马老师说过，员工的离职原因很多，只有两点最真实：

- **钱**，没给到位
- **心**，受委屈了

当然，我当初是想换个平台，换个方向，想清楚为什么要跳槽，如果真的要跳槽，想要拿到一个理想的offer，除了运气，基本功也要足够的扎实，希望下面的面试经验能给你们能够提供一些帮助。

项目经验

面试官在一开始会让你进行自我介绍，主要是想让你介绍一下自己做过的一些项目，看看你对这些项目的了解程度，因为很多人简历上写的项目并非都是从头到尾都参与的，有些只是参与并实现了其中的一些模块而已，或是接手维护别人的项目，所以在你简历上所写的和面试过程中所说的项目经验，你自己必须能够了解来龙去脉，因为面试官肯定会根据你的项目描述，对项目中的实现原理，或为什么要这样实现进行提问，这时不至于木讷住而不知如何作答，如此局面只会大大降低面试分。

场景对话：

面试官：（拿着简历）讲讲你最近做的这个项目

我：&……%￥#*&￥@%￥！，说了一大通（不知道面试官听进去多少，面试官会挑他会的进行提问）

面试官：你说这个项目中用到了netty，能大概讲讲netty的线程模型么？

我：（幸好我看过netty的源码）netty通过Reactor模型基于多路复用器接收并处理用户请求（能讲就多讲一点），内部实现了两个线程池，boss线程池和work线程池，其中boss线程池的线程负责处理请求的accept事件，当接收到accept事件的请求时，把对应的socket封装到一个NioSocketChannel中，并交给work线程池，其中work线程池负责请求的read和write事件（通过口述加画图的方式，把请求的执行过程大概描述了一遍，时间有限，也不可能把所有的细节都说完，挑重点讲，挑记忆深刻的讲）

面试官：嗯，理解的还挺深入的...那你在做这个项目时有没有遇到什么困难，或者是觉得有挑战的地方？

我：（这时面试官想让你自己出题自己回答了，所以一定要回答，不回答就突显不出你这个项目了，要是这个问题没有准备过，只能临时发挥了，当然我就是属于临时发挥的）稍微想一下，因为之前确实碰到了这个问题，当时做这个项目时，对netty的不过熟悉，把请求的业务逻辑放在work线程池的线程中

进行处理，进行压测的时候，发现qps总是上不去，后来看了源码之后才发现，由于业务逻辑的处理比较耗时，完全占用了work线程池的资源，导致新的请求一直处于等待状态。

面试官：那最后是如何解决的？

我：最后把处理业务的逻辑封装成一个task提交给一个新建的业务线程池中执行，执行完之后由work线程池执行请求的write事件。

面试官：好的，你知道nio中selector可能触发bug么？

我：嗯，对的，selector的select方法，因为底层的epoll函数可能会发生空转，从而导致cpu100%。

面试官：那如何解决该问题？

我：这个问题在netty已经解决了，通过&^%&$^（把netty的解决方案说一遍）

面试官：嗯，对了，你们这个项目有给自己定指标么？

我：有的，&&.....￥￥##@，把自己项目的指标说了一通，如何进行AB实验，如何迭代优化指标

面试官：嗯，好的 ，项目的问题先到这里，我们来考察一下java的基本点吧。

如上只是本人所做的一个项目，当然了，具体项目具体分析，也不是每个面试官问的点都一样，如果面试官不懂netty，自然会挑别的问题进行提问，不过你也可以尝试着把问题往自己熟悉的方向去靠。

面试知识点

1、线程池

线程池的实现原理，这个知识点真的很重要，几乎每次面试都会被问到，一般的提问方式有如下几种：

1、 “讲讲线程池的实现原理”

2、 “线程池中的coreNum和maxNum有什么不同”

3、 “在不同的业务场景中，线程池参数如何设置”

场景对话：

面试官：平时线程池用的多么？

我：嗯，我的*项目中就用到了

面试官：那好，你讲讲线程池的实现原理

我：（还好我之前看过源码，但是时间久远有点模糊了），能给我笔和纸么，我画图分析给你看看，&&￥&假设初始化一个线程池，核心线程数是5，最大线程数是10@@@

面试官：嗯，好的，你继续...

我：在纸上画了正方形，这个代表一个线程池，初始化的时候，里面是没有线程的

面试官：嗯，好的，你继续...

我：又画了一个细长的长方形，这个代表阻塞队列，一开始里面也是没有任务的

面试官：嗯，好的，你继续...

我：当来了一个任务时，在正方形中画了一个小圆圈，代表初始化了一个线程，如果再来一个任务，就再画一个圆圈，表示再初始化了一个线程，连续画了5个圆圈之后，如果第6个任务过来了...

面试官：嗯，好的，你继续...

我：这时会把第6个任务放到阻塞队列中..

面试官：嗯，然后呢？

我：现在线程池中不是有5个线程了么，如果其中一个线程空闲了，就会从阻塞队列中获取第6个任务，进行执行..

面试官：嗯，对的，那如果任务产生的速度比消费的速度快呢？

我：如果线程池的5个线程都在running状态，那么任务就先保存在阻塞队列中

面试官：如果队列满了，怎么办？

我：如果队列满了，我们不是设置了最大线程数是10么，而线程池中只有5个线程，这时会新建一个线程去执行不能保存到阻塞队列的任务，然后我又在正方形中画了5个圆圈。

面试官：那如果线程池中的线程数达到10个了，阻塞队列也满了，怎么办？

我：这种情况通过自定义reject函数去处理这里任务了，舒了一口去，以为问完了...

面试官：好的，那如果运行一段时间之后，阻塞队列中的任务也执行完了，线程池中的线程会怎么样？

我：...这个好像超过核心线程数的线程会在空闲一段时间内自动回收...因为有点不记得这个逻辑了，回答的有点虚...

面试官：好的，那这种情况在什么场景下会发生?

我：（有时候真是笨啊，很多东西都知道，但是在面试的时候一紧张，全忘记）这个...那个...我好像没有遇到过这样的情况

面试官：嗯，好的，你回去之后再好好想想

我：.....

我居然忘记了秒杀这个场景

2、锁的实现

在关于锁的面试过程中，一般主要问Synchronized和ReentrantLock的实现原理，更有甚者会问一些读写锁。

场景对话：

面试官：都了解Java中的什么锁？

我：比如Synchronized和ReentrantLock...读写锁用的不多，就没研究了

面试官：那好，你先说说Synchronized的实现原理吧

我：嗯，Synchronized是JVM实现的一种锁，其中锁的获取和释放分别是monitorenter和monitorexit指令，该锁在实现上分为了偏向锁、轻量级锁和重量级锁，其中偏向锁在1.6是默认开启的，轻量级锁在多线程竞争的情况下会膨胀成重量级锁，有关锁的数据都保存在对象头中...&&@@#，（嗯，说了一大堆，面试官也没打断我）

面试官：哦，嗯，理解的还挺透彻，那你说说ReentrantLock的实现吧...

我：ReentrantLock是基于AQS实现的

面试官：什么是AQS？

我：在AQS内部会保存一个状态变量state，通过CAS修改该变量的值，修改成功的线程表示获取到该锁，没有修改成功，或者发现状态state已经是加锁状态，则通过一个Waiter对象封装线程，添加到等待队列中，并挂起等待被唤醒&&&$$（又说了一堆）

面试官：能说说CAS的实现原理么？

我：CAS是通过unsafe类的compareAndSwap方法实现的（心里得意的一笑）

面试官：哦，好的，那你知道这个方法的参数的含义的么？

我：（这是在逼我啊...努力的回想，因为我真的看过啊）我想想啊，这个方法看的时间有点久远了，第一个参数是要修改的对象，第二个参数是对象中要修改变量的偏移量，第三个参数是修改之前的值，第四个参数是预想修改后的值....（说出来之后都有点佩服自己，这个都记得，不过面试官好像还是不肯放过我...）

面试官：嗯，对的，那你知道操作系统级别是如何实现的么？

我：（我去你大爷...）我只记得X86中有一个cmp开头的指令，具体的我忘记了...

面试官：嗯，好，你知道CAS指令有什么缺点么

我：哦，CAS的缺点是存在ABA问题

面试官：怎么讲？

我：就是一个变量V，如果变量V初次读取的时候是A，并且在准备赋值的时候检查到它仍然是A，那能说明它的值没有被其他线程修改过了吗？如果在这段期间它的值曾经被改成了B，然后又改回A，那CAS操作就会误认为它从来没有被修改过。

面试官：那怎么解决？

我：（有完没完了啊...我的心里是崩溃的）针对这种情况，java并发包中提供了一个带有标记的原子引用类"AtomicStampedReference"，它可以通过控制变量值的版本来保证CAS的正确性。

面试官：嗯，好的，这个问题到此为止，我们再看看别的

我：....我能喝口水么

3、ConcurrentHashMap

当考察数据结构时，面试官一开始会问HashMap的实现原理，当你说出HashMap并非线程安全之后，会让你自己引出ConcurrentHashMap，接着就可能开始如下的对话。

场景对话：

面试官：谈谈ConcurrentHashMap实现原理

我：@#¥@@基于分段锁的%%¥#@#¥，但是1.8之后改变实现方式了

面试官：1.8啥方式

我：把1.8的实现原理说了一通，其中提到了红黑树...

面试官：能讲下红黑树的概念吗

我：红黑树是一种二叉树，并且是平衡……%……¥……，

面试官：能讲下红黑树的。。。。。

我：打住，别问了，红黑树我只知道他是二叉树，比其他树多一个属性，其他的我都不知道
面试官：好的，那换个，你知道它的size方法是如何实现的么？

我：size方法？是想要得到Map中的元素个数么？

面试官：对的....

我：我记得好像size方法返回是不准确的，平时也不会用到这个方法...

面试官：如果你觉得size方法返回值不准确，那如果让你自己实现，你觉得应该怎么实现呢？

我：...@#¥@@...两眼一黑

我：等等，让我想想.....应该可以用AtomicInteger变量进行记录...嗯，对的，每次插入或删除的时候，操作这个变量，我得意的一笑...

面试官：哦，是么，那如果我觉得这个AtomicInteger这个变量性能不好，还能再优化么？

我：懵逼脸...（当时居然把volitile变量给忘记了）...好像没有了，我想不出来了...

面试官：哦，那回头你再看看源码吧，jdk中已经实现了...

我：哦，是么....

面试官：那今天的面试到此结束，我们后面会通知你。

我：.....

虚拟机JVM相关

这块内容并非每个面试官都会问，但是如果是应聘高级职位的话，这一环节是不可缺少的，面试的难易程度也不一样，有些面试官或许让你讲讲虚拟机的内存模型即可，有些也会让你解释垃圾回收的实现，当然也会有虚拟机调优的实战经验，线上问题排查等等。

场景对话：

面试官：Java虚拟机有了解么？

我：恩，略有接触过...（水哥说过，话不能说太满，容易打脸）

面试官：那你先讲讲它的内存模型吧

我：Java堆，Java栈，程序计数器，方法区，1.7的永久代，1.8的metaspace....（噼里啪啦概念讲一通，简短描述下每个内存区的用途，能想到的都讲出来，不要保留，不要等面试官问 “还有吗？” ）

面试官：好，一般Java堆是如何实现的？

我：在HotSpot虚拟机实现中，Java堆分成了新生代和老年代，我当时看的是1.7的实现，所有还有永久代，新生代中又分为了eden区和survivor区，survivor区又分成了S0和S1，或则是from和to，（这个时候，我要求纸和笔，因为我觉得这个话题可以聊蛮长时间，又是我比较熟悉的...一边画图，一边描述），其中eden，from和to的内存大小默认是8:1:1（各种细节都要说出来...），此时，我已经在纸上画出了新生代和老年代代表的区域

面试官：恩，给我讲讲对象在内存中的初始化过程？

我：（千万不要只说，新对象在Java堆进行内存分配并初始化，或是在eden区进行内存分配并初始化）要初始化一个对象，首先要加载该对象所对应的class文件，该文件的数据会被加载到永久代，并创

建一个底层的instanceKlass对象代表该class，再为将要初始化的对象分配内存空间，优先在线程私有内存空间中分配大小，如果空间不足，再到eden中进行内存分配...^&&*%

面试官：恩，好，说下YGC的大概过程...

我：先找出根对象，如Java栈中引用的对象、静态变量引用的对象和系统词典中引用的对象等待，把这些对象标记成活跃对象，并复制到to区，接着遍历这些活跃对象中引用的对象并标记，找出老年代对象在eden区有引用关系的对象并标记，最后把这些标记的对象复制到to，在复制过程还要判断活跃对象的gc年龄是否已经达到阈值，如果已经达到阈值，就直接晋升到老年代，YGC结束之后把from和to的引用互换（能多说点就多说点，省的面试官再提问，我把老年代的cms回收也大致说了一遍，以为面试官会跳过这个话题了，还是太年轻了）。

面试官：你刚刚说到在YGC的时候，有些对象可能会发生晋升，如果晋升失败怎么处理？

我：....（断片了几秒钟，我记得我分析过这段代码的，但是印象不深刻了）我记得在标记阶段时，会把对象和对应的对象头数据保存在两个栈中，如果晋升失败的话，就把该对象的对象头复原...

面试官：那你在实际项目中有碰到这种情况么，会导致什么问题？

我：...（这我真没有遇到过）对，有遇到过一次，在分析gc日志的时候，发现YGC发生之后，日志显示gc后的内存变大了，后来查出来是因为对象的晋升失败造成的。（我隐约记得看过笨神的一篇文章，回答的心里很虚）

面试官：（没有反驳，继续问）有过虚拟机性能调优的经验么？

我：（说实话，调优经验真的不多）恩，有一点吧，不是很足，就是我们XX项目上线的时候，发现YGC特别的频繁^^&^8&，通过调整新生代的大小（线上环境的虚拟机参数是默认的），同时检查业务逻辑代码&*&$$~~！

面试官：恩？还有么？

我：（面试这么久，好怕面试官的下一句是 “恩？还有么？”，显然面试官还不满足我的回答，但是我也只能答到这个地步了...）恩，经验确实有限，目前就根据这个项目做过一些相关的优化。

面试官：。。。。。。

我：。。。。。。

面试官： 那我们看看别的吧。

关于虚拟机方面的文章，我针对hotSpot的实现写了一些分析，感兴趣的同学可以看看，这些文章看着确实有点枯燥。

细节问题

细节决定成败，在面试过程中，虽然也有运气的成分存在，但是对于细节的掌握程度，可以很好的衡量应试者的技术水平。

volatile

volatile关键字很热门，面试很大概率会被问到

场景对话：

面试官：说说volatile关键字的实现原理

我：volatile关键字提供了内存可见性和禁止内存重排序

面试官：分别解释一下

我：因为在虚拟机内存中有主内存和工作内存的概念，每个cpu都有自己的工作内存，当读取一个普通变量时，优先读取工作内存的变量，如果工作内存中没有对应的变量，则从主内存中加载到工作内存，对工作内存的普通变量进行修改，不会立马同步到主内存，内存可见性保证了在多线程的场景下，保证了线程A对变量的修改，其它线程可以读到最新值&&%%……

面试官：如何保证的？

我：当对volatile修饰的变量进行写操作时，直接把最新值写到主内存中，并清空其它cpu工作内存中该变量所在的内存行数据，当对volatile修饰的变量进行读操作时，会读取主内存的数据&&&%%￥@

面试官：你知道系统级别是如何实现的么？

我：（what，what are u 说啥呢）我记得操作volatile变量的汇编代码前面会有**lock**前缀指令

面试官：你这说的还是代码层面，我说的是系统级别

我：（懵逼脸...）这个再底层下去我真的没研究过了...

Object.finalize()

面试官：和我讲讲Object类的finalize方法的实现原理

我：（完全没想到面试官会问这个）新建一个对象时，在JVM中会判断该对象对应的类是否重写了finalize方法，且finalize方法体不为空，则把该对象封装成Finalizer对象，并添加到Finalizer链表。

面试官：恩，然后呢？

我：Finalizer类中会初始化一个FinalizerThread类型的线程，负责从一个引用队列中获取Finalizer对象，并执行该Finalizer对象的runFinalizer方法，最终会执行原始对象的finalize方法，&&%%##（这块逻辑有点绕，当时答的也有点虚）

面试官：Finalizer对象什么时候会在引用队列中？

我：（努力回想中）在发生GC的时候，具体在什么时间点或如何被插入到引用队列中，这块实现我已经忘记了...（我真的忘记了，只记得这块逻辑太复杂了）

面试官：恩，你验证过finalize方法是否会执行么？

我：恩，自己写过例子证明过，也看过源码的实现。

面试官：怎么证明的？

我：初始化一个大数组，可以明显看出gc之后是否被回收，然后执行System.gc()，在finalize方法中输出信息 &&%%@@，（把之前做过的验证说一遍）

面试官：恩，可以...

大问题

什么是大问题，就是问题很大，让你自己去理解，把你的毕生所学都拿出来.

场景对话：

面试官：如果给你一个系统，如何去优化？

我：（优化什么？性能，稳定性，还是其它方面，只能硬着头皮上了，结合自己做的一个项目）
1、分析系统，定义指标
2、通过系统埋点，收集指标的度量值，对指标进行迭代优化&&^%&$#

面试官：就这些？没了么？

我：（因为是电话面试，感觉当时脑袋是空白的，估计和面试官的级别也有关系）如果指标是接口性能的话，可以看下系统内存是不是可以使用缓存进行性能上的优化，比如redis，如果是访问很频繁又不会经常变动的数据，如热点数据，可以直接使用本地缓存进行优化，毕竟一次网络请求也需要1~2毫秒

面试官：没了么？

我：（因为自己系统优化的经验确实不丰富，让面试官觉得怎么就只能想到如此少的优化点呢）数据库的读写分离，数据库的分库分表，如果经常条件查询数据库的话，可以引入搜索服务es或则lucene进行优化