浅克隆和深克隆？深克隆的方法，（逐层实现克隆，序列化实现）

浅克隆指的是克隆的时候仅仅克隆对象本身（对象的基本变量），而不拷贝对象的引用变量。

深克隆指的是克隆的时候不仅克隆对象本身，而且拷贝对象包含的引用指向的对象。

==和equals的区别

基本类型的话==判断是实际值，引用类型的话==判断的指向的对象是不是同一个

equals需要看对应的类有没有复写，复写的话需要根据具体逻辑判断，没有复写的话实际上判断的还是指向的对象是不是同一个。例如string的equals方法，就是判断字符串的字符是不是都相等。

自动装箱拆箱机制

装箱就是自动将基本数据类型转换为包装器类型；拆箱就是 自动将包装器类型转换为基本数据类型。

简述一下java的异常体系，什么是受检异常，什么是运行时异常；封装一个API的时候什么情况下抛出异常。

string常量池的考核，可变类和不可变类（intern）

String a =new String(“ning”)；如果常量池没有ning，需要在常量池加入ning，然后在堆上创建ning对象。Intern, 如果常量池中存在当前字符串, 就会直接返回当前字符串. 如果常量池中没有此字符串, 会将此字符串放入常量池中后, 再返回.

不可变类：当你获得这个类的一个实例引用时，你不可以改变这个实例的内容。不可变类的实例一但创建，其内在成员变量的值就不能被修改

是否发现过jdk的bug或者踩过jdk的一些坑（发现Object对象是不能够序列化的，java是值传递，还有一个特殊是String字符串是不可变类，对它的任何修改都是新的对象）

Hash冲突，有哪几种常见的解决方法（开放地址法，链地址法）？ HashMap在高并发情况下不当使用，可能会导致什么样极端情况（扩容后循环链表的发生）

HashMap/ConcurrentHashMap的数据结构实现，扩容机制---

初始长度为16，扩容因子是0.75，entry。

hashtable和hashmap之间对比，为什么hashmap不是线程安全的。HashTable的方法上有锁

hash,hash碰撞攻击，hash表的遍历的时间复杂度，如何将其降低到o(n)的时间复杂度。以及如何实现一个LRU的机制的cache（linkedHashMap的变体），0/1切换和渐进式rehash。

HashMap对于hash碰撞用链地址解决，如果没有hash冲突也就是每一个链表的长度都为1的话，那么遍历的时间复杂度是O（n）;保证没有hash冲突是解决问题的关键，怎么解决就需要一个足够好的hash算法，保证hash的分布是均衡的。

ArrayList和LinkedList的区别，数据结构实现,区别

ArrayList是对数组进行操作的E[]，LinkedList是对双向链表进行操作的，Entry<E> header。

header的pre指向最新的元素，next指向第一个插入的元素

如何拷贝数组，怎样效率最高？【clone,sysytem.arraycopy，**Arrays.copyOf()**】为什么

http协议，在设计http接口得时候需要考虑什么问题？幂等，超时等

幂等性： 在网络请求中，接口幂等性指的是万剑归宗，即该请求不论执行多少次，最终达到的结果总是相同的。

超时：

常见的通信协议，http，tcp，udp，FTP

HTTP 各个 method 的语义

HTTP/2

HTTPS 的实现等

编译装载链接初始化的过程

类的生命周期

加载-验证-准备-解析-初始化-使用-卸载

多态的实现

重写（继承，实际类型，运行时确定，动态分派）和重载（静态类型，编译期就知道运行的版本）

对OOP的理解

面向对象的程序设计，一切基于对象，通过创建对象，执行对象的某个行为来执行一定的代码逻辑。

面向对象编程可以说成是面向接口编程或者是面向超类编程，这样可以充分利用java多态的特性，达到代码的容易维护的特点。

动态类型语言和静态类型语言的区别，强类型语言与弱类型语言

静态语言需要指定变量的类型，用以提供编译器各种检查的功能。Java就是静态语音，编译器就能发现一些编译错误。Java，c++

动态语言，程序可以在运行时改变结构，例如新增和删除函数，而且类型的检查是在运行时判断。强/弱类型是指类型检查的严格程度的。Javascript

有没有使用过其他范式的编程语言，如函数式的语言，动态语言等。他们和Java比有何优劣。

使用过javascript，c++。C++没有gc机制，需要手动写析构函数，优点的话比java更快吧。Javascript没有代码编译过程，不知道语法上的错误。优点，灵活

常见的代码坏味道有哪些。重构的手法

重复代码过多，代码的耦合多，依赖具体的对象类型，而不是依赖接口。一个类的职责过多（例如这个类即干了业务的活，又提供了一些工具方法的处理），命名不规范，方法体太大，使用if语句过多。

git 的使用。git 的简单工作原理

git add，

git commit

git pull

git push

git branch

git checkout branch

Git是一套内容寻址（content-addressable）文件系统

静态代理和动态代理

静态代理是通过代码的代理模式实现的，通过组合让代理对象持有被代理对象，主业务通过代理对象去访问实际对象。

动态代理包括jdk动态代理和cglib动态代理，spring AOP基于此实现，mabatis也大量用了jdk动态代理，sqlSessionTemplate代理了sqlSession。事务用的是cglib动态代理。MapperFactionBean，getObject实际上就是返回这个接口的代理类

java8的新特性

Lambda 表达式

默认方法  默认方法就是一个在接口里面有了一个实现的方法

方法引用

maven中如何规避，解决依赖冲突

解决冲突，使用exclusion标签将冲突的JAR排除。本身的依赖jar包，和依赖jar包依赖的jar包发生了版本冲突。可以在这个依赖里面剔除冲突的jar包

字符串相似度匹配（动态规划）

打麻将游戏，打一张牌之后服务端算法处理（指针追赶）。

链表翻转。搞定

找数游戏。从一个数组中找出满足符合条件的元素:它大于或等于前面所有元素,小于或等于后面所有元素。搞定

一摞烙饼的排序

买书问题

求二进制数中1的个数

寻找发帖“水王”

寻找最大的K个数。搞定

最大公约数问题，搞定

找符合条件的整数

寻找数组中的最大值和最小值，可以用排序（手鲁或者用Arrays.sort），也可以使用2次遍历分别输出最大的值最小的值，复杂度是O（n）。搞定，手鲁用冒泡排序

快速寻找满足条件的两个数

子数组的最大乘积 搞定

求数组的子数组之和的最大值，搞定

子数组之和的最大值（二维）

求数组中最长递增子序列

数组循环移位

数组分割。搞定，使用Arrays.sort排序

区间重合判断

字符串移位包含的问题

电话号码对应英语单词问题，搞定

计算字符串的相似度

从无头单链表中删除节点

最短摘要的生成

编程判断两个链表是否相交

队列中取最大值操作问题

求二叉树中节点的最大距离，搞定

分层遍历二叉树，搞定

偶数个数只有两个不是成对出现的，快速定位到这两个不成对的数

响应码 4XX与5XX的含义(客户端异常与服务器异常), 400,401,403,404,405的含义(请求参数异常, 未授权, 禁止访问, 页面找不到, 请求方法不支持)

服务器跳转与重定向的区别(服务器跳转是指服务器内部跳转, request是同一个, 重定向让浏览器跳转到新地址访问, request不是同一个了, 重定向http code 是301、302)

浏览器怎么知道当前登陆者是谁? (cookie) cookie存在什么位置？(http header)

restful是什么? (一套基于资源定位的设计风格, 关键是答出资源两字),

设计一个restful风格的 url (全是名词, 如果有复数以及版本号 加分)

GET /articles/2， 客户端发出的数据操作指令都是"动词 + 宾语"的结构

POST方法与PUT方法的区别 （幂等, POST只能新建, PUT可以新建/修改）

HTTPS和HTTP的对比，如何加密和认证，建立https通信过程？

HTTPS：是以安全为目标的HTTP通道，简单讲是HTTP的安全版，即HTTP下加入SSL层，HTTPS的安全基础是SSL，因此加密的详细内容就需要SSL。

首先服务器端用非对称加密（RSA）产生公钥和私钥。然后把公钥交给数字证书，并进行包装发给客户端，当公钥到达客户端之后，客户端的TLS首先验证公钥是否有效（颁发机构，公钥有效期，CA数字签名），若存在问题则弹出警告框，提示证书存在问题。若证书没有问题，则客户端会用对称加密产生一个秘钥并用公钥加密后发送给服务器，这个秘钥就是以后用来通信的秘钥。这样服务器端收到公钥加密的秘钥就用自己的私钥解开公钥从而获得秘钥。这样客户端和服务器都获得了秘钥，信息交流相对是安全的

客户端在使用HTTPS方式与Web服务器通信时有以下几个步骤，如图所示。

　　（1）客户使用https的URL访问Web服务器，要求与Web服务器建立SSL连接。

　　（2）Web服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。

　　（3）客户端的浏览器与Web服务器开始协商SSL连接的安全等级，也就是信息加密的等级。

　　（4）客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。

　　（5）Web服务器利用自己的私钥解密出会话密钥。

　　（6）Web服务器利用会话密钥加密与客户端之间的通信。

针对某个熟悉的开源框架里面的设计模式都有哪些？jdk 里面的设计模式如Java io的装饰者模式。

Spring中的设计模式：模板模式，mvc模式，适配器模式，单例模式，装饰模式，观察者模式，工厂模式，代理模式，策略模式。

Jdk里面的设计模式：观察者模式，迭代器模式，代理模式，建造者模式

画出一个类图：从字典中选出符合以下三个条件的字符串，1）包含a或者b或者cd 2）以end结尾3）长度大于7

举例Java线程安全的实现方法，原理、区别

synchronized，ReentrantLock。Synchronnized中的锁是非公平的，ReentrantLock默认情况下也非公平的，但是可以实现公平锁。用法不一样。ReentrantLock是通过AQS实现的，AQS的底层是CAS实现的。

原理，synchronized关键字经过编译后，会再同步块的前后分别形成monitorenter和monitorexit这两个字节码指定。在执行monitorenter之前，首先尝试获取对象的锁，如果和这个对象没有锁，或者当前线程已经有这个对象的锁，把锁的计数器加1，相应的执行monitorexit指令是会将锁计数器减1，当计数器为0的时候，锁就会被释放。如果获取对象锁失败，则当前线程就要阻塞等待，直到对象锁被另一个线程释放为止。

线程同步的方法

互斥同步，synchronized，reetrantLock，readWriteLock

非阻塞同步，先进行操作，如果没有共享数据冲突，就操作成功了，如果有共享数据冲突，采用一定的补偿措施（不断重试，直到成功）。CAS（compare and swap）。

线程池原理，使用，调优

ThreadLocal 作用和实现机制

ThreadLocal作用，将一些数据和本线程绑定，实现线程特有的数据。

实现机制，实际上线程里面有一个map，将ThreadLocal实例和Value关联起来。

作用：事务里面会将hashMap和ThreadLocal关联起来，map里面存放数据源和connection，多数据源会把用户和ThreadLocal关联起来。

Lock与synchronized 的区别,锁的粒度：类级别和对象级别。

Synchronized是java内置的机制实现的，使用的时候不需要手动释放。

Lock一个接口，实现这接口的类，可以用来控制线程并发。使用的时候需要手动加锁，手动释放。

读写锁,CountDownLatch等的使用，

读写锁，将一个资源的操作分为读和写，可以实现一个资源多次读取，但是写的锁是互斥的。

一个线程占用了读锁，其他线程申请读锁可以，申请写锁需要读锁释放掉。

一个线程占用了写锁，其他线程申请读锁和写锁都不行。

CountDownLatch，，它允许一个或多个线程一直等待，直到其他线程的操作执行完后再执行。

例如主线程开启了new了CountDownLatch，然后启动线程池执行多个线程，每个线程都被传入了这个CountDownLatch对象。主线程执行，await（）将会被等待，子线程执行完了之后执行countDown（）计数器减1。直到计数器为0的时候，主线程不在被阻塞

ConcurrentHashMap get是否需要加锁，是如何保证高并发情况下的性能的？(分段锁，hashEntry的final字段结构，segment中volidate类型的count字段的设计)。

不需要加锁

分段锁，segment是继承了Lock，hashEntry的final字段结构，segment中volidate类型的count字段的设计

分布式锁的实现方式（redis,zk）

可以在数据库层面是实现，例如通过悲观锁（对记录for update的方式触发），多线程中只会有一个加锁成功，后面没加锁的会被阻塞直到锁释放。

Redis使用的是setnx，expire，get，getsetnx

代码实现一个读取缓存，缓存失效透传后端的功能。考擦点：如何避免高并发同一key多个同时透传后端，缓存击穿的问题 putifabsent等CAS操作是否会用

java 之DelayQueue，SynchronousQueue原理

SynchronousQueue一个线程执行了put操作后，另一个线程必须去执行

普通Java中的ReentrantLock ，countdownLatch实现原理（如何保证高性能），和synchronized 之间的区别。什么时候选择去用它

ReentrantLock，countdownLatch底层都是AQS，用法不一样，但都是通过java api层面的，synchronized是原生语法上面的互斥锁。Java1.5之前推荐用ReentrantLock，因为更快，还能实现公平锁，1.6之后就没有性能上的差距了。countdownLatch，可以通过计数器计数的方式，保证一个线程阻塞直到其他线程执行countdown导致计数器值为0

volitale的作用，为什么要指令重排序

作用1，被volitate修饰的变量，对所有线程就有可见性，可以理解为，一个线程对此变量的修改会被立马反映到其他线程中。

禁止指令重排序。指令重排序可能在并发中导致很多问题，jvm对程序执行多的优化吧。

AQS的理解，如何运用

像是ReentrantLock，读写锁，countdownLatch，SynchronousQueue等底层都用到了AQS，AQS的底层原理是unasfe包下面的cas操作。AQS维护了一个队列，没有获取锁的线程会加入到这个队列里面阻塞，直到其他线程将锁释放。

还了解concurrent包下的哪些类，使用场景是什么

ConcurrentHashMap，可以用作多线程场景下的缓存

死锁如何产生，如何避免。

避免，加锁顺序，加锁时限，死锁检测。

典型的是线程A持有R1锁，等待获取R2锁，线程B持有R2锁，等待获取R1锁。

诊断，jstack和jconsole诊断

如何理解Java内存模型，及内存可见性

java内存模型规定了所有的变量都存储在主内存中，每条线程有自己的工作内存，工作内存中保存了该线程使用到的变量的主内存副本拷贝，线程只能操作工作内存中的变量，不能直接读写主内存中的变量。不同的线程之间无法访问对方工作内存中的变量。

内存可见性，通过加锁保证，volitale。

java的类加载器体系结构和双亲委托机制

启动类加载器-扩展类加载器-系统类加载器-用户自定义类加载器。

一个类加载器收到了类加载的请求，他首先不会自己去而是委托父类去加载，每一个层的逻辑都是如此，这样类加载请求最终会到达启动类加载器，只有加载，当父类加载器在搜索范围内没有找到这个类的时候，子加载器才会尝试自己去加载。

Jvm堆内存区，有两个S区有什么作用，谈下java对象在JVM的生命周期

如果这个对象被类变量引用，这可能存活时间很长。如果被实例变量引用，这随着实例变量的存活而存活

如果被方法变量引用，这可能函数结束后就被杀死

有遇到过OOM么，什么情况，怎么发现的，怎么查原因的

遇到过栈溢出，stackoverflowerror。在spring创建dao层bean的时候，栈内存设置的太小了。看了debug的执行栈发现执行栈存在递归调用，getBean-createBean-populateBean-getBean

线上JVM参数都怎么设置的，为什么要这么设置，什么时候需要调优。

young gc和full gc触发条件

新生代空间分配不足的时候会触发yong gc。Yong gc后存活的对象放入到幸存者区，如果幸存者区不够的话，会通过空间担保往老年区转移，如果空间担保失败，就会发生full gc

垃圾回收算法及垃圾回收器的选择

标记清除算法，复制算法，标记整理算法，分代收集算法。

一般的都是1.5的是，serial+cms，或者parallel new +cms

jvm内存结构

方法区 虚拟机栈 本地方法栈

堆 程序计数器

FGC的可能原因，以及解决办法。

分配担保失败，

日常调优的技巧（常见的性能优化策略）

如何判断对象存活

可达性算法分析，类常量，类变量，虚拟机栈引用对象，本地方法栈引用对象

常见让java服务器挂掉的内存溢出(栈,堆, 持久态)以及原因

jvm监控工具，

jconsole，visualVm，Jstack，Jmap

CMS工作原理，相比Paralle GC的优劣

标记清理算法会造成空间碎片过多，并行，多线程，用户程序不会处于等待状态。

Parallel 并发，多线程，但是用户线程仍处于等待状态；

有了CMS，为什么还需要G1 GC

采用标记整理算法，空间整合；可预测停顿

JVM常用参数适用及理解

常见的JVM字节码及含义等

数据库事务+事务传播机制

事务特性（4种）：acid

原子性：强调事务的不可分割。一致性：事务的执行的前后数据的完整性保持一致。

隔离性：一个事务执行的过程中，不应该受到其他事务的干扰。

持久性：事务一旦结束，数据就持久到数据库。

事务隔离级别（4种）：

read uncommitted:脏读，不可重复读，虚读都有可能发生

read commited：避免脏读，不可重复读，虚读都有可能发生

repeatable read: 避免脏读和不可重复读，但是虚读有可能否发生

serializable：避免以上所有问题

事务传播机制：

在当前事务开始之前，如果有一个事务上线文已经存在，这是当前事务的传播行为。

PROPAGATION\_REQUIRED -- 支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

PROPAGATION\_SUPPORTS -- 支持当前事务，如果当前没有事务，就以非事务方式执行。

PROPAGATION\_MANDATORY -- 支持当前事务，如果当前没有事务，就抛出异常。

PROPAGATION\_REQUIRES\_NEW -- 新建事务，如果当前存在事务，把当前事务挂起。

PROPAGATION\_NOT\_SUPPORTED -- 以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

PROPAGATION\_NEVER -- 以非事务方式执行，如果当前存在事务，则抛出异常。

PROPAGATION\_NESTED -- 如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则进行与PROPAGATION\_REQUIRED类似的操作

MySQL索引使用的数据结构，B+树

MyISAM,InnoDB有什么区别,各有什么特点

慢查询如何分析

看执行计划，

使用事务的时候应该注意什么？比如事务中不应该有执行时间过长的业务流程。

如果想通过运行时异常来控制事务的提交或者回滚的话，不要使用将异常catch住然后处理了，可以catch住抛出去，或者直接不try，catch。

事务是线程范围的，若干你在一个事务里面开启了另一个线程，这新起的事务不会和原来的事务同一个事务。

还有要考虑事务的传播机制，是不是适合Required这个，如果你记录日志和主业务是同一个事务，日志记录失败导致主业务回滚，显然不合适。这种情况最好新开一个事务。

充值和提现，事务如何保证的，或者以银行A/B用户之间转账的例子将一下，这个场景中事务是如何保证的。 (两阶段提交,分布式事务 2.本地事务,本地落地,补偿发送)

分布式事务（实时一致性【事务补偿机制】还是【基于消息的最终一致性】最终一致性）

联合索引

建立联合索引的时候应该将最常用的或者是筛选力度最大的放在前面。使用的时候带上最左边的列一起查询，否则 联合索引不起作用。

为什么使用B+树做索引(磁盘IO+范围搜索)

因为是多叉树会比二叉树的搜索深度小，B+树的所有的数据都在叶子节点上，遍历起来很方便，不存贮地址树的一个节点对比B树的一个节点更小

事务的4个隔离级别？分别存在哪些问题？举例说明

read uncommitted:脏读，不可重复读，虚读都有可能发生

read commited：避免脏读，不可重复读，虚读都有可能发生

repeatable read: 避免脏读和不可重复读，但是虚读有可能否发生

serializable：避免以上所有问题

有哪些锁？如何解决幻读？

悲观锁（for update触发），乐观锁（针对读比较多的情况）。

数据库的隔离级别串行化

MVCC机制

MVCC机制

快照机制，一致性读是通过 MVCC 为查询提供了一个基于时间的点的快照。这个查询只能看到在自己之前提交的数据，而在查询开始之后提交的数据是不可以看到的

分库分表的思路

sql优化。

是不是没有加索引，没加的话记得加索引。是不是查询结果集太多了。

先看执行计划分析，具体的看看有没有走索引。最后有针对性行的优化sql，like，in，<>,优化掉

jvm优化，可以讲下曾经的遇到的问题以及解决方案。

接口调优。

超时，错误码规范，

系统调优。

问题排查的方法论

1. 复现法，能复现的就复现，问题多复现了，排查起来很简单，看本机日志就行了
2. 直接看服务器日志，更具服务器日志审查代码解决。如果服务器上面没有有用的信息，可以多加一些打印日志的代码，升级。
3. 如果对代码很熟悉的话，或者数据库里面的日志表维护的好的话，就可以看日志表。或者看错误数据就能大致定位。
4. 最关键的就是，一点一点排查，要有耐心，排除了所有的不可能之后，最后那个在不可能也是答案。

设计一个微信红包抢红包的系统。

大key和热key的解决办法

跨机房数据读写和同步的问题的解决方案。

接口流速控制自我保护的实现方式，在一个60s的时间序列上如何保接口的调用流速是均衡的。

设计一个实时流量统计系统。

设计一个直播系统。

设计一个电商秒杀系统

RPC：

你理解的rpc是怎样的。

远程过程调用，例如webservice接口，http接口都算是rpc，java的RMI机制。

http接口调用的话，通过Apache的httpClient或者使用spring的RestTemplate调用，具体的建立连接，客户端把数据（httpHeader，body）发送给服务端，服务端根据具体的url找到对应的hander（spring架构），处理逻辑，结果返回给客户端。

Dubbo调用过程，订阅了某个服务的客户端，发起请求的时候，会被代理类拦截，然后讲message包装成invocation，发送到服务端，服务端根据message找到对应的注册在zk上面的执行类，走具体的执行，执行完了后，将结果包装成RPCResult，通过NIO发送给客户端。

rpc异步调用的是如何实现的？

* 同步：创建DefaultFuture之后，直接get阻塞等待；
* 异步：创建DefaultFuture之后，使用FutureAdapter进行包装，之后设置到当前线程的RpcContext中；后续用户在合适的时候自己从RpcContext获取future，之后get。

rpc调用的可靠性方面是如何保障的？网络超时，闪断，对方进程僵死，处理缓慢情况下是如何保证协议可靠的？

Dubbo的超时和重连机制。

心跳机制。是否懂的ping-pong的双向心跳机制。

心跳, 即在 TCP 长连接中, 客户端和服务器之间定期发送的一种特殊的数据包, 通知对方自己还在线, 以确保 TCP 连接的有效性.

重连机制。断掉之后是立即重连么？为什么？（服务端句柄释放需要一定的时间）

客户端和服务端断掉连接之后的，哪些还没有发送玩得消息（应该还在缓存中）是如何处理的？是否有设置缓存上限？

黏包拆包问题有没有遇到，如何解决的？

服务端异常是如何透传到客户端的？

MQ

mq的常用场景

:**解耦、异步、削峰**

如果让你设计一个MQ,你需要考虑什么问题（rpc,事务，最终一致性，广播，消息确认，负载均衡，存储，流控，错峰，调度），如何设计。

消息确认，最终一致性

mq是怎么保证高吞吐的？

Kafka利用操作系统的页缓存技术+磁盘顺序写保证写，利用零拷贝技术实现读，

对于mq存储，通过哪些原理及手段，可以保证mq tps远高于磁盘I/O

Kafka利用操作系统的页缓存技术+磁盘顺序写保证写，利用零拷贝技术实现读，

如何保证mq消息尽量不丢失

高可用通过多副本冗余和副本同步来保证，isr机制

对于各种不确定(超时,down机,消息没有送达,送达后数据没落地,数据落地了回复没收到)如何做到消息不丢失，如何避免重复消费的问题，如何鉴别是重复消费（messageId）并幂等的处理(状态机/版本号，前者更优)

是否有顺序保证？如果需要一个支持顺序保证，应该如何选型和设计，比如kafka只能保证单个partition

怎么避免重复消费。

这个需要保证业务代码有幂等性，也可以使用Redis缓存记录每笔被消费的信息

Kafka有消息回执，如果因为网络问题没有收到消息回执的话，需要通过其他方式确定消息被消费了。

kafka是如何实现高性能高可靠的。zero copy的原理

Kafka利用操作系统的页缓存技术+磁盘顺序写保证写，利用零拷贝技术实现读；

zero copy从页缓存读出数据知道发送给网卡，不用读取到应用程序缓存在到socket里面

高可用通过副本同步来保证，isr机制

kafka副本同步机制

kafka选主机制，是否是类似paxos的算法，答案否。

数据可靠性和一致性如何保证。

高可用通过多副本冗余和副本同步来保证，isr机制

kafka副本与leader节点不同步的原因。

kafka isr和osr机制，HW设计理念。

自动给每个Partition维护一个ISR列表，这个列表里一定会有Leader，然后还会包含跟Leader保持同步的Follower。只要大于一个follower和leader保持同步，那么leader挂了的时候，推举出来的leader就可以根据这个follower来保证Partition数据不丢失。

kafka部署的时候如果节点小于和大于partition个数的时候都会有什么现象和结果。

开源框架（spring ,hbase ,zookeeper等）

aop实现原理,依赖注入。

依赖注入：AutowiredAnnotationBeanPostProcessor在bean创建之前已经被注册到了BeanFactory中，在bean实例化之后，初始化的过程中会调用处理器处理。

Spring 事务的实现原理

SpringMVC的原理和机制

hbase架构原理，特性，性能。

hbase row key的设计原则。

openstsdb?

zookeeper原理，客户端类图。内部双层队列设计的思想。

Znode，zab协议。

zookeerper observer节点使用场景。

zookeeper 部署的时候需要注意的细节，如何防止脑裂

zab协议吧

服务注册，服务分组 ，服务发现，异常上报，服务降级 ，监控检查，动态upstream（http）

dubbo提供了服务分组的概念：只要服务端和消费端处在同一个分组中，就可以实现指定消费端消费指定的接口实现。

服务注册，一个dubbo servie对应一个servieBean，在servicebean初始化的时候会去进行注册，底层是操作zookeeper的目录结构

nginx lua openResty zookeeper

四层负载均衡和第七层负载均衡区别和好处

问下大型企业架构他用过哪些，对比优劣。

SOA和微服务的区别。

微服务不再强调SOA架构中的ESB服务总线，是一个去中心化的面向服务架构。而SOA也需要依赖ESB的面向服务架构，可以理解为微服务为SOA的进化

服务治理如果是使用spring-cloud，可以进一步询问其对相关组件的理解和简单原理描述。Eureka，Zuul, Consul , Feign, Hystrix 都可以自然引入。

如果使用 Dubbo，深入询问

如果网络异常导致服务器请求失败, 需要怎么做(重试)

可能是超时引起的，可以将timeOut方法，dubbo自身的机制会去重试

重试会导致什么后果(雪崩, 幂等)

但是如果不合理的配置重试次数，当失败时会进行重试多次，这样在某个时间点出现性能问题，调用方再连续重复调用，  
系统请求变为正常值的retries倍，系统压力会大增，容易引起服务雪崩，需要根据业务情况规划好如何进行异常处理，何时进行重试。

雪崩怎么解决(限流, 熔断, 隔离)

限流，通过nginx

熔断，隔离，Hystrix

如果是dubbo重试导致的雪崩可以通过设置重连次数，或者禁用重连机制

幂等怎么解决(token, 乐观锁)

SOA与传统ESB消息总线的差异在哪里？各自的应用场景是什么？

Spring



离职证明和社保缴费证明，公积金不要提出来

[wang.yanfang@iwhalecloud.com](mailto:wang.yanfang@iwhalecloud.com) ，王艳芳，18913397268