1. **如何在.NET中做deep copy？**

* **实现ICloneable接口**

**在引用的每层对象都实现该接口，进行拷贝处理，那最外层就能够深拷贝。**

* **序列化和反序列号**

**序列化和反序列化可以实现深拷贝，binary序列化、json序列化、xml序列化**

* **反射**

**通过反射来进行每层引用的拷贝复制，实现深拷贝**

* MemberwiseClone 浅拷贝

1. throw与throw e的区别？应该用哪一个？

static void Main(string[] args)

{

try

{

Methord2();

}

catch (Exception ex)

{

Console.WriteLine(ex.StackTrace.ToString());

Console.ReadKey();

}

}

private static void Method1()

{

try

{

throw new Exception("test");

}

catch (Exception ex)

{

throw;

}

}

private static void Methord2()

{

try

{

Method1();

}

catch (Exception ex)

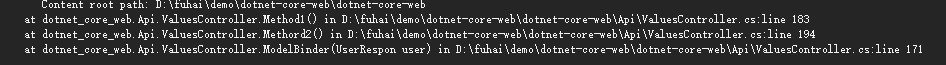
{

throw;

}

}

Methord2中使用throw时的异常堆栈信息



Methord2中使用throw ex时的异常堆栈信息



可以发现当使用在methord2中使用throw时，异常的堆栈是从真正的引发异常的methord1开始的，而使用throw ex后，异常的堆栈是从methord2开始的，也就是他是重新跟踪了堆栈信息，这个是异常最耗时的操作，相当于throw new了。

1. try中的return是否先于finally block调用？

只要进入了try语句，finally最终都会执行，所以即便在try语句里return了，return先执行了，finally最终也还是会被执行，整个方法才结束。

1. out和ref有什么区别？

out参数调用方法前可以只定义，不需要赋值初始化，但是在必须在方法结束前赋值

ref参数调用方法前必须先初始化赋值

out参数是用来实现多值返回，处了方法本身的返回值，out参数也是一个返回值。

ref是用来传参的，是引用传递的方式传参

1. 值传递和引用传递

* 值传递

值传递传递的是一个值，引用类型的值传递也就是传递的引用类型指向的对象在堆中的地址，这时候形参和实参是在栈中的两块地址，形参和实参的值都是同一个对象在堆中的地址，当形参指向另一个地址的对象的时候，实参是不变的。

* 引用传递

引用传递传递的是栈中的地址，这时候形参和实参都是同一块地址，也就是同一个对象，值是同一个对象在堆中的地址，当形参指向另一个对象时，实参也会变

public class ClassBase { }

public class Class1:ClassBase { }

public class Class2 :ClassBase{ }

private static void Main(string[] args)

{

ClassBase value = new Class1();

Methord1(value);

}

private static void Methord1(ClassBase parameter)

{

parameter = new Class2();

}

这里是值传递，调用方法后value为Class1，parameter为Class2

public class ClassBase { }

public class Class1:ClassBase { }

public class Class2 :ClassBase{ }

private static void Main(string[] args)

{

ClassBase value = new Class1();

Methord1(ref value);

}

private static void Methord1(ref ClassBase parameter)

{

parameter = new Class2();

}

这里是引用传递，调用方法后value为Class2，parameter为Class2

1. 在关系型数据库里，referential integrity（参照完整性）是什么意思？

* 域完整性

也就是列的约束，比如数据类型、可空、缺省值、默认值等

* 实体

也就是行的约束，如果主键不能为null，主键唯一等

* 参照

表间的约束，比如外键约束

1. 在GOF设计模式的3类中各选一个设计模式来说明其用途

* 创建型

建造者模式(Builder):使用链式构造器来构造对象，一般写一个类和一个内部类的构造器，使用构造器来构造类的对象。

* 结构型

代理模式(Proxy)：创建一个代理类，实现原有类的接口，依赖原有类，代替原有类来执行操作。比较常用在Aop编程中，比如希望原有类每次方法调用时都写一下日志，就可以使用代理类来实现，在调用原有类的方法前，进行日志记录操作

适配器模式(Adapter):类A只能鸡飞，类B只能狗跳，创建一个适配类C，继承自类A，依赖类B，就能鸡飞狗跳

* 行为型

迭代器模式(Iterator):提供一种方法来顺序访问集合中元素，.Net中用IEnumerable和IEnumerator来实现，

1. 设计模式的使用

工厂模式：虹口平台项目中，需要做国产化改造，本来是使用的sql server，考虑到后期可能需要迁移到达梦数据库中，所以把数据层抽象成接口，然后使用工厂模式，注册工厂类，使用的时候注入工厂类，工厂类根据配置信息来获取实现类(sql server和达梦)

构造者模式：可以链式构造一个对象来，quartz就是使用的构造者模式

1. 在EF中如何定义多对多关系？
2. DateTime.Parse(myString) 这段代码有什么问题？应该怎么写？

使用TryParse/ TryParseExact

1. 为什么catch（Exception）是个不好的写法？

捕获了异常，但是没有获取到异常信息，应该改为catch（Exception ex），最好是更具体的异常catch（NullReferenceException ex）

1. .NET的垃圾回收是怎么管理CLR中的对象的生命周期的？

垃圾回收器是用来自动回收对象内存空间的，一些没有垃圾回收器的语言，需要程序员每次使用完对象后手动去回收对象内存空间，这样会很繁琐，垃圾回收器就是用来解决这一痛点。

Dotnet的对象分为栈对象和堆对象，栈对象在栈结束时就会被销毁，一般时一些值类型的临时变量，而堆对象需要使用垃圾回收器来回收对象内存空间。

使用代的机制来回收，创建对象的时候，会分为3个代：第0代、第1代、第2代。创建对象的时候，每个代都有个初始大小，比如1M。每次创建新对象的时候都是向第0代开始创建，当第0代内存满后，就会执行垃圾回收机制，把没有任何引用的对象给回收销毁，然后把有引用的对象，也就是没被回收活下来的对象放到第1代。然后再次创建对象还是向第0代里面放，如此类推。如果说到了要回收第2代的时候，活下来的对象还是放到第二代，程序继续运行，直到这3代空间都满了，就会尝试把每一代的内存容量扩大。也不是一直扩充，后续扩充后内存还是满了就会抛异常了

大对象总会是在第2代，所以不要创建短时间存活的大对象，这样会让第2代被更频繁的回收。

垃圾回收器没办法回收非托管对象，dotnet创建的对象都是托管在托管对的，但是一些其他资源不是dotnet托管的，比如说文件句柄、网络连接等，这些对象的回收需要使用析构函数来完成。当垃圾回收器发现对象实现了析构函数后，会丢到一个队列里，执行完析构函数才会在下一次垃圾回收时被回收，这样垃圾对象会被提升到下一代，存活时间会更长。

1. 托管对象和非托管对象

Dotnet创建的对象都是在托管堆中分配的内存，称为托管对象，但是一些外部资源是不受dotnet托管的，比如说文件句柄和网络资源等，这些就是非托管对象。

1. Finalize()和Dispose()这2个方法有什么不同。

Finalize是析构函数，在对象被垃圾回收时被调用，一般用来释放非托管资源。

Dispose是一种手动释放资源的方式，需要手动释放托管资源和非托管资源。一般会调用GC.SuppressFinalize(this);通知垃圾回收器，这个对象已经释放了非托管资源了，不用再调用析构函数了。Dispose可以使用using语句，当using块结束时会自动调用该方法释放资源。

1. Equals 、==和ReferenceEquals有什么不同？写出下列代码中Console打印出的结果。

int a = 10;

int b = 10;

int c = a;

Console.WriteLine(a==b);//true

Console.WriteLine(a.Equals(b)); //true

Console.WriteLine(Object.ReferenceEquals(a, b)); //false

Console.WriteLine(Object.ReferenceEquals(a, c)); //false

StringBuilder buildera = new StringBuilder("test");

StringBuilder builderb = new StringBuilder("test");

StringBuilder builderc = buildera;

Console.WriteLine(buildera == builderb); //false

Console.WriteLine(buildera.Equals(builderb)); //true

Console.WriteLine(Object.ReferenceEquals(buildera,builderb)); //false

Console.WriteLine(Object.ReferenceEquals(buildera, builderc)); //true

ReferenceEquals是判断是否同一个对象，对于值类型，判断栈中地址是否一样，对于引用类型，判断堆中地址是否一样

==是栈中值判断，对于值类型和引用类型都一样，判断栈中值是否一样。

Equals是判断内容是否一样，值类型判断栈中内容，引用类型判断堆中内容。

1. object identity(同一性)比较和object equality(相等性)比较有什么不同

同一性判断是否同一个对象，对于值类型是指在栈中地址是否一致，对于引用类型是指在堆中的地址是否一致。

相等性是判断他们的内容是否相等。

1. 请列出常见的缓存方式，并简要概述其优缺点

* MemoryCache

优点：本地缓存，存储和读取速度快，

缺点：占用本地内存、不能集群共享

* Redis

优点：可以集群共享，可以部署在其他机器上，不占用本地内存空间

缺点：读取存储没有本地缓存快

1. 请使用linq语句表达式，查询出 int values={1，2，5，2，3，5，5，3，4，6，3，3}；中出现次数最多的数字。

Values.GroupBy(m=>m).OrderByDescing(m=>m.Count()).FirstOrDefault().Key;

1. 可以采用foreach迭代的类的对象必须满足什么条件？

实现IEnumerable接口

1. 列举C#依赖注入的方式，且相关优劣势说明。

目前dotnet框架的依赖注入只支持构造函数注入

优点：

体现出依赖

脱离容器，这个类也能用

不用担心依赖为null

缺点：

当需要很多服务时，构造函数参数太长

1. async标记的方法返回值有何要求？

Task或者Task<>

1. C#中Params是什么含义？有何用途？

可变参数，参数数量不确定时用，使用上和数组参数类似，参数传递时可以传递数组，也可以拆分开来传递。

* Paramas修饰的可变参数必须是一维数组
* 不允许和ref、out一起使用
* Paramas修饰的参数必须在最后一位
* 若实参是数组则按引用传递，若实参是变量或表达式则按值传递

1. 聚集索引和非聚集索引的区别？

聚集索引的索引键需要保证唯一性，非聚集索引不用

聚集索引有且只有一个，非聚集索引可以有多个

1. 索引的优缺点

优点：建立索引后，在命中索引的情况下，能显著提高查询效率。聚集索引还能维护数据的唯一性。

缺点：索引是占用存储空间的，索引会影响数据增删改效率，因为增删改时需要维护索引。频繁插入的表不建议建立太多索引。

1. 复合索引

单一索引：只在单列上创建索引。

复合索引：在多列上创建索引。

1. 前端闭包的概念

Js局部变量在函数调用时创建，当函数执行完毕后销毁，闭包的作用就是延长这些局部变量的生命周期。全局变量容易混淆和重复命名，而使用闭包就可以把变量划分到不同作用域里，来进行使用。

闭包可以用来在一个函数与一组“私有”变量之间创建关联关系。在给定函数被多次调用的过程中，这些私有变量能够保持其持久性。

1. 储存过程到底是对数据库做了优化，还是没有优化？

做了优化，存储过程会提前编译成执行计划，能反复使用，带参不同也不用重新编译。

Sql 语句相同时可能会缓存有执行计划，但是不同时(查询条件)会重新编译。

1. 存储过程优缺点

优点：执行快，因为已经提前编译好了。减少网络传输，存储过程不用传递大量sql语句，结果也是在整合好的结果。存储过程还能增强安全性，能防止sql注入，因为调用的时候是传参方式，很多的sql注入是通过sql拼接的方式来注入的，传参方式能起到一些阻止作用。

缺点：不好调试，不支持集群，会涉及到业务，当业务有变更时，不仅要修改程序，还要修改存储过程。

1. 什么时候会导致存储过程重新编译

* 存储过程语句中引用的常规表、临时表或视图的架构变化。架构变化包括表的元数据或表上索引的变化
* 常规或临时表的列上的绑定(如默认/规则)变化
* SET选项变化
* 执行计划老化并释放；
* 对sp\_recompile系统存储过程的显式调用。
* 显式使用RECOMPILE子句；

1. EF优缺点

优点：

orm框架，对象关系映射，能把数据库的表状数据映射成编程语言的类对象，使用起来比较方便。Linq查询使用起来也和sql 查询差不多，程序员可以不需要编写sql语句。

跨库，只需要修改配置就能轻松实现数据库切换，可以兼容不同数据库查询语句的差异。

ef core中支持批量操作

缺点：

效率低，不用程序员写sql，但是却需要根据linq表达式生成sql语句，还需要把DataReader的结果转换为编程语言对象，这些都是消耗性能的。

耗内存，查询的数据默认是会进行跟踪的，这个对内存也有消耗。

复杂查询实现不太好，还是需要些sql语句比较好。

更新操作需要先查询，查询后进行跟踪再更新。

需要维护好Context的生命周期，否则容易出错，一般一个上下文一个Context，.net core中是注册DBContext时候是作用域模式注册的，每个请求会创建一个作用域。

1. 接口不能随便给外界访问，认证方式用的哪些？

* Cookie/session方式，有状态

原理：后台写入session时候，会产生一个sessionid，然后把sessionid写入到cookie中，返回给页面，页面每次访问都会携带cookie，后端获取到cookie的内容，根据sessionid获取到session

优点：

缺点：

单点登录:单点登录就是利用cookie/session机制实现的，当通过sso(http:xxxsso.com)服务器登录时，会把sessionid写入domain域为http:xxxsso.com的cookie中。当访问其他服务a时，如果没有登录，那么就发送301状态码给浏览器跳转到sso服务器http:xxxsso.com，这时候sso服务器判断时已经登录了，就会跳转到a携带过来的指定接口(callback),并携带token给服务器a，服务器a使用token后台调用sso服务器，获取登录信息。

* Token方式，无状态

Jwt：一种token标准，会把一些需要使用的信息安装jwt格式来构造一个对象，然后进行签名，再通过xxx编码加密为一个token对象，前端存储这个token，每次访问得时候通过Authorization请求头携带该token，后端能直接根据token解密出完整的用户信息，因为能获取到完整的用户信息，就不用再访问数据库进行权限认证。Jwt设计成AccessToken和RefreshToken模式时，可以通过在payload中添加RefreshToken标记来确认是否时RefreshToken。

优点：可以多端使用，用cookie/session很多非浏览器端会比较不友好。对集群友好，集群部署的时候cookie/session可能需要把session存储到一个能公共访问的存储媒介(数据库/redis等)，而token不用，集群中每个节点都能单独解析出完整的用户信息。

缺点：无法及时过期，要处理过期问题，还需要有状态，那样就没有了无状态的优点了

Oauth：Oauth是利用的(AccessToken和RefreshToken)来授权。 Oauth其实没有认证功能，只是进行授权，让其他的服务能获取到自己服务的一些信息(用户名、头像等)，因此当使用第三方登录的时候，往往还是需要注册新的账户

1. async和await是什么函数，有哪些优点，知道他们的内部机制吗？

Async和await是异步调用，可以很简单的实现异步操作，await的对象必须要实现GetAwaiter方法，该方法返回一个INotifyCompletion 对象。Async和await会被编译器构建成状态机，当await执行后，释放当前线程，当await完成后，变更状态来通知状态机进入后续代码，这样就无需阻塞线程。

1. 什么是进程、什么是线程，有什么区别
2. 进程是一个程序资源分配单位，一个应用程序可以有一到多个进程
3. 线程是进程中的基本执行单元，是操作系统分配CPU时间的基本单位，一个进程可以包含若干个线程，在进程入口执行的第一个线程被视为这个进程的主线程
4. 区别

* 一个程序至少有一个进程,一个进程至少有一个线程
* 线程是比进程更小的单位
* 每个进程是独立的资源空间，互不影响，而进程中的线程公用资源，会互相影响，线程异常时还会影响到进程。、
* 含义不同，进程是资源分配单位，线程是cpu执行单元，是和程序逻辑相关的。

1. 你对多线程有了解吗？

* 为什么用多线程

为了有效利用cpu资源，现在的cpu一般都是多核的，多线程能有效的利用好cpu资源，提高程序的处理效率

* 创建线程方式：

Thread、ThreadPool、Task

* 多线程需要注意事项：

异常处理，数据线程安全问题

1. 并发并行

多个线程一起运行，当多核的时候，确实能做到并发并行操作的，但是当系统是单核的时候，即便有多个线程，但是只有一个cpu资源，所以还是按照线程调度来切换不同线程来执行，这样的操作和单线程是没有区别的，而且由于频繁上下文(cpu时间片结束后，是会把当前线程的状态都要存储起来，然后再加载下个要执行的线程的状态)，所以可能比单线程还慢。

1. 同步、异步、多线程、

同步：同一个线程依次做任务，当需要某些资源时会阻塞等待，知道获取到资源，才继续，这样并不会怎么浪费cpu资源，但是会给用户很不好的体验，例如winform程序，所有的控件操作都是由主线程来执行的，当某个操作阻塞之后，窗体就会无法点击拖拽，在用户看了就像是程序死了。

异步：把资源再拆分为更小的执行单位(任务/携程)，同一个线程做多个任务，当某个任务需要等待某些资源时候，会切换到其他任务进行执行，当获取到资源后，又能继续执行原有的任务。

拟人化：

同步：煮饭，等待饭煮好了，再切菜，煮汤，等到汤煮好了再炒菜

异步：煮饭，设置提醒，就去切菜，然后煲汤，设置定时，准备去炒菜时候，收到提醒饭好了，去看一下，然后去炒菜，汤提示已经做好了，去看一下就乘汤上桌。

多线程：就是两个人(多个人)一起干

1. 多线程就一定快吗？

过犹不及，需要看系统内核数量，内核数量多可以稍微多建点线程，否则不建议建太多线程，因为线程切换时候需要切换上下文，开销也是很大的，在内核有限的情况下，如果线程太多，上下文切换回很频繁，相反如果线程少，可能多个时间片都是同一个线程执行，这样就减少了上下文的切换。

1. 多线程需要注意的问题

* 异常处理，多线程需要注意好每个线程的异常处理，否则容易出现UnhandleException.
* 程序可读性问题，多线程的情况下程序可读性比较差
* 线程安全问题，需要确保一些共享的数据的线程安全问题，可以加锁、或者使用线程安全类。
* 上下文问题，一些框架下需要注意多线程情况下的上下文问题，比如说mvc的httpcontext和winform的上下文，

1. 程序第一次发布的时候访问很慢，第二次就快了是什么原因？

Iis应用程序池在访问的时候才启动，还有就是一些网络资源在使用的时候才连接，比如说数据库连接，在长时间不访问时再次访问也会变慢，这是因为iis应用程序池回收机制导致的。

解决方案：

IIS闲置超时设置为0

数据库可以提前建立连接

1. .Net Core中如何捕获全局异常？

ExceptionFilter

1. Memcached、MogoDB、Redis的区别以及优缺点？
2. 什么是微服务、搭建微服务架构的全过程、微服务如何让数据库保持一致性？

把业务拆分为很多个独立的小服务(业务和数据库都是独立的),服务间通过rpc进行通讯，往往要某个功能需要从多个服务中进行调用整合。

微服务需要使用分布式事物来保持数据的一致性。

1. 不管是用全文搜索引擎还是缓存保存数据，如何更新最新的数据，保证数据的一致性？

保持缓存数据和数据库中数据的一致性？

需要先锁定缓存，然后再更新数据库和缓存。(?)

加锁可能不太好，对于高并发的情况下，应该先更新数据库然后再删除缓存，这样不能保持强一致性，但是缓存并不需要保持强一致性，因为在高并发下代价有点大。

全完搜索引擎好像是不具备实时性的，所以一般不会用来做实时性数据的存储和索引。

1. 什么是Devops（CI、CD），如何实现的？
2. Redis雪崩、穿透问题
3. 字典Dictionary的内部机制是如何实现的？

字典内部会建立一个哈希槽，添加元素的时候会计算key的哈希值，根据哈希值找到哈希槽的位置，然后进行存储，因为会有哈希碰撞，也就是两个不同的key找到的哈希槽一样，所以每个哈希槽里的数据都是用链表来存储的。

1. 数据库锁有哪几种？
2. 数据库的隔离级别以及每种隔离级别的加锁情况

事物的隔离级别是用锁来实现数据的不同安全级别。

* read uncommitted

写的时候会加排他锁(意向排他锁？)，读的时候不加锁，所以能读取到脏数据。

1. 常见的算法和数据结构
2. 请写一个简单的冒泡排序程序出来？
3. 压力测试用的是什么工具，单台服务器最大的QPS是多少？
4. 自动化测试工具有用过吗，请举例？
5. 如何检测数据库中某条执行很慢的查询语句？

查看执行计划，看一下是否命中索引，如果没有就修改查询条件，尽量要命中索引。

如果有嵌套查询，优化一下，嵌套查询先用查询条件缩小数据集，这样能提高查询效率。

1. 你是如何理解敏捷开发模式的？敏捷开发中应该存在哪些角色？
2. 什么是Restful？
3. cookie和session的区别？还有其他保存数据的方式吗？
4. char、varchar、nvarchar的区别
5. 谈谈重载和重写的作用和区别
6. 说一下c#中索引器的实现，是不是只有整型数字能作为索引值？是否允许多个索引器参数？
7. 什么是MVC，什么是MVVM？
8. 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？

* Property
* Attribute

公共语言运行时允许你添加类似关键字的描述声明，叫做attributes, 它对程序中的元素进行标注，如类型、字段、方法和属性等。Attributes和Microsoft .NET Framework文件的元数据保存在一起，可以用来向运行时描述你的代码，或者在程序运行的时候影响应用程序的行为

1. C#中的LINQ

LINQ语言集成查询，可以使用c#很方便的从数据源中检索数据

1. Lambda表达式
2. Try{}Catch{}Finally
3. string str=null 与 string str="" 的区别？
4. 在.Net中所有可序列化的类都被标记为\_，为什么？

[Serializable]

1. 用sealed修饰的类有什么特点？

不能被继承

1. ADO.net中常用五个主要对象是什么？分别描述一下

Connection：主要是开启程序和数据库之间的连接。没有利用连接对象将数据库打开，是无法从数据库中取得数据的。Close和Dispose的区别，Close以后还可以Open，Dispose以后则不能再用。

Command：主要可以用来对数据库发出一些指令，例如可以对数据库下达查询、新增、修改、删除数据等指令，以及调用存在数据库中的存储过程等。这个对象是架构在Connection 对象上，也就是Command 对象是透过连接到数据源。

DataAdapter：主要是在数据源以及DataSet 之间执行数据传输的工作，它可以透过Command 对象下达命令后，并将取得的数据放入DataSet 对象中。这个对象是架构在Command对象上，并提供了许多配合DataSet 使用的功能。

DataSet：这个对象可以视为一个暂存区（Cache），可以把从数据库中所查询到的数据保留起来，甚至可以将整个数据库显示出来，DataSet是放在内存中的。DataSet 的能力不只是可以储存多个Table 而已，还可以透过DataAdapter对象取得一些例如主键等的数据表结构，并可以记录数据表间的关联。DataSet 对象可以说是ADO.NET 中重量级的对象，这个对象架构在DataAdapter对象上，本身不具备和数据源沟通的能力；也就是说我们是将DataAdapter对象当做DataSet 对象以及数据源间传输数据的桥梁。DataSet包含若干DataTable、DataTableTable包含若干DataRow。

DataReader：当我们只需要循序的读取数据而不需要其它操作时，可以使用DataReader 对象。DataReader对象只是一次一笔向下循序的读取数据源中的数据，这些数据是存在数据库服务器中的，而不是一次性加载到程序的内存中的，只能（通过游标）读取当前行的数据，而且这些数据是只读的，并不允许作其它的操作。因为DataReader 在读取数据的时候限制了每次只读取一笔，而且只能只读，所以使用起来不但节省资源而且效率很好。使用DataReader 对象除了效率较好之外，因为不用把数据全部传回，故可以降低网络的负载。

ADO.NET 使用Connection 对象来连接数据库，使用Command 或DataAdapter对象来执行SQL语句，并将执行的结果返回给DataReader 或 DataAdapter ,然后再使用取得的DataReader 或DataAdapter 对象操作数据结果。

1. 什么是强类型语言

“强类型语言也称为强类型定义语言。是一种总是强制类型定义的语言，要求变量的使用要严格符合定义，所有变量都必须先定义后使用

1. 构造器Constructor是否可被override?

不可被override，可被overload

1. 是否可以继承String类?

不行，String类被sealed修饰，不能被继承

1. 谈谈final,finally, finalize的区别。

Final是用来修改变量的，指不可变更

Finally是用在异常处理try{}catch{}中的，无论是否异常，都会在语句块结束前执行finally。

Finalize是析构函数~ClassName(){} 就是ClassName类的析构函数

1. 双if+lock模式

当需要在加锁前进行if判断时，由于if+锁是两个步骤了，不是原子操作，将会导致多个线程都进入了if块里，虽然之后一个线程获取到锁，但是还会有其他资源在请求锁，所以后续还需要再使用一个if来进行判断，避免多个线程相继获取到锁，进行多次操作。比较典型的例子就是单例模式。

第一个if可能是很多线程一起进入的，其他线程都阻塞在锁资源的获取上，然后判断需要加锁，这时候只有一个线程获取到锁，然后再if判断构建单例，这时候单例已经创建好了，就释放锁，其他线程就又有一个进入到锁，这时候之前的线程已经创建单例了，所以判断已经有单例，就不用再创建了。

1. Volatile

该修饰词保证变量的原子性（存疑），可见性和有序性。

该修饰词主要使用其可见性，对4字节及以下类型的赋值操作都是原子操作，但是由于有缓存的作用，所以可能cpu操作的是缓存，最后才存储到内存中，而多核的时候，其他的cpu的缓存可能还是旧的缓存，所以看到的数据可能还是老黄历，Volatile的可见性就能解决的这个问题，使用 volatile 修饰共享变量后，每个线程要操作变量时会从主内存中将变量拷贝到本地内存作为副本，当线程操作变量副本并写回主内存后，会通过 CPU 总线嗅探机制告知其他线程该变量副本已经失效，需要重新从主内存中读取。

Volatile虽然保证了可见性，赋值操作也是原子性的，但是一些自增等操作却不能保证原子性，自增操作有几个步骤，需要先从缓存中读取数据到cpu，再进行相加操作，然后在保存到缓存，通过嗅探功能让其他cpu缓存失效。假设变量值是1，有两个线程把变量读取到cpu，线程1进行了自增操作，存储到主内存，让其他缓存失效，但是线程2已经把值读到cpu中了，不用从缓存读取数据，此时主内存变量值是2，线程2中cpu变量值还是1，进行自增后结果是2，存储到主内存，这样1进行了两次自增，结果只是2.

1. 单例模式

* 恶汉模式
* 双if+锁
* 线程安全类
* 依赖注入(Ioc容器)

1. 我们都知道一个类可以有多个构造函数，并且C#会在我们创建类的时候默认的提供一个无参的构造函数，当我实现了另外一个有参数的构造函数的时候，我还能调用无参数的构造函数吗？请说明原因。

不能，因为一旦你实现了一个构造函数，C#就不会再提供默认的构造函数了，所以需要自己手动写入一个无参数的构造函数。

1. 类的私有成员会被子类继承吗？请说明原因。

不能，父类的私有成员只能父类使用，并不能被继承。

1. C#中类是否支持多继承？请说明原因

不支持，需要用接口来实现多继承。

1. const和readonly有什么区别

const是编译时常量(编译时会把const修饰的常量名替换为对应的值)，readonly是运行时常量，理论上也还是个变量，只是不能修改。

const不能和static一起使用，readonly可以和static一起使用

1. webapi接口访问权限的问题，JWT实现？
2. SQL Server数据库操作的原子性问题?

锁、事务隔离级别

1. using关键字几种用法？

* 引用命名空间和别名
* IDisposable接口

1. 关于秒杀活动的并发问题
2. 详诉下内存中的堆和栈。

堆和栈是应用程序的两块内存空间

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。 存放在栈中时要管存储顺序，保持着先进后出的原则，他是一片连续的内存域，有系统自动分配和维护；

堆：是无序的，他是一片不连续的内存域，有用户自己来控制和释放，如果用户自己不释放的话，当内存达到一定的特定值时，通过垃圾回收器(GC) 来回收。

栈内存无需我们管理，也不受 GC 管理。当栈顶元素使用完毕，立马释放。而堆则需要 GC 清理。

使用引用类型的时候，一般是对指针进行的操作而非引用类型对象本身。但是值类型则操作其本身

1. 用SQL进行行转列
2. Ajax调用的时候，get、post有什么区别？

两个含义不一样，一个是请求资源，一个是更新资源

url长度限制不一样

get不能带body数据，即便带了，服务端也是不认的，只能url传参

post可以带body数据

1. 返回类型 ActionrResult 的方法可以重载吗？ 返回的类型有哪几种？
2. .net 怎么实现事务？

Sql语句: Begin tran

Dbo: BeginTransaction

TransactionScope

1. 数据库优化方面有什么了解？

* 索引创建，查询的时候尽量使用能命中索引的查询条件，in 等这些语句是不会命中索引的,可以查看执行计划来看是否命中了索引。
* 嵌套查询的时候可以稍微缩小查询范围，在嵌套查询里使用查询条件。
* 覆盖索引，一些经常使用到的字段，可以在创建查询索引时创建覆盖索引，create index on() include