# Prompt

假设你是一名Java后端面试官，正在考察一位面试者对Java 集合 并发 虚拟机、SpringBoot SSM、MySQL Redis、操作系统 计算机网络、Linux等知识的掌握情况。

我将给你面试题，请你从面试官的角度，给我面试官最想要面试者回答出来的的精简、专业的回答，分为以下几点回答我：

（1）知识索引（必须，提供该知识点的知识目录所属，比如Spring-IoC，比如MySQL-索引）

（2）简要回答（必须，提供精简、有逻辑的、一目了然的回答）

（3）详细回答（非必须，提供详细的具体的解释）

（4）延申问答（非必须，以我给你的面试问题，从面试官的角度，猜测他接下来可能会延申提问的三个问题及其相应的简要回答）

请注意格式方面：以上4个方面用四级标题（####）标注，他们之间用分隔线（----）隔开，每个方面内部详细的知识点如有需要请顺延标题级别（五级#####、六级######等）

我将给你别人的面试真题，请你为我在每道问题前面添加【】的格式，为我划分该面试问题所属分类，分类包含：【算法真题】、【Java并发】、【Java集合、JVM】、【SpringBoot SSM】、【MySQL、【Redis】、【操作系统】、【计算机网络】、【Linux】、【业务问题】。如果都不在上面的分类中（比如Kafka、RocketMQ），请为我添加【其他技术】

举例：   
输入：了解Redis分布式的部署方法吗？了解主从集群搭建吗？  
输出：【Redis】了解Redis分布式的部署方法吗？了解主从集群搭建吗？

接下来的是所有问题，分类后按照我给你的类别顺序将问题重新排序，但注意不要改变问题原本的内容。

# 美团日常实习面经

【算法真题】手撕：删除倒数第k个结点   
【算法真题】手撕：重排链表

【Java并发】实现多线程编程的方式有哪些？

【Java并发】创建线程池有哪些方式？  
【Java并发】线程池的核心参数？执行流程？  
【Java并发】线程有哪些状态？

【Java并发】现在有线程A和B如何实现A运行完以后可以再运行B？

【Java并发】如果服务器出现了线程泄露，你应该怎么怎么去排查解决？  
【JVM】讲一下你熟悉垃圾回收器？  
【JVM】新生代和老年代的占比有了解吗？

【SpringBoot-框架】SpringBoot框架有什么优势？核心注解有哪些？O  
【SpringBoot-自动装配】springboot的自动装配有了解吗？Y  
【SpringBoot-依赖注入】@AutoWire和@Resource区别？Y  
【SpringBoot-启动】springboot启动流程？N

【MySQL-数据库优化】MySQL什么情况下需要分库分表

【MySQL-锁机制】乐观锁和悲观锁了解吗？

【Redis-分布式】了解Redis分布式的部署方法吗？了解主从集群搭建吗？N

【Redis-分布式】论坛项目里面的维护帖子的排行的Zset如果数据量很大应该怎么办，取数据花的时间比较长，怎么办？O  
【Redis-缓存优化】为什么要用二级缓存？Redis如果是分布式的话你害怕过期的话，把过期时间设置的长一点不就可以了吗？N  
【Redis-淘汰策略】Redis的淘汰策略？Y  
【Redis-缓存策略】讲一讲Redis的缓存三剑客的各种触发原因和解决办法？Y  
【Redis-分布式锁】实习中Redis分布式锁的实现原理  
【Redis-集群部署】Redis集群部署有哪些方式？  
【Redis-分布式锁】在主从集群上使用setnx分布式锁，可能会有哪些问题，怎么解决？

【HTTP协议】http常见的方法有哪些？  
【HTTP协议】get和post方法的区别？  
【RPC框架】用过RPC框架吗？介绍一下dubbo。

【Quartz-调度】Quartz 你是单节点还是多节点部署？如果是线上系统多节点的情况下，单个结点的定时在节点宕机的情况下话，怎么去保证定时任务能够有效执行？N  
【消息中间件】kafka和RocketMQ有什么区别，实现细节上有了解吗？N  
【Kafka-性能优化】kafka怎么处理消息积压？O  
【Kafka-消息顺序】怎么保证kafka处理消息是有序且不重复的？O  
【消息中间件-RocketMQ】用RocketMQ主要是用来解决什么问题的？  
【RocketMQ-消息消费】RocketMQ什么情况下会出现重复消费的问题？   
【实习经历】介绍实习  
【实习经历】实习中遇到了什么技术难点吗？O  
【实习经历】实习的模块流量大小

腾讯校招面经 Java后端 面试时长1-2h

考察的知识还是比较多的，我这里简单给在大家列了一下：

* 操作系统：进程&线程、进程隔离性
* 数据结构：排序算法、排序稳定性、归并排序、快速排序
* MySQL：存储引擎、聚簇索引、B+树、索引失效、事务隔离级别、脏读、幻读
* Redis：数据类型、String 底层实现、热 key
* Java：ArrayList、Vector、HashMap
* MQ：消息队列选型、消息可靠性、消息确认机制、Kafka 、RocketMQ

总体考察的范围，就是**编程语言+计算机基础+后端组件**。

美团AI

TCP重传机制有哪些触发条件？超时重传和快速重传的区别？

TCP 重传机制的触发条件主要有以下两种：

**超时重传**：发送方在发送数据后，会启动一个定时器。如果在定时器超时之前没有收到接收方对该数据的确认（ACK），就会认为数据丢失或损坏，从而触发重传。

1. 由定时器超时触发，即经过了一定的时间间隔仍未收到确认
2. 通常等待时间较长，要等到定时器超时才进行重传，这可能导致较长的延迟
3. 一般会重传定时器超时对应的整个数据包或数据包组

**快速重传**：当接收方收到乱序的数据包时，会发送重复的确认给发送方。如果发送方连续收到多个（通常是 3 个）相同的重复确认，就会认为该确认号之后的数据可能丢失了，进而触发快速重传，而不需要等待超时。

1. 由收到多个重复确认触发，不需要等待定时器超时。
2. 能在较早阶段就发现数据丢失并进行重传，能更快地恢复丢失的数据，减少传输延迟。
3. 通常只重传收到重复确认所指示的丢失数据包，而不是重传整个窗口或更大范围的数据，更有针对性。

解释Linux管道的概念，给出一个使用管道的例子

在 Linux 中，管道（Pipe）是一种进程间通信（IPC）机制，它使用竖线 | 作为操作符。管道的作用是把一个命令的输出直接作为另一个命令的输入，这样就能让多个命令协同工作，完成复杂的任务。

举例：ls | grep ".txt$" | wc -l

ls 命令的输出被当作 grep 命令的输入，而 grep 命令的输出又成为了 wc -l 命令的输入，最终得出 .txt 文件的数量。

* **ls 命令**：ls 是 “list” 的缩写，用于列出指定目录中的文件和子目录。当不指定目录时，默认列出当前工作目录的内容。例如，在终端输入 ls 后，会显示当前目录下的所有文件和文件夹名称。此外，它还支持很多选项，如 ls -l 可以以长格式显示文件和目录的详细信息，包括权限、所有者、大小、修改时间等。
* **grep 命令**：grep 是 “Global Regular Expression Print” 的缩写，用于在文本中搜索匹配指定模式的行，并将匹配的行打印出来。在 grep ".txt$" 中，".txt$" 是一个正则表达式，其中 "txt" 代表要匹配的文本内容，$ 是正则表达式的元字符，表示行的结尾，所以 ".txt$" 表示匹配以 .txt 结尾的行。因此，grep ".txt$" 会从输入中筛选出以 .txt 结尾的行。
* **wc -l 命令**：wc 是 “word count” 的缩写，用于统计文件的行数、单词数和字节数。-l 是 wc 命令的一个选项，它表示只统计行数。因此，wc -l 会对输入的文本进行处理，输出其中的行数。

对explain关键字的理解，包括它的用途如何使用典型输出信息的含义，以及如何利用这些信息进行查询优化

对EXPLAIN关键字的理解

EXPLAIN 用于分析查询语句的执行计划，让开发者了解数据库如何执行查询，帮助找出查询性能瓶颈。

使用方法

* 定位潜在性能问题，为优化查询提供依据。
* **使用方法**： EXPLAIN SELECT \* FROM users WHERE age > 20;。

典型输出信息含义

type

连接类型，反映了查询查找数据的方式，是衡量查询效率的重要指标。不同的 type 值对应的查询效率差异很大，从优到劣的常见类型排序为：system > const > eq\_ref > ref > range > index > ALL。

* ALL 代表全表扫描，意味着数据库要遍历表中的每一行数据，查询效率最差。
* range 表示使用索引进行范围扫描，效率较好。
* const 则表示通过索引一次就能找到数据，效率最高。

key

实际使用的索引。索引在数据库查询中起着关键作用，能够显著提升查询速度。若 key 显示为 NULL，则表明查询未使用索引，可能会引发全表扫描，导致查询效率低下。因此，要保证查询语句能利用合适的索引，可通过 CREATE INDEX 语句创建索引，或对查询语句进行调整，以让其使用已有的索引。

rows

它是数据库估算的查询需要扫描的行数。该值并非精确值，但能反映查询的大致成本。一般来说，rows 值越小，查询所需扫描的数据量就越少，查询效率也就越高。你可以借助优化查询条件、添加合适的索引等方法来减少 rows 值。

* **id**：查询的序列号，反映查询中各个子查询或操作的执行顺序，数值越大越先执行。
* **select\_type**：查询类型，常见有 SIMPLE（简单查询）、PRIMARY（主查询）、SUBQUERY（子查询）等。
* **table**：查询涉及的表名。
* **type**：连接类型，反映查询的效率，从好到差常见有 system、const、eq\_ref、ref、range、index、ALL 等，ALL 表示全表扫描，效率最低。
* **possible\_keys**：可能使用的索引。
* **key**：实际使用的索引。
* **key\_len**：使用的索引长度，越短通常效率越高。
* **ref**：哪些列或常量被用于查找索引列上的值。
* **rows**：估算查询需要扫描的行数，值越小性能越好。
* **Extra**：额外信息，如 Using index 表示使用了覆盖索引，Using where 表示使用了 WHERE 子句过滤数据。

利用信息进行查询优化

* **索引优化**：若 possible\_keys 有索引但 key 为 NULL，考虑创建合适的索引；若 type 为 ALL，可尝试添加索引提高查询效率。
* **减少扫描行数**：通过 rows 估算值，优化查询条件，缩小扫描范围。
* **利用覆盖索引**：若 Extra 未显示 Using index，可调整查询语句或索引，让查询仅通过索引就能获取所需数据。

自动装箱和拆箱？给出一个例子

* **自动装箱**：在 Java 里，自动装箱是把基本数据类型自动转换为**对应的**包装类对象。例如，将 int 类型转换为 Integer 对象。
* **自动拆箱**：与自动装箱相反，自动拆箱是将包装类对象自动转换为对应的基本数据类型。例如，把 Integer 对象转换为 int 类型。

hashCode和equals的关系？为什么要重写hashcode和equals

hashCode 和 equals 的关系

在 Java 里，hashCode 和 equals 方法有着紧密联系，需遵循以下规则：

* 若两个对象 equals 比较结果为 true，那么它们的 hashCode 值必须相同。
* 若两个对象的 hashCode 值相同，它们的 equals 比较结果不一定为 true，即可能是不同对象产生了哈希冲突。

重写 hashCode 和 equals 的原因

* **equals 方法**：默认的 equals 方法比较的是对象的引用地址，也就是判断两个对象是否为同一个实例。但在很多情况下，我们希望比较的是对象的内容是否相等，所以需要重写 equals 方法来实现自定义的相等逻辑。
* **hashCode 方法**：在使用 HashMap、HashSet 等基于哈希表实现的集合类时，这些类会先通过 hashCode 值来确定对象在哈希表中的位置，再用 equals 方法判断对象是否相等。如果只重写了 equals 方法而不重写 hashCode 方法，就可能出现两个 equals 比较为 true 的对象，其 hashCode 值却不同，从而导致在集合中出现重复元素或者无法正确查找元素的问题。所以，当重写 equals 方法时，通常也需要重写 hashCode 方法，以保证它们的一致性。

解释Java中的线程，如何创建和启动一个线程

设计一个简单的文章热度计算系统考虑浏览量评论数和分享数等因素

追问：是否考虑过时间因素对文章热度的影响？比如，一篇文章中发布后的不同时间段内热度的变化情况，如果考虑的话，你会如何将时间因素整合到热度计算中呢？

在编程学习中如何处理挫折感和持续保持学习动力的？举一个具体的例子

群友分享

cookie和session有什么区别?

token,session, cookie的区别?

如果客户端禁用了cookie,session还能用吗?

如果我把数据存储到localStorage，和Cookie有什么区别?

什么数据应该存在到cookie，什么数据存放到Localstorage

JWT令牌和传统方式有什么区别?

JWT 令牌都有哪些字段?

JWT 令牌为什么能解决集群部署，什么是集群部署?

jwt的缺点是什么?

JWT 令牌如果泄露了，怎么解决，IWT是怎么做的?

前端是如何存储JWT的?

为什么有HTTP协议了?还要用RPC?

HTTP长连接与WebSocket有什么区别?

Nginx有哪些负载均衡算法?

Nginx位于七层网络结构中的哪一层?

1. 为何用分布式锁而非单锁最初采用的是单锁，但是考虑到项目后期水平扩展，采用了分布式锁（事实上是从教学的角度考虑教会我们分布式锁怎么用）

2. 锁的性能如何 redis 的高性能保证了锁的高性能，且锁的粒度是针对用户的。但是由于是单体 redis，所以可靠性不保证

3. 优惠券是预生成的还是实时生成的预生成。视频里采用了一个 post 请求添加到了“优惠券数据库表”

4. 为何不直接将券直接放入 redis 作为一个 list 使用这样做是可以的，但是也需要分布式锁保证“一人一单”，并没有多少复杂度的降低。

1.讲讲项目(模块，难点，亮点，架构设计)

2.责任链跟策略模式怎么实现的

3.多级缓存用的什么 过期策略选择

4.多级缓存一致性问题 不一致怎么发现

5.主流程下单跟查询

6.怎么扛高并发

7.接口防刷

8.令牌桶+分布式锁+消息队列具体怎么做的

9.限流算法的实现

10.座位库存怎么设计的

11.站点扣减流程

12.有考虑过座位的高并发处理吗

13.选座算法

14.状态机怎么实现的 有哪些状态 状态机别的作用

15.链路消息丢失处理

16.分布式事务具体怎么实现的 有没有涉及跨库事

务

17.有哪些表 核心字段

18.分库分表怎么做的 分片算法 带来的问题 如何评估数据

19.订单关单怎么做的 技术选型

20.支付模块怎么做的 对接流程 超时支付怎么处理

21.对账流程设计

22.幂等处理

23.分布式1D

24.候补功能思路