## 今日头条

顺序忘了甚至连题目都忘了，记得这些

### AOP原理

Spring AOP使用的动态代理，所谓的动态代理就是说AOP框架不会去修改字节码，而是在内存中临时为方法生成一个AOP对象，这个AOP对象包含了目标对象的全部方法，并且在特定的切点做了增强处理，并回调原对象的方法。

Spring AOP中的动态代理主要有两种方式，JDK动态代理和CGLIB动态代理。JDK动态代理通过反射来接收被代理的类，并且要求被代理的类必须实现一个接口。JDK动态代理的核心是InvocationHandler接口和Proxy类。

如果目标类没有实现接口，那么Spring AOP会选择使用CGLIB来动态代理目标类。CGLIB（Code Generation Library），是一个代码生成的类库，可以在运行时动态的生成某个类的子类，注意，CGLIB是通过继承的方式做的动态代理，因此如果某个类被标记为final，那么它是无法使用CGLIB做动态代理的。

我的理解：切面有两种1.编译时增强对象2.动态代理。Spring aop采用的是第二种。动态代理有jdk动态代理，cglib动态代理两种。实现接口的情况下为jdk，没有时采用cglib。Cglib是一个动态代码库，动态生成某个类（目标类）的子类，若一个类是final的那么，无法用cglib代理。

### sychronize lock 原理

synchronize是jvm软件层面，lock是java代码层面。

sychronize语义上可以实现任何非null对象的锁。

1)当synchronized作用在方法上时，锁住的便是对象实例（this）；

2)当作用在静态方法时锁住的便是对象对应的Class实例，因为 Class数据存在于永久带，因此静态方法锁相当于该类的一个全局锁；

3)当synchronized作用于某一个对象实例时，锁住的便是对应的代码块。

在 HotSpot JVM实现中，锁有个专门的名字：对象监视器。

对象监视器：

1. 线程状态及状态转换

当多个线程同时请求某个对象监视器时，对象监视器会设置几种状态用来区分请求的线程：

Contention List：所有请求锁的线程将被首先放置到该竞争队列

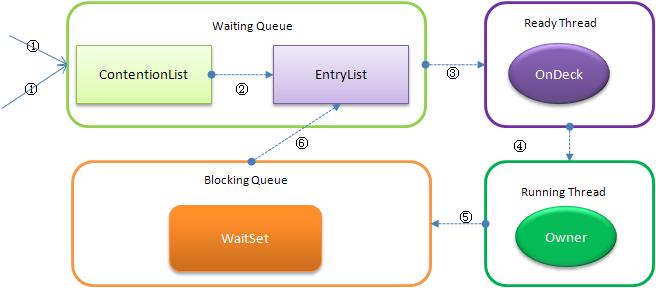
Entry List：Contention List中那些有资格成为候选人的线程被移到Entry List

Wait Set：那些调用wait方法被阻塞的线程被放置到Wait Set

OnDeck：任何时刻最多只能有一个线程正在竞争锁，该线程称为OnDeck

Owner：获得锁的线程称为Owner

!Owner：释放锁的线程



Lock：

Jdk8下Java.util.concurrent.lock包下有三个实现类ReentrantLock,RentrantReadWriteLock,StampedLock

两个必备元素:

一个是表示（锁）状态的变量（我们假设0表示没有线程获取锁，1表示已有线程占有锁）,该变量必须声明为voaltile类型;

另一个是队列，队列中的节点表示因未能获取锁而阻塞的线程。

过程：

1. 读取表示锁状态的变量

2. 如果表示状态的变量的值为0，那么当前线程尝试将变量值设置为1（通过CAS操作完成），当多个线程同时将表示状态的变量值由0设置成1时，仅一个线程能成功，其它线程都会失败:

2.1 若成功，表示获取了锁

2.1.1 如果该线程（或者说节点）已位于在队列中，则将其出列（并将下一个节点则变成了队列的头节点）

2.1.2 如果该线程未入列，则不用对队列进行维护，然后当前线程从lock方法中返回，对共享资源进行访问。

2.2 若失败，则当前线程将自身放入等待（锁的）队列中并阻塞自身，此时线程一直被阻塞在lock方法中，没有从该方法中返回（被唤醒后仍然在lock方法中，并从下一条语句继续执行，这里又会回到第1步重新开始）。

3. 如果表示状态的变量的值为1，那么将当前线程放入等待队列中，然后将自身阻塞（被唤醒后仍然在lock方法中，并从下一条语句继续执行，这里又会回到第1步重新开始）

注意: 唤醒并不表示线程能立刻运行，而是表示线程处于就绪状态，仅仅是可以运行而已

线程释放锁的大致过程

1. 释放锁的线程将状态变量的值从1设置为0，并唤醒等待（锁）队列中的队首节点，释放锁的线程从就从unlock方法中返回，继续执行线程后面的代码

2. 被唤醒的线程（队列中的队首节点）和可能和未进入队列并且准备获取的线程竞争获取锁，重复获取锁的过程

注意：可能有多个线程同时竞争去获取锁，但是一次只能有一个线程去释放锁，队列中的节点都需要它的前一个节点将其唤醒，例如有队列A<-B-<C ，即由A释放锁时唤醒B，B释放锁时唤醒C

### hashmap实现

1. entry数组
2. java8 tablelist超过8变成红黑树
3. 负载因子7.5

### 两个单向链表，求节点

### 乐观锁、悲观锁

### mysql引擎还有几种—

补充一下锁的知识

表级锁：开销小，加锁块；不会出现死锁，锁定粒度大，发生锁冲突的概率最高，并发度最低。

行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发性也最高。

1. myisam

无法处理事务，适用于选择密集型（筛选数据快），插入密集型（允许同时插入和选择数据）。

表级锁。

1. innodb

事务，行级锁，外键约束

行级锁又分共享锁和排他锁。

共享锁：

名词解释：共享锁又叫做读锁，所有的事务只能对其进行读操作不能写操作，加上共享锁后在事务结束之前其他事务只能再加共享锁，除此之外其他任何类型的锁都不能再加了。

用法：SELECT `id` FROM table WHERE　id in(1,2) LOCK IN SHARE MODE 结果集的数据都会加共享锁

排他锁：

名词解释：若某个事物对某一行加上了排他锁，只能这个事务对其进行读写，在此事务结束之前，其他事务不能对其进行加任何锁，其他进程可以读取,不能进行写操作，需等待其释放。

用法：SELECT `id` FROM mk\_user WHERE id=1 FOR UPDATE

### redis内部原理

底层数据结构有7种，其中简单动态字符串有两种编码，如果拆开看就是8种

1. 简单动态字符串，有两种编码embstr（这种更小）、raw
2. 字典
3. 双端链表
4. 压缩列表
5. Long类型整数
6. 整数集合
7. 跳跃表和字典

String 简单动态字符串

链表 ziplist,linkedlist

Hash ziplist hashtable hashtable使用字典实现的

集合对象 intset(整数集合) hashtable

有序集合 ziplist 另一种是skiplist和dict的结合

### mysql postgre区别

### hashmap线程不安全的体现

resize的时候，会把旧的链表里的头节点转移到新hash表中，如果两个线程一起操作，可能会引起环形链。

## 阿里大麦网

## 尚德机构

## 东方国信

## 58赶集

1. linux通过那些指标衡量服务

1)cpu和内存类

top: **top命令**可以实时动态地查看系统的整体运行情况，是一个综合了多方信息监测系统性能和运行信息的实用工具。通过top命令所提供的互动式界面，用热键可以管理。

2)vmstat

**vmstat命令**的含义为显示虚拟内存状态（“Viryual Memor Statics”），但是它可以报告关于进程、内存、I/O等系统整体运行状态。

1. linux指令了解哪些
2. top命令
3. 进程之间通讯的方式

https://blog.csdn.net/truong/article/details/39228249/

1. 管道 他是半双工的，数据往一个方向流动，有读端和写端；只能用于父子或兄弟进程间通信；可看成一种特殊的文件，只存在与内存中的文件
2. Fifo 命名管道，也是一种文件类型，可在无关进程间通信，是一种特殊的文件，有文件名存在与文件系统中
3. 消息队列 消息队列是消息链表，存放在内核中。一个消息队列有一个标识符（即消息ID）来标识。消息具有特定的格式和优先级；消息独立于进程，进程死了也不影响已经发送的消息；可实现消息的随机查询，不一定按次序读写，也可按消息类型读取。
4. 信号 信号是比较复杂的通信方式，用于通知接受进程有某件事情发生。
5. 共享内存 多进程可以访问同一块内存，是最快的可用ipc方式。往往与其他通信机制比如信号量，来达到进程间互斥
6. 内存映射 内存映射允许多个进程把共享的文件映射到自己的进程地址空间
7. 信号量 是一个计数器，用于实现进程间互斥和同步，而不是用于存储进程间通信数据。只能在进程间同步，若要传递参数需要配合共享内存等实现；程序对信号量的操作都是原子操作；信号量值可加减任意整数；支持信号量组。
8. 套接字 socket
9. 接口类和抽象类区别
10. 抽象类有实现方法，接口没有是实现的方法，但有default
11. 抽象类可以有私有变量，接口没有
12. 抽象类单继承，接口多继承
13. 抽象类可以有静态方法，接口没有
14. 抽象类可以有静态变量且无限制，接口必须是 public final static
15. 抽象类可以有protect方法，接口没有
16. 抽象类有构造方法，接口没有
17. http是网络协议中的那层
18. 多线程的实现方式
19. 继承Thread
20. 接口Runnable
21. java网络变成框架
22. netty 提供一组基于事件的api来开发高性能，可管理的tcp/ip服务端的或客户端的引用程序。
23. nio framework

等等还有很多，这个有空学习下netty

1. http是那一层协议
2. osi七层网络模型 物理层 数据链路层 网络层 传输层 会话层 表示层 应用层
3. tcp/ip四层网络模型 网络接口层 网络层 传输层 应用层
4. http和tcp/udp 有什么关系

http:是用于www浏览的一个协议。  
tcp：是机器之间建立连接用的到的一个协议。

1. TCP/IP是个协议组，可分为三个层次：网络层、传输层和应用层。
2. HTTP协议是建立在请求/响应模型上的。首先由客户建立一条与服务器的TCP链接，并发送一个请求到服务器
3. Tcp是面向连接的，udp是无连接的
4. 一个文件，每一行一个无符数字，乱序有重复，想要排序和去重，前提文件特别大，内存很小

桶排序，然后快排，桶满了分裂，小文间快排，但这个去重是个问题

另一种：一个桶就是一个数字，初始都不在桶里状态‘不在’，然后遍历，然后把对应桶的状态改变为‘在’，然后把桶状态‘在’的写到另一个排好序的文件里，然后紧接着下一个桶，注意桶是无符号的那么就从0开始就行了

1. 单向链表传过来，逆序
2. http报文header 和body怎么分割，原问题描述不清了

header和body通过空行分割。

1. mysql行级锁、表级锁
2. mysql 事务的隔离级别
3. 未提交读 读取时不加锁；更新时加行级共享锁，直到事务结束释放
4. 提交读 读取时加行级共享锁，读取完成释放；更新加行级排他锁，事务结束释放
5. 可重复读 读取时加行级共享锁，事务结束释放；更新加行级排他锁，事务结束释放
6. 串行化 读取加表级共享锁，事务结束释放；更新加表级排他锁，事务结束释放
7. redis 底层结构
8. 两个栈实现队列
9. Redis灾备

Rdb和aof，rdb把将内存镜像转储为文件，aof把每个语句存到日志文件，rdb恢复快但不能实时更新，会丢数据，aof恢复慢，但不丢数据

1. Java中有哪些树型结构
2. 共享锁和排他锁

排他锁又称写锁，共享锁又称读锁。

加排他锁后，其他事务不呢个再加锁；加共享锁后，其他事物只能再加共享锁，而不能修改该对象。

1. 死锁

广义的死锁，实际分为死活和活锁

活锁 T1执行，T2等待锁，T3又来了，T1释放，锁被T3获取了，T2轮空，T4来了，T3释放，T4获取锁，T2又轮空了，一次类推，T2老获得不到锁，这是活锁。

避免活锁，先来先服务。

死锁 T1先加锁A，T2加锁B，然后T1等待B的锁，T2等待A的锁，互相等，不限于两个事务，这是死锁

避免死锁

1. 一次封锁，把所有要的锁全加锁
2. 顺序加锁，所有事务顺序加锁

死锁诊断

1. 超时法，容易误杀，超时时间过长又容易不及时发现死锁
2. 等待图法 事务做节点，事务间等待关系做又向连线。出现环就是死锁
3. 锁的粒度

粒度越大，能封锁的数据单元越少，并发度越小，系统开销越小；反之，粒度越小，并发度高，系统开销大

1. 显式加锁，隐式加锁

显式加锁，直接对对象加锁

隐式加锁，比如对表加锁，那么行自然也加了锁

## 常见题目总结

### spring IOC

### 自旋锁、偏向锁、cas和smp架构、cache一致性、锁膨胀

### Classloader

Bootstrap ClassLoader 负责加载java基础类，主要是 %JRE\_HOME/lib/ 目录下的rt.jar、resources.jar、charsets.jar和class等

Extension ClassLoader 负责加载java扩展类，主要是 %JRE\_HOME/lib/ext 目录下的jar和class

App ClassLoader 负责加载当前java应用的classpath中的所有类。

自定义ClassLoader需要继承ClassLoader抽象类，重写findClass方法，这个方法定义了ClassLoader查找class的方式。

主要可以扩展的方法有：

findClass 定义查找Class的方式

defineClass 将类文件字节码加载为jvm中的class

findResource 定义查找资源的方式

### 双亲委派模型

说白了就是先检查要加载的类在父类（加载器）里有没有，如果有就直接用，没有就自己加载。这样保证类步会被重复加载，而且安全，比如java.lang.String类不会被自定义的加载器覆盖掉。

### Https

https是http+ssl，加密方式是用非对称加密取得对称加密的密钥，再用对称密钥进行加密。