初级.NET 开发人员 - 任何使用.NET 的人都应知道的

1. 描述线程与进程的区别？

进程是系统所有资源分配时候的一个基本单位，拥有一个完整的虚拟空间地址，并不依赖线程而独立存在。进程可以定义程序的一个实例，但它只是占据应用程序所使用的地址空间。为了让进程完成一定的工作，进程必须至少占有一个线程，这个线程即为主线程，默认程序的工作都在这个主线程中完成，可以通过程序创建多个线程，使多个线程可以同时运行（多 CPU 支持下），这就是多线程技术。

线程（Thread）与进程（Process）二者都定义了某种边界，不同的是进程定义的是应用程序与应用程序之间的边界，不同的进程之间不能共享代码和数据空间，而线程定义的是代码执行堆栈和执行上下文的边界。同一进程中的不同线程共享代码和数据空间。

2. 什么是 Windows 服务，它的生命周期与标准的 EXE 程序有什么不同？

Windows 服务是运行在 windows 后台指定用户下（默认 System）的应用程序，它没有标准的 UI 界面，相比标准的 EXE 程序，Windows 服务是在服务开始的时候创建，而在服务结束的时候销毁，而且可以设置服务是否与操作系统一起启动，一起关闭。它支持三种方式：1)自动方式 2)手动方式 3)禁用。自动方式下，Windows 服务控制器将指导 OS 启动后自动启动服务并运行，而手动方式则必须手工启动服务，禁用的情况下服务将不能被启动。另外标准的 EXE 默认使用的当前登录的用户，而 Windows 服务则默认使用 System 用户，这在对系统资源访问的时候特别需要注意。

3. Windows 上的单个进程所能访问的最大内存量是多少？它与系统的最大虚拟内存一样吗？这对于系统设计有什么影响？

这个需要针对硬件平台，公式为单个进程能访问的最大内存量=2 的处理器位数次方/2，比如通常情况下，32 位处理器下，单个进程所能访问的最大内存量为:$2^{32}/2 = 2G$。单个进程能访问的最大内存量是最大虚拟内存的 1/2，因为要分配给操作系统一半虚拟内存。

4. EXE 和 DLL 之间的区间是什么？

EXE 文件有一个入口点，比如说 Main()函数，这样 CLR 可以由此处执行这个 EXE 文件，而 DLL 不包含这样一个入口点所以不是可执行的。DLL 文件主要包含一些程序集供其它函数调用。

5. 什么是强类型，什么是弱类型？哪种更好些？为什么?

强类型是在编译的时候就确定类型的数据，在执行时类型不能更改，而弱类型在执行的时候才会确定类型。没有好不好，二者各有好处，强类型安全，因为它事先已经确定好了，而且效率高。一般用于编译型编程语言，如 C#、Java 与 C++等，弱类型相比而言不安全，在运行的时候容易出现错误，但它灵活，多用于解释型编程语言，如 Javascript,VB 等。

6. PID 是什么？在做系统的故障排除时如何使用它？

PID 是进程编号，在系统发现故障的时候，可以根据它寻找故障所发生的具体进程。可通过 Visual Studio 将故障进程附加到进程中进行调试。

7. 单个 TCP/IP 端口上能够被多少个进程侦听？

1 个

8. 什么是 GAC？它解决了什么问题?

GAC 是 Global Assembly Cache，全局程序集缓存的简称。它解决了几个程序共享某一个程序集的问题。不必再将那个被共享的程序集拷贝到应用程序目录了，.NET 应用程序在

加载的时候，会首先查看全局应用程序集缓存，如果有就可以直接使用，没有再到应用程序目录进行查找。

GAC 还有一个很重要的作用，所有被放入 GAC 的程序集都是强名程序集，这样可以保证相同文件名不同版本的程序集被正确的引用，这样解决了 COM 时代的 DLL 地狱问题。同时强名还可以防止程序集伪造。

中级.NET 开发人员

1. 阐述面向接口、面向对象、面向方面编程的区别

面向接口更关注的是概念，它的原则是先定义好行为规范，再根据行为规范创建实现，严格的来说，面向接口应该是面向对象中的一部分，因为面向对象也强调的是本末倒置原则，也就是实现依赖于抽象，而抽象不依赖于具体实现。面向对象是对复杂问题的分解。面向方面的编程是一种新概念，它解决了很多面向对象无法解决的问题，比如面向对象技术只能对业务相关的代码模块化，而无法对和业务无关的代码模块化。而面向方面正是解决这一问题的方案，它的关键思想是"将应用程序中的商业逻辑与对其提供支持的通用服务进行分离"。

2. 什么是 Interface？它与 Abstract Class 有什么区别？

接口(Interface)是用来定义行为规范的，不会有具体实现，而抽象类除定义行为规范外，可以有部分实现，但一个类能实现多个接口，但只能继承一个父类。Interface 可以理解为是一种契约。

3. 什么是反射?

程序集包含模块，而模块又包括类型，类型下有成员（这些信息存于元数据中），反射就是在运行时管理程序集，模块，类型的对象，它能够动态的创建类型的实例，设置现有对象的类型或者获取现有对象的类型，能调用类型的方法和访问类型的字段属性。它是在运行时创建和使用类型实例。反射功能的类主要在 Sytem.Type 命名空间下，使用这些类可以在运行时创建类型实例，然后调用其中方法或访问其成员对象。

4. 使用 ASMX 的 XML Web 服务与使用 SOAP 的.NET Remoting 的区别?

Web 服务使用的 SOAP 协议，而 Remoting 采用的 RPC。Web Service 能用于不同平台，不同语言，Remoting 只适用于.NET 应用程序之间的通信。效率上.NET Remoting 高于 Web Service。一般情况下 Web Service 主要应用于互联网，.NET Remoting 主要应用于局域网。

5. 类型系统是由 XMLSchema 表示的吗？CLS 是 XMLSchema 表示的吗？

XmlSchema 是一个特定的 XML 文档必须满足的一套标准。这些标准能够描述不同的数据类型。如 xs:decimal 是一种 XmlSchema 数据类型，其他类型有像 xs:Boolean，xs:Date，xs:int 等等。CLS 是公共语言规范(Common Language Specification)，它是任何.NET 下语言(C#,VB.NET 等)使用的一套数据类型。如 System.Double 是一种 CLS 数据类型。其他的包括 System.Int32，System.Boolean 等。

CLS 与 XMLSchema 表述的相同的意思，但其内在不同。

6. 从概念上阐述前期绑定（early-binding）和后期绑定（late-binding）的区别?

这个就像是强弱类型的比较相似，前期绑定是在编译的时候就确定了要绑定的数据，而后期绑定是在运行的时候才填充数据。所以前期绑定如果失败，会在编译时报编译错误，而后期绑定失败只有在运行时的时候才发生。

7. 调用 Assembly.Load 算静态引用还是动态引用？

这种反射操作基本都是动态，即运行时引用。

8. 何时使用 Assembly.LoadFrom？何时使用 Assembly.LoadFile？

相比 LoadFile，用 LoadFrom 加载程序集时，要求同时将此程序集所依赖的程序集加载进来。而 LoadFile 加载程序集文件时，只将传入参数的文件加载，不考虑程序集依赖，但如果有相同实现，但位置不同的文件用 LoadFrom 是不能同时加载进来的，而 LoadFile 却可以。由于 LoadFile 加载的是文件，所以调用它之后，可能因为缺少必要的依赖造成无法被执行。

9. 什么叫 Assembly Qualified Name？它是一个文件名吗？它有什么不同？

它不是一个文件名，相比文件名，Assembly Qualified Name（程序集限定名称），更能确定一个程序集，它包含文件名，但同时包含版本，公钥，和区域。因为同样一个名称的文件可能有不同的版本和区域，此时单独靠文件名称，可能会造成不能确定程序集的正确性。

10. Assembly.Load("foo.dll")；这句话是否正确？

错误，正确的应该是 Assembly.Load("foo")或者 Assembly.LoadFrom("foo.dll")

11. 做强签名的 assembly 与不做强签名的 assembly 有什么不同？

强签名程序集可以安装到 GAC 中，而不做强签名的确不能。强名程序集可以根据强名来区分相同文件名不同版本的程序集。

12. DateTime 是否可以为 null?

不能，因为 DateTime 为 Struct 类型，而结构属于值类型，值类型不能为 null，只有引用类型才能被赋值 null

13. 什么叫 JIT？什么是 NGEN？它们分别有什么限制和好处？

Just In Time 及时编译，它是在程序第一次运行的时候才进行编译，而 NGEN 是所谓的 pre-jit，就是说在运行前事先就将程序集生成为本机镜像，并保存到全局缓存中，运用 NGEN 可以提高程序集的加载和执行速度，因为它可以从本机映像中还原代码和数据结构，而不必像 JIT 那样动态生成它们。但是 NGEN 不能得到 JIT 编译时优化（如根据处理器）的好处，这是其有缺陷的一方面。

14. .NET CLR 中一代的垃圾收集器是如何管理对象的生命周期的？什么叫非确定性终结？

垃圾收集执行的时机不定的，这个过程对于程序员是透明的，垃圾收集器自动控制着它。垃圾收集器的执行原理：周期性地遍历被应用当前引用的所有对象的列表。在这个搜索过程中，凡是没有发现的被引用的对象，都将准备予以销毁（标记为 0 代）。说明了对象在什么时候终结是不确定的，我认为这就是非确定性终结。通常来说，堆的耗尽是收集清扫的触发条件。

15. Finalize()和 Dispose()之间的区别?

Finalize()相当于析构函数，我们不能手动调用这个函数（但可以通过 GC.SupperFinalize 方法阻止 CLR 调用 Finalize），CLR 自动判断什么时候调用这个函数。Dispose()同样用于释放托管资源，但是我们可以在任意时刻自行调用 Dispose()方法。总结就是 Finalize 自动释放资源，Dispose()用于手动释放资源。

16. using() 语法有用吗？什么是 IDisposable？它是如何实现确定性终结的。

有用。using 编译后的 MSIL 会调用该对象的 Dispose 方法，释放资源。实现了 IDisposable 的类相当与一个契约，即说明这个类要实现 Dispose()方法告诉 CLR 怎么释放它自身。因

为 Dispose()方法释放的非托管资源，所以可以手工调用它来确定性终结一个非托管资源。（托管资源都是非确定性终结，由 CLR 管理，没法改变）

17. tasklist /m "mscor*" 这句命令是干嘛的？

列出所有使用了以" mscor"作为开头的 dll 或者 exe 的映像名称，PID 和模块信息。

18. in-proc 和 out-of-proc 的区别

in-proc 是进程内，进程内能共享代码和数据块，out-of-proc 是进程外，进程外的互操作需要用进程间通讯来实现。

19. .NET 里的哪一项技术能够实现 out-of-proc 通讯？

.Net Remoting 技术或者 WCF 技术

20. 当你在 ASP.NET 中运行一个组件时，它在 Windows XP, Windows 2000, Windows 2003 上 分别跑在哪个进程里面？

XP: aspnet_Wp.exe、Windows 2000: aspnet_Wp.exe、Windows 2003: w3wp.exe。XP 与 2000 是 IIS5.1，2003 是 IIS6.0 所以有了上述差别。

高级.NET 开发人员

1. DateTime.Parse(myString); 这段代码有什么问题？

A: 区域信息即 CultureInfo 没有指定。如果不指定的话，它将采用默认的机器级的设置（见：控制面板->区域和语言选项）并使用这个设置来决定这个字符串即 myString 怎样被解释。所以如果你传入“5/2/2005”且你的区域设置为 En-US，则它会被解释为 May 2nd 2005，但是如果你的区域设置为 Hindi-India，则它会被解释为 5th Feb 2005!

参考下面的代码示例：

```
string sDate = "5/2/2005"; //本意是 2005 年 5 月 2 号
DateTime[] dt = new DateTime[3];
CultureInfo[] cf = new CultureInfo[3];
cf[0] = new CultureInfo("en-US", true); //指定为 en-US,字符串将被解释为“MM/DD/YYYY”
dt[0] = DateTime.Parse(sDate, cf[0], DateTimeStyles.AllowWhiteSpaces);
dt[0] = dt[0].AddMonths(1); //另日期变为 2005 年 6 月 2 日
Console.WriteLine(dt[0].ToString(cf[0]));
cf[1] = new CultureInfo("hi-IN", true); //这是印度格式，字符串被解释为“DD/MM/YYYY”
dt[1] = DateTime.Parse(sDate, cf[1], DateTimeStyles.AllowWhiteSpaces);
dt[1] = dt[1].AddMonths(1); //让它变成 2005 年 3 月 5 日
Console.WriteLine(dt[1].ToString(cf[1]));
```

好了，这解决所有问题了吗？没有！

如果时间放在一个文本框中 – 有些人输入了“2005/02/05” – 我不知道我应该怎样解释这个输入呢！ “DateTime.ParseExact” 要求你必须告诉计算机怎样处理输入的日期字符串它才可以处理。

```
//抛出 FormatException
```

cf[2] = new CultureInfo("hi-IN", true);

dt[2] = DateTime.ParseExact(sDate, new string[] { "d" }, cf[2], DateTimeStyles.AllowWhiteSpaces); //抛出 Format Exception

dt[2] = dt[2].AddMonths(1);

Console.WriteLine(dt[2].ToString(cf[2]));

第二个参数指定了输入字符串的格式 – "d"意思是短日期 – MM/dd/yyyy – 查看.net 文档中的 DateTimeFormatInfo 类中全部的格式。

上面的代码会抛出一个 FormatException，因为你没有遵守规则。

好了，现在我们几乎完成了，如果想让一个字符串表达式被接受怎么办呢？

使用一个自定义表达式。如下：

sDate = "2005/05/02";
dt[2] = DateTime.ParseExact(sDate, **new string[] { "%yyyy/MM/dd" }**, CultureInfo.InvariantCulture, DateTimeStyles.AllowWhiteSpaces);

dt[2] = dt[2].AddMonths(1);

Console.WriteLine(dt[2].ToString(cf[2]));

这样就可以接受了

2. PDB 是什么东西？在调试时它应该放在哪里？

A: 以下是摘自 MSDN 一段说明 A program database file (extension .pdb) is a binary file that contains type and symbolic debugging information gathered over the course of compiling and linking the project. A PDB file is created when you compile a C/C++ program with /ZI or /Zi or a Visual Basic, Visual C#, or JScript program with the /debug option. Object files contain references into the .pdb file for debugging information. For more information on pdb files, see PDB Files (C++). A DIA application can use the following general steps to obtain details about the various symbols, objects, and data elements within an executable image.

翻译后（部分）：程序数据库文件（扩展名：pdb）是一个二进制文件包含了在编译和链接项目过程中收集的类型与符号调试信息。PDB 文件在你使用/ZI 或/Zi 参数编译一个 C/C++ 程序或使用/debug 参数编译一个 Visual Basic, Visual C#或 Jscript 程序时生成。目标文件含有到.pdb 文件的引用以查找调试信息。

下面是查询此文件内容的方法：

u Acquire a data source by creating an IDiaDataSource interface.

u Call IDiaDataSource::loadDataFromPdb or IDiaDataSource::loadDataForExe to load the debugging information.

u Call IDiaDataSource::openSession to open an IDiaSession to gain access to the debugging information.

u Use the methods in IDiaSession to query for the symbols in the data source.

u Use the IDiaEnum* interfaces to enumerate and scan through the symbols or other elements of debug information.

调试时，PDB 文件应该与待调试文件放在同一目录下。

3. 什么是圈复杂度（Cyclomatic Complexity）?为什么它很重要？

A：圈复杂度一种代码复杂度的衡量标准，中文名称叫做圈复杂度。在软件测试的概念里，圈复杂度"用来衡量一个模块判定结构的复杂程度，数量上表现为独立现行路径条数，即合理的预防错误所需测试的最少路径条数，圈复杂度大说明程序代码可能质量低且难于测试和维护，根据经验，程序的可能错误和高的圈复杂度有着很大关系"。

4. 写一个标准的 lock()，在访问变量的前后创建临界区，要有"双重检查"。

A: 这即是 Jeffrey Richter 在其经典著作 CLR Via C#中提到的双检锁(double-check locking)技巧。如下是原书代码：

```
public sealed class Singleton
{
private static Object s_lock = new Object();
private static Singleton s_value;
//私有构造器组织这个类之外的任何代码创建实例
private Singleton() { }
// 下述共有，静态属性返回单实例对象
public static Singleton Value
{
get
{
// 检查是否已被创建
if (s_value == null)
{
// 如果没有，则创建
lock (s_lock)
{
// 检查有没有另一个进程创建了它
if (s_value == null)
{
// 现在可以创建对象了
s_value = new Singleton();
}
}
}
return s_value;
}
```

}

}

5. 什么叫 FullTrust？放入 CAC 的 assembly 是否是 FullTrust 的？

FullTrust 权限基本上允许跳过所有的 CAS 验证，持有该权限的程序集能够访问所有的标准权限。GAC 中的程序集是具有 FullTrust 权限的。详细信息参见此文章。

6. 代码加上需要安全权限的特性有什么好处？

A: 有可能会通过具有元权限程序集或 permview.exe 工具访问这些 attribute 及其参数。有可能在程序集级别使用其中的一些 attribute。可以通过反射访问设置信息。最关键一点使程序集开始执行时而不是在执行期间声明需要权限，这样可以确保用户不会在程序开始工作后遇到障碍。

7. gacutil /l | find /i "Corillian"这句命令的作用是什么？

A: gacutil 全称.NET Global Assembly Cache Utility。参数/l [<assembly_name>] 的作用：列出通过 <assembly_name> 筛选出的全局程序集缓存。find 命令作用：在文件中搜索字符串，其参数/i 作用是搜索字符串时忽略大小写。所以这条命令的意思就是在所有的全局程序缓存中搜索名称包含字符串"Corillian"的程序集。

8. sn -t foo.dll 这句命令的作用？

A: sn 是.NET Framework 强名称实用工具，其中参数-T[p] <assembly>作用是显示 <assembly> 的公钥的标记(如果使用了-Tp，则还同时显示公钥本身)。此命令显示了 foo.dll 的公钥。如果 foo.dll 非强名程序集产生如下提示：未能将密钥转换为标记 -- 程序集"(null)"的公钥无效。

9. DCOM 需要防火墙打开哪些端口？端口 135 是干嘛用的？

A: DCOM 动态的选择 1024～65535 之间的网络端口。此外，DCOM 要用 135 端口实现一些重要的功能。DCOM 是使用 RPC 进行通讯的。利用 RPC 功能进行通信时，就会向对方电脑的 135 端口询问可以使用那个端口进行通讯。这样，对方的电脑就会告知可以使用的端口号，实际的通讯将使用这个端口来进行。

135 端口起的是动态的决定实际的 RPC 通讯使用的端口映射功能，主要用于使用 RPC（Remote Procedure Call，远程过程调用）协议并提供 DCOM（分布式组件对象模型）服务。

10. 对比 OOP 和 SOA，它们的目的分别是什么？

面向对象主要用于对具体的事物进行抽象建模，而 SOA 的主要目的是定义部署和管理服务的方式，采用 SOA 的架构可以提高可重用性，降低总成本，提高快速修改与演化 IT 系统的能力。

11. XmlSerializer 是如何工作的？使用这个类的进程需要什么 ACL 权限？

XmlSerializer 提供了方法，使你能够将自己的对象序列化和反序列化为 XML，同时在这个过程中提供一定的控制。这个过程由一些 attribute 来控制。以下是代码示例：

序列化：

```
Theater theater = … ;
XmlSerializer xs = new XmlSerializer( typeof ( Theater ) );
FileStream fs = new FileStream( args[0], FileMode.Create );
```

xs.Serialize(fs, theater);

反序列化：

XmlSerializer xs = new XmlSerializer(typeof (Theater));

FileStream fs = new FileStream(args[0], FileMode.Open);

Theater theater = (Theater)xs.Deserialize(fs);

控制这个过程：

[XmlRoot("theater")] -- 序列化 xml 的根节点

[XmlElement("name")] -- 序列化为 xml 的元素及元素名称

[XmlAttribute("minutes")] -- 序列化为 xml 中的特性及其名称

[XmlElement("showing", DataType="time")] -- 序列化为 xml 的元素及其类型

ACL 权限上需要执行程序集的用户拥有对 C:\WINDOWS\Temp\的访问权限

12. 为什么不提倡 catch(Exception)？

Exception 类包含许多子类，程序执行的时候要将每一个类都搜索一遍以找到符合的异常类，这样是蛮消耗资源的，影响程序运行效率。另外，只捕捉特定环境，特定条件下的异常，并进行处理。不应该捕捉所有异常，因为有些异常是我们所无法预料到的，比如，内存溢出或其他错误，这种情况下，不应该让系统以一种未知状态继续运行。

13. Debug.Write 和 Trace.Write 有什么不同？何时应该使用哪一个？

Debug 类提供一组帮助调试代码的方法和属性。Trace 类提供一组帮助跟踪代码执行的方法和属性，通俗的说就是为在不打断程序的调试或跟踪下，用来记录程序执行的过程。

Debug 只在 debug 状态下会输出，Trace 在 Release 下也会输出，在 Release 下 Debug 的内容会消失。

14. Debug Build 和 Release Build 的区别，是否会有明显的速度变化？请说明理由。

首先以一个表格说明问题：

项目	Debug	Release
条件编译常数	Debug;Trace	Trace
优化代码	False	True
输出路径	bin\Debug	bin\Release
生成调试信息	True	False

Debug 模式下生成的程序集为调试版本，未经优化；在 bin\debug\目录中有两个文件，除了要生成的.exe 或.dll 文件外，还有个.pdb 文件，这个.pdb 文件中就记录了代码中的断点等调试信息；Release 模式下不包含调试信息，并对代码进行了优化，\bin\release\目录下只有一个.exe 或.dll 文件。在项目文件夹下除了 bin 外，还有个 obj 目录。编译是分模块编译的，每个模块的编译结果就保存在了 obj 目录下。最后会合并为一个 exe 或者 dll 文件保存到 bin 之中。因为每次编译都是增量编译，也就是只重新编译改变了的模块，所以这个 obj 的目录的作用就是保存这些小块的编译结果，加快编译速度。

15. JIT 是以 assembly 为单位发生还是以方法为单位发生？这对于工作区有何影响?

JIT 编译是以方法为单位完成的，当用到相应的方法才把其编译执行，并将编译结果暂存当下一次用到时就不用再编译。可以一定程度上提高效率。

16. 对比抽象基类和接口的使用

如果要让自定义类包含基类的实现，应该选用抽象基类，接口多做契约使用。.NET 中类只允许单继承，但接口允许多继承。

17. a.Equals(b)和 a==b 相同吗？

对于值类型的比较两者效果相同(值类型 ValueType 重写了 Equals 方法)。对于除 string 以外的引用类型==比较对象是否为同一引用，对于 string，==比较字符串的内容。而 Equals 方法对所有引用对象都是比较其是否为同一对象的引用。

很多自定义类都会重写 Equals 所以不要只注意默认实现。

18. 在对象比较中，对象一致和对象相等分别是指什么？

对象一致指的是两个对象是否指向托管堆中同一个目标，其比较对象的类型与值两个方面。对象相等指的是两个对象中的数据成员是否相等。前者是一种浅比较，后者是一种深比较。

19. 在.NET 中如何实现深拷贝(deep copy)？

实现深拷贝的程序结构基本如下：

示例代码：

```
//实现 ICloneable 接口

public class Copy:ICloneable
{
//实现 ICloneable 接口的 Clone()方法
public override object Clone()

{

//在此实现深拷贝

    }
}
```

实现深拷贝的代码中，新建一个对象，然后将对象的成员逐个赋值给新对象的成员。最后将新对象返回。

20. 请解释一下 ICloneable

当用户需要实现除 MemberwiseClone 之外的拷贝方法时（如深拷贝），用户可以实现 ICloneable 接口，并重写 Clone()方法。在 Clone()方法中编写自己的实现。

21. 什么叫装箱。

将值类型对象放到托管堆中需要一个装箱操作。需要装箱的情况有很多，比如将值类型数据存入一个数组(Array 类)时，值类型会被装箱然后存入托管堆。

22. string 是值类型还是引用类型？

string 这个引用类型是非常特殊一个引用类型。

它有两点特殊的地方。第一点对象分配的特殊。例如，有如下语句：

```
string str1 = "abcd";
```

string str2 = "abcd";

那么.net 在分配 string 类型的时候，先查看当前 string 类型列表是否有相同的，如果有的话，直接返回其的引用，否则重新分配。

第二点对象引用操作的特殊，即不同于真正意义上的引用操作。例如如下语句：

string str1 = "abcd";

string str2 = str1;

str2 = "efgh";// str1 is still "abcd" here

当对于一个新的 string 类型是原有对象引用的时候，这点和一般的引用类型一样，但是当新的对象发生变化的时候，要重新分配一个新的地方，然后修改对象指向。

因此对于 string 操作的时候，尤其发生变化的时候，会显得比较慢，因为其牵扯到内存地址的变化。（所以对于数据量比较大的字符操作时候，使用 StringBuilder 来说效率会提升很高。）

23. XmlSerializer 使用的针对属性的模式有什么好处？解决了什么问题？

这里的属性指 attribute。可以将不需要序列化的数据标记为[XmlIgnore]，只序列化有用的数据，而不是序列化整个对象。可以省去没有必要序列化的数据序列化工作，提升序列化时的性能。使用 attribute 还可以指导序列化的过程，以自定义生成 xml 文档的格式。（11 题对 XmlSerializer 有介绍）

24. 为什么不应该在.NET 中使用 out 参数？它究竟好不好？

当想要一个函数产生多个输出时，可以使用 out 参数返回函数的输出值。这应该是 out 参数最大的作用。out 参数的缺陷在于，它允许一个未初始化变量就在函数中作为 out 参数使用。这样并不能保证在访问一个作为 out 参数的变量时，它已经被初始化过，容易产生错误的结果。（个人观点）

25. 特性能够放到某个方法的参数上？如果可以，这有什么用？

可以，作用可以对参数有进一步限定，比如输入参数为 int 类型，可以通过允许 AttributeTargets=ParameterInfo 的 Attribute 自定义实现来限定输入参数的大小，比如当输入参数小于 100 的时候便抱错。

对方法的参数设置 Attribute 的例子：

```
  [AttributeUsage(AttributeTargets.Parameter)]
    public class ParameterAtt : Attribute
    {
       public int Min = 100;
    }
public class AttributeTest
    {
       public void TestMethod([ParameterAtt(Min = 100)] int par1)
       {
          ParameterInfo para =
MethodInfo.GetCurrentMethod().GetParameters()[0];
          ParameterAtt att = ParameterAtt.GetCustomAttribute
```

```
(para, typeof(ParameterAtt)) as ParameterAtt;
          if (att.Min > par1)
          {
             throw new Exception("要求 para1 最小为" + att.Min);
          }
       }
    }
```