**京东：我的：**

1. **ConcurrentHashMap和HashTable的区别**

**答案：主要体现在实现线程安全的方式上不同，**

**底层数据结构：ConcurrentHashMap跟HashMap的结构一样，数组+链表/红黑树；HashTable采用数组+链表，数组是HashMap的主体，链表是为了解决哈希冲突存在的；实现线程安全的方式：ConcurrentHashMap用Node数组+链表+红黑树的数据结构实现，并发控制使用synchronized和CAS来操作，synchronized只锁定当前链表或红黑树的首节点，这样只要hash不冲突，就不会产生并发，效率又提升N倍；HashTable（同一把锁），使用synchronized来保证线程安全，效率非常低下，当一个线程访问同步方法时，其他线程也访问同步方法，可能会进入阻塞或轮训状态，竞争会越来越激烈，效率会越来越低**

1. **HashMap线程不安全是在并发执行put操作时会引起死循环，导致CPU利用率接近100%，因为多线程会导致HashMap的Node链表形成环形数据结构，一旦形成环形结构，那么Node的next永远不为null,产生死循环。**
2. **如何使HashMap线程安全？**

**答案：Collections.synchronizedMap(),这个类中使用了synchronized同步关键字来保证Map的操作时线程安全的。**

1. **乐观锁和悲观锁**

**答案：悲观锁：总是假设最坏的情况，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会阻塞直到他拿到锁，共享资源只给一个线程使用，其他线程阻塞，用完后再把他让给别的用，适用于多写的场景下使用。乐观锁：总是假设最好的情况，每次去拿数据的时候都认为不会修改数据，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有更新这个数据，可以使用版本号机制和CAS算法，适用于多读的场景下使用，提高吞吐量。版本号机制是version代表数据被修改的次数，当被修改一次，version+1，提交版本必须大于记录当前版本才能执行更新。CAS（比较与交换），是一种有名的无锁算法，**

1. **Redis缓存击穿？缓存穿透？缓存雪崩？**

**答案：缓存击穿是某个key频繁被访问，然后就失效了，导致后面的请求直接落到了数据上。**

1. **NIO、AIO、BIO？然后问了一下NIO应用到了什么框架上？**
2. **Servlet生命周期？**
3. **JVM的参数？**
4. **垃圾回收算法？**
5. **Synchronized和Reentranlock?**
6. **Tomcat默认端口号？**
7. **项目中Redis过期缓存时间设了多少？**
8. **Spring IOC和Spring AOP？AOP代理？**
9. **数据库表怎么设计的？有没有使用外键？**
10. **MySQL索引？优缺点？可能会带来什么问题？**
11. **Char和varchar的区别？**

**答案：char的长度是不可变的，而varchar的长度是可变的，如果定义的长度同样都是10，那么存进去三个字符，char的话后面跟6个空格，varchar的话，长度就变成4，两个都不能存进去大于10个字符，最多存10个字符。Varchar的存取速度没有char快，char因为是固定长度的，方便程序的存储于查找，但是付出的空间代价，char取数据的时候要用trim()去掉多余的空格，varchar不需要。**

**霍霞：**

1. **AOP实现日志？**
2. **讲一下Spring的核心、优点？**

**答案：Spring是很多模块的集合，使用这些模块能很方便地协助我们进行开发，核心技术有Spring IOC，AOP，事件支持**

1. **Spring注解@Configuration @Bean**

**答案：configuration用于定义配置类，可替换xml配置文件，该注解的类内部包含一个或多个@Bean注解的方法。用@Configuration加载Spring并启动Spring，@Bean注解作用于方法，通常是我们在标有该注解的方法中定义产生这个bean，@Bean注解来注册bean。比如第三方类库要装配到spring容器时，就用@Bean。**

1. **Reentrantlock?了解原理不？**
2. **ArrayList和LinkedList？**
3. **线程怎么创建？几种方式及区别？**
4. **线程池怎么创建？**
5. **数据库用的什么？慢查询了解吗？事务的隔离级别？SQL语句优化？explain？**

**答案：慢查询就是在日志中记录运行比较慢的SQL语句，需要开始这个功能在能使用。查询慢分两种，1.偶尔很慢：a.数据库在刷新脏页，有四种场景，（1）redolog写满了，就要stop the world，全身心把内存中的数据同步到磁盘中去；（2）内存不够用了，（3）MySQL认为系统空闲的时候（4）MySQL正常关闭的时候。b.拿不到锁，涉及到的表别人在用，然后加锁了。得等。**

**2.一直很慢的情况，（1）没用到索引（2）索引没用对（3）函数操作导致有索引没用上（4）选错索引**

1. **Jdk用的哪个版本？lambda表达式**

**答案：也称为闭包，java8的新特性，他允许函数作为一个方法的参数（parameter）->expression**

**面经：**

1. **怎么确定索引生效？**

**答案：用explain这个命令去查看该SQL语句有么有使用上了索引，有没有做全表搜索等。**

1. **Spring的IOC讲一讲，然后是单例？可以实现多实例吗？**

**答案：spring bean的默认是单例的，多例的话，就加上@Scope注解把属性设置为prototype，prototype就是多例，每次请求都会去创建一个新的bean实例。**

1. **两个对象怎么比较？**
2. **那两个Integer 100用==么？**
3. **如何实现一个生产者消费者模型，使用什么数据结构？**
4. **Hash冲突解决方法**
5. **多线程的创建方式**
6. **服务器CPU数量及线程池数量关系**
7. **分布式session设置**

**答案：**

1. **数据库垂直和水平拆分**

**答案：水平分区可以支持很大的数据量，但是最好分库，跨节点join性能比较差**

**垂直分区：优点可以使得列数变小，减少读取时的block数，IO数，表易于维护，缺点：主键出现冗余，需要管理冗余列，并会引起join操作，可以通过在应用层使用join来解决，垂直分区会让食物变得更复杂。**

1. **分布式session一致性**

**答案：session是客户端与服务器间通讯会话的跟踪技术**

1. **项目中如何操作redis的？**
2. **ArrayList、vector、HashSet的主要区别CopyOnWriteArrayList的原理**
3. **HashMap（）在并发环境下回出现什么问题？如果两个线程同时访问map.get(key),key是HashMap已经存在的，会出现什么问题**

**答案：多并发的put()可能导致数据丢失，put和get并发时，并发可能导致get为null；并发put会造成循环链表，造成get出现死循环。**

1. **讲讲如何实现三个线程交替打印ABC**
2. **讲讲线程协作的方式？SyclicBarrier的底层实现**
3. **Volatile底层原理？不是回答内存屏幕，是问你调用了什么CPU的API，在不同的硬件情况下有什么不同吗**
4. **Redis的五种数据类型，讲讲sorted set是基于什么实现的？为什么既用压缩列表，又用跳跃表？讲讲跳表这种数据结构？为什么要加随机函数**

**答案：跳跃表是Zset的底层数据结构，因为zset要支持随机的插入和删除，所以不宜用数组实现，也没有用红黑树或者平衡树，而是采用跳跃表，是因为1.性能考虑：在高并发的情况下，属性结构需要执行一些类似于Rebalance这样的可能涉及整棵树的操作，相对来说跳跃表只涉及局部。2.实现考虑，在复杂度与红黑树相同的情况下，跳跃表实现起来更简单，看起来也更直观。**

1. **可重入锁是啥，跟synchronized区别？**
2. **可重入性指的是什么**
3. **线程池，scheduled的实现？scheduled如何实现定时**
4. **创建线程两种方式区别？runnable和callable**
5. **Redis单线程如何保证高效工作？redis持久化的操作，RDB具体如何实现**

**答案：redis是基于内存的，内存的读写比较快，是单线程的，省去了很多上下文切换线程的时间；redis使用非阻塞io多路复用技术，可以处理并发的连接；非阻塞IO多路复用内部实现采用epoll,epoll中的读写、关闭、连接都转化成了事件，然后利用epoll的多路复用特性，觉不在IO上浪费一点时间**

1. **I/O多路复用，eporll有什么优点，比较了一些select和poll**

**答案：非阻塞IO多路复用内部实现采用epoll,epoll中的读写、关闭、连接都转化成了事件，然后利用epoll的多路复用特性，觉不在IO上浪费一点时间**

1. **数据库特别大怎么办，具体怎么实现**
2. **线程池有哪些种类？说一下线程池的参数**
3. **线程池的拒绝策略**
4. **线程数量是不是越高越好呢**

**答案：如果创建非常多的线程，因为服务器的CPU核数有限，CPU利用率满了，内存页满了，说明线程不是越多越好，并且线程的创建和销毁需要很多时间的，如果创建线程的消耗大于执行任务，那就没必要用多线程了，同时CPU频繁在线程之间切换也影响性能。**

1. **CPU在线程之间切换都涉及了哪些东西**

**答案：**

1. **JVM的内部结构**
2. **双亲委派机制**
3. **Redis常用命令**
4. **Redis在项目中是怎么使用的**
5. **是怎么实现redis分布式锁的**

**答案：基于zookeeper临时有序节点可以实现分布式锁，**

1. **讲讲join，left join？**

**答案：通过join可以从多个表中读取数据，join有三种：1，inner join，获取两个表中字段匹配关系的记录。2.left join,获取左表所有列，即使右表没有对应匹配的记录。3.right join，有left join恰恰相反**

1. **@Autowired是怎么回事？项目中有没有使用该注解报错的情况**

**答案：@Autowired是根据类型进行自动装配的，这时候Spring容器中匹配的候选bean数目必须且仅有一个。当Spring上下文中存在不止一个候选bean的时候，就会抛出BeanCreationException；如果Spring上下文中不存在候选Bean，也会抛出BeanCreationException异常。怎么解决：通过xml配置文件的方式在Spring的配置文件里实现一个类型多个bean；也可以通过使用配置类+注解的方式实现一个类型多个bean。**

1. **如果有两个类实现了同一个接口，那么在使用@Autowired时能正确加载到吗**

**答案：不能，上面有解决方法。**

1. **Bean在初始化后，如果还想对其进行其他属性设置改怎么办？面试官说的要用afterPropertiesSet方法**
2. **Redis的hash是怎么存的？怎么扩容的？**

**答案：hash是一个string类型的field和value的映射表，内部实现就跟HashMap类似，都是通过数组+链表的链地址法来解决哈希冲突，实际上字典结构的内部包含两个HashTable，通常情况下只有一个HashTable是有值的，但是在字典扩容的时候，需要分配新的HashTable，然后进行渐进式搬迁，搬迁结束之后，旧的HashTable被删除，新的HashTable取而代之。往字典中添加键值对时，需要根据键的大小算出哈希值和索引值，然后根据索引值，将包含新键值对的哈希表节点放到哈希表数组的指定索引上面。**

**扩容：当hash表中元素个数等于第一维数组的长度时，就会开始扩容，扩容的数组是原来的数组大小的2倍，不过，如果redis正在做bgsave,为了减少内存页的过多分离，redis就尽量不去扩容，但是如果hash表非常满了，元素的个数硬达到了第一维数组长度的5倍，表明过于拥挤，这个时候就要去扩容。**

1. **有三个接口，等到三个接口都做完自己的事之后再对他们做一个类似的收束归并的操作，使用java多线程这块是怎么设计？**
2. **ThreadLocal？**
3. **数据库性能存在问题，该如何排查**

**答案：1.捕获查询到查询日志文件中；（1）使用慢查询日志，可以使用long\_query\_time = 0来捕获所有查询日志，或者要是权限不够，可以通过抓取TCP包，然后根据MySQL客户端/服务器端通信协议进行解析。2.分析查询日志，使用pt-query-digest工具生成剖析报告**

1. **操作类型从网卡读取数据的过程？IO类型**
2. **http和https,dns,tcp和UDP？**
3. **字节码变为运行类的机制是什么**
4. **消息队列出现积压如何处理**

**答案：1.如果仅仅是consumer消费的速度落后于消息生产的速度的话，可以考虑采用扩容消费者群组的方式；2.如果积压比较严重，积压了上百万，上千万的消息，修复现有consumer的问题，并将其停掉，重建一个容量更大的topic，比如patition是原来的10倍，编写一个临时consumer程序，消息原来积压的队列，该consumer不做任何耗时的操作，将消息均写入新创建的队列里，将修复好的consumer部署到原来10倍的机器上消费新队列，等消息几页解决后，恢复原有架构。3.如果消息已经丢失，只能将那些丢失的数据，写个临时程序，一点一点的排查出来，然后重新灌入mq里面去。**

1. **三次握手，四次挥手，time\_wait状态**
2. **在一次数据交换过程，IP地址和MAC地址会不会变？（IP地址不变，MAC地址会变，因为跨域了）**
3. **IOC容器，循环依赖解决方案**

**答案：bean A依赖B，B又依赖A，先实例化A，接着实例化B，在容器发现B又依赖于A的时候，容器会获取A对象的一个早期引用，并把这个早期引用注入到B中，让B先完成实例化，B完成实例化，A就可以获取到B的引用，Bean A随之完成实例化。**

1. **threadLocal内存泄露，解决方案**

**答案：threadlocal类用来提供线程内部的局部变量。这些变量在多线程环境下访问时能保证各个线程里的变量相对独立于其他线程内的变量。Threadlocal不是为了解决多线程访问线程共享变量，而是为每个线程创建一个单独的变量副本，提供了保持对象的方法和避免参数传递的复杂性。**

1. **java反射机制**

**答案：任何时候都可以获取，类的成员变量和成员方法，任何时候都可以调用对象的属性和方法。**

1. **布隆过滤器线程安全实现？**

**答案：多线程读，和多线程写是线程安全的；多线程读写不是线程安全的。是通过AtomicLongArray原子数组的方式保证线程安全的。之前的版本只是普通的long数组**

1. **HashMap和concurrentHashMap的多线程安全实现**

**答案：Collections.SynchronizedMap()**

1. **Bean的创建方式**

**答案：1.调用构造器创建bean;2.调用静态工厂方法创建Bean；3.调用实例工厂方法创建Bean**

1. **数据库压力太大怎么办？被面试的人说redis和本地缓存（面试官：分片算法）**

**答案：1.合理增加索引；2.数据截转：一般锁处理的数据都具有一定的时间间隔，所以可以通过对业务进行梳理，将当前时间间隔外的数据进行截转，截转到历史数据库中，通过对业务进行拆分，当需要历史数据是，可以转到历史数据库中查询或者修改，通过减少当前数据库的数据量来减轻当前业务数据的压力；3.增加缓存；4.可以修改关系型数据库为非关系型数据库，查询速度非常快；5.读写分离：通过将数据拆分为两个实例，读写操作改善了数据单点的瓶颈，分摊了数据库的压力，而且当主机宕机时可以迅速到从库去查，同时也起到了备份的作用，也不会耽误业务（但是需要考虑主从同步，主从之前延迟引发的数据不一致问题，以及怎么分离读和写操作）；6.数据库拆分：采用读写分离之后，数据库就已经变为两份实例，数据库的压力得到平摊；7.数据表的拆分：水平拆分，还有垂直拆分**

1. **京东如果要秒杀，数据库受的住吗？改怎么处理？（从基本的分部分表讲到数据库的读写分离，另外还讲了缓存的优化方案）**

**答案：55和56差不多**

1. **ArrayList扩容机制？**

**答：以无惨构造方法创建ArrayList时，实际上初始化赋值的是空数组，当真正对数组进行添加元素的时候，才真正分配容量，添加第一个元素的时候，数组扩容为10，一直到添加10个元素，数组容量都是10，直到添加11个元素，数组会扩容为1.5倍容量，往后每次扩容都是1.5倍容量。然后注意最好在每次add大量元素前用ensureCapacity方法，以减少增量重新分配的次数。有参构造方法时，如果参数大于0，就直接分配容量为参数大小，如果参数等于0，那么就空数组，如果参数小于0，抛出异常。**

**项目中为什么使用MySQl**

**非关系型数据库的优势：1.性能：nosql是基于键值对的，可以想象成表中的主键和值的关系，不需要经过SQL层的解析，性能非常高，2.可扩展性，因为基于键值对，所以数据之间没有耦合性，所以非常容易水平扩展**

**关系型数据库的优势：1.复杂查询，可以用SQL语句在一个表以及多个表中进行非常复杂的查询，2.事务支持，使得对于安全性能很高的数据访问要求得以实现**

**电竞项目：**

**1. ELK日志中心 https://mp.weixin.qq.com/s?\_\_biz=MzU4ODI1MjA3NQ==&mid=2247484928&idx=2**

**&sn=4db40348f22f82b1106122cafbd77c33&chksm=fdded0c4caa959d25f914c79849cdef56045ce**

**53ce18256c26fad0b9d967b575e4e9902da211&scene=21%23wechat\_redirect**

**2. nginx的相关配置https://segmentfault.com/a/1190000018359026**

**3. docker compose https://www.jianshu.com/p/4f14637f4b35**

**4. 性能测试 https://testerhome.com/articles/17052**

**安全性就是使用的简单的cookie-session 用户登录做了哈希加密 也做了一些，把用户的IP每次都在redis里面记录一下，在一定时间段内，如果同一个IP访问的次数，超过我们设置的值，就不让他访问。**

**吞吐量这些没有考虑过，同时在线的，我记得之前测试是大约1000多的用户同时在线 解决方法就是调nginx的配置，和使用pm2，利用多核心来提高并发**

**是这样的，nginx是一个独立的服务器软件与真正实现业务的服务器软件（nodejs）是独立的，如果没有这个nginx的话所有的静态资源都要从nodejs中去拿，**

**就会成为业务处理的一个累赘，所以就把静态资源的请求响应交给nginx,换句话说对于静态资源的请求不是由nodejs处理的，而是nginx处理的，并且**

**NGINX本身对于高并发的io支持是很好的**

**使用Nginx作为静态资源缓存和请求转发，Nginx是轻量级的服务器**

**Redis缓存实际上就是类似于MySQL 只要一个模块就可以进行增删改查的操作，缓存的就是用户的登录状态，也就是cookie-session中的session，然后实现跟踪是在cookie中加了session ID方式来跟踪，如果cookie被禁用了，那就利用URL重写把session ID直接附加在URL路径后面说 和缓存**

**静态电竞数据（复盘数据都是经过缓存的），与此同时我们还做了hystrix限流和降级，防止缓存雪崩**

**使用的是一个叫做koa2-ratelimit的nodejs模块**

**pm2 是一个带有负载均衡功能的Node应用的进程管理,pm2可以把你的应用部署到服务器所有的CPU上,即利用CPU多核心部署了多个web服务器实例，提高并发量，**

**做分布式session一致性：说白了就是服务器集群session共享的问题。因为有多个KOA实例，所以用户验证就必须提取出来做多个实例的共享，于是将用户登录信息存储到Redis中；我是用了一个叫做koa-session的插件存放在redis中，其大概原理就是每次有用户请求的时候就调用一次redis查询对应的session id是否存在，若存在就是登陆状态，如果没有就存到redis，然后每个实例都从这个redis进行读写，就可以做到session一致性了**

**持久化机制是用了AOF持久化，appendsync everysec每秒钟同步一次，主要因为我们这个是比赛数据，设置成每秒同步一次对比赛数据没有太大的影响，一秒之内发生的竞技数据变化不是很大**

**redis缓存过期时间是按照不同的资源类型设置的，一些不怎么变动的数据，比如队伍信息，以及复盘的数据 我就设置了1小时和10天的的缓存过期时间，**

**而对于其余的变动频繁的，就设置了半分钟的缓存失效时间**

**生命周期：** **Web容器加载Servlet并将其实例化后，Servlet生命周期开始**，容器运行其**init()方法**进行Servlet的初始化；请求到达时调用Servlet的**service()方法**，service()方法会根据需要调用与请求对应的**doGet或doPost**等方法；当服务器关闭或项目被卸载时服务器会将Servlet实例销毁，此时会调用Servlet的**destroy()方法**。**init方法和destroy方法只会执行一次，service方法客户端每次请求Servlet都会执行**。

**Get（）和post的底层都是TCP**

Servlet为先编译后部署。而JSP则是先部署后编译。JSP会在客户端第一次请求JSP文件时被编译为HttpJspPage类（接口Servlet的一个子类）

开发Web程序时经常需要修改JSP。Tomcat能够自动检测到JSP程序的改动。如果检测到JSP源代码发生了改动。Tomcat会在下次客户端请求JSP时重新编译JSP，而不需要重启Tomcat。这种自动检测功能是默认开启的，检测改动会消耗少量的时间，在部署Web应用的时候可以在web.xml中将它关掉。

**Cookie和Session的的区别**

Cookie 和 Session都是用来跟踪浏览器用户身份的会话方式，但是两者的应用场景不太一样。

**Cookie 一般用来保存用户信息** 比如①我们在 Cookie 中保存已经登录过得用户信息，下次访问网站的时候页面可以自动帮你登录的一些基本信息给填了；②一般的网站都会有保持登录也就是说下次你再访问网站的时候就不需要重新登录了，这是因为用户登录的时候我们可以存放了一个 Token 在 Cookie 中，下次登录的时候只需要根据 Token 值来查找用户即可(为了安全考虑，重新登录一般要将 Token 重写)；③登录一次网站后访问网站其他页面不需要重新登录。**Session 的主要作用就是通过服务端记录用户的状态。** 典型的场景是购物车，当你要添加商品到购物车的时候，系统不知道是哪个用户操作的，因为 HTTP 协议是无状态的。服务端给特定的用户创建特定的 Session 之后就可以标识这个用户并且跟踪这个用户了。

Cookie 数据保存在客户端(浏览器端)，Session 数据保存在服务器端。

Cookie 存储在客户端中，而Session存储在服务器上，相对来说 Session 安全性更高。如果使用 Cookie 的一些敏感信息不要写入 Cookie 中，最好能将 Cookie 信息加密然后使用到的时候再去服务器端解密。

**因为我男朋友也在北京，所以我们想在北京长期发展，而且北京的公司多，机会多，成长大，自己还年轻，想奋斗，也是因为刚毕业，希望能够快速磨练一下技术啥啥的，能够快速成长起来**

**我这个一般不怎么和人发生矛盾，如果真有矛盾了，合理沟通，找到大家矛盾点在哪，解决这个矛盾**

**KOA框架：我的理解**

**KOA框架（基于nodejs，是express的下一代web框架），express很流行，但是实现异步的代码很难看，所以KOA里面引入了async,使用：async function ()**

**轻轻松松把function（）函数变为异步模式**

**KOA MVC可以结合处理不同的URL和Nunjucks渲染模板；当一个用户通过浏览器请求一个URL时，KOA将调用某个异步函数处理该URL，通过Nunjucks把**

**数据用指定的模板渲染成HTML，然后输出给浏览器，用户就可以看到渲染后的页面了**

**MVC（model-view-controller），异步函数是controller，负责业务逻辑，比如检查用户是否存在，取出用户信息等等**

**view负责显示逻辑，其实就是HTML页面**

**model是用来传给view的，这样view在替换变量的时候，就可以从model中取出相应的数据，model就是js**

**async function(ctx, next),其中，参数ctx是由koa传入的封装了request和response的变量，**

**我们可以通过它访问request和response，next是koa传入的将要处理的下一个异步函数。**

**可以用await next();处理下一个异步函数**