
| EventObject | System.EventArgs |
| GregorianCalendar | System.Globalization.GregorianCalendar |
| HashMap | System.Collections.HashMap |
| Hashtable | System.Collections.HashMap |
| ListResourseBundle | System.Resources.ResourceManger |
| Locale | System.Globalization.CultureInfo |
| Random | System.Random |
| ResourseBundle | System.Resources.ResourceSet |
| SimpleTimeZone | System.Datetime |
| Stack | System.Collections.Stack |
| Timer | System.Threading.Timer |
| TimerTask | System.Threading.TimerCallback |
| TimeZone | System.DataTime |
| TreeSet | System.Collections.SortedList |
| Vector | System.Collections.ArrayList |

java.util.regex

|  |  |
| --- | --- |
| java | .net |
| Matcher | System.Text.RegularExpressions.Match  System.Text.RegularExpressions.MatchCollection  System.Text.RegularExpressions.Group  System.Text.RegularExpressions.GroupCollection |
| Pattern | System.Text.RegularExpressions.Regex  System.Text.RegularExpressions.RegexCompilationInfo  System.Text.RegularExpressions.RegexOption |
| PatternSyntaxExcetpion | System. ArgumentExcetion |

java.util.logging/java.util.jar/java.util.zip/java.util.pref 包中的类在.NET中没有与之对应的部分

java.io包

|  |  |
| --- | --- |
| java | .net |
| BufferedInputStream | System.IO.BufferedStream |
| BufferedOutputStream | System.IO.BufferedStream |
| BufferedReader | System.IO.StreamReader |
| BufferedWriter | System.IO.StreamWriter |
| ByteArrayInputStream | System.IO.MemoryStream |
| ByteArrayOutputStream | System.IO.MemoryStream |
| CharArrayReader | System.IO.StreamReader |
| CharArrayWriter | System.IO.StreamWriter |
| DataInputStream | System.IO.BinaryReader |
| DataOutputStream | System.IO.BinaryWriter |
| File | System.IO.File |
| FileInputStream | System.IO.FileStream |
| FileOutputStream | System.IO.FileStream |
| FileReader | System.IO.StreamReader |
| FileWriter | System.IO.StreamWriter |
| InputStream | System.IO.Stream |
| OutputStream | System.IO.Stream |
| PrintStream | System.IO.StreamWriter |
| PrintWriter | System.IO.StreamWriter |
| PushbackInputStream | System.IO.StreamReader |
| PushbackOutputStream | System.IO.StreamReader |
| RamdomAccessFile | System.IO.FileStream |
| StringBufferInputstream | System.IO.StringReader |
| StringReader | System.IO.StringReader |
| StringWriter | System.IO.StringWriter |

java.sql包

.net 中没有与javax.sql包直接对应的部分，System.Data命名空间提供了连接池和事务处理类。

java的Rowset类在某些方面与.net的DataDet类相似

|  |  |
| --- | --- |
| java | .net |
| Blob | System.Data.Sqlclient.SqlDataReader  System.Data.OleDb.OleDbDataReader |
| CallableStatement | System.Data.Sqlclient.Sqlcommand  System.Data.OleDb.OleDbcommand |
| Clob | System.Data.Sqlclient.SqlDataReader  System.Data.OleDb.OleDbDataReader |
| Connection | System.Data.Sqlclient.Sql  System.Data.OleDb.OleDb |
| Date | System.Data. SqlTypes . SqlDateTime |
| ParameterMetaData | System.Data.Sqlclient.SqlParameter  System.Data.OleDb.OleDbParameter |
| PreparedStatement | System.Data.Sqlclient.SqlCommand  System.Data.OleDb.OleDbCommand |
| ResultSet | System.Data.Sqlclient.SqlDataReader  System.Data.OleDb.OleDbDataReader |
| ResultSetMetaData | System.Data.Sqlclient.SqlDataReader  System.Data.OleDb.OleDbDataReader |
| Savepoint | System.Data.Sqlclient. SqlTransaction |
| SQLPermission | System.Data.Sqlclient.SqlclientPermission  System.Data.OleDb.OleDbPermission |
| Statement | System.Data.Sqlclient.SqlParameter  System.Data.OleDb.OleDbCommand |
| Time | System.Data. SqlTypes . SqlDateTime |
| Types | System.Data.DbTypes |

java.text包

|  |  |
| --- | --- |
| java | .net |
| DataFormat | System.DateTime |
| DateFomatSymbols | System.Globalization.DateaTimeFormatInfo |
| NumberFormat | System.Globalization.NumberFormatInfo |
| SimpleDateFormat | System.Globalization.DateTimeFormatInfo |
| StringCharacterIterator | System.IEnumerator (System,String) |

org.w3c.dom包

|  |  |
| --- | --- |
| java | .net |
| Attr | System.Xml.XmlAttribute |
| CdatSection | System.Xml.XmlCDataSetion |
| CharacterData | System.Xml.XmlCharacterData |
| Comment | System.Xml.XmlComment |
| Document | System.Xml.XmlDocument |
| DocumentFragment | System.Xml.XmlDocumentFragment |
| DocumentType | System.Xml.XmlDocumentType |
| DOMImplementation | System.Xml.XmlDOMImplementation |
| Element | System.Xml.XmlElement |
| Entity | System.Xml.XmlEntity |
| EntityReference | System.Xml.XmlEntityReference |
| NameNodeMap | System.Xml.XmlNameNodeMap |
| Node | System.Xml.XmlNode |
| NodeList | System.Xml.XmlNodeList |
| Notation | System.Xml.XmlNotation |
| ProcessingInstruction | System.Xml.XmlProcessingInstruction |
| DOMException | System.ArgumentException |

javax.swing包

.net没有提供与下列包直接等价的类：

javax.swing.plaf/javax.swing.border/javax.swing.table/javax.swing.text( 可用System.Windows.Form.RichTextBox 代替 ) 和 javax.swing.filechooser （ 可用System.Windows.Form.OpenFileDialog 和 System.Windows.Form.SaveFileDialog 代替 ）

javax.swing

|  |  |
| --- | --- |
| java | .net |
| AbstractButton | System.Windows.Forms. ButtonBase |
| AbstractListModel | System.Windows.Forms.ListControl |
| AbstractSpinnerModel | System.Windows.Forms.UpDownBase |
| ImageIcon | System.Windows.Forms.Image |
| Jbutton | System.Windows.Forms.Button |
| JCheckBox | System.Windows.Forms.CheckBox |
| JColorChooser | System.Windows.Forms.ColorDialog |
| JComboBox | System.Windows.Forms.ComboBox |
| JComponent | System.Windows.Forms.UserControl |
| JDialog | System.Windows.Forms.CommonDialog |
| JEditorpane | System.Windows.Forms.TextBoxBase |
| JFileChooser | System.Windows.Forms.OpenFileDialog |
| JFormattedTextField | System.Windows.Forms.RichTextBox |
| JFrame | System.Windows.Forms.Form |
| JLabel | System.Windows.Forms.Label |
| JList | System.Windows.Forms.ListBox |
| JMenuBar | System.Windows.Forms.MainMenu |
| JMenuItem | System.Windows.Forms.MenuItem |
| JPanel | System.Windows.Forms.Panel |
| JPasswordField | System.Windows.Forms.TextBox |
| JPopupMenu | System.Windows.Forms.ContextMenu |
| JProgressBar | System.Windows.Forms.ProgressBar  StatusBar |
| JRadioButton | System.Windows.Forms.RadioButton |
| JScrollBar | System.Windows.Forms.HScrollBar  System.Windows.Forms.VScrollBar |
| JScrollPane | System.Windows.Forms.Panel |
| JSlider | System.Windows.Forms.TrackBar |
| JSpinner | System.Windows.Forms.DomainUpDown |
| JSpitPane | System.Windows.Forms.Splitter |
| JTable | System.Windows.Forms.ListView |
| JTextArea | System.Windows.Forms.TextBox |
| JTextField | System.Windows.Forms.TextBox |
| JTextPane | System.Windows.Forms.RichTextBox |
| JTogleButton | System.Windows.Forms. ButtonBase |
| JToolBar | System.Windows.Forms.ToolBar |
| JToolTip | System.Windows.Forms.ToolTip |
| JTree | System.Windows.Forms.ListView |
| Popup | System.Windows.Forms.ContextMenu |
| Timer | System.Windows.Forms.Timer |

javax.swing.event

|  |  |
| --- | --- |
| java | .net |
| CaratEvent | System.EventArgs |
| ListDataEvent | System.EventArgs |
| ListSelectionEvent | System.EventArgs |
| MenuEvent | System.EventArgs |
| PopupMenuEvent | System.EventArgs |

java.awt包

java.awt

|  |  |
| --- | --- |
| java | .net |
| AWTEvernt | System.EventArgs |
| AWTKeyStroke | System.Windows.Forms.KeyPressEventArgs |
| AWTPermission | System.Security.Permissions.UIPermission |
| BasicStroke | System.Drawing.Pen |
| Button | System.Windows.Forms.Button |
| Canvas | System.Windows.Forms. Control |
| CheckBox | System.Windows.Forms.RadioButton |
| Choice | System.Windows.Forms.ComboBox |
| Color | System.Drawing.Color |
| Component | System.Windows.Forms. Control |
| [Container](http://lib.csdn.net/base/docker) | System.Windows.Forms. Control |
| Cursor | System.Windows.Forms.Cursor |
| Dialog | System.Windows.Forms.CommonDialog |
| Dimension | System.Drawing.Size |
| Event | System.EventArgs |
| FileDialog | System.Windows.Forms.FileDialog |
| Font | System.Drawing.Font |
| Frame | System.Windows.Forms.Form |
| GradientPaint | System.Drawing.Drawing2D.LinearGradientBrush |
| Graphics | System.Drawing.Graphics |
| Graphics2D | System.Drawing.Graphics |
| Image | System.Drawing.Image |
| JobAttribute | System.Drawing.Printing. PageSettings |
| Label | System.Windows.Forms.Label |
| List | System.Windows.Forms.ListBox |
| Menu | System.Windows.Forms.MainMenu |
| MenuItem | System.Windows.Forms.MenuItem |
| PageAttributes | System.Drawing.Printing. PageSettings |
| Panel | System.Windows.Forms. Panel |
| Point | System.Drawing.Point |
| PopupMenu | System.Windows.Forms.ContextMenu |
| PrintJob | System.Drawing.Printing.PrintDocument |
| Rectangle | System.Drawing.Rectangle |
| Scrollbar | System.Windows.Forms.HScrollbar  System.Windows.Forms.VScrollbar |
| ScrollPane | System.Windows.Forms. Panel |
| TextArea | System.Windows.Forms. TextBox |
| TextField | System.Windows.Forms. TextBox |
| TexturePaint | System.Drawing.TextreBrush |

java.awt.event

|  |  |
| --- | --- |
| java | .net |
| ActionEvent | System.EventArgs |
| AdjustmentEvent | System.Windows.Forms.ScrollEventArgs |
| ComponentEvent | System.EventArgs |
| ContainerEvent | System.Windows.Forms.ControlEventArgs |
| FocusEvent | System.EventArgs |
| ItemEvent | System.EventArgs |
| KeyEvent | System.Windows.Forms.KeyPressEventArgs |
| MouseEvent | System.Windows.Forms.MouseEventArgs |
| PaintEvent | System.Windows.Forms.PaintEventArgs |
| TextEvent | System.EventArgs |
| WindowEvent | System.EventArgs |

java.awt.print

|  |  |
| --- | --- |
| java | .net |
| PageFormat | System.Drawing.Printing.PageSettings |
| Paper | System.Drawing.Printing.PageSize |
| PrinterJob | System.Drawing.Printing.PrintDocument |