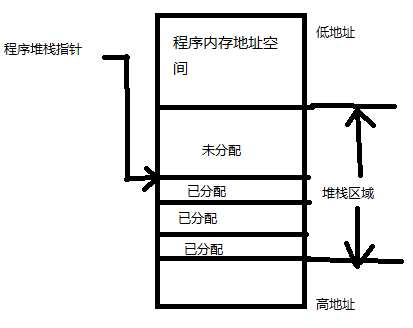
框架基础,语法基础,字符串&集合&流,常见类和接口;  
  
02.net基础（重点）  
-第一部分 框架基础  
1.基础概念  
CTS(Common Type System),CLS(通用语言规范),CLR(Common Language Runtime);  
环境基础-.net framework;  
程序集(Assembly)，AppDomain:[进程  [CLR  [AppDomain  ]  ]  ]  
  
2.运行机制  
\* C#--csc.exe-->IL--JIT-->机器码（关键理解：在开发环境(部署时缓存)/生产环境(jit)生成机器代码）  
概念补充：jit指令与jmp指令  
工具ILDasmde  
[PE/coff头  [CLR头  [中间代码，资源数据，静态数据池，元数据表，程序集清单]]]  
\* 程序集加载  
顺序:应用程序版本策略->CODEBASE中的位置->应用程序域->应用程序目录.  
a = Assembly.loadfrom()/load(强命名);  a.CreateInstance();  
\*配置版本策略：应用程序策略->发行者策略(针对放入GAC的程序集)->计算机策略;  
  
3.生成，部署，管理  
\*强签名程序集：带有公匙和数字签名的程序集;包括4个元素:文件名，版本号，语言文化，公钥;  
注：.net也是用非对称秘钥加密的。  
\*GAC(Global Assembly Cache): c:\windows\assembly\,注意必须是带有公钥的强签名程序集;  
\*延迟签名:把强签名程序的私钥加密和数字签名延迟到世纪发布时进行;  
  
4.名企习题  
\*使用C++编写的程序是否能在.net上跑?可以引用dll,import;  
\*什么是受托管的代码?.net平台帮你管理内存释放的代码，也叫垃圾回收;  
\*什么是应用程序域，他与进程有什么区别?Appdomain是CLR中提供代码运行范围，错误隔离，安全设置隔离的逻辑单元。一个或多个appdomain在os进程中运行;  
\*强签名和弱签名的区别?强签名包含公钥，数字签名，可放入GAC；  
\*如何单独升级系统中的某一程序集?如何表达。  
\*公钥密钥的概念和作用?加密学，不对称加密算法，私钥加密，公钥数字签名;  
\*程序集放入GAC的好处?方便共享，保证程序集能够被CLR找到并加载;  
\*,net采用什么技术解除dll hell?com与.net的区别，.net使用元数据和逻辑类型解决此问题;注:答案不太满意;  
\*编译时如何指定版本?app中的AssemblyInfo.cs;  
\*延迟签名的作用?方便开发测试;  
  
-第二部分 语法基础  
1.基本类型与语法  
System.Object的方法:Equals()，GetHashCode(),GetType()涉及反射,ToString(),ReferenceEquals(),MemberwiseClone(),Finalize();  
\*值类型与引用类型的差别  
System.Object->System.ValueType(其他则为引用类型):赋值是的区别，内存分配的区别，继承结构的区别;  
\*拆装箱以及避免拆装箱  
拆箱时注意InvalidCastException;  
避免的情形：值类型格式化输出(使用如toString()),System.Object类型的容器(如ArrayList,使用泛型);  
\*全局变量的替代者：公共静态变量;  
\*struct与class的区别：struct不能继承，不能定义无参构造方法，自动初始为0并不能在定义时设置初始值;  
\*类型初始化器的概念:具有和类型相同名字，无参数返回值且以static定义的方法。.cctor(),调用有2中策略,BeforeFieldInit(拥有效率高);  
\*参数传递：ref,out;params,允许方法在定义时不确定参数的数量,注意申明params后，不在允许有其他参数;  
\*string与String完全相同，别名，尽量统一;  
\*.net支持的访问级别以及不同类型成员默认的访问级别：  
Private             priavte  
Family              protected  
Assembly            internal  
Family&Assembly     未实现  
Assembly or Family  protected internal  
Public              public  
---------------------------------------  
Enum                public  
Class               private  
Interface           public  
Struct              private  
\*属性的特点以及属性与方法的异同:本质相同，属性方便扩展;  
\*C#中的深复制和浅复制：MemberwiseClone()就是浅复制--复制原始对象中的所有非静态值类型成员和所有的引用类型的引用;可以使用IClonable接口Clone()方法来实现你所需要的复制，但需要注意可被继承的类型需要谨慎的实现此接口;  
\*C#中循环的语法：while,do...while,for,foreach（优点，健壮性，不足，不能改变成员的值）；  
\*C#using与非托管资源的释放:任何带有非托管资源的类型，都有必要实现IDisposable的Dispose方法,using语句提供一个高效调用对象Dispose()的方法;  
对应的有:try/catch/finally;  
  
2.内存管理和垃圾回收  
\*堆栈与堆的异同:比如一个32位的OS，4GB的虚拟内存空间呢的开辟--堆栈，托管堆，非托管堆;  
[](http://blog.photo.sina.com.cn/showpic.html#url=http://album.sina.com.cn/pic/0024gZI6gy6GVbNYc5e84)  
  
\*执行string abc="aaa"+"bbb"+"ccc"共分配了多少内存？有意思，编译器优化,与字符串池的关系.  
\*.net中GC的运行机制：通过算法找到不再使用的对象，移动对象使所用仍被使用的对象紧靠托管堆的一边和调整各个状态变量;尽量避免使用GC.Collect来执行垃圾回收;  
Finalize()方法类似与析构函数，但CLR内部实现机制比较复杂，包括待析构对象表，等待析构表等结构;  
Dispose() 与Finalize()组成的设计模板:注意Dispose(true);GC.SuppressFinalize(true)--通知.net对象被回收时不用调用Finalize方法，改善性能;  
小贴士:Dispose方法被使用者主动调用，而Finalize方法在对象被垃圾回收的第一轮回收后，由一个专用的.net线程调用;  
\*GC中代的概念：GC机制按照对象不被使用的可能性把托管堆内对象分为3代：0代，1代，2代,越小的代获得越多的释放机会，而每一次GC中仍存活的对象实例将移到下一个代上；  
\*GC如何判断一个对象是否仍然在被使用:  
区分根引用，非根引用;GC使用根引用遍历所有引用对象的实例，当一个对象不能被遍历时，将被视为不能再被使用；同时，当根引用遍历抵达一个已经被视为在被使用的对象时，将结束这一分支遍历，避免死循环;  
\*.net内存托管堆中是否存在内纯泄露：大对象的分配（String）,不恰当的保存根引用，错误的finalize方法;  
  
3.面向对象的实现  
\*C#中最多只有一个父类，但可以实现多个接口;  
\*重写，重载，隐藏的概念:  
重写和隐藏:定义一个基类和两个子类，在基类中定义一个虚方法:getString(),两个子类分别实现重写和隐藏;  
当使用和对象类型相符的引用去调用对象方法时，结果相同;但当通过基类的引用去调用对象内的方法时，重写和隐藏就完全不一样了;重写(override)仍然能找到定义在对象真正类型中的getString(),而隐藏(new)则调用的是基类中的方法;  
重载是拥有相同的名字和返回值的方法拥有不同的参数列表，是实现多态的理想方法;  
\*为什么构造方法中调用虚方法会导致问题?  
类型构造方法的调用顺序:  
son的初始化表达式->father的初始化表达式->father构造方法->son构造方法;  
构造中调用虚方法产生的问题(理解的不够):当虚方法在基类的构造方法中被调用时，他的类型仍然保持子类类型，子类的虚方法将被执行，而这时子类的构造方法还没完成，任何对于子类构造成员的访问将产生异常;避免方法，永远不要在非叶子类的构造方法中调用虚方法;  
\*声明一个类不能被继承(sealed);继承带来的3个问题:  
为了让派生类型能顺利序列化，非叶子类需要实现恰当的序列化方法;  
当非叶子类实现ICloneable等接口时,意味所有派生类被迫也需要实现接口中方法;  
非叶子类的构造方法中不能调用虚方法，也不应该将this指针传递给其他对象和方法;  
  
4.异常的处理  
\*针对不同的异常进行捕获;  
try,catch(nullreferenceException){handleExpectedException(可以当前类处理);throw ex(可以抛到上层处理);handleCrash(){system.threading.thread.currentThread.Abort();}};  
注意异常一定要继承自exception;养成分别处理异常的习惯;  
\*使用conditional特性:debug与release,debug.Assert();  
用于编写特定编译版本中运行的方法[Conditional("DEBUG")]  
\*避免类型转换的异常;使用as,is(注意as性能更高，减少一次尝试操作);  
  
5.面试真题  
\*什么情况下使用虚方法？它与接口有什么不同？当它的派生类(一般为多个，比如图形的draw,圆形，矩形的draw)具有与之不同实现的时候;它与接口不同时它是派生类型都具有的方法;虚方法时一个纵向的概念，而接口是个横向的概念;  
\*值类型和引用类型的不同？在内存中的分配空间不同，前者是堆栈，后者是堆;  
\*怎样理解静态变量?它是一种全局变量的替代，如记录类的对象被创建的次数等;如asp.net程序中的application;  
\*如何是一个类型可以在foreach中使用:实现IEnumerable接口;  
\*new的用法:new一个新的对象和调用构造函数;隐藏基类的方法时使用 public new void tostring(),用于在泛型声明中约束可能的类型参数;  
\*sealed修饰的类:不能被继承;  
\*C#中能否直接对内存进行操作?(可以，默认情况下不能，但通过添加unsafe关键字，可以定义可使用指针的不安全上下文);  
#.net错误处理机制是什么？应该是异常处理，try,catch...（针对不同的exception使用不同的catch,针对不同的异常进行不同的捕获）,finally;  
#面向对象语言具有那些特性?封装，继承，多态;  
#.net垃圾回收的机制？gc,分代;通过算法找到不再被使用的对象，移动对象使仍在被使用的对象集中在托管堆的一边(即上一代,generation)和调整各个状态变量;析构方法的运行，调用finalize();  
  
#类成员有哪几种访问性?public ,protected,private;internal(assemble),internal protected(assemble&family);  
#什么使用Assert,在debug程序中，需要调试，回显提示信息是使用;  
  
-第三部分 字符串，集合，流  
1.字符串处理  
\*System.string:是引用类型，它的对象再初始化后不能进行任何的修改，任何修改字符串的操作都会导致一个新的字符串的产生;  
\*system.stringBuilder:因为string类型的不可修改性，为了性能的需要，通过使用构造器设计模式(解决复杂对象的创建问题),同时stringBuilder还可用于与非托管代码的交互;  
\*string与byte[]对象的转换;  
bit,字节，编码;  
编码:utf-8,gb2312,utf-7,unicode;  
uft-8:字符采用1-6个8bit字节的序列进行编码。设字节数为n:若n=1,字节高位为0，其他7位编码;若n>1,则初始字节高n位为1，下一位为0，剩余位数用于编码，剩下字节高位为1，下一位为0，余下进行编码;  
\*BASE64编码:用于混淆8字节流，但不属于加密;  
算法，分开所有位，再在高2位填0，接着填充低6位;  
try{Convert.ToBase64String(bytes),fromBase64String(base64)}catch{}  
\*SecureString的分配和释放:特殊场合，安全字符串能起很大作用;  
使用原因:字符串即使加密，也在内存中驻留很久了;  
public unsafe static void PrintSecureString(SecureString ss){  
char\* buffer=null;//C#中的指针，很好的例子  
try{  
//只能逐字符访问SecureString  
buffer=(char\*)Marshal.SercureStringToCoTaskMemUnicode(ss);  
for(int i=0;\*(buffer+i)!='\0';i++)  
console.Write(\*(buffer+i));  
}finally{  
if(buffer!=null)  
Marshal.ZeroFreeCoTaskMemUnicode((System.IntPtr)buffer);  
}  
}  
\*什么是字符串池机制?  
使用原因，因为字符串的不可变性，程序中若重复使用相同字符串则会消耗大量内存;  
在CLR启动时，会创建一个字符串容器，容器的键就是字符串的内容，值是字符串的引用;使用字符串时，先在字符串池中查找，若存在相同值，则直接返回其引用，否则创建该字符串并返回引用;  
该机制可以通过程序集元数据进行控制;System.Runtime.CompilerServices.CompilationRelaxtionsAttribute/NoStringInterning；  
  
2.常用集合和泛型  
\*int[]是引用类型还是值类型?必须是引用，数组是一族类型，继承自system.array;  
\*数组之间如何转换？  
类型转化机制:包含值类型的数组不能隐式转换为其他任何类型;两个数组能相互转换的前提是为数相同;[,]与{"a","b"}，ArrayTypeMismatchException;  
内容转换时:使用Array.ConvertAll(-,-(会用到匿名对象，委托等));  
\*泛型(也被称为开放式类型，不能被实例化，建议参数名用T开头);  
注意:为开放的类型提供泛型的实例导致新的封闭类型的生成，但这不代表新的封闭类型和开放类型有任何派生继承关系，实际上，两者处在同一层次;  
  
\*什么是泛型的主要约束和次要约束?  
每个泛型参数可以之多有一个主要约束，泛型的主要约束指泛型参数必须是或者继承某一引用类型，有两个特殊的约束:class,struct,分别代表泛型参数是引用类型或者值类型;  
每个泛型参数可以有无限个次要约束，它规定泛型参数必须实现所有的次要约束指定的接口;  
\*.net是否可以使用标准模板库;  
STL:(algorithm,container,iterator)wintellect团队提供;  
  
3.流和序列化（重点）  
\*什么是流，.net中有哪些流?  
  
流的使用举例:(注意C#默认访问级别class,method,field分别为internal,private,private)  
class UseStream{  
//  
static Byte[] ReadAllBytes(Stream stream, int bufferlength){  
byte[] buffer = new byte[bufferlength];  
list result = new List();  
int read = 0;  
while(read = stream.Read(buffer, 0, bufferLength) > 0)  
{  
//读到最后  
if(read < bufferlength){  
byte[] temp = new byte[read];  
Array.Copy(buffer, temp, read);  
result.addRange(temp);  
}else result.addRange(buffer);  
}}  
//  
static void WriteAllBytes(Stream stream, Byte[] data,int bufferlength){  
byte[] buffer = new byte[bufferlength];  
for(long i=0;i  
int len = bufferlength;  
//写到最后  
if(i+bufferlength>data.LongLength)  
len = (int)(data.Longlength - i);  
  
Array.Copy(data,i,buffer,0,len);  
stream.Write(buffer,0,len);  
}}  
//  
private const int bufferLength =1024;  
static void main(string[] args){  
String filename="C:\\TestStream.txt";  
String fileContent=GetTestString();  
try{  
//创建并写入文件  
using(FileStream fs = new FileStream(filename, FileMode.Create)){  
byte[] bytes=Encoding.Default.GetBytes(fileContent);  
WriteAllBytes(fs, bytes, bufferLength);  
fs.close();  
}  
//读取文件并打印  
using(FileStream fr = new FileStream(filename, FileMode.Open))  
{  
byte[] result= ReadAllBytes(fr, bufferlength);  
console.Write(Encoding.Default.GetString(result));  
}}  
finally{  
try{  
if(File.Exists(filename))  
File.Delete(..);  
finally{console.Read();  
}}}}  
//  
static String GetTestString(){  
StringBuilder builder = new ..  
for(int i=0;i<10;i++)  
builder.Append("这是测试数据\r\n");  
}}  
注意:Stream类型均实现了IDisposeable接口，可用using确保Dispose方法被调用;  
\*如何使用压缩流:SharpZipLib组件算法更好,System.IO.GZipStream/DeflateStream的系统自带但算法效率较低;  
\*Serializable特性的作用;思考:流类型可以方便的操作各种字节流，但现有实例如何转化为字节流呢?  
对象实例的序列化:指把实例对象转换为可方便存储、传输、交互的流;在.net中，当一个类型被申明为Serializable,它就能被诸如BinaryFormatter等实现了IFormatter接口的类型对象进行序列化和反序列化.  
[Serializable][NonSerializable]  
例子:  
class UseSerializable{  
public static byte[] Serialize(MyObject obj){  
IFormatter format = new BinaryFormatter();  
 using (MemoryStream ms = new MemoryStream())  
{  
format.Serialize(ms, obj);  
return ms.ToArray();  
}}  
public static MyObject DeSerialize(byte[] data){  
IFormatter format = new BinaryFormatter();  
using (MemoryStream ms = new MemoryStream(data)){  
return (MyObject)format.Deserialize(ms);  
}}  
static void main(String[] args)  
{  
MyObject myObj = new MyObject(26, "baoshan");  
Console.WriteLine("before");  
Console.WriteLine(myObj);  
byte[] temps = Serialize(myObj);  
MyObject newObj = DeSerialize(temps);  
Console.WriteLine("After");  
Console.WriteLine(newObj);  
Console.Read();     }  }         
\*.net提供放入可进行序列化类型:BinaryFormatter,SoapFormatter,XmlSerializer  
BinaryFormatter:将可序列化的对象序列化为二进制字节流;  
SoapFormatter则致力于把可序列化的类型序列化为符合SOAP规范的xml文档;  
  
XmlSerializer:必须拥有一个无参的公共构造方法;只能序列化公共成员变量;XmlIgnore属性;创建时需要输入待处理对象的类型,XmlSerializer xs = new XmlSerializer(typeof(MyObject));  
对象->字节流->指定编码的字符串;  
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"  
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"..  
\*如何自定义序列化和反序列化的过程:需要实现ISerializable接口;提供一个特定的再反序列化中构建对象的方法,通常情况下该构造函数因为安 全的原因，不能设定为public，这也是该方法为什么不在接口中定义的原因;GetObjectData和特殊的构造方法有两个参 数，serializationInfo相当于一个哈希表,而StreamContext记录当前流的状态;  
  
5.名企面试真题;  
\*补:接口的一般调用与显式调用的区别?显式调用的益处:隐藏实现;在接口访问的系统中，只能通过接口调用而不是底层的类;  
\*如何把一个数组复制到ArrayList,反过来呢?ArrayList动态数组:方便添加删除元素(顺序表，链表);动态分配大小;实现 ICollection,IList接口;转 换:ArrayList.ToArray(typeof(...))/CopyTo(Array);ArrayList.Adapter(Array);  
\*string str=null与string str=""有什么区别?前者:内存在堆中并未分配空间;后者在堆中分配空间，但空间里内容为空;  
\*StringBuilder的作用:由于.net中string类型一旦初始化就不能改变，如果要进行字符串的大量基本操作(如拼接)，就会创建很多临 时的字符串，大量消耗内存;StringBuilder很好的建解决了这个问题，动态的改变字符串，最终使用toString()方法返回所需字符串;默 认大小为16;  
注意:因为StringBuilder中有一个私有的string类型变量，并且在组装的过程中，内存中的变量会变化，符合非托管代码字符串类型修改内存的特性，可以用于与非托管类型的交互;  
\*泛型的好处:减少拆装箱，提高性能;保证类型安全;  
\*拥有字符串数组{"2","3","4"},如何转换为整数数组:基本遍历转换;Array.ConvertAll(...)  
\*什么是序列化?.net中如何实现序列化?序列化就是讲对象转化为一定格式的字节流，便于跨域操作;标注Serializable特性;使用XmlSerializer类型;  
\*什么是编码?什么是unicode?编码指一种表示字符的形式，国际上有很多种标准如utf-8,unicode,ascII,我国也有GB2312等.unicode是一种国际通用的编码，并且适应性广泛;  
准确版:数字信息和现实信息的转换机制，一种编码定义一种字符集和转换的原则;  
\*BASE64使用场合?主要用于网络传输数据。。如电子邮件？解决网络传输明码的问题，但它不是加密机制;  
\*字符串池是如何提高系统性能的?CLR提供字符串池机制。在CLR启动时，会专门的创建一个存储字符串的容器，也就是字符串池，每当要新建一个字符串 时，会先到池中查找，是否有相同内容的字符串，如有直接返回引用，若没有则新建并返回引用。这样在大量使用重复字符串时，能大量节省内存消耗;  
\*加密字符串和普通字符串的区别?加密字符串在内存中也是密文的，防止被从内存中直接读取(对其操作时，智能逐个字符操作);虽然字符串也可以加密，但它可能在内存中驻留，同时还可用于与非托管代码的交互;  
  
-第四部分 常用类和接口  
\*System.Object:通过ILasm.exe的noautoinherit开关，可以生成不从System.Object继承的类型，这种类型 不是安全的类型，也不提倡使用。但这种机制的存在，是的程序员不能随意的将对象默认为System.Object;我们能做的就是做好捕捉一切异常;  
\*Object的3种比较方法的异同:  
静态方法ReferenceEquals实现了引用的比较，若是值类型，永远不能为True,即使是同一变量，想想拆装箱。静态Equals实现了比较高效的调用实例Equals方法的功能。实例Equals方法是个虚方法，默认的实现是引用比较，实际中可以根据需要重写Equals方法。值类型的基类System.ValueType重写了Equals方法，实现了内容的比较;  
\*重写GetHashCode方法(有难度，暂时定性理解)：  
Object中的GetHashCode的算法保证了同一对象返回统一HashCode，而不同对象返回不同的HashCode,但对于值类型等视内容相同的对象为相等对象时，默认的Gethashcode算法并不正确;  
重写GethashCode必须保证3条:同一对象无论何时都返回统一HashCode;而相等的对象也必须返回相同的值。在此基础上保证HashCode尽量随机分布;  
  
2.时间操作System.DateTime  
\*DateTime如何存储时间:内部存储一个64位的长整型，其中低62位表示时间，代表从0001年1月1日0时0分0秒开始到表示的时间经过的毫微秒数;高2位，存储UTC标志;  
\*DateTime对象和字符串对象的转换:使用ToString可以将一个DateTime对象格式化输出到字符串。相反的使用Parse，TryParse可以从字符串构建一个DateTime对象;例子:  
String s = "2014-3-16 19:50:00";  
DateTime dt;  
if(Datetime.TryParse(s, out dt)){  
Console...(dt);  
}  
\*UTC时间:格林威治标准时间(GMT)，它是在0度经度测量得到的时间。通过ToUniversalTime和ToLocal方法可以在本地时间和UTC时间间转换。两个方法在转换时会相应的设置保存时间的高2位标志，并且转换算法考虑了夏时令;  
注:夏时令，日光节约时间，节约能源；夏时制（Daylight Saving Time：DST）；  
\*IFormattable和IFormatProvider的使用;(硬骨头啊):因为有时无参数的ToString方法不能满足所有格式化的需要，就需要考虑实现IFormattable接口;例子:  
public string ToString(string format, IFormatProvider provider){  
if(provider!=null){  
ICustomFormatter fmt = provider.GetFormat(this.GetType()) as ICustomFormatter ;//this->UseIFormattable:IFormattable  
if(fmt!=null)return fmt.Format(format, this, provider);//给我的感觉是灵活，有根据当前类型以及format字符串来处理的感觉。在框架中只需要改变provider，是这个意思？}}  
注意事项:  
实现IFormattable.ToString()可以自定义几首的格式字符串，但有3个字符串应该被实现:"G",null,empty,因为.net内建规范;  
当实现了IFormattable.ToString()后，所有的内建类型都将调用IFormattable.ToString(),而不是System.Object的ToString();较难理解;  
内建方法会自动调用IFormattable.ToString()，使用的参数可能是"G",也可能是空字符串，所以设计时，统一具有Object.ToString()的输出。  
开始时对IFormatProvider进行检查，实现让类型提供者提供格式化的方法。。。很有框架的味道;  
\*类型格式化输出的方法:IFormatProvider的功能，IFormatable接口的实现;IFormatProvider和IFormattable接口共同使用，使得类型设计者实现部分常用格式化需求，并允许使用者提供自己的格式化方法，比较灵活;  
\*管理文件和文件夹的类型:FileInfo,DirectoryInfo;  
File与FileInfo的区别:FileInfo可以对摸个文件生成具体的实例，而File是静态类;FileInfo致力于操作文件在文件系统中的结构，而File致力于文件本身的读写操作;  
注意:由于FileInfo和DirectoryInfo涉及资源的争用，并且不能确保文件和文件夹的存在与否，一般需要放在try,catch,finally中;  
\*如何实现文件和文件夹的监控:使用FileSystemWatch实现对文件系统的监控，可以通过设置监控目录，监控类型和回调方法来方便监控。但需要小心FileSystemWatcher缓存溢出的情况;  
  
5.net中的定时器:  
System.Windows.Forms.Timer;用于窗体设计，并且运行在窗体线程，导致计时不准确和遗漏节拍;当其构造时，会与当先线程关联, 当计时器的计时到达后，一个定时器的消息将被插入到当前线程的消息队列中。当前线程将逐一处理消息队列中的所有消息，并一一派发给各自的处理方法。这样的机制和利用工作者线程有很大的区别，实际上，该类型并未涉及到多线程的操作，定时器的设置，定时方法的执行都在同一个线程上;  
System.Threading.Timer;每个计时回调均在工作者线程上执行,计时相对准确;  
ThreadTimer = new System.Threading.Timer(new System.Threading.TimerCallback(ThreadTimerHandler), null, System.Threading.Timeout.Infinite,System.Threading.Timeout.Infinite);  
ThreadTimer.Change(INTERVAL,INTERVAL);  
System.Timers.Timer;是system.Threading.Timer的一个包装，设计落后，不建议使用;  
\*.net的内建类型的定时器是否会发生回调方法重入;  
方法重入:是一个多线程编程的概念，当程序中多个线程执行时，就可能发生同一个方法被多个线程调用的情况;当这哥方法中存在一些非线程安全代码是，可能会出现数据不一致，产生严重BUG;  
Form.Timer单线程，不会出现;  
Threading.Timer:回调方法在工作者线程执行，每当定时时间发生时，控制该Timer的线程就会负责从线程池分配一个新的工作者线程，会出现方法重入，需要考虑同步;在回调方法中使用lock;暂时理解到这里，puzzle;  
  
6.名企面试真题;  
\*DateTime.Parse(myString);代码有何问题？可能会出现异常，应该用TryParse避免;  
\*Equals,==,ReferenceEquals有何区别?Equals是基本方法，若是系统默认值类型就比较内容，引用类型比较引用,同时可以重写;"=="则是比较折中(当时引用类型时比较引用，值类型时比较被容);ReferenceEquals比较引用;  
\*简述DateTime和长整型的转换算法?高2位为UTC标志位，低62位为时间戳(0001年01月01日00时00分00秒);  
private long InternalTicks{get({return (long)this.dateData&0x3fffffffffffffffL}};  
\*本地时间和UTC时间的转化:通过ToUniversalTime和ToLocal方法可以在本地时间和UTC时间间转换;DateTime dt = new DateTime(DateTime.Now, DateTimeKind.Utc);  
\*GetHashCode方法的作用，什么时候用到?获得哈希码，用于同一类型不同对象的比较，集合操作等情形;  
同时注意3点:统一对象任何时间hashcode应该相同;同一类型不同对象如果值相同，hashcode应该相同;尽可能的散列(随机分布);  
\*.net提供哪几种定时器，各自原理?3中:Form.Timer,与UI同一进程，容易出现误差;Threading.Timer每当计时器触发时到 线程池中创建工作进程，计时准确，但存在方法重入问题，可用同步解决;Timers.Timer，封装Threading.timer,过时;  
\*为什么要设计system.object？类型默认继承该类，可使得代码被托管，CLR可以帮助做很多的事情，方便程序员;  
\*谈谈格式化输出的场所:时间，日期，货币，信件;globalization,localization,translation;  
\*递归  
\*冒泡排序