# C#

## grammar

### 命名空间

可以通过using重定义类型名

using NewMsgSizeType=UInt32;

可以在命名空间A内嵌套命名空间B，相当于在A内定义一个变量B，所以A和B不能互相访问元素。

命名空间内只能包含类，枚举，结构体，委托等，不能包含变量和函数。

默认情况下存在一个全局命名空间，所以在命名空间外定义的类可以直接进入到此全局命名空间。在全局命名空间中包含的类内写static变量，就相当于全局变量。

### 基本数据类型

#### 基本类型

|  |  |  |
| --- | --- | --- |
| 类型 | 字节数 | 描述 |
| Byte | 1 | unsigned byte |
| Sbyte | 1 | signed byte |
| Short | 2 | signed short |
| Ushort | 2 | unsigned short |
| Int | 4 | signed integer |
| Uint | 4 | unsigned integer |
| Long | 8 | signed long |
| Ulong | 8 | unsigned long |
| Float | 4 | floating point number |
| Double | 8 | double precision number |
| Decimal | 8 | fixed precision number |
| String |  | Unicode string |
| Char | 2 | Unicode char |
| Bool |  | true，false，boolean |

#### 可空类型

int? x = 5; //x可以赋值为null

与非可空类型运算时，需要显示转换

int y = (int)x\* 2;或者int y = x.Value \* 2;

??运算

允许给值可能为null的表达式赋另外一个值

x??y 等价x==null?y:x

使用时如：

int? x = null;

int result = x \* 2 ?? 5;

结果为5

#### Var

可以指代任何类型的变量

var x="\*\*\*";

var a = new A();

### 类

class类型必须用new，在函数中传递时是引用传递（可以不用ref，但在函数中不能改变class对象的指向）。

#### 静态成员

因为C#没有全局变量和函数。Static成员不属于对象实例，所以可以在不实例化类的情况下直接调用。

C#中，const变量也可以被static函数调用。

静态变量可以用来在类与类之间进行数据交互和共享。

class A{ public static int x=10;}

在其他任何地方，都可以使用A.x进行访问。

#### 属性

属性比字段（变量）包含的内容多，在修改状态时还可以进行额外的操作。

包含两个函数块（get和set），一个用于获取属性，一个用于设置属性。这两个块也称为访问器。

class Data

{

int month=10;//私有的，外部不能直接访问

public int Month //指代变量

{

get { return month;}

set { month = value;}

}

}

使用时

Data data = new Data();

data.Month = 20;//设置

int x=data.Month;//获取

自动属性

public int x { set; get; }

class Cu

{

public string Name { get; set; }

public string City { get; set; }

}

Cu c = new Cu() { Name = "ab", City = "12" };

#### 权限

只能修饰单个成员，不能用public:修饰下面的所有成员。

类可以用internal修饰，表示只能在当前项目访问。子类的可访问性不能高于父类（internal可以继承public，反之不可）。

可以用public，private，protected修饰

readonly

只能在初始化或者执行构造时赋值。仅用于修饰类的数据成员，和const不同之处在于const要求在声明时进行直接初始化，readonly是指一旦进行了写操作或者初始化，就不可以修改。

readonly修饰变量不能在函数中定义，只能修饰类的成员变量。

sealed

可以用来修饰类，该类不允许被继承。

#### 构造

class A

{

int x, y;

public A(int x,int y)

{ this.x = x; this.y = y;}

}

使用时：A a = new A(1, 3); //带参数的构造函数只能用new调用

A a; //该方式和无参构造无关，在类内定义无参构造没有意义，即使定义为private也可以。

#### 多态

父类使用virtual关键字，子类实现虚函数使用override关键字。

class B

{

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

}

class C : B

{

public override void fun(){Console.WriteLine("C");}

}

使用时：

B b=new C();

b.fun(); //输出C

和C++概念相同，父类指向子类，本质还是父类（忽略子类中除了父类以外的内容），子类只是用来改写父类的虚函数。

如果父类和子类有重名的函数或者变量，使用base在子类中引用父类的成员：base.fun();

#### 继承

如果没有指定继承的父类，那么就是默认只继承自System.Object。所以，每个类都是继承自System.Object。

子类中有和父类相同的成员，就会隐藏父类的成员。使用base关键字就可以访问被隐藏或者重写的父类成员,如base.fun()。

class A{A(int x){} }

class B : A {//没有protected和private继承

B():base(5){} //初始化参数列表中用base关键字指定父类构造中需要的参数

}

#### Abstract

主要用作对象系列的基类，共享某些主要特性。

抽象类不能直接实例化，也可以和接口一样声明函数，由派生类实现。

abstract class A

{

int x;

public void fun1(){Console.WriteLine("aa");}

public abstract void fun2();

}

class B : A

{

override public void fun2(){Console.WriteLine("xx");}

}

#### 释放对象

using(x){.......}

在using的代码块中使用变量x，代码块结束后，立即释放对象。

#### partial

使用partial关键字可以让一个类在不同的文件中定义

============文件1内

namespace N

{

partial class A

{

public void fun1() { Console.WriteLine("fun1"); }

partial void fun3();//这种只声明，不实现的函数必须是void，而且不能用public这种修饰。如果声明的这个函数没有被实现，编译器就会在编译时自动删除相关代码。

}

}

============文件2内

namespace N

{

partial class A

{

public void fun2() { Console.WriteLine("fun2"); fun3(); }

partial void fun3() { Console.WriteLine("fun3"); }

}

}

============使用时

using N;

A a = new A();

a.fun1();

a.fun2();

### Object

可以指代任何对象

void fun(object[] obj)

{

foreach(object o in obj)

Console.WriteLine(o);

}

fun(new object[]{1,3.3,"adf"});

### 接口

interface X1{ void fun1(); void fun2();}//声明接口

interface X2 { int a { get; } }

class X : X1, X2 //用多继承接口，并实现接口

{

void X1.fun1() {Console.WriteLine("fun1");}//显示实现

public void fun2(){Console.WriteLine("fun2");}//隐式实现

public int a { get;protected set; } //实现时必须要get，set同时存在，如果接口中不存在其中一个，就要添加。添加时修饰符必须要比接口中的严格（接口是public，所以此处可以用protected）

}

调用时

X x = new X();

X1 x1 = (X1)x; //将类转换为某一接口的内容

x1.fun1();

x.fun2();

### 委托

可以把函数引用保存在变量中，类似于函数指针。使用关键字delegate。

delegate void fun(int x);

void fun1(int x){}

void fun2(int x){}

//创建对象调用

fun ff = new fun(fun1);

ff(1);

//直接调用

fun ff = fun1;

ff(1);

//Lambda

fun ff = (int x) => { Console.WriteLine("{0}", x); };

ff(1);

//事件

fun ff = fun2;

ff += fun1;

ff += fun3;

ff -= fun1;

ff(1);

### Attribute

using System.Diagnostics;

添加额外信息

#### 内置attribute

函数只在debuge下有效

[Conditional("DEBUG")]

void fun(){}

[Obsolete("过时的方法",true)] 第二个参数为编译时是否报错

void fun(){}

#### 自定义attribute

[AttributeUsage(AttributeTargets.Class, AllowMultiple = false, Inherited = false)]

//代表该Attribute只能用在class上一行，不能连续两行出现该Attribute，类在继承时不继承该Attribute

class MyAttrbute : Attribute

{

public string info;

public MyAttrbute(string msg)

{

info = msg;

}

public int vale = 0;

}

使用时：

[MyAttrbute("xxxxx", vale = 11)]

class Test{}

通过反射获取信息：

foreach (var attr in typeof(Test).GetCustomAttributes(true))

{//获取Test类的所有Attribute，找到MyAttrbute进行处理

MyAttrbute myAt = attr as MyAttrbute;

if (null != myAt)

{

Console.WriteLine(myAt.vale);

}

}

### 函数

#### 引用传递

void fun(ref int x){....}

调用时(ref也是函数签名的一部分，所以调用时候也要写上。传入的参数必须有初始化[赋值])，class类型如果只是在函数中改变值，不改变指向，可以直接传递，不用ref。

int x=10;

fun(ref x);

#### 输出参数

类似于ref，不同处在于传入的变量可以没有初始化（赋值）。在函数中，要把传入的变量作为未初始化的来处理，也就是在函数中必须要进行初始化（赋值）。

void fun(out int x){x = 10;....}

int x;

fun(out x);

#### params

可以声明为不确定参数个数(数组类型)，该参数必须总是函数最右边的参数，且只有一个参数可以是数组类型。

下面的函数调用时可以使用：int max=fun(1,3,243,434,12);

int fun(params int[] a)

### 数据类型

#### 用户类型

包括类，结构体，枚举（和C++完全一样），接口。

除了内存分配，类和结构与C++完全相同。

类的对象在堆中分配，使用new创建（构造函数必须加上（）），当不再使用时，将自动进行垃圾清理。

结构是在栈中分配。属于轻量级快速数据类型。

#### 结构体

结构体是值类型，而类是引用类型。

结构体没有构造函数，也不能在定义变量时赋值。

struct data

{

public int a ;

public float b;

}

data d;

d.a=10;

d.b=1.1f;

#### 枚举

enum EE

{

吃饭,

睡觉,

}

int index = EE.吃饭.GetHashCode();

string val = EE.睡觉.ToString();

### 字符串

string s1 = "scca";//”scca”字面值本身就是string类型

string s2 = new string('c', 5);

char[] a = { 'a', 'b', 'c','\0'};//可以不加\0

string s3 = new string(a);

========@转译

适合地址的转换

如[@”d:\1.avi](mailto:如@)”

或者string addr=”d:\1.avi”; string str=@addr;

========转换为其他类型

int x = Convert.ToInt32("12");

========其他类型转换为字符串

int x=10;

string s=x.ToString();

========转换为字符串数组

char[] a = s.ToCharArray();

========去除开头和结尾多余的空格：

s=s.Trim();

========加入字符

string s = "abc";

s=s.PadLeft(5');//可以在s前面加入两个空格，因为s原长3，要补足到5，所以加入2个

s=s.PadLeft(5,'\_');//加入两个\_

========分割字符串

str = str.Substring(pos,length); //从pos截取length长度（无length代表到结尾）

string s = "abc|de|fg ";

string[] str = s.Split('|');//用str.Length可以求出数组长度

foreach (string i in str)

{

Console.WriteLine(i);

}

========字符串查找

bool b=str.Contains("xxx");

### 字符串格式化

string s = string.Format("{0:D3},{1:f2}", 10, 1.23234);

//:D3表示整数至少3位，不足高位补0

//:f2表示小数点后保留2位

:c 货币形式

:x 十六进制

:000.00 用0填充不足的位数，小数点后固定两位

### StringBuilder

StringBuilder sb = new StringBuilder();//构造函数可以传入初始化需要分配的内存大小

sb.AppendFormat("{0},{1}:", 1, 2);

sb.Append('\n');

sb.ToString();

### 数组

数组分配于堆中，是引用类型。

#### 定义

int[] a=new int[10];

int[] a =new int[5]{ 34, 12, 32, 43, 32}

int[] a={1,2,3,4,5};

二维数组

int[,] a = new int[5, 10];//或者int[,] a={ { 3, 4 }, { 3, 2 } };

a[1,2]=5;// 赋值

#### 遍历

数组可以用foreach循环

foreach (int x in array){...}

#### 拷贝

将数组a的内容拷贝到数组b，数组b接收拷贝的位置是从b[2]开始

a.CopyTo(b, 2);

#### 功能函数

int len = a.GetLength(0);//得到数组中元素个数，0代表这是一维数组

或者使用int len=a.Length;

int index = a.GetUpperBound(0);//得到数组最大下标数

#### Linq

int[] numbers = {3,5,2,5,12,5,36,75,42};

var qur = numbers.Where((n) => n % 2 == 0).OrderBy((n) => n);

foreach (var i in qur) Console.WriteLine(i + "");

int[] ret = qur.ToArray<int>();

### 预处理

#### region

当代码比较长，可以定义分组，这样就可以折叠这个区域

#region A

.................

#endregion

#### define

#define De //在文件最开头定义一个标志

#undef De 可以进行取消定义

#### 条件判断

如果有定义则代表会进行编译，否则会隐藏

#if De

Console.WriteLine("de");

#endif

#### 编译消息

在文件中定义后，编译文件会弹出警告或者错误

#warning xxx

#error xxx

可以修改在编译时弹出的行号，字符串可以修改当前提示的文件名

#line 100 "flileName"

#### Pragma

#pragma warning disable 123

#pragma warning restore 123

### 修饰符

#### unsafe

修饰在C#中定义的不安全的上下文，如C++的指针等。如：

public unsafe fun(int \*p){ }

unsafe

{

.........

}

属性-》生成-》允许不安全的代码

#### fixed

在不安全代码中固定内存位置

fixed (double\* pa = &a[0],pb = &b[0]){}

#### checked

checked（….）可以检查表达式是否有溢出，如果有溢出，则程序会崩溃。

#### lock

类似于互斥锁。

将语句块标记为临界区，方法是获取给定对象的互斥锁，执行语句，然后释放该锁，lock确保当一个线程位于代码的临界区时，另一个线程不进入临界区。如果其他线程试图进入锁定的代码，将一直等待，直到该对象被释放。

object locker = new object();

lock(locker){ ......... }

#### case

可以用常数变量作为判断值。必须有break跳转，不支持执行完一个case执行别的case。

可以使用以下方法执行多个case

case 2:

case 3: Console.WriteLine("3"); break;

#### async

async void AsyncMethod()

{

string result = await MyMethod();

Console.WriteLine(result);

}

async Task<string> MyMethod()

{

string ret = await Task.Run(() => {

Console.WriteLine("aaaaaa");

Thread.Sleep(4000);

Console.WriteLine("bbbbbb");

return "--------";

});

return ret;

}

### 转换类型

#### is

用于检查操作数类型是否相等或可以转换，使用两个操作数，其结果是布尔值。适合于多态的情形。

if(a is ClassA){......................}

#### as

检查操作数的类型是否可以转换或者相等（是由is完成的），如果是，则处理结果是已转换或已装箱的对象。如果否，则返回null。

ClassA a=new ClassA();

ClassB b=a as ClassB; //返回值null，不可转换

ClassC c=a;

ClassA a2=c as ClassA; //将进行转换

#### 重载转换

============定义

class A

{

public int val=1;

public static implicit operator B(A a) //隐式转换

{

B b = new B();

b.val = a.val;

return b;

}

}

class B

{

public long val=1000;

public static explicit operator A(B b)//显示转换

{

A a = new A();

checked { a.val = (int)b.val; }

return a;

}

}

============使用

A a = new A();

B b = a;

B b = new B();

A a = (A)b;

### 装箱/拆箱

所有的数据类型都是从System命名空间的基类Object继承，所以基础或是原始的类型打包为一个对象，称为装箱。相反的处理称为拆箱。

int a1 = 10; object obj = a1; //装箱

int a2 = (int)obj;//拆箱

一个int值可以被转换为对象，并且能再次转换成int。

当某种值类型的变量需要被转换为一个引用类型时，便会产生一个对象箱保存该值，拆箱则完全相反。当某个对象箱被转换会其原值类型时，该值从箱中拷贝至适当的存储空间。

因为是装箱到obj，所以可以做一个List<object>，在读取的时候，通过使用is判断类型再拆箱(或使用as)，实现不同类型放在同一个list中。

### 运算符重载

class A

{

public int x =10;

public static A operator+(A a1, A a2)

{

A a = new A();

a.x = a1.x + a2.x;

return a;

}

}

可以重载的运算符

一元：+，-，！，~，++，true，false

二元：+，-，\*，/,%,&,|,^,<<,>>

比较：==，！=，<,>,<=,>=

重载bool运算符就可以用 if(a1){}

### 初始化器

class A

{

public string x { get; set; }

public int y { get; set; }

}

A a = new A { x="a", y=1 };//这种方法可以不使用其他构造函数或者对象来初始化成员

List<A> l=new List<A>{new A{x="a",y=1},new A{x="b",y=2}};//可以替代l.add以及赋值的语法

### 异常

class MyExp:Exception

{

public MyExp(string \_msg) : base(\_msg) { }

}

void Fun()

{

try

{

double y = Convert.ToDouble("xx");

Console.WriteLine(y);

}catch(Exception e)

{

throw new MyExp("err");

}

}

try

{

Fun();

}

catch (MyExp e)

{

Console.WriteLine(e.Message);

}

catch (Exception e)

{//必须写最后

Console.WriteLine(e.ToString());

}

### 索引器

使类可以通过[]语法访问和赋值。

class Test

{

string Content1 = "aaa";

string Content2 = "bbb";

public string this[int index]

{

get

{

if(index == 1) return Content1;

if (index == 2) return Content2;

return "erro";

}

set { }

}

}

Test test = new Test();

Console.WriteLine(test[1]);

### 类型获取

string s = "";

Type t1 = s.GetType();

Type t2 = Type.GetType("system.string", false, true);

Type t3 = typeof(string);

### 反射

#### 类信息

获取类中包含的可以访问的方法

Type t = typeof(Cl);

MethodInfo[] mi = t.GetMethods();

foreach (MethodInfo info in mi)

Console.WriteLine(info.Name);

//获取单个方法

MethodInfo mi = t.GetMethod("fun1");

#### 程序集

获取程序集中包含的类

//Assembly obj = Assembly.Load("");

Assembly obj = Assembly.GetExecutingAssembly();

Type[] types = obj.GetTypes();

foreach (Type t in types) Console.WriteLine(t.Name);

#### 加载使用

加载当前程序集中的Cl类，void fun1(int i,string s)函数

Assembly ass = Assembly.GetExecutingAssembly();

Type t = ass.GetType("ConsoleApplication1.Cl", false, true);

if (null == t) return;

object obj = Activator.CreateInstance(t);//创建类的实例

MethodInfo mi = t.GetMethod("fun1");

mi.Invoke(obj,new object[]{1,"a"});

## file

### 命名空间

using System.IO;

### 获取当前路径

string path = System.AppDomain.CurrentDomain.BaseDirectory;

获取的路径末尾存在一个\

### File

静态类

#### 文件复制

File.Copy(orgFile, newName);

#### 文件否存在

bool b=File.Exists(Path);

#### 文件删除

File.Delete(Path);

### Directory

静态类

#### 目录是否存在

Directory.Exists(Path);

#### 删除目录

如果使用第二个参数，则只删除非空目录

Directory.Delete(Path,true);

#### 创建目录

可以创建多层次目录

Directory.CreateDirectory(newName);

### FileInfo

FileInfo fin = new FileInfo(Path);

#### 文件是否存在

fi.Exists

#### 创建文件

FileStream fs = fi.Create();

#### 时间获取

string modiytime =

fin.LastWriteTime.ToLongDateString() + " " + fin.LastWriteTime.ToLongTimeString();

#### 删除文件

fin.Delete();

#### 获取文件流

FileStream fs = fi.OpenRead()

### DirectoryInfo

string dir = @"D:\";

@可以将\转换为\\

DirectoryInfo TheFolder = new DirectoryInfo(dir);

属性

TheFolder.Name

TheFolder.FullName

//遍历dir中所有目录 不包含点目录

foreach (DirectoryInfo NextFolder in TheFolder.GetDirectories())

///遍历dir中所有文件

foreach (FileInfo NextFile in TheFolder.GetFiles())

///遍历文件夹下所有jpg文件

foreach (FileInfo NextFile in TheFolder.GetFiles("\*.jpg"))

//遍历目录及子目录下所有txt

void AddPath(string path, List<string> PathNameList)

{

DirectoryInfo TheFolder = new DirectoryInfo(path);

foreach (FileInfo NextFile in TheFolder.GetFiles("\*.txt"))

{

PathNameList.Add(NextFile.FullName);

}

foreach (DirectoryInfo NextFolder in TheFolder.GetDirectories())

{

AddPath(path + "\\" + NextFolder.Name, PathNameList);

}

}

### FileStream

使用try判断文件是否打开或创建成功

#### 读取

FileStream file = new FileStream("e:\\1.txt", FileMode.OpenOrCreate);

byte[] data = new byte[10];

file.Read(data, 0, 10);

string s = Encoding.UTF8.GetString(data);

#### 写入

FileStream file = new FileStream("e:\\1.txt", FileMode.OpenOrCreate);

string s = "abcdefg";

byte[] data =Encoding.UTF8.GetBytes(s);

file.Write(data, 0, data.Length);

#### 文件指针移动

file.Seek(2, SeekOrigin.Begin); //文件指针从头向后移动两个

file.Seek(-2, SeekOrigin.End);从末尾向前移动两个位置

### 文件流操作

#### 文本流

========逐行读取

using (StreamReader sr = new StreamReader(new FileStream("D:\\tt.txt", FileMode.Open), Encoding.Default))

{ string strline;

while((strline = sr.ReadLine() )!= null){}

}

============保存为指定编码

StreamWriter file = new StreamWriter("D:\\.txt", false,Encoding.GetEncoding("gbk"));

file.WriteLine(...);

============追加

追加文本

using (StreamWriter sw = File.AppendText("D:\\tt.txt"))

{

sw.WriteLine("\nxabc");

}

追加字符串，并在一个字符串后加\n

string[] val = new string[2];

val[0] = "aaa";

val[1] = "bbb";

File.AppendAllLines("D:\\1.txt", val, Encoding.Default);

#### 二进制流

构造函数传入文件流

BinaryReader br = new BinaryReader(new FileStream("D:\\tt.txt",FileMode.Open));

BinaryWriter bw = new BinaryWriter(new FileStream("D:\\tt.txt", FileMode.Create));

### 读写压缩数据

using System.IO;

using System.IO.Compression;

void SaveCompressdFile(string filename, string data)

{

FileStream fileStream = new FileStream(filename, FileMode.Create, FileAccess.Write);

GZipStream compresssionStream = new GZipStream(fileStream, CompressionMode.Compress);

StreamWriter writer = new StreamWriter(compresssionStream);

writer.Write(data);

writer.Close();

}

MemoryStream fs = new MemoryStream();

GZipStream compresssionStream = new GZipStream(fs, CompressionMode.Compress);

BinaryWriter writer = new BinaryWriter(compresssionStream);

writer.Write(data);

writer.Close();

byte[] dst = fs.GetBuffer();

string LoadCompressedFile(string filename)

{

FileStream fileStream = new FileStream(filename, FileMode.Open, FileAccess.Read);

GZipStream compressionstream = new GZipStream(fileStream, CompressionMode.Decompress);

StreamReader reader = new StreamReader(compressionstream);

string data = reader.ReadToEnd();

return data;

}

### 序列化数据

using System.IO;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

==============可以序列化的类

[Serializable]

class A

{

int x;

[NonSerialized]

string age;//不进行序列化

public A(int a)

{

x = a;

}

}

==============序列化

List<A> data = new List<A>();

data.Add(new A(1));

data.Add(new A(2));

FileStream file = new FileStream("e:\\x.bin", FileMode.Create, FileAccess.Write);

IFormatter serializer = new BinaryFormatter();

serializer.Serialize(file, data);

file.Close();

==============反序列化

FileStream file = new FileStream("e:\\x.bin", FileMode.Open, FileAccess.Read);

IFormatter serializer = new BinaryFormatter();

List<A> data = serializer.Deserialize(file) as List<A>;

file.Close();

### 监控文件系统

FileSystemWatcher watcher = new FileSystemWatcher();

watcher.Path = "E:\\"; //监控目录为e盘，不能写成e:\\

//watcher.Filter = "\*.txt"; //设置监控的文件类型

watcher.EnableRaisingEvents = true;

watcher.Deleted += new \*\*\*\*\*\*Tab键进行插入; //监控删除事件

### XML

#### 文件说明

不同于关系型数据库的行列方式，采用类似目录的展开方式（只能有一个根元素）

可以在其中定义任意的标记对,如<x a=”\*\*”>\*\*\*</x>

在vs中，可以将文件另存为别的格式。

=========xml文件声明

<?xml version="1.0" encoding="utf-8"?>

=========引用命名空间

<abc xmlns="\*\*\*\*\*\*\*\*\*\*\*\*\*"> //表示abc及其子元素属于名称空间，名称空间一般是一个地址。

<a></a>

<b></b>

</abc>

或者

<abc xmlns:x="urn:schemas-microsoft-com:xml-data">

<x:a></x:a>

<x:b></x:b>

</abc>

#### 读写

using System.Xml;

XmlDocument doc = new XmlDocument();

doc.Load("config.xml");

<config>

<PicWidth>500</PicWidth>

</config>

XmlNode rootNode = doc.SelectSingleNode("config");

===============访问某节点

string s = rootNode.SelectSingleNode("a").InnerText; //获取节点a内的值

//string s = rootNode.InnerText; //会将根节点中所有节点的的值合并为一个字符串

=============获取节点属性

rootNode.Attributes["属性名"].Value;

===============修改某节点

rootNode.SelectSingleNode("a").InnerText = "xx";

document.Save("D:\\x.xml");

===============删除某节点

rootNode.SelectSingleNode("a").RemoveAll();

document.Save("D:\\x.xml");

===============插入节点

XmlElement node = document.CreateElement("z");

rootNode.AppendChild(node);

rootNode.AppendChild(document.CreateComment("\*\*\*"));//插入注释

===============遍历根节点下的所有节点

if (rootNode.HasChildNodes)

{

foreach (XmlNode node in rootNode)

{

string str = node.InnerText;

}

}

## thread

using System.Threading;

### 运行线程

Thread thread = new Thread(fun);//线程函数，fun不能有参数

或者使用lambda传入函数：new Thread(() => {

Console.WriteLine("xx");

});

thread.IsBackground = true;//前台线程退出则后台线程也退出，如果使用默认的false，则前台线程退出后，程序也不会退出

thread.Start();//开始运行

### 结束线程

thread.Abort();

### 线程名称

获取和设置当前线程的名称

Thread.CurrentThread.Name

### 暂停当前线程

Thread.Sleep(1000);

### 互斥对象

Mutex mutex = new Mutex();

mutex.WaitOne();//调用多次进行多次上锁，需要多次解锁

.........保护代码........

mutex.ReleaseMutex();

### 事件

public ManualResetEvent manualEvent = new ManualResetEvent(false);//true代表终止状态，可以被使用,fasle代表非终止，不可被使用

线程代码（无限循环中，执行一次就暂停，等待信号）

manualEvent.WaitOne();

.................

manualEvent.Reset();

让线程代码运行

manualEvent.Set();

判断线程是否正在运行

manualEvent.WaitOne(0)//返回值为ture代表正在被使用

## Generics

//using System.Collections.Generic;

### 线型

#### List

======定义

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

可以直接用下标访问 list[3]

======遍历

foreach(string x in list)

======排序

class Dat

{

public int a = 0;

}

======使用lambda

ll.Sort((Dat x, Dat y) =>

{

return x.a - y.a;

});

======实现接口IComparable

（右键菜单，选择实现接口，可以自动生成代码）

基本类型已实现该接口

class Dat : IComparable

{

int a = 0;

int IComparable.CompareTo(Object obj)

{

Dat temp = (Dat)obj;

return a - temp.a;

}

}

ll.Sort();

=====只排序部分元素

实现接口IComparer

class Cmp : IComparer<Stu>

{

public int Compare(Stu x, Stu y)

{

return x.x - y.x;

}

}

从下标4的元素开始，一共排序3个

Cmp cmp = new Cmp();

l.Sort(4, 3, cmp);

==========求最大最小值

实现接口IComparable

Stu max = l.Max<Stu>();

Stu min = l.Min<Stu>();

========查找

//查找满足某种条件的元素，在函数中写查询条件

bool find(Stu st)

{

return st.x == 10;

}

Stu st = l.Find(find);

if (null != st){}

//也可以使用委托

Stu st = l.Find((Stu temp)=> temp.x == 10);

#### ArrayList

存储空间连续

可以存储不同类型元素

当每个元素存储类型相同时，与list用法相同

存储元素类型不同时：

const int capacity = 100;

ArrayList ll = new ArrayList(capacity);

ll.Add(1);

ll.Add(1.5);

ll.Add("abc");

ll.Add(new Dat(2));

Console.WriteLine(ll[0]);

Console.WriteLine(ll[1]);

Console.WriteLine(ll[2]);

Dat dd = (Dat)ll[3];

dd.Show();

### HashSet

可以对数据进行去重，不进行排序

非基本类型需要重载函数

class Dat

{

int a = 0;

public override bool Equals(object obj)

{

return a == ((Dat)obj).a;

}

public override int GetHashCode()

{

return a;

}

}

HashSet<Dat> set = new HashSet<Dat>();

set.Add(new Dat(0));

set.Add(new Dat(2));

set.Add(new Dat(3));

set.Add(new Dat(0));

### queue

Queue<int> que = new Queue<int>();

que.Enqueue(1);//向尾部加入元素

que.Enqueue(2);

que.Enqueue(2);

int x = que.Peek();//查看顶部元素

int y = que.Dequeue();//查看并取出顶部元素

### 键值对

#### Hashtable

可以存储不同元素，加入数据时，key不能相同，否则异常

const int capacity = 100;

Hashtable ht = new Hashtable(capacity);

ht.Add(0, 1);

ht.Add(1, new Dat(111));

ht.Add(2, 1.5);

Console.WriteLine(ht[0]);

((Dat)ht[1]).Show();

Console.WriteLine(ht[2]);

#### Dictionary

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

dir.Add("a", 1);

dir.Add("b", 2);

//通过键值查找

if (dir.ContainsKey("a")){int x = dir["a"];}

//遍历

foreach (string key in dir.Keys)

foreach(int value in dir.Values)

foreach (KeyValuePair<string, int> d in dir)

//排序遍历

foreach (var item in dic.OrderBy(dicItem=> dicItem.Key))

{//用key进行排序

MessageBox.Show(item.Value);

}

#### SortedDictionary

#### 排序

对key进行排序，key需要实现IComparable接口

struct Stu:IComparable

{

public int a;

public int b;

int IComparable.CompareTo(object obj)

{//从小到大排序

return a - ((Stu)obj).a;

}

}

#### 获取首部数据

using System.Linq;

string val = SortedDictionaryObj.Values.First<string>();

### 定义泛型

class A<S, T>

{

public S s;

public T t;

public void fun(){s = default(S);}//如果是引用类型，则赋值null，如果是int就赋值0。

}

A<int, string> a = new A<int, string>();

a.s = 10;

a.t = "xx";

### 泛型约束

class A<S,T>where S:classX S约束为classX或者继承自classX

class A<S,T>where S:struct S被约束为struct

class A<S,T>where S:T 把一个类型参数用作另一个的约束

class A<S,T>where S:classX,classyY

class A<S,T>where S:classX where T:classY

### 泛型继承

class B : A<int>

class B<T> : A<T>

将模板实例化后继承或继承模板的类型

### 模板泛型

delegate void Fun<T>(T t);

void fun(int x) {}

Fun<int> df = new Fun<int>(fun);

df(12);

## function

### 基本数据类型转换

可以对基本类型做任意转换

string s = "12";

int i = Convert.ToInt32(s,10);

第二个参数默认10，代表10进制

转为2位16进制字符串数字

int x = 10;

string val = x.ToString("X2");

### 字节操作

#### 字节数组拷贝

Array.Copy(sourceArray, sourceIndex, destinationArray, destinationIndex, length);

#### 字节流

MemoryStream ms = new MemoryStream();

预先分配指定大小new MemoryStream(1024\*1000);

ms.Position = 0;//再次写入时从起始位置开始

//将每次获取的字节放入字节流

byte[] data = new byte[10];

ms.Write(data, 0,len);//ms.Length获取当前字节流长度

=======将字节流转出

拷贝方式转出

byte[] result = ms.ToArray();

不进行拷贝

sendByte = ms.GetBuffer();

#### 转换成数字

UInt32 datSize = BitConverter.ToUInt32(recvBytes, POS\_DATA\_SIZE);

转float

BitConverter.ToSingle(bytes, pos);

#### 数字转成字节流

UInt32 datSize;

BitConverter.GetBytes(datSize);

#### 转换base64串

二进制数据转换为base64字符串

Convert.ToBase64String(Imgbytes);

#### 字符解码

byte[] data = new byte[1024];

string str = Encoding.Default.GetString(data);//代表系统编码

str = Encoding.UTF8.GetString(data);

#### 字符串转字节流

byte[] u = Encoding.ASCII.GetBytes(str);

### 图像操作

FileStream fs = new FileStream("D:\\1.jpg", FileMode.Open);

Bitmap bmp = new Bitmap(fs);//new Bitmap("D:\\1.jpg")

Image image = Image.FromFile(imgPathName);

image = Image.FromStream(Stream);

### 获取图像宽高

using (FileStream fs = new FileStream("D:\\1.tif", FileMode.Open, FileAccess.Read))

{

System.Drawing.Image image = System.Drawing.Image.FromStream(fs);

int width = image.Width;

int height = image.Height;

}

### 修改图像质量

public byte[] ChangeImageQuality(byte[] img)

{

MemoryStream ms = new MemoryStream(img);

byte[] buffer = null;

Bitmap bmp = null;

ImageCodecInfo ici = null;

System.Drawing.Imaging.Encoder ecd = null;

EncoderParameter ept = null;

EncoderParameters eptS = null;

try

{

bmp = new Bitmap(ms);

ici = this.getImageCoderInfo("image/jpeg");

ecd = System.Drawing.Imaging.Encoder.Quality;

eptS = new EncoderParameters(1);

ept = new EncoderParameter(ecd, 50L);

eptS.Param[0] = ept;

//bmp.Save("D:\\xxxx.jpg", ici, eptS);

MemoryStream stream = new MemoryStream();

bmp.Save(stream, ici, eptS);

buffer = new byte[stream.Length];

stream.Seek(0, SeekOrigin.Begin);

int len = stream.Read(buffer, 0, buffer.Length);

if (len != buffer.Length) buffer = null;

}

catch (Exception ex)

{

return buffer;

}

finally

{

bmp.Dispose();

ept.Dispose();

eptS.Dispose();

}

return buffer;

}

private ImageCodecInfo getImageCoderInfo(string coderType)

{

ImageCodecInfo[] iciS = ImageCodecInfo.GetImageEncoders();

ImageCodecInfo retIci = null;

foreach (ImageCodecInfo ici in iciS)

{

if (ici.MimeType.Equals(coderType))

retIci = ici;

}

return retIci;

}

### 定时器

System.Timers.Timer tt = new System.Timers.Timer(1000);

tt.Elapsed += \*\*\*\*\*

tt.Enabled = true;

### 类库

使用.NET中的任何类都是在使用外部程序集中的类，因为它们的处理方式是相同的。

创建类库项目，可以生成dll，让其他C#工程调用。

namespace ClassLibrary1 //该名称为生成的dll名

{

public class AA

{

public void fun(){}

}

}

在其他工程中，可以在解决方案中，右键引用，选择dll所在目录。

使用时：

using ClassLibrary1;

AA a = new AA();

a.fun();

### 调试信息

MessageBox.Show(string, "名称");

using System.Diagnostics;

Debug.WriteLine("xx");

### 时间

//using System.Data;

string s=DateTime.Now.ToString("yyyy-MM-dd-HH-mm-ss");

//设置一个3s的时间范围

TimeSpan span = new TimeSpan(0, 0, 3);

TimeSpan span = TimeSpan.FromSeconds(3);

int count = (int)span.TotalSeconds;//获取总的秒数

### 打开/关闭程序

using System.Diagnostics;

Process[] process = Process.GetProcessesByName("a");

if (process.GetLength(0) > 0)

{

process[0].CloseMainWindow();

process[0].Close();

}

else

{

Process.Start("a.exe");

}

### 打开文件/文件夹

Process.Start(DatSourcePathname);

### 启动程序

Process.Start(new ProcessStartInfo("python.exe", "aa.py"));

### 退出程序

Process.GetCurrentProcess().Kill();

### Socket

客户端发送，服务端接收

//using System.Text;

using System.Net.Sockets;

using System.Net;

using System.Threading;

#### 客户端

private Socket sokClient = null;

private string localIP = "127.0.0.1";//服务器地址

private int localPort = 5555;//端口

private string sendStr="";//发送内容

void StartSocket()

{

IPAddress address = IPAddress.Parse(localIP);

IPEndPoint endpoint = new IPEndPoint(address, localPort);//把ip和端口转化为IPEndPoint实例

sokClient = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);//创建一个Socket

sokClient.Connect(endpoint);//连接到服务器

byte[] bs = Encoding.ASCII.GetBytes(sendStr);

sokClient.Send(bs, bs.Length, 0);//发送

}

#### 服务端

private Thread threadSocket = null;

private Socket socketServer = null;

private string ip = "127.0.0.1";

private int port = 5555;

void Start()

{

try

{

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

IPAddress address = IPAddress.Parse(ip);

IPEndPoint endpoint = new IPEndPoint(address, port);

socketServer.Bind(endpoint);

threadSocket = new Thread(StartSocket);

threadSocket.IsBackground = true;

threadSocket.Start();

}

catch (Exception ex)

{

}

}

void StartSocket()

{

socketServer.Listen(1);

Socket temp = socketServer.Accept();

byte[] recvBytes = new byte[1024];

int revbytes = temp.Receive(recvBytes, recvBytes.Length, 0);

string recvStr = Encoding.ASCII.GetString(recvBytes, 0, revbytes);

}

调用时：

Start();

### web

#### GET请求

HttpWebRequest req = (HttpWebRequest)HttpWebRequest.Create(reqstStr);

req.Method = "GET";

req.Headers["diyKey"] = "val";//自定义报头中的key-val

using (WebResponse wr = req.GetResponse())

{

Stream responseStream = wr.GetResponseStream();

StreamReader streamReader = new StreamReader(responseStream, Encoding.UTF8);

string retString = streamReader.ReadToEnd();

}

#### POST请求

string param = string.Format("userName={0}&passWord={1}", user, password);

byte[] bs = Encoding.ASCII.GetBytes(param);

HttpWebRequest req = (HttpWebRequest)HttpWebRequest.Create(reqstStr);

req.Method = "POST";

req.KeepAlive = false;

req.ContentType = "application/x-www-form-urlencoded";

req.ContentLength = bs.Length;

using (Stream reqStream = req.GetRequestStream())

{

reqStream.Write(bs, 0, bs.Length);

}

using (WebResponse wr = req.GetResponse())

{

Stream responseStream = wr.GetResponseStream();

XmlDocument doc = new XmlDocument();

doc.Load(responseStream);

}

#### HTTP服务

class HttpSer

{

HttpListener httpPostRequest = new HttpListener();

public void Start()

{

httpPostRequest.Prefixes.Add("http://127.0.0.1:8080/abc/");

httpPostRequest.Start();

Thread ThrednHttpPostRequest = new Thread(httpPostRequestHandle);

ThrednHttpPostRequest.IsBackground = true;

ThrednHttpPostRequest.Start();

}

void httpPostRequestHandle()

{

while (true)

{

HttpListenerContext requestContext = httpPostRequest.GetContext();

Thread threadsub = new Thread(() =>

{

HttpListenerContext request = (HttpListenerContext)requestContext;

string requestMethod = request.Request.HttpMethod;

if (requestMethod == "GET")

{

string val = request.Request.QueryString["user"];

string requestKeyVal = request.Request.Headers["diyKey"];//获取自定义报头中的key-val

}

else

{

string requestDatType = request.Request.ContentType;

Stream SourceStream = request.Request.InputStream;

if (requestDatType == "text/plain")

{

StreamReader streamReader = new StreamReader(SourceStream, Encoding.UTF8);

string retString = streamReader.ReadToEnd();

}

else if(requestDatType == "image/jpeg")

{

int requestDatLen = (int)request.Request.ContentLength64;

BinaryReader br = new BinaryReader(SourceStream);

byte[] img = new byte[requestDatLen];

int revLen = 0;

while (revLen < requestDatLen)

{

int read = br.Read(img, revLen, requestDatLen - revLen);

revLen += read;

}

}

}

request.Response.StatusCode = 200;

request.Response.Headers.Add("Access-Control-Allow-Origin", "\*");

request.Response.ContentType = "application/json";

request.Response.ContentEncoding = Encoding.UTF8;

byte[] buffer = System.Text.Encoding.UTF8.GetBytes("success");

request.Response.ContentLength64 = buffer.Length;

var output = request.Response.OutputStream;

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

output.Close();

});

threadsub.IsBackground = true;

threadsub.Start();

}

}

}

### 发邮件

public void SendMail()

{

System.Net.Mail.MailMessage msg = new System.Net.Mail.MailMessage();

msg.To.Add("jiyanglin@roadmaint.com");

// msg.To.Add("a@a.com"); 可以发送给多人

//msg.CC.Add(c@c.com); 可以抄送给多人

String sendMailAddr = "jiyanglin@qq.com";

String sendName = "jiyanglin";

msg.From = new System.Net.Mail.MailAddress(sendMailAddr, sendName, System.Text.Encoding.UTF8);

msg.Subject = "我是标题";

msg.SubjectEncoding = System.Text.Encoding.UTF8;

msg.Body = "邮件内容";

msg.BodyEncoding = System.Text.Encoding.UTF8;

msg.Attachments.Add(new System.Net.Mail.Attachment("D:\\123-纪阳林.xlsx"));

msg.IsBodyHtml = false;

msg.Priority = System.Net.Mail.MailPriority.High;//邮件优先级

System.Net.Mail.SmtpClient client = new System.Net.Mail.SmtpClient();

client.Credentials = new System.Net.NetworkCredential("jiyanglin@qq.com", "\*\*\*\*\*\*\*\*\*");

client.EnableSsl = true;

client.Host = "smtp.qq.com";

object userState = msg;

try

{

//client.SendAsync(msg, userState);

client.Send(msg);

MessageBox.Show("发送成功");

}

catch (System.Net.Mail.SmtpException ex)

{

MessageBox.Show(ex.Message, "发送邮件出错");

}

}

### 串口

using System.IO.Ports;

类内变量：

SerialPort serialPort1 = new SerialPort("COM2", 9600, Parity.None, 8, StopBits.One);

#### 打开

serialPort1.Open();

#### 关闭

serialPort1.Close();

#### 发送

======发送字符串

serialPort1.Write("xx");

======直接发送16进制byte数组

byte[] send = new byte[3];

send[0] = 0xae;

send[1] = 0x05;

send[2] = 0x12;

serialPort1.Write(send, 0, 3);

======将16进制字符串转换为byte数组发送

private static byte[] strToToHexByte(string hexString,ref int size)

{

hexString = hexString.Replace(" ", "");

if ((hexString.Length % 2) != 0)

hexString += "";

byte[] returnBytes = new byte[hexString.Length / 2];

int i = 0;

for (; i < returnBytes.Length; i++)

returnBytes[i] = Convert.ToByte(hexString.Substring(i \* 2, 2), 16);

size = i;

return returnBytes;

}

int size = 0;

byte[] send = strToToHexByte(str,ref size);//str为ae 01 03这种

serialPort1.Write(send, 0, size);

#### 接收

serialPort1.DataReceived+=new SerialDataReceivedEventHandler(serialPort1\_DataReceived);

/////////对应函数

private void serialPort1\_DataReceived(object sender, SerialDataReceivedEventArgs e)

{

string s = serialPort1.ReadExisting();//接收串口的字符串

//throw new NotImplementedException();自动生成的函数需要将这句注释掉，否则运行该句后程序会退出

}

十六进制接收：

public string byteToHexStr(byte[] bytes)

{

string returnStr = "";

if (bytes != null)

{

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

{

returnStr += bytes[i].ToString("X2");

}

}

return returnStr;

}

使用时

int DataCount = serialPort1.BytesToRead;

byte[] ReCMD = new byte[DataCount];

serialPort1.Read(ReCMD, 0, DataCount);

string RecStr = byteToHexStr(ReCMD);

### 正则表达式

using System.Text.RegularExpressions;

===========字符条件判断

string str = "scca";

Regex reg = new Regex("^s.\*a$");

if (reg.IsMatch(str))

{

Console.WriteLine("满足条件");

}

Regex r = new Regex("abc");

MatchCollection matches = r.Matches("123abc45abcd");

foreach (Match match in matches)

{

Console.WriteLine("{0} found at pos {1}", match.Value, match.Index);

string str = match.Result("$&\_r");//将匹配结果加上\_r返回字符串

}

===========匹配内容

var input = "this \_is# a \_big \_apple#";

var pattern = @"\_(\w+)#";//\w+表示一个单词(连续非特殊字符) 匹配句子中所有以\_开头#结尾的单词 结果为is和apple

Match match = Regex.Match(input, pattern);

while(match.Success)

{

Console.WriteLine(match.Groups[1].Value);

match = match.NextMatch();

}

string input = "Born: July 12, 1010";

//有三个括号，所以匹配出来后有三组外加总体的匹配字符

string pattern = @"(\w+)\s(\d{1,2}),\s(\d{4})";

Match match = Regex.Match(input, pattern);

if (match.Success)

{

for (int ctr = 0; ctr < match.Groups.Count; ++ctr )

{

Console.WriteLine("group {0}: {1}", ctr, match.Groups[ctr].Value);

}

}

===========替换

//将匹配出的12.34前加上价格

string pattern = @"\b\d+\.\d{2}\b";

//\b代表单词边界 \d+代表多个数字 \.代表. d{2}代表两个数字

//匹配带小数点两位的数字

string replacement = "价格$&";//$&代表匹配上的内容 $是特殊转义字符 输出$用$$

string input = "apple: 12.34 banana: 6.5";

Console.WriteLine(Regex.Replace(input, pattern, replacement));

==========拆分

string input = "1.aa 2.bb 3.cc";

string pattern = @"\b\d+\.";//匹配序号加.

foreach(string item in Regex.Split(input,pattern))

{

if (!String.IsNullOrEmpty(item))

Console.WriteLine(item);

}

### LINQ

实现IEnumerable的类都可以用linq

#### 查询

string[] arry = { "a1","a2","b1","b2","c"};

var query = from n in arry where n.StartsWith("a") select n;//LINQ语句

string str="";

foreach(var s in query){str += s;}

from n in arry类似foreach

where后是限定选择的条件

select字句是必须的，指定结果集中包含那些元素。

int[] numbers = {3,5,2,5,12,5,36,75,42};

var qur = from num in numbers where num % 2 == 0 orderby num descending select num;

//只有调用语句才会运行

//var qur = numbers.Where(n => n % 2 == 0).OrderBy(n => n);

foreach (var i in qur) Console.WriteLine(i + "");

int[] ret = qur.ToArray<int>();

#### 方法语句

var query = arry.Where(n=>n.StartsWith("a"));

var query = arry.Where(n=>n.Length==1);

#### 排序

var query = from n in arry where n.StartsWith("a") orderby n select n;

var query = arry.OrderBy(n=>n).Where(n=>n.StartsWith("a"));

var query = arry.OrderBy(n => n.Substring(n.Length - 1)).Where(n=>n.StartsWith("a"));//排序时，按照最后一个字符排

#### 聚合运算符

可以对查询结果进行分析。

int count=query.Count();

================let

string[] strs = { "aA-11", "bb-22","cc" };

var qur = from s in strs

let words = s.Split('-')

from word in words

let w = word.ToUpper()

select w;

foreach (var s in qur)

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

================分组与表连接

class Customer

{

public string Name { get; set; }

public string City { get; set; }

}

class Employee

{

public string Name { get; set; }

public int ID { get; set; }

}

List<Customer> cust = new List<Customer>();

cust.Add(new Customer() { Name="aa", City = "beijing" });

cust.Add(new Customer() { Name = "bb", City = "beijing" });

cust.Add(new Customer() { Name = "cc", City = "xian" });

List<Employee> em = new List<Employee>();

em.Add(new Employee() { Name = "aa", ID = 1 });

em.Add(new Employee() { Name = "dd", ID = 2 });

=========分组查询

var quer = from c in cust

group c by c.City;

foreach (var cg in quer)

{

Console.WriteLine(cg.Key);

foreach (var c in cg) Console.WriteLine(" " + c.Name);

}

var quer = from c in cust

group c by c.City into custGroup

where custGroup.Count() >= 2

select new { City = custGroup.Key, number = custGroup.Count() };

foreach (var c in quer)

Console.WriteLine("{0} {1}", c.City,c.number);

============表连接

var queryJoin = from c in cust

join e in em on c.Name equals e.Name

select new { PersonName = c.Name, PersonID = e.ID, PersonCity = c.City };

foreach (var p in queryJoin)

Console.WriteLine("{0} {1} {2}", p.PersonName, p.PersonID, p.PersonCity);

### guid

string guid = Guid.NewGuid().ToString("N");

### 获取主板信息

System.Management;

ManagementClass mc = new ManagementClass("Win32\_BaseBoard");

ManagementObjectCollection moc = mc.GetInstances();

string strID = null;

foreach (ManagementObject mo in moc)

{

strID = mo.Properties["SerialNumber"].Value.ToString();

break;

}

### 打开文件夹对话框

wpf需要添加dll引用（解决方案文件浏览中引用目录）

System.Windows.Forms.FolderBrowserDialog fb = new System.Windows.Forms.FolderBrowserDialog();

fb.Description = "选择目录";

if (System.Windows.Forms.DialogResult.OK == fb.ShowDialog())

{

string path = fb.SelectedPath;

}

### 打开文件对话框

var of = new Microsoft.Win32.OpenFileDialog() { Filter = "jpg|\*.jpg||" };

if (true == of.ShowDialog())

{

string file = of.FileName;

}

### 限制输入数字

lvds1.PreviewTextInput += new TextCompositionEventHandler(EditPreviewTextInput

);

void EditPreviewTextInput(object sender, TextCompositionEventArgs e)

{

e.Handled = false;

if (!isNumberic(e.Text)) e.Handled = true;

}

public bool isNumberic(string \_string)

{

if (string.IsNullOrEmpty(\_string))

return false;

foreach (char c in \_string)

{

if (!char.IsDigit(c))

return false;

}

return true;

}

### 嵌入资源

在项目dirname目录下存在filename.txt，生成操作设置为嵌入的资源

string resourceName = MethodBase.GetCurrentMethod().DeclaringType.Namespace + ".dirname.filename.txt";

Stream stream = Assembly.GetExecutingAssembly().GetManifestResourceStream(resourceName);

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

string strline = sr.ReadToEnd();

### Resource

工程WpfApp1加入目录dirname，复制到输出目录：不复制，生成操作：Resource

代码中可以"/WpfApp1;component/dirname/文件名"进行引用（用于图像内容的url）

img.Source = new BitmapImage(new Uri("/prjName;component/dirname/1.jpg", UriKind.RelativeOrAbsolute));

### 环境变量

加入临时环境变量

string dllDirectory = System.AppDomain.CurrentDomain.BaseDirectory + "dirname\\dir";

Environment.SetEnvironmentVariable("PATH", Environment.GetEnvironmentVariable("PATH") + ";" + dllDirectory);

## c++mix

### C++调用C#

c++工程必须clr支持

导出类和函数必须public

#using "Csharp.dll"

using namespace AddSpace;

Add^ add = gcnew Add();

add.fun();

捕获c#异常

try{ ....}

catch (System::Exception^ e){}

托管类不能作为类成员或全局变量

可以通过在类内创建类本身的全局变量进行全局使用

public class AA

{

public static AA obj = new AA();

public void run(string str){.....}

｝

c++调用时：

AA::obj->run(str)

### CLR类库

建立C++CLR类库，该类库生成的dll，可以直接在c#中添加引用后使用该命名空间内的类。

与C#交互的函数中不能包含c#不支持的类型，否则在C#中不会提示相应函数

字符串传递

String^ Fun()

{

CString ss = L"吃饭睡觉";

String^ str = gcnew String(ss.GetBuffer());

return str;

}

void Fun(String^ str)

{

CString ss(str);

char\* ch2 = (char\*)(void\*)System::Runtime::InteropServices::Marshal::StringToHGlobalAnsi(str);

### 数据类型

|  |  |  |
| --- | --- | --- |
| C/C++ | C# | 长度 |
| short | short | 2 |
| int | int | 4 |
| long | int | 4 |
| bool | bool | 1 |
| char | byte | 1 |
| wchar\_t | char | 2 |
| float | float | 4 |
| double | double | 8 |

### 调用方式

[DllImport("\*\*.dll", EntryPoint = "fun", CallingConvention = CallingConvention.StdCall, CharSet = CharSet.Auto)]

public static extern void fun();

using System.Runtime.InteropServices;

导入的函数中，字符串类型都用string，数字可以用uint、long等。

句柄可以用uint或者IntPtr

using System.Windows.Interop;

IntPtr hwnd = new WindowInteropHelper(this).Handle;

### c++回调c#

typedef void (\_\_stdcall \*RR)(int x);

extern "C" \_\_declspec(dllexport) void fun(RR rr)

public delegate void RR(int x);

[DllImport("Dll1.dll", EntryPoint = "fun") ]

public static extern void fun(RR rr);

fun((int xx)=> { Console.WriteLine(xx); });

### 字符串传递

extern "C" \_\_declspec(dllexport) void \_\_stdcall show(wchar\_t \*str)

public static extern unsafe void show(char\* str);

unsafe

{

string str = "abc吃饭def";

fixed (char\* p = &(str.ToCharArray()[0]))

{

show(p);

}

}

extern "C" \_\_declspec(dllexport) void \_\_stdcall getStr(char \*outStr)

{

strcpy\_s(outStr, MAX\_PATH, "abc吃饭def");

}

public static extern unsafe void getStr(byte\* outStr);

unsafe

{

byte[] str = new byte[260];

fixed (byte \*p = &str[0])

{

getStr(p);

}

string ret = Encoding.Default.GetString(str);

int pos = ret.IndexOf('\0');

ret = ret.Substring(0, pos);

}

### 数组传递

extern "C" \_\_declspec(dllexport) void \_\_stdcall showArry(float \*pDat,int len)

public static extern unsafe void showArry(float\* pDat, int len)

float[] dat = new float[3] { 1.1f, 2.2f, 3.3f };

fixed (float\* parry = &dat[0])

{

showArry(parry, dat.Length);

}

### 结构体传递

struct INFO

{

int aa; //结构体对齐 8

double bb;

char cc[100];

};

void \_\_stdcall testrun(INFO \*info)

{

info->aa = 1;

info->bb = 2.2;

strcpy\_s(info->cc, "吃饭");

}

=======调用方式1

[StructLayout(LayoutKind.Sequential)]

public struct INFO

{

public int aa;

public double bb;

[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 100)]

public string cc;

}

[DllImport("AA.dll", EntryPoint = "testrun")]

public static extern void testrun(ref INFO info);

INFO info = new INFO();

testrun(ref info);

=======调用方式2

[DllImport("AA.dll", EntryPoint = "testrun")]

public static extern void testrun(IntPtr pv);

IntPtr pv = Marshal.AllocHGlobal(8+8+100);

testrun(pv);

int aa = Marshal.ReadInt32(pv, 0);

byte[] b = new byte[8];

for(int i = 0; i < 8;++i)

b[i] = Marshal.ReadByte(pv, 8+i);

double bb = (double)BitConverter.ToDouble(b,0);

string cc = Marshal.PtrToStringAnsi(pv + 8+8);

Marshal.FreeHGlobal(pv);

### 内存拷贝

IntPtr datpr = datptr;

Marshal.Copy(dat, 0, datpr,dat.Length);

### WIN32DLL

#### 分配控制台

private const string Kernel32\_DllName = "kernel32.dll";

[DllImport(Kernel32\_DllName)]

private static extern bool AllocConsole();

#### 隐藏鼠标

[DllImport("user32.dll", EntryPoint = "ShowCursor", CharSet = CharSet.Auto)]

//或者CharSet.Unicode

public static extern int ShowCursor(bool bShow);

使用时：

ShowCursor(false);

#### 读写ini

[DllImport("kernel32")]

private static extern long WritePrivateProfileString(string section, string key, string val, string filePath);

[DllImport("kernel32")]

private static extern int GetPrivateProfileString(string section, string key, string def, StringBuilder retVal, int size, string filePath);

写入：

WritePrivateProfileString("section", "key", "xx", ".\\1.ini");

读取：

StringBuilder temp = new StringBuilder(1024);

string result;//接收用的字符串

if (0 == GetPrivateProfileString("section"," key", "", temp, 500, ".\\1.ini"))

{

result = null;

}

else

{

result = temp.ToString();

}

#### 发送消息

[DllImport("User32.dll")]

private static extern uint SendMessage(uint hWnd, uint Msg, uint wParam, uint lParam);

[DllImport("User32.dll")]

private static extern uint FindWindow(string className, string windowName);

uint hWnd=FindWindow(null, "name");

if (hWnd >0)

{

SendMessage(hWnd, 0x0400 + 1, 0, 0);//代表WM\_USER

}

#### 关机

[DllImport("user32.dll")]

static extern bool ExitWindowsEx(uint uFlags, uint dwReason);

ExitWindowsEx(0x01,0); //注销:0x00 关机：0x01 重启：0x02

## openXML

### Word

#### 安装

<https://msdn.microsoft.com/en-us/library/office/cc850833.aspx>

OpenXMLSDKv2.msi

OpenXMLSDKTool.msi

可以不安装，添加引用即可:

C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.0\WindowsBase.dll

C:\Program Files (x86)\Open XML SDK\V2.0\lib\DocumentFormat.OpenXml.dll

#### 创建

会和System.Windows.Documents重名

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Wordprocessing;

using DocumentFormat.OpenXml.Packaging;

using OpenXmlParagraph = DocumentFormat.OpenXml.Wordprocessing.Paragraph;

using OpenXmlWordRun = DocumentFormat.OpenXml.Wordprocessing.Run;

WordprocessingDocument doc = WordprocessingDocument.Create(pathName, WordprocessingDocumentType.Document);

MainDocumentPart mainPart = doc.AddMainDocumentPart();

mainPart.Document = new Document();

Body body = mainPart.Document.AppendChild(new Body());

//....写入段落

OpenXmlWordRun run = new OpenXmlWordRun(new Text(str));

OpenXmlParagraph paragraph = body.AppendChild(new OpenXmlParagraph());

paragraph.AppendChild(run);

//....写入

doc.Close();

#### 打开

WordprocessingDocument.Open(filePath, true)

#### 文字格式

RunFonts fonts = new RunFonts() { EastAsia = "黑体" };

FontSize size = new FontSize() { Val = "70" };

DocumentFormat.OpenXml.Wordprocessing.Color color = new DocumentFormat.OpenXml.Wordprocessing.Color() { ThemeColor = ThemeColorValues.Accent2};

//或使用 runProperties.Append(color); runProperties.Append(size);

RunProperties runProperties = new RunProperties(color,size,fonts);

Text txt = new Text("word文字");

OpenXmlWordRun run = new OpenXmlWordRun(runProperties, txt);

OpenXmlParagraph paragraph = body.AppendChild(new OpenXmlParagraph());

paragraph.AppendChild(run);

#### 插入图像

using A = DocumentFormat.OpenXml.Drawing;

using DW = DocumentFormat.OpenXml.Drawing.Wordprocessing;

using PIC = DocumentFormat.OpenXml.Drawing.Pictures;

public void AddPictureIntoWord(string docfilePath, string picturePath)

{

using (WordprocessingDocument doc = WordprocessingDocument.Open(docfilePath, true))

{

ImagePartType imagePartType = ImagePartType.Jpeg;

//string picType = picturePath.Split('.').Last();

//if (!Enum.TryParse<ImagePartType>(picType, true, out imagePartType)) return; // 通过后缀名判断图片类型, true 表示忽视大小写

ImagePart imagePart = doc.MainDocumentPart.AddImagePart(imagePartType);

imagePart.FeedData(File.Open(picturePath, FileMode.Open)); // 读取图片二进制流

AddImageToBody(doc, doc.MainDocumentPart.GetIdOfPart(imagePart));

}

}

private void AddImageToBody(WordprocessingDocument wordDoc, string relationshipId)

{

// Define the reference of the image.

var element =

new Drawing(

new DW.Inline(

new DW.Extent() { Cx = 990000L, Cy = 792000L }, // 调节图片大小

new DW.EffectExtent()

{

LeftEdge = 0L,

TopEdge = 0L,

RightEdge = 0L,

BottomEdge = 0L

},

new DW.DocProperties()

{

Id = (UInt32Value)1U,

Name = "Picture 1"

},

new DW.NonVisualGraphicFrameDrawingProperties(

new A.GraphicFrameLocks() { NoChangeAspect = true }),

new A.Graphic(

new A.GraphicData(

new PIC.Picture(

new PIC.NonVisualPictureProperties(

new PIC.NonVisualDrawingProperties()

{

Id = (UInt32Value)0U,

Name = "New Bitmap Image.jpg"

},

new PIC.NonVisualPictureDrawingProperties()),

new PIC.BlipFill(

new A.Blip(

new A.BlipExtensionList(

new A.BlipExtension()

{

Uri =

"{28A0092B-C50C-407E-A947-70E740481C1C}"

})

)

{

Embed = relationshipId,

CompressionState =

A.BlipCompressionValues.Print

},

new A.Stretch(

new A.FillRectangle())),

new PIC.ShapeProperties(

new A.Transform2D(

new A.Offset() { X = 0L, Y = 0L },

new A.Extents() { Cx = 990000L, Cy = 792000L }), //与上面的对准

new A.PresetGeometry(

new A.AdjustValueList()

) { Preset = A.ShapeTypeValues.Rectangle }))

) { Uri = "http://schemas.openxmlformats.org/drawingml/2006/picture" })

)

{

DistanceFromTop = (UInt32Value)0U,

DistanceFromBottom = (UInt32Value)0U,

DistanceFromLeft = (UInt32Value)0U,

DistanceFromRight = (UInt32Value)0U,

EditId = "50D07946"

});

// Append the reference to body, the element should be in a Run.

wordDoc.MainDocumentPart.Document.Body.AppendChild(new Paragraph(new Run(element)));

}

### Excel(EPPlus.dll)

#### 创建/打开

ExcelPackage \_pakage = new ExcelPackage(new FileInfo(excelFilePathName));

#### 选择sheet页

ExcelWorksheet \_curSheet = \_pakage.Workbook.Worksheets[1];

\_curSheet = m\_pakage.Workbook.Worksheets[sheetName];

#### 创建sheet页

\_curSheet = \_pakage.Workbook.Worksheets.Add(sheetName);

#### 保存

\_pakage.Save();

#### 数据操作

##### 获取行列数

int rowMax = \_curSheet.Dimension.End.Row;

int colMax = \_curSheet.Dimension.End.Column;

##### 获取单元格数据

string val = (string)\_curSheet.GetValue(row, col);

if (null == val) val = "";

##### 设置单元格数据

\_curSheet.SetValue(row, col, val);

#### 格式

##### 网格线条

\_curSheet.View.ShowGridLines = false；

##### 合并单元格

\_curSheet.Cells[row1, col1, row2, col2].Merge = true;

##### 对齐方式

\_curSheet.Cells[row, col].Style.HorizontalAlignment = ExcelHorizontalAlignment.Left

##### 字体颜色

\_curSheet.Cells[row, col].Style.Font.Color.SetColor(System.Drawing.Color.FromArgb(r, g, b));

##### 单元格背景色

\_curSheet.Cells[row, col].Style.Fill.PatternType = ExcelFillStyle.Solid;

\_curSheet.Cells[row, col].Style.Fill.BackgroundColor.SetColor(Color.FromArgb(r, g, b));

##### 超链接

m\_curSheet.Cells[row, col].Hyperlink = new ExcelHyperLink(path, UriKind.Relative);

m\_curSheet.Cells[row, col].Style.Font.Color.SetColor(System.Drawing.Color.FromArgb(0, 0, 255));

m\_curSheet.Cells[row, col].Style.Font.UnderLine = true;

sheet页链接：

curSheet.Cells[row, col].Hyperlink = new ExcelHyperLink("sheet名!A1", "显示文字");

##### 单元格大小

\_curSheet.Column(col).Width = 10；

## mongodb

var collection = database.GetCollection<BOOK>("jylcl");

//BsonDocument

//var document = new BsonDocument{{ "name", "a" },{ "Price", 10 }, };

//插入数据

BOOK bk = new BOOK();

bk.Name = "a";

bk.Price = 10;

collection.InsertOne(bk);

bk = new BOOK();

bk.Name = "b";

bk.Price = 15;

collection.InsertOne(bk);

bk = new BOOK();

bk.Name = "c";

bk.Price = 17;

collection.InsertOne(bk);

bk = new BOOK();

bk.Name = "d";

bk.Price = 25;

collection.InsertOne(bk);

//查询

var filter = Builders<BOOK>.Filter.Eq("Name", "a");

var result = collection.Find(filter).ToList();

foreach (var item in result) { }

var builder = Builders<BOOK>.Filter;

filter = builder.Gt("Price", 10) & builder.Lt("Price", 20);//10 > val && val < 20

result = collection.Find(filter).ToList();

//删除

filter = Builders<BOOK>.Filter.Eq("Name", "a");

var delResult = collection.DeleteOne(filter);