.Net：一般指.Net Framework框架(.Net 类库；CLR公共语言运行时，点击F5时，CLR首先加载所有的类文件到内存中，找到主函数执行)，是一种平台(生产笔记本的机器)，一种技术，.Net实现了一个语言无关性，平台将编程语言编译为中间语言由CLR调用

C#：一种编程语言，可以开发基于.Net平台的应用

Java既是一种平台又是一种编程语言

客户端(客户信息保存在服务器)，服务器(特殊电脑，存储信息，硬盘大)

桌面应用程序 Winform(.Net开发的桌面应用程序叫Winform应用程序)

Internet应用程序 ASP.NET(.Net开发的internet应用程序叫ASP.NET应用程序)

Unity3D游戏开发或者虚拟现实

.NET两种交互模式：C/S(客户端需要安装专门的客户端软件，多用于桌面应用程序)，B/S(客户只需要安装浏览器，多用于Internet应用程序)

MSDN查询

启动：devenv

.cs为类文件

.sln解决方案文件

.csproj项目文件

Ctrl+K+D 快速对齐代码

shift end shift home选中某一行

F1转到帮助文档

#region #endregion 折叠冗余

ctrl+K+C 注释

Ctrl+ K+U 取消注释

Ctrl + L 快速剪切一行

Ctrl+D 快速复制粘贴一行

Ctrl + Tab 程序内部的页签切换

Alt + Tab 程序与程序的切换

单行注释 // 多行注释 、/\* \*/ 文档注释 ///

变量的声明：

数据类型：int(整数)，double(小数和整数)，string(多个文本可以为空，被双引号引起来)，char(一个文本且不能为空，被单引号引起来)，decimal(金钱类型后缀加上m，精度要高于double类型)

值类型：int ,double,char,bool,decimal,struct,enum(值存在栈中)

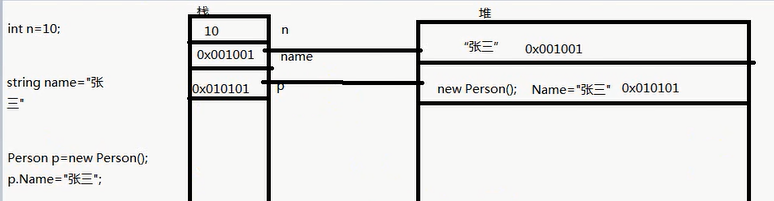
引用类型：string,数组，自定义类(值存在堆中)，集合，object，接口

值传递和引用传递：

值传递：传递的值本身；引用传递：传递的是地址

ref将一个值传递变成引用传递

内存：堆，栈，静态存储区



由于字符串的不可变性(字符串的不可变性指的是内存中的字符串不可变，而不是变量不变)，对字符串进行大量的拼接，赋值等操作时，会产生大量的垃圾，此时使用StringBuilder，不会占用太多的内存空间

StringBuilder 类名 = new StringBuilder()

类名.Append(参数)

类名.Insert(索引，”字符串”)

类名.ToString()

例：拼接html表格

使用WebBrowser控件，对象.DocumentText = 拼接的字符串

string可以看作是char的只读数组，string s = “hello”,则s[0]= ‘h’

string s = s1;s指向s1指向的字符串，如果s1指向改变，s指向的仍是hello

变量的命名规范：

必须以字母，下划线，@开头，不能以数字开头，后面可以跟任意的字母，数字，下划线

Camel:驼峰命名法，多用于变量命名，首单词首字母小写，其它单词的首字母大写

Pascal：多用于类和方法的命名，每个单词的首字母都要大写

控制台语句：

Consel.WriteLine();打印到控制台都是字符串类型

string Var = Consel.ReadLine();从控制台接受的都是字符串类型

Console.ForegroundColor = ConsoleColor.Blue;

逻辑控制：

对于类似标志变量的使用；最值的获取(定义变量与其它值比较)；字符串反转(数组名[i]与数组名[数组名.Length-1-i]互换)；冒泡排序(数组中的元素依次与后续的元素比较交换，上次已经交换过的数字已不再进行比较)；字符串反转(将字符串转成数组转换后再转换成字符串;反序输出)；字符串中单词反转(先将空格去掉截取到字符串数组)

运算符：

+号：如果加号两边有一边是字符串，就是连接的作用；如果两边都是数字则为相加的作用

占位符：与printf的格式控制字符有相似的用法但是不区分数据类型，每个输出语句都从{0}开始,{0:0.00}表示保留两位小数，改变了结果的显示

字符串的连接：+,$，string.Format

转义符：特殊意义的字符，\”表示英文半角的双引号，\表示转义将双引号打印出来不表示字符；\n仅局限在控制台换行，在文本中换行需要使用\r\n,windows操作系统只认识\r\n；\b表示退格键，但是放在字符串的两边没有效果；\t表示Tab键，用于排版对齐；\\表示一个\

@：写在字符串前，取消\在字符串中的转义作用，使其单纯的表示为一个\；让代码按照编译的原格式输出

算术运算符(++：a=5,a+++(++a)=5+7=12)，赋值运算符，复合运算符，关系运算符，逻辑运算符

算术运算符：算术运算符的操作数类型要一致，操作的结果类型也一致，不一致转换

转换：

隐式或自动转换：两种类型兼容(如都为数字类型)，必须小的类型转为大的类型

显示或强制类型转换：丢失精度，两种类型兼容(如都为数字类型)，必须大的类型转为小的类型

类型不兼容并且面上要过去的转换：Convert类型转换，如string类型的数字转换为int类型或者double型

两个变量的互换：n1=n1-n2;n2=n1+n2;n1=n2-n1;

运算符优先级：

一元运算符比二元运算符的优先级高

算术运算符>关系运算符>按位运算符>逻辑运算符>赋值运算符

顺序结构，分支结构(if,if-else(三元表达式))，选择结构(if else-if,switch-case)，循环结构(while,do while, for , foreach)

try-catch捕获异常

类型转换：

int a = int.Parse(字符串) (Convert内部调用此语句,转换失败抛异常)

bool b = int.TryParse(字符串，结果);(如果转换失败返回值为false，结果为0)

字符串：

字符串转换成char类型数组：char [] 数组名 = 字符串.ToCharArray()

char类型数组转换成字符串：字符串= new string(char类型数组名)

字符串长度：字符串.Length()

字符串转成大/小写：字符串.ToUpper()/ToLower()

字符串比较：字符串1.Equals(字符串2,StringComparison.OrdinalIgnoreCase)

两个字符串比较：直接用逻辑运算符即可判断，也可使用比较函数，但是大小写需要一致；

字符串的分割：string[] 字符串名 = 字符串.Split(new char[]{‘要截取的字符’},StringSplitOptions.RemoveEmptyEntries)

字符串的替换：string 字符串名= 字符串.Replace(‘旧字符串’,’新字符串’)

字符串截取(提取字符串)：string 字符串名= 字符串.Substring(指定位置，指定长度)(以字符出现的位置辅助提取)

字符串中是否包含子字符串：bool 字符串.Contains(“字符串”)

字符串中是否以子字符串开始：bool 字符串.StartsWith(“字符串”)

字符串中是否以子字符串结尾：bool 字符串.EndsWith(“字符串”)

某字符在字符串中开始出现的位置：int 字符串.IndexOf(‘字符’,开始位置)(先找到第一个字符的位置，循环查找所有出现的地方，当字符串不存在时返回值为-1)

某字符在字符串中最后出现的位置：int 字符串.LastIndexOf(‘字符’,开始位置)

字符串中的空格去掉：string 字符串名 = 字符串.Trim/TrimStart/TrimEnd()

串联对象数组中的各个元素： string 字符串名 =string.Join(“分隔符”，字符串数组名)

字符串是否为空：bool = string.IsNullOrEmpty(“字符串”)

空字符串：string.Empty

正则表达式：

^b：以b开头

$g：以g结尾(^b.\*$g表示以b开头以g结尾，b.\*g表示出现bg即可)

.：除了\n外的任意字符

\.：表示字符.

\[：表示[

\*：0次或多次

\d：表示数字

\d{4}：表示4位数字

\d{1,4}：表示1-4位数字

[0-9a-zA-Z]：其中的任意一个字符

[0-9a-zA-Z]+：其中的任意一个字符出现一次或多次

+：1次或多次

？：0次或1次

\s：空格

|：多种匹配的可能

()：分组(由分组提取字符串)((\w+)@(\w+)(\.\w){1,2}括号里面的东西出现一次或两次)

?<名字>：给组命名(可以直接使用名字提取字符串)

\w：特殊字符

1111。111。1。(.+。(贪婪模式).+?。(利用？终止贪婪模式))

Regex.IsMatch(“字符串”，”正则表达式”)

Match mc = Regex.Match(“字符串”， ”正则表达式”)

if(mc.Success)

mc.Value(匹配成功的字符串)

mc.Groups[0](匹配成功的字符串)

mc.Groups[1].Value(分组的第一个字符串)

MatchCollection mc = Regex.Matches(“字符串”， ”正则表达式”)

foreach(Match item in mc)

string str = Regex.Replace (“字符串”， ”正则表达式”，”替换的字符串”)

常量：

const 类型 常量名 = 常量

枚举：

[public] enum 枚举名

{

………………………

}

枚举名 变量名 = 枚举名.常量

枚举类型和int类型互相兼容，可以将枚举类型强制转换为int类型，也可以将int类型转换为枚举，枚举类型的值默认从0开始

枚举类型转换为字符串类型，调用ToString()

字符串类型转换为枚举类型，Enum.Parse(typeof(枚举名)，”1”或”男”)，如果要转化的字符串非数字在枚举类型中没有对应的值会抛异常

结构体：一次声明多个变量

[public] struct 结构名

{

结构成员(public修饰符才能访问)

}

结构名 变量名;然后给变量的成员赋值

结构成员称为字段，字段与变量相比可以存储多个值(多个变量的赋值)，而变量只能存储一个值

数组：

数组类型[] 数组名 = new 数组类型[数组长度]

数组类型[] 数组名 = {1,2,3,4,5}

new 数组类型[]{1,2,3,4,5}简写的常用类型

数组长度：数组名.Length

反转：Array.Reverse(数组名)

升序排序：Array.Sort(数组名)，如果想降序排列则先升序后反转

params可变参数数组，只能放在参数最后，识别调用的最后参数，可传递数组也可直接传递数组中的值(将实参列表中和可变参数数组类型一样的参数当作是可变参数数组中的元素，可变参数必须是形参列表中的最后一个元素)

方法：

方法的返回值可以是数组(相同类型的多个值)

out参数(方法返回不同类型的多个值，out参数需要在方法内赋值)

ref参数(将一个变量以参数的形式带到一个方法中进行改变，改变之后再将改变的值带出来，ref参数需要在方法外赋值)

方法递归：自己调自己，需要有结束的条件，返回时也有多次跳出

方法重载：函数名相同，参数不同(类型或个数)，方法的重载和返回值没有关系

将大象塞入冰箱

面向过程：打开，塞进去，关闭

面向对象：将冰箱提取出来作为对象，冰箱门被打开，大象被塞入冰箱，冰箱门被关闭

面向对象：

三大特征：封装，继承(类中相同的属性和方法，解决冗余)，多态

类：类是模子，确定对象将会拥有的特征(属性)和行为(方法)

[访问修饰符] class 类名

{

类的成员；

// 字段(存储数据)，属性(保护字段，对字段的取值和设值进行限定，get对字段判断，set对value进行判断，只需要限定一个即可)，索引器(方便使用类中的集合)，方法

//自动属性public string Name{get;set;}

//方法的参数一般通过属性获取

}

索引器：

person p = new p();

p[0] = 1;//可以直接赋值

class person

{

int[] nums = new int[100];

public int this[int index]

{

get{return nums[index];}

set{nums[index]=value;}

}

}

类名 变量名 = new 类名();类不占内存(静态类占内存)，对象占内存(new,在内存中开辟空间，在开辟的空间中创建对象，调用构造函数)

对象：类的实例化，this代表当前类的对象

对象初始化：对象的属性赋值

构造函数：对对象进行初始化，特殊函数(无返回值无void，构造函数名与类名一致)，可以重载，类中默认会有一个无参的构造函数，当有新的构造函数后，默认的无参构造函数被弄掉了，如果需要则需要自己创建，构造函数也可以对字段进行保护

public 构造函数名()

{

}

静态与非静态：

非静态类中，可以出现非静态和静态成员；静态类中，只能出现静态成员

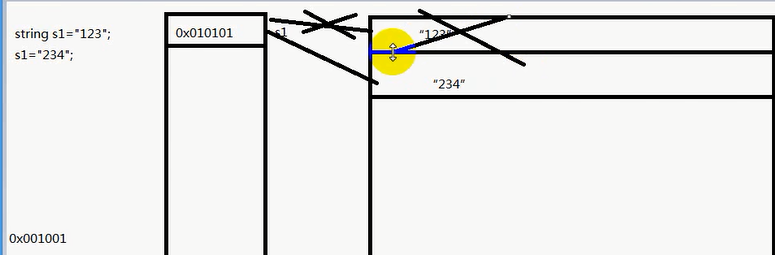
非静态函数中，可以访问非静态和静态成员；静态函数中，只能访问静态成员

调用：调用实例成员通过“对象”调用，调用静态成员通过“类名.成员”调用，静态类无法创建对象

静态可以资源共享，但是越少越好 ，工具类可以写成静态类

析构函数：程序结束调用

GC：Gargbage Collection 程序结束之后，如果那块空间没有被指向，则被销毁



在一个项目中引用另一个项目的类：

添加要引用的类所在的项目；引用命名空间；引用的类的修饰符为public(internal只能在程序集内使用)

类成员修饰符：public, private(当前类),protected(当前类子类),internal(当前项目中访问),protecte internal(同一个项目中internal比protected访问权限高，不同项目中protected比internal访问权限高)

继承：

子类继承了父类的属性和方法；不能继承父类的私有字段；没有继承父类的构造函数，会默认调用父类的无参构造函数，如果父类只有有参构造函数，则在父类写一个无参的构造函数或让子类调有参的父类构造函数使用关键字base()，才能成功调用子类对象(调用父类的构造函数时不需要再在子类的构造函数中对来自父类的属性赋值)

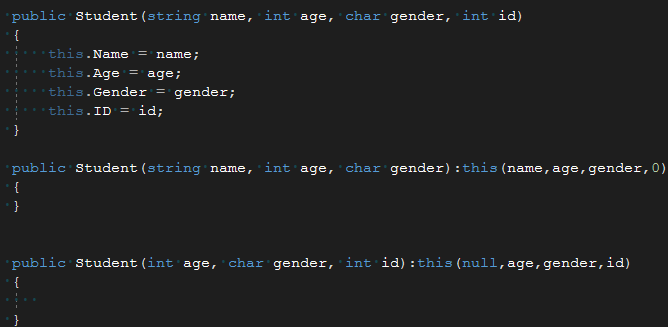
继承特性：单根性(只有单继承)，传递性

在C#中所有的类都直接或间接继承了object类，是一切类型的基类

隐藏：

new ：在子类中new关键字隐藏从父类继承的成员

this：当同一个类中的构造函数重载时，防止赋值重复使用this()调用自己的构造函数



多态：

一个对象能够表现出多种状态，父类的方法标记为虚函数，子类的方法标记为override，将子类对象赋值给父类，父类调用方法时方法被子类重写，调用的子类的函数实现多态

如果父类的方法没有标记为虚函数，当子类对象赋值给父类时，父类调用方法时仍是自己的方法

如果父类中的方法有默认的实现，并且父类需要被实例化，这时可以考虑将父类定义成一个普通类，用虚方法来实现多态

抽象类：

public abstract void 函数名();没有方法体，存在的意义是让子类重写；抽象成员只能在抽象类中，修饰符不能是private；无法创建抽象类的对象，抽象类中可以有构造函数(为属性赋值)，也可以包含实例成员如属性；子类必须重写父类的抽象方法，除非子类也是一个抽象类

如果父类中的方法没有默认实现，父类也不需要被实例化，则可以将该类定义为抽象类

接口：

主要用于解决多重继承，类可以继承多个接口(只能有方法，属性，索引器，事件，不能有字段和构造函数，接口可以继承接口，但不能继承类)，接口(一般以I开头以able结尾)中成员函数默认的修饰符为public且不实现，在非接口子类实现所有继承的接口成员函数，不同的类可以继承相同的接口，将子类赋值给接口类(不能创建对象)，同样可以实现多态

当子类有独特的功能不能从父类继承，可以用继承接口，具有某种能力，将子类赋给接口类调用接口实现函数；如果子类不能提取出父类则用接口实现多态

接口需要单一，不然一些函数对子类没有意义

当一个抽象类实现接口的时候，需要子类去实现接口(抽象类可以有非抽象成员实现接口，此时需要子类继承抽象类才能调用函数，因为抽象类无法创建对象)

显示实现接口：

防止子类中的方法与继承接口的方法重名 且参数相同

密封类：

sealed，表示这个类不能被继承，但是可以继承其他类

部分类：

里氏转换：

子类可以赋值给父类(父类的参数可以用子类代替)

父类中装的是子类对象，可以将父类强转为对应的子类对象

类型转换：is 类型转换是否成功，如果成功返回true，失败则为false；as类型转换，转换成功返回转换的对象，转换失败则返回null

WebClient类：

提取页面(.html文件)的内容(同样可以使用File类提取内容)

WebClient web = new WebClient()

string html = web.DownloadString(@“路径”)(可能会乱码)

byte[] buffer = web.DownloadData(@”路径”)

string html = Encoding.UTF8.GetString(buffer)

web.DownloadFile(“原路径”，”本地路径”)

File类：

只能操作小文件，一次性读写耗内存

引用命名空间快捷键：alt + shift+F10

bool File.Exist(string path)

File.Create(string path)

File.Delete(string path)

File.Copy(“旧文件路径”,”新文件路径”)

File.Move(“旧文件路径”,”新文件路径”)剪切

string [] 字符串数组名= File.ReadAllLines(“文本文件(拖到记事本中仍然看的见的文件)路径”, Encoding.UTF-8)(文件保存的编码格式与打开的编码格式应该一致否则出现乱码)

string 字符串名 = File.ReadAllText(“文本文件路径”, Encoding.UTF-8)

byte[] 字节数组名= File.ReadAllBytes(“任何形式文件路径”)(字节数组名[i].ToString()得到的内容需要解码，采用string = Encoding.Default/UTF8.GetString(字节数组名，索引，个数)，应用于文件复制，ToString直接转换成字符串，GetString转换为有意义的字符)

File.WriteAllLines(“文件路径”， new string[]{“1”, “2”})(会覆盖源文件)

File.WriteAllText(“文件路径”， “1”)(会覆盖源文件)

File.WriteAllBytes(“文件路径”，字节数组名)(会覆盖源文件byte[] = Encoding.Default/UTF8.GetBytes(字符串名)，应用于文件粘贴)

File.AppendAllText(“文件路径”， “1”， Encoding.UTF-8)

Directory类：

Directory.CreateDirectory(“文件夹路径”)

Directory.Delete(“文件夹路径”,true(里面没有东西仍然删除))

Directory.Move(“文本文件源路径”,”目标路径”)剪切

string[] path = Directory.GetFiles(“文本文件路径”， “\*.txt”)(获取文件路径)

string[] path = Directory.GetDirectories(“文本文件路径”)(获取文件夹路径)

Path类：

专门操作路径的字符串(相对路径是相对于应用程序的路径)

文本文件名称= Path.GetFileName(路径字符串名)

文本文件名不包含扩展名= Path.GetFileNameWithoutExtention(路径字符串名)

文本文件扩展名= Path.GetFileExtention(路径字符串名)

文本文件改变后的扩展名= Path.GetChangeExtention(路径字符串名)

文本文件的文件夹名= Path.GetDirectoryName (路径字符串名)

文本文件的绝对路径= Path.GetFullPath (路径字符串名)

string = Path.Combine(“字符串1”,”字符串2”)

文件流：

操作大文件(可以多次读取)，FileStream操作字节，StreamWriter StreamReader操作字符

FileStream类：

FileStream 流对象 = new FileStream(“操作文件路径”，FileMode.OpenOrCreate/Append(追加写入)，FileAccess.Read/Write)

流对象.Length表示实际文件的大小

int r = 流对象.Read(字节数组名，偏移量，大小(一般为字节数组大小))(从流中读取数据并写入指定的缓冲区，返回实际读到的字节数，多次读取时返回值为0 时结束)

流对象.Write(字节数组名，偏移量，大小(读取的有效字节数))( byte[] = Encoding.Default/UTF8.GetBytes(字符串名))

流对象.Close()

流对象.Dispose()释放资源

将创建FileStream对象放在using(创建对象)｛｝微软帮助关闭和释放流资源

StreamReader类：

StreamReader 流对象 = new StreamReader(“文件路径”，Encoding.UTF-8)

string = 流对象.ReadLine()

流对象.EndOfStream(流是否结尾)

StreamWriter类：

StreamWriter 流对象 = new StreamWriter (“文件路径”，TRue(追加写入))

流对象.write(“字符串”)

ArrayList集合：

数组的长度固定，集合的长度可以任意改变，可以添加任意类型的元素，但是从集合中取值必须要做不停的转换，这时使用泛型集合会更安全

ArrayList 集合名 = new ArrayList();

集合名.Add(单个成员，对象)

集合名.AddRange(数组等集合)

集合名.Count(如过使用Add添加，添加一个数组个数为1，用AddRange添加一个数组个数为数组元素个数)

集合名[索引]访问集合成员(对于对象，数组或集合，如果使用add添加需要用is判断后强转输出((int [])集合名[集合索引])[数组索引]，如果是对象则可以(类名)集合名[集合索引].集合成员来访问成员；如果使用AddRange添加，集合名[集合索引]只表示数组中的某个元素)

集合名.Clear()删除集合中所有的元素

集合名.Remove(单个成员(也可以是用索引表示的数组中的某个元素)，如果是Add添加，集合名[集合索引]即可删除整个数组集合，无法删除数组中的某个元素)

集合名.RemoveAt(索引)(如果是Add添加，某个索引即可删除整个数组元素)

集合名.RemoveRange(索引，个数)(通过索引删除数组等集合中的多个元素)

集合名.Insert(单个成员)

集合名.InsertRange(索引，数组等集合)

集合名.Contain()

集合名.Sort()(同种类型的数据才可以比较)

集合名.Capacity 集合中实际包含的元素个数(空间不够时成倍增加)

Hashtable集合：

键是唯一的，值可以重复

Hashtable集合名 = new Hashtable ();

集合名.Add(键，值)

集合名[键]=值 通过此种方式给集合中添加元素，如果键与Add添加的重复则会替换

集合名.Count

集合名[键]才能访问值，只能使用foreach(var item in 集合名.键)访问集合中的元素集合名[item]

(强类型语言C#，变量的类型必须有明确的定义，弱类型语言Javascript，变量名.GetType()获得变量类型，var推断类型，必须要赋初始值来判断类型)

集合名.Clear()

集合名.ContainsKey(键)

集合名.Remove(键)

泛型：

List(ArrayList)，Dictionary(Hashtable)

List：

List<类型> 集合名= new List<类型>()

集合名.Add(类型值)(new 类名(){字段 = 值，})

集合名.AddRange(new 类型[]{1,2,3,4,5})(可以直接添加集合名)

集合名.Count

集合名[索引]

foreach(类型 itme in 集合名)(在循环次数超级多时效率更高)

集合名.ToArray()将集合转换为数组

Dictionary：

Dictionary<int，string> 集合名 = new Dictionary<int，string>()

集合名.Add(键，值);

集合名[键] = 值

foreach(var item in 集合名.Keys)

foreach(KeyValuePair<int，string> item in 集合名){item.Key，item.value}

例：统计Welcome to China中每个字符出现的次数，不考虑大小写

string str = “Wlcome to China”;

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

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

{

if(str[i] == ‘ ’ )

{

continue;

}

if(!dic.ContainsKey(str[i]))

{

dic.Add(str[i],1)

}

else

{

dic[str[i]]++;

}

}

装箱和拆箱：

装箱：将值类型转换为引用类型，object o = n，拆箱：将引用类型转换为值类型，int n = (int)o

装箱或拆箱要观察发生类型转换的这两个类型是否存在继承关系，存在可能会发生拆箱或装箱

序列化和反序列化：

序列化：将对象转换为二进制；反序列化：将二进制转换为对象；作用：传输数据

1、将序列化的对象标记为可以被序列化[Serializable]

2、创建写入流对象；BinaryFormatter 对象名 = new BinaryFormatter()；对象名.Serialize(流对象，传输的对象)

3、创建读出流对象；BinaryFormatter 对象名 = new BinaryFormatter()；传输的对象= (传输的对象)对象名.Deserialize(流对象)

设计模式：

简单工厂设计模式：

抽象类(不知道用户的具体操作且操作不同时用子类继承)

例：超市收银系统

Guid.newGuid().ToString产生独一无二的编号

单例模式：

私有的构造函数；静态字段，静态的方法中判断静态的字段是否有值返回一个对象，创建单例

MD5加密：

MD5 md5 = MD5.Create();

byte[] = Encoding.Default/UTF8.GetBytes(string)

byte[] = md5.ComputeHash(字节数组名)

//string = Encoding.Default/UTF8.GetString(字节数组名)并不是转成有意义的字符而是将每个元素都ToString()并转换为十六进制

字节数组名[i].ToString(“X2”)每次都是2位十六进制

WinForm窗体应用程序：

智能的客户端技术，智能的获取数据和显示数据

随机数：

Random 对象 = new Random()

对象.Next(n1,n2)

日期：

DateTime.Now.ToString()(获取系统当前时间)

DateTime.Now.Year/Hour/Minute/Second

音乐：

SoundPlayer

对象.SoundLocation = “.wav的文件路径”

对象.Play()

打开对话框：

OpenFileDialog

对象.Title

对象.Multiselect

对象.InitialDirectory

对象.Filter = 视频文件|\*.wav

对象.ShowDialog()==DialogResult.OK(对话框是否打开)

string[] = 对象.FileNames(所有选中文件的全路径)

保存对话框：

SaveFileDialog

对象.Title

对象.InitialDirectory

对象.Filter = \*.wav

对象.ShowDialog()

string[] = 对象.FileName(所有选中文件的全路径)

颜色对话框：

ColorDialog

对象.ShowDialog()

文本框对象.ForeColor =对象.Color获得颜色

对话框：

FontDialog

对象.ShowDialog()

文本框对象.ForeColor = 对象.Font获得字体

MessageBox.Show(“字符串”)

鼠标坐标：

Point = 对象.PointToClient(MousePosition(鼠标的坐标相对于屏幕))(鼠标相对于对象坐标)

**窗体**：

事件：

Load，Paint，FormClosing(关闭窗体前发生)，Closing(关闭窗体时发生)

对象.Close()

对象.Show()

对象.ShowDialog()(创建的窗体永远都在老窗体的前面)

对象.ClientSize.Width动态获得窗体宽度

对象.Location = new Point(x,y);窗体的位置

**MDI窗体设计：**

属性：

窗口样式：IsMdiContainer(使用MenuStrip控件)

对象.MdiParent(为对象的父窗体赋值)

LayoutMdi(MdiLayout. TileHorizontal)在MDI父窗体内排列多文档界面

**窗体间控件的移动**：



**按钮**：

属性：

布局：Anchor(与窗体边缘的位置保持不变)，Dock(摆放位置)

行为：Enable，Visible

设计：Name

Text

事件：



Click，MouseEnter

对象.Location(x,y)

**TextBox**：

属性：

行为：Multiline(多行)，WordWrap(自动换行)，PasswordChar(密码显示字符一般为\*)

外观：ScrollBars(当为多行显示时会自动换行没有横着的滚动条)

事件：

TextChanged

对象.AppendText

对象.Clear()

对象.Focus()

**Timer**：

**WebBrowser**：

属性：

行为：Url = www.baidu.com(Uri uri = new Uri(string(包含”http://”)))

**CheckBox**：

多选

属性：

外观：Checked

**RadioButton**：

单选：需要分组(使用GroupBox控件)

属性：

外观：Checked

**GroupBox**：

**PictureBox**：

属性：

行为：SizeMode

外观：Image(以Image.FromFile(“图片路径”)赋值)

**ComboBox**：

属性：

数据：Items(对象.Items.Add()添加数据，当添加的是类对象，显示信息是string类型的信息，对象.Items.Clear()清除数据)

外观：DropDownStyle

事件：

SelectedIndexChanged

对象.SelectedIndex(当等于0时默认选中第一行)

对象.SelectedItem获取object类型对象

**ListBox**：

属性：

数据：Items(对象.Items.Add()添加数据，对象.Items.Clear()清除数据)

事件：

Click

对象.SelectedIndex获取选中项下标

**Panel**：

自己内部的控件跟随自己显示和隐藏

**DataGridView**：

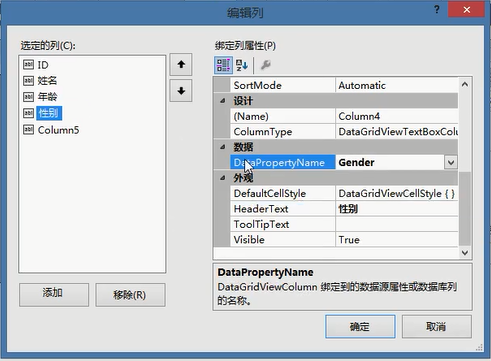
属性：

外观：RowHeadersVisible(不显示行的头，取消显示第一列)

行为：SelectionMode(选中一行或者选中一个)，ContextMenuStrip(将右键菜单与该控件关连)，MultiSelect(是否多行选择)

事件：

CellClick，Click(单机组件时发生)，SelectionChange(所选内容发生变化)，doubleClick



对象.DataSource = list集合名

对象.SelectedRows.Count

对象.SelectedRows[0].Cells[0].Value(选中行的第一行)

对象.ClearSelection()(取消选中行)

**ContexMenuStrip**：

将右键菜单与相应的控件关联

**PropertyGird：**

SqlConnectionStringBuilder conn = new SqlConnectionStringBuilder()

对象.SelectedObject = conn

GDI：

Graphics 对象 = 窗体对象.CreateGraphics()(通过窗体对象创建一个GDI对象)

Point 点对象 = new Point(x,y)

Pen 笔对象 = new Pen(Brushes.Yellow)

对象.DrawLine(笔，点1，点2)

对象.DrawRectangle(笔，new Rectangle(new Point(x,y)，new Size(50,50)))

对象.FillRectangle(Brushes.Yellow，new Rectangle(new Point(x,y)，new Size(50,50)))

对象.DrawString(“字符串”，new Font(“宋体”，20，FontStyle.Bold)，Brushes.Yellow，new Point(x,y))( 在窗体重绘的事件中重画)

绘制验证码：

Bitmap 图片对象 = new Bitmap(x,y)

Graphics 对象 = Graphics.FromImage(图片对象)

对象.DrawString(，，new solidBrush(Color.\*\*)，)

图片对象.SetPixel(x,y,Color.Black) // 在图片上画像素颗粒

PictrueBox对象.Image = 图片对象

多线程：

进程：每一个应用程序都是一个进程，进程是由多个线程组成的

Process[] 进程数组名 = Process.GetProcesses()(获得当前进程)

进程名.kill()

Process.Start(“calc/mspaint(命令)”)进程的形式启动应用程序

通过Process打开指定的文件

ProcessStartInfo 对象 = new ProcessStartInfo(“文件路径”)

Process.Start(对象)

线程：

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关闭，自动关闭后台程序

Thread 线程名= new Thread(方法名)(方法中不允许访问主线程中的控件，在主线程加载时使用Control.CheckForIllegalCrossThreadCalls = false不进行跨线程检查，此时当主线程关闭而新线程还没有关闭就会抛异常，所以当主线程正在关闭时将新线程关闭(主线程中的FormClosing事件中关闭新线程，如果线程名不等于null，线程名.Abort()，线程一旦被关闭了就不能再打开了)

线程名.IsBackground = TRUE

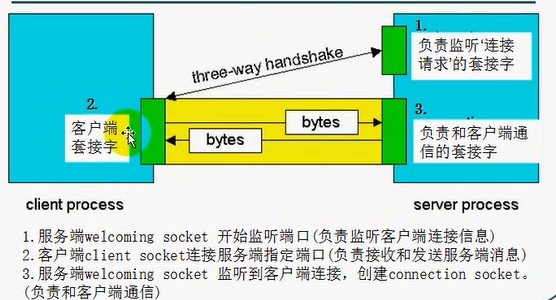
线程名.Start(线程执行方法的参数，必须是object类型)

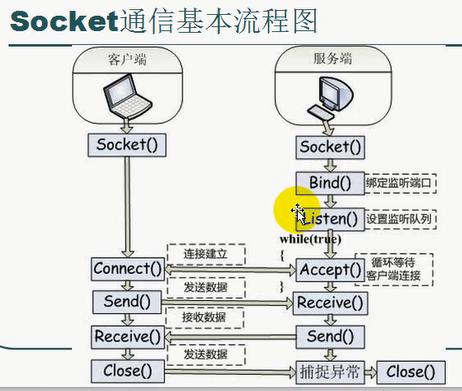
线程名.Sleep(一段时间)

Socket：

程序通过Socket(电话)通信，HTTP协议(规定好语言)

客户端连接服务器，通过IP地址只能找到服务器，通过端口号找到应用程序(电话号码)，客户端只有一个Socket，服务器需要一个负责监听的Socket和一个负责通信的Socket，(传输协议：TCP(流式Socket)，UDP(数据报式Socket))





服务器：

Socket 监听对象 = new Socket(AddressFamily.InterNetwork，SocketType.Stream，ProtocloType.Tcp)

IPAddress IP地址= IPAddress.Any(IPAddress.Parse(txtServer.Text))

IPEndPoint 端口号= new IPEndPoint(IP地址，Convert.ToInt32(txtPort.Text))

监听对象.Bind(端口号)

监听对象.Listen(个数)(服务器允许客户端连入的最大个数)

将以下语句放在循环中不断监听，为防止主线程卡住放在新线程中：

Socket 通信对象 = 监听对象.Accept()(一直等待客户端的连接)

通信对象.RemoteEndPoint(获得客户端的IP地址和端口号)

将以下语句放在循环中不断接受字符，为防止主线程卡住放在新线程中，直到接收的字符个数为0时跳出循环：

int = 通信对象.Receive(字节数组)、

接收的数据是文字还是文件，此时就需要协议(字节数组前加标识，创建一个byte类型的集合，添加元素之后再将集合转为数组)

通信对象.Send(字节素组)

当有多个客户端连接时将客户端的信息存储(字典键是字符串，值是通信的Socket)

客户端：

Socket 对象 = new Socket(AddressFamily.InterNetwork，SocketType.Stream，ProtocloType.Tcp)

IPAddress IP地址= IPAddress.Any(IPAddress.Parse(txtServer.Text))

IPEndPoint 端口号= new IPEndPoint(IP地址，Convert.ToInt32(txtPort.Text))

对象.Connect (端口号)

将以下语句放在循环中不断接受字符，为防止主线程卡住放在新线程中，直到接收的字符个数为0时跳出循环：

int = 通信对象.Receive(字节数组)

接收的数据是文字还是文件，此时就需要协议(字节数组前加标识，创建一个byte类型的集合，添加元素之后再将集合转为数组)

对象.Send(字节素组)

委托：

将方法作为参数传递时的类型就是委托，委托指向的就是方法

delegate 返回值 委托名(参数1，参数2)

泛型委托：

delegate 返回值 泛型委托名<T>(T 参数1，T参数2)

委托调用时类型就会知道了，会免去使用object类型时拆箱和装箱的麻烦

当传递不同类型的参数时使用泛型委托

where T : class/new()/UserInfo(必须是类或构造函数或继承或实现某个类或接口)

匿名函数：

当函数使用次数比较少时，使用匿名函数,直接用变量名作为函数名使用

委托名 变量名= delegate(参数类型 参数名)

｛

return 返回值;

｝

lamda表达式本质上是匿名函数，匿名函数实质是函数

委托名 变量名 = (参数名) => (return 返回值)

泛型集合中委托的参数传递：

list.RemoveAll(n=>n>2)(除了n还可以使用其它的变量)

IEnumerable<int> 集合名= list.Where<int>(n => {return n > 5})

多播委托：

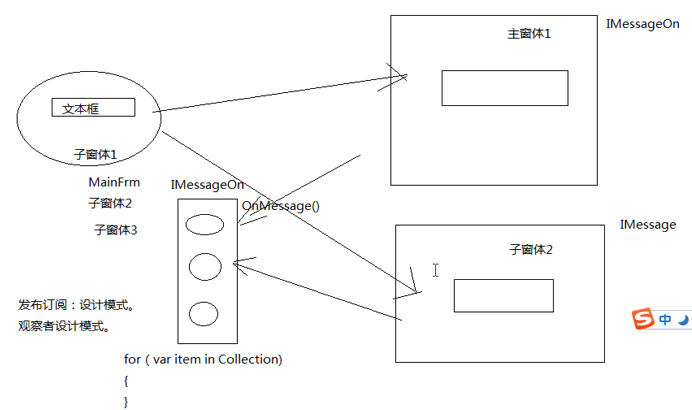
委托可以指向多个函数，委托变量名 += 函数名，如果最后委托变量名 = 函数名只会调用一个，这时需要用事件解决

委托的应用：

类函数接收参数用于另外函数(构造函数)或不同的类传递参数(窗体的属性赋值)时需要类本身的字段存储

窗体传值：子窗体利用委托在创建实例时接收主窗体在创建子窗体时在构造函数中传递的函数

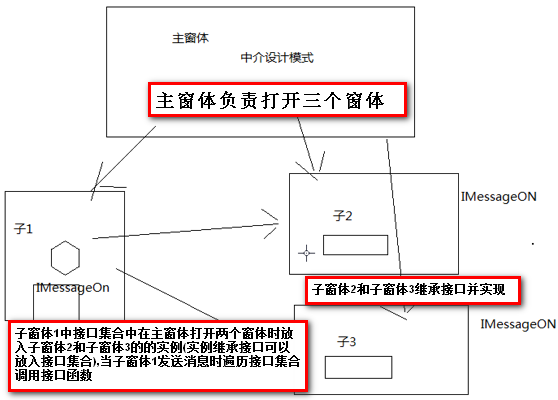
子窗体的值变化时显示到主窗体上：主窗体在创建子窗体时将主窗体的实例传递过去,此时主窗体的控件是私有的，只能在主窗体中创建函数，然后在子窗体中调用此函数，使子窗体的内容传值到主窗体中。这种做法当需要传值到多个窗体时就会比较麻烦而且窗体之间的控件有关联，需要子窗体变化时不再通知，有以下做法



发布者模式：

中介者模式配合观察者模式可以实现观察者和被观察者彻底解耦

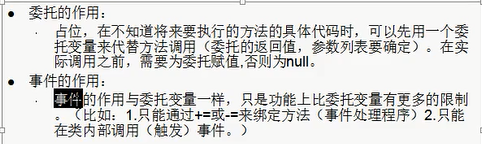
除了接口方式实现，也可以使用委托和事件的方式实现



事件：

事件就是给委托外面穿上了衣服，对委托有保护的作用，委托是不安全的，委托可以在任何地方调用，但是事件只能在类内部调用，是类型安全的委托

事件比委托多了event关键字，并且需要使用+=进行赋值



反射：

程序集:.Net中的dll和exe文件都是程序集，可以看作是一堆相关类打一个包，包含资源文件，类型元数据，IL代码，程序集可以减小程序的尺寸，也可以封装一些代码，提供必要的访问接口，反射就是动态获取元数据

Type是对类的描述

string path = Path.Combine(AppDomain.CurrentDomain.BaseDirectory(exe文件所在的路径)，”Common.dll”)

Assembly 程序集文件 = Assembly.LoadFile(path)(加载程序集文件)

Type[] types = 程序集文件.GetExportedTypes()(公开的数据)

Type[] types = 程序集文件.GetTypes()(公开和不公开的数据)

foreach(Type item in tyeps)

{

item.Name(类的名称)

item.Namespace(命名空间)

item.FullName(全名)

}

Type t = 程序集文件.GetType(“类库.类名”)

Type t = typeof(类名)

Type t = 对象.GetType()

// 动态创建对象

object 对象 = 程序集文件.CreateInstance(“类库.类名”)(调用了person默认的构造函数)

object 对象= Activator.CreateInstance(t，(构造函数的参数))

// 获得数据源的属性数组

PropertyInfo[] pros = 对象.GetType().GetProperties()

foreach(PropertyInfo item in pros)

{

item.Name(属性名称)

}

// 获得数据源中的方法

MethodInfo[] mds = 对象.GetType().GetMethods()

foreach(MethodInfo item inmds)

{

item(方法名称)

}

// 执行函数

MethodInfo md = 对象.GetType().GetMethod(“函数名称”)

md.Invoke(对象，函数参数)

bool b = typeof(父类名).IsAssignableFrom(typeof(子类名))(子类可以赋值给父类)

bool b = typeof(类名/父类名/接口).IsInstanceOfType(实例)(对象是否是当前类的实例)

bool b = typeof(子类).IsSubclassOf(typeof(父类))(当前类是否是父类的子类，不适用接口)

bool b = typeof(类名).isAbstract(当前类是否为抽象类)

HTML：

网页中的内容，是一种超文本标记语言，有很多标签

<html>

<head>

<title>标题</title>

</head>

<body text = “yellow(文本颜色)” link = “Black(链接颜色)” alink=”(点击链接时的颜色)” vlink = “(点完之后的颜色)” bgcolor = “(背景颜色)” background = “(背景图片)”>

<p>段落</p>(各个段落之间有间隔)

<br/>(换行的语句之间没有间隔)

&nbsp;(表示空格)

&gt(>);

&lt(<);

&amp(&);

<h1>一级标题</h1>

<h2>二级标题</h2>

<!—要注释的内容 -->

<hr/>(水平线)

<img src = “图片路径(相对路径是相对于html文件位置的路径)” width = “300px” height = “300px” border = “10px(表示边框)” alt = “图片无法显示(图片不能显示的文本)” title = “哈哈(当鼠标移至图片显示的文本)”(属性赋值，以空格相隔)/>

<font color = “red” size = “6” face = “华文行楷”></font>

<marquee direction = “right(图片往右走)” behavior = “slide(走至边缘停止)”>

<a href = “http://www.baidu.com(链接到地址)” target = “\_self(当前页面跳转)/\_blank(新页面跳转)”>百度</a>

<a name = “顶端” href = “#底端”>回到底端</a> <a name = “底端”>底端</a>(页面内的跳转，对于不同页面的跳转只需要在href中的添加文件名#底端即可)

<a href = “mailto:2998926207@qq.com”>发邮件</a>

物理字体：

<b>粗体

<i>意大利斜体

<em>倾斜

<tt>打字机文本等宽

<u>下划线

<s>删除线

<sup>上标

<sub>下标

逻辑字体：

<pre><code>(预定义格式文本(保留空格和换行)，计算机文本)

HTML列表：

<ul type = “disk/circle/square(实心/空心/方块)”>

<li></li>(每一项)

</ul>(无序列表，ol表示有序列表(type = “A/I”))

<dl>

<dt></dt>

<dd></dd>(比dt缩进一些)

</dl>(自定义列表)

HTML表格(布局)：

<table border = “1px” width = “100px” height = “100px” cellspacing = “(格子间隙)” cellpadding = “(单元格中文本与边框的距离)”>

<tr>

<td align = “left/right/center” valign = “top/bottom/middle”>单元格</td>(行元素单元格)

<th></th>(黑体居中加粗，一般作为第一行出现)

</tr>(行)

</table>

跨行跨列的表格：

<table border = “1px” width = “100px” height = “100px” cellspacing = “(格子间隙)” cellpadding = “(单元格中文本与边框的距离)”>

<tr>

<td >单元格</td>

<td colspan = “3(跨3列)”>单元格</td>

</tr>(行)

<tr>

<td rowspan = “3(跨3行)”>单元格</td>

<td>单元格</td>

</tr>(行)

</table>

表单(向服务器传输数据)：

<form action = “<http://www.baidu.com(发送的地址)>” method = “get(发送的方式，url方式)/post(报文)”>

用户名：<input type = “text” name = “txtName(会将名字传给提交后的网站在url中显示)”/><br/>

密码：<input type = “password” name = “txtPwd”/><br/>

<fieldset>

<legend>性别：</legend>

男：<input type = “radio” name = “sex”/>

女：<input type = “radio” name = “sex”/><br/>

</fieldset>

<fieldset>

<legend>婚姻状况：</legend>

未婚：<input type = “radio” name = “married/”>

已婚：<input type = “radio” name = “married”/><br/>

</fieldset>

<fieldset>

<legend>爱好：</legend>

吃饭：<input type = “checkbox” /”><br/>

睡觉：<input type = “checkbox” /><br/>

打豆豆：<input type = “checkbox” /><br/>

</fieldset>

<select size = “4(显示项个数)”>

<option>河北省</option>

<option>河南省</option>

<option>湖北省</option>

<option>湖北省</option>

</select>

<select size = “4(显示项个数)”>

<optgroup lable = “河北省”>

<option>石家庄</option>

<option>张家口 </option>

</ optgroup >

</select>

照片：<input type = “file”><br/>

<textarea cols = “40%” rows = “60%” disable = “disable”>

</textarea>(协议内容)

<input type = “submit” value = “提交” />

<input type = “reset” value = “清空”/>

</form>

</body>

</html>

分层页面：

在承载页面中代码：

<head>

</head>

<frameset rows = “15%(左右分),\*”>

<frame src = “top.html” noresize = “noresize(页面间不动)”>

<frameset cols="30%(上下分),\*">

<frame src="left.html" noresize="noresize"/>

<frame src="right.html" name="Main"/>

</frameset>

</frameset>(代替body)

在左边页面的代码(在左边的页面上放上一个链接界面显示在右边页面上)：

<head>

</head>

<body>

<a href = <http://www.baidu.com> target = “Main”>百度</a><br/>

</body>

CSS(层叠样式表)：

网页中内容的样子

ctrl + j智能提示

将CSS写入HTML文件中的三种方式：

内联样式表，嵌入样式表，外部样式表

内联样式表：

<p style = “background-color:red;font-size:xx-large”>微笑</p>

嵌入样式表(内联样式表的优先级高)：

在head标签内添加样式标签

<style type = “text/css”>

p{

background-color:yellow;

font-size:xx-small;

}

</style>

外部样式表(与嵌入样式表谁后执行是谁)：

在项目里面添加样式表，在样式表Test.css中添加

p{

background-color:yellow;

}

然后在\*\*.html文件的head标签中添加

<link rel = “stylesheet” type = “text/css” href = “Test.css”/>

样式选择器：

获得页面上要设置样式的元素

HTML选择器：嵌入样式表或外部样式表

Class选择器：通过设置class属性区别

<p class = “c1”>微笑</p>

<p class = “c2”>微笑</p>

在嵌入样式表中格式

p.c1{

background-color:yellow;

}

p.c2{

background-color:red;

}

ID选择器：通过id属性设置(id尽量一致，如果想两个相同可以用Class选择器)

<p id = “p1”>微笑</p>

<p id = “p2”>微笑</p>

在嵌入样式表中格式

#p1{

background-color:yellow;

}

#p2{

background-color:red;

}

关联选择器：可以用class或id来区别嵌入的标签，但是也可以通过访问

<em>微笑</em>

<p ><em class/id = “em1”>微笑</em></p>

<p >微笑</p>

在嵌入样式表中格式

em{

background-color:yellow;

}

em.em1/#em{

background-color:red;

}

p em{

background-color:black;

}

组合选择器：

<table border = “1px” width = “100px” height = “100px” cellspacing = “” cellpadding = “”>

<tr>

<td>单元格</td>(行元素单元格)

</tr>(行)

</table>

<h1>

<h2>

在嵌入样式表中的格式

h1,h2,td{

background-color:black;

}

伪元素选择器：

改变标签的部分内容

在嵌入样式表中的格式

A：active{

background-color:black;(背景颜色)

color:yellow;(字体颜色)

}(选中的状态)

A：hover {

font-size:xx-large;

}(鼠标移上的状态)

A：link {

background-color:black;(背景颜色)

}(访问前的状态)

A：visited {

background-color:black;(背景颜色)

}(访问过的状态)

P：first-line {

font-size:xx-large;

}(段落中的第一行文本)

P：first-letter {

font-size:xx-large;

}(段落中的第一个字母)

CSS样式属性：

字体：

font-family(设置字体)

font-size(设置大小)

color (设置颜色)

font-style(正体斜体)

text-decoration(下划线上划线删除线)

font-weight(是否加粗)

背景：

background-color(背景颜色)

background-image:url(images/1.jpg)(背景图片)

background-repeat(只在横向或纵向重复或不重复)

background-attachment(背景图像是否滚动)

background-position(显示位置)

文本：

word-spacing(单词(有空格才会有单词)间距)

letter-spacing(字符间距)

text-align(水平对齐方式)

text-index(第一行文本缩进)

line-height(行高)

位置：

position(绝对位置，固定位置(仍然是绝对位置但是会固定)，这个位置可以脱离文档流)

z-index(值最大先显示)

top/left/right/bottom(离边缘的距离)

width/height

盒子：

盒子与盒子内元素距离属性padding(padding-top)

盒子与盒子之间距离属性margin(margin-top)

<div style = “display:inline”></div>(默认占一层，通过改变display的属性变成和span一样)

<span style = “display:block”></span >

\*{ margin :0px;font:12px}

margin:0px,auto居中；float :right;右边

导航div：无序列表(list-style-type :none(无点);float:left(将单元格飘起来横向排列);)

设置元素边框属性：

border-top-width

border-top-color

border-top-style

XML：

可扩展的标记语言，自定义标签

HTML：显示网页内容

XML：存储数据，小型数据库

XML文件只能有一个根节点，节点中包含元素

XmlDocument 对象 = new XmlDocument()

XmlDeclaration 第一行数据= 对象.CreateXmlDeclaration(“1.0”,”utf-8”,null)

对象.AppendChild(第一行数据)

XmlElement 根节点名=对象.CreateElement(“根节点”);

对象.AppendChild(根节点名)

XmlElement 子节点名=对象.CreateElement(“子节点”);

根节点名.AppendChild(子节点名)

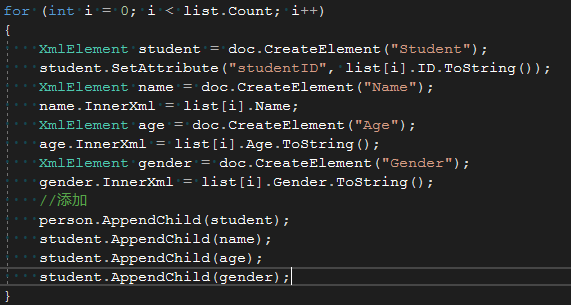
XmlElement子节点中的节点名=对象.CreateElement(“子节点中的节点”);

子节点中的节点名.SetAttribute(“属性名”，”赋值”);(设置属性并赋值)

子节点中的节点名.Innertext(会将内容中特殊的字符转义)/InnerXml = “”;

子节点名.AppendChild(子节点中的节点名)

通过循环List集合获得所有对象，以节点的形式添加到XML文档中：



对象.Save(“文件名”)

Xml文档增删改查：

XmlDocument 对象 = new XmlDocument()

if(File.Exists(“文件名”))

{

对象.Load(“文件名”)

XmlElement 根节点名 = 对象.DocumentElement

// 在根节点中创建子节点

// 寻找子节点

XmlNodeList 子节点列表名 = 根节点名.ChildNodes

foreach(XmlNode item in 子节点列表名)

{

Console.WriteLine(item.InnerText)

}

// 寻找子节点下的子节点

XmlElement items = 根节点名[“子节点”]

XmlNodeList 子节点下的子节点列表名 = items.ChildNodes

foreach(XmlNode item in子节点下的子节点列表名)

{

Console.WriteLine(item.Attributes[“属性名”].Value)

//也可以对值进行更改

}

//根据不同属性 寻找同名的某个节点，使用XmlPath方式读取

XmlNode 某个子节点下的子节点名 = 根节点.SelectSingleNode(“/根节点/子节点/子节点下节点[@属性名=’属性值’]”)

//根节点不能删除，可以删除根节点下的所有节点

根节点名.RemoveAll()

// 删除指定的子节点下的子节点

子节点名.RemoveChild(子节点下的子节点名)

// 删除指定的子节点下的子节点的属性

某个子节点下的子节点名.Attributes.RemoveNamedItem(“属性名”)

//更改节点的值

某个子节点下的子节点名[“下的节点名”].InnerText = 赋值

}

else

{

创建文件再添加

}

对象.Save(“文件名”)

将文档转换为对象

