asp.net c# 网上搜集面试题目大全(附答案) - 恆松

C#习题大全

1.String str=new String("a")和String str = "a"有什么区别?

String str = "a"; 这个只是一个引用，内存中如果有“a"的话，str就指向它，如果没有才创建如后还用到"a"这个字符串的话并且是这样用： String str1 = "a"; String str2 = "a"; String str2 = "a"; 这4个变量都共享一个字符串"a" 而String str = new String("a");是根据"a"这个String对象再次构造一个String对象，将新构造出来的String对象的引用赋给str

2.判断字符串变量str是否为空的一下三种方法哪个性能更优

a、str=="";b、str==String.Empty;c、str.Length==0;? 答案是c;

3.string与String的区别

string、int是C#定义的类型，而String、Int32是.net类型即是CTS类型； string 是 .NET 框架中 System.String 的别名。string在编译的时候会转化为String类

4.虚方法（virtual ）和抽象方法（abstract）的区别?

1：抽象方法仅有声明，而没有任何实现，如abstract someMethod();，虚方法却不能如此

virtual用于修饰方法、属性、索引器或事件声明，并使它们可以在派生类中被重写。
2：子类继承父类，可以对父类中的虚方法进行重写、覆盖、不处理三种处理（见5），对抽象方法却必须实现

5.子类对父类中虚方法的处理有重写（override）和覆盖（new），请说明它们的区别?

有父类ParentClass和子类ChildClass、以及父类的虚方法VirtualMethod。有如下程序段：
ParentClass pc = new ChildClass();pc.VirtualMethod(...);
如果子类是重写（override）父类的VirtualMethod，则上面的第二行语句将调用子类的该方法
如果子类是覆盖（new）父类的VirtualMethod，则上面的第二行语句将调用父类的该方法

6.抽象类（abstract）和接口（interface）的区别

抽象类可以有自己的实现，接口却仅有声明，可以有自己的静态数据字段;
java和C#中可以实现多个接口，却只能继承一个抽象类（或者非抽象类）（单继承，和c++的多继承不同）;

7.填空:

(1)面向对象的语言具有 继承性、多态性、封装性。
(2)能用foreach遍历访问的对象需要实现 _IEnumerable___接口或声明_GetEnumerator_方法的类型。
(3)列举ADO.net中的五个主要对象

Connection,Command,DataReader,DataAdapter,DataSet
connection 连接对象
command 命令对象，指示要执行的命令和存储过程!

datareader是一个向前的只读的数据流。
dataadapter是功能强大的适陪器，支持增删改查的功能
dataset是一个数据级对象，相当与内存中的一张表或多张表

8.不定项选择：

(1) 以下叙述正确的是：BC

A. 接口中可以有虚方法。 B. 一个类可以实现多个接口。

C. 接口不能被实例化。 D. 接口中可以包含已实现的方法。
(2) 从数据库读取记录，你可能用到的方法有： BCD

A. ExecuteNonQuery B. ExecuteScalar C. Fill D. ExecuteReader
9.简述 private、 protected、 public、 internal 修饰符的访问权限。

A.Private:关键字是一个成员访问修饰符。私有访问是允许的最低访问级别。私有成员只有在声明它们的类和结构体中才是可访问的, 同一体中的嵌套类型也可以访问那些私有成员

B.Protected 关键字是一个成员访问修饰符。受保护成员在它的类中可访问并且可由派生类访问。有关 protected 与其他访问修饰符的比较，请参见可访问性级别,只有在通过派生类类型发生访问时，基类的受保护成员在派生类中才是可访问的

C.Internal关键字是类型和类型成员的访问修饰符。只有在同一程序集的文件中，内部类型或成员才是可访问的

D Protected Internal 它可以看作是Protected与Internal的并集，意思是：如果是继承关系，无论在不在同一程序集里都能访问；如果不是继承关系，那么只能在同一程序集内访问。

E.Public具有最高级别的访问权限，对访问成员没有限制。

10.写出一条Sql语句：取出表A中第31到第40记录（SQLServer, 以自动增长的ID作为主键, 注意：ID

可能不是连续的。）
11.列举ASP.NET 页面之间传递值的几种方式。

QueryString,Session和Server.Transfer
12.写出程序的输出结果

```
class Class1 {

private string str = "Class1.str";private int i = 0;  static void StringConvert(string str) {str = "string being converted.";  }static void StringConvert(Class1 c) {c.str = "string being converted.";  }static void Add(int i) {i++;}static void AddWithRef(ref int i) {i++;}static void Main() {int i1 = 10;  int i2 = 20;  string str = "str";Class1 c = new Class1();Add(i1);  AddWithRef(ref i2);  Add(c.i);  StringConvert(str);  StringConvert(c);  Console.WriteLine(i1);  Console.WriteLine(i2);  Console.WriteLine(c.i);  Console.WriteLine(str);  Console.WriteLine(c.str);  } }
```

13.写出程序的输出结果

```
public abstract class A {public A(){Console.WriteLine('A');  }public virtual void Fun() {Console.WriteLine("A.Fun()");}}public class B: A {public B(){Console.WriteLine('B');  }public new
```

```
void Fun() {Console.WriteLine("B.Fun()");}public static void Main(){A a = new B();a.Fun();}
}
```

14.写出程序的输出结果：

```
  public class A {public virtual void Fun1(int i) {Console.WriteLine(i);  }public void Fun2(A a)
{a.Fun1(1);  Fun1(5);  }}public class B : A {public override void Fun1(int i) {base.Fun1 (i +
1);  }public static void Main(){B b = new B();A a = new A();a.Fun2(b);  b.Fun2(a); }}
```

15.一列数的规则如下: 1、1、2、3、5、8、13、21、34.....求第30位数是多少， 用递归算法实现。

16.程序设计: 猫大叫一声，所有的老鼠都开始逃跑，主人被惊醒。(C#语言)

要求： 1.要有联动性，老鼠和主人的行为是被动的。2.考虑可扩展性，猫的叫声可能引起其他联动效应。

参考答案

1.(1) 继承性、封装性、多态性。 (2) IEnumerable 、 GetEnumerator (3) 对ADO.net的了解

2. (1) B、C (考对接口的理解) (2) B、C、D (考查对ADO.net的熟练程度)

3. private : 私有成员, 在类的内部才可以访问。protected : 保护成员，该类内部和继承类中可以访问。public : 公共成员，完全公开，没有访问限制。internal: 在同一命名空间内可以访问。

4. 解1: select top 10 * from A where id not in (select top 30 id from A)

解2: select top 10 * from A where id > (select max(id) from (select top 30 id from A)as A)

5. 1.使用QueryString, 如....id=1; response. Redirect()2.使用Session变量3.使用Server.Transfer

6. (考查值引用和对象引用)

10210Strstring being converted.7. A BA.Fun()
(考查在继承类中构造函数, 以及new 方法,)

8. 2
5
1
6

评分标准： 答对一点得2分，两点得5分，3点得7分。全对得10分。

9.

```
public class MainClass
{
public static void Main()
{
Console.WriteLine(Foo(30));
}
public static int Foo(int i)
{
if (i <= 0)
return 0;
else if(i > 0 && i <= 2)
return 1;
else return Foo(i -1) + Foo(i - 2);
```

```
}
}
```
评分标准： 写出return Foo(i -1) + Foo(i - 2); 得5分。
写出if(i > 0 && i <= 2) return 1; 得5分。
方法参数过多需要扣分（扣除分数 = 参数个数 － 1）
不用递归算法扣5分
(递归算法在树结构建立等方面比较常用)
10.要点：1. 联动效果，运行代码只要执行Cat.Cryed()方法。2. 对老鼠和主人进行抽象
评分标准: <1>.构造出Cat、Mouse、Master三个类，并能使程序运行(2分)
<2>从Mouse和Master中提取抽象（5分）
<3>联动效应，只要执行Cat.Cryed()就可以使老鼠逃跑，主人惊醒。(3分)
```
public interface Observer
{
void Response(); //观察者的响应，如是老鼠见到猫的反映
}
public interface Subject
{
void AimAt(Observer obs); //针对哪些观察者，这里指猫的要扑捉的对象---老鼠
}
public class Mouse : Observer
{
private string name;
public Mouse(string name, Subject subj)
{
this.name = name;
subj.AimAt(this);
}
public void Response()
{
Console.WriteLine(name + " attempt to escape!");
}
}
public class Master : Observer
{
public Master(Subject subj)
{
subj.AimAt(this);
}

public void Response()
{
Console.WriteLine("Host waken!");
}
}
public class Cat : Subject
```

```
{
private ArrayList observers;
public Cat()
{
this.observers = new ArrayList();
}
public void AimAt(Observer obs)
{
this.observers.Add(obs);
}
public void Cry()
{
Console.WriteLine("Cat cryed!");
foreach (Observer obs in this.observers)
{
obs.Response();
}
}
}
class MainClass
{
static void Main(string[] args)
{
Cat cat = new Cat();
Mouse mouse1 = new Mouse("mouse1", cat);
Mouse mouse2 = new Mouse("mouse2", cat);
Master master = new Master(cat);
cat.Cry();
}
}
//-----------------------------------------------------------------------------------------设
计方法二: 使用event -- delegate设计..
public delegate void SubEventHandler();
public abstract class Subject
{
public event SubEventHandler SubEvent;
protected void FireAway()
{
if (this.SubEvent != null)
this.SubEvent();
}
}
public class Cat : Subject
{
public void Cry()
```

```
{
Console.WriteLine("cat cryed.");
this.FireAway();
}
}
public abstract class Observer
{
public Observer(Subject sub)
{
sub.SubEvent += new SubEventHandler(Response);
}
public abstract void Response();
}
public class Mouse : Observer
{
private string name;
public Mouse(string name, Subject sub) : base(sub)
{
this.name = name;
}
public override void Response()
{
Console.WriteLine(name + " attempt to escape!");
}
}
public class Master : Observer
{
public Master(Subject sub) : base(sub){}
public override void Response()
{
Console.WriteLine("host waken");
}
}
class Class1
{
static void Main(string[] args)
{
Cat cat = new Cat();
Mouse mouse1 = new Mouse("mouse1", cat);
Mouse mouse2 = new Mouse("mouse2", cat);
Master master = new Master(cat);
cat.Cry();
}
}
```

18.有哪几种方法可以实现一个类存取另外一个类的成员函数及属性,并请举列来加以说明和分析.
19. A类是B类的基类,并且都有自己的构造,析构函数,请举例证明B类从实例化到消亡过程中构造,析构函

数的执行过程.请附code

构造先父后子，析够反之

```
public class TestB

      {              public TestB()              {                        Console.Write("begin
create B object"r"n");              }              ~TestB()
{                      Console.Write("begin destory B object"r"n");              }       }
public class TestA : TestB       {              public TestA()
{                      Console.Write("begin create A object"r"n");              }
~TestA()              {                      Console.Write("begin destory A
object"r"n");              }       }
```

21..Net中读写数据库需要用到哪些类？他们的作用

sqlconnection 连接数据库,sqlcommand 执行T-SQL语句，或存储过程
22.ASP.net的身份验证方式有哪些？分别是什么原理？

asp.net提供了3种认证方式: windows身份验证, Forms验证和Passport验证.windows, 身份验证: IIS根据应用程序的设置执行身份验证.要使用这种验证方式,在IIS中必须禁用匿名访问.Forms验证:用Cookie来保存用户凭证,并将未经身份验证的用户重定向到自定义的登录页.Passport验证:通过Microsoft的集中身份验证服务执行的,他为成员站点提供单独登录和核心配置文件服务.

23.解释一下UDDI、WSDL的意义及其作用。

UDDI（Universal Description, Discovery and Integration）统一描述、发现和集成协议，是为解决Web服务的发布和发现问题而制订的新一代基于Internet的电子商务技术标准。它包含一组基于Web的、分布式的Web服务信息注册中心的实现标准，以及一组使企业能将自己提供的Web服务注册到该中心的实现标准。

UDDI利用SOAP消息来查找和注册Web服务。并为应用程序提供了一系列接口来访问注册中心。

24.常用的调用webservice方法有哪些？三种

利用webservice.htc调用WebService方法

在Internet上调用WebService方法
25.讲一讲你理解的web service,在dot net framework中，怎么很好的结合xml（讲概念就行了）

Web Service就是一个应用程序，它向外界暴露出一个能够通过Web进行调用的API。这就是说，你能够用编程的方法通过Web调用来实现某个功能的应用程序。Web Service是一种新的Web应用程序分支，它们是自包含、自描述、模块化的应用，可以在网络(通常为Web)中被描述、发布、查找以及通过Web来调用。Web Service便是基于网络的、分布式的模块化组件，它执行特定的任务，遵守具体的技术规范，这些规范使得Web Service能与其他兼容的组件进行互操作。它可以使用标准的互联网协议，像超文本传输协议HTTP和XML，将功能体现在互联网和企业内部网上。Web Service平台是一套标准，它定义了应用程序如何在Web上实现互操作性。你可以用你喜欢的任何语言，在你喜欢的任何平台上写Web Service。可扩展的标记语言XML 是Web Service平台中表示数据的基本

格式。除了易于建立和易于分析外，XML主要的优点在于它既与平台无关，又与厂商无关。XML是由万维网协会(W3C)创建，W3C制定的XML SchemaXSD 定义了一套标准的数据类型，并给出了一种语言来扩展这套数据类型。Web Service平台是用XSD来作为数据类型系统的。当你用某种语言如VB.NET或C# 来构造一个Web Service时，为了符合Web Service标准，所有你使用的数据类型都必须被转换为XSD类型。如想让它使用在不同平台和不同软件的不同组织间传递，还需要用某种东西将它包装起来。这种东西就是一种协议，如 SOAP。

26.C#可否对内存进行直接的操作？(这可是个难点哦？要注意！),如果能,如何操作;如果不能,为什么

可以,用指针来操作
27.描述一下C#中索引器的实现过程，是否只能根据数字进行索引？

C#通过提供索引器，可以象处理数组一样处理对象。特别是属性，每一个元素都以一个get或set方法暴露。索引器不单能索引数字（数组下标），还能索引一些HASHMAP的字符串，所以，通常来说，C#中类的索引器通常只有一个，就是THIS，但也可以有无数个，只要你的参数列表不同就可以了索引器和返回值无关，索引器最大的好处是使代码看上去更自然，更符合实际的思考模式.

28.面向对象的思想主要包括什么？

封装：用抽象的数据类型将数据和基于数据的操作封装在一起，数据被保护在抽象数据类型内部。
继承：子类拥有父类的所有数据和操作。

多态：一个程序中同名的不同方法共存的情况。有两种形式的多态 - 重载与重写。
29.什么是ASP.net中的用户控件

有时可能需要控件中具有内置 ASP.NET Web 服务器控件未提供的功能。在这种情况下，您可以创建自己的控件。有两个选择。您可以创建：用户控件。用户控件是能够在其中放置标记和 Web 服务器控件的容器。然后，可以将用户控件作为一个单元对待，为其定义属性和方法。

自定义控件。自定义控件是编写的一个类，此类从 Control 或 WebControl 派生。

ASP.NET Web 用户控件与完整的 ASP.NET 网页（.aspx 文件）相似，同时具有用户界面页和代码。可以采取与创建 ASP.NET 页相似的方式创建用户控件，然后向其中添加所需的标记和子控件。用户控件可以像页面一样包含对其内容进行操作（包括执行数据绑定等任务）的代码。

用户控件与 ASP.NET 网页有以下区别：

用户控件的文件扩展名为 .ascx。

用户控件中没有 @ Page 指令，而是包含 @ Control 指令，该指令对配置及其他属性进行定义。

用户控件不能作为独立文件运行。而必须像处理任何控件一样，将它们添加到 ASP.NET 页中。

用户控件中没有 html、body 或 form 元素。这些元素必须位于宿主页中。

可以在用户控件上使用与在 ASP.NET 网页上所用相同的 HTML 元素（html、body 或 form 元除外）和 Web 控件。例如，如果您要创建一个将用作工具栏的用户控件，则可以将一系列 Button Web服务器控件放在该控件上，并创建这些按钮的事件处理程序。

创建用户控件要比创建自定义控件方便很多，因为可以重用现有的控件。用户控件使创建具有复杂用户界面元素的控件极为方便。

30.什么叫应用程序域？什么是受管制的代码？什么是强类型系统？什么是装箱和拆箱？什么是重载？

CTS、CLS和CLR分别作何解释？

应用程序域应用程序域为安全性、可靠性、版本控制以及卸载程序集提供了隔离边界。应用程序域通常由运行库宿主创建，运行库宿主负责在运行应用程序之前引导公共语言运行库。应用程序域提供了一个更安全、用途更广的处理单元，公共语言运行库可使用该单元提供应用程序之间的隔离。托管代码使用基于公共语言运行库的语言编译器开发的代码称为托管代码；托管代码具有许多优点，例如：跨语言集成、跨语言异常处理、增强的安全性、版本控制和部署支持、简化的组件交互模型、调试和分析服务等。装箱和拆箱装箱和拆箱使值类型能够被视为对象。对值类型装箱将把该值类型打包到 Object 引用类型的一个实例中。这使得值类型可以存储于垃圾回收堆中。拆箱将从对象中提取值类型。重载每个类型成员都有一个唯一的签名。方法签名由方法名称和一个参数列表（方法的参数的顺序和类型）组成。只要签名不同，就可以在一种类型内定义具有相同名称的多种方法。当定义两种或多种具有相同名称的方法时，就称作重载。CTS通用类型系统 (common type system)一种确定公共语言运行库如何定义、使用和管理类型的规范。CLR公共语言运行库.NET Framework 提供了一个称为公共语言运行库的运行时环境，它运行代码并提供使开发过程更轻松的服务。CLS公共语言规范要和其他对象完全交互，而不管这些对象是以何种语言实现的，对象必须只向调用方公开那些它们必须与之互用的所有语言的通用功能。为此定义了公共语言规范 (CLS)，它是许多应用程序所需的一套基本语言功能。强类型C# 是强类型语言；因此每个变量和对象都必须具有声明类型。

31.列举一下你所了解的XML技术及其应用

可扩展标记语言XML(eXtensible Markup Language)[1]是一种简单灵活的文本格式的可扩展标记语言,起源于SGML(Standard Generalized Markup Language),是SGML的一个子集合,也就是SGML的一个简化版本,非常适合于在Web上或者其它多种数据源间进行数据的交换.

32.值类型和引用类型的区别？写出C#的样例代码。

基于值类型的变量直接包含值。将一个值类型变量赋给另一个值类型变量时，将复制包含的值。这与引用类型变量的赋值不同，引用类型变量的赋值只复制对对象的引用，而不复制对象本身。所有的值类型均隐式派生自System.ValueType。与引用类型不同，从值类型不可能派生出新的类型。但与引用类型相同的是，结构也可以实现接口。与引用类型不同，值类型不可能包含 null 值。然而，可空类型功能允许将 null 赋给值类型。每种值类型均有一个隐式的默认构造函数来初始化该类型的默认值。值类型主要由两类组成：结构、枚举,结构分为以下几类：Numeric（数值）类型、整型、浮点型、decimal、bool、用户定义的结构。引用类型的变量又称为对象，可存储对实际数据的引用。声明引用类型的关键字：class、interface、delegate、内置引用类型：object、string

33.ADO.net中常用的对象有哪些？分别描述一下。

Connection 数据库连接对象Command 数据库命令DataReader 数据读取器DataSet 数据集

34.如何理解委托？

委托类似于 C++ 函数指针，但它是类型安全的。委托允许将方法作为参数进行传递。委托可用于定义回调方法。委托可以链接在一起；例如，可以对一个事件调用多个方法。方法不需要与委托签名精确匹配。有关更多信息，请参见协变和逆变。C# 2.0 版引入了匿名方法的概念，此类方法允许将代码块作为参数传递，以代替单独定义的方法。

35.C#中的接口和类有什么异同。

异：不能直接实例化接口。接口不包含方法的实现。接口、类和结构可从多个接口继承。但是C# 只支持单继承：类只能从一个基类继承实现。类定义可在不同的源文件之间进行拆分。

同：接口、类和结构可从多个接口继承。接口类似于抽象基类：继承接口的任何非抽象类型都必须实现接口的所有成员。接口可以包含事件、索引器、方法和属性。一个类可以实现多个接口。
37.UDP连接和TCP连接的异同。

TCP---传输控制协议,提供的是面向连接、可靠的字节流服务。当客户和服务器彼此交换数据前，必须先在双方之间建立一个TCP连接，之后才能传输数据。TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。
UDP---用户数据报协议，是一个简单的面向数据报的运输层协议。UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快。

39.进程和线程分别怎么理解？

进程和线程都是由操作系统所体会的程序运行的基本单元，系统利用该基本单元实现系统对应用的并发性。进程和线程的区别在于：简而言之,一个程序至少有一个进程,一个进程至少有一个线程.线程的划分尺度小于进程，使得多线程程序的并发性高。另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位.线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.一个线程可以创建和撤销另一个线程;同一个进程中的多个线程之间可以并发执行.

40.什么是code-Behind技术。

就是代码隐藏，在ASP.NET中通过ASPX页面指向CS文件的方法实现显示逻辑和处理逻辑的分离，这样有助于web应用程序的创建。比如分工，美工和编程的可以个干各的，不用再像以前asp那样都代码和html代码混在一起，难以维护。

41..net中读写XML的类都归属于哪些命名空间？

System.XML类

42.什么是SOAP?有哪些应用。

SOAP（Simple Object Access Protocol ）简单对象访问协议是在分散或分布式的环境中交换信息并执行远程过程调用的协议，是一个基于XML的协议。使用SOAP，不用考虑任何特定的传输协议（最常用的还是HTTP协议），可以允许任何类型的对象或代码，在任何平台上，以任何一直语言相互通信。

SOAP 是一种轻量级协议，用于在分散型、分布式环境中交换结构化信息。 SOAP 利用 XML 技术定义一种可扩展的消息处理框架，它提供了一种可通过多种底层协议进行交换的消息结构。这种框架的设计思想是要独立于任何一种特定的编程模型和其他特定实现的语义。SOAP 定义了一种方法以便将 XML 消息从 A 点传送到 B 点。 为

此，它提供了一种基于 XML 且具有以下特性的消息处理框架：1）可扩展，2）可通过多种底层网络协议使用，3）独立于编程模型。

43.如何理解.net中的垃圾回收机制。

.NET Framework 的垃圾回收器管理应用程序的内存分配和释放。每次您使用 new 运算符创建对象时，运行库都从托管堆为该对象分配内存。只要托管堆中有地址空间可用，运行库就会继续为新对象分配空间。但是，内存不是无限大的。最终，垃圾回收器必须执行回收以释放一些内存。垃圾回收器优化引擎根据正在进行的分配情况确定执行回收的最佳时间。当垃圾回收器执行回收时，它检查托管堆中不再被应用程序使用的对象并执行必要的操作来回收它们占用的内存。

44.什么是WEB控件？使用WEB控件有那些优势？

运行在服务器端的控件,只要将HTML控件加上runat=server.
45.请谈谈对正则表达式的看法？

正则表达式是一种处理文本的有用工具。无论是验证用户输入、搜索字符串内的模式、还是以各种有效方式重新设置文本格式，正则表达式都非常有用。
46.WEB控件可以激法服务端事件，请谈谈服务端事件是怎么发生并解释其原理？自动传回是什么？为什么

要使用自动传回。
在web控件发生事件时，客户端采用提交的形式将数据交回服务端，服务端先调用Page_Load事件,然后根据传回的状态信息自动调用服务端事件自动传回是当我们在点击客户端控件时，采用提交表单的形式将数据直接传回到务端只有通过自动传回才能实现服务端事件的机制，如果没有自动回传机制就只能调用客户端事件，而不能调用服务端事件

47.WEB控件及HTML服务端控件能否调用客户端方法？如果能，请解释如何调用？

可以调用,例如:<asp:TextBox id="TextBox1" onclick="clientfunction();" runat="server">
</asp:TextBox>
<INPUT id="Button2" value="Button" name="Button2"runat="server" onclick="clientfunction();">

48.请解释web.config文件中的重要节点

Web.config文件是一个XML文本文件，它用来储存 ASP.NET Web 应用程序的配置信息（如最常用的设置ASP.NET Web 应用程序的身份验证方式），它可以出现在应用程序的每一个目录中。当你通过VB.NET新建一个Web应用程序后，默认情况下会在根目录自动创建一个默认的Web.config文件，包括默认的配置设置，所有的子目录都继承它的配置设置。如果你想修改子目录的配置设置，你可以在该子目录下新建一个Web.config文件。它可以提供除从父目录继承的配置信息以外的配置信息，也可以重写或修改父目录中定义的设置。 1、<authentication> 节作用：配置 ASP.NET 身份验证支持（为Windows、Forms、PassPort、None四种）。该元素只能在计算机、站点或应用程序级别声明。<authentication> 元素必需与<authorization> 节配合使用。示例：以下示例为基于窗体（Forms）的身份验证配置站点，当没有登陆的用户访问需要身份验证的网页，网页自动跳转到登陆网
页。<authentication mode="Forms" > <forms loginUrl="logon.aspx" name=".FormsAuthCookie"/>
</authentication> 其中元素loginUrl表示登陆网页的名称，name表示Cookie名称2、<authorization> 节作用：控制对 URL 资源的客户端访问（如允许匿名用户访问）。此元素可以在任何级别（计算机、站点、应用程序、子目录或页）上声明。必需与<authentication> 节配合使用。3、<compilation>节作用：配置 ASP.NET 使用的所有编译设置。默认的debug属性为“True”.在程序编译完成交付使用之后应将其设

为True（Web.config文件中有详细说明，此处省略示例）4、<customErrors>作用：为 ASP.NET 应用程序提供有关自定义错误信息的信息。它不适用于 XML Web services 中发生的错误。5、<httpRuntime>节作用：配置 ASP.NET HTTP 运行库设置。该节可以在计算机、站点、应用程序和子目录级别声明。 6、 <pages>作用：标识特定于页的配置设置（如是否启用会话状态、视图状态，是否检测用户的输入等）。<pages>可以在计算机、站点、应用程序和子目录级别声明。7、<sessionState> 作用：为当前应用程序配置会话状态设置（如设置是否启用会话状态，会话状态保存位置）。 8、<trace> 作用：配置 ASP.NET 跟踪服务，主要用来程序测试判断哪里出错。

49.请解释ASP.NET中的web页面与其隐藏类之间的关系？

一个ASP.NET页面一般都对应一个隐藏类,一般都在ASP.NET页面的声明中指定了隐藏类例如一个页面Tst1.aspx的页面声明如下<%@ Page language="c#" Codebehind="Tst1.aspx.cs" AutoEventWireup="false" Inherits="T1.Tst1" %>Codebehind="Tst1.aspx.cs" 表明经编译此页面时使用哪一个代码文件Inherits="T1.Tst1" 表用运行时使用哪一个隐藏类

50.什么是viewstate，能否禁用？是否所用控件都可以禁用

Viewstate是保存状态的一种机制，EnableViewState属性设置为false即可禁用

ViewState 是由 ASP.NET 页面框架管理的一个隐藏的窗体字段。当 ASP.NET 执行某个页面时，该页面上的 ViewState 值和所有控件将被收集并格式化成一个编码字符串，然后被分配给隐藏窗体字段的值属性（即<input type=hidden>）。由于隐藏窗体字段是发送到客户端的页面的一部分，所以 ViewState 值被临时存储在客户端的浏览器中。如果客户端选择将该页面回传给服务器，则ViewState 字符串也将被回传。回传后，ASP.NET 页面框架将解析ViewState字符串，并为该页面和各个控件填充 ViewState属性。然后，控件再使用 ViewState 数据将自己重新恢复为以前的状态

51.当发现不能读取页面上的输入的数据时很有可能是什么原因造成的？怎么解决

很有可能是在Page_Load中数据处理时没有进行Page的IsPostBack属性判断

52.请解释一个WEB页面中代码执行次序。

Init,Load,PreRender事件执行顺序:

1）控件的Init事件

2）控件所在页面的Init事件

3）控件所在页面的Load事件

4）控件的Load事件

5）控件所在页面的PreRender事件

6）控件的PreRender事件

规律：

1）Init事件从最里面的控件（包括用户控件及普通控件）向最外面的控件（页面）引发，Load及PreRender等其他事件从最外面的控件向最里面的控件引发；

2）控件之间相同事件的执行顺序依控件在页面的位置按从左到右，从上到下的先后顺序执行。

注意：
1）切记用户控件也被视为页面中的一个控件；

2）把用户控件作为单独的一个特殊页面来看，它本身及其所包含的控件同样遵守相同的规律；

3）有时在客户端程序（如javascript）中会用到客户端body对像的onload事件，注意这个客户端事件是最后执行，即在服务器端所有事件执行完后才执行.
53.请解释什么是上下文对象，在什么情况下要使用上下文对象

上下文对象是指HttpContext类的Current　属性，当我们在一个普通类中要访问内置对象(Response,Request,Session,Server,Appliction等)时就要以使用此对象

54.请解释转发与跳转的区别？

转发就是服务端的跳转A页面提交数据到B页面,B页面进行处理然后从服务端跳转到其它页面跳转就是指客户端的跳转

55.请解释ASP.NET中不同页面之间数据传递有那些方式？

session(viewstate) 简单，但易丢失,application 全局,cookie 简单，但可能不支持，可能被伪造input ttype="hidden" 简单，可能被伪造,url参数 简单，显示于地址栏，长度有限,数据库 稳定，安全，但性能相对弱

56.请解释ASP.NET中button linkbutton imagebutton 及hyperlink这四个控件之间的功别

1.Button和ImageButton用于将数据传递回服务器.

2.Hyperlink用于在页面之间导航

3.LinkButton用于将数据保存到服务器或访问服务器上的数据

4.LinkButton 控件具有与 HyperLink 控件相同的外观，不过却具有与 Button 控件相同的功能

57.请解释一下.NET多层应用程序中层与层之间以那几种方式进行数据传递。并解释你自己的项目中采用那种方式进行。

自定义类结构传数据
58.如果需要在GridView控件中的某一列中添加下拉列表框并绑定数据怎么解决？

后台的Rowdatabound事件可以进行绑定，比如这样

```
if（数据行）
{
        DropDownList ddl = (DropDownList)e.row.FindControl("DropDownListID");
        ddl.datasource = 数据源；(假定你已经设置了key和value绑定字段）
        ddl.databind();
```

}

59.请解释asp.net中的数据绑定与传统数据绑定有什么区别？

传统的数据绑定是一种“连接数据绑定”，即在数据绑定期间，客户端一直保持与数据库的连接，这种状态下，数据库的性能大受影响。asp.net的数据绑定是一种“非连接数据绑定”，即只在读取和更新数据的瞬间，才与数据库连接并交换数据，之后便可释放与数据库的连接，数据库的性能因此将大大提高。

60.请解释接口的显式实现有什么意义？

接口可以有静态成员、嵌套类型、抽象、虚拟成员、属性和事件。实现接口的任何类都必须提供接口中所声明的抽象成员的定义。接口可以要求任何实现类必须实现一个或多个其他接口。
1、因为显式接口成员执行体不能通过类的实例进行访问，这就可以从公有接口中把接口的实现部分单独分离开。如果一个类只在内部使用该接口，而类的使用者不会直接使用到该接口，这种显式接口成员执行体就可以起到作用。

2、显式接口成员执行体避免了接口成员之间因为同名而发生混淆。如果一个类希望对名称和返回类型相同的接口成员采用不同的实现方式，这就必须要使用到显式接口成员执行体。如果没有显式接口成员执行体，那么对于名称和返回类型不同的接口成员，类也无法进行实现。

61.您在什么情况下会用到虚方法？它与接口有什么不同？

当在继承类中想重写某一方法时会用到虚方法;虚方法是类的成员函数,接口相当于抽象类
62.Override与重载有什么区别？

重载是提供了一种机制, 相同函数名通过不同的返回值类型以及参数来表来区分的机制

override是用于重写基类的虚方法,这样在派生类中提供一个新的方法

覆写（Override）的两个函数的函数特征相同，重载（Overload）的两个函数的函数名虽然相同，但函数特征不同。

函数特征包括函数名，参数的类型和个数。1. override
使用 override 修饰符来修改方法、属性、索引器或事件。重写方法提供从基类继承的成员的新实现。由重写声明重写的方法称为重写基方法。重写基方法必须与重写方法具有相同的签名。

不能重写非虚方法或静态方法。重写基方法必须是虚拟的、抽象的或重写的。
也就是说，用 override 修饰符重写的基类中的方法必须是 virtual, abstract 或 override 方法

2. 重载

当类包含两个名称相同但签名不同的方法时发生方法重载。
使用重载方法的指南:

a. 用方法重载来提供在语义上完成相同功能的不同方法。

b. 使用方法重载而不是允许默认参数。默认参数的版本控制性能不好，因此公共语言规范(CLS)中不允许使用默认参数。

c. 正确使用默认值。在一个重载方法系列中，复杂方法应当使用参数名来指示从简单方法中假定的默认状态发生的更改。

d. 对方法参数使用一致的排序和命名模式。提供一组重载方法，这组重载方法带有递增数目的参数，以使开发人员可以指定想要的级别的信息，这种情况很常见。您指定的参数越多，开发人员就可指定得越详细。

e. 如果必须提供重写方法的能力，请仅使最完整的重载是虚拟的并根据它来定义其他操作。

// 下面具体解释一下这种模式，只有最后一个方法(参数最完整的方法)是虚方法，在继承了这个类的子类中只要重写(override)这个方法就行了。

63.怎样理解静态变量？

静态变量具有在某一个类中具有全局型.
64.向服务器发送请求有几种方式？

Post,Get.
65.DataReader与Dataset有什么区别？

DataReader和DataSet最大的区别在于,DataReader使用时始终占用SqlConnection,在线操作数据库.任何对SqlConnection的操作都会引发DataReader的异常.因为DataReader每次只在内存中加载一条数据,所以占用的内存是很小的..因为DataReader的特殊性和高性能.所以DataReader是只进的.你读了第一条后就不能再去读取第一条了. DataSet则是将数据一次性加载在内存中.抛弃数据库连接.读取完毕即放弃数据库连接.因为DataSet将数据全部加载在内存中.所以比较消耗内存.但是确比DataReader要灵活.可以动态的添加行,列,数据.对数据库进行回传更新操作

66.如果在一个B/S结构的系统中需要传递变量值，但是又不能使用Session、Cookie、Application，您有

几种方法进行处理？
使用Request["string"].Request.QueryString["flag"]
67.用.net做B/S结构的系统，您是用几层结构来开发，每一层之间的关系以及为什么要这样分层？

一般为3层 ， 数据访问层，业务层，表示层。数据访问层对数据库进行增删查改。业务层一般分为二层，业务表观层实现与表示层的沟通，业务规则层实现用户密码的安全等。表示层为了与用户交互例如用户添加表单。优点：分工明确，条理清晰，易于调试，而且具有可扩展性。缺点：增加成本。

68.软件开发过程一般有几个阶段？每个阶段的作用？

1）问题定义；2）可行性研究；3）需求分析；4）总体设计；5）详细设计；6）编码和单元测试；7）综合测试；8）软件维护。

69.微软推出了一系列的Application Block，请举出您所知道的Application Block并说明其作用？

SqlHelper 列如:SqlHelper.ExcuteDataSet()执行存储过程.
70.请列举一些您用到过的设计模式以及在什么情况下使用该模式？

抽象工廠
71.通过超链接怎样传递中文参数？

传递时用HttpUtility.UrlEncodeUnicode("中文参数")，获取时直接用Request.QueryString["参数"]就行了<%@ import namespace="System.Web.Util"%>

72.请编程遍历页面上所有TextBox控件并给它赋值为string.Empty？

```
foreach (Control  a in this.Page.Form.Controls)                {                        string name
= a.GetType().Name;                        if (a.GetType().Name == "TextBox")
        {                                TextBox mm = a as TextBox;
mm.Text = string.Empty;
}                                                                                        }
```

73.请编程实现一个冒泡排序算法？

```
private static int[] Sort(int[] arrayNum)                {                        int i, j,
k;                        bool exchange;                        for (i = 1; i < arrayNum.Length;
i++)                        {                                exchange =
false;                                for (j = arrayNum.Length - 1; j >= i; j-
-)                                {                                        if (arrayNum[j - 1] >
arrayNum[j])                                        {
k = arrayNum[j];                                                arrayNum[j] = arrayNum[j -
1];                                                arrayNum[j - 1] =
k;                                                exchange =
true;                                        }
}                                if (!exchange)                                        {
                        break;                                        }
}                        return arrayNum;                }
```

75.进程和线程的区别

进程是系统进行资源分配和调度的单位；线程是CPU调度和分派的单位，一个进程可以有多个线程，这些线程共享这个进程的资源。

76.成员变量和成员函数前加static的作用

它们被称为常成员变量和常成员函数，又称为类成员变量和类成员函数。分别用来反映类的状态。比如类成员变量可以用来统计类实例的数量，类成员函数负责这种统计的动作。

77.堆和栈的区别

栈：由编译器自动分配、释放。在函数体中定义的变量通常在栈上。堆：一般由程序员分配释放。
用new、malloc等分配内存函数分配得到的就是在堆上。栈是机器系统提供的数据结构，而堆则是C/C++函数库提供的。栈是系统提供的功能，特点是快速高效，缺点是有限制，数据不灵活；而栈是函数库提供的功能，特点是灵活方便，数据适应面广泛，但是效率有一定降低。栈是系统数据结构，对于进程/线程是唯一的；堆是函数库内部数据结构，不一定唯一。不同堆分配的内存无法互相操作。栈空间分静态分配和动态分配两种。静态分配是编译器完成的，比如自动变量(auto)的分配。动态分配由alloca函数完成。栈的动态分配无需释放(是自动的)，也就没有释放函数。为可移植的程序起见，栈的动态分配操作是不被鼓励的！堆空间的分配总是动态的，虽然程序结束时所有的数据空间都会被释放回系统，但是精确的申请内存/释放内存匹配是良好程序的基本要素。

78. C#中 property 与 attribute的区别，他们各有什么用处，这种机制的好处在哪里？

在C#中有两个属性，分别为Property和Attribute， Property比较简单，就是我们常用的get和set，主要用于为类中的private和protected变量提供读取和设置的接口。Attribute用来说明这个事物的各种特征的一种描述。而Attribute就是干这事的。它允许你将信息与你定义的C#类型相关联，作为类型的标注。这些信息是任意的，就是说，它不是由语言本身决定的，你可以随意建立和关联任何类型的任何信息。你可以作用属性定义设计时信息和运行时信息，甚至是运行时的行为特征。关键在于这些信息不仅可以被用户取出来作为一种类型的标注，它

更可以被编译器所识别，作为编译时的一种附属条件参加程序的编译。定义属性：属性实际上是一个派生自System.Attribute基类的类。System.Attribute类含有几个用于访问和检查自定义属性的方法。尽管你有权将任何类定义为属性，但是按照惯例来说，从System.Attribute派生类是有意义的

79.C#可否对内存进行直接的操作？

可以,我们知道，.NET相比C++最值得称赞的是他的GC（垃圾回收机制）。GC会在系统空闲或内存不足的时候自动回收不再被使用的对象。因此，我们不再需要向C++编程一样处处小心内存泄漏。同时，为了提高内存的使用效率，GC在回收内存的时候，会对内存进行整理，有些类似硬盘整理的原理。从而导致对象被在内存中移位。

80.维护数据库的完整性、一致性、你喜欢用触发器还是自写业务逻辑？为什么

尽可能用约束（包括CHECK、主键、唯一键、外键、非空字段）实现，这种方式的效率最好；其次用触发器，这种方式可以保证无论何种业务系统访问数据库都能维持数据库的完整性、一致性；最后再考虑用自写业务逻辑实现，但这种方式效率最低、编程最复杂，当为下下之策。

81.ADO.NET相对于ADO等主要有什么改进？

ADO以Recordset存储，而ADO.NET则以DataSet表示。Recordset看起来更像单表，如果让Recordset以多表的方式表示就必须在SQL中进行多表连接。反之，DataSet可以是多个表的集合。ADO 的运作是一种在线方式，这意味着不论是浏览或更新数据都必须是实时的。ADO.NET则使用离线方式，在访问数据的时候ADO.NET会利用XML制作数据的一份幅本，ADO.NET的数据库连接也只有在这段时间需要在线。由于ADO使用COM技术，这就要求所使用的数据类型必须符合COM规范，而ADO.NET基于XML格式，数据类型更为丰富并且不需要再做COM编排导致的数据类型转换，从而提高了整体性能。

82.C#中要使一个类支持FOREACH遍历，实现过程怎样？

若要循环访问集合，集合必须满足特定的要求。例如，在下面的 foreach 语句中：

```
foreach (ItemType item in myCollection)
```

myCollection 必须满足下列要求：

集合类型：
必须是 interface、class 或 struct。

必须包括返回类型的名为 GetEnumerator 的实例方法，例如 Enumerator。

Enumerator 类型（类或结构）必须包含：

一个名为 Current 的属性，它返回 ItemType 或者可以转换为此类型的类型。属性访问器返回集合的当前元素。

一个名为 MoveNext 的 bool 方法，它递增项计数器并在集合中存在更多项时返回 true。

有三种使用集合的方法：
1.使用上述指导创建一个集合。此集合只能用于 C# 程序。

2.使用上述指导创建一个一般集合，另外实现 IEnumerable 接口。此集合可用于其他语言（如 Visual Basic）。

3.在集合类中使用一个预定义的集合。

83.接口和抽象类有什么区别？你选择使用接口和抽象类的依据是什么？

接口是一个纯粹的抽象类，没有任何实际的东西，只是定义了一个框架，而抽象类里面可以有实际的一个方法，并不要求所有的方法都是抽象的。可以实现一个接口中的所有方法，也可以继承一个抽象的类，然后覆写其中的方法。接口一般只有方法，而没有数据成员或属性。抽象类有方法，也有数据成员或属性,一般情况下，优先考虑用接口，只有当可能要访问到数据成员或属性时，用抽象类。

84.自定义控件和一般用户控件的异同？如果要用这两者之一，你会选择哪种？为什么

用户控件模型适合创建内部，应用程序特定的控件，而自定义控件模型更适合创建通用的和可再分发的控件
85.大概描述一下ASP.NET服务器控件的生命周期

1. 初始化 - Init 事件 (OnInit 方法)

2. 加载视图状态 - LoadViewState 方法

3. 处理回发数据 - LoadPostData 方法:s对实现 IPostBackDataHandler 接口的控件，即可以自动加载回发数据的控件，如 TextBox, DropDownList 等。

4. 加载 - Load 事件 (OnLoad 方法)

5. 发送回发更改通知 - RaisePostDataChangedEvent 方法 :对实现 IPostBackDataHandler 接口的控件，即可以自动加载回发数据的控件。 在第 3 步中加载回发数据，如果回发前后数据发生更改，则在这一步触发相应的服务端事件。

6. 处理回发事件 - RaisePostBackEvent 方法:对实现 IPostBackEventHandler 接口的控件，即能引起回发的控件，如 Button, LinkButton, Calendar 等

7. 预呈现 - PreRender 事件 (OnPreRender 方法)

8. 保存视图状态 - SaveViewState 方法

9. 呈现 - Render 方法

10. 处置 - Dispose 方法

11. 卸载 - UnLoad 事件 (OnUnLoad 方法)

86.UML

统一建模语言（ U M L）是一个通用的可视化建模语言，用于对软件进行描述、可视化处理、构造和建立软件系统制品的文档。它记录了对必须构造的系统的决定和理解，可用于对系统的理解、设计、浏览、配置、维护和信息控制。

87.谈谈final, finally, finalize的区别。

final 修饰符用于指定类不能扩展或者方法或属性不能重写。它将防止其他类通过重写重要的函数来更改该类的行为。带有 final 修饰符的方法可以由派生类中的方法来隐藏或重载。finally 块用于清除在 try 块中分配的任何资源。控制总是传递给 finally 块，与 try 块的存在方式无关。finalize允许 Object 在“垃圾回收”回

收 Object 之前尝试释放资源并执行其他清理操作。

88.&和&&的区别。

&&为逻辑与，输出结果为布尔（真假）型； &为按位与，输出结果为数值型。

89.GC是什么 为什么要有GC

GC是垃圾收集器。这个类别中的方法会影响什么时候对对象执行内存回收，以及什么时候释放对象所配置的资源。这个类别的属性会提供系统中可用内存的总数量以及配置至对象的内存之年龄分类或层代等等相关信息。内存回收行程会追踪并重新利用在 Managed 内存中配置的对象。内存回收行程会定期执行内存回收来重新利用配置给对象的内存 (该对象并无有效的参考)。当没有可用的内存来因应内存的要求时，会自动发生内存回收。或者，应用程序可以使用 Collect 方法来强制进行内存回收。

内存回收包含下列步骤：
a.内存回收行程会搜寻在 Managed 程序代码中所参考的 Managed 对象。

b.内存回收行程会尝试最终处理未参考的对象。

c.内存回收行程会释放未参考并且回收其内存的对象。

90.Math.round(11.5)等於多少 Math.round(-11.5)等於多少

Math.round(11.5)==12;Math.round(-11.5)==-11;round方法返回与参数最接近的长整数，参数加1/2

求其floor

91.short s1 = 1; s1 = s1 + 1;有什么错 short s1 = 1; s1 += 1;有什么错

short s1 = 1; s1 = s1 + 1; （s1+1运算结果是int型，需要强制转换类型）

short s1 = 1; s1 += 1;（可以正确编译）

92.数组有没有length()这个方法 String有没有length()这个方法:

在c#数组和string只有length属性，没有方法

93.abstract class和interface有什么区别

abstract 修饰词可用于类别、方法、属性、索引子 (Indexer) 和事件。在类别宣告里使用 abstract

饰词，表示该类别只是当做其它类别的基底类别而已。成员如果标记为抽象，或是包含在抽象类(Abstract Class) 内，则必须由衍生自此抽象类别的类别实作这个成员。

在静态属性上使用 abstract 修饰词是错误的。

在抽象方法宣告中使用 static 或 virtual 修饰词是错误的。

接口只包含方法、委派或事件的签章。方法的实作 (Implementation) 是在实作接口的类别中完成，

94.是否可以继承String类

否,无法继承自密封类

95.try {}里有一个return语句，那么紧跟在这个try后的finally {}里的code会不会被执行，什么时候被执行，在return前还是后

会,在return后.

96.两个对象值相同(x.equals(y) == true)，但却可有不同的hash code，这句话对不对

不对,有相同的GetHashCode();

97.当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那

这里到底是值传递还是引用传递
引用传递
98.swtich是否能作用在byte上，是否能作用在long上，是否能作用在String上

都可以
99.编程题: 写一个Singleton出来。
100.c#中的三元运算符是

a?b:c
101.当整数a赋值给一个object对象时，整数a将会被

装箱

102.类成员有_____种可访问形式?

this.;new Class().Method;
103.public static const int A=1;这段代码有错误么？是什么？

const不能用static修饰
104.float f=-123.567F; int i=(int)f; i的值现在是

-123
105.委托声明的关键字是_____ delagete
106.用sealed修饰的类有什么特点？

密封，不能继承sealed 修饰词可套用至类别 (Class)、执行个体方法 (Instance Method) 和属性。密封类别无法被继承。密封方法会覆写基底类别 (Base Class) 中的方法，但在任何衍生类别中却无法进一步覆写密封方法本身。当套用至方法或属性时，sealed 修饰词必须一律和 override (C# 参考) 搭配使用。

107.在Asp.net中所有的自定义用户控件都必须继承自

System.Web.UI.UserControl
108.在.Net中所有可序列化的类都被标记为_____

[Serializable]
109.在.Net托管代码中我们不用担心内存漏洞

GC

110..当类T只声明了私有实例构造函数时，则在T的程序文本外部，___不可以___（可以 or 不可以）从T

派生出新的类，不可以 直接创建T的任何实例。

111.下面这段代码有错误么？

```
錯誤  switch (i){  case():CaseZero();break;  case 1:  CaseOne();break;  case 2:  dufault;
//wrongCaseTwo();break;
  }
```

112.在.Net中，类System.Web.UI.Page 可以被继承么？

可以

113.在c#中using和new这两个关键字有什么意义，请写出你所知道的意义？

new 关键词可当做运算子、修饰词或条件约束 (Constraint) 使用。

用来建立对象并叫用 (Invoke) 建构函式

```
Class1 o  = new Class1();
```

当 new 关键词做为修饰词时，会明确隐藏继承自基底类别的成员。隐藏继承的成员表示成员的衍生版本取代了基底类别版本。可以不使用 new 修饰词隐藏成员，但这么做会产生警告。使用 new 明确隐藏成员会隐藏这个警告，并记录使用衍生版本来替代。

new 条件约束 (Constraint) 指定在泛用类别宣告中的任何型别参数，都必须具有公用的无参数建构函式。当您的泛用类别建立型别的新执行个体时，将此条件约束套用至型别参数

```
class ItemFactory<T> where T : new(){
          public T GetNewItem()

           {

                return new T();

             }

}
```

using 关键词有两种主要的用法：

做为指示词，此时它是用来建立命名空间的别名，或是用来汇入在其它命名空间中定义的型别。请参阅 using 指示词。

做为陈述式，此时它是用来定义一个范围，对象会在此范围结尾处进行处置 (Dispose)。请参阅 using 陈述式。

114.谈谈类和结构的区别？

类是引用类型、结构是值类型结构与类别使用的语法几乎相同，不过结构的限制比类别多：结构执行个体字段宣告不可使用初始设定式，即使结构上的静态字段可初始化也一样结构不可宣告预设建构函式，即没有参数的建构函式或解构函式结构有下列属性：结构是值类型，而类别为引用类型当传递结构到方法上时，是以传值而非以当做参考的方式传递

与类别不同的是，结构不需使用 new 运算子就能执行个体化

结构可以宣告建构函式，但是必须采用参数
结构无法从另一个结构或类别继承而来，且它不能成为类别的基底。所有结构都是从继承自 System.Object 的 System.ValueType 直接继承而来

结构可实作接口
在结构里初始化执行个体字段是错误的
115.一个长度为10000的字符串，通过随机从a-z中抽取10000个字符组成。请用c＃语言编写主要程序来

实现。

```
string[] LetterList = new string[] { "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l",
"m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z" };

                        StringBuilder sbRe = new StringBuilder();                        Random
rd=new Random();                          for (int i = 0; i < 10000; i++)
{                                  sbRe.Append(LetterList[rd.Next(26)]);                          }
               Console.Write(sbRe.ToString());
```

116.对于这样的一个枚举类型：

enum Color:byte{Red, Green, Blue, Orange}试写一段程序显示出枚举类型中定义的所有符号名称以及它们对应的数值。

```
foreach (string aa in Enum.GetNames(typeof(Color)))

                        {
Console.Write(aa);                                  Console.Write(""r"n");
}                        foreach (byte aa in Enum.GetValues(typeof(Color)))
{                                  Console.Write(aa);
Console.Write(""r"n");                           }
```

117.写一个实现对一段字符串翻转的方法，附加一些条件，如其中包括“，”、“.”.

```
  string kk = "abc',.字符串";

                        string reKK = "";                          for (int i = 0; i < kk.Length;
i++)                          {                                  reKK = kk[i] +
reKK;                          }
                        Console.Write(reKK);
```

118.什么是虚函数？

```
Virtual CallSomeOne();
```

119.什么是抽象函数？

```
public abstract void CallSomeOne();
```

120.触发器的作用

触发器是一种特殊的存储过程，它在插入，删除或修改特定表中的数据时触发执行，它比数据库本身标准的功能有更精细和更复杂的数据控制能力。数据库触发器有以下的作用：安全性。可以基于数据库的值使用户具有操作

数据库的某种权利。可以基于时间限制用户的操作，例如不允许下班后和节假日修改数据库数据。
可以基于数据库中的数据限制用户的操作，例如不允许股票的价格的升幅一次超过10%。

审计。可以跟踪用户对数据库的操作。审计用户操作数据库的语句。把用户对数据库的更新写入审计表。实现复杂的数据完整性规则。实现非标准的数据完整性检查和约束。触发器可产生比规则更为复杂的限制。与规则不同，触发器可以引用列或数据库对象。例如，触发器可回退任何企图吃进超过自己保证金的期货。提供可变的缺省值。
实现复杂的非标准的数据库相关完整性规则。触发器可以对数据库中相关的表进行连环更新。例如，在auths表author_code列上的删除触发器可导致相应删除在其它表中的与之匹配的行。

在修改或删除时级联修改或删除其它表中的与之匹配的行。
在修改或删除时把其它表中的与之匹配的行设成NULL值。

在修改或删除时把其它表中的与之匹配的行级联设成缺省值。
触发器能够拒绝或回退那些破坏相关完整性的变化，取消试图进行数据更新的事务。当插入一个与其主健不匹配的外部键时，这种触发器会起作用。例如，可以在 books.author_code列上生成一个插入触发器，如果新值与auths.author_code列中的某值不匹配时，插入被回退。

121.求以下表达式的值，写出您想到的一种或几种实现方法：

1-2+3-4+……+m

```
public static int Foo(int i)

            {                       int result = 0;                       for (int j = 1; j <
i; j++)                     {                           if (j % 2 ==
0)                          {                                   result = result -
j;                          }                               else if (j % 2 ==
1)                          {                                   result = result +
j;                          }                   }                       return result;
            }
```

122.C#中的委托是什么？事件是不是一种委托？

委派是参考方法的一种型别，一旦将一个方法指定给某委派，则该委派的行为便会与该方法完全相同。委派方法可以当做任何其它方法一般使用，也有参数和传回值，
任何与委派的签名码（即由传回型别和参数所组成）相符的方法，都可指派给委派。如此便可利用程序设计的方式变更方法呼叫，也可将新的程序代码外挂至现有类别中。只要您知道委派的签名码为何，即可指派自己的委派方法。

由于委派能够将方法当做参数来参考，使得委派很适合用来定义回呼方法。例如，您可以将比较两个对象的方法参考传递至排序算法。分开撰写比较程序代码，可让您撰写更适合通用的算法。委派有下列属性：
a.委派与 C++ 函式指标类似，但为型别安全。

b.委派允许将方法当做参数传递。

c.委派可用于定义回呼方法。

d.您可将委派链接在一起，例如，可在单一事件上呼叫多个方法。

123.C#中，执行以下代码后S的结果：

string[] a=new string[7];

aa[0]="33"; aa[6]="66";string s=""; foreach(string m in aa) s+=m;
3366
124.适配器datadapter的作用

表示 SQL 命令集和数据库连接，用来填入 DataSet 并更新数据来源。

DataAdapter 是 DataSet 和数据来源之间的桥接器 (Bridge)，用来撷取和储存数据。DataAdapter 藉由对应 Fill (它会变更 DataSet 中的数据来符合数据来源中的数据) 和 Update (它会变更数据来源中的数据来符合 DataSet 中的数据) 来提供这个桥接器。

125.所有类中最顶层的类是哪个类

System.Object
126.跳转页面有哪几种方式?

Response.Redirect(""); Server.Transfer("");
Server.Execute("");
127.类包含哪些成员

1.建构函式 2.解构函式 3.常数 4.字段 5.方法 6.属性 7.索引子 8.运算子 9.事件 10.委派

11.类别 12.界面 13.结构

128.索引器

索引子 (Indexer) 允许使用与数组相同的方式来索引类别或结构的执行个体。索引子除了其存取子需要使用参数以外，其余特性都与属性相似。

a.索引子让对象能以类似数组的方式来索引。

b.get 存取子会传回一个值。set 存取子会指定一个值。

c.this 关键词的用途为定义索引子。

d.value 关键词是用来定义 set 索引子所指定的值。

e.索引子不需要以整数值来索引；您可以决定如何定义特定的查询机制。

f.索引子可以多载。

g.索引子可以具有一个以上的型式参数，例如，在存取二维数组时便是如此。

```
class SampleCollection<T>{       private T[] arr = new T[100];       public T this[int i]
{               get               {                        return arr[i];
}               set               {                        arr[i] = value;              }       }
}
```

129. HYPERLINK 和 linkbotton控件的差别

HYPERLINK导航。

使用 LinkButton 控件，在 Web 网页上建立超级链接样式按钮。LinkButton 控件具有与 HyperLink 控件相同的外观，但拥有与 Button 控件相同的功能。

130.DataReader与Dataset有什么区别？

执行查询时会传回结果，并一直储存于客户端上的网络缓冲区中，直至您使用 DataReader 的 Read 方法要求它们为止。使用 DataReader 可以提高应用程序的效能，方法是立即撷取可用的数据，及（依预设）一 次只将一个数据列储存到内存中，从而减少系统负荷

DataSet 是以常驻内存表示的数据，不论内含数据来源为何，都可提供一致的关系型程序设计模型。DataSet 表示一组完整的数据，包括内含、排序和约束数据的数据表，以及数据表间的关联性。

131.简要说出private、protected的区别

private 存取只限于包含类别。

protected 存取只限于包含的类别或衍生自包含类别的型别。
132. 说出下面几个函数的区别:

private void test(string str){…}

private void test(ref string str){…}

private void test(out string str){…}

out 关键词会导致以传址 (By Reference) 方式传递自变量。这点与 ref 关键词相类似，除了 ref 需要在传递变量之前先初始化变量以外。若要使用 out 参数，方法定义和呼叫方法都必须明确使用 out 关键词。

133.写代码：取得服务器时间并显示（弹出消息框即可），要求完全在服务器端实现(提示：在C#中使用

Response.Write()方法)

Response.Write("<script >alert('當前時間:" + DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss") + "') </script>");

134.说出下面各处正确或错误的理由（写在该行）

class Test { int x; static int y; void F() { x = 1; //ok y = 1; //ok } static void G() { x = 1; //error 非静態成員不能在靜態

y = 1; //ok } static void Main() { Test t = new Test();t .x = 1; // Ok t .y = 1; // Error Test.x = 1; // Error Test.y = 1; // Ok } }
135.简要的谈谈你对多态的理解，并简单的实现一个多态的例子
136.下面关于索引的描述中。哪些是正确的？（AB ）

A、索引能够加快查询的速度

B、索引可能会降低数值的更新速度

C、应该尽可能多的为表建立索引

137.请描述一下在.net中Forms认证模式下，怎样用代码（用户名单存在数据库中，而不是WebConfig中）

实现一个基于角色的授权？
138.在Vs.net中，怎样配置数据库连接，可以使开发环境到应用环境迁移数据库时不修改程序？

Machine.config
139.假设有一个数据库字段name,需要在网页中只显示该字段的姓，而隐藏名字，类似“张**”形式，请写

出数据绑定的自定义表达式。
140.请简单描述一下制作一个用户控件（UserControl）的过程。
141.您有没有制作过自定义的webcontrol（注意不是用户控件）？如果有，请描述一下自定义控件制作基本

过程，主要重载哪些方法？。
142.请描述构成ADO.net的主要对象以及他们的作用。
143.你编写一段程序来判断服务器请求是get,还是post请求？

```
string mm=Request.ServerVariables["REQUEST_METHOD"];

       string ll = Request.ServerVariables["QUERY_STRING"];
```

144.如果需记录类的实例个数,该如何实现,请写一个简单的类于以证明.

```
public class TestA

      {               public static int Count;               public TestA()
{                       Count = Count + 1;              }               public int
CountA               {                       get { return Count; }               }       }
TestA testa = new TestA();

                      TestA testb = new TestA();                      TestA testb1 = new
TestA();                      TestA testb2 = new TestA();                      TestA testb3 =
new TestA();                      TestA testb4 = new TestA();                      TestA testb5
= new TestA();
                      Console.Write(testb.CountA);
```

145.是否了解 URL Rewrite？请简要说明其原理和在 ASP.NET 中的实现方式。

URL 重写是截取传入 Web 请求并自动将请求重定向到其他 URL 的过程.
146. ASP.NET 中如何调用 Web Service 的方法？

```
添加引用               TestService.Service ts = new TestService.Service();               DataSet
ds = ts.GetUserInfo("", "");
        Response.Write(ds.Tables[0].Rows.Count);
```

147.ViewState 的作用和实现方式？

ViewState 属性提供 Dictionary 对象来保留对相同网页的多个要求之间的值。这是网页用来在来回往返之间保留网页和控件属性值的预设方法。处理网页时，网页目前的状态和控件会杂凑至字符串中，并且储存在网页中当做一个隐藏字段，或是如果储存在 ViewState 属性的数据量超过 MaxPageStateFieldLength 属性中指定的值时，则会当做多个隐藏字段。网页回传到服务器时，网页会在网页初始化时剖析检视状态字符串，并且还原网页

中的属性资

148.如何实现页面分段缓存？

页面分段缓存
页面分段缓存是用用户控件实现的。通过为页面每个区域创建单独的用户控件来定义页面的区域。在每个用户控件中，可以使用Output-Cache指令指出如何缓存控件的输出。

注意：
1、注意分段缓存不支持Location特性；缓存页面分段惟一合法的地方是web服务器。这是因为分段缓存在ASP.NET中是新的功能，所以浏览器和代理服务器不支持。

2、分段缓存有另外一个在页面缓存中没有的特性——VaryByControl。VaryByControl特性允许指定一个由分号分隔的字符串列表，代表用户控件内使用的控件的名称；ASP.NET将针对值的每个不同的组合生成用户构件的一个缓存版本。

页面分段缓存的限制
1、如果为用户控件启用了缓存，就不能在包含它的页面中通过程序访问此控件了；

例如：ctlContents.Message = "Hello!"

2、不应该对缓存的用户控件使用数据绑定语法；

不允许：<myControls:PageContents CategoryID='<%# CategoryID%>' Runat='Server'/> 会产生一个错误信息； 允许： <myControls:PageContents CategoryID='2' Runat='Server'/>

149.你是否知道下列名字：.NET Pet Shop，IBuySpy Store，DotNetNuke，NUnit，Data Access Application

Block？说说它们分别都是什么。
150.如何实现XML系列化（给出简单的代码示例）

XML 序列化只会将对象的公用字段和属性值序列化为 XML 数据流。

XML 序列化不会转换方法、索引子、私用字段或只读属性 (只读集合则除外)。若要序列化对象的所有字段和属性 (包括公用和私用的)，请使用 BinaryFormatter，而不要使用 XML 序列化。

```
public class TestXML

      {              public string UserName;              public string Address;
public string Company;
             public string Description;

      }
XmlSerializer serializer =

                      new XmlSerializer(typeof(TestXML));              string FilePath =
Server.MapPath("XML/woody.xml");              TestXML tx = new TestXML();
tx.Address = "MITC";              tx.Company = "MDS";              tx.Description =
"WOODY";              tx.UserName = "WU";              Stream writer = new FileStream(FilePath,
```

FileMode.Create);　　serializer.Serialize(writer, tx);　　writer.Close();

151.你知道 AJAX 吗？说说它的特点和一般实现方式

152.写出一段利用XMLHTTP工作的简单代码

153.如何定义嵌套的CSS样式类

table.hover {　　background-color: white; border-right:1px solid #f1f1f1 }

154..NET Remoting的工作原理是什么？请简要地写出一个.NET Remoting的示例

分布式处理方式,在Windows操作系统中，是将应用程序分离为单独的进程。这个进程形成了应用程序代码和数据周围的一道边界。如果不采用进程间通信（RPC）机制，则在一个进程中执行的代码就不能访问另一进程。这是一种操作系统对应用程序的保护机制。然而在某些情况下，我们需要跨过应用程序域，与另外的应用程序域进行通信，即穿越边界.

Remoting的通道主要有两种：Tcp和Http。在.Net中，System.Runtime.Remoting.Channel中定义了IChannel接口。IChannel接口包括了TcpChannel通道类型和Http通道类型。它们分别对应Remoting通道的这两种类型。

TcpChannel类型放在名字空间System.Runtime.Remoting.Channel.Tcp中。Tcp通道提供了基于Socket的传输工具，使用Tcp协议来跨越Remoting边界传输序列化的消息流。TcpChannel类型默认使用二进制格式序列化消息对象，因此它具有更高的传输性能。HttpChannel类型放在名字空间System.Runtime.Remoting.Channel.Http中。它提供了一种使用Http协议，使其能在Internet上穿越防火墙传输序列化消息流。默认情况下，HttpChannel类型使用Soap格式序列化消息对象，因此它具有更好的互操作性。通常在局域网内，我们更多地使用TcpChannel；如果要穿越防火墙，则使用HttpChannel.

155.从程序请求远程http站点，有哪些可用的类？

WebRequest request = WebRequest.Create(PageUrl);　　WebResponse response = request.GetResponse();　　Stream resStream = response.GetResponseStream();　　StreamReader sr = new StreamReader(resStream, System.Text.Encoding.Default);　　string KK = sr.ReadToEnd();
resStream.Close();
sr.Close();

156.对于Web Services,.NET Remoting,MSMQ,Enterprise Services这四个中接触过多少？能否简要的介绍

他们的特点

157.可否简要的介绍asp.net 2.0 Membership,WebPart和C#的匿名函数和泛型，希望可以简要地阐述其中

的特点 Membership:

Membership 类别是用在 ASP.NET 应用程序中，以验证使用者认证，并管理使用者设定，例如密码和电子邮件地址。Membership 类别可以单独使用，也可以结合 FormsAuthentication 使用，以建立验证 Web 应用程序或网站使用者的完整系统。Login 控件会封装 Membership 类别，以提供验证使用者的便利机制。

Membership 类别提供多项功能，以供进行：建立新使用者。

将成员资格信息（使用者名称、密码、电子邮件地址，以及支持数据）存放于 Microsoft SQL Server 或替代数据存放区中。

验证造访您网站的使用者。您可以透过程序设计方式验证使用者，您也可以使用 Login 控件，建立需要少数或不需要程序代码的完整验证系统。

管理密码，其中包括建立、变更、撷取，及重设等动作。您也可以选择设定 ASP.NET 成员资格，向遗忘密码的使用者要求密码问题和解答，以验证密码重设或撷取要求。

WebPart:
ASP.NET Web 组件是用于建立网站的整合式控件集合，可让使用者直接从浏览器修改 Web 网页的内容、外观和行为。这些修改可以套用至网站上的所有使用者或个别使用者。当使用者修改页面和控件时，可以储存这些设定，以保留使用者的个人偏好设定，在未来的浏览器工作阶段 (Session) 使用，这个功能称为个人化。这些 Web 组件能力让开发人员可以授权使用者动态地个人化 Web 应用程序，而不需要开发人员或管理员介入。

藉由使用 Web 组件控件集合，开发人员可让使用者执行下列功能：

个人化页面内容。使用者可以将新的 Web 组件控件加入至页面，或是移除、隐藏或最小化控件，如同使用普通窗口一样。

个人化页面配置。使用者可以将 Web 组件控件拖曳至页面上的不同区域，或变更其外观、属性和行为。

汇出和汇入控件。使用者可以汇入或汇出 Web 组件控件设定以用于其它页面或网站，并保留控件中的属性、外观或甚至数据，如此可让使用者减少数据输入和组态设定的需要。

建立连接。使用者可以在控件间建立连接，例如，图表控件可以将股票行情实时广告牌控件中的数据显示为图表。使用者不但能个人化连接本身，也能个人化图表控件显示数据之外观和细节的方式。管理和个人化网站层级设定。授权的使用者可以设定网站层级设定，决定谁可以存取网站或网页，以及设用者共享，并防止不是管理员的使用者个人化共享的控件。
与区域变量不同的是，外部变量的存留期会延续，直到参考匿名方法的委派可进行内存回收为止。n 的参考是在建立委派时所撷取。

匿名方法:

将程序代码区块当做委派参数传递的一种方式藉由使用匿名方法，您无须另外建立方法，因而可以减少在执行个体化委派时所需要另外撰写的程序代码。
匿名方法不能存取外部范围的 ref 或 out 参数。

anonymous-method-block 内不能存取 Unsafe 程序代码。

```
button1.Click += delegate(System.Object o, System.EventArgs e)
{ System.Windows.Forms.MessageBox.Show("Click!"); };
```

泛型:

泛型是指一些类别、结构、接口和方法，而它们具有其储存或使用的一或多个型别之替代符号 (型别参数)。泛型集合类别可能会将型别参数当做它所储存的对象型别之替代符号；这些型别参数会以其字段型别及其方法的参数型别之形式出现。泛型方法可能会将它的型别参数当做其传回值的型别或是其中一个正规参数的型别使用。

System.Collections.Generic 和 System.Collections.ObjectModel 命名空间中的泛型集合类别
158.如何理解责任链和命令模式的异同？

159.数据量下的列表显示分页如何处理？
160.附件上传后的保存方式以及如何浏览？

```
string FileVName = DateTime.Now.ToString("yyyyMMddHHmm") + WebCom.RandomnAlpha(7) + fileExt;
```

```
file_up_load.PostedFile.SaveAs(loadFilePath + FileVName);
```

161.面向对象中的基类指什么，什么时候用到基类

162.存储过程跟SQL语句比较，各有什么优点和缺点？

163.描述怎样区分使用ExecuteNonQuery和ExecuteScalar方法？

ExecuteNonQuery:

使用 ExecuteNonQuery 以执行数据库目录 (Catalog) 作业 (例如，查询数据库结构或建立如数据表的数据库对象)，或藉由执行 UPDATE、INSERT 或 DELETE 陈述式变更数据库的数据。

虽然 ExecuteNonQuery 不传回任何数据列，但是对应至参数的任何输出参数或传回值会填入 (Populate) 资料。对 UPDATE、INSERT 和 DELETE 陈述式而言，传回值是受命令影响的数据列数目。对其他类型的陈述式而言，传回值为 -1。

ExecuteScalar:
执行查询，并传回查询所传回的结果集中第一个数据列的第一个资料行。会忽略所有其它的数据行和数据列。

164.SQL语句中是否用过Sum,Count,Top,Group By,Case...When这些关键字，请描述具体用法？

Sum加總;Count:記錄總數; Top:前多少條; Group By集合函數; Case...When條件語句

165.是否用过Xml Schema或者DTD，怎样使用一个Xml Schema或者DTD去校验一个xml的格式是否正确？

```
     XmlTextReader tr = new XmlTextReader(Server.MapPath("XML/woody.xml"));
XmlValidatingReader vr = new XmlValidatingReader(tr);                vr.ValidationType =
ValidationType.DTD;                vr.ValidationEventHandler += new
ValidationEventHandler(ValidationHandler);
                while (vr.Read()) ;
```

166.是否使用过Xsl样式表？解释xsl:apply-templates，xsl:call-template，xsl:choose，xsl:value-of

的用法？
XSL 之于 XML ,就像 CSS 之于 HTML。它是指可扩展样式表语言 (EXtensible Stylesheet Language)。这是一种用于以可读格式呈现 XML 数据的语言。XSL 实际上包含两个部分：

* XSLT - 用于转换 XML 文档的语言

*XPath - 用于在 XML 文档中导航的语言

XSLT 是指 XSL 转换 (XSL Transformation)，它是 XSL 最重要的部分。

XSLT 可以将 XML 文档转换为其它 XML 文档、XHTML 输出或简单的文本。这通常是通过将每个 XML 元素转换为 HTML 元素来完成的。由于 XML 标签是用户定义的，浏览器不知道如何解释或呈现每个标签，因此必须使用 XSL。XML 标签的意义是为了方便用户（而不是计算机）理解。

XSLT 还可以对 XML 树进行下列操作：

* 添加和删除元素

* 添加和删除属性

* 对元素进行重新排列或排序

* 隐藏或显示某些元素

* 查找或选择特定元素

XSL 语法

您可能还记得 XML 概述文章中提到过，所有 XML 文档都是以 XML 声明开头。XSL 样式表也是一样。任何 XSL 文档的第一行实际上都是 XML 声明：

<?xml version="1.0" encoding="ISO-8859-1"?>
XSL 是否与 XML 相同?

既是又不是。说是，是因为它们遵循相同的语法规则（只有少许差异，下面我将会讲到）。说不是，是因为它们的用途不同：XML 用于承载数据，而 XSL 则用于设置数据的格式。

在 XML 声明之后，就是 XSL 声明，例如：

<xsl:stylesheet>或<xsl:transform>

但是，在大多数实际情况下，XSL 声明看起来要稍微复杂一些：

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
这是因为，根据 W3C 联盟的建议，它还包含命名空间和 XSL 规范的版本。

XSL 声明与 XML 声明的不同之处在于，XML 声明只写一行，而且没有结束标签，而 XSL 声明必须包含结束标签，该标签表示 XSL 样式表结束：

</xsl:stylesheet>
请注意，这并不与 XML 语法规则冲突：XSL 样式表是完全有效的 XML 文档，因为它有一个根元素，由 <xsl:stylesheet> 标签指定。

在什么情况下应该使用 XSL?

XSL 在设计时有几个目标用途，这些用途使它成为某些开发情况下的完美解决方案，而对另一些开发情况则毫无用处。

XSL 专门用于处理 XML 文档，并且遵循 XML 语法。因此，它只能在支持 XML 的应用程序中与 XML 结合使用。最合适使用 XML 和 XSL 的情况是：Web 门户、新闻聚合器、社区网站或其它任何需要向各种设备及大 量客户端提供信息的 Web 应用程序。

XSLT 是一种基于模式匹配的语言。它会查找匹配特定条件的节点，然后应用相应的规则。因此，它不具备大多数编程语言的计算能力。例如，XSL 无法在运行时更改变量的值。它不应该用于从使用复杂公式的动态数据源（例如在线商店）来计算值。Web 编程语言更适于此用途。

XSL 不能代替或补充 CSS。它不应（也不能）用于设置 HTML 的样式。但是，您可以将其用于需要频繁重新设计可视化效果、频繁更换布局以及以灵活格式处理数据的网站。

XSL 不是内容管理工具。它不应（也不能）用于更改 XML 文档的内容或者编辑信息。但是，您可以将 XML 和

XSL 用于需要处理几种不同格式文档的内容管理系统。

167. 讲一讲你理解的web service,在dot net framework中，怎么很好的结合xml（讲概念就行了）

168.用C＃实现以下功能

a 产生一个int数组，长度为100，并向其中随机插入1-100，并且不能重复。

b 对上面生成的数组排序，需要支持升序、降序两种顺序

169.需要实现对一个字符串的处理,首先将该字符串首尾的空格去掉,如果字符串中间还有连续空格的话,仅保留一个空格,即允许字符串中间有多个空格,但连续的空格数不可超过一个.

```
FilterStr = Regex.Replace(FilterStr, "[\\s]{2,}", " ");
```

170.设有关系框架R(A,B,C,D)及其上的函数相关性集合Ｆ＝{B→A,BC→D},那么关系框架Ｒ最高是.....(a)

a. 第一范式的
b. 第二范式的
c. 第三范式的
d. BCNF范式的

172.设有关系EMP（ENO，ENAME，SALARY，DNO ），其中各属性的含义依次为职工号、姓名、工资和所在部门号，以及关系DEPT（DNO，DNAME，MANAGER），其中各属性含义依次为部门号、部门名称、部门经理的职工号。试用SQL语句完成以下查询：

a) 列出各部门中工资不低于600元的职工的平均工资。

```
select dno , avg(salary) as average from emp where salary>=600 group by dno
```

b) 查询001号职工所在部门名称。

```
select DNAME from dept where DNO = (select DNO from emp where eno=’001’)
```

或者

```
select d.dname from dept as d left jon emp as e on e.dno = d.dno where e.eno=’001’
```

c)请用SQL语句将“销售部”的那些工资数额低于600的职工的工资上调10%。

```
update EMP set SALARY =SALARY*(1+0.1) where SALARY<600 and DNO = ( select DNO from dept where dname= ‘销售部’ )
```

173. 程序补充

```
using System;class Point{         public int x,y,z;         public Point()
{                 x = 0;y = 0;z = 0;         }         public Point(int x, int y,int z)
{                 this.x=x;                 this.y=y;                 this.Z=y;         }
public override string ToString()         {                 return(String.Format("({0},{1},{2})",
x, y,z));         }}class MainClass{         static void Main()         {                 Point
p1 = new Point();                 Point p2 = new Point(10,20,30);
                Console.WriteLine("三维中各点坐标：");

                Console.WriteLine("点1的坐标为{0}", p1);
```

```
                Console.WriteLine("点2的坐标为{0}", p2);

        }}using System;class SwitchTest {          public static void Main()          {
                Console.WriteLine("邮箱种类: 1：免费邮箱 2：企业邮箱 3：VIP邮箱");

                Console.Write("请输入您的选择: ");

                string s = Console.ReadLine();                  int n =
int.Parse(s);                  int cost = 0;                switch(n)
{                                       case 0:
case 1:
<!--把cost赋值为0-->
cost = 0;

Break;                          case 2:
<!--把cost赋值为100-->
cost = 100;

Break;                          case 3:
<!--把cost赋值为300-->
cost = 300;

Break;                          default:
                                    Console.WriteLine("错误的选择.请输入1, 2,或者
3.");

                                    Break;                  }                  if (cost != 0)
                        Console.WriteLine("您选择的邮箱每年的费用为{0}元.", cost);

        }}
```

175. 设计模式

我们公司有很多产品，有些产品比较早，设计上不是非常完善。现在我们有一个系统整合了其它产品的身份认证，即其它产品的身份认证全部由该系统完成。现在的问题是，由于各个系统采用的密码加密算法不一致（都是不可逆算法），请你选择合适的设计模式，完成密码校验的功能。

176. 装箱、拆箱操作发生在: (C)

A.类与对象之间　　B.对象与对象之间

C.引用类型与值类型之间　　D.引用类型与引用类型之间

177. 用户类若想支持Foreach语句需要实现的接口是: (AB)

A.IEnumerable　　B.Ienumerator

C.ICollection　　D.ICollectData

178 .Net Framework通过什么与COM组件进行交互操作？(C)

A.Side By Side　　B.Web Service

C.Interop　　D.PInvoke

179. 装箱与拆箱操作是否是互逆的操作？(B)

A.是　　B.否

180.　以下哪个是可以变长的数组？(　　BD　　)

A.Array　　B.string[]

C.string[N]　　D.ArrayList

181.　用户自定义异常类需要从以下哪个类继承：(　A　　)

A.Exception　　B.CustomException

C.ApplicationException　　D.BaseException

182.　以下代码段中能否编译通过？请给出理由。

Try{}catch(FileNotFoundException e1){}catch(Exception e2){}catch(IOException e3){}Catch{}

不可以,

IOException继承于Exception

之前的 catch 子句已经取得所有属于此类别或超级类别 ('System.Exception') 的例外状况

183.　对于一个实现了IDisposable接口的类，以下哪些项可以执行与释放或重置非托管资源相关的应用程序定义的任务？(多选) (　　bcd　　)

A.Close　　B.Dispose　　C.Finalize

D.using　　E.Quit

184.　.Net依赖以下哪项技术实现跨语言互用性？(　C　　)

A.CLR　　B.CTS　　C.CLS　　D.CTT

185.　请问: String类与StringBuilder类有什么区别？为什么在.Net类库中要同时存在这2个类？

String 或 StringBuilder 对象之串连作业的效能是根据内存的配置频率而定。String 串连作业永远都会配置内存，而 StringBuilder 串连作业只有在 StringBuilder 对象缓冲区太小而无法容纳新数据时，才会配置内存。因此，如果要串连固定数目的 String 对象，最好使用 String 类别的串连作业。在这种情况下，编译器甚至可能将个别的串连作业结合成一个单一作业。如果要串连任意数目的字符串 (例如，如果循环串连任意数目的使用者输入字符串)，则对于串连作业来说最好使用 StringBuilder 对象。

186.　以下哪个类是int的基类？(　B　)

A.Int32　　B.Object　　C.ValueType　　D.Int16

187.　以下哪些可以作为接口成员？(多选) (　　ABD　　)

A.方法　　B.属性　　C.字段　　D.事件　　E.索引器F.构造函数　　G.析构函数

188.　以下关于ref和out的描述哪些项是正确的？(多选) (　　AC　　)

A.使用ref参数，传递到ref参数的参数必须最先初始化。

B.使用out参数，传递到out参数的参数必须最先初始化。

C.使用ref参数，必须将参数作为ref参数显式传递到方法。

D.使用out参数，必须将参数作为out参数显式传递到方法。

189.“访问范围限定于此程序或那些由它所属的类派生的类型”是对以下哪个成员可访问性含义的正确描述

(　　B　)

A.public　B.protected　　C.internal　　D.protected internal

190.class Class1

{　　private static int count = 0;　　static Class1()　　{　　count++;
}　　public Class1()　　{　　count++;　　}}Class1 o1 = new Class1();Class1 o2 = new Class1();

请问，o1.Count的值是多少？(　　B　)

A.1　　B.2　　C.3　　D.4

191.　abstract class BaseClass

{　　public virtual void MethodA()　　{　　}　　public virtual void MethodB()
{　　}}class Class1: BaseClass{　　public void MethodA(string arg)　　{　　}
public override void MethodB()　　{　　}}class Class2: Class1{　　new public void MethodB()　　{　　}}class MainClass{　　public static void Main(string[] args)
{　　Class2 o = new Class2();　　Console.WriteLine(o.MethodA());　　}}

请问，o.MethodA调用的是: (　　A　)

A.BaseClass.MethodA　　B.Class2.MethodA

C.Class1.MethodA　　D.都不是

192.　您需要创建一个ASP.NET应用程序，公司考虑使用Windows身份认证。所有的用户都存在于AllWin这个域中。您想要使用下列认证规则来配置这个应用程序：

1.匿名用户不允许访问这个应用程序。

2.所有雇员除了Tess和King都允许访问这个应用程序。

请问您应该使用以下哪一个代码段来配置这个应用程序？(　　)

A.　　<authorization>

B.　　<deny users=”allwin″tess, allwin″king”>
<allow users=”*”>
<deny users=””>
</authorization>

C.　　<authorization>
<deny users=”allwin″tess, allwin″king”>
<deny users=””>
<allow users=”*”>
</authorization>

D.　　<authorization>
<allow users=”allwin″tess, allwin″king”>
<allow users=”*”>
</authorization>

E. <authorization>
<allow users=”*”>
<deny users=”allwin″tess, allwin″king”>
</authorization>

193要创建一个显示公司员工列表的应用程序。您使用一个DataGrid控件显示员工的列表。您打算修改这个控件以便在这个Grid的Footer显示员工合计数。请问您应该怎么做？(BC)

A.重写OnPreRender事件，当Grid的Footer行被创建时显示合计数。

B.重写OnItemCreated事件，当Grid的Footer行被创建时显示合计数。

C.重写OnItemDataBound事件，当Grid的Footer行被创建时显示合计数。

D.重写OnLayout事件，当Grid的Footer行被创建时显示合计数。

194.创建ASP.NET应用程序用于运行AllWin公司内部的Web站点，这个应用程序包含了50个页面。

您想要配置这个应用程序以便当发生一个HTTP代码错误时它可以显示一个自定义的错误页面给用户。

您想要花最小的代价完成这些目标，您应该怎么做？(多选)(AD)

A.在这个应用程序的Global.asax文件中创建一个Application_Error过程去处理ASP.NET代码错误。

B.在这个应用程序的Web.config文件中创建一个applicationError节去处理ASP.NET代码错误。

C.在这个应用程序的Global.asax文件中创建一个CustomErrors事件去处理HTTP错误。

D.在这个应用程序的Web.config文件中创建一个CustomErrors节去处理HTTP错误。

E.在这个应用程序的每一页中添加一个Page指示符去处理ASP.NET 代码错误。

F. 在这个应用程序的每一页中添加一个Page指示符去处理ASP.NET HTTP错误。

195. 您为AllWin公司创建了一个ASP.NET应用程序。这个应用程序调用一个Xml Web Service。这个Xml Web Service将返回一个包含了公司雇员列表的DataSet对象。请问您该如何在这个程序中使用这个Xml Web Service？(b)

A.在“引用”对话框的.Net标签中选择System.Web.Services.dll。

B.在“Web引用”对话框中输入这个XML Web service的地址。

C.在您的Global.asax.cs中添加一条using语句并指定这个XML Web service的地址。

D.在您的Global.asax.cs中写一个事件处理器导入这个Xml Web Service相应的.wsdl和.disco文件。

C sharp面试模拟题基于MSDN的回答

1、请解释在new 与override 的区别？

new 运算符用于在堆上创建对象和调用构造函数。

new 运算符用于创建对象和调用构造函数。

new 修饰符　用于隐藏基类成员的继承成员。

使用 new 修饰符显式隐藏从基类继承的成员。若要隐藏继承的成员，请使用相同名称在派生类中声明该成员，并用 new 修饰符修饰它。

通过继承隐藏名称采用下列形式之一：引入类或结构中的常数、指定、属性或类型隐藏具有相同名称的所有基类成员。 引入类或结构中的方法隐藏基类中具有相同名称的属性、字段和类型。同时也隐藏具有相同签名的所有基类方法。引入类或结构中的索引器将隐藏具有相同名称的所有基类索引器。
在同一成员上同时使用 new 和 override 是错误的。

在不隐藏继承成员的声明中使用 new 修饰符将生成警告。

使用 override 修饰符来修改方法、属性、索引器或事件。重写方法提供从基类继承的成员的新实现。由重写声明重写的方法称为重写基方法。重写基方法必须与重写方法具有相同的签名。

不能重写非虚方法或静态方法。重写基方法必须是虚拟的、抽象的或重写的。重写声明不能更改虚方法的可访问性。重写方法和虚方法必须具有相同的访问级修饰符。
不能使用下列修饰符修改重写方法：new　static　virtual　abstract重写属性声明必须指定与继承属性完全相同的访问修饰符、类型和名称，并且重写属性必须是虚拟的、抽象的或重写的。

2．请解释virtual的含义？

virtual 关键字用于修改方法或属性的声明，在这种情况下，方法或属性被称作虚拟成员。虚拟成员的实现可由派生类中的重写成员更改。调用虚方法时，将为重写成员检查该对象的运行时类型。将调用大部分派生类中的该重写成员，如果没有派生类重写该成员，则它可能是原始成员。默认情况下，方法是非虚拟的。不能重写非虚方法。不能将 virtual 修饰符与以下修饰符一起使用：static　abstract　override除了声明和调用语法不同外，虚拟属性的行为与抽象方法一样。在静态属性上使用 virtual 修饰符是错误的。通过包括使用 override 修饰符的属性声明，可在派生类中重写虚拟继承属性。

3．请解释.net采用委托实现的事件模型与JAVA中采用接口实现的事件模型有什么区别，以图示方式解释。

事件是对象发送的消息，以发信号通知操作的发生。操作可能是由用户交互（例如鼠标单击）引起的，也可能是由某些其他的程序逻辑触发的。引发（触发）事件的对象叫做事件发送方。捕获事件并对其作出响应的对象叫做事件接收方。
在事件通信中，事件发送方类不知道哪个对象或方法将接收到（处理）它引发的事件。所需要的是在源和接收方之间存在一个媒介（或类似指针的机制）。.NET Framework 定义了一个特殊的类型（Delegate），该类型提供函数指针的功能。

委托是一个可以对方法进行引用的类。与其他的类不同，委托类具有一个签名，并且它只能对与其签名匹配的方法进行引用。这样，委托就等效于一个类型安全函数指针或一个回调。这里只讨论委托的事件处理功能。
观察者模式（Observer）是对象的行为模式，又称：

发布-订阅模（ublish/Subscribe）源-监听器 Source/Listener)

•观察者模式定义了一个一对多的依赖关系，让一个或多个观察者对象监察一个主题对象。

•这样一个主题对象在状态上的变化能够通知所有的依赖于此对象的那些观察者对象，使这些观察者对象能够自动更新

4．请解释接口的显式实现有什么意义？

接口可以有静态成员、嵌套类型、抽象、虚拟成员、属性和事件。实现接口的任何类都必须提供接口中所声明的抽象成员的定义。接口可以要求任何实现类必须实现一个或多个其他接口。对接口有以下限制：接口可以用任何可访问性来声明，但接口成员必须全都具有公共可访问性。不能向成员或接口自身附加安全性限制。接口可以定义类构造函数，但不能定义实例构造函数。每种语言都必须为需要成员的接口映射一个实现提供规则，因为不只一个接口可以用相同的签名声明成员，且这些成员可以有单独的实现。为了实现接口，类或结构可以声明“显式接口成员实现”。显式接口成员实现就是一种方法、属性、事件或索引器声明，它使用完全限定接口成员名称作为标识符。
某些情况下，接口成员的名称对于实现该接口的类可能是不适当的，此时，可以使用显式接口成员实现来实现该接口成员。例如，一个用于“文件抽象”的类一般会实现一个具有释放文件资源作用的 `Close` 成员函数，同时还可能使用显式接口成员实现来实现接口的方法

在方法调用、属性访问或索引器访问中，不能直接访问“显式接口成员实现”的成员，即使用它的完全限定名也不行。“显式接口成员实现”的成员只能通过接口实例访问，并且在通过接口实例访问时，只能用该接口成员的简单名称来引用。
显式接口成员实现中包含访问修饰符属于编译时错误，而且如果包含 `abstract`、`virtual`、`override` 或 `static` 修饰符也属于编译时错误。

显式接口成员实现具有与其他成员不同的可访问性特征。由于显式接口成员实现永远不能在方法调用或属性访问中通过它们的完全限定名来访问，因此，它们似乎是 private（私有的）。但是，因为它们可以通过接口实例来访问，所以它们似乎又是 public（公共的）。

显式接口成员实现有两个主要用途：由于显式接口成员实现不能通过类或结构实例来访问，因此它们就不属于类或结构的自身的公共接口。当需在一个公用的类或结构中实现一些仅供内部使用（不允许外界访问）的接口时，这就特别有用。显式接口成员实现可以消除因同时含有多个相同签名的接口成员所引起的多义性。如果没有显式接口成员实现，一个类或结构就不可能为具有相同签名和返回类型的接口成员分别提供相应的实现，也不可能为具有相同签名和不同返回类型的所有接口成员中的任何一个提供实现。为了使显式接口成员实现有效，声明它的类或结构必须在它的基类列表中指定一个接口，而该接口必须包含一个成员，该成员的完全限定名、类型和参数类型与该显式接口成员实现所具有的完全相同。
请以图示方式解释性 .net framework?

.NET Framework 是一种新的计算平台，它简化了在高度分布式 Internet 环境中的应用程序开发。.NET Framework 旨在实现下列目标：

- 提供一个一致的面向对象的编程环境，而无论对象代码是在本地存储和执行，还是在本地执行但在 Internet 上分布，或者是在远程执行的。
- 提供一个将软件部署和版本控制冲突最小化的代码执行环境。
- 提供一个保证代码（包括由未知的或不完全受信任的第三方创建的代码）安全执行的代码执行环境。
- 提供一个可消除脚本环境或解释环境的性能问题的代码执行环境。
- 使开发人员的经验在面对类型大不相同的应用程序（如基于 Windows 的应用程序和基于 Web 的应用程序）时保持一致。
- 按照工业标准生成所有通信，以确保基于 .NET Framework 的代码可与任何其他代码集成。

.NET Framework 具有两个主要组件：公共语言运行库和 .NET Framework 类库。公共语言运行库是 .NET Framework 的基础。您可以将运行库看作一个在执行时管理代码的代理，它提供核心服务（如内存管理、线程管理和远程处理），而且还强制实施严格的类型安全以及可确保安全性和可靠性的其他形式的代码准确性。事实

上，代码管理的概念是运行库的基本原则。以运行库为目标的代码称为托管代码，而不以运行库为目标的代码称为非托管代码。.NET Framework 的另一个主要组件是类库，它是一个综合性的面向对象的可重用类型集合，您可以使用它开发多种应用程序，这些应用程序包括传统的命令行或图形用户界面 (GUI) 应用程序，也包括基于 ASP.NET 所提供的最新创新的应用程序（如 Web 窗体和 XML Web services）。

.NET Framework 可由非托管组件承载，这些组件将公共语言运行库加载到它们的进程中并启动托管代码的执行，从而创建一个可以同时利用托管和非托管功能的软件环境。.NET Framework 不但提供若干个运行库宿主，而且还支持第三方运行库宿主的开发。

例如，ASP.NET 承载运行库以为托管代码提供可伸缩的服务器端环境。ASP.NET 直接使用运行库以启用 ASP.NET 应用程序和 XML Web services（本主题稍后将对这两者进行讨论）。

Internet Explorer 是承载运行库（以 MIME 类型扩展的形式）的非托管应用程序的一个示例。使用 Internet Explorer 承载运行库使您能够在 HTML 文档中嵌入托管组件或 Windows 窗体控件。以这种方式承载运行库使得托管移动代码（类似于 Microsoft® ActiveX® 控件）成为可能，但是它具有只有托管代码才能提供的重大改进（如不完全受信任的执行和安全的独立文件存储）。

下面的插图显示公共语言运行库和类库与应用程序之间以及与整个系统之间的关系。该插图还显示托管代码如何在更大的结构内运行。
.NET Framework 环境

下面的章节将更加详细地描述 .NET Framework 的主要组件和功能。

公共语言运行库的功能公共语言运行库管理内存、线程执行、代码执行、代码安全验证、编译以及其他系统服务。这些功能是在公共语言运行库上运行的托管代码所固有的。
至于安全性，取决于包括托管组件的来源（如 Internet、企业网络或本地计算机）在内的一些因素，托管组件被赋予不同程度的信任。这意味着即使用在同一活动应用程序中，托管组件既可能能够执行文件访问操作、注册表访问操作或其他须小心使用的功能，也可能不能够执行这些功能。

运行库强制实施代码访问安全。例如，用户可以相信嵌入在 Web 页中的可执行文件能够在屏幕上播放动画或唱歌，但不能访问他们的个人数据、文件系统或网络。这样，运行库的安全性功能就使通过 Internet 部署的合法软件能够具有特别丰富的功能。

运行库还通过实现称为通用类型系统 (CTS) 的严格类型验证和代码验证基础结构来加强代码可靠性。CTS 确保所有托管代码都是可以自我描述的。各种 Microsoft 和第三方语言编译器生成符合 CTS 的托管代码。这意味着托管代码可在严格实施类型保真和类型安全的同时使用其他托管类型和实例。

此外，运行库的托管环境还消除了许多常见的软件问题。例如，运行库自动处理对象布局并管理对对象的引用，在不再使用它们时将它们释放。这种自动内存管理解决了两个最常见的应用程序错误：内存泄漏和无效内存引用。
运行库还提高了开发人员的工作效率。例如，程序员可以用他们选择的开发语言编写应用程序，却仍能充分利用其他开发人员用其他语言编写的运行库、类库和组件。任何选择以运行库为目标的编译器供应商都可以这样做。以 .NET Framework 为目标的语言编译器使得用该语言编写的现有代码可以使用 .NET Framework 的功能，这大大减轻了现有应用程序的迁移过程的工作负担。

尽管运行库是为未来的软件设计的，但是它也支持现在和以前的软件。托管和非托管代码之间的互操作性使开发人员能够继续使用所需的 COM 组件和 DLL。

运行库旨在增强性能。尽管公共语言运行库提供许多标准运行库服务，但是它从不解释托管代码。一种称为实时 (JIT) 编译的功能使所有托管代码能够以它在其上执行的系统的本机语言运行。同时，内存管理器排除了出现零碎内存的可能性，并增大了内存引用区域以进一步提高性能。

最后，运行库可由高性能的服务器端应用程序（如 Microsoft® SQL Server™ 和 Internet 信息服务 (IIS)）承载。此基础结构使您在享受支持运行库宿主的行业最佳企业服务器的优越性能的同时，能够使用托管代码编写业务逻辑。

.NET Framework 类库

.NET Framework 类库是一个与公共语言运行库紧密集成的可重用的类型集合。该类库是面向对象的，并提供您自己的托管代码可从中导出功能的类型。这不但使 .NET Framework 类型易于使用，而且还减少了学习 .NET Framework 的新功能所需要的时间。此外，第三方组件可与 .NET Framework 中的类无缝集成。

例如，.NET Framework 集合类实现一组可用于开发您自己的集合类的接口。您的集合类将与 .NET Framework 中的类无缝地混合。

.NET Framework 类型使您能够完成一系列常见编程任务（包括诸如字符串管理、数据收集、数据库连接以及文件访问等任务）。除这些常见任务之外，类库还包括支持多种专用开发方案的类型。例如，可使用 .NET Framework 开发下列类型的应用程序和服务：

- 控制台应用程序。
- Windows GUI 应用程序（Windows 窗体）。
- ASP.NET 应用程序。
- XML Web services。
- Windows 服务。

例如，Windows 窗体类是一组综合性的可重用的类型，它们大大简化了 Windows GUI 的开发。如果要编写 ASP.NET Web 窗体应用程序，可使用 Web 窗体类。

客户端应用程序开发
客户端应用程序在基于 Windows 的编程中最接近于传统风格的应用程序。这些是在桌面上显示窗口或窗体从而使用户能够执行任务的应用程序类型。客户端应用程序包括诸如字处理程序和电子表格等应用程序，还包括自定义的业务应用程序（如数据输入工具、报告工具等等）。客户端应用程序通常使用窗口、菜单、按钮和其他 GUI 元素，并且它们可能访问本地资源（如文件系统）和外围设备（如打印机）。

另一种客户端应用程序是作为 Web 页通过 Internet 部署的传统 ActiveX 控件（现在被托管 Windows 窗体控件所替代）。此应用程序非常类似于其他客户端应用程序：它在本机执行，可以访问本地资源，并包含图形元素。

过去，开发人员将 C/C++ 与 Microsoft 基础类 (MFC) 或应用程序快速开发 (RAD) 环境（如 Microsoft® Visual Basic®）一起使用来创建这样的应用程序。.NET Framework 将这些现有产品的特点合并到了单个且一致的开发环境中，该环境大大简化了客户端应用程序的开发。

包含在 .NET Framework 中的 Windows 窗体类旨在用于 GUI 开发。您可以轻松创建具有适应多变的商业需求所需的灵活性的命令窗口、按钮、菜单、工具栏和其他屏幕元素。

例如，.NET Framework 提供简单的属性以调整与窗体相关联的可视属性。某些情况下，基础操作系统不支持直

接更改这些属性，而在这些情况下，.NET Framework 将自动重新创建窗体。这是 .NET Framework 集成开发人员接口从而使编码更简单更一致的许多方法之一。

和 ActiveX 控件不同，Windows 窗体控件具有对用户计算机的不完全受信任的访问权限。这意味着二进制代码或在本机执行的代码可访问用户系统上的某些资源，例如 GUI 元素和访问受限制的文件，但这些代码不能访问或危害其他资源。由于具有代码访问安全性，许多曾经需要安装在用户系统上的应用程序现在可以通过 Web 安全地部署。您的应用程序可以在像 Web 页那样部署时实现本地应用程序的功能。

服务器应用程序开发在托管领域中，服务器端应用程序是通过运行库宿主实现的。非托管应用程序承载公共语言运行库，后者使您的自定义托管代码可以控制服务器的行为。此模型在获得主服务器的性能和可伸缩性的同时提供给您公共语言运行库和类库的所有功能。
下面的插图显示在不同服务器环境中运行托管代码的基本网络架构。在应用程序逻辑通过托管代码执行时，服务器（如 IIS 和 SQL Server）可执行标准操作。

服务器端托管代码
ASP.NET 是使开发人员能够使用 .NET Framework 开发基于 Web 的应用程序的宿主环境。但是，ASP.NET 不止是一个运行库宿主；它是使用托管代码开发 Web 站点和通过 Internet 分布的对象的完整结构。Web 窗体和 XML Web services 都将 IIS 和 ASP.NET 用作应用程序的发布机制，并且两者在 .NET Framework 中都具有支持类集合。

XML Web services 作为基于 Web 的技术的重要发展，是类似于常见 Web 站点的分布式服务器端应用程序组件。但是，与基于 Web 的应用程序不同，XML Web services 组件不具有 UI 并且不以浏览器（如 Internet Explorer 和 Netscape Navigator）为目标。XML Web services 由旨在供其他应用程序使用的可重用的软件组件组成，所谓的其他应用程序包括：传统的客户端应用程序，基于 Web 的应用程序，甚至是其他 XML Web services。因此，XML Web services 技术正迅速地将应用程序开发和部署推向高度分布式 Internet 环境。

如果您使用过 ASP 技术的早期版本，很快就会注意到 ASP.NET 和 Web 窗体提供的改进。例如，您可以用支持 .NET Framework 的任何语言开发 Web 窗体页。此外，您的代码不再需要与 HTTP 文本共享同一个文件（尽管如果您愿意，代码还可以继续这样做）。Web 窗体页用本机语言执行，这是因为与所有其他托管应用程序一样，它们充分利用运行库。与此相对照，非托管 ASP 页始终被写成脚本并解释。ASP.NET 页比非托管 ASP 页更快、更实用并且更易于开发，这是因为它们像所有托管应用程序一样与运行库进行交互。

.NET Framework 还提供类和工具的集合来帮助开发和使用 XML Web services 应用程序。XML Web services 是基于 SOAP（一种远程过程调用协议）、XML（一种可扩展的数据格式）和 WSDL（Web 服务描述语言）这些标准生成的。基于这些标准生成 .NET Framework 的目的是为了提高与非 Microsoft 解决方案的互操作性。

例如，.NET Framework SDK 所包含的 Web 服务描述语言工具可以查询在 Web 上发布的 XML Web services，分析它的 WSDL 描述，并产生 C# 或 Visual Basic 源代码，您的应用程序可以使用这些代码而成为 XML Web services 的客户端。这些源代码可以创建从类库中的类派生的类，这些类使用 SOAP 和 XML 分析处理所有基础通信。虽然您可以使用类库来直接使用 XML Web services，Web 服务描述语言工具和包含在 SDK 中的其他工具可以使您更加方便地用 .NET Framework 进行开发。

如果您开发和发布自己的 XML Web services，.NET Framework 为您提供了一组符合所有基础通信标准（如 SOAP、WSDL 和 XML）的类。使用这些类使您能够将注意力集中在服务的逻辑上，而无需关注分布式软件开发所需要的通信基础结构。

最后，与托管环境中的 Web 窗体页相似，您的 XML Web services 将使用 IIS 的可伸缩通信以本机语言的速度运行。

什么是.net?

Visual Studio .NET 是一套完整的开发工具，用于生成 ASP Web 应用程序、XML Web services、桌面应用程序和移动应用程序。Visual Basic .NET、Visual C++ .NET、Visual C# .NET 和 Visual J# .NET 全都使用相同的集成开发环境 (IDE)，该环境允许它们共享工具并有助于创建混合语言解决方案。另外，这些语言利用了 .NET Framework 的功能，此框架提供对简化 ASP Web 应用程序和 XML Web services 开发的关键技术的访问。

7、Ref 与out有什么不同?

方法参数上的 ref方法参数关键字使方法引用传递到方法的同一个变量。当控制传递回调用方法时，在方法中对参数所做的任何更改都将反映在该变量中。若要使用 ref 参数，必须将参数作为 ref 参数显式传递到方法。ref 参数的值被传递到 ref 参数。传递到 ref 参数的参数必须最先初始化。将此方法与 out 参数相比，后者的参数在传递到 out 参数之前不必显式初始化。属性不是变量，不能作为 ref 参数传递。如果两种方法的声明仅在它们对 ref 的使用方面不同，则将出现重载。但是，无法定义仅在 ref 和 out 方面不同的重载

方法参数上的 out 方法参数关键字使方法引用传递到方法的同一个变量。当控制传递回调用方法时，在方法中对参数所做的任何更改都将反映在该变量中。当希望方法返回多个值时，声明 out 方法非常有用。使用 out 参数的方法仍然可以返回一个值。一个方法可以有一个以上的 out 参数。若要使用 out 参数，必须将参数作为 out 参数显式传递到方法。out 参数的值不会传递到 out 参数。不必初始化作为 out 参数传递的变量。然而，必须在方法返回之前为 out 参数赋值。属性不是变量，不能作为 out 参数传递。如果两个方法的声明仅在 out 的使用方面不同，则会发生重载。不过，无法定义仅在 ref 和 out 方面不同的重载。

params 关键字可以指定在参数数目可变处采用参数的方法参数。在方法声明中的 params 关键字之后不允许任何其他参数，并且在方法声明中只允许一个 params 关键字。

8. 值类型与引用类型有什么不同？请举例说明？并分别列举几种相应的数据类型。

大多数编程语言提供内置的数据类型（比如整数和浮点数），这些数据类型会在作为参数传递时被复制（即，它们通过值来传递）。在 .NET Framework 中，这些称为值类型。运行库支持两种值类型：

内置值类型 ：.NET Framework 定义了内置值类型（如 System.Int32 和 System.Boolean），它们对应于编程语言使用的基元数据类型并与之相同。

用户定义的值类型 ：您的语言将提供各种方法来定义派生自 System.ValueType 的您自己的值类型。如果您想定义一个表示小值的类型，比如复数（使用两个浮点数），则可以选择将其定义为值类型，因为您可以有效地通过值来传递值类型。如果您要定义的类型通过引用传递时会更高效，则应将其定义为类。

值类型与基元类型有着同样的存储效率，然而您可以对它们调用方法，包括对 System.Object 和 System.ValueType 类定义的虚方法，以及对值类型本身定义的任何方法。您可以创建值类型的实例，将它们作为参数传递，将它们存储为局部变量，或将它们存储在另一值类型或对象的字段中。值类型没有与存储类的实例相关的系统开销，并且它们不需要构造函数。

对于每一种值类型，运行库都提供一种相应的已装箱类型，这是与值类型有着相同状态和行为的类。当需要已装箱的类型时，某些语言要求使用特殊的语法；而另外一些语言会自动使用已装箱的类型。在定义值类型时，需要同时定义已装箱和未装箱的类型。

值类型可以有字段、属性和事件。它们也有静态和非静态方法。当它们被装箱时，会从 System.ValueType 继承虚方法，并可实现零个或更多接口。

值类型是密封的，这意味着不能从它们派生出其他类型。但是，可以直接对值类型定义虚方法，并且既可对该类型的已装箱形式，也可对未装箱形式调用这些方法。尽管不能从一种值类型派生出另一种类型，但是当所用语言处理虚方法比处理非虚方法或静态方法更方便时，可以对值类型定义虚方法。
引用类型 (reference type)

由类型的实际值引用（类似于指针）表示的数据类型。如果为某个变量分配一个引用类型，则该变量将引用（或“指向”）原始值。不创建任何副本。引用类型包括类、接口、委托和装箱值类型。引用类型值是对该类型的某个实例的一个引用，后者称为对象。`null` 值比较特别，它适用于所有引用类型，用来表示“没有被引用的实例”。

C#中引用类型的声明：类使用 class 关键字声明。接口使用interface声明。delegate 声明定义一种引用类型，该类型可用于将方法用特定的签名封装。委托实例封装静态方法或实例方法。委托大致类似于 C++ 中的函数指针；但是，委托是类型安全和可靠的。

C#中内置引用类型：

object 类型在 .NET Framework 中是 System.Object 的别名。可将任何类型的值赋给 object 类型的变量。所有数据类型无论是预定义的还是用户定义的，均从 System.Object 类继承。object 数据类型是同对象进行相互已装箱的类型。

string 类型表示一个 Unicode 字符的字符串。string 是 .NET Framework 中 System.String 的别名。尽管 string 是引用类型，但相等运算符（== 和 !=）被定义为比较 string 对象（而不是引用）的“值”（7.9.7 字符串相等运算符）。这使得对字符串相等性的测试更为直观。

C# 支持两种类型：“值类型”和“引用类型”的比较：

值类型包括简单类型（如 `char`、`int` 和 `float`）、枚举类型和结构类型。引用类型包括类 (Class) 类型、接口类型、委托类型和数组类型。

值类型与引用类型的区别在于值类型的变量直接包含其数据，而引用类型的变量则存储对象引用。对于引用类型，两个变量可能引用同一对象，因此对一个变量的操作可能影响另一个变量所引用的对象。对于值类型，每个变量都有自己的数据副本，对一个变量的操作不可能影响另一个变量。
数据类型分隔为值类型和引用类型。值类型要么是堆栈分配的，要么是在结构中以内联方式分配的。引用类型是堆分配的。引用类型和值类型都是从最终的基类 Object 派生出来的。当值类型需要充当对象时，就在堆上分配一个包装（该包装能使值类型看上去像引用对象一样），并且将该值类型的值复制给它。该包装被加上标记，以便系统知道它包含一个值类型。这个进程称为装箱，其反向进程称为取消装箱。装箱和取消装箱能够使任何类型像对象一样进行处理。

9．结构体是值类型还是引用类型的？

结构与类很相似，都表示可以包含数据成员和函数成员的数据结构。但是，与类不同，结构是一种值类型，并且不需要堆分配。结构类型的变量直接包含结构的数据，而类类型的变量包含对数据的引用（后者称为对象）。

结构对于具有值语义的小的数据结构特别有用。复数、坐标系中的点或字典中的“键-值”对都是结构的典型示例。这些数据结构的关键之处在于：它们只有少量数据成员，它们不要求使用继承或引用标识，而且它们适合使

用值语义（赋值时直接复制值而不是复制它的引用）方便地实现。

struct 类型是一种可包含构造函数、常数、字段、方法、属性、索引器、运算符、事件和嵌套类型的值类型

备注
在类上调用“新建”(New) 运算符时，它将在堆上进行分配。但是，当实例化结构时，将在堆栈上创建结构。这样将产生性能增益。而且，您不会像对待类那样处理对结构实例的引用。您将直接对结构实例进行操作。鉴于此原因，向方法传递结构时，结构将通过值传递，而不是作为引用传递。

结构可以声明构造函数，但它们必须带参数。声明结构的默认（无参数）构造函数是错误的。结构成员不能有初始值设定项。总是提供默认构造函数以将结构成员初始化为它们的默认值。使用 New 运算符创建结构对象时，将创建该结构对象，并且调用适当的构造函数。与类不同的是，结构的实例化可以不使用 New 运算符。如果不使用“新建”(new)，那么在初始化所有字段之前，字段将保持未赋值状态，且对象不可用。对于结构，不像类那样存在继承。一个结构不能从另一个结构或类继承，而且不能作为一个类的基。但是，结构从基类对象继承。结构可实现接口，而且实现方式与类实现接口的方式完全相同。通过使用属性可以自定义结构在内存中的布局方式。结构使用简单，并且有时证明很有用。但要牢记：结构在堆栈中创建，并且您不是处理对结构的引用，而是直接处理结构。每当需要一种将经常使用的类型，而且大多数情况下该类型只是一些数据时，结构可能是最佳选择。

struct 类型适合表示如点、矩形和颜色这样的轻量对象。尽管可能将一个点表示为类，但结构在某些方案中更有效。例如，如果声明一个含有 1000 个点对象的数组，则将为引用每个对象分配附加的内存。在此情况下，结构的成本较低。

声明结构的默认（无参数）构造函数是错误的。总是提供默认构造函数以将结构成员初始化为它们的默认值。在结构中初始化实例字段是错误的。
使用 new 运算符创建结构对象时，将创建该结构对象，并且调用适当的构造函数。与类不同的是，结构的实例化可以不使用 new 运算符。如果不使用 new，那么在初始化所有字段之前，字段将保持未赋值状态且对象不可用。

对于结构，不像类那样存在继承。一个结构不能从另一个结构或类继承，而且不能作为一个类的基。但是，结构从基类 Object 继承。结构可实现接口，其方式同类完全一样。

与 C++ 不同，无法使用 struct 关键字声明类。在 C# 中，类与结构在语义上是不同的。struct（结构）是值类型，而 class（类）是引用类型。除非需要引用类型语义，否则，小于 16 字节的类被系统作为结构处理可能更高效。

结构和类的比较：

Visual Basic .NET 统一了结构和类的语法，结果就是两个实体都支持大多数的相同功能。但是，在结构和类之间还有着重要的区别。

相同点结构和类在以下方面相同：

- 两者都属于“容器”类型，表示它们可以包含其他类型作为成员。
- 两者都具有成员，成员可以包括构造函数、方法、属性、字段、常数、枚举、事件和事件处理程序。
- 两者的成员都具有单独的可访问性。例如，一个成员可以声明为 Public，而另一个可以声明为 Private。
- 都可实现接口。
- 都有共享的构造函数，有或没有参数。
- 两者都可以公开默认属性，只要该属性至少带有一个参数。

- 两者都可以声明和引发事件，而且两者都可以声明委托。

不同点结构和类在以下方面有所不同：
- 结构是值类型，而类是引用类型。
- 结构使用堆栈分配，类使用堆分配。
- 所有的结构成员都默认为 Public；类变量和常量默认为 Private，而其他的类成员默认为 Public。类成员的这一行为提供与 Visual Basic 6.0 默认值系统的兼容。
- 结构必须至少具有一个非共享变量或事件成员；而类可以完全是空的。
- 结构成员不能声明为 Protected，类成员可以。
- 只有 Shared Sub 结构过程才能处理事件，并且只能使用 AddHandler 语句；而任何类过程都可以处理事件，并且可以使用 Handles 关键字或 AddHandler 语句。
- 结构变量声明不能指定初始值、New 关键字或数组初始大小，类变量声明可以。
- 结构从 ValueType 类隐式继承，不能从其他类型继承，类可以从除 ValueType 之外的其他任何类继承。
- 结构是不可继承的；而类可以继承。
- 结构从不终止，所以公共语言运行库 (CLR) 从不在任何结构上调用 Finalize 方法，类可由垃圾回收器终止，当检测到没有剩下的活动引用时，垃圾回收器将在类上调用 Finalize。
- 结构不需要构造函数；而类需要。
- 结构仅当没有参数时可以有非共享的构造函数；类无论有没有参数都可以。

每一个结构都有不带参数的隐式公共构造函数。此构造函数将结构的所有数据成员初始化为默认值。不能重定义此行为。实例和变量由于结构是值类型，每个结构变量都永久地绑定到一个单独的结构实例。但类是引用类型，对象变量可引用各种类实例。此区别在下列方面影响结构和类的使用：
- 结构变量使用结构的无参数构造函数隐式包含成员初始化。因此，`Dim S As Struct1` 等效于 `Dim S As Struct1 = New Struct1()`。
- 当将一个结构变量赋给另一个，或传递一个结构实例到过程参数，所有变量成员的当前值都被复制到新结构中。当将一个对象变量赋给另一个，或传递一个对象变量到过程，仅有引用指针被复制。
- 可以将值 Nothing 赋给结构变量，但实例继续保持与变量的关联。尽管赋值重新初始化了变量成员，仍可以调用其方法并访问其数据成员。相比之下，如果将对象变量设为 Nothing，将其与任何类实例断开关联，在给它赋予另一个实例前，不能通过变量访问其他成员。
- 对象变量可以有在不同时间赋给它的不同的类实例，几个对象变量可以同时引用同一个类实例。当通过指向同一实例的另一个变量访问时，更改的类成员的值会影响这些成员。但是，结构成员独立存在于其自身实例中。更改其值不会在其他任何结构变量中反映出来，即使是在同一 Structure 声明的其他实例中。
- 两个结构的等效性测试必须在成员对成员的测试中进行。两个对象变量可使用 Equals 方法进行比较。Equals 指示两个变量是否指向同一实例。

10、C#中有没有静态构造函数，如果有是做什么用的？

静态构造函数用于初始化类。在创建第一个实例或引用任何静态成员之前，将自动调用静态构造函数来初始化类。静态构造函数是一种用于实现初始化类所需操作的成员。静态构造函数是使用静态构造函数声明来声明的。

静态构造函数声明可包含一组属性和一个 `extern`修饰符。静态构造函数声明的标识符必须是声明了该静态函数的那个类的名称。如果指定了任何其他名称，则发生编译时错误。当静态构造函数声明包含 `extern` 修饰符时，称该静态构造函数为外部静态构造函数。因为外部静态构造函数声明不提供任何实际的实现，所以它的静态构造函数体由一个分号组成。对于所有其他的静态构造函数声明，静态构造函数体都是一个块，它指定当初始化该类时需要执行的语句。这正好相当于具有 `void` 返回类型的静态方法的方法体。

静态构造函数是不可继承的，而且不能被直接调用。类的静态构造函数在给定应用程序域中至多执行一次。应用程序域中第一次发生以下事件时将触发静态构造函数的执行：

- 创建类的实例。
- 引用类的任何静态成员。

如果类中包含用来开始执行的 `Main` 方法，则该类的静态构造函数将在调用 `Main` 方法之前执行。如果类包含任何带有初始值设定项的静态字段，则在执行该类的静态构造函数时，先要按照文本顺序执行那些初始值设定项。

11、在C#中如何实现多态？

多态性是类为方法（这些方法以相同的名称调用）提供不同实现方式的能力。多态性允许对类的某个方法进行调用而无需考虑该方法所提供的特定实现。例如，可能有名为 Road 的类，它调用另一个类的 Drive 方法。这另一个类 Car 可能是 SportsCar 或 SmallCar，但二者都提供 Drive 方法。虽然 Drive 方法的实现因类的不同而异，但 Road 类仍可以调用它，并且它提供的结果可由 Road 类使用和解释。

可以用不同的方式实现组件中的多态性：

- 接口多态性 - 多个类可实现相同的“接口”，而单个类可以实现一个或多个接口。接口本质上是类需要如何响应的定义。接口描述类需要实现的方法、属性和事件，以及每个成员需要接收和返回的参数类型，但将这些成员的特定实现留给实现类去完成。
- 继承多态性 - 多个类可以从单个基类“继承”。通过继承，类在基类所在的同一实现中接收基类的所有方法、属性和事件。这样，便可根据需要来实现附加成员，而且可以重写基成员以提供不同的实现。请注意，继承类也可以实现接口，这两种技术不是互相排斥的。
- 通过抽象类实现的多态性 - 抽象类同时提供继承和接口的元素。抽象类本身不能实例化，它必须被继承。该类的部分或全部成员可能未实现，该实现由继承类提供。已实现的成员仍可被重写，并且继承类仍可以实现附加接口或其他功能。

接口和抽象类使您可以创建组件交互的定义。通过接口，可以指定组件必须实现的方法，但不实际指定如何实现方法。抽象类使您可以创建行为的定义，同时提供用于继承类的一些公共实现。对于在组件中实现多态行为，接口和抽象类都是很有用的工具。下面将更详细地讨论这三种类型的多态性。接口多态性：组件编程中的一项强大技术是能够在一个对象上实现多个接口。每个接口由一小部分紧密联系的方法、属性和事件组成。通过实现接口，组件可以为要求该接口的任何其他组件提供功能，而无需考虑其中所包含的特定功能。这使后续组件的版本得以包含不同的功能而不会干扰核心功能。其他开发人员最常使用的组件功能自然是组件类本身的成员。然而，包含大量成员的组件使用起来可能比较困难。可以考虑将组件的某些功能分解出来，作为私下实现的单独接口。根据接口来定义功能的另一个好处是，可以通过定义和实现附加接口增量地将功能添加到组件中。优点包括：

- 简化了设计过程，因为组件开始时可以很小，具有最小功能；之后，组件继续提供最小功能，同时不断插入其他的功能，并通过实际使用那些功能来确定合适的功能。
- 简化了兼容性的维护，因为组件的新版本可以在添加新接口的同时继续提供现有接口。客户端应用程序的后续版本可以利用这些接口的优点（如果这样做有意义）。

通过继承实现的多态性：

Visual Basic 和 C# 也通过继承提供多态性。对于小规模开发任务而言，这是一个功能强大的机制，但对于大规模系统，通常证明会存在问题。过分强调继承驱动的多态性一般会导致资源大规模地从编码转移到设计，这对于缩短总的开发时间没有任何帮助。如果对软件真正的测试是看其是否适用于最终用户，则与面向对象编程的工具相比，快速原型法和快速应用程序开发 (RAD) 的工具受到了更广泛地欢迎。

何时使用继承驱动的多态性使用继承首先是为了向现有基类添加功能。若从经过完全调试的基类框架开始，则程

序员的工作效率将大大提高，方法可以增量地添加到基类而不中断版本。当应用程序设计包含多个相关类，而对于某些通用函数，这些相关类必须共享同样的实现时，您也可能希望使用继承。重叠功能可以在基类中实现，应用程序中使用的类可以从该基类中派生。抽象类合并继承和实现的功能，这在需要二者之一的元素时可能很有用。通过抽象类实现的多态性：
抽象类提供继承和接口实现的功能。抽象（在 Visual Basic 中是 MustInherit）类不能示例化，必须在继承类中实现。它可以包含已实现的方法和属性，但也可以包含未实现的过程，这些未实现过程必须在继承类中实现。这使您得以在类的某些方法中提供不变级功能，同时为其他过程保持灵活性选项打开。抽象类的另一个好处是：当要求组件的新版本时，可根据需要将附加方法添加到基类，但接口必须保持不变。

何时使用抽象类：当需要一组相关组件来包含一组具有相同功能的方法，但同时要求在其他方法实现中具有灵活性时，可以使用抽象类。当预料可能出现版本问题时，抽象类也具有价值，因为基类比较灵活并易于被修改。
12、什么是反射？如何实现反射？

通过反射命名空间中的类以及 System.Type，您可以获取有关已加载的程序集和在其中定义的类型（如类、接口和值类型）的信息。您也可以使用反射在运行时创建类型实例，然后调用和访问这些实例。

System.Reflection 类中最常用的方法都使用统一的模式。Module、Type 和 MemberInfo 类的成员使用下表中所示的设计模式。System.Type 类对于反射起着核心的作用。当反射请求加载的类型时，公共语言运行库将为它创建一个 Type。您可以使用 Type 对象的方法、字段、属性和嵌套类来查找有关该类型的所有信息。在使用 Assembly.GetType 或 Assembly.GetTypes 时传入所需类型的名称，可以从尚未加载的程序集中获取 Type 对象。使用 Type.GetType 可从已加载的程序集中获取 Type 对象。使用 Module.GetType 和 Module.GetTypes 可获取模块 Type 对象。另一个常用的设计模式是使用委托。它们通常在反射中用来支持对返回对象数组的方法的结果集进行筛选。

反射提供了由语言编译器（例如 Microsoft Visual Basic .NET 和 JScript）用来实现隐式晚期绑定的基础结构。绑定是查找与唯一指定的类型相对应的声明（即实现）的过程。由于此过程在运行时而不是在编译时发生，所以称作晚期绑定。Visual Basic .NET 允许您在代码中使用隐式的晚期绑定；Visual Basic 编译器将调用一个帮助器方法，该方法使用反射来获取对象类型。传递给帮助器方法的参数有助于在运行时调用正确的方法。这些参数包括：对其调用方法的实例（对象），被调用方法的名称（字符串），以及传递给被调用方法的参数（对象数组）。

13. 请解释流与文件有什么不同？

抽象基类 Stream 支持读取和写入字节。Stream 集成了异步支持。其默认实现根据其相应的异步方法来定义同步读取和写入，反之亦然。所有表示流的类都是从 Stream 类继承的。Stream 类及其派生类提供数据源和储存库的一般视图，使程序员不必了解操作系统和基础设备的具体细节。

流涉及三个基本操作：

- 可以从流读取。读取是从流到数据结构（如字节数组）的数据传输。
- 可以向流写入。写入是从数据源到流的数据传输。
- 流可以支持查找。查找是对流内的当前位置进行查询和修改。

根据基础数据源或储存库，流可能只支持这些功能中的一部分。例如，NetworkStreams 不支持查找。Stream 的 CanRead、CanWrite 和 CanSeek 属性及其派生类决定不同的流所支持的操作。

FileObject 提供文本文件的表示形式。使用 FileObject 类来执行大多数典型的文本文件操作，例如读取、写入、追加、复制、删除、移动或重命名。还可以使用 FileObject 来检查并（在某些

情况下）设置文件属性、编码和路径信息。File、FileInfo、Path、Directory 和 DirectoryInfo 是密封类。可以创建这些类的新实例，但它们不能有派生类。

文件和流的差异：
System.IO 命名空间包含允许在数据流和文件上进行同步和异步读取及写入的类型。文件是一些具有永久存储及特定顺序的字节组成的一个有序的、具有名称的集合。因此，对于文件，人们常会想到目录路径、磁盘存储、文件和目录名等方面。相反，流提供一种向后备存储器写入字节和从后备存储器读取字节的方式，后备存储器可以为多种存储媒介之一。正如除磁盘外存在多种后备存储器一样，除文件流之外也存在多种流。例如，还存在网络流、内存流和磁带流等。

14、程序集与命名空间有什么不同？

程序集是 .NET Framework 应用程序的生成块；程序集构成了部署、版本控制、重复使用、激活范围控制和安全权限的基本单元。程序集是为协同工作而生成的类型和资源的集合，这些类型和资源构成了一个逻辑功能单元。程序集为公共语言运行库提供它识别类型实现所需的信息。对于运行库，类型不存在于程序集上下文之外。

程序集是 .NET Framework 编程的基本组成部分。程序集执行以下功能：

- 包含公共语言运行库执行的代码。如果可移植可执行 (PE) 文件没有相关联的程序集清单，则将不执行该文件中的 Microsoft 中间语言 (MSIL) 代码。请注意，每个程序集只能有一个入口点（即 DllMain、WinMain 或 Main）。
- 程序集形成安全边界。程序集就是在其中请求和授予权限的单元。程序集形成类型边界。每一类型的标识均包括该类型所驻留的程序集的名称。
- 程序集形成引用范围边界。程序集的清单包含用于解析类型和满足资源请求的程序集元数据。它指定在该程序集之外公开的类型和资源。该清单还枚举它所依赖的其他程序集。
- 程序集形成版本边界。程序集是公共语言运行库中最小的可版本化单元，同一程序集中的所有类型和资源均会被版本化为一个单元。程序集的清单描述您为任何依赖项程序集所指定的版本依赖性。
- 程序集形成部署单元。当一个应用程序启动时，只有该应用程序最初调用的程序集必须存在。其他程序集（例如本地化资源和包含实用工具类的程序集）可以按需检索。这就使应用程序在第一次下载时保持精简。
- 程序集是支持并行执行的单元。
- 程序集可以是静态的或动态的。静态程序集可以包括 .NET Framework 类型（接口和类），以及该程序集的资源（位图、JPEG 文件、资源文件等）。静态程序集存储在磁盘上的可移植可执行 (PE) 文件中。您还可以使用 .NET Framework 来创建动态程序集，动态程序集直接从内存运行并且在执行前不存储到磁盘上。您可以在执行动态程序集后将它们保存在磁盘上。

有几种创建程序集的方法。您可以使用过去用来创建 .dll 或 .exe 文件的开发工具，例如 Visual Studio .NET。您可以使用在 .NET Framework SDK 中提供的工具来创建带有在其他开发环境中创建的模块的程序集。您还可以使用公共语言运行库 API（例如Reflection.Emit）来创建动态程序集。

C# 程序是利用命名空间组织起来的。命名空间既用作程序的“内部”组织系统，也用作“外部”组织系统（一种向其他程序公开自己拥有的程序元素的方法）。using 指令是用来使命名空间用起来更方便。using 指令方便了对在其他命名空间中定义的命名空间和类型的使用。using 指令仅影响命名空间或类型名称和简单名称的名称解析过程，与声明不同，using 指令不会将新成员添加到它们与所在的编译单元或命名空间相对应的声明空间中。namespace 关键字用于声明一个范围。此命名空间范围允许您组织代码并为您提供了创建全局唯一类型的方法即使未显式声明命名空间，也会创建默认命名空间。该未命名的命名空间（有时称为全局命名空间）存在于每一个文件中。全局命名空间中的任何标识符都可用于命名的命名空间中。命名空间隐式

具有公共访问权，并且这是不可修改的。

C# 程序中的若干上下文要求指定命名空间名称或类型名称。两种形式的名称都写为以“.”标记分隔的一个或多个标识符。

namespace-name:（命名空间名称：）

namespace-or-type-name（命名空间或类型名称）

type-name:（类型名：）

namespace-or-type-name（命名空间或类型名称）

namespace-or-type-name:（命名空间或类型名称：）

identifier（标识符）
namespace-or-type-name . identifier（命名空间或类型名称 . 标识符）

“类型名”是一个“命名空间或类型名称”，它引用一个类型。需遵循下述的决策：“类型名”的“命名空间或类型名称”必须引用一个类型，否则将发生编译时错误。“命名空间名称”是一个“命名空间或类型名称”，它引用一个命名空间。需遵循下述的决策：“命名空间名称”的“命名空间或类型名称”必须引用一个命名空间，否则将发生编译时错误。“命名空间或类型名称”的含义按下述步骤确定：

- 如果“命名空间或类型名称”由单个标识符组成：
  - 如果“命名空间或类型名称”出现在类或结构声明体内，则从该类或结构声明开始查找，遍及每个封闭它的类或结构声明（若它们存在的话），如果具有给定名称的成员存在、可访问且表示类型，则“命名空间或类型名称”引用该成员。请注意，当确定“命名空间或类型名称”的含义时，会忽略非类型成员（常数、字段、方法、属性、索引器、运算符、实例构造函数、析构函数和静态构造函数）。
  - 否则，从发生“命名空间或类型名称”的命名空间开始，遍及每个封闭它的命名空间（若它们存在的话），直至全局命名空间结束，对下列步骤进行评估，直到找到实体：
  - 如果命名空间包含具有给定名称的命名空间成员，则“命名空间或类型名称”引用该成员，并根据该成员归为命名空间或类型类别。
  - 否则，如果命名空间有一个对应的命名空间声明，且“命名空间或类型名称”出现的位置包含在该命名空间声明中，则：

如果该命名空间声明包含一个将给定名称与一个导入的命名空间或类型关联的 using 别名指令，则“命名空间或类型名称”引用该命名空间或类型。

否则，如果该命名空间声明中有一个“using 命名空间指令”，它导入的那个命名空间内含有一个与给定名称完全匹配的类型，则“命名空间或类型名称”引用该类型。

否则，如果该“using 命名空间指令”导入的命名空间包含多个具有给定名称的类型，则“命名空间或类型名称”就被认为是含义不清的，将导致发生错误。

  - 否则，“命名空间或类型名称”就被认为是未定义的，导致发生编译时错误。

- 否则，“命名空间或类型名称”的形式为 `N.I`，其中 `N` 是由除最右边的标识符以外的所有标识符组成的“命名空间或类型名称”，`I` 是最右边的标识符。`N` 最先按“命名空间或类型名称”解析。如果对 `N` 的解析不成功，则发生编译时错误。否则，`N.I` 按下面这样解析：
  - 如果 `N` 是一个命名空间而 `I` 是该命名空间中可访问成员的名称，则 `N.I` 引用该成员，并根据该成员归

为命名空间或类型类别。

- 如果 `N` 是类或结构类型而 `I` 是 `N` 中可访问类型的名称，则 `N.I` 引用该类型。
- 否则，`N.I` 是无效的命名空间或类型名称并将发生编译时错误。

15、请编写一个捕获所有错误的错误处理代码？

```
Try{ (程序代码) }catch(System.Exception e){ (错误处理代码) }
```

一个try块只能有一个常规catch块。不允许一个try块有多个常规catch块，否则将导致编译时错误。（常规catch块：特殊的catch块，可以扑获任何类型的异常）。

异常是程序执行时遇到的任何错误情况或意外行为。以下这些情况都可以引发异常：您的代码或调用的代码（如共享库）中有错误，操作系统资源不可用，公共语言运行库遇到意外情况（如无法验证代码），等等。对于这些情况，应用程序可以从其中一些恢复，而对于另一些，则不能恢复。尽管可以从大多数应用程序异常中恢复，但不能从大多数运行库异常中恢复。

在 .NET Framework 中，异常是从 Exception 类类继承的对象。异常从发生问题的代码区域引发，然后沿堆栈向上传递，直到应用程序处理它或程序终止。

运行库如何管理异常
运行库使用基于异常对象和受保护代码块的异常处理模型。发生异常时，创建一个 Exception 对象来表示该异常。

运行库为每个可执行文件创建一个异常信息表。在异常信息表中，可执行文件的每个方法都有一个关联的异常处理信息数组（可以为空）。数组中的每一项描述一个受保护的代码块、任何与该代码关联的异常筛选器和任何异常处理程序（Catch 语句）。此异常表非常有效，在没有发生异常时，在处理器时间或内存使用上没有性能损失。仅在异常发生时使用资源。

异常信息表对于受保护的块有四种类型的异常处理程序：

- Finally 处理程序，它在每次块退出时都执行，不论退出是由正常控制流引起的还是由未处理的异常引起的。
- 错误处理程序，它在异常发生时必须执行，但在正常控制流完成时不执行。
- 类型筛选的处理程序，它处理指定类或该类的任何派生类的任何异常。
- 用户筛选的处理程序，它运行用户指定的代码，来确定异常应由关联的处理程序处理还是应传递给下一个受保护的块。

每种语言根据自己的规范实现这些异常处理程序。例如，Visual Basic .NET 通过 Catch 语句中的变量比较（使用 When 关键字）提供对用户筛选的处理程序的访问；C# 不实现用户筛选的处理程序。

异常发生时，运行库开始执行由下列两步组成的过程：

1. 运行库在数组中搜索满足下列条件的第一个受保护块：

- 保护包含当前执行的指令的区域，而且
- 包含异常处理程序或包含处理异常的筛选器。

2. 如果出现匹配项，运行库创建一个 Exception 对象来描述该异常。然后运行库执行位于发生异常的语句与处理该异常的语句之间的所有 Finally 语句或错误处理语句。请注意，异常处理程序的顺序很重要：最里面的异常处理程序最先计算。还请注意，异常处理程序可以访问捕捉异常的例程的局部变量和本地内

存，但引发异常时的任何中间值都会丢失。

如果当前方法中没有出现匹配项，则运行库搜索当前方法的每一个调用方，并沿着堆栈一直向上查找。如果任何调用方都没有匹配项，则运行库允许调试器访问该异常。如果调试器不能附加到该异常，则运行库引发 UnhandledException 事件。如果没有 UnhandledException 事件的侦听器，则运行库转储堆栈跟踪并结束程序。

筛选运行库异常可以按类型或按某些用户定义的条件对捕捉和处理的异常进行筛选。类型筛选的处理程序处理特定类型的异常（或从该异常派生的类）。最常见形式的类型筛选的异常处理程序指定仅捕捉特定类型的异常。有两种类型的异常：由执行程序生成的异常和由公共语言运行库生成的异常。错误发生时，运行库引发 SystemException 的适当派生类。这些错误是失败的运行库检查（如数组超出界限错误）导致的，它们可在任何方法的执行过程中发生。ApplicationException 由用户程序引发，而不是由运行库引发。如果设计创建新异常的应用程序，应从 ApplicationException 类派生那些异常。不建议捕捉 SystemException，在应用程序中引发 SystemException 也不是好的编程做法。

程序必须能够统一处理在执行期间发生的错误。公共语言运行库提供了一个平台，以统一的方式通知程序发生的错误，这样为设计容错软件提供了极大的帮助。所有的 .NET Framework 操作都通过引发异常来指示出现错误。

传统上，语言的错误处理模型依赖于语言检测错误和查找错误处理程序的独特方法，或者依赖于操作系统提供的错误处理机制。运行库实现的异常处理具有下列特点：

- 处理异常时不考虑生成异常的语言或处理异常的语言。
- 异常处理时不要求任何特定的语言语法，而是允许每种语言定义自己的语法。
- 允许跨进程甚至跨计算机边界引发异常。

与其他错误通知方法（如返回代码）相比，异常具有若干优点。不再有出现错误而不被人注意的情况。无效值不会继续在系统中传播。不必检查返回代码。可以轻松添加异常处理代码，以增加程序的可靠性。最后，运行库的异常处理比基于 Windows 的 C++ 错误处理更快。

由于执行线程例行地遍历托管代码块和非托管代码块，因此运行库可以在托管代码或非托管代码中引发或捕捉异常。非托管代码可以同时包含 C++ 样式的 SEH 异常和基于 COM 的 HRESULT。

16. 委托与事件是什么关系？为什么要使用委托？

运行库支持称为委托的引用类型，其作用类似于 C++ 中函数指针的用途。与函数指针不同，委托实例独立于它所封装的方法的类；最主要的是那些方法与委托的类型是兼容的。另外，函数指针只能引用静态函数，而委托可以引用静态和实例方法。委托主要用于 .NET Framework 中的事件处理程序和回调函数。

委托适用于那种在某些其他语言中需用函数指针来解决的情况（场合）。但是，与函数指针不同，委托是面向对象和类型安全的。delegate 声明定义一种引用类型，该类型可用于将方法用特定的签名封装。委托实例封装静态方法或实例方法。委托大致类似于 C++ 中的函数指针。委托声明定义一个类，它是从 `System.Delegate` 类派生的类。委托实例封装了一个调用列表，该列表列出了一个或多个方法，每个方法称为一个可调用实体。对于实例方法，可调用实体由一个实例和该实例的方法组成。对于静态方法，可调用实体仅由一个方法组成。如果用一组合适的参数来调用一个委托实例，则该委托实例所封装的每个可调用实体都会被调用，并且用的都是上述的同一组参数，就是用此给定的参数集来调用该委托实例的每个可调用实体。

委托是用来处理其他语言（如 C++、Pascal 和 Modula）需用函数指针来处理的情况的。不过与 C++ 函数指针不同，委托是完全面对对象的；另外，C++ 指针仅指向成员函数，而委托同时封装了对象实例和方法。

委托声明定义一个从 System.Delegate 类派生的类。委托实例封装一个调用列表，该列表列出一个或多个方法，其中每个方法均作为一个可调用实体来引用。对于实例方法，可调用实体由该方法和一个相关联的实例组成。对于静态方法，可调用实体仅由一个方法组成。用一个适当的参数集来调用一个委托实例委托实例的一个有趣且有用的属性是：它既不知道也不关心有关它所封装的方法所属的类的种种详情；对它来说最重要的是这些方法与该委托的类型兼容。这使委托非常适合“匿名”调用。这是一个强大的功能。委托是一个可以对方法进行引用的类。与其他的类不同，委托类具有一个签名，并且它只能对与其签名匹配的方法进行引用。这样，委托就等效于一个类型安全函数指针或一个回调。委托使您得以将函数作为参数传递。委托的类型安全要求作为委托传递的函数拥有同委托声明相同的签名。

定义和使用委托分三个步骤：声明、实例化和调用。
所有委托都是从 System.Delegate 继承而来的，并且有一个调用列表，这是在调用委托时所执行的方法的一个链接列表。产生的委托可以用匹配的签名引用任何方法。没有为具有返回类型并在调用列表中包含多个方法的委托定义返回值。

可以使用委托的 Combine 和 Remove 方法在其调用列表中添加和移除方法。若要调用委托，可使用 Invoke 方法，或者使用 BeginInvoke 和 EndInvoke 方法异步调用委托。委托类的实现是由运行库提供的，而不是由用户代码提供的。

委托是事件的基础，.NET Framework 中的事件是基于委托模型的。

event指定一个事件:

event 关键字使您得以指定当代码中的某些“事件”发生时调用的委托。此委托可以有一个或多个关联的方法，当代码指示该事件已发生时将调用关联的方法。可使一个程序中的事件用于面向 .NET Framework 公共语言运行库的其他程序。

为了创建并使用 C# 事件，必须采取以下步骤：

1. 创建或标识一个委托。如果正在定义自己的事件，还必须确保有与事件关键字一起使用的委托。如果已经预定义了事件（例如在 .NET Framework 中），则事件的使用者只需要知道委托的名称。

2. 创建一个类，包含：

a. 从委托创建的事件。

b. （可选）验证用 event 关键字声明的委托实例是否存在的方法。否则，该逻辑必须放置在引发此事件的代码中。

c. 调用此事件的方法。这些方法可以重写一些基类功能。

3. 定义一个或多个将方法连接到事件的类。所有这些类都包括：

- 使用 += 运算符和 -= 运算符将一个或多个方法与基类中的事件关联。
- 将与事件关联的方法的定义。

4. 使用此事件：

- 创建包含事件声明的类对象。

- 使用定义的构造函数，创建包含事件定义的类对象。

事件是对象发送的消息，以发信号通知操作的发生。操作可能是由用户交互（例如鼠标单击）引起的，也可能是由某些其他的程序逻辑触发的。引发（触发）事件的对象叫做事件发送方。捕获事件并对其作出响应的对象叫做事件接收方。
在事件通信中，事件发送方类不知道哪个对象或方法将接收到（处理）它引发的事件。所需要的是在源和接收方之间存在一个媒介（或类似指针的机制）。.NET Framework 定义了一个特殊的类型（Delegate），该类型提供函数指针的功能。

事件委托是多路广播的，这意味着它们可以对多个事件处理方法进行引用。有关详细信息，请参见 Delegate。委托考虑了事件处理中的灵活性和精确控制。通过维护事件的已注册事件处理程序列表，委托为引发事件的类担当事件发送器的角色。

事件功能是由三个互相联系的元素提供的：提供事件数据的类、事件委托和引发事件的类。.NET Framework 具有命名与事件相关的类和方法的约定。如果想要您的类引发一个名为 EventName 的事件，您需要以下元素。

- 持有事件数据的类，名为 EventNameEventArgs。该类必须从 System.EventArgs 导出。
- 事件的委托，名为 EventNameEventHandler。
- 引发事件的类。该类必须提供：

a. 事件声明。

b. 引发事件的方法。

.NET Framework 类库或第三方类库中可能已经定义了事件数据类和事件委托类。

17、一个类中有几种元素？

一个类可包含下列成员的声明：构造函数、析构函数、常数、字段、方法、属性 、索引器、运算符、事件、委托、类、接口、结构类可以定义对象可执行的操作（方法、事件或属性），并定义保存对象（字段）状态的值。尽管类通常同时包含定义和实现，但它也可以包含没有实现的一个或多个成员。类的实例是对象。可以通过调用对象的方法并访问其属性、事件和字段，来访问对象的功能。

特征	说明
sealed	指定不能从这种类型派生出另一种类型。
implements	指出该类通过提供接口成员的实现，使用一个或多个接口。
abstract	指定不能创建类的实例。若要使用它，必须由其派生出另一个类。
inherits	指出可以在指定了基类的任何地方使用类的实例。从基类继承的派生类可以使用基类提供的任何虚方法的实现，或者派生类可以用自己的实现重写它们。
exported 或 not exported	指出某个类在定义它的程序集之外是否可见。仅适用于顶级类。

嵌套类也有成员特征。没有实现的类成员是抽象成员。有一个或更多抽象成员的类其本身也是抽象的；不可以创建它的新实例。以运行库为目标的某些语言允许将类标记为抽象，即使其成员都不是抽象的也是如此。当需要封装一组派生类可在适当时候继承或重写的基本功能时，可以使用抽象类。非抽象的类称为具体类。类可以实现任何数量的接口，但它只能从一个基类继承。所有的类都必须至少有一个构造函数，该函数初始化此类的新实例。
18、请解释这种语法现象Session[“name”]=20;

索引器索引器允许类或结构的实例按照与数组相同的方式进行索引。索引器类似于属性，不同的是它们的访问器采用参数。
索引器使您得以按照与数组相同的方式为类或结构实例建立索引。索引器的 get 访问器体与方法体类似。它返回索引器的类型。get 访问器使用与索引器相同的 formal-index-parameter-list。索引器的 set 访问器体与方法体类似。除了 value 隐式参数外，它还使用与索引器相同的 formal-index-parameter-list。索引器的类型和 formal-index-parameter-list 中引用的每个类型必须至少与索引器本身一样是可访问的。索引器的签名由其形参的数量和类型组成。它不包括索引器类型或形参名。如果在同一类中声明一个以上的索引器，则它们必须具有不同的签名。索引器值不作为变量来分类；因此，不可能将索引器值作为 ref 或 out 参数来传递。若要为索引器提供可由其他语言用于默认索引属性的名称，可在声明中使用 name 属性。

索引器允许类或结构的实例按照与数组相同的方式进行索引。索引器类似于属性，不同的是它们的访问器采用参数。属性的访问器包含与获取（读取或计算）或设置（写）属性有关的可执行语句。访问器声明可以包含 get 访问器或 set 访问器，或者两者均包含。get 访问器体与方法体相似。它必须返回属性类型的值。执行 get 访问器相当于读取字段的值。set 访问器与返回 void 的方法类似。它使用称为 value 的隐式参数，此参数的类型是属性的类型。

属性是类、结构和接口的命名成员。它们提供了通过访问器读、写或计算私有字段值的灵活机制。属性按如下方式，根据所使用的访问器进行分类：

- 只带有 get 访问器的属性称为只读属性。无法对只读属性赋值。
- 只带有 set 访问器的属性称为只写属性。只写属性除作为赋值的目标外，无法对其进行引用。
- 同时带有 get 和 set 访问器的属性为读写属性。

在属性声明中，get 和 set 访问器都必须在属性体的内部声明。

索引器与属性类似。除下表中显示的差别外，为属性访问器定义的所有规则同样适用于索引器访问器。

属性	索引器
通过名称标识。	通过签名标识。
通过简单名称或成员访问来访问。	通过元素访问来访问。
可以为静态成员或实例成员。	必须为实例成员。
属性的 get 访问器没有参数。	索引器的 get 访问器具有与索引器相同的形参表。
属性的 set 访问器包含隐式 value 参数。	除了 value 参数外，索引器的 set 访问器还具有与索引器相同的形参表。

19、装箱与取消装箱是什么含义？

装箱 (boxing)

值类型实例到对象的转换，它暗示在运行时实例将携带完整的类型信息，并在堆中分配。Microsoft 中间语言 (MSIL) 指令集的 box 指令，通过复制值类型，并将它嵌入到新分配的对象中，将值类型转换为对象。

取消装箱 (unboxing)

将对象实例转换为值类型。

20、一个构造函数能否调用另一个构造函数，如果能请写出简单的代码

创建新对象时将调用类构造函数，一个类可以有多个构造函数。可以声明一个不带参数的构造函数，和一个带参数的构造函数。如果类没有构造函数，将自动生成一个默认的无参数构造函数，并使用默认值初始化对象字段（例如，int 将初始化为 0）。类构造函数可通过初始值设定项来调用基类的构造函数，通过基类构造函数初始化字段。

```
public Cylinder(double readius,double height):base(radius,height)
{                       }
```

类构造函数也可通过关键字 this 调用同一个类的另一个构造函数。

```
Public Point():this(0,20){                      }
```

无参数构造函数 Point() 调用了另一个带有两个参数的构造函数，将默认位置初始化为 (0, 20)。

三种类构造函数：

类构造函数的类型	注释
实例	用于创建并初始化类的实例。
私有	在类之外不可访问的特殊类型实例构造函数。无法用私有构造函数来实例化类。
静态	在创建第一个实例或引用任何静态成员之前，将自动调用这种构造函数来初始化类。无法直接调用这种构造函数。

私有构造函数是一种特殊的实例构造函数。它通常用在只包含静态成员的类中。如果类具有一个或多个私有构造函数而没有公共构造函数，则不允许其他类（除了嵌套类）创建该类的实例。声明空构造函数可阻止自动生成默认构造函数。注意，如果您不对构造函数使用访问修饰符，则在默认情况下它仍为私有构造函数。但是，通常显式地使用 private（私有）修饰符来清楚地表明该类不能被实例化。

静态构造函数用于初始化类。在创建第一个实例或引用任何静态成员之前，将自动调用静态构造函数来初始化类。静态构造函数既没有访问修饰符，也没有参数。在创建第一个实例或引用任何静态成员之前，将自动调用静态构造函数来初始化类。无法直接调用静态构造函数。在程序中，用户无法控制何时执行静态构造函数。静态构造函数的典型用途是：当类使用日志文件时，将使用这种构造函数向日志文件中写入项。当没有实例字段或实例方法（如 Math 类）或调用方法以获得类的实例时，私有构造函数可用于阻止创建类。

结构构造函数类似于类构造函数，只是存在以下差异：

- 结构不能包含显式的无参数构造函数。结构成员将自动初始化为它们的默认值。
- 结构不能有以下形式的初始值设定项：base (argument-list)。

析构函数：析构函数用于销毁类的实例不能对结构使用析构函数。只能对类使用析构函数。一个类只能有一个析构函数。无法继承或重载析构函数。无法调用析构函数。它们是被自动调用的。析构函数既没有修饰符，也没有参数。

21、请编写创建一个线程的代码。

.NET Framework 通过提供面向对象的线程模型来使您能够快速、方便地创建多线程应用程序。新建一个线程相当简单，只需将其声明并为其提供线程起始点处的方法委托。当您准备好在线程上开始执行时，请调用 Thread.Start 方法。当使用多个执行线程时，涉及到多个特殊注意事项。

创建新的执行线程

1. 声明该线程。

```
System.Threading.Thread  myThread;
```

2. 用线程起始点的适当委托实例化该线程。使用 AddressOf 运算符在 Visual Basic 中创建委托，或在 C# 中创建新的 ThreadStart 对象。

```
MyThread=new  System.Threading.Thread(new  System.Threading.ThreadStart(myStartingMethod));
```