C#.NET笔试题 高级进阶篇：共计34道10396字

1.说说什么是架构模式。

1，分层。
2，分割。
分层是对网站进行横向的切分，那么分割就是对网站进行纵向的切分。将网站按照不同业务分割成小应用，可以有效控制网站的复杂程度。
3，分布式。
在大型网站中，分层和分割后主要是为了让网站能够便于分布式部署，也就是将不同的模块部署到不同的服务器上。常用的分布式方案有如下几种。
3.1 分布式应用和服务。

3.2 分布式静态资源。
3.3 分布式数据和存储
3.4 分布式计算。
3.5 此外还有分布式配置，分布式锁，分布式文件系统等。
4，集群。
分布式方案只是将不同的模块或服务独立部署到服务器上，但通常还是单台服务器。集群则是将同一个模块或服务同时部署到多台服务器上，通过负载均衡设备对外提供服务。
5，缓存。
常用的缓存有如下几种，CDN，反向代理，本地缓存，分布式缓存。
6，异步。
异步一般通过队列的方式来实现。在单一服务器中，可以通过多线程共享内存队列实现异步。在分布式系统中，可以通过分布式消息队列实现。
异步有一些作用，描述如下。
6.1，提高系统可用性。
6.2，加快网站响应速度。
6.3，消除并发访问高峰。
7，冗余
冗余的目的是实现高可用性。是通过使用集群来实现的。即使是再小的服务，也要部署到至少2台服务器上。
数据库的冗余有冷备份和热备份两种。
冗余还包括在异地建立灾备数据中心。
8，自动化
自动化包括自动化代码管理，自动化部署，自动化测试，自动化安全检查等。
9，安全
安全主要通过哦密码和手机验证码的方式实现。

2.架构的5大要素是哪5大要素?

1.高性能架构

2.高可用架构

3.伸缩性架构

4.可扩展架构

5.安全架构

3.说说什么事集群，什么是分布式。

集群: 多台服务器来完成业务处理，业务请求来了以后，每一台服务器都能独立的完成业务计算；每个服务器都是独立的个体；多台服务器集合起来，通常也需要做负载均衡；

分布式：多台服务器完成业务处理，也去请求来了以后，需要多个服务器合作来完成，比方说一个业务处理有5个环节，可能在处理的时候，A服务器处理第一个环节，B服务器处理第二个化解，C服务器处理第三个环节，D服务器处理第四个环节，E服务器来处理第五个环节；一个业务请求的5个环节，被五台服务器来合作完成了一个业务连，此类架构叫分布式架构；

4.说说对Redis的理解

Redis是一种基于键值对的NoSQL数据库（非关系型数据库）；是一个key-value存储系统
Redis有两个特点：高能性 可靠性
高能性：Redis将所有数据都存储在内存中，所有读写性特别高
可靠性：Redis将内存中的数据利用RDB和AOF的形式保存到硬盘中，这样就可以避免发生断点或机器故障时内存数据丢失的问题

功能应用：
1.数据缓存功能，减少对数据库的访问压力
2.消息队列功能(轻量级): Redis提供了发布订阅功能和阻塞队列功能
3.计数器-应用保存用户凭证
比如计算浏览数，如果每次操作都要做数据库的对应更新操作，那将会给数据库的性能带来极大的挑战
缓存：优化网站性能，首页 （不常变的信息）
存储：单点登录，购物车
计数器：登录次数限制，incr
时效性：验证码expire
订单号：数字
redis有哪些应用场景场景?
1.缓存数据服务器
eg: SSO单点登录
2.应对高速读写的场景
eg: 秒杀高可用
3.分布式锁
eg: 秒杀数据一致性
4.数据共享
eg: 库存数据

5.你所了解的NoSql除了Redis还有哪些?

Memcache/MongoDb

6.谈谈对消息队列的理解

对于传统的应用程序，如果需要向另一个应用程序发送信息，只需要向其发出请求即可！这种方式虽然简单直接，但是如果应用程序2突然挂了，应用程序1可能会因为服务异常，而无法继续提供服务！
设想一下，在应用程序1和应用程序2之间，插入一个消息服务，主要用于接受消息和发送消息，这样应用程序1和应用程序2之间的依赖关系就解耦了，同时也不会因为任何一方当服务不可用时，无法继续提供服务！

其中插入的消息服务被称为消息队列！

消息队列带来的优势：

1.程序解耦：应用程序1和应用程序2在进行交互时，不会因为一方服务中断而导致服务停止；

2.异步处理：程序解耦之后，带来的最大的好处就是可以异步处理，应用程序1只管把消息发送到消息中间件，应用程序2只需要从消息中间件中接受消息然后进行处理即可；
同时，基于异步处理特性，在某些业务场景下，例如商品秒杀活动，引入消息队列之后，当客户端请求量很大的时候，可以有效的进行流量削峰！
如果没有中间层做缓冲，当进行商品秒杀时，一下突然大量请求涌入，很可能造成系统直接瘫痪，甚至宕机！

消息队列也会带来很明显的弊端：

1.系统可用性降低：在引入消息队列之前，你不用考虑消息丢失或者消息队列服务挂掉等等的情况，但是引入消息队列之后你就需要去考虑这些问题！

2.系统复杂性提高：加入消息队列之后，你需要保证消息没有被重复消费、处理消息没有被正确处理的情况等等问题！

引入消息队列虽然会带来一些问题，俗话说，兵来将挡、水来土掩，这句话同样适用于 IT 开发者，有坑填坑！

对于系统可用性降低方面，通常常用的解决方案就是搭建消息服务集群，具体技术实现上可以是主从架构或者分布式架构，即时一台消息队列服务机器挂了，也不会影响消息队列无法提供服务！

对于系统复杂性提高方面，常用的解决方案也很多，例如接受者接受到消息之后，可以先将消息写入数据库，即时没有被正确处理，还可以走人工处理，或者消息消费失败，将消息重新入队等待下一次消费等等。

7.说说你对数据库读写分离的理解

读写分离，基本的原理是让主数据库处理事务性增、改、删操作（INSERT、UPDATE、DELETE），而从数据库处理SELECT查询操作。数据库复制被用来把事务性操作导致的变更同步到集群中的从数据库。

为什么要分库、分表、读写分?
单表的数据量限制，当单表数据量到一定条数之后数据库性能会显著下降。数据多了之后，对数据库的读、写就会很多。分库减少单台数据库的压力。接触过几个分库分表的系统，都是通过主键进行散列分裤分表的。这类数据比较特殊，主键就是唯一的获取该条信息的主要途径。比如：京东的订单、财付通的交易记录等。。。该类数据的用法，就是通过订单号、交易号来查询该笔订单、交易。

还有一类数据，比如用户信息，每个用户都有系统内部的一个userid，与userid对应的还有用户看到的登录名。那么如果分库分表的时候单纯通过userid进行散列分库，那么根据登录名来获取用户的信息，就无法知道该用户处于哪个数据库中。

8.如果你的系统功能中出现性能问题，说说你的排查计划。

通过浏览器访问，定位到性能最差（耗时最长）的请求,然后定位是代码层面的性能损失，还是数据库层面的性能损失，然后根据不同环节出现的性能问题再加以解决。

9.请列出常见的缓存方式，并简要概述其优缺点。

客户端缓存，本地缓存，反向代理缓存，分布式缓存；

10.谈谈对通信加密解密的理解。

加密解密分为：对称可逆加密，非对称可逆加密两大类；

可逆加密：加密后得到密文，可以通过加密后的密文得到原文；

对称可逆加密特点：有一个公开的加密算法，任何人都知道；有一组Key，分为加密Key和解密Key，且两个Key是相同的；使用当前这个Key加密，可以得到一段密文；同时如果这段密文想要得到原文，也必须得使用这个Key才能解密； 此类被称为对称可逆加密，性能很高，但是安全性较差；只要是key被泄密了，密文就可以被攻破得到原文；因为加密算法是公开的；

非对称可逆加密特点：：有一个公开的加密算法，任何人都知道；同时有一对Key,这一组Key是成套的，两个Key不相同，二者且不能相互推导；一个Key作为加密Key，一个Key作为解密Key，且加密Key加密后，只能由这个解密Key才能解开；此类被称为非对称可逆加密；

在非对称可逆加密的应用中，有一个公钥和私钥的概念；公钥：把其中的一个Key公开，私钥：把其中的一个Key私有化；那这样就有一下场景：

1.公开加密Key，私有解密Key：那么任何一个拥有公开加密Key的人给我这个拥有私有解密Key的人发密文，我都能解开，且只有我能解开；这样就可以保证在通信传输的时候，保证信息的安全；在传输的时候不会被泄密，因为就算在传输的过程中，密文被拦截了，也无法得到原文；因为没有这个解密Key，有密文是无法得到原文的；

2.公开解密Key,私有化加密Key: 那么任何一个拥有解密Key的人都能够接收到来自于我这个拥有加密Key的人发送的消息，只要是我这个私有的加密Key加密后的密文，任何一个拥有解密Key的人都能够解开密文得到原文；那这样就可以实现一个功能：防止抵赖，也就是说，如果我是有解密Key的人，我得到的密文只要能够解开，那就说明这段密文一定是拥有加密Key的人发出来的；不然我是解不开这段密文

的；

11.CTS、CLS、CLR分别作何解释?

CTS：Common Type System 通用系统类型，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。
Int32、Int16->int、String->string、Boolean->bool，前者都是.Net中的类型，后者是C#中对这些类型的别名。

CLR：Common Language Runtime 公共语言运行时，也即使项目运行需要的环境，就如我们人生存需要水。

CLS：Common Language Specification 通用语言规范，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。

12.PDB是什么东西? 在调试中它应该放在哪里?

PDB是用于保存调试和项目状态信息的文件，在debug的时候将产生pdb文件，调试的时候应该放在和对应应用程序集相同目录。

13.C#中Params是什么含义？有何用途?

Params 关键字在方法成员的参数列表中使用，为该方法提供了参数个数可变的能力
它在只能出现一次并且不能在其后再有参数定义，之前可以

实例：

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
    class App
    {
        //第一个参数必须是整型，但后面的参数个数是可变的。
        //而且由于定的是object数组，所有的数据类型都可以做为参数传入
        public static void UseParams(int id, params object[] list)
        {
            Console.WriteLine(id);
            for (int i = 0; i < list.Length; i++)
            {
                Console.WriteLine(list);
            }
        }

        static void Main()
        {
            //可变参数部分传入了三个参数，都是字符串类型
            UseParams(1, "a", "b", "c");
            //可变参数部分传入了四个参数，分别为字符串、整数、浮点数和双精度浮点数数组
            UseParams(2, "d", 100, 33.33, new double[] { 1.1, 2.2 });
            Console.ReadLine();
        }
    }
}
```

结果：

```
1
a
b
c
2
d
100
33.33
System.Double[]
```

14.说说对http 301、302、303，304、400、405、415状态码的认识；

301表示永久重定向（301 moved permanently），表示请求的资源分配了新url，以后应使用新url。
302表示临时性重定向（302 found），请求的资源临时分配了新url，本次请求暂且使用新url。302与301的区别是，302表示临时性重定向，重定向的url还有可能还会改变。
303 表示请求的资源路径发生改变，使用GET方法请求新url。她与302的功能一样，但是明确指出使用GET方法请求新url。

304 not modified
客户端发送附带条件的请求时（if-matched,if-modified-since,if-none-match,if-range,if-unmodified-since任一个）服务器端允许请求访问资源，但因发生请求未满足条件的情况后，直接返回304Modified（服务器端资源未改变，可直接使用客户端未过期的缓存）。304状态码返回时，不包含任何响应的主体部分。304虽然被划分在3xx类别中，但是和重定向没有关系。

400 bad request
表示请求的报文中存在语法错误，比如url含有非法字符。提交json时，如果json格式有问题，接收端接收json，也会出现400 bad request，比如常见的json串，数组不应该有",但是有"了。

405 method not allowed
问题原因： 请求的方式（get、post、delete）方法与后台规定的方式不符合。比如： 后台方法规定的请求方式只接受get，如果用post请求，就会出现 405 method not allowed的提示

415
后台程序不支持提交的content-type，就会返回415，spring mvc错误信息如下
The server refused this request because the request entity is in a format not supported by the requested resource for the requested method.
unsupported media type

15.请使用C#的相关知识，封装一个泛型的数据库访问查询方法；

```
 public T Get<T>(int id) where T : BaseModel
        {
            string ConnectionString = "Data Source=DESKTOP-63QE7M1;
Database=CustomerDB; User ID=sa;                Password=sa123;
MultipleActiveResultSets=True";
            Type type = typeof(T);
            var propList = type.GetProperties().Select(p => $"[{p.Name}]");
            string props = string.Join(',', propList);
            string tableName = type.Name;
            string StringSql = $"select {props} from [{tableName}] where id=" +
id;
            object oInstance = Activator.CreateInstance(type);
            using (SqlConnection connection = new
SqlConnection(ConnectionString))
```

```
        {
            connection.Open();
            SqlCommand sqlCommand = new SqlCommand(StringSql, connection);
            SqlDataReader reader = sqlCommand.ExecuteReader();
            reader.Read();
            foreach (var prop in type.GetProperties())
            {
                prop.SetValue(oInstance, reader[prop.Name]);
            }
        }
        return (T)oInstance;
    }
```

16.什么是异步编程?

1.异步编程就是在方法调用后立即返回，不会阻塞后续代码执行。
2.异步代码执行完毕后一般会通过回调的形式调用指定的方法，从而完成异步代码块与主代码块（主线程）的通讯。
3.javascript中的异步编程能力都是由浏览器提供的，如setTimeout， XMLHttpRequest，还有DOM的事件机制，还有HTML5新增加的webwork, postMessage，等等很多。这些东西都有一个共同的特点，就是拥有一个回调函数。
4.消除需要长时间执行的代码块阻塞整个代码的执行。
5.延迟执行一段代码。

17.架构模式(Architectural Pattern) 设计模式(Design Pattern) 代码模式(Coding Pattern)的区别是什么?

区别：在于三种不同的模式存在于它们各自的抽象层次和具体层次。
架构模式是一个系统的高层次策略，涉及到大尺度的组件以及整体性质。架构模式的好坏可以影响到总体布局和框架性结构。
计模式是中等尺度的结构策略。这些中等尺度的结构实现了一些大尺度组件的行为和它们之间的关系。模式的好坏不会影响到系统的总体布局和总体框架。设计模式定义出子系统或组件的微观结构。
码模式是特定的范例和与特定语言有关的编程技巧。代码模式的好坏会影响到一个中等尺度组件的内部、外部的结构或行为的底层细节，但不会影响到一个部件或子系统的中等尺度的结构，更不会影响到系统的总体布局和大尺度框架

18.软件架构的目标是什么?

可靠性（Reliable）：
–软件系统对于用户的商业经营和管理来说极为重要，因此软件系统必须非常可靠。
全性（Secure）：
–软件系统所承担的交易的商业价值极高，系统的安全性非常重要。
可伸缩性（Scalable）：
–软件必须能够在用户的使用率、用户的数目增加很快的情况下，保持合理的性能。只有这样，才能适应用户的市场扩展得可能性。
可定制化（Customizable）：
–同样的一套软件，可以根据客户群的不同和市场需求的变化进行调整。
可扩展性（Extensible）：
–在新技术出现的时候，一个软件系统应当允许导入新技术，从而对现有系统进行功能和性能的扩展
维护性（Maintainable）：
–软件系统的维护包括两方面，一是排除现有的错误，二是将新的软件需求反映到现有系统中去。一个易于维护的系统可以有效地降低技术支持的花费。

客户体验（Customer Experience）：
–软件系统必须易于使用。
市场时机（Time to Market）：
–软件用户要面临同业竞争，软件提供商也要面临同业竞争。以最快的速度争夺市场先机非常重要。

19.什么是迪米特法则?

在软件系统中，一个模块设计得好不好的标志，就是该模块在多大的程度上将自己的内部数据和其他与实现有关的细节隐藏起来。这一概念就是"信息的隐藏"，或者叫做"封装"，也就是大家熟悉的软件设计的基本教义之一。
信息的隐藏非常重要的原因在于，它可以使各个子系统之间脱耦。这种脱耦化可以有效地加快系统的开发过程，因为可以独立地同时开发各个模块。它可以使维护过程变得容易，因为所有的模块都容易读懂，特别是不必担心对其他模块的影响。
旦确认某一个模块是性能的障碍时，设计人员可以到对这个模块本身进行优化，而不必担心影响到其他的模块。 信息的隐藏可以促进软件的复用。一个系统的规模越大，信息的隐藏就越是重 要，而信息隐藏的威力也就越明显。

20.解释一下里氏替换原则?

里氏代换原则的严格表达是：一个软件实体如果使用的是一个基类的话，那么一定适用于其子类，而且它 根本不能察觉出基类对象和子类对象的区别。
里氏代换原则是继承复用的基石。只有当衍生类可以替换掉基类，软件单位的功能不会受到影响时，基类才能真正被复用，而衍生类也才能够在基类的基础上增加新的行为。

21.什么是依赖倒转IOC原则?

在面向对象的系统里面，两个类之间可以发生三种不同的耦合关系：
–零耦合关系：如果两个类没有耦合关系，就称之为零耦合。
–具体耦合关系：具体耦合发生在两个具体的(可实例化 的)类之间，经出一个类对另外一个具体类的直接引用造成。
–抽象耦合关系：抽象耦合关系发生在一个具体类和一个抽象类之间，使两个必须发生关系的类之间存有最大的灵活性。
简单地说，依赖倒转原则要求客户端依赖于抽象耦合

22.为何说基于 SOAP 的服务是重量级的服务，Rest是轻量级的?

Restful Web服务是一种基于 REST 和 HTTP 协议的轻量级Web 服务，它把Web应用系统中的一切都当作是资源，它利用标准的HTTP请求方法（GET、POST、PUT 和 DELETE等），以URL的形式访问(功能调用)Web资源。
JAVA中共有三种Web Service规范，分别JAX-WS(JAX-RPC)、JAX-RS、JAXM&SAAJ。其中有两种SOAP Web Service规范：JAX-WS和SAAJ。JAX-RS是Rest服务定义。使用最多的是JAX-WS和JAX-RS。
JAX-RS (JSR 311) JAX-WS (JSR 224)
适用范围 适用于简单的远程数据访问 适用于复杂的数据交互模式
消息传输 请求和响应通过 HTTP 消息正文来传输 请求和响应被放在 SOAP 消息，作为传输层（HTTP, JMS, SMTP 等）消息正文进行传输
客户端调用 只需要一个 URL就能调用 Web 服务 需要编写标准的客户端代码，对 Web Services进行访问
安全性 简单的认证和授权机制 丰富的安全策略，可实现各种安全需求
访问终端 桌面浏览器或移动终端浏览器 桌面、浏览器
Java EE6引入了对JSR-311的支持。JSR-311（JAX-RS：Java API for RESTful Web Services）旨在定义一个统一的规范，使得 Java 程序员可以使用一套固定的接口来开发 REST 应用，避免了依赖于第三方框架。同时，JAX-RS使用POJO编程模型和基于标注的配置，并集成JAXB，从而可以有效缩短 REST 应用的开发周期。

JAX-RS定义的API位于javax.ws.rs包中。
JAX-RS的具体实现由第三方提供，例如Sun的参考实现 Jersey、Apache 的 CXF以及JBoss的RESTEasy等。
SOAP比较复杂，基于XML，有对应规范；REST利用HTTP请请求方式GET，POST，PUT约定事务操作。简单的说，SOAP通过传输XML，XML定义了请求和响应的具体数据，要进行的操作等等；而REST则是另一种约定，比如请求/user/1001这个RUL，GET方式返回id为1001的user信息，POST方式则是更新id为1001的user信息，DELETE删除等。

23.基于REST 和基于SOAP的服务的区别是什么?

传统的 Web 服务通过简单对象访问协议 (SOAP) 进行消息的交换。它是一种用于一单向通信的消息格式，将消息组合成 XML 文档。描述了消息的传输，主要是通过 HTTP 协议。它定义一组 RPC 调用与 SOAP 消息互相转换的契约，将 RPC 调用封装为 SOAP 消息进行传输，并在服务器端反向转换为服务器端 RPC 调用，最终结果再以类似机制返回给客户端。
基于 SOAP 的服务是重量级的服务，因为它有严格的约束和标准，开发人员需要深入了解基于 SOAP 的 web 服务中用到的关键技术：XML，WSDL、SOAP 及 UDDI，这样就缺少了开发的灵活性。
相比基于 XML 技术的其它臃肿的 web 服务而言，REST 显得更加简洁，更轻量级。增删查改是应用软件里面最常见的操作，而在 HTTP 方法中，正好有其对应的方法实现，所以可以有效的降低复杂度。同时也能够满足异构平台之间的交互。
当前，基于SOAP的服务和REST二者都有一定的应用场景，互联网上的很多应用，同时提供基于SOAP的服务和REST。

24.工厂模式,简单工厂模式,抽象工厂模式三者有什么区别

工厂模式，也叫做说虚构造器，在简单工厂中间插入了一个具体产品工厂，这个工厂知道产品构造时候的具体细节，而简单工厂模式的产品具体构造细节是在一个个if/else分支，或者在switch/case分支里面的。工厂模式的好处就在于将工厂和产品之间的耦合降低，将具体产品的构造过程放在了具体工厂类里面。在以后扩展产品的时候方便很多，只需要添加一个工厂类，一个产品类，就能方便的添加产品，而不需要修改原有的代码。而在简单工厂中，如果要增加一个产品，则需要修改工厂类，增加if/else分支，或者增加一个case分支，工厂模式符合软件开发中的OCP原则（open close principle），对扩展开放，对修改关闭。
抽象工厂模式：这个模式我总是感觉和builder模式非常相似。
工厂方法模式提供的是对一个产品的等级模式，，而抽象工厂方法提供的是对多个产品的等级模式，注意，这里的多个具体产品之间是相互耦合的，也就是说这里的抽象工厂提供的产品之间是存在某种联系的。
有人做如下的比较：
工厂方法模式：一个抽象产品类，可以派生出多个具体产品类。
一个抽象工厂类，可以派生出多个具体工厂类。
每个具体工厂类只能创建一个具体产品类的实例。
抽象工厂模式：多个抽象产品类，每个抽象产品类可以派生出多个具体产品类。
一个抽象工厂类，可以派生出多个具体工厂类。
每个具体工厂类可以创建多个具体产品类的实例。
区别：工厂方法模式只有一个抽象产品类，而抽象工厂模式有多个。
工厂方法模式的具体工厂类只能创建一个具体产品类的实例，而抽象工厂模式可以创建多个。
下面是一个形象的比喻：
无论是简单工厂模式、工厂模式还是抽象工厂模式，它们本质上都是将不变的部分提取出来，将可变的部分留作接口，以达到最大程度上的复用。拿一个生产水杯（cup）的工厂举例：起初，不用工厂模式，我必须在生产水杯之前知道水杯的材料和形状等水杯的所有特征才能生产，这就是我们的new Cup();这个Cup必须是具体的。厂主发现同一形状的被子，只是材料不同,如一个是玻璃(glass)的，一个是瓷(china)的,但是确要两条生产线，显然有资源浪费的嫌疑。现在厂主生产杯子时先不让生产线知道我要产的是玻璃的还是瓷的，而是让它在不知道具体材料的情况下先做它能做的，等到它把模具做好，只需要向其中填充玻璃原料或者瓷原料就可以造出同一形状的具体杯子了。但是很可惜，java并不能new一个抽象的Cup，所以就有了简单工厂模式。原来是Cup cup=new Cup;现在是

SimpleCupFactory.createCup(String cupName),根据cup的名字生产Cup,而createCup返回的是一个实现了 Cup接口或抽象类的具体Cup。简单抽象工厂模式有一个问题，就是当我现在想生产一个同样形状的铁杯时，工厂里并没有定义相应的处理流程，只能更改createCup方法，这就不合理了。我现在只是想生产铁杯，你只要在最后的时候把玻璃原料换成铁的不就行了吗，干嘛还要更改整条生产线呢？于是就有了工厂模式。原来生产线在生产模具的时候还要考虑是为玻璃杯生产的模具还是为铁杯生产的模具，现在它不用管了。CupFactory.createCup()创建Cup.CupFactory是接口或抽象类。实现它的具体子类会创建符合Cup接口的具体Cup。那么现在厂主想要生产水壶(kettle)，用工厂模式就不得不再造一条水壶生产线，能不能在水杯生产线同时生产水壶呢？这就是抽象工厂模式。在原CupFactory中加一个createKettle()方法，用来生产水壶

25.Session有什么重大BUG，微软提出了什么方法加以解决?

是iis中由于有进程回收机制，系统繁忙的话Session会丢失，可以用Sate server或SQL Server数据库的方式存储Session不过这种方式比较慢，而且无法捕获Session的END事件。

26.NET下内存分配机制是什么?

对于值类型的实例，CLR在运行时有两种分配方式：(1) 如果该值类型的实例作为类型中的方法(Method)中的局部变量，则该实例被创建在线程栈上；(2) 如果该值类型的实例作为类型的成员，则该实例作为引用类型(引用类型在GC堆或者LOH上创建)的实例的一部分，被创建在GC堆上。
对于引用类型的实例，CLR在运行时也有两种分配方式：(1) 如果该引用类型的实例的Size<85000Byte，则该实例被创建在GC(Garbage Collection)堆上(当CLR在分配和回收对象时，GC可能会对GC堆进行压缩)；(2) 如果该引用类型的实例的Size>=85000byte，则该实例被创建在LOH(Large Object Heap)上(LOH不会被压缩)。
要注意的是，对于引用对象，他包括了引用和对象实例两部分，实例需要通过对其存储位置的引用来访问，对于private Object o = new Object()，其实可以分解为两句话：
private Object o；o = new Object()；
其中private Object o是定义了对象的引用，也就是记录对象实例的指针，而不是对象本身。这个引用存储于堆栈中，占用4个字节；当没有使用o = new Object()时，引用本身的值为null，也就是不指向任何有效位置；
当o = new Object()后，才真正根据对象的大小，在托管堆中分配空间给对象实例，然后将实例的指针位置赋值给前面的引用。这才完成一个对象的实例化。
值类型如果嵌套在引用类型时，也就是值类型在内联的结构中时，其内存分配是什么样子呢？ 其实很简单，例如类的私有字段如果为值类型，那它作为引用类型实例的一部分，也分配在托管堆上。
引用类型嵌套在值类型时，内存的分配情况为：该引用类型将作为值类型的成员变量，堆栈上将保存该成员的引用，而成员的实际数据还是保存在托管堆中

27.如何提高.NET的性能？

1 使用异步方式调用Web服务和远程对象
只要有可能就要避免在请求的处理过程中对Web服务和远程对象的同步调用，因为它占用的是的ASP.NET 线程池中的工作线程，这将直接影响Web服务器响应其它请求的能力。
2 使用适当的Caching策略来提高性能
3 判断字符串，不要用""比较。
//避免
if(strABC!=null && strABC!="")
{}
//推荐
if(!strABC.IsNullOrEmpty)
{}
4 页面优化
5 用完马上关闭数据库连接

6 尽量使用存储过程，并优化查询语句
7 只读数据访问用SqlDataReader，不要使用DataSet托管堆中

28.如果你有无穷多的水，一个3公升的提桶，一个5公升的提桶，两只提桶形状上下都不均匀，问你如何才能准确称出4公升的水?

3升装满；3升-〉5升（全注入）；3升装满；3升-〉升（剩1升）；5升倒掉;升-〉5 升（注入1升）；3升装满；3升-〉5升；完成（另：可用回溯法编程求解）

29.在一天的24小时之中，时钟的时针、分针和秒针完全重合在一起的时候有几次？都分别是什么时间？你怎样算出来的?

23次，因为分针要转24圈，时针才能转1圈，而分针和时针重合两次之间的间隔显然>1 小时，它们有23次重合机会，每次重合中秒针有一次重合机会，所以是23次重合时间可以对照手表求出，也可列方程求出

30.一张长方形的桌面上放n个一样大小的圆形硬币。这些硬币中可能有一些不完全在桌面内，也可能有一些彼此重叠；当再多放一个硬币而它的圆心在桌面内时，新放的硬币便必定与原先某些硬币重叠。请证明整个桌面可以用4n个硬币完全覆盖

要想让新放的硬币不与原先的硬币重叠，两个硬币的圆心距必须大于直径。也就是说，对于桌面上任意一点，到最近的圆心的距离都小于2，所以，整个桌面可以用n个半径为2的硬币覆盖。

把桌面和硬币的尺度都缩小一倍，那么，长、宽各是原桌面一半的小桌面，就可以用n个半径为1的硬币覆盖。那么，把原来的桌子分割成相等的4块小桌子，那么每块小桌子都可以用n个半径为1的硬币覆盖，因此，整个桌面就可以用4n个半径为1的硬币覆盖。

31.三名传教士和三个野蛮人同在一个小河渡口，渡口上只有一条可容两人的小船。问题的目标是要用这条小船把这六个人全部渡到对岸去，条件是在渡河的过程中，河两岸随时都保持传教士人数不少于野蛮人的人数，否则野蛮人会把处于少数的传教士吃掉。这六个人怎样才能安全渡过去?

1． 一名牧师和一个野蛮人过河；2． 留下野蛮人，牧师返回；3． 两个野蛮人过河；4． 一个野蛮人返回；5． 两名牧师过河；6． 一名牧师和一个野蛮人返回；7． 两名牧师过河；8． 一个野蛮人返回；9． 两个野蛮人过河；10． 一个野蛮人返回；11． 两个野蛮人过河。这里关键的一步是第6步，许多人不能解决此题，就是没有想到这一步。
我的思路：实质到第6步时也只是在重复开始前面的思路，先安排3人，再安排2人的情况。只要抓住“传教士大于等于野蛮人”这个要点分配即可。

32.网站优化：网站运行慢，如何定位问题？发现问题如何解决?

前端：1.减少http的请求，每次发送http请求都会消耗一定的时间。

2.可以使用js缓存，浏览器缓存，能直接从缓存中读取数据，不在请求服务器。

3.使用压缩后的css和js，避免css和js的重复使用，减少 js里面的循环次数。

4.css放在在里面，js放在页面的 底部。因为请求js文件很花费时间，如果放在里面，会导致DOM树需要等待js文件加载完成。

后端：

5.优化SQL，避免使用*查询，使用索引，避免sp中出现大量逻辑的事务，减少in或and和or的查询使用。

6.使用memcache缓存，减少数据库的访问。

7.减少代码的层级接口，避免循环嵌套，优化算法等等。

8.读写分离，负载均衡，面向接口编程，降低耦合性。

33.IEnumerable和IQueryable两接口的区别

（1）所有对于IEnumerable的过滤、排序、分组、聚合等操作，都是在内存中进行的。也就是说把所有的数据不管用不用得到，都从数据库倒入内存中，只是在内存中进行过滤和排序操作，但性能很高，空间换时间，用于操作本地数据源。

（2）所有对于IQueryable的过滤、排序、分组、聚合等操作，只有在数据真正用到的时候才会到数据库中查询，以及只把需要的数据筛选到内存中。Linq to SQL引擎会把表达式树转化成相应的SQL在数据库中执行，这也是Linq的延迟加载核心思想所在，在很复杂的操作下可能比较慢了，时间换空间。

（3）操作本地数据源用IEnumerable，操作远程数据源用IQueryable

34.lock为什么要锁定一个参数，可不可锁定一个值类型？这个参数有什么要求?

锁引用类型，引用类型都是指向一个对象，不可以锁值类型，值类型在装箱时候会产生不同的对象。