c#数据类型

整形

sbyte 有符号数

byte 无符号数

short 有

ushort 无

int 有

uint 无

long 有

ulong 无

浮点型

float

double

字符和字符串

char

String

字符串的不可变性：当给字符串重新赋值时，会在堆内存中重开空间记录字符串值，然后更改变量栈内存地址，原来的字符串堆内存不会消失，等待GC垃圾回收

可以把字符串看做char的只读数组

ToCharArray（） 字符串转char数组

new String （char[] chs） char数组转string

当对字符串进行大量修改时，使用StringBuilder类操作，否则浪费内存

属性

Length 字符串长度

方法

ToUpper（）将字符转换为大写形式

ToLower（）将字符转换为小写形式

Equals（字符串，可选参数）比较两个字符串

Split（）分割字符串

Substring（）截取字符串

IndexOf（）字符串首次出现位置

LastIndexOf（）字符串最后一次出现位置

StartsWith（）判断以。。。。开始

EndsWith（）判断以。。。结束

Replace（）字符串替换

Contains（）是否包含指定字符串

Trim（）去掉左右两边空格

TrimEnd（）去掉结尾空格

TrimStart（）去掉开头空格

String.IsNullOrEmpty() 字符串是否为空或null

String.Join() 字符串拼接

布尔型

bool

c#变量

数据类型 变量名；

变量名=值；

数据类型 变量名=值；

C#常量

const 数据类型 变量名=值；

c#值类型和引用类型

区别：

值类型直接存储在栈内存

应用类型存储在堆内存

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

引用类型：string，自定义类，数组，集合，object，接口

c#类

类的访问修饰符 修饰符 类名{

类成员

}

访问修饰符

public 公有 private 私有 internal 仅能同一项目访问 protected 类或派生类访问

修饰符

readload 只读 static 静态

定义成员变量 ： 访问修饰符 修饰符 数据类型 变量名;

成员方法

访问修饰符 修饰符 返回值类型 方法名（参数列表）{

方法体

}

方法修饰符

virtual 虚拟的 abstract 抽象的 override 重写的 static 静态的 sealed 密封的

get set访问器 ，分别用于获取和设置成员变量的值

public 数据类型 属性名{

get{

获取属性的语句块；

return 值；

}

set{

设置属性

}

}

注意：

1. get访问器需要return一个值
2. set访问器省略后无法在其他类复制，变为只读属性
3. 可以简写 访问修饰符 数据类型 变量名{set；get；}

调用类成员

类名 类的对象名 = new 类名（）；

对象名.类成员

对象名.成员方法（参数）

构造方法的方法名与类名一致

析构方法，析构方法不带任何参数

~类名（）{

方法体；

}

重载 ： 方法名称相同，参数个数或类型不同

方法中的参数除了定义数据类型外，还有引用参数和输出参数

引用参数关键字 ref

输出参数关键字 out

引用参数使用案例

//定义一个类

class OutClass{

public void Pri( ref string name , out int age){

if( name != ‘’ || name != null ){

age = 1;

}else{

age = 0;

}

}

}

//在main方法中调用

class Program{

static void main(){

OutClass ooo = new OutClass();

string user\_name = “nvdia”;

int age;

ooo.Pri(ref user\_name , out age);

Console.WriteLine(age);

}

}

使用引用参数时和正常变量没啥区别，但是使用了ref关键字时，方法体中改变变量的值，外部引用的值会直接改变，相当于共用一个变量地址

使用输出参数时，必须给输出参数赋值，在调用含有输出参数的方法时，没必要提前赋值

输出参数的功能是在方法中返回多个值

方法中的可变参数params

必须是形参的最后一个参数

案例

public void Test(string name , params int[] iii ){

}

//调用

Test(“adsd” , 11 , 22 , 33 , 44)

会把后面的整数自动放进形参数组中

Lambda表达式，表达式只能写一个语句，是一种简写方式

访问修饰符 修饰符 返回值类型 方法名（参数类表）=> 表达式；

嵌套类

案例

class OutClass{

class InClass{

public string name{set; get;}

public void age(){

方法体；

}

}

}

//调用方法

OutClass.InClass in = new OutClass.InClass();

in.name = “hello”; in.age();

c#序列化和反序列化

序列化：对象转为二进制

反序列化：二进制转对象

序列化步骤：

1. 在类的上面加 [Serializable] 指示一个类可以序列化，不能被继承
2. 使用 BinaryFormatter bf = new BinaryFormatter(); 创建序列化对象
3. 使用 bf.Serialize(文件流对象，需要被序列化的对象)

反序列化

1. 使用 BinaryFormatter bf = new BinaryFormatter(); 创建序列化对象
2. 使用bf.Deserialize(二进制序列化对象)；

c#部分类

在类的访问修饰符后加partial 表示部分类，是类的一部分

c# Console类

主要用于程序的输入和输出

方法

Write 控制台输出内容且不换行

WriteLine 输出内容且换行

Read 读取一个字符

ReadLine 读取一行字符

输出内容可以进行格式化输出

Console.WriteLine(格式化字符串，输出项1，输出项2，，，，，);

案例

Console.WriteLine("{0}同学在{1}学习", name, school);

c# Math类

方法

Abs 绝对值

Ceiling 四舍五入最小整数值

Floor 四舍五入最大整数值

Equals 对象实例是否相等

Max 两数取大

Min 两数取小

Sqrt 平方根

Round 四舍五入的值

C#Random类

用于产生随机数

构造方法

New Random（） 根据系统时间生成

New Random（Int32） 自己设定生成

方法

Next（）随机正整数

Next（int max Value）小于max Value的随机正整数

Next（int min Value ， int max Value）在区间产生随机正整数

NextDouble（） 产生0.0 ~ 1.0 的浮点数

NextBytes（byte[] buffer） 用随机数填充指定字节数的数组

C#DateTime类

获取当前日期和时间 DateTime.Now

方法

Date 获取实例的日期部分

Day 一个月的第几天

DayOfWeek 星期几

DayOfYear 哪一年

Add（Timespan value） 添加时间间隔value， Timespan是时间间隔类

AddDays（double value） 添加天数value

AddHours（double value）添加小时

AddMinutes（double value） 添加分钟

AddSeconds（double value） 添加秒

AddMonths（int value） 添加月

AddYears（int value） 添加年

C#字符串及常用方法

属性

Length 字符串长度

字符串.Length

方法

IndexOf 返回整数，指定字符串第一次出现的位置 , 找不到返回-1

字符串.IndexOf(“查找的字符串”)

LastIndexOf 返回整数，指定字符串最后一次出现的位置，找不到返回-1

字符串.LastIndexOf(“查找的字符串”)

StartsWith 返回bool，是否以指定字符串开头

EndsWith 返回bool，是否以指定字符串结尾

ToLower 返回字符串，大写转小写

ToUpper 返回字符串，小写转大写

Trim 返回string，不带参数表示删除左右两边的空格，有参数删掉包含的参数

Remove 返回string，删除指定位置的字符串

TrimStart 返回string，删除左侧空格

TrimEnd 返回string，删除右侧空格

PadLeft 返回string，左侧填充空格达到指定长度

PadRight 返回string，右侧填充空格达到长度

Split 返回string[] ，字符串分割

Replace 返回string，字符串替换

字符串.Replace(“被替换的”,”替换的”)

Substring 返回string，字符串截取

Substring（指定位置）//从指定位置截取到结束

Substring（指定位置，截取个数） //指定位置截取指定个数

Insert 返回string，字符串插入到指定位置

字符串.Insert(位置，字符串)

Concat 返回string，字符串合并

C#数据类型转换

显示转换，常用方法

ToBoolean 转布尔

ToByte 转字节

ToChar 转字符

ToDateTime 转日期-时间结构

ToDecimal 浮点或整形转十进制类型

ToDouble 转浮点

ToInt16 转16位整形

ToInt32 转32位整形

ToInt64 转64位整形

ToSbyte 转有符号字节

ToSingle 转小浮点数

ToString 转字符串

ToType 转指定类型

ToUint16 转16位无符号整形

ToUint32 转32位无符号整形

ToUint64 转64位无符号整形

强制类型转换语法

数据类型 变量名 = （数据类型）变量名或值

Parse方法将字符串类型转任意类型

数据类型 变量名 = 变量二数据类型.Parse(字符串类型的值)

案例

Int num = int.Parse(Console.ReadLine());

Convert方法，将任意数据类型转为任意数据类型

数据类型 变量名 = convert.To数据类型(变量名);

Convert常用类方法

Convert.ToInt16() 转整形short

Convert.ToInt32() 转整形int

Convert.ToInt64() 转整形long

Convert.ToChar() 转字符

Convert.ToString() 转字符串

Convert.ToDateTime() 转日期

Convert.ToSingle() 转单精度

Convert.ToDouble()转双精度

ToString（）方法将任意类型转字符串

变量.ToString();

装箱和拆箱

在 C# 语言中数据类型分为值类型和引用类型，将值类型转换为引用类型的操作称为装箱，相应地将引用类型转换成值类型称为拆箱

C#数组

一维数组

//定义数组

数据类型[] 数组名;

//初始化数组元素

数据类型[] 数组名 = new 数据类型[长度]；

数据类型[] 数组名 = {值1，值2，值3……….};

数据类型[] 数组名· = new 数据类型[长度]{值1，值2，……};

//数组长度属性

Length

多维数组

//定义多维数组

数据类型[ , , …] 数组名；

//创建多维数组并初始化

数据类型[ , , …] 数组名 = new 数据类型[ m , n , …] { { , , …} , { , , …} … }

案例

double[ , ] points = { {10 , 20} , {20, 30} };

Console.WriteLine( points[0,0]);

遍历多维数组时使用GetLength(维度) ，获取数组中每一维的元素

//定义锯齿型数组 ，即多维数组每一维存放值的个数不同

数据类型[][] 数组名 = new 数据类型[数组长度][]；

数组名[0] = new 数据类型[数组长度];

……

数组中常用方法

Clear() 清空数组元素

Sort() 从小到大冒泡排序

Reverse() 逆序排列

IndexOf() 查找元素首次出现位置，没找到返回-1

LastIndexOf() 查找最后一次出现位置

C#foreach

Foreach语句仅能用于数组，字符串或集合数据类型

foreach(数据类型 变量名 in 数组名){

}

C#Split

拆分字符串，返回字符串数组

字符串.Split( condition , StringSplitOption.Node );

参数说明

Condition ： 拆分的条件数组，可存放多个拆分条件

StringSplitOption.Node : 拆分的选项， 表示遇到空字符也拆分出一个元素，若不需要空字符，改成 StringSplitOption.RemoveEmptyEntries

C#枚举类型

注意 ： 枚举类型不能定义到方法中 , 通过枚举变量名获得的值是整数类型

访问修饰符 enum 变量名 ：数据类型{

枚举值1， //默认0开始

枚举值2=4， //设置初始值

…

}

调用

类名.枚举变量名.枚举值

枚举变量名.枚举值

1.将枚举类型装换为int类型

枚举类型名 变量名 = 枚举类型名.枚举值

int 整数变量名 = （int）枚举变量名.枚举值

案例：

QQ q = QQ.name;

int n = (int)q; 或者 int n = (int)QQ.name;+

1. 将int类型装换枚举类型

枚举类型名 变量名 = （枚举类型名）整数变量

案例

int n = 1;

QQ q = (QQ)n; 如果整数不存在于枚举类型，则直接输出整数，不会报错

1. 枚举类型转string类型

string s = 枚举变量名.ToString();

1. string类型转枚举类型

使用函数 ， 将string类型转换成枚举类型

Object Enum.Parse(Type enumType , string value);

enumType : 枚举类型 , 使用 typeof(枚举类型名) 传入

案例

QQ q = (QQ)Enum.Parse(typeof(QQ) , s);

c#结构

可以一次性声明多个不同类型的变量

语法

访问修饰符 struct 结构名{

成员；

}

案例

public struct User{

public string \_name; //这里声明成员名的规范是前面加下划线

public int \_age;

public ArrayList \_score;

}

//调用

User myuser; //这是声明

myuser.\_name = “hah”; //这是赋值

myuser.\_name; //这是获取

C#Equals方法

比较两个对象是否相等

Equals( object o1 , object o2 );

待比较对象变量名.Equals( object o );

C#GetHashCode

返回对象hash值

对象变量名.GetHashCode()

C#GetType

返回System.Type 类型

对象变量名.GetType()

C#ToString

返回对象实例字符串，默认返回类型限定名，可以重写ToString方法返回自定义字符串

对象变量名.ToString()

C#base

调用父类成员 ， this表示当前类，base表示父类

base.父类成员

c#virtual

虚拟关键字， 将成员定义为虚拟，表示成员将会在继承后重写其中的内容 ， 不能与static共用

Public virtual 数据类型 属性名{set;get;}

访问修饰符 virtual 返回值类型 方法名{

方法体；

}

C#override

重写

访问修饰符 override 返回值类型 方法名(){

方法体;

}

C#abstract

抽象类

访问修饰符 abstract class 类名{

}

抽象方法

访问修饰符 abstract 返回值类型 方法名(参数列表);

C#sealed

密封修饰符。如果希望类或方法不希望被继承或重写，使用sealed关键字，和override一起使用，可用于类或方法

C#继承关系中的构造方法

实例化无参数子类，首先调用父类无参构造函数，在调用子类无参构造函数

实例化有参子类。首先调用父类无参构造函数，在调用子类有参构造函数

·· 如果希望子类调用父类有参构造函数，使用 : base(参数)的形式

public Bclass (string name) : base(参数){

方法体；

}

当子类同名方法要覆盖父类同名方法时，需要在方法声明上使用new关键字，声明这个方法覆盖了父类的同名方法,这样做的后果是，子类对象调用不到父类的同名方法

案例

public new void Test(){

}

C#里氏转换

多态条件

1. 子类继承父类必须有重写的方法
2. 调用父类方法，必须创建父类对象指向子类
3. 两个关键字

is if（p is Student） 用于判断变量是否属于某个类的

as Teacher t = p as Teacher 将p转换为Teacher，转换失败返回null

里氏转换和多态的区别

1. 里氏替换原则是针对继承而言的，如果继承是为了实现代码重用，也就是为了共享方法，那么共享的父类方法就应该保持不变，不能被子类重新定义  
。子类只能通过新添加方法来扩展功能，父类和子类都可以实例化，而子类继承的方法和父类是一样的，父类调用方法的地方，子类也可以调用同一个继承得来的，逻辑和父类一致的方法，这时用子类对象将父类对象替换掉时，当然逻辑一致，相安无事。

2. 如果继承的目的是为了多态，而多态的前提就是子类覆盖并重新定义父类的方法，为了符合LSP，我们应该将父类定义为抽象类，并定义抽象方法，让子类重新定义这些方法，当父类是抽象类时，父类就是不能实例化，所以也不存在可实例化的父类对象在程序里。也就不存在子类替换父类实例（根本不存在父类实例了）时逻辑不一致的可能。

不符合LSP的最常见的情况是，父类和子类都是可实例化的非抽象类，且父类的方法被子类重新定义，这一类的实现继承会造成父类和子类间的强耦合，也就是实际上并不相关的属性和方法牵强附会在一起，不利于程序扩展和维护。

C#接口

一个类可同时实现多个接口，同时能继承其他类，接口之间也能继承,接口也能多态

interface 接口名称{

接口成员；

}

实现接口

class 类名：接口名{

//

}

显式实现接口

interface class p{

Void fun();

}

public class ch ：p{

void fun(){ //自身方法

//qqq

}

void p.fun(){ //不能添加 显式方法

//asd

}

}

p pp = new ch(); pp.fun(); //创建接口对象就调用显式方法

ch c = new ch(); c.fun(); //创建自身对象调用自身方法

C#ArrayList集合

构造方法

ArrayList（） 集合的容量是默认初始容量

ArrayList（ICollection c ） 包含从指定实例中复制的元素，且初始容量与复制的元素个数相同

ArrayList（int capacity） 设置初始容量

初始化集合，添加默认元素

ArrayList a = new ArrayList（） {值1，值2，值3………}

ArrayList属性

Capacity 获取或设置集合中可以包含的元素个数，就是容量

Count 实际包含的元素个数

ArrayList方法

int Add(object value) 添加元素，返回下标

void AddRange（ICollection c） 添加集合

void Clear（） 清空元素

bool Contains（object item） 是否包含元素

void CopyTo（Array array） 复制集合中元素到类型兼容的数组array里

void CopyTo（Array array ， int arrayIndex） 从指定数组array中的指定下标arrayIndex开始，复制集合中元素到类型兼容的数组array里

void CopyTo（int index ， Array array， int arrayIndex ， int count） 指定array的指定下标arrayIndex开始，将集合指定索引index开始的count个元素复制到array里

int IndexOf（object value） 查找首次出现位置

int IndexOF（object value， int startIndex） 从指定位置开始首次出现的位置

int IndexOf（object value， int startIndex ， int count） 从指定位置开始的count个元素中首次出现的位置

int LastIndexOf（object value）

int LastIndexOf（object value， int startIndex）

int LastIndexOf（object value， int startIndex ， int count）

void Insert（int index ， object value） 指定索引处插入值

void InsertRange（int index ， ICollection c） 指定索引处插入集合

void Remove（object obj） 移除元素

void Remove（int index） 移除指定位置的元素

void RemoveRange（int index ， int count） 移除指定位置开始的count个元素

void Reverse（） 元素顺序反转

void Reverse （int index ， int count ） 指定位置开始的count个元素反转

void Sort（） 元素从小到大排列

void Sort（IComparer comparer） 按照比较器排列

void Sort（int index ， int count ， IComparer comparer） 设置范围，按照比较器排列

void TrimToSize（） 集合大小设置为集合中实际元素个数

比较器Comparer

class mycom ： IComparer{

public int Compare（object x ， object y）{

…….

return 结果；

}

}

c#Dictionary集合

Dictionary<K,V> dic = new Dictionary<K,V>();

C#Queue队列

增长因子：当需要扩大容量时，用当前容量capacity乘以增长因子factor来自动加容量

构造方法

Queue（） 默认容量32个元素，使用默认增长因子

Queue（ICollection c） 复制指定元素，且初始容量与复制的元素个数，增长因子相同

Queue（int capacity） 设置指定元素个数

Queue（int capacity ， float growFactor） 指定元素个数和增长因子

属性

Count 实际元素个数

方法

Void Clear（） 清空元素

Bool Contains（object obj） 是否包含元素

Void CopyTo（Array array ， int index） 将array数组从index索引处开始复制带队列中

Object Dequeue（） 移除并返回首个元素

Void Enqueue（） 尾部添加元素

Object Peek（） 返回头部元素但不删除

Object[] ToArray（） 复制队列元素到数组

Void TrimToSize（） 设置容量为Queue中实际的元素个数

IEnumerator GetEnumerator（） 返回循环访问队列的枚举数

C#Stack

栈是先进后出的结构

构造方法

Stack（） 初始容量创建对象

Stack（ICollection c）复制元素，且初始容量和增长因子与复制元素相同

Stack（int capacity） 设置初始容量

属性

Count 实际元素个数

方法

Push（object obj） 添加元素

Peek（）获取顶部元素但不移除

Pop（）获取并移除顶部元素

Clear（）清空元素

Contains（object obj） 是否包含元素

Object[] ToArray（） 复制元素到数组

C#Hashtable

哈希表，使用键值对存放数据，无序列表

Hashtable 对象名 = new Hashtable（）；

属性

Count 实际元素个数

Keys 所有键的集合

方法

Void Add（object key ， object value） 添加元素

Void Remove（object key） 移除元素

Void Clear（） 清空集合

ContainsKey（object key）是否包含指定key元素

ContainsValue（object value） 是否包含指定value元素

C#SortedList

有序列表，按key值进行排序

属性和方法与Hashtable类似

C#遍历各类数据集合

1. 遍历枚举
   1. foreach( string sp in Enum.GetNames( typeof(Sample) ) ){

//获取枚举名称

}

* 1. foreach( string sp in Enum.GetValues( typeof(Sample) ) ){

//获取枚举值

}

·· 2.遍历ArrayList（Queue，Stack）

1. foreach( string str in arrayList){

}

3.遍历HashTable

DictionaryEntry类需要引用System.Collections

foreach( DictionaryEntry item in hashTable){

Console.WriteLine( “哈希键”+item.Key+”哈希值”+item.Value.ToString() );

}

//只遍历哈希键

foreach( string key in hashTable.Keys){}

//只遍历哈希值

foreach( string value in hashTable.Values ){}

1. 遍历Dictionary

foreach(KeyValuePair<int,string> item in dic）{

Console.WriteLine(item.Key);

Console.WriteLine(item.Value);

}

c#泛型

c#泛型是2.0新提供的功能，泛型是在System.Collections.Generic命名空间中的，用于约束类或方法中的参数类型

例如集合可以添加任意类型的值，但是遍历集合时如果规定的值得类型，就可能报System.InvalidCastException异常，即指定的转换无效，所以限定值类型，为泛型的作用之一

1. 泛型--可空类型

System.Nullable<T> 变量名;

其中Nullable为所在命名空间，T代表任意类型，System可以省略

例如

Nullable<int> a;

Nullable<int> a = null;

换种写法

int? a; int? 等同于Nullable<int>

可空类型也能通过HasValue属性判断是否为null值

可空类型允许将值类型变量的值设置为null，并可以通过HasValue属性判断是否为null

1. 泛型--泛型方法

通过泛型来约束方法中的参数类型，也可理解为对数据类型设置了参数

如果没有泛型，方法的参数类型是固定的，不能更改

使用泛型后，方法数据类型由指定泛型约束，可根据泛型传递不同类型的参数

语法

public void 方法名<T> (T 变量1 ， T 变量2 , ......){

方法体

}

案例

class Program{

Static void Main(string[] args){

Add<double>(3.3 , 4);

Add<int> (100,1);

}

private static void Add<T> (T a , T b){

double s = double.Parse(a.ToString())+double.Parse(b.ToString());

}

}

1. 泛型--泛型类

泛型类与泛型方法类似，在泛型类名称后面加上<T>

class 类名<T1,T2,......>{

}

案例

class MyTest<T>{

public T[] arr = new T[3];

public void Show(){

foreach( T t in arr){

}

}

}

1. 泛型--泛型集合

由于集合能存放任意类型的值，所以经常遇到数据类型转换异常的情况，因此推荐定义集合时使用泛型集合

ArrayList和HashTable在泛型集合中分别使用List<T>和Dictionary<K,V>表示

C#IComparable和IComparer

IComparable和IComparer用于比较集合中的对象值，主要用于集合中的元素排序

IComparer用于单独的类，用于比较两个对象，泛型IComparer<T>

IComprable用于要比较的对象类，用于比较两个对象，泛型IComparable<T>

1, ICompare<T>接口

方法

CompareTo（T obj） 比较两个对象值

注意：CompareTo方法返回大于0表示第一个对象值大于第二个对象值，小于0表示小于，等于0表示相等

案例

class Student : IComparable<Student>{

public Student(int age){

this.age = age;

}

pubic age {set; get;}

public CompareTo(Student other){

If( this.age > other.age){return -1;}

Return 1;

}

}

在调用IComparable接口的比较方法时，直接 集合名.Sort（）就行

方法

Compare(T obj1 , T obj2) 比较两个对象值

在调用IComparer接口的比较方法时，需要 集合名.Sort( new 比较器() );

C#文件操作

1. DriveInfo类 查看计算机驱动信息

构造方法

DriveInfo（string driveName）

driveName指有效驱动器路径或驱动器号，Null值无效

属性

AvailableFreeSpace 只读属性，获取可用空闲空间量，单位字节

DriveFormat 只读属性，获取文件系统格式的名称，例如NTFS或FAT32

DriveType 只读属性，获取驱动器类型

IsReady 只读属性，驱动器是否准备好

Name 只读属性，驱动器名称

RootDirectory 只读属性，驱动器根目录

TotalFreeSpace 只读属性，可用空闲空间总量

TotalSize 只读属性，空间总大小

VolumeLabel 属性，获取或设置驱动器卷标

静态方法

DriveInfo[] GetDrives() 所有驱动器的名称

1. DirectoryInfo类 操作文件夹

构造方法

DirectoryInfo（string path）

属性

Exists 只读属性，目录是否存在

Name 只读属性，获取目录名称

Parent 只读属性，获取父目录

Root 只读属性，获取目录根部分

方法

void Create（） 创建目录

DirectoryInfo CreateSubdirectory（string path）创建一个或多个子目录

void Delete（） 如果目录为空，则将目录删除

void Delete（bool recursive）指定是否删除子目录和文件，如果recursive为true则删除，否则不删除

IEnumerable<DirectoryInfo> EnumerateDirectories（）返回当前目录中目录信息的可枚举集合

IEnumerable<DirectoryInfo> EnumerateDirectories（string searchPattern） 返回与指定搜索规则匹配的目录信息可枚举集合

IEnumerable<FileInfo> EnumerateFiles() 返回当前目录中文件信息的可枚举集合

IEnumerable<FileInfo> EnumerateFiles（string searchPattern）返回与指定搜索模式匹配的文件信息的可枚举集合

IEnumerable<FileSystemInfo> EnumerateFileSystemInfos（） 返回当前目录信息中文件系统信息的可枚举集合

IEnumerable<FileSystemInfo> EnumerateFileSystemInfos（string searchPattern） 返回与指定搜索模式匹配的文件系统信息可枚举集合

DirectoryInfo[] GetDirectories（） 返回子目录

DirectoryInfo[] GetDirectories（string searchPattern） 返回匹配所搜条件的子目录

FileInfo[] GetFiles（） 返回文件列表

FileInfo[] GetFiles（string searchPattern） 返回符合规则的文件列表

FileSystemInfo[] GetFileSystemInfos（） 返回所有文件和目录的子目录中的项

FileSystemInfo[] GetFileStstemInfos（string searcgPattern） 返回匹配规则的所有文件和目录的子目录中的项

void MoveTo（string destDirName） 移动目录到新路径

1. Directory类

Directory类是一个静态类，不能创建实例 ， 其他操作与DirectoryInfo类似

1. FileInfo类 操作文件

构造方法

FileInfo（string fileName）

属性

Directory 只读属性，获取父目录实例

DirectoryName 只读属性，获取表示目录的完整路径的字符串

Exists 只读属性，文件是否存在

IsReadyOnly 属性，设置指定文件是否为可读

Length 获取文件大小

Name 获取文件名称

方法

FileInfo CopyTo（string destFileName）现有文件复制到新文件，不允许覆盖现有文件

FileInfo CopyTo（string destFileName，bool overWrite）复制到新文件，允许覆盖现有文件

FileStream Create（） 创建文件

void Delete（） 删除文件

void MoveTo（string destFileName）指定文件移动到新位置

FileInfo Replace（string destinationFileName，string destinationBackupFileName）使用当前文件对象替换指定的文件内容，先删除原始文件，再创建被替换文件的备份

1. File类

File类是静态类

方法

DateTime GetCreationTime（string path）返回指定文件或目录的创建日期和时间

DateTime GetLastAccessTime（string path）返回上次访问指定文件或目录的日期和时间

DateTime GetLastWriteTime（string path）返回上次写入指定文件或目录的日期和时间

void SetCreationTime（string path，DateTime CreationTime）设置创建改文件的日期和时间

void SetLastAccessTime（string path，DateTime lastAccessTime）

void SetLastWriteAccessTime（string path，DateTime lastWriteTime）

1. Path类

主要用于文件路径操作，也是静态类

方法

String ChangeExtension（string path，string extension）更改路径字符串的扩展名

String Combine（params string[] paths）将字符串数组组合成一个路径

String Combine（string path1，string path2）将两个字符串组合成一个路径

String GetDirectoryName（string path）返回指定路径字符串的目录信息

String GetExtension（string path）返回扩展名

String GetFileName（string path）返回文件名和扩展名

String GetFileNameWithoutExtension（string path）只返回文件名

String GetFullPath（string path）获取绝对路径

Char[] GetInvalidFileNameChars（）获取包含不允许在文件名中使用的字符的数组

Char[] GetInvalidPathChars（） 获取包含不允许在路径名中使用的字符的数组

String GetPathRoot（string path） 获取指定路径的根目录信息

String GetRandomFileName（） 返回随机文件夹名或文件名

String GetTempPath（） 返回当前用户临时文件夹路径

Bool HasExtension（string path） 返回路径是否包含文件的扩展名

Bool IsPathRooted（string path） 路径字符串是否包含根

1. StreamReader类 从流中读取字符串，操作字符的

构造方法

StreamReader（Stream stream） 给指定的流创建StreamReader实例

StreamReader（string path）指定路径文件创建实例

StreamReader（Stream stream，Encoding encoding）用指定字符编码为指定的流初始化一个实例

StreamReader（string path，Encoding encoding）用指定字符编码为指定的文件初始化一个实例

属性

Encoding CurrentEncoding 只读属性，获取当前流使用的编码方式

Bool EndOfStream 只读属性，获取当前流位置是否在流结尾

方法

Void Close（）关闭流

Int Peek（）获取流中的下一个字符的整数，如果没有获取到字符，返回-1

Int Read（）获取流中的下一个字符的整数

Int Read（char[] buffer，int index，int count）从指定的索引位置开始将来自当前流的指定的最多字符读到缓冲区

String ReadLine（）读取一行字符串，自动移到下一行

String ReadToEnd（）读取来自流的当前位置到结尾的所有字符

1. StreamWriter类 写入文件，操作字符的

构造方法

StreamWriter（Stream stream）指定流创建实例

StreamWriter（string path）指定路径文件创建实例

StreamWriter（Stream stream，Encoding encoding）指定字符编码的流创建实例

StreamWriter（string path，Encoding encoding）指定字符编码的文件创建实例

属性

Bool AutoFlush 设置或获取是否自动刷新缓冲区

Encoding encoding 只读属性，获取当前流编码方式

方法

Void Close（）关闭流

Void Flush（）刷新缓冲区

Void Write（char value）字符写入流中

Void WriteLine（char value）字符换行写入流中

Task WriteAsync（char value）将字符异步写入流中

Task WriteLineAsync（char value）将字符异步换行写入流中

1. FileStream类 用于文件的读写,操作字节的，

创建实例时的其他参数，参数均为枚举类型

FileAccess参数，设置文件的访问方式

Read 只读

Write 写方式

ReadWrite 读写方式

FileMode参数，设置文件的打开或创建方式

CreateNew 创建新文件，若文件已存在则抛出异常

Create 创建文件，若已存在则删除重建

Open 打开已存在文件，若不存在则抛出异常

OpenOrCreate 打开已存在文件，不存在就创建

Truncate 打开已存在文件，清除文件内容，保留文件创建日期，若不存在则抛出异常

Append 打开文件，向文件追加内容，若不存在，则新建

FileShare参数，主要用于设置多个对象同时访问一个文件时的访问控制

None 谢绝共享当前文件

Read 允许随后打开文件读取信息

ReadWrite 允许随后打开文件读写信息

Write 允许随后打开文件写入信息

Delete 允许随后删除文件

Inheritable 文件句柄可由子进程继承

FileAccess参数，用于确定 FileStream 对象访问文件的方式。

FileOptions参数，用于设置文件的高级选项，包括文件是否加密，访问后是否删除等

WriteThrough 指示系统应通过任何中间缓存，直接写入磁盘

None 指示在生成System.IO.File.Stream对象时不应使用其他选项

Encrypted 指示文件是加密的，只能通过用于加密的同一账户来解密

DeleteOnClose 指示当不在使用某个文件时自动删除该文件

SequentialScan 指示按从头到尾的顺序访问文件

RandomAccess 指示随机访问文件

Asyncchronous 指示文件可用于异步读取和写入

构造方法

FileStream（string path，FileMode mode） 指定文件，指定模式

FileStream（string path，FileMode mode，FileAccess access）

FileStream（string path，FileMode mode，FileAccess access，FileShare share）

FileStream（string path，FileMode mode，FileAccess access，FileShare share，int buffrsize，FileOptions options）

属性

Bool CanRead 只读属性，当前流是否支持读取

Bool CanSeek 只读属性，当前流是否支出查找

Bool CanWrite 只读属性，当前流是否支持写入

Bool IsAsync 只读属性，filestream是异步还是同步打开的

Long Length 只读属性，获取用字节表示的流长度

String Name 只读属性，获取传递给构造方法的FileStream的名称

Long Position 属性，获取或设置此流的当前位置

方法

Int Read（byte[] array，int offset，int count）从流中读取字节块并将该数据写入给定缓冲区中 Int ReadByte（）从文件中读取一个字节，并将读取位置提升一个字节，返回实际读取的字节数，当返回0时意味着文件读完了

Long Seek（long offset，SeekOrigin origin）将该流的当前位置设置为给定值

Void Unlock（long position，long length）允许其他进程访问以前锁定的某个文件的全部或部分

Void Lock（long position，long length）防止其他进程读取或写入System.IO.FileStream

Void Write（byte[] array，int offset，int count）将字节块写入文件流

Void WriteByte（byte value）将一个字节写入文件流中的当前位置

注意：如果字符串是中文，byte数组长度不够用，使用 byte[] bytes = Encoding.UTF8.GetBytes(message); 将数据从字符串类型转换为字节类型

Void Close() 关闭流

Void Dispose() 释放占用的资源

将创建文件流对象的过程写在using中， 会自动释放流所占用的资源

using(创建文件流){

读取或写入的过程

}

案例

using(FileStream fw = new FileStream(@“路径” , FileMode.OpenOrCreate , FileShare.Write) ){

string str = “asdasdsadsd”;

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

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

}

1. BinaryReader类 二进制形式读取数据

构造方法

BinaryReader（Stream input） input是输入流

BinaryReader（Stream input，Encoding encoding）encoding是编码方式

BinaryReader（Stream input，Encoding encoding，bool leaveOpen）leaveOpen是指在流读取后是否包括流的打开状态

方法

Int Read（） 从指定的流中读取字符

Int Read（byte[] buffer，int 0index，int count）从字节数组中index索引开始读取count个字节

Int Read（char[] buffer，int index，int count） 从字符数组中index索引开始读取count个字符

Bool ReadBoolean（）从当前流读取bool值，并使该流的当前位置提升一个字节

Byte ReadByte（） 当前流读取下一个字节，当前流位置提升一个字节

Byte[] ReadBytes（int count） 当前流读取指定字节数写入字节数组，并把位置前移相应字节数

Char ReadChar（） 读取下一个字符，前移相应的字符位

Char[] ReadChar（int count）读取指定数量字符，位置前移相应字符位

Void FillBuffer（int numBytes）用从流中读取的指定字节数填充内部缓冲区

1. BinaryWriter类 向流中写入内容

构造方法和BinaryReader类似

方法

Void Close（） 关闭流

Void Flush（） 清理当前编写器的所有缓冲区，使所有缓冲数据写入基础设备

Long Seek（int offset，SeekOrigin origin） 返回查找的当前流的位置

Void Write（char[] chars） 将字符数组写入当前流

Write7BitEncodedint（int value） 以压缩格式写出32位整数

c#委托

命名委托

修饰符 delegate 返回值类型 委托名(参数列表);

实例化委托,可以是静态方法，也可以是实例方法

委托名 委托对象名 = new 委托名（方法名）;

委托名 委托对象名 = 方法名;

调用委托

委托对象名（参数列表）;

多播委托：在一个委托中注册多个方法，注册方法时使用+或-运算符实现添加或撤销方法

委托名 委托对象名 += 方法名;

委托名 委托对象名 -= 方法名;

匿名委托：使用匿名方法注册到委托上

委托名 委托对象 = delegate{

//代码块

}；

事件 ： 是一种引用类型，也是一种特殊委托

定义事件

访问修饰符 event 委托名 事件名；

c#进程与线程

进程类 Process 该类的命名空间是System.Diagnostics

属性

MachineName 获取进程正在其上运行的计算机的名称

Id 进程唯一标识符

ExitTime 进程退出的时间

ProcessName 进程名称

StartTime 进程启动时间

Threads 获取进程中运行的一组线程

TotalProcessorTime 进程总的处理器时间

UserProcessorTime 此进程的用户处理器时间

方法

Close（） 释放与此组件关联的所有资源

CloseMainWindow（） 通过向进程的主窗口发送关闭消息来关闭拥有用户界面的进程

Dispose（） 释放由Component使用的所有资源

GetCurrentProcess（） 获取新的Process组件，并将其与当前活动的进程关联

GetProcesses（） 为本地计算机上的每个进程资源创建一个新的Process组件

GetProcessesByName（string） 创建新的Process组件的数组，并将它们与本地计算机上共享指定的进程名称的所有进程资源关联

Kill（） 停止关联的进程

Start（） 启用此process组件的startinfo属性指定的进程资源，并将其与该组件关联

Start（string）起动指定程序

线程Thread类

主要用于实现线程的创建以及执行，在命名空间System.Threading中

主要使用的类

Thread 在初始的应用程序中创建其他的线程

ThreadState 指定Thread的执行状态，包括开始，运行，挂起等

ThreadPriority 线程在调度时的优先级枚举值，包括Highest，AboveNormal，Normal，BelowNormal，Lowest

ThreadPool 线程池，用于执行任务，发送工作项，处理异步IO等操作

Monitor 提供同步访问对象的机制

Mutex 用于线程间同步的操作

ThreadAbortException Thread处于对方法调用无效的ThreadState时出现的异常

Thread类属性

Name 获取或设置线程的名称

Priority 获取或设置线程优先级

ThreadState 获取线程当前的状态

IsAlive 获取当前线程是否处于启动状态

IsBackground 获取或设置值，表示该线程是否为后台线程

CurrentThread 获取当前正在运行的线程

Thread类方法

Start（） 启动线程

Sleep（int time） 当前线程暂停指定毫秒数

Join（） 阻塞调用线程，直到某个线程终止

Interrpt（） 中断当前线程

Abort（） 终止线程

Winform

属性：

name 在后台要获得前台控制对象，需要使用name属性

visible 控件是否可见

enabled 控件是否可用

事件

注册事件

触发事件

Main函数创建的窗体对象，称为主窗体，主窗体关闭，整个程序关闭