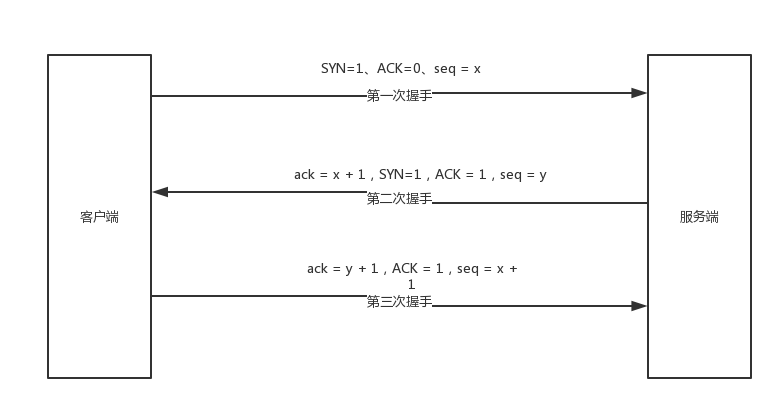
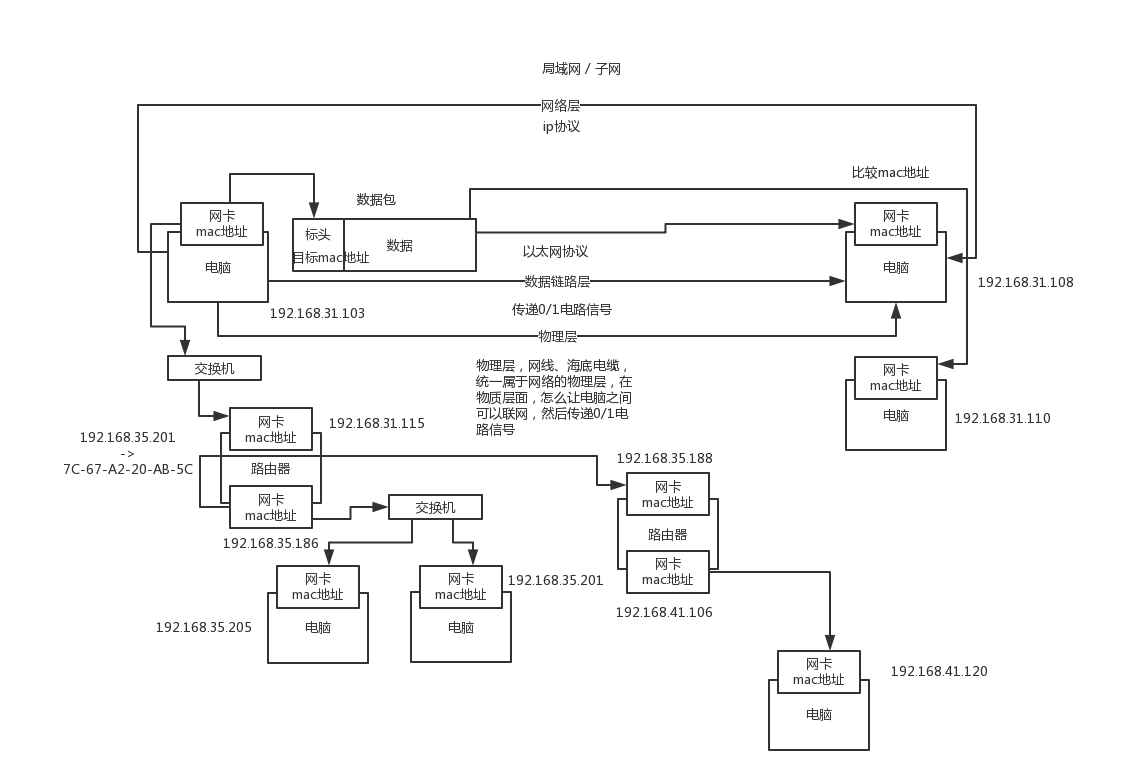
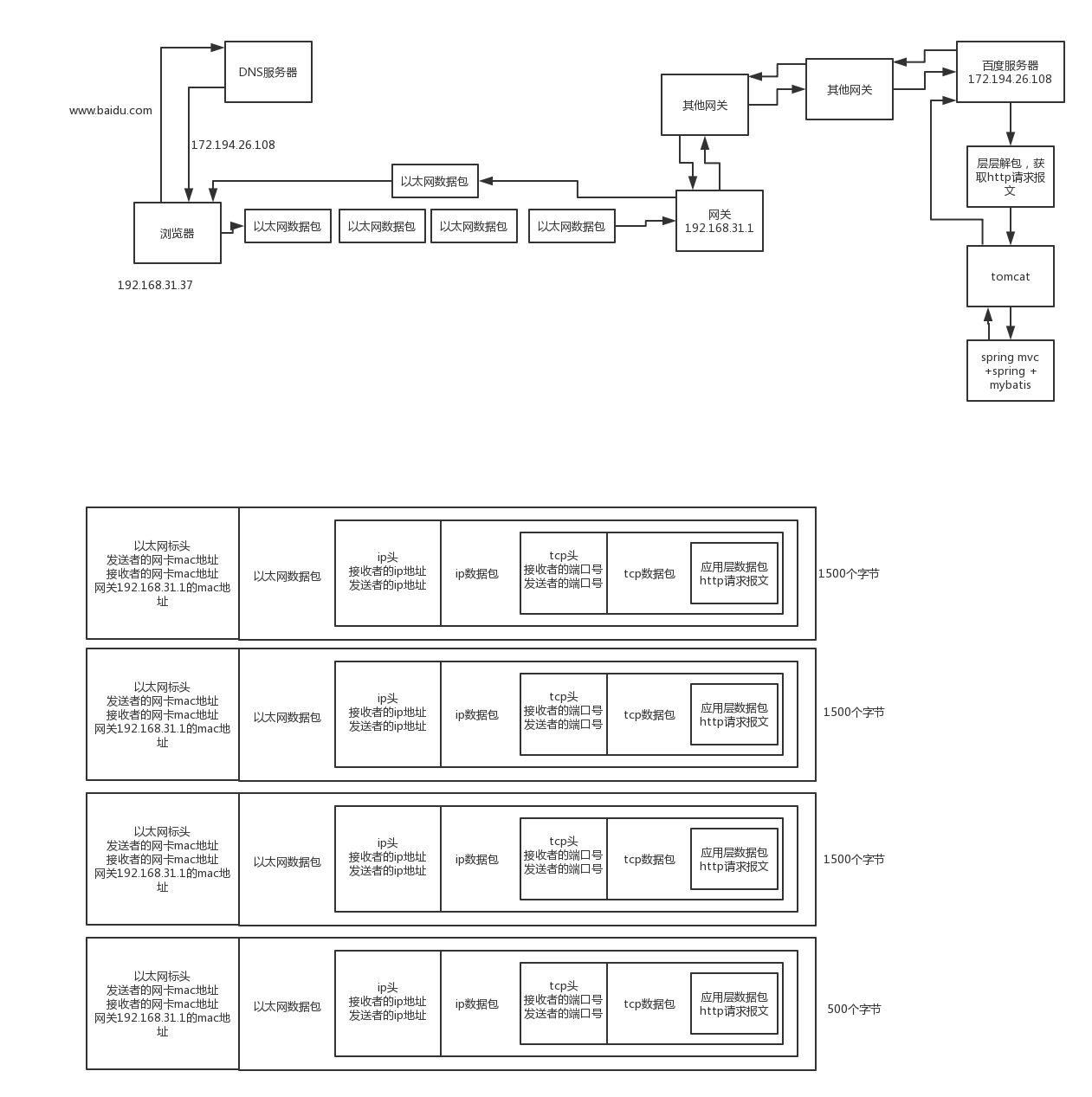
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/04\_画一下TCP三次握手的流程图？为啥是三次而不是二次或者四次呢？/01\_tcp三次握手.png**



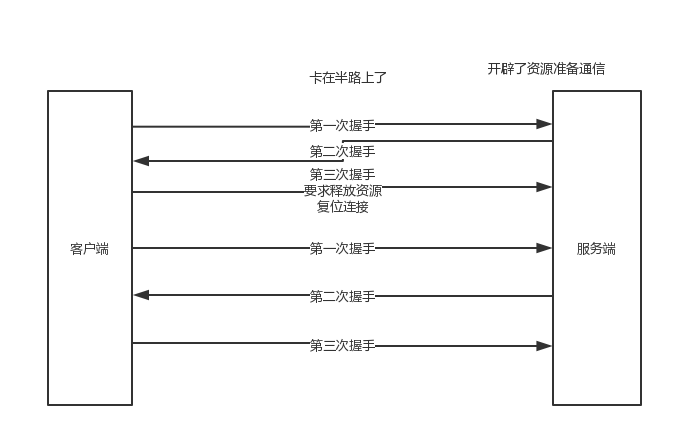
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/02\_聊聊TCP IP四层网络模型？那OSI七层网络模型呢？/01\_从底向上学习TCP\_IP四层模型.png**



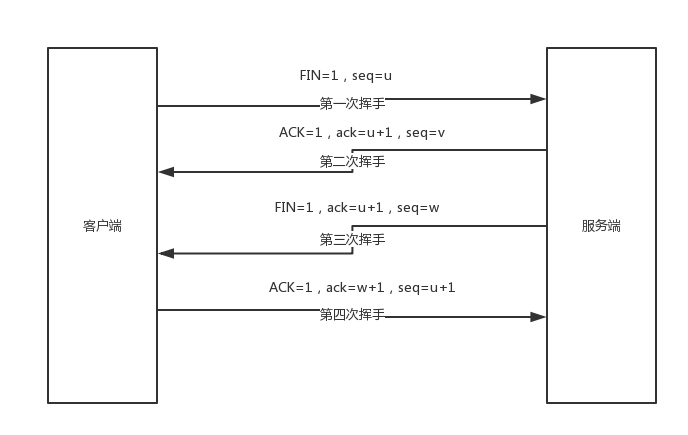
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/03\_浏览器请求www.baidu.com的全过程大概是怎么样的？/01\_浏览器发送一次请求的过程.png**



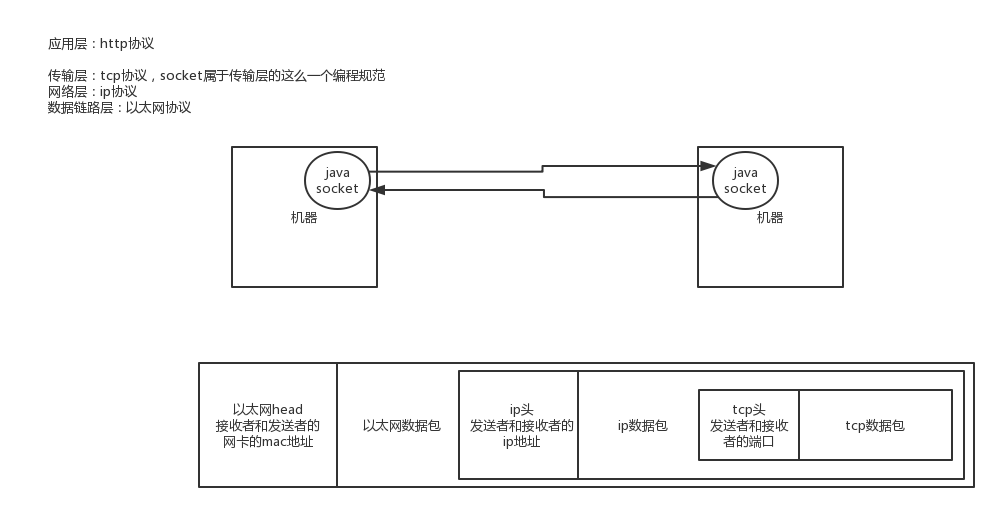
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/04\_画一下TCP三次握手的流程图？为啥是三次而不是二次或者四次呢？/02\_tcp连接为什么是3次握手.png**



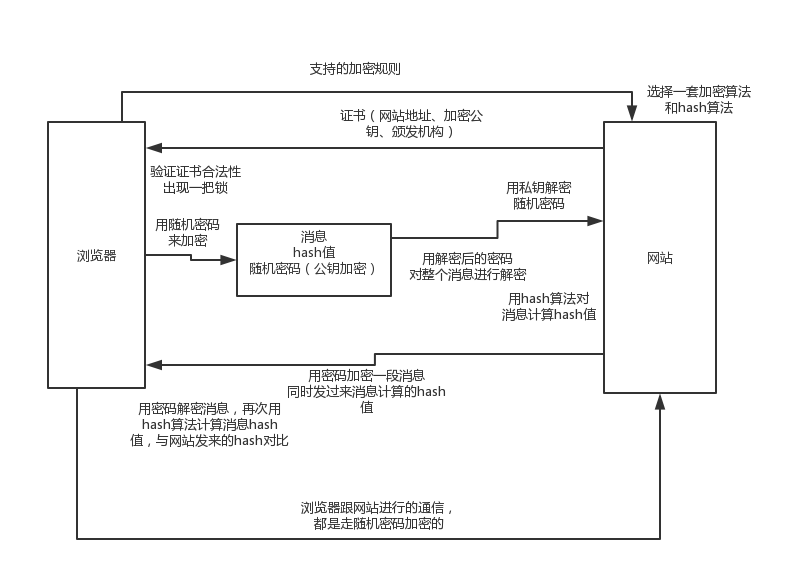
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/04\_画一下TCP三次握手的流程图？为啥是三次而不是二次或者四次呢？/03\_tcp的4次挥手断开连接.png**



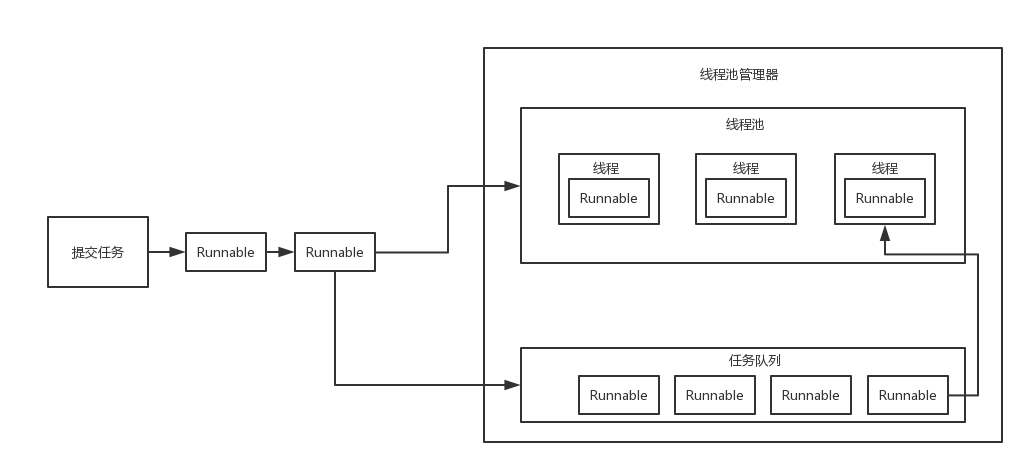
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/05\_聊聊Socket的工作原理？Socket跟TCP IP之间是啥关系？/05\_01\_socket工作原理.png**



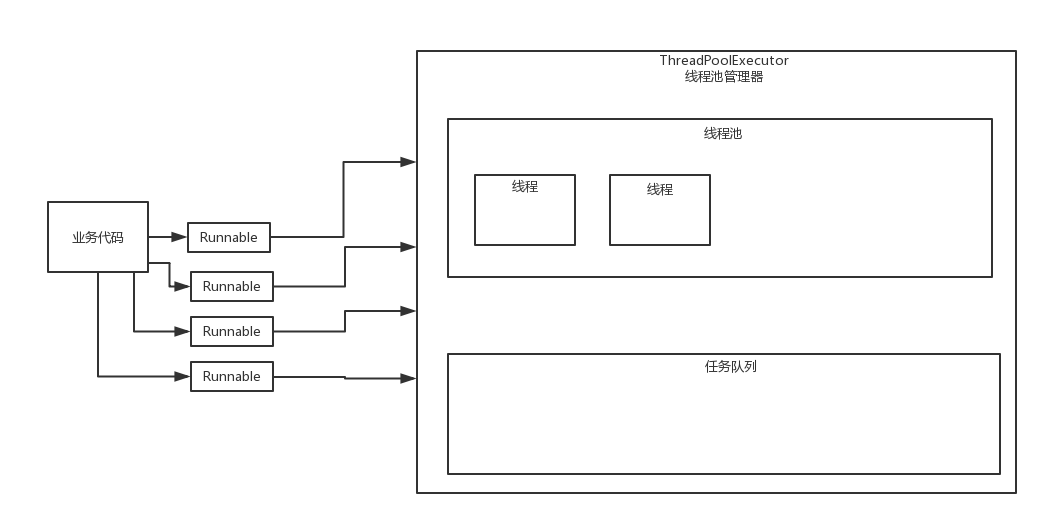
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/07\_聊聊HTTPS的工作原理？为啥用HTTPS就可以加密通信？/07\_01\_https原理.png**



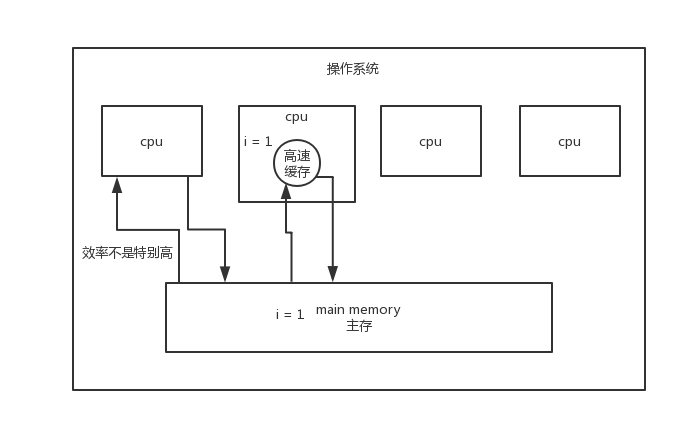
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/14\_如何抗住面试官的线程池N个连环炮的发问/14\_01\_线程池的最最基本的工作原理.png**



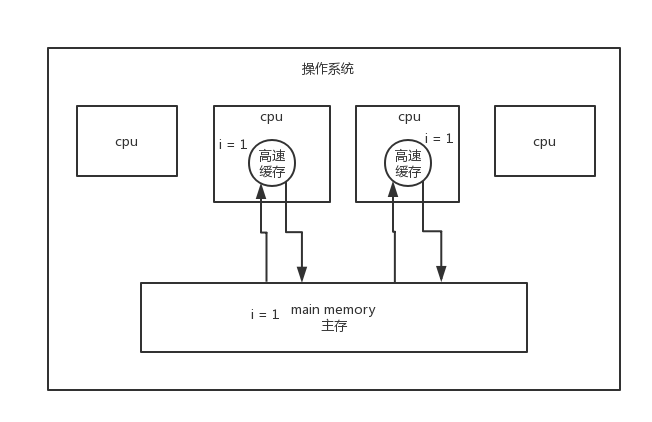
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/14\_如何抗住面试官的线程池N个连环炮的发问/14\_02\_线程池的真真正正的工作原理.png**



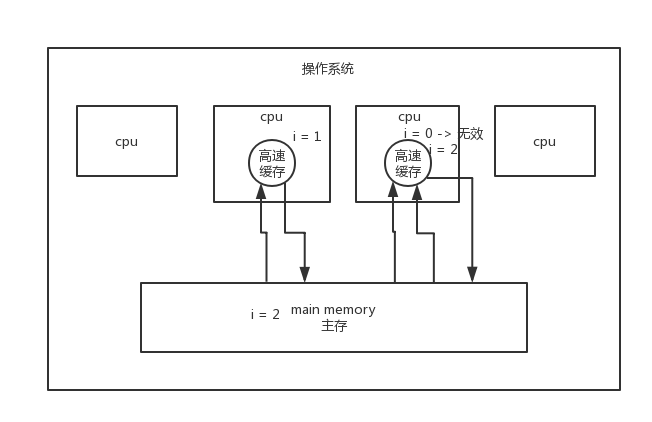
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/16\_能不能结合java内存模型来聊聊volatile关键字的原理？/16\_01\_初探操作系统的内存模型.png**



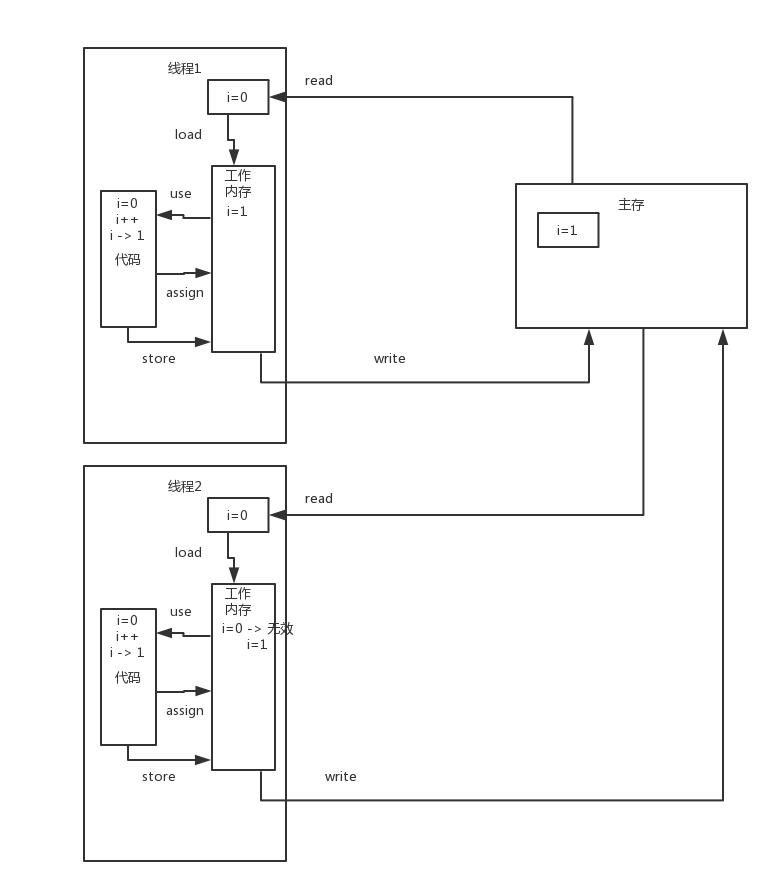
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/16\_能不能结合java内存模型来聊聊volatile关键字的原理？/16\_02\_操作系统的内存模型的并发问题.png**



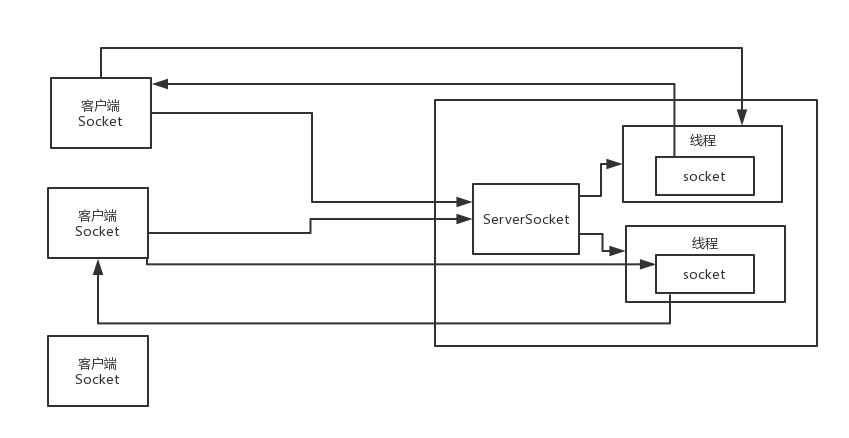
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/16\_能不能结合java内存模型来聊聊volatile关键字的原理？/16\_03\_操作系统的缓存一致性协议.png**



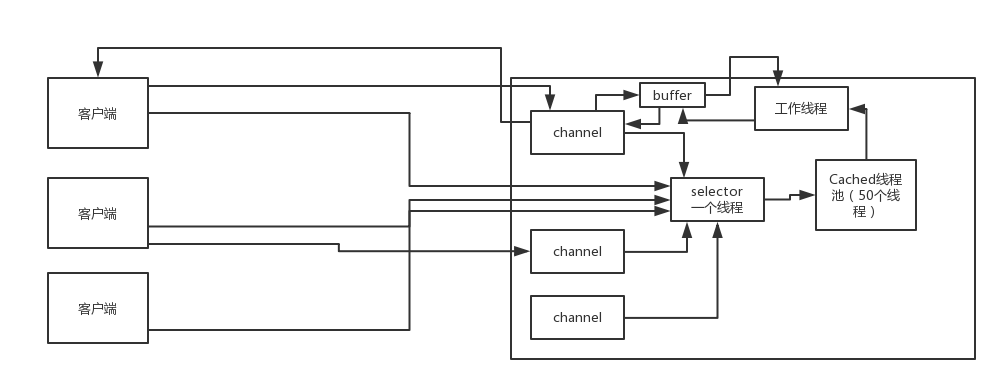
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/16\_能不能结合java内存模型来聊聊volatile关键字的原理？/16\_04\_Java内存模型.png**



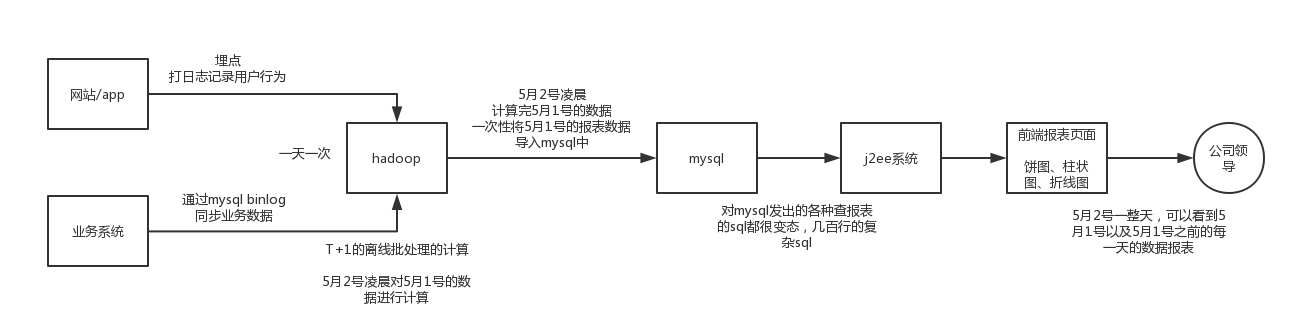
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/21\_你能聊聊BIO、NIO、AIO分别都是啥？有什么区别？/21\_01\_BIO的网络通信原理.png**



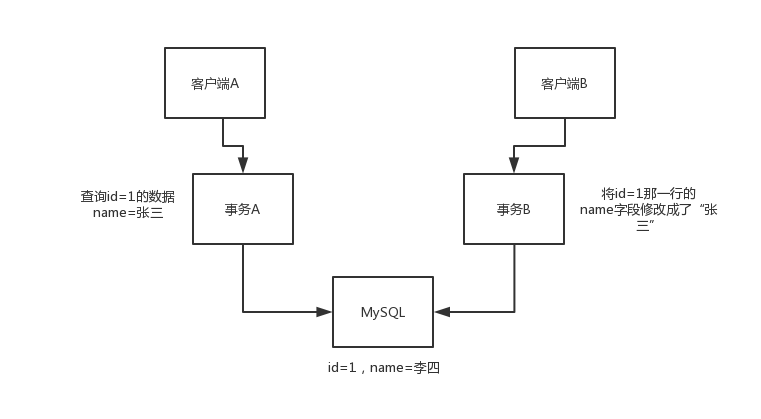
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/21\_你能聊聊BIO、NIO、AIO分别都是啥？有什么区别？/21\_02\_NIO通信原理.png**



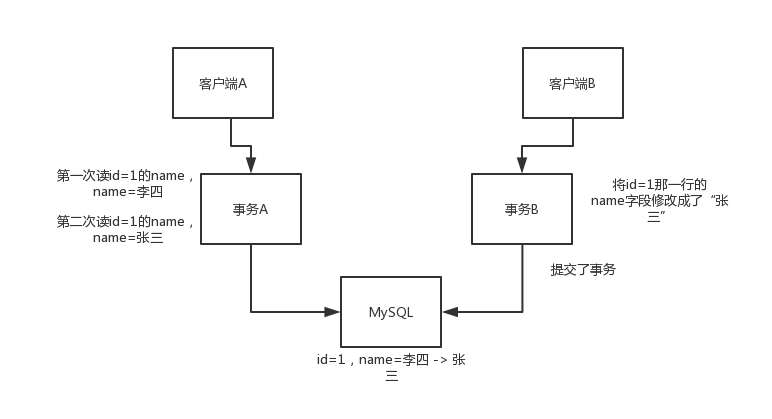
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/25\_MySQL MyISAM和InnoDB存储引擎的区别是啥？/25\_01\_基于myisam存储引擎做报表系统.png**



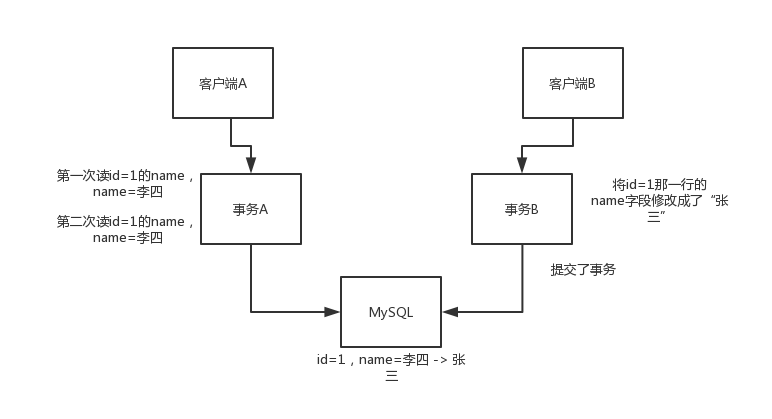
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/27\_你能说说事务的几个特性是啥？有哪几种隔离级别？/27\_01\_读未提交.png**



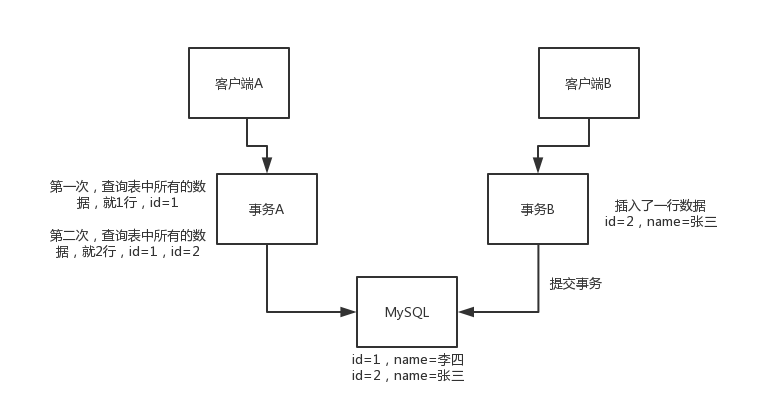
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/27\_你能说说事务的几个特性是啥？有哪几种隔离级别？/27\_02\_读已提交.png**



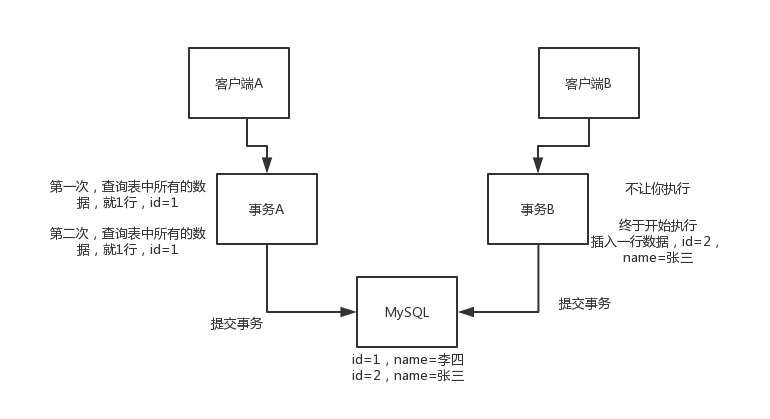
# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/27\_你能说说事务的几个特性是啥？有哪几种隔离级别？/27\_03\_可重复读.png**



# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/27\_你能说说事务的几个特性是啥？有哪几种隔离级别？/27\_04\_幻读.png**



# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/27\_你能说说事务的几个特性是啥？有哪几种隔离级别？/27\_05\_串行化.png**



# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/16\_能不能结合java内存模型来聊聊volatile关键字的原理？/笔记.docx**

1、面试题

java的内存模型是什么？能结合内存模型说一下volatile的工作原理吗？指令重排序，内存栅栏，happen-before等概念是指的什么意思？

2、面试官心里分析

我作为面试官的话，也是经常问volatile的，确实是，但是确实比较凸显一个人的java基本功，而且一般聊到并发编程这块，一般都会问一下这个问题。说实话，我遇到过的人里，能把volatile讲清楚的很少很少；你如果去网上查文章和博客，能把volatile看明白的，我觉得你很厉害，文章或者是博客，几乎我没见过一个真正讲明白的，写的很好的。

这个volatile，说实话吧，不推荐大家大量使用，为什么呢？因为volatile背后的原理还是比较高级的，如果你用了，有两种情况会发生：第一种，你很牛，你明白volatile啥意思，用了，没问题；第二种，你就略微懂一点点，结果还瞎用volatile，就导致出问题。

但是无论是哪种情况用了volatile，都有一个很大的问题，对于你研发普通的业务系统来说，要明白一个道理，铁打的硬盘，流水的兵，可能以后你都走了，结果某个人来接替你，他可不一定熟悉volatile这个东西，那可能他维护的时候就会出问题。

所以就我个人经验来说，我不推荐在业务系统里用volatile，你可以找别的方式来保证并发安全性，但是如果是那种底层框架或者分布式系统研发团队，里面个个都是牛人，走了一个，进来的也是比较厉害的，这种精英团队，可以用volatile。

面试的时候，volatile还是很高频的会问到的，所以大家知道肯定要知道，自己要结合内存模型能回答一下，但是你说用，但是还是少用。

3、面试题

**3.1 操作系统内存模型**

先聊下内存模型这个事儿。

操作系统这个层面上，所有指令都是cpu里执行的，但是执行指令的时候肯定是要读写数据的，数据一般都放主存里，但是直接读写内存速度不够快，所以一般会在cpu里放高速缓存，就是说主存的数据会放一份在cpu内部的高速缓存中，这样cpu指令直接对cpu内部的高速缓存里的数据读写就可以了。

举个例子，i++

但是这样就会带来数据一致性的问题，因为每次cpu都是从内存读取数据到cpu高速缓存，操作完以后，再写回主存里。比如i++，这个操作吧，一般就是从主存读取i（i = 1）到cpu高速缓存里，然后对这个值累加，i = 2，先写入高速缓存，接着写回主存。

但是如果是多线程，可不是这么玩儿的，多线程的时候，在多cpu场景下，可能两个线程会将主存的i = 1都读到高速缓存里，然后都累加，i = 2，接着写回自己的高速缓存，然后刷回主存，此时就会导致主存里的数据是i = 2，而不是我们期望的i = 3。

可以用总线lock锁机制，这个上一讲还提到过，但是这个机制其实很重，会导致只有一个cpu可以操作主存，别的cpu都没法操作了。所以最新的cpu里，都不会用总线锁了，一般都会用优化的缓存一致性协议（MSI），就是某个cpu写数据的时候，发现写的是共享变量，会通知其他cpu这个数据在他们内部的高速缓存是无效的，让其他cpu读这个变量的时候从写数据的那个cpu缓存里读取，保证大家的缓存是一致的。当然，也有一种实现，是说修改变量的线程将数据从自己缓存立即刷新到内存，然后其他cpu发现自己的缓存行失效，如果要读写数据的时候重新从内存里加载到缓存里来。

并发问题里，有3个至关重要的点，原子性、可见性、有序性

**（1）原子性**

这个很简单，就是说比如i++这个操作吧，根据操作系统内存模型，是需要先从主存中读取数据到cpu高速缓存中，然后给它加1后写入高速缓存，最后再从高速缓存刷入主存，这个过程是有很多个步骤的。如果这个过程能保证绝对可以执行成功，不会出现任何意外，那么就是原子性。

你可以保证并发操作的原子性，无论多少个线程怎么i++，最后都是不断累加1的，数据不会错，就是原子性的

**（2）可见性**

//线程1

int i = 1;

i = 5; -> 还放在高速缓存里，还没写入主存

//线程2

int j = i; -> 从主存加载i的值，还是1，将i = 1的值，赋值给了j，j = 1

比如上面的代码，如果两个线程在不同的cpu运行，此时可能会出现说，线程1将i的值赋值为1，然后写入了cpu高速缓存，同时可能写入了主存；接着将i再次更新为5，然后写入了cpu高速缓存，但是还没来得及写入主存；此时线程2要将i的值赋值给j，那么就会从主存中读取i的值，此时还是1呢，所以就会将1这个值赋值给j

这就是不可见了，因为线程1做的i = 5的修改停留在cpu高速缓存里，还没来得及写入主存，导致线程2没看到。

**（3）有序性**

同时还有一个问题是指令重排序，编译器和指令器，有的时候为了提高代码执行效率，会将指令重排序，就是说比如下面的代码

//线程1:

prepare(); // 准备资源

flag = true;

//线程2:

while(!flag){

Thread.sleep();

}

execute(); // 基于准备好的资源执行操作

重排序之后，让flag = true先执行了，会导致线程2直接跳过while等待，执行某段代码，结果prepare()方法还没执行，资源还没准备好呢，此时就会导致代码逻辑出现异常。

**3.2 java内存模型**

Java内存模型，规定的东西跟操作系统是相关的，规定了所有变量的值都在主存中，每个线程都有自己的工作内存（类似前面说的cpu高速缓存），线程对变量的操作都必须在工作内存中完成，是不能直接对主存进行操作的，而且每个线程不能访问其他线程的工作内存，其实在上面都能找到对应的概念。

所以先熟悉了操作系统内存模型，再看java内存模型，就能看得懂了。

另外，你要明白一个线程对工作内存和主存的操作模型，read（从主存读取），load（将主存读取到的值写入工作内存），use（从工作内存读取数据来计算），assign（将计算好的值重新赋值到工作内存中），store（将工作内存数据写入主存），write（将store过去的变量值赋值给主存中的变量）

**（1）原子性**

举个例子，i = 1，必须线程先在自己的工作内存中将i赋值为1，然后再写入主存中，不能直接修改主存的。而且java中仅仅保证i = 1这种基本数字类型的赋值操作，是原子的，如果是什么i = y，i++这种，都需要先将某个值从主存读入工作内存，操作过后再写回主存，都不是原子的。

voaltile是不能保证原子性的，这个稍后来分析。

原子性，实际上只能靠synchronized和lock来解决，就是仅仅在一段时间内允许一个线程获取锁，然后执行某段代码，操作完之后强制将工作内存数据刷回主存，其他线程获取锁之后立马可以看到数据然后使用。

**（2）可见性**

volatile是用来保证可见性的，就是加了volatile关键字的变量，你在修改之后，会立即刷入主存，接着其他线程会感知到强制从主存读取最新的值，可以保证可见性。

**（3）有序性**

java中有一个happens-before原则：

程序次序规则：一个线程内，按照代码顺序，书写在前面的操作先行发生于书写在后面的操作

锁定规则：一个unLock操作先行发生于后面对同一个锁额lock操作

volatile变量规则：对一个变量的写操作先行发生于后面对这个变量的读操作

传递规则：如果操作A先行发生于操作B，而操作B又先行发生于操作C，则可以得出操作A先行发生于操作C

线程启动规则：Thread对象的start()方法先行发生于此线程的每个一个动作

线程中断规则：对线程interrupt()方法的调用先行发生于被中断线程的代码检测到中断事件的发生

线程终结规则：线程中所有的操作都先行发生于线程的终止检测，我们可以通过Thread.join()方法结束、Thread.isAlive()的返回值手段检测到线程已经终止执行

对象终结规则：一个对象的初始化完成先行发生于他的finalize()方法的开始

上面这8条原则的意思很显而易见，就是程序中的代码如果满足这个条件，就一定会按照这个规则来保证指令的顺序。

但是如果没满足上面的规则，那么就可能会出现指令重排，就这个意思。这8条原则是避免说出现乱七八糟扰乱秩序的指令重排，要求是这几个重要的场景下，比如是按照顺序来，但是8条规则之外，可以随意重排指令。

**3.3 volatile的作用**

**（1）volatile对可见性的影响**

volatile run = false;

// 线程1，先执行

run=true // 已经变成1了

// 线程2，后执行

// 读取到的还是0

while(!run) {

// 无限循环

}

如果不用volatile，可能导致run变量的修改对其他线程是不可见的，所以上面的代码，不一定能保证说修改了run变量的值之后，理解让线程2跳出来

但是只要你加了volatile关键词，保证了每个线程对一个变量的修改，立即都是对其他线程是可见的，只要线程1一旦修改了run = true，然后理解就是其他线程会看到最新的值，然后while循环就会立即跳出

就是某个线程对一个变量先修改了值，然后另外一个线程去读取那个变量一定会看到最新的值，这个叫做可见性

有的时候我们会用上面的代码来让一个线程控制另外一个线程终止无限循环然后结束，但是如果直接这么搞可能就是有问题的，因为线程1读取了run到自己工作内存，然后依据工作内存中的run=true不断的循环；而线程2虽然将run的值修改为了false而且写入了主存，但是线程1可能确实就是没感知到，因为还是一直在依赖自己工作内存中的run = true在工作。

如果加了volatile关键字之后，可以保证的是，一个线程对数据的写操作，会立即同步到主存，其他线程也是直接从主存来读取的，就是忽略了cpu高速缓存的作用了。因为加了volatile之后，相当于是在cpu层面加了lock指令。这个lock指令，会基于缓存一致性协议来实现，让cpu写数据到高速缓存之后，立即同步到主存，同时其他cpu发现这个共享数据被修改之后，就会标记自己高速缓存中的那个数据失效了，那么如果要对这个失效数据进行修改的时候，会强制重新从内存里把这个数据读到自己缓存里来，再进行修改。

比如上面的场景，如果对i这个共享变量加了volatile关键字之后，线程1修改i++，会强制立即将工作内存中的值刷入主存，而且会导致线程2的工作内存中的i变量的缓存行无效，因为线程2的i变量缓存行无效了，此时就会重新从主存中读取run变量的最新值，就是i = 1。

所以说，volatile是可以保证共享变量的可见性的。

只要是对volatile变量并发操作

某个线程，先修改了值，立即刷回主存；如果另外一个线程此时再来读，会直接从主存里读到最新的值；如果值钱另外一个线程已经将这个变量的值读到自己的工作内存里去了，此时其他线程要从工作内存里读的时候，会发现那个变量的缓存行已经失效了，此时会重新从主存里加载最新的值

**（2）volatile对原子性的影响**

网上没有一篇文章对volatile关于原子性的事情说清楚了，其实坦白说，很多人完全不理解volatile的作用，所以出现了瞎用的情况，比如说下面的例子：

volatile int i = 0

// 线程1

i++

// 线程2

i++

如果说你以为volatile是轻量级锁的人，很傻X，我个人是绝对不认可volatile是一把锁；可以保证多线程并发修改一个共享资源的话 -> JDK傻乎乎的搞了什么synchronized、ReentrantLock、Semaphore。

如果你以为加了个volatile就是可以保证万无一失，那你真的是too young too simple

如果某一时刻，两个线程都先后将这个i的值，i = 0加载到工作内存里来了，结果可能线程1都执行到use步骤了，此时已经将i = 0弄出来准备加1了；同时线程2在这个瞬间就完成了use、assign、store、write等步骤，i = 1，同时写回了主存，并且让线程1的工作内存缓存行失效了；但是然而有什么用呢？线程1此时不需要工作内存了，之前use已经从工作内存读取了数据，所以也同时将i加1变成了1，然后接着assign、store、write回了主存，所以相当于写了两次1回主存。

看明白了java内存模型，你就理解了所谓的可见性是啥，原子性是啥了。

可见性是说，你可能某一次读取是错的，但是下一次再次读取的时候，一定会发现失效的缓存行，重新从主存去read和load进行最新的值；原子性是说，你还是可能会出现某一次读取到的值是旧值，没人告诉你一定能保证100%没问题。

正是对这块的不理解，导致几乎没几个程序员能用好volatile关键字的。

**（3）volatile对有序性的影响**

volatile可以在一定程度上禁止部分指令重排序

//线程1:

prepare(); // 准备资源

volatile flag = true;

//线程2:

while(!flag){

sleep()

}

execute(); // 基于准备好的资源执行操作

比如这个例子，如果用volatile来修饰flag变量，一定可以让prepare()指令在flag = true之前先执行，这就禁止了指令重排。因为volatile要求的是，volatile前面的代码一定不能指令重排到volatile变量操作后面，volatile后面的代码也不能指令重排到volatile前面。

看完了上面的东西，就可以最后说一说这几个概念了，指令重排已经说过了，编译器和指令器为了效率，有的时候会对指令重新排序，但是指令重排的时候一定会遵守happens-before这套规则，如果代码不符合happens-before规则，那么就可以随意进行指令重排，否则就要按照happens-before规则的顺序来走。

内存屏障，其实就是volatile关键字加了之后，就会加一个操作系统层面的lock指令，这就形成了一个所谓内存屏障，或者是内存栅栏吧。其实就是起到一个作用，一个是禁止指令重排，就是上面说的；另外就是说写的时候一定会强制刷主存；写完之后一定会导致其他cpu的高速缓存失效

**3.4 volatile的使用场景**

我觉得说到这里，大家基本上对于如何用volatile关键字就呼之欲出了吧

说白了，就看看上面那个例子，如果说，你可以忍受没有原子性，但是只要保证可见性就好，比如while(run)的时候，某一次run的值没读到最新的是ok的，但是只要下一次读一定能立马读到最新值就好了，那用volatile其实很合适。

如果你要的是i++这种复杂操作要实现线程并发安全，那要么是CAS的AtomicInteger这种，要么是synchronized或者lock加锁，因为只有加锁是能禁止别的线程执行这段操作的。用volatile是不靠谱的。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/23\_提醒一下自行复习集合的常见基础面试题/笔记.docx**

（1）hashmap是如何扩容的？每次扩容多少？已有元素如何处理？说一下hashmap底层的数据结构以及工作原理？hashmap如何处理hash碰撞问题以及key重复问题？什么情况下hashmap会内存泄露？hashmap是否是线程安全的？ConcurrentHashMap的实现原理？treemap的实现原理是什么？linkedhashmap的工作原理？hashmap和hashtable底层实现有什么区别？hashtable和concurrenthashtable有什么区别呢？hashmap和treemap有什么区别？

（2）Java中的队列都有哪些？这些队列的区别是什么？ConcurrentLinkedQueue的原理是什么？

（3）java集合框架中哪些是线程安全的，哪些是线程不安全的？concurrent包里有哪些类？

这些东西，每一块，自行找一些博客和文章，自己看一下，了解一下原理

基础，自己去看

集合这块，真正牛的一点在哪儿呢？JDK源码，架构班后续，有专门的一块看JDK源码的课程，会给大家去讲解集合这块的源码

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/33\_提醒复习一下Tomcat的原理以及参数配置等基础知识/笔记.docx**

1、面试题

tomcat的架构、类加载流程以及线程模型是什么？tomcat如何进行jvm配置？tomcat如何优化以及有哪些参数？

2、面试官心里分析

说实话，这块如果玩儿深了，会很深，要讲很多，而且tomcat本身精通对java架构师来说非常重要，但是不是在这里花费时间去讲解，在我们的高并发那个环节，我们会非常深入的讲解tomcat、nginx等技术，然后给大家讲解平时如何配置参数去优化并发、性能，以及他们的内部原理，等等。

3、面试题剖析

但是这里，作为面试突击常见的就是一些简单的问题，我自己看了下，这些基础问题，大家搜几篇博客，一般都能自己搞定，就花费几个小时看一下就行了，面试，这些都不是重点，都是说，最后问你个问题，看你是否了解这个东西。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/09\_进程间是如何通信的？线程间又如何切换呢？/笔记.docx**

1、面试题

进程间是如何通信的？线程间又如何切换呢？

2、面试官心里分析

这个问题不是高频基础问题，但是确实可能有人会问，因为怎么说呢，计算机基础，就这点儿东西，网络、cpu、磁盘、内存、进程，所以可能有人会看看你的基础知识咋样，所以问问你这个问题。

3、面试题剖析

进程间的通信有很多种方式，比如说：管道（pipe）、命名管道（fifo）、消息队列，共享内存（System V）

（1）管道（pipe）

unix操作系统里面，有一个fork操作，可以创建进程的子进程，或者说是复制一个进程完全一样的子进程，共享代码空间，但是各自有独立的数据空间，不过子进程的数据空间是拷贝父进程的数据空间的。

管道机制要求的是两个进程之间是有血缘关系的，就比如fork出来的父子进程。

linux操作系统里，管道用来缓存要在进程间传输的数据，管道是一个固定大小的缓冲区，是4kb。管道中的数据一旦被读取出来，就不在管道里了。但是如果管道满了，那么写管道的操作就阻塞了，直到别人读了管道的数据；反之如果管道是空的，那么读操作就阻塞了。就这个意思。管道一边连着一个进程的输出，一边连着一个进程的输入，然后就一个进程写数据，另外一个进程读数据，两个进程都没了，管道也就没了。管道是半双工的，就是数据只能流向一个方向，比如说你架设一个管道，只能一个进程写，另外一个进程读。

linux里面对管道的实现，是用了两个文件，指向了一个VFS（虚拟文件系统）的索引节点inode，然后VFS索引节点指向一个物理页面，接着一个进程通过自己关联的那个文件写数据，另外一个进程通过自己关联的那个文件读数据。

（2）命名管道（fifo）

管道的通信，要求必须是父子关系的进程间通信，就受到了限制，所以可以用命名管理来解决这个问题。

之前的管道，是没有名字的，所以必须是有父子关系的进程才能使用。但是这个命名管道是有名字的。这个命名管道，相当于是一个有名字的文件，是有路径的，所以没有血缘关系的进程多可以通过这个命名管道来通信，名字在文件系统上，数据在内存里。其他的跟管道一样，一个进程写，一个进程读，也是半双工的，数据只能单向流动。

（3）消息队列

linux的消息队列可以认为是个链表结构，linux内核有一个msgque链表，这个链表里每个指针指向一个msgid\_ds结构，这个结构就描述了一个消息队列。然后进程之间就通过这个消息队列通信就可以，一样是写入数据和消费数据。消息队列的好处就是对每个消息可以指定类型，消费的时候就消费指定类型的消息就行了，功能更多一些。这种方式其实用的不多的。

（4）共享内存

一块物理内存被映射到两个进程的进程地址空间，所以进程之间互相都可以立即看到对方在共享内存里做出的修改，但是因为是共享内存，所以需要锁来保证同步。这个说对了很复杂，我在这里就不多说了，我觉得如果被人问到这个问题，短期内突击的话，回答到这个程度就行了，就是知道有哪些方式。如果你要深入理解各种机制，那是要好好学习linux的各种东西了。

（5）线程间如何切换

一个进程的多个线程间切换的时候就涉及到了上下文切换，这个东西说复杂了就很复杂，但是简单来说，就是有一个时间片算法，cpu给每个线程一个时间片来执行，时间片结束之后，就保存这个线程的状态，然后切换到下一个线程去执行，这就是所谓多线程并发执行的原理，就是多个线程来回来去切换，每个线程就一个时间片里执行。太复杂的我也不讲了，大家就记住一个线程上下文切换指的是什么就行了。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/08\_聊聊http的长连接的工作原理到底是啥？/笔记.docx**

1、面试题

什么是长连接？http长连接是什么？

2、面试官心里分析

一期学员，在外面面试的时候，正好还碰到了，聊到dubbo，dubbo://协议，是走的长连接，你聊聊什么是长连接？什么是http长连接？

3、面试题剖析

http本身没什么所谓的长连接短连接之说，其实说白了都是http下层的tcp连接是长连接还是短连接，tcp连接保持长连接，那么多个http请求和响应都可以通过一个链接来走。其实http 1.1之后，默认都是走长连接了，就是底层都是一个网页一个tcp连接，一个网页的所有图片、css、js的资源加载，都走底层一个tcp连接，来多次http请求即可。

http 1.0的时候，底层的tcp是短连接，一个网页发起的请求，每个请求都是先tcp三次握手，然后发送请求，获取响应，然后tcp四次挥手断开连接；每个请求，都会先连接再断开。短连接，建立连接之后，发送个请求，直接连接就给断开了

http 1.1，tcp长连接，tcp三次握手，建立了连接，无论有多少次请求都是走一个tcp连接的，走了n多次请求之后，然后tcp连接被释放掉了

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/07\_聊聊HTTPS的工作原理？为啥用HTTPS就可以加密通信？/笔记.docx**

1、面试题

http和https的区别是什么？https的原理是什么？

2、面试官心里分析

聊到http了，那肯定会聊聊https

3、面试题剖析

http协议都是明文的，是没有加密的，所以其实现在一般大部分应用都是用https协议的。之前是基于SSL协议对http进行加密，后来又升级到了TSL协议来加密，现在称之为SSL / TSL吧。

https的工作原理大概是这样的：

（1）浏览器把自己支持的加密规则发送给网站

（2）网站从这套加密规则里选出来一套加密算法和hash算法，然后把自己的身份信息用证书的方式发回给浏览器，证书里有网站地址、加密公钥、证书颁发机构

（3）浏览器验证证书的合法性，然后浏览器地址栏上会出现一把小锁；浏览器接着生成一串随机数密码，然后用证书里的公钥进行加密，这块走的非对称加密；用约定好的hash算法生成握手消息的hash值，然后用密码对消息进行加密，然后把所有东西都发给网站，这块走的是对称加密

（4）网站，从消息里面可以取出来公钥加密后的随机密码，用本地的私钥对消息解密取出来密码，然后用密码解密浏览器发来的握手消息，计算消息的hash值，并验证与浏览器发送过来的hash值是否一致，最后用密码加密一段握手消息，发给浏览器

浏览器解密握手消息，然后计算消息的hash值，如果跟网站发来的hash一样，握手就结束，之后所有的数据都会由之前浏览器生成的随机密码，然后用对称加密来进行进行加密。

常用的非对车呢加密是RSA算法，对称加密是AES、RC4等，hash算法就是MD5

就好比，有个人说我加密的时候是用了一个公钥去加密，然后你解密的时候是用私钥去解密；我加密的时候用的算法，跟解密的时候用的算法，是一样的，对称加密

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/10\_线上服务器CPU 100%了！该怎么排查、定位和解决？/笔记.docx**

1、面试题

线上服务器的cpu使用达到100%了，如何排查、定位和解决该问题？

2、面试官心里分析

说实话，这个问题是面试的时候，聊基础，最常问的一个问题，就是看看你有没有处理过高负载的线上问题场景。所以很多大公司考察你的基本功，肯定会问这个。其实这个你干过就是干过，掌握就是掌握，只要干过，所有人都是一样的步骤，没区别。

3、面试题剖析

其实核心思路，就是找到这台服务器上，是哪个进程的哪个线程的哪段代码，导致cpu 100了，主要就是考察你是否熟练运用一些线上的命令。

这里我可以给大家说一个我们线上的经验，就是之前有一个bug，是一个很年轻的同学写的，就是我们当时是定了异常日志是写到es里去的

public void log(String message) {

try {

// 往es去写

} catch(Exception e) {

log(message);

}

}

线上事故，es集群出了点问题，没法写，最后出现线上几十台机器，全部因为这一行代码，全体cpu 100%，卡死了

（1）定位耗费cpu的进程

top -c，就可以显示进程列表，然后输入P，按照cpu使用率排序，你会看到类似下面的东西

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

43987 root 20 0 28.2g 4.5g 68m S 99.0 24.0 44333.4 java -Xms。。。

大概类似上面这样，能看到哪个进程，CPU负载最高，还有启动这个进程的命令，比如一般就是java啥啥的。

（2）定位耗费cpu的线程

top -Hp 43987，就是输入那个进程id就好了，然后输入P，按照cpu使用率排序，你会看到类似下面的东西

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

16872 root 20 0 28.2g 4.5g 68m S 95.0 12.0 65543.3 java

大概类似上面那样，你就可以看到这个进程里的哪个线程耗费cpu最高

（3）定位哪段代码导致的cpu过高

printf “%x\n” 16872，把线程pid转换成16进制，比如41e8

jstack 43987 | grep ‘0x41e8’ -C5 --color

这个就是用jstack打印进程的堆栈信息，而且通过grep那个线程的16进制的pid，找到那个线程相关的东西，这个时候就可以在打印出的代码里，看到是哪个类的哪个方法导致的这个cpu 100%的问题

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/12\_线上机器的磁盘空间快满了应该怎么处理？/笔记.docx**

1、面试题

服务器存储空间快满了（95%），还有一个小时存储就满了，在不影响服务正常运行的情况下，该如何解决？

2、面试官心里分析

这个确实没什么好说的，无非就是用一些一些线上的场景和问题来考考你平时一般怎么处理的，线上机器磁盘满，一般啥原因，不就是日志太多了给写满了么。。。对吧，我们不说别的，就说说这最基本的就行了

3、面试题剖析

df -h，先看看磁盘使用的情况

然后就是到你的系统部署的地方，一般就是tomcat下的日志、spring boot的日志，去看看，如果过多，就删除掉一些日志就行了，自己注意让tomcat或者nginx之类的日志输出，按天切割，这样你还可以写个shell脚本，crontab定时，定期删除7天以前的日志

要是不行，那就：find / -size +100M |xargs ls -lh，找找大于100m的文件，但是如果有大量的小文件，那么这样是不行的

或者是用：du -h >fs\_du.log，看看各个目录占用的磁盘空间大小，看看是不是哪个目录有大量的小文件

其实面试官无非就是看看是不是知道常见的命令罢了，如果不是。那那个面试官就得再提示多一些细节，到底要考察你什么。但是简单问一个磁盘占用排查，就是常见这几个命令罢了。ok。。。。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/26\_聊聊MySQL的索引实现原理？各种索引你们平时都怎么用的？/笔记.docx**

1、面试题

MySQ索引的原理和数据结构能介绍一下吗？b+树和b-树有什么区别？MySQL聚簇索引和非聚簇索引的区别是什么？他们分别是如何存储的？使用MySQL索引都有哪些原则？MySQL复合索引如何使用？

2、面试官心里分析

这个数据库是30k以内的工程师面试必问的问题，而且如果问数据库，一定是问mysql，N年前可能java工程师出去面试，oracle这块的技能是杀手锏，现在已经没人说，会oracle是加分项了，现在都是熟悉大数据hadoop、hbase等技术是加分项。

3、面试题剖析

**3.1 索引的数据结构**

其实就是让你聊聊mysql的索引底层是什么数据结构实现的，弄不好现场还会让你画一画索引的数据结构，然后会问问你mysql索引的常见使用原则，弄不好还会拿个SQL来问你，就这SQL建个索引一半咋建？

索引是啥？总不能让我来解释了吧，这个问题太基础了，大家都知道，mysql的索引说白了就是用一个数据结构组织某一列的数据，然后如果你要根据那一列的数据查询的时候，就可以不用全表扫描，只要根据那个特定的数据结构去找到那一列的值，然后找到对应的行的物理地址即可。

可以用一颗随意准备的，不知道是什么的树，来演示一把，大家就随便看看，理解个以上。

那么回答面试官的一个问题，mysql的索引是怎么实现的？答案是，不是二叉树，也不是一颗乱七八糟的树，而是一颗b+树。这个很多人都会这么回答，然后面试官一定会追问，那么你能聊聊b+树吗？

所以下面咱们先来聊聊b-树是啥，从数据结构的角度来看，b-树要满足下面的条件：

（1）d为大于1的一个正整数，称为B-Tree的度。

（2）h为一个正整数，称为B-Tree的高度。

（3）每个非叶子节点由n-1个key和n个指针组成，其中d<=n<=2d。

（4）每个叶子节点最少包含一个key和两个指针，最多包含2d-1个key和2d个指针，叶节点的指针均为null 。

（5）所有叶节点具有相同的深度，等于树高h。

（6）key和指针互相间隔，节点两端是指针。

（7）一个节点中的key从左到右非递减排列。

（8）所有节点组成树结构。

（9）每个指针要么为null，要么指向另外一个节点。

（10）如果某个指针在节点node最左边且不为null，则其指向节点的所有key小于v(key1)，其中v(key1)为node的第一个key的值。

（11）如果某个指针在节点node最右边且不为null，则其指向节点的所有key大于v(keym)，其中v(keym)为node的最后一个key的值。

（12）如果某个指针在节点node的左右相邻key分别是keyi和keyi+1且不为null，则其指向节点的所有key小于v(keyi+1)且大于v(keyi)。

上面那段规则，我也是从网上找的，说实话，没几个java程序员能耐心去看明白或者是背下来，大概知道是个树就好了。真是疯了，好多好多，那就拿个网上的图给大家示范一下吧：

比如说我们现在有一张表：

(

id int

name varchar

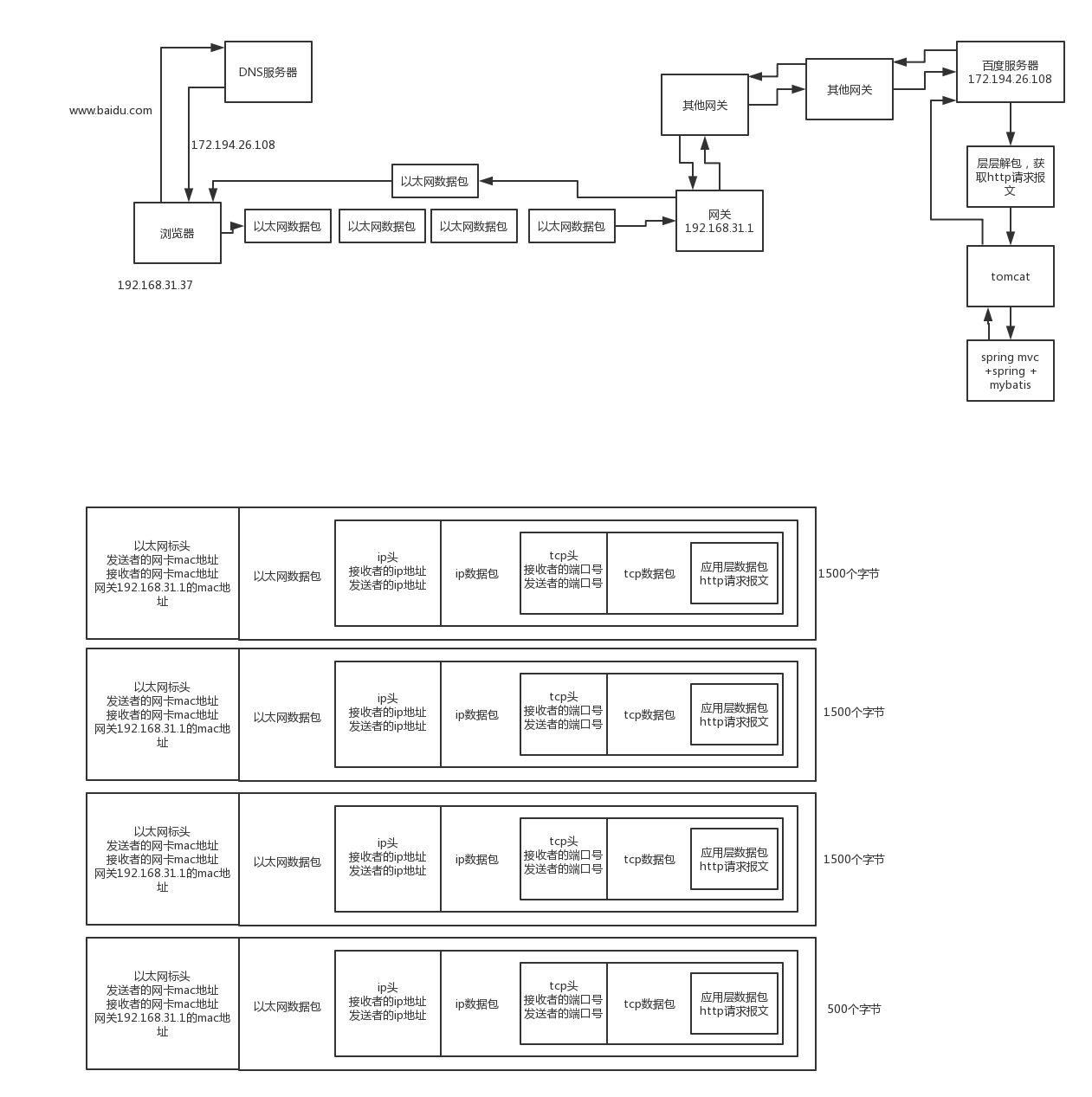
age int

)

我们现在对id建个索引：15、56、77、20、49

select \* from table where id = 49

select \* from table where id = 15



反正大概就长上面那个样子，查找的时候，就是从根节点开始二分查找。大概就知道这个是事儿就好了，深讲里面的数学问题和算法问题，时间根本不够，面试官也没指望你去讲里面的数学和算法问题，因为我估计他自己也不一定能记住。

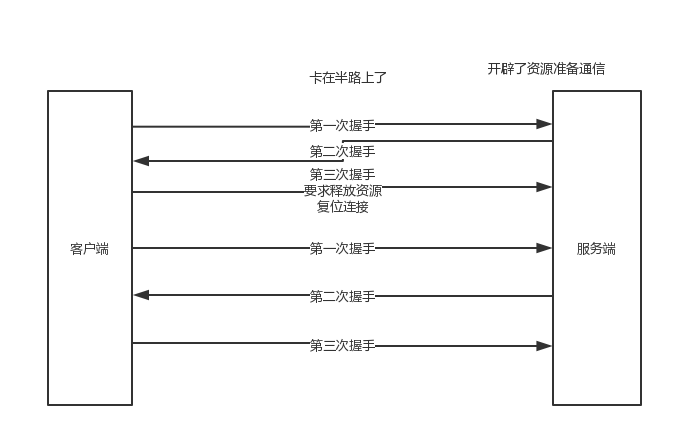
好了，过，直接看下一个，b+树。b+树是b-树的变种，啥叫变种？就是说一些原则上不太一样了，稍微有点变化，同样的一套数据，放b-树和b+树看着排列不太一样的。而mysql里面一般就是b+树来实现索引，所以b+树很重要。

b+树跟b-树不太一样的地方在于：

（1）每个节点的指针上限为2d而不是2d+1。

（2）内节点不存储data，只存储key；叶子节点不存储指针。

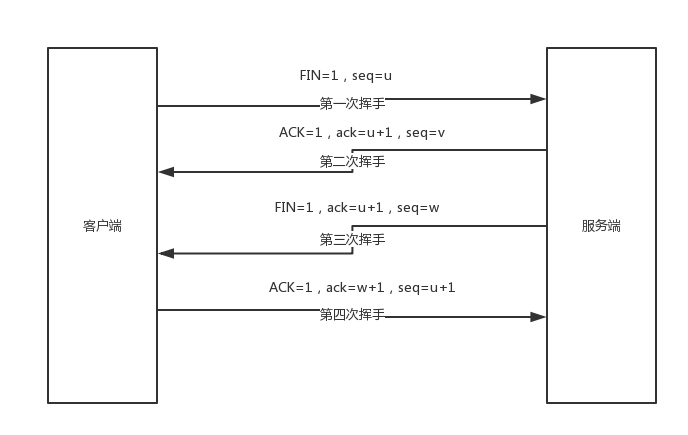
这图我就不自己画了，网上弄个图给大家瞅一眼：



select \* from table where id = 15

select \* from table where id>=18 and id<=49

但是一般数据库的索引都对b+树进行了优化，加了顺序访问的指针，如网上弄的一个图，这样在查找范围的时候，就很方便，比如查找18~49之间的数据：



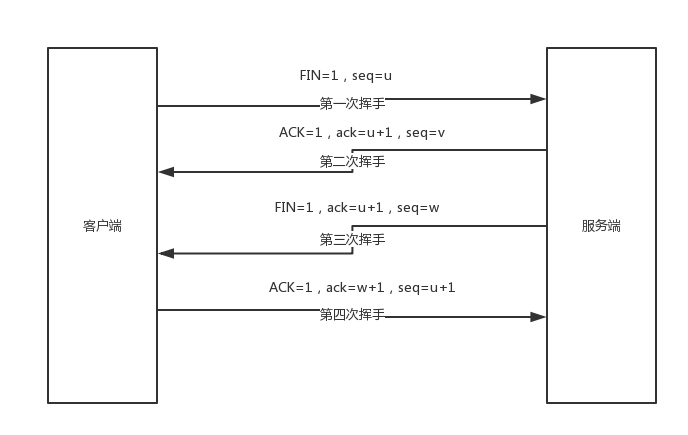
其实到这里，你就差不多了，你自己仔细看看上面两个图，b-树和b+树都现场画一下，然后给说说区别，和通过b+树查找的原理即可。

接着来聊点稍微高级点的，因为上面说的只不过都是最基础和通用的b-树和b+树罢了，但是mysql里不同的存储引擎对索引的实现是不同的。

3.2 myism存储引擎的索引实现

先来看看myisam存储引擎的索引实现。就拿上面那个图，咱们来现场手画一下这个myisam存储的索引实现，在myisam存储引擎的索引中，每个叶子节点的data存放的是数据行的物理地址，比如0x07之类的东西，然后我们可以画一个数据表出来，一行一行的，每行对应一个物理地址。

索引文件



id=15，data：0x07，0a89，数据行的物理地址

数据文件单独放一个文件

物理地址 id name age

0x07 15 张三 22

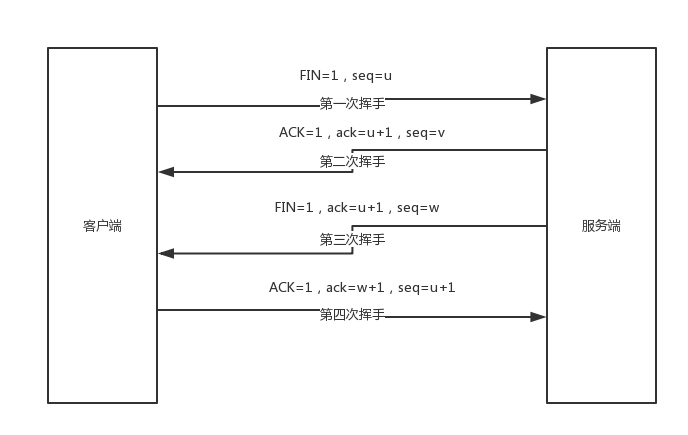
0a89 22 李四 25

select \* from table where id = 15 -> 0x07物理地址 -> 15，张三，22

myisam最大的特点是数据文件和索引文件是分开的，大家看到了么，上面是索引文件里搜索，然后到数据文件里定位一个行的。

**3.3 innodb存储引擎的索引**

再来看看innodb存储引擎的索引实现，跟myisam最大的区别在于说，innodb的数据文件本身就是个索引文件，就是key就是主键，然后叶子节点的data就是那个数据行。我们还是用上面那个索引起来现场手画一下这个索引好了，给大家来感受一下。



innodb存储引擎，要求必须有主键，可以默认内置的就会根据主键建立一个索引，叫做聚簇索引，innodb的数据文件本身同时也是个索引文件，这个索引就是默认根据主键建立的叫做聚簇索引

data：~~0x07~~，完整的一行数据，（15,张三,22）

22，data：完整的一行数据，（22,李四,30）

innodb这种原生的数据文件就是索引文件的组织结构，就叫默认的主键索引为聚簇索引。就是因为这个原因，innodb表是要求必须有主键的，但是myisam表不要求必须有主键。另外一个是，innodb存储引擎下，如果对某个非主键的字段创建个索引，那么最后那个叶子节点的值就是主键的值，因为可以用主键的值到聚簇索引里根据主键值再次查找到数据。

select \* from table where name = ‘张三’

先到name的索引里去找，找到张三对应的叶子节点，叶子节点的data就是那一行的主键，id=15，然后再根据id=15，到数据文件里面的聚簇索引（根据主键组织的索引）根据id=15去定位出来id=15这一行的完整的数据

所以这里就明白了一个道理，为啥innodb下不要用UUID生成的超长字符串作为主键？因为这么玩儿会导致所有的索引的data都是那个主键值，最终导致索引会变得过大，浪费很多磁盘空间。

还有一个道理，一般innodb表里，建议统一用auto\_increment自增值作为主键值，因为这样可以保持聚簇索引直接加记录就可以，如果用那种不是单调递增的主键值，可能会导致b+树分裂后重新组织，会浪费时间。

**3.4 索引的使用规则**

我现在在指导我们架构班的同学在外面跳槽，一般来说，索引这块必问，b+树索引的结构，一般是怎么存放的，出个题，针对这个SQL，索引应该怎么来建立

select \* from table where a=1 and b=2 and c=3，你知道不知道，你要怎么建立索引，才可以确保这个SQL使用索引来查询

好了，各位同学，聊到这里，你应该知道具体的myisam和innodb索引的区别了，同时也知道什么是聚簇索引了，现场手画画，都ok了。然后我们再来说几个最最基本的使用索引的基本规则。

其实最基本的，作为一个java码农，你得知道最左前缀匹配原则，这个东西是跟联合索引（复合索引）相关联的，就是说，你很多时候不是对一个一个的字段分别搞一个一个的索引，而是针对几个索引建立一个联合索引的。

给大家举个例子，你如果要对一个商品表按照店铺、商品、创建时间三个维度来查询，那么就可以创建一个联合索引：shop\_id、product\_id、gmt\_create

一般来说，你有一个表（product）：shop\_id、product\_id、gmt\_create，你的SQL语句要根据这3个字段来查询，所以你一般来说不是就建立3个索引，一般来说会针对平时要查询的几个字段，建立一个联合索引

后面在java系统里写的SQL，都必须符合最左前缀匹配原则，确保你所有的sql都可以使用上这个联合索引，通过索引来查询

create index (shop\_id,product\_id,gmt\_create)

（1）全列匹配

这个就是说，你的一个sql里，正好where条件里就用了这3个字段，那么就一定可以用到这个联合索引的：

select \* from product where shop\_id=1 and product\_id=1 and gmt\_create=’2018-01-01 10:00:00’

（2）最左前缀匹配

这个就是说，如果你的sql里，正好就用到了联合索引最左边的一个或者几个列表，那么也可以用上这个索引，在索引里查找的时候就用最左边的几个列就行了：

select \* from product where shop\_id=1 and product\_id=1，这个是没问题的，可以用上这个索引的

（3）最左前缀匹配了，但是中间某个值没匹配

这个是说，如果你的sql里，就用了联合索引的第一个列和第三个列，那么会按照第一个列值在索引里找，找完以后对结果集扫描一遍根据第三个列来过滤，第三个列是不走索引去搜索的，就是有一个额外的过滤的工作，但是还能用到索引，所以也还好。

select \* from product where shop\_id=1 and gmt\_create=’2018-01-01 10:00:00’

就是先根据shop\_id=1在索引里找，找到比如100行记录，然后对这100行记录再次扫描一遍，过滤出来gmt\_create=’2018-01-01 10:00:00’的行

这个我们在线上系统经常遇到这种情况，就是根据联合索引的前一两个列按索引查，然后后面跟一堆复杂的条件，还有函数啥的，但是只要对索引查找结果过滤就好了，根据线上实践，单表几百万数据量的时候，性能也还不错的，简单SQL也就几ms，复杂SQL也就几百ms。可以接受的。

（4）没有最左前缀匹配

那就不行了，那就在搞笑了，一定不会用索引，所以这个错误千万别犯

select \* from product where product\_id=1，这个肯定不行

（5）前缀匹配

这个就是说，如果你不是等值的，比如=，>=，<=的操作，是like操作，那么必须要是like ‘XX%’这种才可以用上索引，比如说

select \* from product where shop\_id=1 and product\_id=1 and gmt\_create like ‘2018%’

（6）范围列匹配

如果你是范围查询，比如>=，<=，between操作，你只能是符合最左前缀的规则才可以范围，范围之后的列就不用索引了

select \* from product where shop\_id>=1 and product\_id=1

这里就在联合索引中根据shop\_id来查询了

（7）包含函数

如果你对某个列用了函数，比如substring之类的东西，那么那一列不用索引

select \* from product where shop\_id=1 and 函数(product\_id) = 2

上面就根据shop\_id在联合索引中查询

**3.5 索引的缺点以及使用注意**

索引是有缺点的，比如常见的就是会增加磁盘消耗，因为要占用磁盘文件，同时高并发的时候频繁插入和修改索引，会导致性能损耗的。我们给的建议，尽量创建少的索引，比如说一个表一两个索引，两三个索引，十来个，20个索引，高并发场景下还可以。

字段，status，100行，status就2个值，0和1

你觉得你建立索引还有意义吗？几乎跟全表扫描都差不多了

select \* from table where status=1，相当于是把100行里的50行都扫一遍

你有个id字段，每个id都不太一样，建立个索引，这个时候其实用索引效果就很好，你比如为了定位到某个id的行，其实通过索引二分查找，可以大大减少要扫描的数据量，性能是非常好的

在创建索引的时候，要注意一个选择性的问题，select count(discount(col)) / count(\*)，就可以看看选择性，就是这个列的唯一值在总行数的占比，如果过低，就代表这个字段的值其实都差不多，或者很多行的这个值都类似的，那创建索引几乎没什么意义，你搜一个值定位到一大坨行，还得重新扫描。

就是要一个字段的值几乎都不太一样，此时用索引的效果才是最好的

还有一种特殊的索引叫做前缀索引，就是说，某个字段是字符串，很长，如果你要建立索引，最好就对这个字符串的前缀来创建，比如前10个字符这样子，要用前多少位的字符串创建前缀索引，就对不同长度的前缀看看选择性就好了，一般前缀长度越长选择性的值越高。

好了，各位同学，索引这块能聊到这个程度，或者掌握到这个程度，其实普通的互联网系统中，80%的活儿都可以干了，因为在互联网系统中，一般就是尽量降低SQL的复杂度，让SQL非常简单就可以了，然后搭配上非常简单的一个主键索引（聚簇索引）+ 少数几个联合索引，就可以覆盖一个表的所有SQL查询需求了。更加复杂的业务逻辑，让java代码里来实现就ok了。

项目阶段一的代码，你会发现我们互联网公司的系统写代码的风格，SQL达到95%都是单表增删改查，如果你有一些join等逻辑，就放在java代码里来做。SQL越简单，后续迁移分库分表、读写分离的时候，成本越低，几乎都不用怎么改造SQL。

我这里给大家说下，互联网公司而言，用MySQL当最牛的在线即时的存储，存数据，简单的取出来；不要用MySQL来计算，不要写join、子查询、函数放MySQL里来计算，高并发场景下；计算放java内存里，通过写java代码来做；可以合理利用mysql的事务支持

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/15\_你能聊聊CAS一般怎么用以及CAS工作原理是啥？/笔记.docx**

1、面试题

java里玩儿悲观锁和乐观锁一般怎么玩儿？synchronized相当于是悲观锁，CAS相当于是乐观锁。知道CAS是什么吗？CAS是如何实现的？

2、面试官心里分析

这个高级点的面试，肯定会问CAS，还是比较重要的

3、面试题剖析

悲观锁：我现在要操作一个共享数据，我很悲观，我认为我操作的过程中，一定会被人给修改，会导致数据错误；我在操作这个数据之前，先给这个数据加了一把锁，synchronized，在我操作这个数据的期间，就只能是我来操作，其他任何人都操作不了。

乐观锁：我感觉在我操作这个数据的过程中，应该不会被人给修改。我先修改吗，然后修改完之后要设置这个变量的最新的值，此时我会对比一下，当前的这个值，是不是跟我在操作前看到的这个值是一样的，如果是一样的，那么说明可能就没有被人给修改过，如果没有被人修改过，那么我就可以来设置最新的值。

CAS，Compare and Swap，就是比较和交换。java并发包借助CAS实现了乐观锁的思想，synchronized是悲观锁思想。

CAS会操作3个数字，当前内存中的值，旧的预期值，新的修改值，只有当旧的预期值跟内存中的值一样的时候，才会将内存中的值修改为新的修改值。举个例子吧，比如int a = 3，这是内存中的当前值，然后你CAS（3, 5），第一个是旧的预期值，如果3和a是一样的，那么就将a修改为5。

public class Test {

private int i = 0; // 假设i是一个类中的共享变量

public synchronized void incr() {

i++; // 不是线程安全的

}

}

可能2个线程进来，执行incr()方法，期望的是i -> 2，但是其实可能会变成1

public class Test {

private AtomicInteger i = new AtomicInteger(0);

public void incr() {

int next = i.getAndIncrement();

}

}

无论是多少个线程并发执行这段代码，都是ok的，2个线程过来，一定是i -> 2，不可能是1，CAS这个东西，可以保证一个变量的操作都是原子的

CAS里的经典用法就是Atomic系列类，比如说AtomicInteger

还是挺常用的，常见于什么呢？常见于内存计数，比如说你要在内存里维护一个变量，记录每个请求过来的一个次数这样子，你就可以用这个AtomicInteger，可以安全的去给他累加对应的值，原子的

public final int incrementAndGet() {

for (;;) {

int current = get(); // 先拿到i当前的值，0

int next = current + 1; // 对i加1 -> 1

if (compareAndSet(current, next))

// 看一下，i这个变量，当前的值是不是0，如果是0的话，就认为在上面两行代码执行的时候，没有人修改过i这个变量的值，所以就可以将i的值设置为1；但是如果i变量的值已经变成1了，会发现跟自己期望的0，不是一个值，说明上面两行代码执行期间，有人已经修改过了i的值，所以你就不能再次将这个1这个值设置给i了

// 此时compareAndSet()返回一个false，直接进入下一轮循环

// 干了一样的事儿，先获取i当前的值，比如说现在变成了1这个值了

// 再次将i加1 -> 2

// 再次执行compareAndSet(1, 2)，看一下i变量当前的值是不是1，如果是，就证明这个期间没人修改过这个值，就将最新的值2设置给i变量

// 然后compareAndSet()方法就返回true

// return next;，直接跳出一个死循环

// 对于调用者来说，可以拿到本次累加完以后的一个当前值

return next;

}

}

就是实现一个++i的效果，对一个数字累加1后返回，如何实现原子的呢？

就是获取这个数字当前的值，然后加1之后，用compareAndSet(current, next)方法来设置，这个方法会比较内存中的当前值是不是current，如果是才会设置为next也就是加1后的数字，否则的话，就会再次重试，for(;;)就是个无限循环的操作

public final boolean compareAndSet(int expect, int update) {

return unsafe.compareAndSwapInt(this, valueOffset, expect, update);

}

这个compareAndSet()方法，其实是用了cpu的指令，JNI本地方法调用实现的，现代cpu的compareAndSwapInt的指令，就是可以实现原子的操作，他会保证说，当前时刻就一个线程可以针对个数字，比较当前值，然后设置最新值，如果当前值不一致，那么会返回false

其实吧，这里比较关键的一点就是cpu的compareAndSwap操作的原理是啥，以CPU（Intel X86）来举个例子。这块底层指令，会根据当前处理器类型，来决定要不要对一个cmpxchg指令加lock前缀，如果是单处理器，就不要加，因为自动保证顺序；但是如果是多处理器，就加个lock。intel对lock的定义，就是说加了lock之后，就会自动锁掉一块内存区域，然后同一时间只有一个处理器可以读写这块内存区域，其他处理器就不行了。

而且intel对这个lock做了优化，以前都是走一种总线锁，就是一旦锁住之后，只有一个处理器可以操作内存，别的处理器完全不能操作内存，保证说，只有一个cpu可以操作内存，其他cpu都不能操作。相当于是在cpu层面干了类似synchronized这个事儿，这个效率太低了，所以现在现代cpu都不会干这个事儿。整个内存可能包含了很多的数据，就导致同一时间只有一个cpu可以操作内存，这个效率实在是太低了。

但是后来用了缓存锁优化，缓存一致性协议，其实就是处理器对自己内部的缓存锁了，有多个cpu，其中一个cpu就会锁定i变量对应的一行内存，其他cpu就不能对i这个变量进行操作了，同一时间就只能是一个cpu对i变量查看值和设置直；然后就一个内存地址，别的处理器不能操作，但是别的处理器可以处理别的内存地址，其实就是降低了锁的粒度。

CAS其实有3个缺点：

1、ABA问题：如果某个值一开始是A，后来变成了B，然后又变成了A，你本来期望的是值如果是第一个A才会设置新值，结果第二个A一比较也ok，也设置了新值，跟期望是不符合的。所以atomic包里有AtomicStampedReference类，就是会比较两个值的引用是否一致，如果一致，才会设置新值

假设一开始变量i = 1，你先获取这个i的值是1，然后累加了1，变成了2

但是在此期间，别的线程将i -> 1 -> 2 -> 3 -> 1

这个期间，这个值是被人改过的，只不过最后将这个值改成了跟你最早看到的值一样的值

结果你后来去compareAndSet的时候，会发现这个i还是1，就将它设置成了2，就设置成功了

说实话，用AtomicInteger，常见的是计数，所以说一般是不断累加的，所以ABA问题比较少见

2、无限循环问题：大家看源码就知道Atomic类设置值的时候会进入一个无限循环，只要不成功，就不停循环再次尝试，这个在高并发修改一个值的时候其实挺常见的，比如你用AtomicInteger在内存里搞一个原子变量，然后高并发下，多线程频繁修改，其实可能会导致这个compareAndSet()里要循环N次才设置成功，所以还是要考虑到的。

3、多变量原子问题：一般的AtomicInteger，只能保证一个变量的原子性，但是如果多个变量呢？你可以用AtomicReference，这个是封装自定义对象的，多个变量可以放一个自定义对象里，然后他会检查这个对象的引用是不是一个。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/21\_你能聊聊BIO、NIO、AIO分别都是啥？有什么区别？/笔记.docx**

1、面试题

nio、bio、aio都是什么以及有什么区别？说说nio的原理？

2、面试官心里分析

如果聊io这块，我就必问这个问题，因为io的那些过于基础的知识，各种流的使用，是用来考察应届生和培训班刚出来的同学的，正常问一个有经验的开发人员，io这块就是聊聊几种io模式，以及同步、异步、阻塞和非阻塞几种io的概念。

3、面试题剖析

**3.1 BIO**

这个其实就是最传统的网络通信模型，就是BIO，同步阻塞式IO，简单来说大家如果参加过几个月的培训班儿应该都知道这种BIO网络通信方式。就是服务端创建一个ServerSocket，然后客户端用一个Socket去连接那个ServerSocket，然后ServerSocket接收到一个Socket的连接请求就创建一个Socket和一个线程去跟那个Socket进行通信。

然后客户端和服务端的socket，就进行同步阻塞式的通信，客户端socket发送一个请求，服务端socket进行处理后返回响应，响应必须是等处理完后才会返回，在这之前啥事儿也干不了，这可不就是同步么。

这种方式最大的坑在于，每次一个客户端接入，都是要在服务端创建一个线程来服务这个客户端的，这会导致大量的客户端的时候，服务端的线程数量可能达到几千甚至几万，几十万，这会导致服务器端程序的负载过高，最后崩溃死掉。

要么你就是搞一个线程池，固定线程数量来处理请求，但是高并发请求的时候，还是可能会导致各种排队和延时，因为没那么多线程来处理。

**3.2 NIO**

JDK 1.4中引入了NIO，这是一种同步非阻塞的IO，基于Reactor模型。

NIO中有一些概念：

比如Buffer，缓冲区的概念，一般都是将数据写入Buffer中，然后从Buffer中读取数据，有IntBuffer、LongBuffer、CharBuffer等很多种针对基础数据类型的Buffer。

还有Channel，NIO中都是通过Channel来进行数据读写的。

包括Selector，这是多路复用器，selector会不断轮询注册的channel，如果某个channel上发生了读写事件，selector就会将这些channel获取出来，我们通过SelectionKey获取有读写事件的channel，就可以进行IO操作。一个Selector就通过一个线程，就可以轮询成千上万的channel，这就意味着你的服务端可以接入成千上万的客户端。

这块其实相当于就是一个线程处理大量的客户端的请求，通过一个线程轮询大量的channel，每次就获取一批有事件的channel，然后对每个请求启动一个线程处理即可。

这里的核心就是非阻塞，就那个selector一个线程就可以不停轮询channel，所有客户端请求都不会阻塞，直接就会进来，大不了就是等待一下排着队而已。

这里的核心就是因为，一个客户端不是时时刻刻都要发送请求的，没必要死耗着一个线程不放吧，所以NIO的优化思想就是一个请求一个线程。只有某个客户端发送了一个请求的时候，才会启动一个线程来处理。

所以为啥是非阻塞呢？因为无论多少客户端都可以接入服务端，客户端接入并不会耗费一个线程，只会创建一个连接然后注册到selector上去罢了，一个selector线程不断的轮询所有的socket连接，发现有事件了就通知你，然后你就启动一个线程处理一个请求即可，但是这个处理的过程中，你还是要先读取数据，处理，再返回的，这是个同步的过程。

所以NIO是同步非阻塞的。

工作线程，从channel里读数据，是同步的，是工作线程自己去干这个事儿，卡在那儿，专门干读数据的这个活儿，数据没读完，你就卡死在这儿了；然后往channel里写数据，也是你自己去干这个事儿，卡死在这儿了，数据没写完，你就卡在这儿了

**3.3 AIO**

AIO是基于Proactor模型的，就是异步非阻塞模型。

每个连接发送过来的请求，都会绑定一个buffer，然后通知操作系统去异步完成读，此时你的程序是会去干别的事儿的，等操作系统完成数据读取之后，就会回调你的接口，给你操作系统异步读完的数据。

然后你对这个数据处理一下，接着将结果往回写。

写的时候也是给操作系统一个buffer，让操作系统自己获取数据去完成写操作，写完以后再回来通知你。

工作线程，读取数据的时候，是说，你提供给操作系统一个buffer，空的，然后你就可以干别的事儿了，你就把读数据的事儿，交给操作系统去干，操作系统内核，读数据将数据放入buffer中，完事儿了，来回调你的一个接口，告诉你说，ok，buffer交给你了，这个数据我给你读好了

写数据的时候也是一样的的，把放了数据的buffer交给操作系统的内核去处理，你就可以去干别的事儿了，操作系统完成了数据的写之后，级会来回调你，告诉你说，ok，哥儿们，你交给我的数据，我都给你写回到客户端去了

**3.4 同步阻塞、同步非阻塞、异步非阻塞**

但是这里为啥叫BIO是同步阻塞呢？这个其实不是针对网络编程模型来说的，是针对文件IO操作来说的，因为用BIO的流读写文件，是说你发起个IO请求直接hang死，必须等着搞完了这次IO才能返回

BIO的这个同步阻塞，不是完全针对的网络通信模型去说的，针对的是磁盘文件的IO读写，FileInputStream，BIO，卡在那儿，直到你读写完成了才可以

NIO为啥是同步非阻塞？就是说通过NIO的FileChannel发起个文件IO操作，其实发起之后就返回了，你可以干别的事儿，这就是非阻塞，但是接下来你还得不断的去轮询操作系统，看IO操作完事儿了没有。

你呢也可以使用FileChannel这种NIO的模型，去读写磁盘文件，读数据，发起读数据的请求之后，你不是阻塞住的，你可以干别的事儿，但是你在干别的事儿的同时，还得来时不时的自己去轮询操作系统读数据的状态，看看人家读好了没有

AIO为啥是异步非阻塞？就是说通过AIO发起个文件IO操作之后，你立马就返回可以干别的事儿了，接下来你也不用管了，操作系统自己干完了IO之后，告诉你说ok了。同步就是自己还得主动去轮询操作系统，异步就是操作系统反过来通知你。

你也可以基于AIO的文件读写的api去读写磁盘文件，你发起一个文件读写的操作之后，交给操作系统，你就不去管他了，直到操作系统自己完成之后，会来回调你的一个接口，通知你说，ok，这个数据读好了，那个数据写完了

**3.5 BIO、NIO、AIO的demo代码**

上网找

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/34\_目前为止指导学员跳槽就业的一些经验分享/笔记.docx**

一边在我录制这个面试突击课程的时候，录制完了第一季的课程，好几个同学就学习过后，直接出去跳槽找工作了。我一边在录制第二季的课程。

1、我会指导你们来改造自己的简历：简历改完了，看着就是高大上，像个很牛的人，根本不像自己了，将你们的项目、过往履历改造的比较豪华一些

2、针对简历中改造后的内容：进行准备，融入各种技术，亿级流量、es、面试突击第一季、架构班项目阶段一，细细的梳理项目的细节，考虑清楚各种技术如何融入项目的每一个细节之中，针对你写的各种技术要仔细过一遍

3、采用田忌赛马的策略，面试：分为3轮，一轮一轮的投递，每一轮我都会指导大家是投递哪些公司，要注意什么，要收货什么，要积累什么

4、谈offer谈薪资

5、入职新的公司获取一个更好的未来

第1季，仔细过一遍；第2季，我讲解的内容仔细看一下；我没讲的提醒自行复习的内容自己过一遍

再加上我后面严格指导的田忌赛马策略

出去都会很好，找到一些比较不错的互联网公司，薪资一般都能提高40%以上，50%

10k~18k，15k~22k，18k~25k，20k~28k

让你说说项目，接着的会围绕项目里的各种技术来聊，MQ、Dubbo、ES、Redis，会围绕项目业务和细节让你说怎么弄的，小小连环炮，基本上都能被我的面试突击课给cover掉

基础这块，基本上我看了下，问的都是我面试突击第2季提到的东西，要么是我讲的，要么是我提醒自行复习的

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/03\_浏览器请求www.baidu.com的全过程大概是怎么样的？/笔记.docx**

1、面试题

用浏览器请求一个链接的时候，经历了哪些过程（DNS解析过程）？

2、面试官心里分析

这个问题，其实就是跟之前的那个一样，他就是考察考察你的基本功，看你对基本的网络通信知识有没有了解。而且话说回来，考察一个人的基本功，就这个问题应该是相当经典和直接的一个问题，你理解清楚了，那么网络通信这块的一些基本概念，你基本都了解了。否则有的人连个网卡是啥都不知道，mac地址更是懵逼。

3、面试题剖析

现在我们知道网络七层模型大概都是怎么回事了，然后四层模型其实就是会话层、表示层和应用层，合并为了一个应用层，同时没把物理层算在内。就这样。

而且我们也大概知道每一层的协议和作用，网络通信的时候都是怎么回事了，现在我们来看看假设通过浏览器发送一个请求，你访问到那个网站对应的机器，然后人家再给你一个响应的全过程。

现在我们先假设，我们给电脑设置了几个东西：

ip地址：192.168.31.37

子网掩码：255.255.255.0

网关地址：192.168.31.1

DNS地址：8.8.8.8

完了，我们打开一个浏览器，请求www.baidu.com地址，这个时候找DNS服务器，DNS服务器解析域名之后，返回一个ip地址，比如172.194.26.108。接着会判断两个ip地址是不是一个子网的，用子网掩码255.255.255.0，对两个ip地址做与运算，拿到192.168.31.0和172.194.26.0，明显不是一个子网的。

那就得发送一个数据包给网关，其实你就认为是我们的路由器吧，就是192.168.31.1，而且我们是可以拿到网关ip地址的mac地址的，现在我们从应用层出发，通过浏览器访问一个网站，是走应用层的http协议的。

既然要把浏览器发出的请求打包成数据包，要把哪些东西给放到数据包中去呢？应该是把http请求给打包到数据包中去。

http协议分为几个部分：

请求方法 + URL地址 + http版本：比如GET http://172.194.26.108/test HTTP/1.1，类似这种

请求头，类似下面这种：

Host: upload.jiangsu.io

Proxy-Connection: keep-alive

User-Agent: Mozilla/5.0

空行

请求体，比如常见的可以放一个json

这就构成了一个http请求报文

浏览器请求一个地址，先按照应用层的http协议，封装一个应用层数据包，数据包里就放了http请求报文

这个时候会将这个http请求报文打包成一个数据包，仅仅只是数据包的数据部分，此时是数据包是没有头的。上面根据http协议搞一个http请求报文，然后搞一个数据包出来，就是网络模型中到的应用层干的事儿了。

接着就是跑传输层来了，这个层是tcp协议，这个tcp协议会让你设置端口，发送方的端口随机选一个，接收方的端口一般是默认的80端口。这个时候，会把应用层数据包给封装到tcp数据包中去，而且会加一个tcp头，这个tcp数据包是对应一个tcp头的，这个tcp头里就放了端口号信息。

接着跑到网络层来了，走ip协议，这个时候会把tcp头和tcp数据包，放到ip数据包里去，然后再搞一个ip头，ip头里本机和目标机器的ip地址。这里本机ip地址是192.168.31.37，目标机器是172.194.26.108。

应为，通过ip协议，可以判断说，两个ip地址不是在一个子网内的，所以此时只能将数据包先通过以太网协议广播到网关上去，通过网关再给他发送出去

接着是数据链路层，这块走以太网协议，这里是把ip头和ip数据包封到以太网数据包里去，然后再加一个以太网数据包的头，头里放了本机网卡mac地址，和网关的mac地址。但是以太网数据包的限制是1500个字节，但是假设这个时候ip数据包都5000个字节了，那么需要将ip数据包切割一下。

这个时候一个以太网数据包要切割为4个数据包，每个数据包包含了以太网头、ip头和切割后的ip数据包，4个数据包的大小分别是1500，1500,1500，560。ip头里包含了每个数据包的序号。

这4个以太网数据包都会通过交换机发到你的网关上，然后你的路由器是可以联通别的子网的，这个是时候你的路由器就会转发到别的子网的可能也是某个路由器里去，然后以此类推吧，N多个路由器或者你叫网关也行，N多个网关转发之后，就会跑到百度的某台服务器，接收到4个以太网数据包。

百度服务器接收到4个以太网数据包以后，根据ip头的序号，把4个以太网数据包里的ip数据包给拼起来，就还原成一个完整的ip数据包了。接着就从ip数据包里面拿出来tcp数据包，再从tcp数据包里取出来http数据包，读取出来http数据包里的各种协议内容，接着就是做一些处理，然后再把响应结果封装成htp响应报文，封装在http数据包里，再一样的过程，封装tcp数据包，封装ip数据包，封装以太网数据包，接着通过网关给发回去。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/31\_提醒一下自行复习j2ee框架的常见基础面试题/笔记.docx**

（1）spring的ioc和aop的实现机制是什么？了解cglib吗？cglib和jdk动态代理的区别是什么？

（2）spring循环依赖怎么处理？prototype和singleton模式下的处理分别是怎么样的？

（3）spirng mvc的工作原理是什么？

（4）hibernate/mybatis中的一二级缓存的实现原理是什么？

（5）mybatis的工作原理？

（6）你有没有读过哪个开源框架的源码啊？你自己上网随便搜一些spring内部原理的博客，你就看看spring的内部原理，看看spring内部都用了哪些设计模式，就给面试官讲讲这个。spring源码大致看过一些。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/29\_你能说说MySQL数据库锁的实现原理吗？如果死锁了咋办？/笔记.docx**

1、面试题

数据库锁有哪些类型？锁是如何实现的？MySQL行级锁有哪两种？一定会锁定指定的行么？为什么？悲观锁和乐观锁是什么？使用场景是什么？mysql死锁原理以及如何定位和解决？

2、面试官心里分析

说实话，聊mysql的话，我肯定也是循序渐进慢慢问的，先聊下存储引擎，然后问问索引一半怎么用，一半用innodb存储引擎，加上联合索引能玩儿好，明白什么是聚簇索引，再熟悉事务那套东西，包括spring的事务传播之类的，那么数据库常见的开发都没问题了。

接着就是聊聊锁，因为如果对锁没了解的话，线上系统其实进场有时候，在高并发访问下，会出现一些死锁的问题，或者是等待锁时间过长就超时了，偶尔会有这种问题的，所以会问问你锁的问题。

3、面试题剖析

（1）mysql锁

先跟面试官聊下，mysql的锁类型吧，一般其实就是表锁、行锁和页锁。

一般myisam会加表锁，就是myisam引擎下，执行查询的时候，会默认加个表共享锁，也就是表读锁，这个时候别人只能来查，不能写数据的；然后myisam写的时候，也会加个表独占锁，也就是表写锁，别人不能读也不能写。

这个myisam因为很少用了，所以别去管他了，面试的时候来这么一句就ok了。所以话说回来，大家也发现了，myisam其实在实际生产中，我们曾经就是在报表系统里用的是最多的，当年es和kylin没出来的时候，大数据系统计算好的报表数据，都是放mysql的myisam里的，一般就是每天凌晨导入一批数据，那个时候别人不需要查询，没人凌晨来看报表；然后白天也没有写入，就是别人纯查询，建好索引，查询性能还是不错的，单表支撑千万级别数据没问题。

报表系统，有一次，一般来说hadoop计算完大批量的报表数据在凌晨就算完了，没有人看报表的；但是确实有一次是hadoop出了问题，是在上午11点还在计算往表里面大规模大批量的插入数据，当时造成了很严重的锁表，别人查就查不出来，我们的报表系统的用户在查看报表的时候，点504，点504，超时。

超大的case。。。。

这个页级锁，一般几乎很少用，你提一句就ok了，我们不多说了。

其实面试官重点还是跟你聊聊行锁就好了，是innodb引擎一般用行锁，但是也有表锁。

innodb的行锁有共享锁（S）和排他锁（X），两种，其实说白了呢，共享锁就是，多个事务都可以加共享锁读同一行数据，但是别的事务不能写这行数据；排他锁，就是就一个事务可以写这行数据，别的事务只能读，不能写。

innodb的表锁，分成意向共享锁，就是说加共享行锁的时候，必须先加这个共享表锁；还有一个意向排他锁，就是说，给某行加排他锁的时候，必须先给表加排他锁。这个表锁，是innodb引擎自动加的，不用你自己去加。

insert、update、delete，innodb会自动给那一行加行级排他锁

select，innodb啥锁都不加，因为innodb大家记得么，默认实现了可重复读，也就是mvcc机制，所以多个事务随便读一个数据，一般不会有冲突，大家就读自己那个快照就可以了，不涉及到什么锁的问题

但是innodb从来不会自己主动加个共享锁的，除非你用下面的语句自己手动加个锁：

手动加共享锁：select \* from table where id=1 lock in share mode，那你就给那一行加了个共享锁，其他事务就不能来修改这行数据了

手动加排他锁：select \* from table where id=1 for update，那你就给那一行加了个排他锁，意思就是你准备修改，别的事务就别修改了，别的事务的修改会hang住。这个要慎用，一般我们线上系统不用这个，容易搞出问题来。

所以看到这儿，我们琢磨琢磨默认的数据库锁机制，各位同学

对一行数据，如果有人在修改，会加个排他锁，然后你不能修改，你只能等着获取这把锁，但是这个时候你可以随便select，你就是查询你的事务开始之前那行数据的某个版本而已。然后如果你修改某行数据，会同时拿这个表的排他锁，但是呢，如果不同的事务修改不同的行，会拿不同行的行级排他锁，但是大家都会拿一个表的排他锁，ok，实际上innodb的表级排他锁可以随便拿，这个是没冲突的。

所以这个就是mysql innodb存储引擎的默认锁模式，其实还挺不错的。相当于就是一行数据，同一个时刻只能一个人在修改，但是别人修改，你可以随便读，读是读某个版本的，走mvcc机制。大家理解这个就好。

（2）悲观锁和乐观锁是啥？

mysql里的悲观锁是走select \* from table where id=1 for update，就这个，意思是我很悲观，我担心自己拿不到这把锁，我必须先锁死，然后就我一个人可以干这事儿，别人都干不了了，不能加共享锁，也不能加排他锁。

乐观锁，就是说我觉得应该没啥问题，我修改的时候感觉差不多可以获取到锁，不需要提前搞一把锁，我就先查出来某个数据，select id,name,version from table where id=1，接着再执行各种业务逻辑之后再修改，update table set name=’新值’,version=version+1 where id=1 and version=1，就是说每次修改，比较一下这条数据的当前版本号跟我之前查出来的版本号是不是一样的，如果是一样的就修改然后把版本号加1，否则就不会更新任何一行数据，此时就重新查询后再次更新。

一般悲观锁什么时候用呢？比如你查出来了一条数据，要在内存中修改后再更新到数据库中去，但是如果这个过程中数据被别人更新了，你是不能直接干这个操作的，这个时候，你就得走上面那个操作，查询之后就不让别人更新了，你搞完了再说。

但是真有这种场景，推荐你还是用乐观锁把，悲观锁实现简单一点，但是太有风险了，很容易很容易死锁，比如事务A拿了数据1的锁，事务B拿了数据2的锁，然后事务A又要获取数据2的锁就会等待，事务B又要获取数据1的锁，也会等待，此时尴尬了，死锁，卡死，互相等待，永不释放。

所以select ... for update这个语法，轻易不要用，我们几乎线上很少用。

（3）死锁

事务A

select \* from table where id=1 for update

事务B

select \* from table where id=2 for update

事务A

select \* from table where id=2 for update

事务B

select \* from table where id=1 for update

常见的死锁就是类似上面那种，给大家说过了，分别都持有一个锁，结果还去请求别人持有的那把锁，结果就是谁也出不来，死锁了

情况太多，不一一列举了，其实就给大家说下发现死锁的时候怎么排查吧

其实很简单，就是找dba看一下死锁日志，就ok了，然后根据对应的sql，找下对应的代码，具体判断一下为啥死锁了

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/11\_如果线上机器的一个进程用kill命令杀不死该怎么办？/笔记.docx**

1、面试题

线上进程kill不掉怎么办

2、面试官心里分析

但是可能就是想考察一下你有没有处理过类似的问题

3、面试题剖析

我们公司有一套自己研发的发布系统，你每次部署，都是走发布系统，告诉他一个git仓库的地址，那个系统会自动从git仓库拉取代码，基于maven来打包，你还可以指定你要用的profile，maven打包的时候会用对应的profile打对应环境的包，打完jar包之后，就会java -jar之类的来启动。

当时那个发布系统，他自己在每台机器上有一个进程，发布和启动的时候，他启动的那个进程，不是直接java -jar来启动的，发布系统的一个进程搞了一个子进程，子进程是我们的系统进程。

这个其实就是线上可能遇到的一个问题，我们之前确实就是遇到过这个问题，kill一个进程死活杀不死，那个进程成了僵尸进程，就是zombie状态。这是因为这个进程释放了资源，但是没有得到父进程的确认。

ps aux，看看STAT那一栏，如果是Z，那么就是zombie状态的僵尸进程

ps -ef | grep 僵尸进程id，可以找到父进程id

然后先kill掉父进程即可

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/19\_如何解决线上系统JVM的OOM故障和GC性能调优？/笔记.docx**

1、面试题

jvm栈在哪些情况下会溢出？java堆在哪些情况下会溢出？做过哪些jvm优化？用了哪些方法？达到了什么样的效果？jvm问题（内存泄露、线程卡死、jvm崩溃、内存溢出、频繁gc），该如何定位和排查（jmap和jstack）？

2、面试官心里分析

这块其实是jvm面试必问，因为其实基础的jvm知识大家都知道，什么jvm区域划分，什么时候回收了，gc算法了，分代gc，几种垃圾回收器之类的，包括类加载之类的。但是其实在线上，我们经常遇到的jvm问题就两种，一种是oom，就是内存溢出了；另外一种是频繁gc导致系统一直卡顿。

所以如果是我，也肯定会问问你平时对于oom问题，如果发生之后，你怎么排查、定位和解决的；如果出现频繁gc，你怎么排查、定位和解决的，这个是高级工程师很重要的线上问题解决能力。

3、面试题剖析

**3.1 线上oom问题**

常见的问题是突然之间进程就没了，就是线上的java进程跑着跑着就没了。

有个一个命令：dmesg |grep -E ‘kill|oom|out of memory’，可以查看操作系统启动后的系统日志，这里就是查看跟内存溢出相关联的系统日志。如果你确实是oom导致进程被杀死了，那么系统日志里会出现这样的字眼：Out of memory，kill process 13987（java）。。。。。意思就是说，OOM了，然后就杀了你的那个java进程。那么至少用这个命令查看系统日志，就可以确认是OOM问题发生了。

如果一旦oom，就会导致你的jvm死掉，然后你肯定得重启吧。可能你重启之后，先用ps -aux|grep java命令查看一下你的java进程，就可以找到你的java进程的进程id。然后你可以用top命令看一下，top命令显示的结果列表中，会看到%MEM这一列，这里可以看到你的进程可能对内存的使用率特别高。

接着，可以用jstat -gcutil 20886 1000 10命令，就是用jstat工具，对指定java进程（20886就是进程id，通过ps -aux | grep java命令就能找到），按照指定间隔，看一下统计信息，这里会每隔一段时间显示一下，包括新生代的两个Survivor区、Eden区，以及老年代的内存使用率，还有young gc以及full gc的次数。

看到的东西类似下面那样：

S0 S1 E O YGC FGC

26.80 0.00 10.50 89.90 86 954

其实如果大家了解原理，应该知道，一般来说大量的对象涌入内存，结果始终不能回收，会出现的情况就是，快速撑满年轻代，然后young gc几次，根本回收不了什么对象，导致survivor区根本放不下，然后大量对象涌入老年代。老年代很快也满了，然后就频繁full gc，但是也回收不掉。

然后对象持续增加不就oom了，内存放不下了，爆了呗。

所以jstat先看一下基本情况，马上就能看出来，其实就是大量对象没法回收，一直在内存里占据着，然后就差不多内存快爆了。

接着就是用jmap来一把，jmap -histo:live 20886，看看现在现在java进程里的对象分布情况，就是根据每个对象占用内存从大到小来排列的，你可能会看到下面的东西：

1: 485009 489005612 [B

2: 2609794 98034943 [C

。。。。N多行

其实你一下就可以发现是哪些对象占据内存过多了

接着就是要获取一个堆内存快照了，jmap -dump:format=b,file=文件名 [pid]，就可以把指定java进程的堆内存快照搞到一个指定的文件里去，但是jmap -dump:format其实一般会比较慢一些，也可以用gcore工具来导出内存快照。

先在linux上安装gdb，这个自己百度一下就好，很多的

gdb -q --20886　//启动gdb命令

(gdb) generate-core-file //这里调用命令生成gcore的dump文件

(gdb) gcore /tmp/dump.core　//dump出core文件

(gdb) detach //detach是用来断开与jvm的连接的

(gdb) quit //退出

就用上面的命令即可

jmap -dump:format=b,file=heap.hprof /opt/zhss/java/bin /tmp/dump.core

接着用上面这行命令，将dump.core文件转换成hprof文件即可

接着就是可以用MAT工具，或者是Eclipse MAT的内存分析插件，来对hprof文件进行分析，看看到底是哪个王八蛋对象太多了，导致内存溢出了

用MAT分析一下hprof文件，马上就是出来一个图

byte[] 489005612 489005612 489005612

。。。。。下面一大堆

这个其实也是给你显示每个对象占用的内存大小，大概就是这个

其实到此为止就差不多了，我们没时间带着大家来实战，以后架构班都会带着大家深入学习和实战，如果面试，你能把上面的步骤说出来就不错了，很标准的步骤，一些工具的使用都ok了，你就说看到具体哪个对象之后，就去看代码，解决问题就行了。

一般常见的OOM，要么是短时间内涌入大量的对象，导致你的系统根本支持不住，此时你可以考虑优化代码，或者是加机器；要么是长时间来看，你的很多对象不用了但是还被引用，就是内存泄露了，你也是优化代码就好了；这就会导致大量的对象不断进入老年代，然后频繁full gc之后始终没法回收，就撑爆了

要么是加载的类过多，导致class在永久代理保存的过多，始终无法释放，就会撑爆

我这里可以给大家最后提一点，人家肯定会问你有没有处理过线上的问题，你就说有，最简单的，你说有个小伙子用了本地缓存，就放map里，结果没控制map大小，可以无限扩容，最终导致内存爆了，后来解决方案就是用了一个ehcache框架，自动LRU清理掉旧数据，控制内存占用就好了。

另外，务必提到，线上jvm必须配置-XX:+HeapDumpOnOutOfMemoryError，-XX:HeapDumpPath=/path/heap/dump。因为这样就是说OOM的时候自动导出一份内存快照，你就可以分析发生OOM时的内存快照了，到底是哪里出现的问题。

**3.2 系统频繁full gc**

比OOM稍微好点的是频繁full gc，如果OOM就是系统自动就挂了，很惨，你绝对是超级大case，但是频繁full gc会好多，其实就是表现为经常请求系统的时候，很卡，一个请求卡半天没响应，就是会觉得系统性能很差。

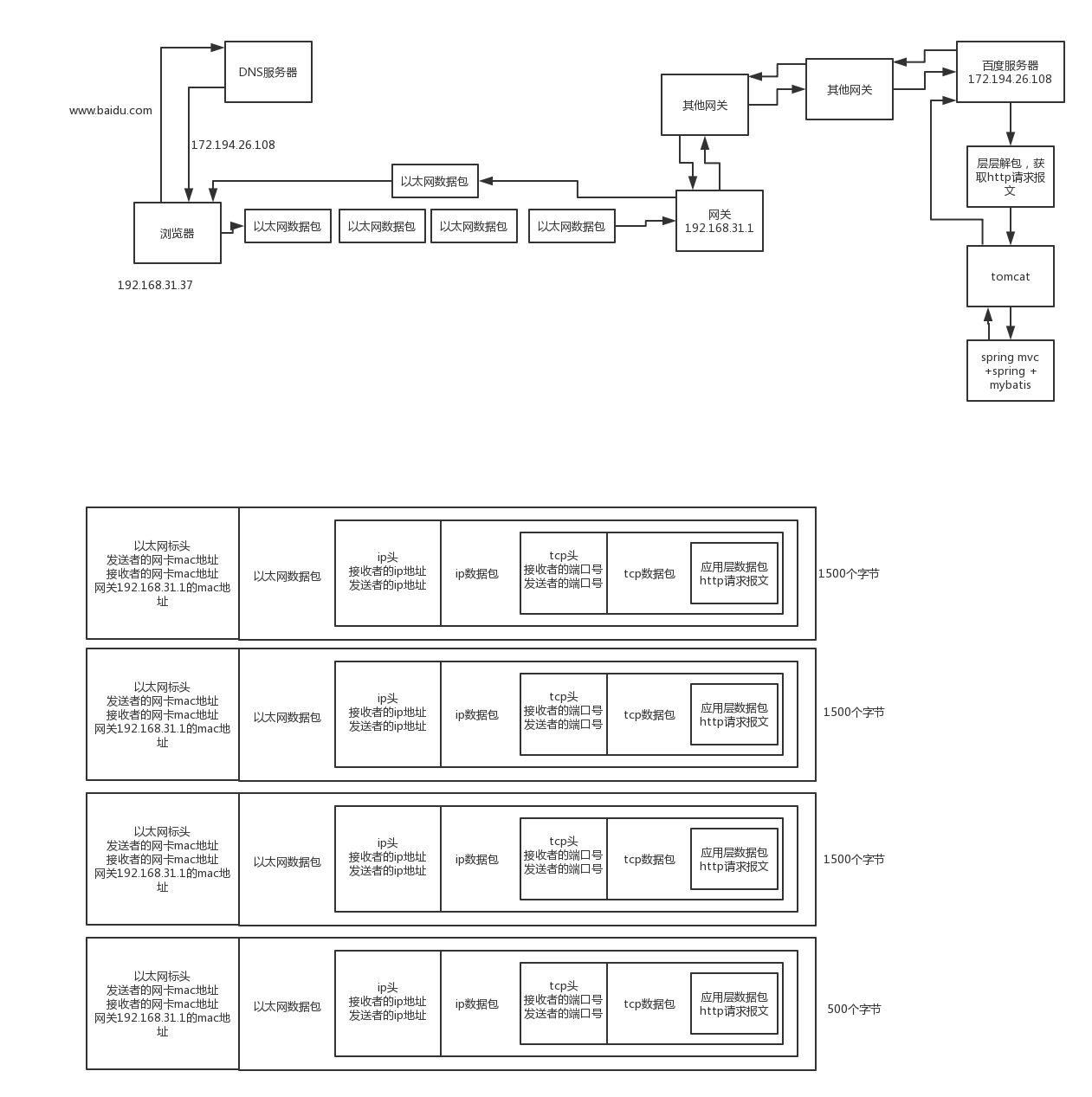
首先，你必须先加上一些jvm的参数，让线上系统定期打出来gc的日志：

-XX:+PrintGCTimeStamps

-XX:+PrintGCDeatils

-Xloggc:<filename>

这样如果发现线上系统经常卡顿，可以立即去查看gc日志，大概长成这样：



如果要是发现每次Full GC过后，ParOldGen就是老年代老是下不去，那就是大量的内存一直占据着老年代，啥事儿不干，回收不掉，所以频繁的full gc，每次full gc肯定会导致一定的stop the world卡顿，这是不可能完全避免的

接着采用跟之前一样的方法，就是dump出来一份内存快照，然后用Eclipse MAT插件分析一下好了，看看哪个对象量太大了

接着其实就是跟具体的业务场景相关了，要看具体是怎么回事，常见的其实要么是内存泄露，要么就是类加载过多导致永久代快满了，此时一般就是针对代码逻辑来优化一下。

给大家还是举个例子吧，我们线上系统的一个真实例子，大家可以用这个例子在面试里来说，比如说当时我们有个系统，在后台运行，每次都会一下子从mysql里加载几十万行数据进来各种处理，类似于定时批量处理，这个时候，如果对几十万数据的处理比较慢，就会导致比如几分钟里面，大量数据囤积在老年代，然后没法回收，就会频繁full gc。

当时我们其实就是根据这个发现了当时两台机器已经不够了，因为我们当时线上用了两台4核8G的虚拟机在跑，明显不够了，就要加机器了，所以增加了机器，每台机器处理更少的数据量，那不就ok了，马上就缓解了频繁full gc的问题了。

面试就先到这里，以后我们架构师课程会走实战派，大量线上系统jvm问题在机器上给大家模拟出来，然后带着大家实战一步一步演练和处理。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/01\_注意！面试中小心别被一些基本功问题给干倒了！/笔记.docx**

很多java工程师，基本功很不扎实，连环炮的一个例子：

1、TCP/IP四层网络模型？OSI七层模型？浏览器请求一个地址，www.baidu.com，从你点击浏览器开始，到最终百度给你一个响应，这个过程中经历了哪些事情，里面的细节都说一下？聊聊https是怎么回事？线上服务器cpu负载100%了，怎么排查和定位？

2、mysql死锁是怎么回事？mysql索引结构是什么样的？linux下的awk、find、grep命令让你来现场写个命令，查找个什么文件的？tomcat的线程模型是什么样的？一般优化和配置哪些参数？

很多java同学，常见于工作3年以内的年轻同学，刚毕业就开始干CRUD的活儿，其实很多基本功可能平时都不怎么接触，可能都快忘了；还有一种是工作经验在5年以上的同学，8年，10年，很多基本功，也是在长期的crud过程中，淡忘的就更加严重了；很多同学平时主要在干传统IT行业，所以说平时对一些底层的东西，接触的确实不怎么多。

很多同学，确实，容易被一些基础的问题给问懵

我主要讲的是面试常问，很多人经常答不出或者答不好的问题，很多特别简单的一些问题，大部分人都会能回答出来的一些问题，spring ioc/aop，我就不说了。

定位，就是围绕着一个核心，面试，面试，面试。大白话 + 现场手工画图，全程对每个问题，我就是讲，我不会落地写代码，部署。

面试突击第1季，大家会发现说都是分布式、高并发、高可用，难一些的问题，你上网查博客，查到的都是一堆没用的信息，大部分同学都不太会，都没怎么做过

面试突击第2季，都是些基础，面试突击，我讲了一些重点（你自己去网上查资料，很难看懂），大部分的基础性的问题（自行上网搜一些博客，快速突击）

4、计算机基础

4.1 网络

（1）说一下TCP/IP的四层网络模型？那么七层网络模型呢？用浏览器请求一个链接的时候，经历了哪些过程（DNS解析过程）？TCP三次握手和四次握手的工作流程是什么？画一下流程图？为什么不是两次握手？

（2）说说socket通信的原理？

（3）说一下http的工作流程？http 1.0、http 1.1、http 2.0具体有哪些区别？http和https的区别是什么？https的原理是什么？

http长连接的原理是什么？

4.2 CPU

（4）线上服务器的cpu使用达到100%了，如何排查、定位和解决该问题？

4.3 进程

（1）跨进程通讯的原理是什么？进程内通讯的原理是什么？线程切换的工作原理是什么？

（2）kill进程杀不掉怎么处理？

4.4 存储

（1）服务器存储空间快满了（95%），还有一个小时存储就满了，在不影响服务正常运行的情况下，该如何解决？

5、Java基础

5.1 并发编程

（1）常用的线程池有哪些？不同线程池的使用场景是什么？说一下线程池配置的几个参数以及线程池启动线程的原理（说说线程池的底层原理）？线程池被关闭的方式有哪几种？线程池队列满的时候怎么处理（FixedThreadPool满了之后会怎么办）？

（2）threadlocal是什么？底层工作原理是什么？

（3）知道CAS是什么吗？CAS是如何实现的？

（4）java的内存模型是什么？能结合内存模型说一下volatile的工作原理吗？

（5）突然断电了，线程池会怎么样？正在处理的阻塞队列的请求怎么办？

5.2 JVM

（1）什么情况下一个对象会被gc掉？为什么要在这个时候让对象被gc？了解哪些jvm垃圾回收器？jvm有哪些gc算法？各种gc算法的优缺点是什么以及适用场景？新生代和老生代的gc回收策略是什么？cms垃圾回收器的原理是什么？何时触发minor gc？何时触发full gc？线上系统频繁发生full gc该如何定位和处理？

（2）jvm栈在哪些情况下会溢出？java堆在哪些情况下会溢出？

（3）做过哪些jvm优化？用了哪些方法？达到了什么样的效果？

（4）jvm问题（内存泄露、线程卡死、jvm崩溃、内存溢出、频繁gc），该如何定位和排查（jmap和jstack）？

（5）说说jvm的类加载机制？都有哪些类加载器以及分别加载哪些文件？手写一个类加载器的demo？类加载器之间的父子关系是什么？什么是双亲委派模型？如何自定义自己的类加载器？自己的类加载器和java自带的类加载器的关系如何处理？以下两种类加载方式有什么区别？class.forName()和classLoader？

（6）了解java字节码吗？都有哪些字节码？java字节码文件的格式是什么？Integer x = 5，int y = 5，比较x == y，都经历了哪些步骤？

（7）JVM的编译优化都有哪些技巧？

（8）指令重排序，内存栅栏，happen-before等概念是指的什么意思？

（9）线上jvm如何配置的？-server -Xms512m -Xmx512m -Xss1024K  
-XX:PermSize=256m -XX:MaxPermSize=512m -XX:MaxTenuringThreshold=20  
XX:CMSInitiatingOccupancyFraction=80 -XX:+UseCMSInitiatingOccupancyOnly

5.3 IO

（1）nio、bio、aio都是什么以及有什么区别？说说nio的原理？

（2）知道netty吗？netty的使用场景都有什么？能说说netty的工作原理？

5.4 集合

（1）hashmap是如何扩容的？每次扩容多少？已有元素如何处理？说一下hashmap底层的数据结构以及工作原理？hashmap如何处理hash碰撞问题以及key重复问题？什么情况下hashmap会内存泄露？hashmap是否是线程安全的？ConcurrentHashMap的实现原理？treemap的实现原理是什么？linkedhashmap的工作原理？hashmap和hashtable底层实现有什么区别？hashtable和concurrenthashtable有什么区别呢？hashmap和treemap有什么区别？

（2）Java中的队列都有哪些？这些队列的区别是什么？ConcurrentLinkedQueue的原理是什么？

（3）Java数组和链表两种结构的操作效率如何？在哪些情况下(从开头开始，从结尾开始，从中间开始)，哪些操作(插入，查找，删除)的效率高？

（4）java集合框架中哪些是线程安全的，哪些是线程不安全的？concurrent包里有哪些类？

（5）AQS的实现原理是什么？哪些场景会用到AQS？

（6）使用无界阻塞队列会出现什么问题？

6、MySQL

（1）MySQ索引的原理和数据结构能介绍一下吗？b+树和b树有什么区别？使用MySQL索引都有哪些原则？MySQL复合索引如何使用？MySQL聚簇索引和非聚簇索引的区别是什么？他们分别是如何存储的？

（2）MySQL有哪些存储引擎啊（myisam和innodb）？都有什么区别？请详细说明一下。innodb存储引擎的文件结构是什么样的？

（3）数据库锁有哪些类型？锁是如何实现的？MySQL行级锁有哪两种？一定会锁定指定的行么？为什么？悲观锁和乐观锁是什么？使用场景是什么？mysql死锁原理以及如何定位和解决？

（4）数据库事务有哪些隔离级别？MySQL的默认隔离级别是？事务的几个特点是什么？项目中如何解决这些问题？

（5）SQL调优的常用手段

7、J2EE

（1）spring循环依赖怎么处理？prototype和singleton模式下的处理分别是怎么样的？懒加载一定能处理好循环依赖的问题么？

（2）spring的事务支持（注解事务、声明事务、编程事务、事务的传播机制）？执行某个操作，前50次成功，第51次失败。a 全部回滚；b 前50次提交，第51次抛异常。ab场景分别如何设置spring事务。

8、Linux

（1）大的log文件中，统计异常出现的次数、排序，或者指定输出多少行多少列的内容。(主要考察awk)

（2）grep命令的使用，文件查找

（3）find命令

（4）poll和epoll的工作原理，有哪些io模型？

9、Tomcat

（1）tomcat的线程模型是什么？tomcat如何进行jvm配置？tomcat的bio在多线程环境下如何处理的？

（2）tomcat架构以及类加载流程？

（3）tomcat如何优化以及有哪些参数？

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/02\_聊聊TCP IP四层网络模型？那OSI七层网络模型呢？/笔记.docx**

1、面试题

说一下TCP/IP的四层网络模型？那么七层网络模型呢？

2、面试官心里分析

结构化面试：不是瞎面，cs基础（computer science），java基础，j2ee，mysql，linux，tomcat，分布式，高并发，高可用，系统设计，数据结构和算法，软素质，工程素质

坦白讲，一些大的公司，计算机基础必面，尤其是针对薪资30k以内的工程师，因为薪资30k以内，你还是要干活儿的吧，还没上升到就设计架构就可以的程度吧，你还没到那个高度吧。所以只要你干活儿，你就不可避免要跟机器、网络、cpu、磁盘、内存，成天打交道。而线上系统，计算基础的一些东西，网络、cpu、磁盘、内存，都是关联很大的，比如说你线上系统会不会因为网络故障导致一些问题？cpu负载达到100%了咋办？磁盘读写很慢快满了咋办？内存使用率过高咋办？

你起码得有一套自己的计算机功底去支撑你玩儿线上系统吧。所以很多人呢，都说计算机基础没啥用，那这个话呢，也对，也不对。对就在于，你如果毕业出来干简单的crud，这些东西你确实不需要；不对就在于，你如果当个高工，带几个小弟干高并发有压力的线上系统，机器负载很高，很容易出问题，结果你连机器都不敢摸，或者也不知道怎么摸，那不是尴尬了么。

所以说，计算机基础，网络、磁盘、cpu、内存，还是得会一点儿基础的

作为一个大公司的面试官，一定会考察你这些东西

3、面试题剖析

（1）简单介绍一下

四层模型和七层模型可以一块儿聊

为啥需要协议？没有协议，各个电脑厂商，比如IBM、苹果啥的，都弄自己的协议，结果就苹果电脑和苹果电脑自己可以通信，和IBM电脑就不可以通信，这不是尴尬么。所以搞一个国际通行的协议，大家都按照这个来，所有电脑都可以通信，不是很好么。

OSI七层模型，就是搞一个标准的网络模型出来，大家都按照这个来走，那么大家都有统一的规范。OSI七层模型，是应用层、表示层、会话层、传输层、网络层、数据链路层、物理层。TCP/IP四层模型，数据链路层、网络层、传输层、应用层。

（2）从底向上的网络分层

1）物理层

物理层，物理层干啥的，就是电脑之间要联网，一般咋弄？类似于说，你有台电脑，现在要联网，咋联？以前N年前，大家记不记得都是在电脑上插根线是吧，然后才能上网，结果现在就是联个wifi就行了，还有中国美国之前联网靠的是海底的光缆。所以物理层就指的这个，就是怎么把各个电脑给联结起来，形成一个网络，这就是物理层的含义，物理层负责传输0和1的电路信号。学过一些计算机的同学，计算机的最最底层，就是0/1，电信号。

2）数据链路层

数据链路层，物理层给各个电脑连接起来了，还传输最底层的0和1电路信号，关键不行啊，你得定义清楚哪些0和1分为一组，这些信号啥意思？这才能进行通信。所以数据链路层就干这事儿，定义一下电路信号咋分组。

00000011（从电脑1出发，要到电脑2去）

00101（从电脑1出发，要到电脑3去）

0101（从电脑2触发，要到电脑4去）

01（从电脑3出发，要到电脑5去）

很多年前，每个公司都定义自己的电路信号分组方式，但是后来出来了以太网协议，以太网。一组电信号是一个数据包，叫一个帧（frame），每个帧分成两个部分，标头（head）和数据（data），标头包含一些说明性的东西，比如说发送者、接收者和数据类型之类的。

每台电脑要往另外一台电脑发送数据，一堆0/1电路信号，封装成数据包，包含头和数据，头里包含了从哪儿来到哪儿去，必须从一台电脑的一个网卡，发送到另外一个电脑的一个网卡，所以以太网发送的数据包必须得指定，目标电脑的网卡的mac地址。

以太网规定了，每个网卡必须得包含一个mac地址，mac地址就是这个网卡的唯一标识，

以太网协议规定了，接入网络里的所有设备，都得有个网卡，以太网协议里的那个数据包，在数据链路层传输的数据包，必须从一个电脑的网卡传输到另外一个电脑的网卡，而这个网卡地址就叫做所谓的mac地址。每块网卡出厂的时候，就有一个唯一的mac地址，48位的二进制，但是一般用12个16进制数字表示，前6个16进制是厂商编号，后6个16进制是网卡流水号。

windows上，ipconfig /all，看看物理地址，就是mac地址，7C-67-A2-20-AB-5C

所以在以太网里传输数据包的时候，必须指定接收者的mac地址才能传输数据。

但是以太网的数据包怎么从一个mac地址发送到另一个mac地址？这个不是精准推送的，以太网里面，如果一个电脑发个数据包出去，会广播给局域网内的所有电脑设备的网卡，然后每台电脑都从数据包里获取接收者的mac地址，跟自己的mac地址对比一下，如果一样，就说明这是发给自己的数据包。

但是上面这种广播的方式，仅仅针对一个子网（局域网）内的电脑，会广播，否则一个电脑不能广播数据包给全世界所有的其他电脑吧，是仅仅广播给一个子网里面的电脑的。

3）网络层

上面说到，子网内的电脑，通过以太网发个数据包，对局域网内的电脑，是广播出去的。那么怎么知道哪些电脑在一个子网内呢？这就得靠网络层了，这里就有一套IP地址，IP地址就可以让我们区分哪些电脑是一个子网的。

网络层里有IP协议，IP协议定义的地址就叫做IP地址。IP地址有IPv4和IPv6两个版本，目前广泛使用的是IPv4，是32个二进制数字组成的，但是一般用4个十进制数字表示，范围从0.0.0.0到255.255.255.255之间。

每台计算机，都会分配一个ip地址，ip地址的前24位（就是前面3个十进制数字），代表了网络，后8位（就是最后1个十进制数字），代表了主机。如果几台电脑是一个子网的，那么前面的3个十进制数字一定是一样的。举个例子，大家平时做实验，玩儿虚拟机吧，自己win上开几个linux虚拟机，你会发现，win上的ip地址可能是192.168.0.103，然后几个虚拟机的ip地址是192.168.0.182，192.168.0.125，192.168.0.106，类似这样的。这个win机器和几个虚拟机，前面3个十进制数字都是192.168.0，就代表大家是一个子网内的，最后那个数字是这个子网的不同主机的编号。

但是实际上上面就是举个例子，其实单单从ip地址是看不出来哪些机器是一个子网的，因为从10进制是判断不出来的。需要通过ip地址的二进制来判断，结合一个概念来判断，叫做子网掩码。比如说ip地址是192.168.56.1，子网掩码是255.255.255.0。知道了子网掩码之后，如果要判断两个ip地址是不是一个子网的，就分别把两个ip地址和自己的子网掩码进行二进制的与运算，与运算之后，比较一下代表网络的那部分。

192.168.56.1和192.168.32.7，判断是不是一个子网的，拿子网掩码255.255.255.0，跟两个ip地址的二进制做与运算

11000000.10101000.00111000.00000001

11111111.11111111.11111111.00000000

子网掩码的二进制是：11111111.11111111.11111111.00000000，然后就跟ip地址的二进制做与好了，通过二进制来比较网络部分的地址是不是一模一样的。

有了网络层的ip地址之后，两台在子网内的电脑终于可以通过广播+mac地址判断来传输数据包进行通信了。

但是如果发现要接受数据包的计算机不在子网内，那么就不能通过广播来发送数据包，需要通过路由来发送数据包。看到路由，就想到了路由器了，对了，路由器大家都熟悉吧，自己平时也会去买对吧，比如小米的路由器啥的，家里上网一般都会弄个路由器对吧，ok。路由器负责将多个子网进行连接，因为比如你在自己家里，其实你就只是你自己的一个子网，你要是访问网站啥的，是跟那个网站机器所在的子网进行通信。

每个电脑都可以搞多个网卡的，不是只有一个网卡，一般笔记本电脑都有以太网网卡和wifi网卡，发送数据包的时候要决定走哪个网卡。路由器，其实就是配置了多个网卡的一个专用设备，可以通过不同的网卡接入不同的网络。

网关其实是就是路由器的一种，运作在网络层，这个概念不多解释了，大家可以就把路由器上的ip地址认为是网关，路由器上每个网卡都有mac地址和对应的ip地址。路由器虽然有mac地址，但是不能通过mac地址寻址的，必须通过ip地址寻址，所以路由器其实是工作在网络层的设备。

网络交换机，也是一种设备，是工作在数据链路层的，路由器是工作在网路层的。网络交换机是通过mac地址来寻址和传输数据包的；但是路由器是通过ip地址寻址和传输数据包的。网络交换机主要用在局域网的通信，一般你架设一个局域网，里面的电脑通信是通过数据链路层发送数据包，通过mac地址来广播的，广播的时候就是通过网络交换机这个设备来把数据广播到局域网内的其他机器上去的；路由器一般用来让你连入英特网。

LAN，就是local area network，就是局域网；WAN，就是wide area network，就是广域网。WLAN是wireless local area network，就是无线局域网，也就是wifi，在局域网内，直接通过wifi无线联网。

家里的路由器是包含了交换机和路由的两个功能的，如果是连接到局域网内的设备就把线插LAN那儿；如果是连接到英特网，就把线插在WAN那儿。

这儿给大家举个例子，就是两个局域网之间，如果要是通过一个路由器进行通信的话，是怎么弄的。

大概过程就是，路由器配置了两块网卡，每个网卡可以连到一个局域网内。

局域网1内的电脑，要发送数据包到局域网2内的电脑，在数据包里写上自己的ip地址和对方的ip地址。但是他们俩不在一个局域网内，于是局域网1内的电脑，先通过交换机将数据包发送给路由器，这个过程需要将路由器的一块网卡的ip地址对应的mac地址写到数据包的头部，然后才能通过交换机广播出去，路由器接收到之后比较自己一块网卡的mac地址，就知道是来找自己的。

接着路由器接收到数据包之后，就会在局域网2内，将目标机器的ip地址对应的mac地址写入头部，接着再次通过交换机发送广播通知，发送给局域网2内的电脑。

一个局域网内的每台机器都有自己的ARP cache，这个ARP就是用来在一个局域网内让各个设备都知道每个设备的ip地址和mac地址的对应关系的，一般就是某个机器发送广播通知自己的ip地址和mac地址的对应关系，然后每个机器给他一个回应。以此类推，大家都互相这样广播一把，ip地址和mac地址的对应关系，大家不就都知道了。

一个ip地址对应着一个mac地址

所以大家在上面可以看到，一个子网内的机器之间通信，就是在数据包里写上对方的mac地址，然后交换机广播出去ok了；但是如果是跨子网的通信，就是写上对方的ip地址，然后先通过mac地址广播到路由器，让路由器再根据另外一个子网的ip地址转换为mac地址，通过另外一个子网的交换机广播过去。就这个意思。

4）传输层

上面我们大概明白了通过网络层的ip地址怎么划分出来一个一个的子网，然后在子网内部怎么通过mac地址广播通信；跨子网的时候，怎么通过ip地址 -> mac地址 -> 交换机 -> 路由器 -> ip地址 -> mac地址 -> 交换机的方式来通过路由器进行通信。

但是这里还有一个问题，就是一台机器上，是很多个程序用一个网卡进行网络通信的，比如说浏览器、QQ、视频直播，这些软件都用了一个网卡往外面发送数据，然后从网卡接收数据，对吧。

所以还需要一个端口号的概念，就是你得发送数据包到某个机器的一个网卡的某个端口上去，然后那个机器上监听那个端口的程序，就可以提取发送到这个端口的数据，知道是自己的数据。端口号是0~65536的范围内，0~1023被系统占用了，别的应用程序就用1024以上的端口就ok了。

电脑1，是在端口48362监听的，通过网卡发送了一条数据 -> 电脑2的ip地址的20386这个端口 -> 电脑2的上面的某个QQ，监听着20386的端口 -> 电脑2的网卡接收到一条数据之后，发现人家找的是20386这个端口，就去找谁哪个哥儿们在监听20386端口，QQ在监听，我就把这个网卡过来的数据，传递给QQ，通过端口知道，哪条数据是给你的

所以其实大家会发现一点，网络层，是基于ip协议，进行主机和主机间的寻址和通信的，然后传输层，其实是建立某个主机的某个端口，到另外一个主机的某个端口的连接和通信的。这个通信，就是通过socket来实现的，通过socket就可以基于tcp/ip协议完成刚才上面说的一系列的比如基于ip地址和mac地址转换和寻址啊，通过路由器通信啊之类的，而且会建立一个端口到另外一个端口的连接。

udp和tcp都是传输层的协议，作用就是在数据包里加入端口号，可以通过端口号进行点对点的通信了。udp协议是不可靠的，发出去人家收到没有就不知道了；tcp协议是可靠的，要求三次握手，而且要求人家接收到数据必须回复你。

传输层的tcp协议，仅仅只是规定了一套基于端口的点对点的通信协议，包括如何建立连接，如何发送和读取消息，但是实际上如果你要基于tcp协议来开发，你一般是用socket，java socket网络编程

5）应用层

通过传输层的tcp协议可以传输数据，但是人家收到数据之后，怎么来解释？比如说收到个邮件你怎么处理？收到个网页你怎么处理？类似这个意思，所以针对各种不同的应用，邮件、网页之类的，都是定义不同的应用层协议的。这个应用层，我们就假设综合了会话层、表示层和应用层了，3层合成1层。

电脑1走tcp协议发送了一段东西过来，发送到电脑2的20386端口

GET http://localhost:8080/ http/1.1

key:valuel

key:value

电脑2走tcp协议读取到了属于自己这个20386端口 的一段数据

GET http://localhost:8080/ http/1.1

key:valuel

key:value

发送了一段响应

200

key;value

key:value

又通过底层的tcp发了出去，电脑1的30987端口，ip

电脑1，网卡，走以太网协议收到一个数据包

200

key;value

key:value

比如最常见的，应用层的协议就是http协议，进行网络通信。

4层：数据链路层（以太网协议），网络层（ip协议），传输层（tcp协议），应用层（http协议）

7层：物理层（网线，海底光缆，传递0/1电路信号），会话层、表示层、应用层 -> 应用层

然后我们看下自己的网络设置，一般包含了ip地址、子网掩码、网关地址、DNS地址。前面3个我们其实都知道啥意思了。ip地址和子网掩码用来划分子网的，判断哪些ip地址在一个子网内。同时你的ip地址和mac地址关联起来的，唯一定位了你的网卡。网关地址，你就认为是路由器上的那个网卡的ip地址吧，路由器的网卡也有mac地址，mac地址对应了一个ip地址。

DNS地址是啥呢？Domain Name System。因为我们一般定位是通过ip地址+mac地址+端口号来定位一个通信目标的，但是如果在浏览器上输入一个www.baidu.com，咋整？这个时候是先把www.baidu.com发给DNS服务器，然后DNS服务器告诉你www.baidu.com对应的ip地址的。

8.8.8.8

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/24\_线上系统使用无界阻塞队列的时候可能会有什么问题？/笔记.docx**

1、面试题

使用无界阻塞队列会出现什么问题？

2、面试官心里分析

其实集合这块，大部分都是些基础的知识，当然可以算是高级的基础知识吧，你短时间内也没法去读JDK集合类的源码，那个以后架构班里可以带着你们来读，但是这里面试，模拟把常见的那些集合问题，尤其是hashmap的，都搞熟就够用了。

然后有可能有的面试官会问你比较刁钻的一个线上的使用问题，就是用无界阻塞队列可能会有什么问题？

3、面试题剖析

常见的无界阻塞队列就是LinkedBlockingQueue，如果是ArrayBlockingQueue，那么就是有界阻塞队列。其实无界阻塞队列最常见的问题，你就是跟线程池结合起来说，就能显得你有点水平，因为你了解线程池的一些实现原理，比如常见的FixedPool就是无界阻塞队列，那么就会出现，如果你的线程全部hang死，或者执行太慢，会导致无界阻塞队列的大小无限扩张，最后耗尽内存，搞死你的系统。

对于这个问题，之前分析过了，你就用有界阻塞队列来替代即可，然后用了有界阻塞队列之后，需要在队列满的时候做任务的离线存储，这块我们的架构班里都有了，就是基于备忘录模式来离线存储，和恢复数据即可。

如果你能把这一块给回答好，那么还是有点水平的。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/22\_提醒一下自行快速了解Netty的基础概念/笔记.docx**

知道netty吗？netty的使用场景都有什么？能说说netty的工作原理？

架构师课程里，在研发底层中间件的时候，才会用到netty，玩儿java web的同学，很少用netty，底层分布式系统中间件的时候，进行分布式系统内部高性能读写的时候，用到netty的特别多。

hadoop、spark很多分布式大数据系统，内部就是有很多都是用netty的

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/13\_提醒一下自行复习并发编程的常见基础面试题/笔记.docx**

提醒一下，我不讲了，实在是太基础了，像下面的一些基础面试题，如果只是聊聊基本的一些原理，我觉得每个人都应该会，但是如果要聊深了，聊到源码级别的，那肯定大家暂时是做不到的，但是以后架构班里面，深入源码层面，我都会带着大家来学习的

1、synchronized和lock的区别、ReentrantLock可重入锁是什么

2、在static方法上加synchronized是锁了什么

3、java中的死锁如何排查和解决：提示一下，打印堆栈来排查

**4、CountDownLatch、CyclicBarrier、Semaphore分别是什么**

5、threadlocal的实现原理是什么：提示一下，其实里面是个map，每个线程对应的副本放map的value里了

6、聊聊你对jdk并发包的理解，concurrent包，lock等等吧，你就聊聊就可以了

我要讲的这几块东西：线程池内部原理、CAS内部原理、volatile内部原理、AQS内部原理，上网去看，搜很多篇博客，都不一定能看懂，而且这几块是面试中高频的常问的高级一点的并发相关的面试题

java并发编程，有很多的知识，但是这次面试突击课，我不会让大家去花很多的时间去系统学习，为了短期面试突击，你自己要复习什么，我给你讲什么，基本上都是针对现在面试的高频问题去讲的

等到后面咱们架构班的课程里，并发编程会放在中间件的研发里面，大量的要用多线程并发编程，那个时候我们再把java并发这块深入的学习，各种技术、各种jdk并发包的源码、锁优化、并发编程模式，结合底层中间件的开发，去做

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/25\_MySQL MyISAM和InnoDB存储引擎的区别是啥？/笔记.docx**

1、面试题

MySQL有哪些存储引擎啊（myisam和innodb）？都有什么区别？请详细说明一下。

2、面试官心里分析

这个其实说实话也是聊MySQL必备的问题，我已经在指导我们架构班的同学在外面跳槽了，根据大家的反馈来看，我觉得确实是，数据库这块还是经常问的，确实可以看到，20多k的职位的话，这块问的不会太深的，就是问问常规的一些问题。

3、面试题剖析

mysql支持的存储引擎有很多种，innodb、myisam、memory，很多，但是我就讲其中两种，因为其实现在，常用的就一种，innodb，myisam以前可能还有一些场景会用，现在用的已经非常少了

（1）myisam

myisam，不支持事务，不支持外键约束，索引文件和数据文件分开，这样在内存里可以缓存更多的索引，对查询的性能会更好，适用于那种少量的插入，大量查询的场景。

比如说最经典的就是报表系统，比如大数据的报表系统，给大家画个图聊聊一半都是怎么玩儿的，常见的就是走hadoop生态来搞，hdfs来存储数据，然后基于hive来进行数仓建模，每次hive跑出来的数据都用sqoop从hive中导出到mysql中去。然后基于mysql的在线查询，就接上j2ee写个简单的web系统，每个报表开发一套代码，写sql查数据，组织数据，按照前端要求的格式返回数据，展现出来一个报表。

这种报表系统，是最适合mysql的myisam存储引擎的，不需要事务，就是一次性批量导入，接下来一天之内就是纯查询了。

这个是比较low的做法，说实在的，现在你要让我说myisam的场景其实不多了，在很多大数据场景里是不适用的，因为真正的大数据系统，很多时候hadoop跑出来的结果还是很大，一天就几千万结果数据，几十亿明细数据，那mysql是抗不住这么大量的数据的。所以现在大数据一般用kylin做离线数据的分析引擎，直接hive数据导入kylin里面去了，或者也可以走elasticsearch。

尝试过做过一个事情，用mysql分库分表来抗，抗不住了，单表一般建议是控制在几万的数据量级，500w以内的数据量，多少表？多少库？多少台数据库服务器？sql多达几百行，各种子查询、join、函数、行转列、列传行，非常不适合用mysql -> 数据量很大 -> sql很复杂 -> 导致mysql数据库服务器cpu负载过高

比较高端一点了，我们会基于自己研发的可配置化BI系统 + kylin + elasticsearch，支持大规模数据的复杂报表的支持，做的非常好，效果远远超出基于mysql的那套方案

后来还有那种实时数据报表，就是storm或者是spark streaming，跑数据出来，来一条算一条，然后结果立马写入mysql中，这个的话，一般就保留当天数据，其实压力不会太大，但是问题在于说，可能写并发会超高，每秒并发轻易就可以几千甚至上万。所以大数据实时报表不会写mysql了，现在一般都是写es。

你可以按照我上面的这套说辞去说说，如果是java方向的同学，就说你们之前配合你们公司的数据团队开发过这种报表系统的j2ee部分，所以当时用myisam比较多，但是后来人家几乎都不用了，借此体现出你是有实际经验的，这回答的档次都不一样了。

（2）innodb

说真的，现在一般用mysql都是innodb，我真很少用其他的存储引擎，而且国内用其他存储引擎的场景和公司也不多，所以用innodb就可以了，而且这个也是mysql 5.5之后的默认存储引擎。

主要特点就是支持事务，走聚簇索引，强制要求有主键，支持外键约束，高并发、大数据量、高可用等相关成熟的数据库架构，分库分表、读写分离、主备切换，全部都可以基于innodb存储引擎来玩儿，如果真聊到这儿，其实大家就可以带一带，说你们用innodb存储引擎怎么玩儿分库分表支撑大数据量、高并发的，怎么用读写分离支撑高可用和高并发读的，用上第1季的内容就可以了。

说实话，关于存储引擎，现在因为其实真的主要就是innodb，聊到这儿就可以了，反而被问到这问题，多拓展根据你的经验来回答。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/14\_如何抗住面试官的线程池N个连环炮的发问/笔记.docx**

1、面试题

常用的线程池有哪些？不同线程池的使用场景是什么？说一下线程池配置的几个参数？线程池队列满的时候怎么处理（FixedThreadPool满了之后会怎么办）？线程池启动线程的原理（说说线程池的底层原理）？线程池被关闭的方式有哪几种？

2、面试官心里分析

如果是对一个20多k的工程师，一般并发这块第一个问题就是聊一下线程池，因为非常常用，上一讲的简单问题基本都是默认你会的，但是如果是20k以内的同学，上一讲的问题应该也会抽几个问问。

而且线程池这块，就是会跟上面那样来几个连环炮问问的，从一般怎么用，到参数设置，再到工作原理，实际上这块很多人都是模模糊糊知道一点儿，但是能说清楚的人很少

3、面试题剖析

**3.1 线程池的最最基本的原理**

线程池，说白了就是为了避免频繁创建和销毁线程带来的巨大开销，就维护一定数量的线程池，创建以后别销毁，用完了扔回去再个下一个任务来使用，这样就可以提高线程的使用效率。一般常见于后台处理类的系统，或者是复杂接口中的多数据并发获取。

池子，里面比如说固定就是100个线程，后面的人过来了就是从线程池里获取线程来工作，线程干完活儿了，别销毁，直接还到线程池里去，避免频繁的创建和销毁线程，会导致系统的运行效率很低

Java线程池包含4个部分

（1）线程池管理器（ThreadPool）：就是负责创建和销毁线程池的

（2）工作线程（PoolWorker）：就是线程池中的一个线程

（3）工作任务（Task）：这个就是线程池里的某个线程需要执行的业务代码，这个是你自己编写的业务逻辑

（4）任务队列（TaskQueue）：这个是扔到线程池里的任务需要进行排队，要进任务队列

**3.2 常用的几种线程池和API**

常用的线程池有：

（1）SingleThreadExecutor（单线程线程池，很少用 -> 自己做一个内存队列 -> 启动后台线程去消费）

（2）FixedThreadExecutor（固定数量线程池）：比如说，线程池里面固定就100个线程，超过这个线程数就到队列里面去排队等待

（3）CachedThreadExecutor（自动回收空闲线程，根据需要自动新增线程，传说中的无界线程池）：无论有多少任务，根据你的需要，无限制的创建任意多的线程，在最短的时间内来满足你，但是高峰过去之后，如果有大量的线程处于空闲状态，没有活儿可以干，等待60s之后空闲的线程就被销毁了

（4）ScheduledThreadExecutor（线程数量无限制，支持定时调度执行某个线程）：提交一个任务，对于这个任务不是立马执行的，是可以设定一个定时调度的逻辑，比如说每隔60s执行一次，这个一般不用，一般来说就用spring schedule的支持

一般其实最常用的是FixedThreadExecutor和CachedThreadExecutor

ScheduleThreadExecutor，也可能会使用，但是就是除非你的那个线程任务要定时调度，才会用这个线程池，不过说实话，简单的定时调度一般就是走spring的schedule支持就行了，当然如果你要用这个也行

Java的线程池比较重要的几个API

（1）Executor：代表线程池的接口，有个execute()方法，扔进去一个Runnable类型对象，就可以分配一个线程给你执行

（2）ExecutorService：这是Executor的子接口，相当于是一个线程池的接口，有销毁线程池等方法 -> ExecutorService就代表了一个线程池管理器，会负责管理线程池 -> 线程的创建和销毁 -> 队列排队

（3）Executors：线程池的辅助工具类，辅助入口类，可以通过Executors来快捷的创建你需要的线程池。创建线程池的入口类，包含newSingleThreadExecutor()、newCachedThreadPool()、newScheduleThreadPool()、newFixedThreadPool()，这些方法，就是可以让你创建不同的线程池出来

（4）ThreadPoolExecutor：这是ExecutorService的实现类，这才是正儿八经代表一个线程池的类，一般在Executors里创建线程池的时候，内部都是直接创建一个ThreadPoolExecutor的实例对象返回的，然后同时给设置了各种默认参数。

如果我们要创建一个线程池，两种方式，要么就是Executors.newXX()方法，快捷的创建一个线程池出来，线程池的所有参数设置都采取默认的方式；要么是自己手动构建一个THreadPoolExecutor的一个对象，所有的线程池的参数，都可以自己手动来调整和设置

public class Executors {

public static ExecutorService newFixedThreadPool(int nThreads) {

return new ThreadPoolExecutor(nThreads, nThreads,

0L, TimeUnit.MILLISECONDS,

new LinkedBlockingQueue<Runnable>());

}

public static ExecutorService newCachedThreadPool() {

return new ThreadPoolExecutor(0, Integer.MAX\_VALUE,

60L, TimeUnit.SECONDS,

new SynchronousQueue<Runnable>());

}

}

如果我们要用线程池，大部分同学，我估计如果之前使用过线程池的话，一遍都是怎么用的呢？

public class MyTask implements Runnable {

public void run() {

// 实现你的业务逻辑

// 这段业务逻辑，就会交给线程池里的某个线程去执行

}

}

// 创建一个固定线程数量为100的FixedThreadPool线程池

ExecutorService myThreadPool = Executors.newFixedThreadPool(100);

// 要往线程池里提交一个任务

Runnable myTask = new MyTask();

myThreadPool.execute(myTask);

// 上面的代码就是平时大家最最常用的线程池的使用代码

// 执行了线程池的execute()方法，就相当等于是给这个线程提交了一个任务

// 线程池会优先用有已有的线程来处理这个任务

// 但是如果所有的线程池里的线程都处于一个繁忙的状态，此时就会将这个任务扔到队列里去排队，等待某个线程空闲之后来处理这个任务

到此为止，都是讲的很easy的，但是你会发现只是知道这么点是完全不够用的

**3.3 线程池的构造参数和真正的工作原理**

ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, BlockingQueue<Runnable> workQueue, ThreadFactory threadFactory, RejectedExecutionHandler handler)

ThreadPoolExecutor就是线程池，那么这个类的构造函数的所有入参，就是你可以设置的参数，我们来解释一下这些参数吧

corePoolSize：线程池里的核心线程数量

maximumPoolSize：线程池里允许有的最大线程数量

keepAliveTime：如果线程数量大于corePoolSize的时候，多出来的线程会等待指定的时间之后就被释放掉，这个就是用来设置空闲线程等待时间的

unit：这个是上面那个keepAliveTime的单位

workQueue：这个是说，通过ThreadPoolExecutor.execute()方法扔进来的Runnable工作任务，会进入一个队列里面去排队，这就是那个队列

threadFactory：如果需要创建新的线程放入线程池的时候，就是通过这个线程工厂来创建的

handler：假如说上面那个workQueue是有固定大小的，如果往队列里扔的任务数量超过了队列大小，咋办？就用这个handler来处理，AbortPolicy、DiscardPolicy、DiscardOldestPolicy，如果说线程都繁忙，队列还满了，此时就会报错，RejectException

这些参数的含义先解释一下：

假设我们自己手动创建一个ThreadPoolExecutor线程池，设置了以下的一些参数

corePoolSize：2个

mamximumPoolSize：4个

keepAliveTime：60s

workQueue：ArrayBlockingQueue，有界阻塞队列，队列大小是4

handler：默认的策略，抛出来一个ThreadPoolRejectException

（1）一开始线程池里的线程是空的，一个都没有。有一个变量维护的是当前线程数量，这个变量是poolSize，poolSize = 0，如果当前线程的数量小于corePoolSize（2），poolSize < corePoolSize，那么来了一个任务优先创建线程，直到线程池里的线程数量跟corePoolSize一样；poolSize = 1，poolSize < corePoolSize（2），又创建一个线程来处理这个任务；poolSize = 2

（2）如果当前线程池的线程数量（poolSize = 2）大于等于corePoolSize（2）的时候，而且任务队列没满（最大大小是4，但是当前元素数量是0），那么就扔到任务队列里去

（3）如果当前线程池的线程数量大于等于corePoolSize的时候，而且任务队列满了（最大大小是4，当前已经放了4个元素了，已经满了），那么如果当前线程数量小于最大线程数（poolSize = 2，maimumPoolSize = 4，poolSize < maximumPoolSize），就继续创建线程；poolSize = 3，提交了一个任务，poolSize >= corePoolSize，任务队列满，poolSize < maximumPoolSize，再次创建一个任务

（4）如果此时poolSize >= corePoolSize，任务队列满，poolSize == maximumPoolSize，此时再次提交一个任务，当前线程数已经达到了最大线程数了，那么就使用handler来处理，默认是抛出异常，ThreadPoolRejectExeception

（5）此时线程池里有4个线程，都处于空闲状态，corePoolSize指定的是就2个线程就可以了，但是此时超过了corePoolSize 2个线程，所以如果那超出的2个线程空闲时间超过了60s，然后线程池就会将超出的2个线程给回收掉

如何设置池的这些参数？先来看看创建线程池的默认代码

其实上面都说过了，啥时候会创建新线程？其实就是线程数没到corePoolSize的时候，会创建线程；接着就是任务队列满了，但是线程数小于maximumPoolSize的时候，也会创建线程；创建的时候通过threadFactory来创建即可

**3.4 常用线程池的工作原理**

FixedThreadPool

public static ExecutorService newFixedThreadPool(int nThreads) {

return new ThreadPoolExecutor(nThreads, nThreads,

0L, TimeUnit.MILLISECONDS,

new LinkedBlockingQueue<Runnable>());

}

（1）corePoolSize = 100，maximumPoolSize = 100，keepAliveTime = 0，workQueue = 无界队列

（2）刚开始的时候，比如说假设一开始线程池里没有线程，你就不断的提交任务，瞬间提交了100个任务，一下子创建100个线程出来，其实poolSize == corePoolSize，再提交任务，直接就会发现LinkedBlockQueue根本就没有大小的限制，所以说根本就不会满，所以此时后续的所有任务直接扔到LinkedBlockingQueue里面去排队

（3）100个线程，只要出现了空闲，就会从队列里面去获取任务来处理，以此类推，就100个线程，不停的处理任务

LinkedBlockingQueue根本就不会满，直到扔任务扔的内存溢出，扔了几百万个任务，几千万个任务，队列实在是太大太大了，导致内存溢出，满了，就死了

maximumPoolSize根本就没用，而且其实也是直接跟corePoolSize设置成一样

keepAliveTime = 0，也就是，创建出来的线程，根本就不会去判断是否超过了指定的空闲时间，不会去回收空闲线程数量，后面就维护这么固定数量的一个线程，有任务就往队列里面怼，固定数量的比如100个线程不停的处理任务，如果100个线程都处理不过来，那么就无限制的往LinkedBlockingQueue里面去排队，直到内存溢出

public static ExecutorService newCachedThreadPool() {

return new ThreadPoolExecutor(0, Integer.MAX\_VALUE,

60L, TimeUnit.SECONDS,

new SynchronousQueue<Runnable>());

}

corePoolSize = 0，maximumPoolSize = 最大值（无限大），keepAliveTime = 60s，workQueue=SynchronousQueue

SynchronousQueue（实际上没有存储数据的空闲，是用来做多线程通信之间的协调作用的），一开始提交一个任务过来，要求线程池里必须有一个线程对应可以处理这个任务，但是此时一个线程都没有，poolSize >= corePoolSize , workQueue已经满了，poolSize < maximumPoolSize（最大值），直接就会创建一个新的线程来处理这个任务

如果短期内有大量的任务都涌进来，实际上是走一个直接提交的思路，对每个任务，如果没法找到一个空闲的线程来处理它，那么就会立即创建一个新的线程出来，来处理这个新提交的任务

短时间内，如果大量的任务涌入，可能会导致瞬间创建出来几百个线程，几千个线程，是不固定的

但是当这些线程工作完一段时间之后，就会处于空闲状态，就会看超过60s的空闲，就会直接将空闲的线程给释放掉

总结一下：

（1）FixedThreadPool：线程数量是固定的，如果所有线程都繁忙，后续任务全部在一个无界队列里面排队，无限任务来排队，直到内存溢出

（2）CachcedThreadPool：线程数量是不固定的，如果一下子涌入大量任务，没有空闲线程，那么就创建新的线程来处理；如果有空闲线程，就是一个任务配对一个空闲线程来处理；如果线程空闲时间超过60s，就给回收掉空闲线程

**3.5 各种线程池在什么样的场景下使用**

FixedThreadPool：比较适用于什么场景呢？负载比较重，而且负载比较稳定的这么一个场景，我给大家来举个例子，我们之前线上有一套系统，负载比较重，后台系统，每分钟要执行几百个复杂的大SQL，就是用FixedThreadPool是最合适的。

因为负载稳定，所以一般来说，不会出现说突然瞬间涌入大量的请求，100个线程处理不过来，然后就直接无限制的排队，然后oom内存溢出，死了

CachedThreadPool：负载很稳定的场景，用CachedThreadPool就浪费了；每天大部分时候可能就是负载很低的，CachedThreadPool，用少量的线程就可以满足低负载，不会给系统引入太大的压力；但是每天如果有少量的高峰期，比如说中午或者是晚上，高峰期可能需要一下子搞几百个线程出来，那么CachedThreadPool就可以满足这个场景；高峰期应付过去之后，线程如果处于空闲状态超过60s，自动回收空闲线程，避免给系统带来过大的负载

**3.6 如何设置线程池的参数**

这是Executors自己创建的，其实无非就是实例化一个ThreadPoolExecutor对象，你要是不想用Executors创建，就自己构造一个ThreadPoolExecutor对象也行，那么参数你就自己设置不就得了

通常来说，建议大家就用Executors提供的默认的线程池就可以了，我觉得还是挺合适的，因为他们的默认的参数设置可以满足大部分的场景了，但是如果你在学习了这一课之后，确实需要自己去动手定制一下线程池的策略，那么就自己动手构建ThreadPoolExecutor实例就可以了，所有的参数自己设置

先来看看默认的参数设置，其实固定大小的线程池，说白了，就是corePoolSize和maximumPooSize是一样的，那么只要达到了你传入的那个线程数量，而且任务队列满了，就报错；cached线程池，说白了，就是可以无限的创建线程，因为maximumPooSize是无限大的，但是超过60秒空闲就给你回收

如果你要自己创建个线程池，一般自己设置参数好了：

corePoolSize：这个其实就是说你算一下每个任务要耗费多少时间，比如一个任务大概100ms，那么每个线程每秒可以处理10个任务，然后你算算你每秒总共要处理多少个任务啊，比如说200个任务，那么你就需要20个线程，每个线程每秒处理10个任务，不就可以处理200个任务。但是一般都会多设置一些，比如你可以设置个30个线程。

坦白来讲，如果你面试被问到这个问题，体现你水平的地方，其实在于不同场景的理解：

你希望用类似于FixedThreadPool的这个线程池，corePoolSize和maximumPoolSize按照上面说的策略设置成一样的就可以了

如果用的是FixedPool的话，一般在于workQueue和handler的理解，因为你看下默认的实现，其实线程数量达到corePoolSize的时候，就会放入workQueue排队，但是默认使用的是无界队列，LinkedBlockingQueue，所以会无限制往里面排队，然后就是你corePooSize指定数量的线程不断的处理，队列里的任务可能会无限制的增加

这个其实就是适合处理那种长期不断有大量任务进来，长期负载都很重，所以你不能用CachedPool，否则长期让机器运行大量线程，可能导致机器死掉，cpu耗尽。所以你就只能控制线程的数量，用有限的线程不断的处理源源不断进入的任务，有时高峰时任务较多，就做一下排队即可。

所以FixedPool的参数里，对于workQueue，你要考虑一点，默认的是无界队列，可能会有问题，就是要是无限排队，别把机器给搞死了，那么这个时候你可以换成ArrayBlockingQueue，就是有界队列，自己设置一个最大长度，一旦超出了最大长度，就通过handler去处理，你可以自己对handler接口实现自己的逻辑，我给你举个例子，此时你可以把数据放到比如数据库里去，做离线存储或者是什么的

需要去实现CachedThreadPool的这么一个策略

corePoolSize可以设置为0，但是maximumPoolSize考虑不用设置成无限大，有一个风险，假设突然进来的流量高峰，导致你的线程池一下子出来了几万个线程，瞬间会打满cpu负载，直接机器会死掉

maximumPoolSize可以设置成一个，你的机器cpu能负载的最大的线程数，一个经验值，4核8G的虚拟机，你线程池启动的线程数量达到100个就差不多了，如果同时有100个线程，而且做很频繁的操作，cpu可能就快到70%，80%，90%

corePoolSize = 0，maximumPoolSize = 150 -> handler报错 -> 实现一个handler，将多余的线程给离线存储起来，后续高峰过了，再重新扫描出来重新提交

你看下CachedPool，他那里用的是SynchronousQueue，这个queue的意思是如果要插入一个任务，必须有一个任务已经被消费掉了，所以很可能出现说，线程数量达到corePoolSize之后，大量的任务进来，此时SynchronousQueue里的任务如果还没被拿走，那么就会认为队列满了，此时就会创建新的线程，但是maximumPoolSize是无限大的，所以会无限制的创建新的线程。但是如果后续有线程空闲了，那么就会被回收掉。

所以如果你用CachedPool，相当于是在高峰期，无限制的创建线程来拼命耗尽你的机器资源来处理并发涌入的大量的任务

所以CachedPool，可以用在那种瞬时并发任务较高，但是每个任务耗时都较短的场景，就是短时间内突然来个小高峰，那么就快速启动大量线程来处理，但是每个线程处理都很快，而且高峰很快就过去了

**3.7 线程池关闭原理**

shutdown()：调用之后不允许提交新的任务了，所有调用之前提交的任务都会执行，等所有任务执行完了，才会真正关闭线程池，这就是优雅的关闭方式

shutdownNow()：返回还没执行的task列表，然后不让等待的task执行，尝试停止正在执行的task，非优雅关闭，强制关闭

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/20\_说说你们的线上系统的JVM都配置了哪些参数？/笔记.docx**

1、面试题

线上jvm如何配置的？

-server -Xms512m -Xmx512m -Xss1024K  
-XX:PermSize=256m -XX:MaxPermSize=512m -XX:MaxTenuringThreshold=20  
XX:CMSInitiatingOccupancyFraction=80 -XX:+UseCMSInitiatingOccupancyOnly

2、面试官心里分析

这块说实话，其实就是一样的，看看你到底玩儿没玩儿线上系统的jvm配置，如果你是培训班儿刚出来的，绝对会被这些线上问题的解决处理的提问给难住，然后一看就是没实际玩儿过，那你就糟糕了。或者如果你一直就是负责开发，没弄过线上系统jvm配置，那也肯定是不行的。

3、面试题剖析

上一节课已经看了吧，各位同学，最起码的就是OOM自动导出内存快照，还有打印gc日志，都是必须加的，因为这是线上系统jvm常见的问题。然后我们来解释别的常用参数。

-server -Xms4g -Xmx4g -Xss256k  
-XX:PermSize=512m -XX:MaxPermSize=512m -XX:MaxTenuringThreshold=20  
-XX:CMSInitiatingOccupancyFraction=80 -XX:+UseCMSInitiatingOccupancyOnly -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/path/heap/dump -XX:+PrintGCTimeStamps -XX:+PrintGCDeatils -Xloggc:<filename>

-server：必须得加，因为你是服务端程序，用-server启动较慢，但是启动后性能会更好

-Xms4g：堆内存的初始大小是4g，一般我们线上常用的普通应用系统是4核8G的虚拟机，足够了，所以一般堆内存给4g，稍微留点富裕，毕竟操作系统自己也要用内存的

-Xmx4g：堆内存的最大大小是4g，一般就是跟初始大小是一样的，一般不建议设置比-Xms4g大，因为导致运行时动态增加堆内存，会有问题

-Xss256k：栈内存大小，这个一般设置成256k就差不多够了，毕竟主要内存都是放堆里的，栈里就是一些变量什么的

-XX:PermSize=512m，这是永久代大小，这是放加载的类之类的东西的，一般设置512m是足够了，太小了，有的时候在就是那种动态生成字节码的场景下，可能会有问题

-XX:MaxPermSize=512m，这就是永久代的最大大小，一般就跟上面那个一样就行了

~~-XX:MaxTenuringThreshold=20：这个就是说新生代里多少次没回收掉就进入老年代~~

~~-XX:CMSInitiatingOccupancyFraction=80，设置老年代占用多大比例后触发cms垃圾回收~~

~~-XX:+UseCMSInitiatingOccupancyOnly，这个跟上面那个参数配合起来使用，就是说仅仅使用上面指定的那个比例，否则不指定这个参数，jvm第一次使用上面那个比例后，后续会自动调整那个比例~~

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/17\_你能聊下AQS的实现原理么？这到底是什么东西？/笔记.docx**

1、面试题

AQS的实现原理是什么？哪些场景会用到AQS？

2、面试官心里分析

这个AQS说实在的，是比较高级的并发这块的面试题了，但是确实有些公司会问，所以我这边还是给大家来讲一下

3、面试题剖析

AQS，AbstractQueuedSynchronizer，抽象的队列式同步器，AQS其实主要定义了一套对共享资源访问的同步框架，比如ReentrantLock、Semaphore、CountDownLatch都是依赖这个AQS的。AQS是java用来构建锁和其他同步组件的一个基础框架。

内部有一个FIFO队列，如果某个线程获取同步状态失败了，那么AQS就会把这个线程构造成一个node，塞入FIFO队列的尾部，然后如果同步状态释放的时候，就会从FIFO队列的头部唤醒一个node。其实说白了，就是让线程获取一个共享资源的时候，自动进行排队。

这就是所谓java中的最核心的同步这块的底层逻辑。

AQS里是有一个volatile修饰的state的，就代表了某个共享资源的同步状态，或者更加方便的理解，可以认为是一把锁，比如多个线程访问某段被锁的代码，会去尝试获取锁，其实就是在这里去尝试获取同步状态。

一个AQS，你可以认为是代表了用于获取一把锁的这么一个控制器

AQS里面，有一个FIFO队列，用来排队的；还有一个state，代表了锁的这么一个状态

Lock.lock()

// 干活儿

Lock.unlock()

获取一把锁的时候，尝试去获取AQS里的state状态（锁的状态）

刚开始state的值是0（代表没有人获取这把锁），线程1是可以获取到锁的，然后就会将state设置为1；接着线程2来尝试获取state的值，但是此时state = 1，表示被别人锁了，所以线程2的信息构造成一个node放入FIFO队列中来排队，同时阻塞住线程2

如果线程1释放锁的时候，就会将state设置为0，此时会唤醒FIFO队列中的排在队头的那个线程2，然后线程2发现state = 0，就会获取锁，将state设置为1，同时线程2的node从FIFO队列里面出队。

其实这块我不打算在这里太过于深入的分析，因为反正是应付面试题，你能理解到这里就ok了，简单来说AQS就是一个基础并发框架，ReentractLock之类的都是基于AQS来的，比如ReentractLock里面，某个线程想要获取锁，那么就会走上面AQS定义好的基础逻辑，获取state值啥的。

包括多个线程争用一把ReentractLock锁的时候，就会出现排队，然后某个线程释放锁之后，按照FIFO的顺序后面的线程自动获取锁。就这个意思。所以AQS就是个基础框架，提供了获取锁，FIFO排队，释放锁的基础逻辑，让ReentractLock、Semaphore、CountDownLatch之类的并发包下的类来使用。

暂时理解到这个程度就ok了。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/06\_聊聊HTTP协议的工作原理是啥？HTTP 1.0 1.1 2.0的区别是啥？/笔记.docx**

1、面试题

说一下http的工作流程？http 1.0、http 1.1、http 2.0具体有哪些区别？

2、面试官心里分析

这个就是让你聊聊http，说白了http工作原理，你都知道了，发起个http，底层都是tcp、ip、以太网那块再走，一层一层包裹数据包。所以http的关键就是让你聊聊http请求和http响应的规范。

3、面试题剖析

http发起请求的底层原理，大家其实都知道了，理解了那个原理，就一通百通了。那么来聊下http请求和响应的规范吧。其实请求的报文，就是请求头、请求方法、请求正文，GET/POST啥的，应该都知道；请求头，自己百度一下吧，作为一个工程师必须知道。响应，状态行，响应头，响应正文，状态行，200,400,500，实在不想讲了；响应头，自己查一下。

http请求封装到应用层数据包，封装在tcp数据包，封装在ip数据包，封装在以太网数据包，如果过大，可能会拆成几个包，走以太网协议+交换机 -> 广播 -> 网关 -> 多个网关 -> 目标的机器 -> 一层一层拆包 -> http请求报文 -> 传递给tomcat -> spring mvc -> http响应 -> 一样的路径会去

最最底层，这个数据如何传输？走的是物理层，网线、光缆，所有数据都是0/1电路信号

http协议，其实是每个搞java必须会的基础。

互联网初期，一般一个网页几乎都没什么图片，当时就是挂一些文字，一个网页里就是一大坨的文字。http 1.0版本。

浏览器 -> 网站，互相之间是先要通过tcp三次握手，建立一个连接，浏览器和网站互相都给对方留出一份资源，浏览器发起http请求 -> tcp -> ip -> 以太网，网站上面去，网站返回一个响应，连接关闭，tcp四次挥手。释放掉浏览器和网站各自给对方保持的一份资源。

http 1.0要指定keep-alive来开启持久连接，默认是短连接，就是浏览器每次请求都要重新建立一次tcp连接，完事儿了就释放tcp连接。早期的网页都很low，没啥东西，就一点文字，就用这个没问题。但是现在，一个网页打开之后，还要加载大量的图片、css、js，这就坑爹了，发送多次请求。

早期，2000年之前，那个时候网页，都很low，当时你打开一个网页，就是说现场底层tcp三次握手，跟网站建立一个tcp连接，然后通过这个tcp连接，发送一次http请求，网站返回一个http响应（网页的html，里面有一大段文字），浏览器收到html渲染成网页，浏览器就走tcp四次挥手，跟网站断开连接了

到了后面，发现说2000之后，2010之后更不用说了，网页发展很迅猛，一个网页包含着大量的css、js、图片等资源。比如你请求一个网页，这个网页的html先过来，过来之后，浏览器再次发起大量的请求去加载css、js、图片，打开一个网页可能浏览器要对网站服务器发送几十次请求。

http 1.0，疯了，刚开始请求网页的html，tcp三次握手建立连接 -> 请求/响应 -> tcp四次挥手断开连接，接着再次要加载css、js、图片，要发送30个请求，上面的过程来30次，30次频繁的建立tcp连接以及释放tcp连接。很慢很慢。

其实最慢的不是说发送请求和获取响应，打开和释放连接，这都是很重的过程

http 1.1默认支持长连接，就是说，浏览器打开一个网页之后，底层的tcp连接就保持着，不会立马断开，之后加载css、js之类的请求，都会基于这个tcp连接来走。http 1.1还支持host头，也就可以支持虚拟主机；而且对断点续传有支持。

浏览器，第一次请求去一个网站的一个页面的时候，就会打开一个tcp连接，接着就在一段时间内都不关闭了，然后接下来这个网页加载css、js、图片大量的请求全部走同一个tcp连接，频繁的发送请求获取响应，最后过了一段时间，这些事儿都完了，然后才会去释放那一个tcp连接。大幅度的提升复杂网页的打开的速度，性能。

http 2.0，支持多路复用，基于一个tcp连接并行发送多个请求以及接收响应，解决了http 1.1对同一时间同一个域名的请求有限制的问题。二进制分帧，将传输数据拆分为更小的帧（数据包），frame（数据包，帧），提高了性能，实现低延迟高吞吐。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/27\_你能说说事务的几个特性是啥？有哪几种隔离级别？/笔记.docx**

1、面试题

事务的几个特点是什么？数据库事务有哪些隔离级别？MySQL的默认隔离级别是？

2、面试官心里分析

用mysql开发的三个基本面：存储引擎、索引，然后就是事务，你必须得用事务。因为一个业务系统里，肯定要加事务保证一堆关联操作，要么一起成功要么一起失败，对不对？所以这是聊数据库必问的一个问题。

最最最最最最基本的用mysql来开发，就3点：存储引擎（了解），索引（能建索引，写的SQL都用上索引），事务（了解事务的隔离级别，基于spring的事务支持在代码里加事务），咱们项目阶段一里的在mysql相关的也就这么几点。

存储引擎 -> innodb，索引，基本按照你的SQL的需求都建了索引（可能漏了部分索引忘了建），事务（@Transactional注解，对service层统一加了事务）

3、面试题剖析

**3.1 事务的ACID**

这个先说一下ACID，必须得知道：

（1）Atomic：原子性，就是一堆SQL，要么一起成功，要么都别执行，不允许某个SQL成功了，某个SQL失败了，这就是扯淡，不是原子性。

（2）Consistency：一致性，这个是针对数据一致性来说的，就是一组SQL执行之前，数据必须是准确的，执行之后，数据也必须是准确的。别搞了半天，执行完了SQL，结果SQL对应的数据修改没给你执行，那不是坑爹么。

（3）Isolation：隔离性，这个就是说多个事务在跑的时候不能互相干扰，别事务A操作个数据，弄到一半儿还没弄好呢，结果事务B来改了这个数据，导致事务A的操作出错了，那不就搞笑了。

（4）Durability：持久性，事务成功了，就必须永久对数据的修改是有效的，别过了一会儿数据自己没了，不见了，那就好玩儿了。

**3.2 事务隔离级别**

总之，面试问你事务，先聊一下ACID，然后聊聊隔离级别

（1）读未提交，Read Uncommitted：这个很坑爹，就是说某个事务还没提交的时候，修改的数据，就让别的事务给读到了，这就恶心了，很容易导致出错的。这个也叫做脏读。

（2）读已提交，Read Committed（不可重复读）：这个比上面那个稍微好一点，但是一样比较尴尬，就是说事务A在跑的时候， 先查询了一个数据是值1，然后过了段时间，事务B把那个数据给修改了一下还提交了，此时事务A再次查询这个数据就成了值2了，这是读了人家事务提交的数据啊，所以是读已提交。这个也叫做不可重复读，就是所谓的一个事务内对一个数据两次读，可能会读到不一样的值。

（3）可重复读，Read Repeatable：这个就是比上面那个再好点儿，就是说事务A在执行过程中，对某个数据的值，无论读多少次都是值1；哪怕这个过程中事务B修改了数据的值还提交了，但是事务A读到的还是自己事务开始时这个数据的值。

串行化：幻读，不可重复读和可重复读都是针对两个事务同时对某条数据在修改，但是幻读针对的是插入，比如某个事务把所有行的某个字段都修改为了2，结果另外一个事务插入了一条数据，那个字段的值是1，然后就尴尬了。第一个事务会突然发现多出来一条数据，那个数据的字段是1。如果要解决幻读，就需要使用串行化级别的隔离级别，所有事务都串行起来，不允许多个事务并行操作。

MySQL的默认隔离级别是Read Repeatable，就是可重复读，就是说每个事务都会开启一个自己要操作的某个数据的快照，事务期间，读到的都是这个数据的快照罢了，对一个数据的多次读都是一样的。

但是另外几个隔离级别都是提供的。

我们聊下MySQL是如何实现Read Repeatable的吧，因为一般我们都不修改这个隔离级别，但是你得清楚是怎么回事儿，MySQL是通过MVCC机制来实现的，就是多版本并发控制，multi-version concurrency control。

innodb存储引擎，会在每行数据的最后加两个隐藏列，一个保存行的创建时间，一个保存行的删除时间，但是这儿存放的不是时间，而是事务id，事务id是mysql自己维护的自增的，全局唯一。

事务id，在mysql内部是全局唯一递增的，事务id=1，事务id=2，事务id=3

id name 创建事务id 删除事务id

张三 120 122

2 李四 119 空

2 小李四 122 空

事务id=121的事务，查询id=1的这一行的时候，一定会找到创建事务id <= 当前事务id的那一行，select \* from table where id=1，就可以查到上面那一行

事务id=122的事务，将id=1的这一行给删除了，此时就会将id=1的行的删除事务id设置成122

事务id=121的事务，再次查询id=1的那一行，能查到吗？能查到，要求创建事务id <= 当前事务id，当前事务id < 删除事务id

事务id=121的事务，查询id=2的那一行，查到name=李四

事务id=122的事务，将id=2的那一行的name修改成name=小李四

事务id=121的事务，查询id=2的那一行，答案是：李四，创建事务id <= 当前事务id，当前事务id < 删除事务id

在一个事务内查询的时候，mysql只会查询创建时间的事务id小于等于当前事务id的行，这样可以确保这个行是在当前事务中创建，或者是之前创建的；同时一个行的删除时间的事务id要么没有定义（就是没删除），要么是必当前事务id大（在事务开启之后才被删除）；满足这两个条件的数据都会被查出来。

那么如果某个事务执行期间，别的事务更新了一条数据呢？这个很关键的一个实现，其实就是在innodb中，是插入了一行记录，然后将新插入的记录的创建时间设置为新的事务的id，同时将这条记录之前的那个版本的删除时间设置为新的事务的id。

现在get到这个点了吧？这样的话，你的这个事务其实对某行记录的查询，始终都是查找的之前的那个快照，因为之前的那个快照的创建时间小于等于自己事务id，然后删除时间的事务id比自己事务id大，所以这个事务运行期间，会一直读取到这条数据的同一个版本。

记住，聊到事务隔离级别，必须把这套东西给喷出来，尤其是mvcc，说实话，市面上相当大比重的java程序员，对mvcc是不了解的，这个东西很简单，结构大家居然不知道，真是相当大的差异！

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/05\_聊聊Socket的工作原理？Socket跟TCP IP之间是啥关系？/笔记.docx**

1、面试题

说说socket通信的原理？

2、面试官心里分析

其实不知道大家发现没有，网络相关的问题，都是围绕着所谓的七层模型，或者是四层模型去走的。聊完四层模型，接着就是一次请求的全过程，紧接着就是聊传输层的tcp的连接，然后就是传输层的tcp协议之上的socket编程，接下来还会聊聊应用层的http协议。

所以说，来吧，这都是最最基础的网络知识。

3、面试题剖析

其实说白了，socket就是在传输层里把tcp/ip协议给封装了一下，我们程序员一般都是面向socket来编程的，比如java原生就支持socket网络编程的。

大体来说这个步骤，就是我们搞一个ServerSocket无限等待别人来连接你，然后某个机器要跟你连接，就在本地创建一个socket去连接你，然后建立连接之后，在服务器上，ServerSocket也会创建出来一个socket的。通过客户端的socket跟服务端的socket进行通信，我给你写数据，你读数据，你给我写数据，我读数据，就这个过程。

当然这个底层，比如建立连接和释放连接，都是基于tcp三次握手和四次挥手的规范来搞的，包括基于tcp协议传输数据，其实就跟我们之前说的一样，都是封装个tcp数据包，里面有tcp报头，整了端口号啥的，然后封装在ip数据包里，最后封在以太网数据包里传递。

就这么回事，大家理解了就好了。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/30\_MySQL的SQL调优一般都有哪些手段？你们一般怎么做？/笔记.docx**

1、面试题

SQL调优的常用手段

2、面试官心里分析

说实话，这个其实就是针对你有没有最最基础的线上SQL跑的慢的优化能力

3、面试题剖析

如果是应付面试，我们实在是不可能深入讲mysql的SQL优化，以后架构班里都会深入讲解，但是这里给大家说一句，互联网公司的系统，一般很少需要复杂的SQL优化，为啥呢？因为我说过很多次了，保持SQL简单，一般90%的SQL都建议是单表查询，join等逻辑放java代码里实现，不要放SQL里。

既然是单表查询了，你觉得还能有什么性能问题么？对吧

如果某个线上SQL跑的慢，十有八九就是因为那个SQL没有用索引，所以这个时候，第一步就是去看MySQL的执行计划，看看那个SQL有没有用到索引，如果没有，那么就改写一下SQL让他用上索引，或者是额外加个索引。

我的面试突击课里就讲这种互联网公司最经典和常用的SQL优化手段，其他的大家为了面试准备，可以临时去网上搜个帖子，MySQL SQL优化，随便记住一些到时候说说即可。

我这里其实主要就是讲下怎么看SQL的执行计划，这个是码农必备能力，必须能看懂执行计划，一般其实就是看SQL有没有走索引，你倒是可以在这个环节重点说下你对执行计划这块的理解就ok

explain select \* from table，就ok了

table | type | possible\_keys | key | key\_len | ref | rows | Extra

table：哪个表

type：这个很重要，是说类型，all（全表扫描），const（读常量，最多一条记录匹配），eq\_ref（走主键，一般就最多一条记录匹配），index（扫描全部索引），range（扫描部分索引）

possible\_keys：显示可能使用的索引

key：实际使用的索引

key\_len：使用索引的长度

ref：联合索引的哪一列被用了

rows：一共扫描和返回了多少行

extra：using filesort（需要额外进行排序），using temporary（mysql构建了临时表，比如排序的时候），using where（就是对索引扫出来的数据再次根据where来过滤出了结果）

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/28\_你们项目里基于Spring的事务是怎么做的？事务传播怎么设置的？/笔记.docx**

1、面试题

spring的事务支持（注解事务、声明事务、编程事务、事务的传播机制）？执行某个操作，前50次成功，第51次失败。a 全部回滚；b 前50次提交，第51次抛异常。ab场景分别如何设置spring事务。

2、面试官心里分析

聊完上面那个问题，面试官估计心里对你感觉相当不错了，但是呢，事儿没玩，还得聊聊实际项目中，你的java系统里的事务咋玩儿的啊？这就涉及到了spring对事务的支持，然后重要的事务传播机制！

3、面试题剖析

这个，你一般就聊下，spring支持编程式事务，和声明式事务。编程式事务就是用个事务类TransactionTemplate来管理事务，这个一般现在没人傻到干这个事儿了；声明式事务分成在xml里配置个AOP来声明个切面加事务，一般现在也没人傻到干这个了；大部分情况下，都是用@Transactional注解。

这个@Transactional注解呢，根据阿里编码规范，一般建议加在方法级别，就是要事务的方法就加事务，不要事务的方法就别加事务，否则很多一个数据库操作的方法，你还加事务，你这不是。。。？可能有同学注意到我们项目阶段一，都是傻乎乎的在类上加了事务对吧？

对，其实不是我傻，是我装傻，我当时为了快速开发，所以都给扔类上了，类里所有方法都开启了事务，但是后续我们会慢慢优化这些细节的。

另外这个注解一般要加rollbackFor，就是指定哪些异常类型才要回滚事务

还有比较重要的，就是有个isolation属性，你可以自己手动调整事务的隔离级别，但是这个一般不调整，记住，别乱调整事务隔离级别，一般可重复读+mysql mvcc机制跑的很好，你别瞎折腾。

另外一个重要的事务属性，就是propagation，事务的传播行为，我们就重点先来聊一下事务的传播行为，这个在项目里可能确实是要用到的。其实说白了，这个事务的传播机制，就是说，一个加了@Transactional的事务方法，和嵌套了另外一个@Transactional的事务方法的时候，包括再次嵌入@Transactional事务方法的时候，这个事务怎么玩儿？

我们的项目阶段一里是不是有大量这种场景？呵呵，所以啊，真实复杂业务系统的好处就在这里，真好，后面我会慢慢优化这些细节，包括事务的传播机制等等，让你各种技术都在复杂业务系统里看看怎么玩儿的。

public class ServiceA {

@Autowired

private ServiceB b;

@Transactional

public void methodA() {

// 一坨数据库操作

for(int i = 0; i < 51; i++) {

try {

b.methodB();

} catch(Exception e) {

// 打印异常日志

}

}

// 一坨数据库操作

}

}

public class ServiceB {

@Transactional(propagation = PROPAGATION\_REQUIRES\_NEW)

public void methodB() throws Exception {

// 一坨数据库操作

}

}

项目阶段一里面，是不是有好多OrderService调用了MembershipService调用了WmsService

一共有7种事务传播行为：

（1）PROPAGATION\_REQUIRED：这个是最常见的，就是说，如果ServiceA.method调用了ServiceB.method，如果ServiceA.method开启了事务，然后ServiceB.method也声明了事务，那么ServiceB.method不会开启独立事务，而是将自己的操作放在ServiceA.method的事务中来执行，ServiceA和ServiceB任何一个报错都会导致整个事务回滚。这就是默认的行为，其实一般我们都是需要这样子的。

（2）PROPAGATION\_SUPPORTS：如果ServiceA.method开了事务，那么ServiceB就将自己加入ServiceA中来运行，如果ServiceA.method没有开事务，那么ServiceB自己也不开事务

（3）PROPAGATION\_MANDATORY：必须被一个开启了事务的方法来调用自己，否则报错

（4）PROPAGATION\_REQUIRES\_NEW：ServiceB.method强制性自己开启一个新的事务，然后ServiceA.method的事务会卡住，等ServiceB事务完了自己再继续。这就是影响的回滚了，如果ServiceA报错了，ServiceB是不会受到影响的，ServiceB报错了，ServiceA也可以选择性的回滚或者是提交。

（5）PROPAGATION\_NOT\_SUPPORTED：就是ServiceB.method不支持事务，ServiceA的事务执行到ServiceB那儿，就挂起来了，ServiceB用非事务方式运行结束，ServiceA事务再继续运行。这个好处就是ServiceB代码报错不会让ServiceA回滚。

（6）PROPAGATION\_NEVER：不能被一个事务来调用，ServiceA.method开事务了，但是调用了ServiceB会报错

（7）PROPAGATION\_NESTED：开启嵌套事务，ServiceB开启一个子事务，如果回滚的话，那么ServiceB就回滚到开启子事务的这个save point。

大家回头想想那个面试题，其实就是ServiceA里循环51调用ServiceB，第51次调用ServiceB失败了。第一个选项，就是两个事务都设置为PROPAGATION\_REQUIRED就好了，ServiceB的所有操作都加入了ServiceA启动的一个大事务里去，任何一次失败都会导致整个事务的回滚；第二个选项，就是将ServiceB设置为PROPAGATION\_REQUIRES\_NEW，这样ServiceB的每次调用都在一个独立的事务里执行，这样的话，即使第51次报错，但是仅仅只是回滚第51次的操作，前面50次都在独立的事务里成功了，是不会回滚的。

其实一般也就PROPAGATION\_REQUIRES\_NEW比较常用，要的效果就是嵌套的那个事务是独立的事务，自己提交或者回滚，不影响外面的大事务，外面的大事务可以获取抛出的异常，自己决定是继续提交大事务还是回滚大事务。

一般在单块系统开发，多人协作的时候比较常见，就是小A调用小B的模块，小A不管小B是成功还是不成功，自己都要提交，这个时候可以这么弄，就是说小B的操作不是构成小A的事务的重要组成部分，就是个分支。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/18\_提醒一下自行复习JVM的常见基础面试题/笔记.docx**

基础的jvm知识，我不打算讲，当然后续高性能的环节，jvm这块会重点讲解，因为线上比较常见的就是jvm full gc。

（1）说一下jvm内存分为哪几个区域？哪些是线程独享？哪些是线程共享？对象从创建到销毁的生命周期里是如何在内存区域中转移的？

（2）哪些内存区域会参与gc回收？什么情况下一个对象会被gc掉？为什么要在这个时候让对象被gc？了解哪些jvm垃圾回收器？jvm有哪些gc算法？各种gc算法的优缺点是什么以及适用场景？新生代和老生代的gc回收策略是什么？cms垃圾回收器的原理是什么？何时触发minor gc？何时触发full gc？

（3）说说jvm的类加载机制？都有哪些类加载器以及分别加载哪些文件？类加载器之间的父子关系是什么？什么是双亲委派模型？如何自定义自己的类加载器？自己的类加载器和java自带的类加载器的关系如何处理？以下两种类加载方式有什么区别？class.forName()和classLoader？

上面3块，都是jvm的基础知识，说白了，就是jvm内存区域、gc、类加载，大家完全可以根据问题查阅对应的博客或者是书籍，我觉得都说的很明白了，大家快速熟悉一下即可。

预报一下，高性能架构这块，会对jvm有非常深入的研究和介绍，让你成为jvm这个领域的专家，性能调优、oom故障处理、优化

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/04\_画一下TCP三次握手的流程图？为啥是三次而不是二次或者四次呢？/笔记.docx**

1、面试题

TCP三次握手和四次握手的工作流程是什么（画一下流程图）？为什么不是五次握手或者两次握手？

2、面试官心里分析

这个问题相当经典，大家可别以为就是考察应届生的，实际上在普通社招java面试中，一些大公司，很喜欢考察这个问题，尤其是后面第二个追加问题，让你聊聊为啥必须是三次握手，而不是两次呢？

3、面试题剖析

（1）tcp三次握手过程

通过传输层的tcp协议建立网络连接的时候，其实走的是三次握手的过程

建立三次握手的时候，TCP报头用到了下面几个东西，ACK、SYN、FIN。

第一次握手，客户端发送连接请求报文，此时SYN=1、ACK=0，这就是说这是个连接请求，seq = x，接着客户端处于SYN\_SENT状态，等待服务器响应。

第二次握手，服务端收到SYN=1的请求报文，需要返回一个确认报文，ack = x + 1，SYN=1，ACK = 1，seq = y，发送给客户端，自己处于SYN\_RECV状态。

第三次握手，客户端收到了报文，将ack = y + 1，ACK = 1，seq = x + 1

其实三次握手说白了，就是来回来去三次请求，每次请求带上一堆TCP报文头，根据报文头是否正确，就是越好的协议来建立连接。简单说就是这样。

（2）为啥不是2次或者4次握手呢？

假设两次握手就ok了，要是客户端第一次握手过去，结果卡在某个地方了，没到服务端；完了客户端再次重试发送了第一次握手过去，服务端收到了，ok了，大家来回来去，三次握手建立了连接。

结果，尴尬的是，后来那个卡在哪儿的老的第一次握手发到了服务器，服务器直接就返回一个第二次握手，这个时候服务器开辟了资源准备客户端发送数据啥的，结果呢？客户端根本就不会理睬这个发回去的二次握手，因为之前都通信过了。

但是如果是三次握手，那个二次握手发回去，客户端发现根本不对，就会发送个复位的报文过去，让服务器撤销开辟的资源，别等着了。

因为3次握手就够了，不需要4次或者5次浪费资源了。

（3）tcp断开连接的4次挥手

第一次挥手，客户端发送报文，FIN=1，seq=u，此时进入FIN-WAIT-1状态

第二次挥手，服务端收到报文，这时候进入CLOSE\_WATI状态，返回一个报文，ACK=1，ack=u+1，seq=v。客户端收到这个报文之后，直接进入FIN-WAIT-2状态，此时客户端到服务端的连接就释放了。

第三次挥手，服务端发送连接释放报文，FIN=1，ack=u+1，seq=w，服务端进入LAST-ACK状态

第四次挥手，客户端收到连接释放报文之后，发应答报文，ACK=1，ack=w+1，seq=u+1，进入TIME\_WAIT状态，等待一会儿客户端进入CLOSED状态，服务端收到报文之后就进入CLOSED状态。

# **File Path: /Users/xiaotingting/Downloads/学习资料/05\_互联网Java工程师面试突击训练unzip/资料/32\_提醒复习一下Linux的命令以及IO多路复用机制等基础知识/笔记.docx**

1、面试题

大的log文件中，统计异常出现的次数、排序，或者指定输出多少行多少列的内容。(主要考察awk) 、grep命令的使用（文件查找）、find命令

poll和epoll的工作原理

2、面试官心里分析

一般其实就是给你出个场景，让你写linux命令，常见的就是awk在日志文件里找某个异常的次数和排序之类的，或者是指定输出一个日志的多少行多少列。还有就是grep命令，查找东西么，包括find命令。

其实很简单，你大概知道常见的场景就可以了

坦白讲，BAT里的B系面试官爱问这个，虽然我也是B系出来的，但是我确实不爱问这个问题，因为B系面试官喜欢彰显自己的深厚计算机技术功底，所以有这样面试别人的一个传统，但是我后来在BAT里的A也待了很久，我面试的风格已经完全偏向A系了，就求落地、接地气，围绕项目问技术问题

像poll和epoll之类的，我平时一般确实很少直接这样子干问

3、面试题剖析

自己上网找几个例子，比如awk分析日志文件的博客，自己看看，我实在不想讲这类东西，也没什么可讲的。

然后grep命令，我们常用的，就是cat 日志文件 | grep ‘字符串啥的’，你自己在某个日志文件里搜某个东西，比如某个订单id对应的日志，常用的

find，自己去看看吧

我这里不愿意去讲这个东西，因为干讲这一个点讲不出来什么的，以后架构班课程里再深入讲解计算机基础的一些东西，这块，给大家推荐些博客自己去看看，比如：https://www.cnblogs.com/jeakeven/p/5435916.html

然后死记硬背一些，被人问到可以大概说说就行