拼多多面经

一：看你在项目中用了 Redis，我们先聊聊 Redis 吧，常用的数据结构有哪几种，在你的项目中用过哪几种，以及在业务中使用的场景。Redis 的 hash 怎么实现的，rehash 过程讲一下和 JavaHashMap 的 rehash 有什么区别？Redis cluster 有没有了解过，怎么做到高可用的？

1. 常用的数据结构：

字符串（String），散列/哈希（hash），列表（list），无序集合类型（set），有序集合类型（sorted set）

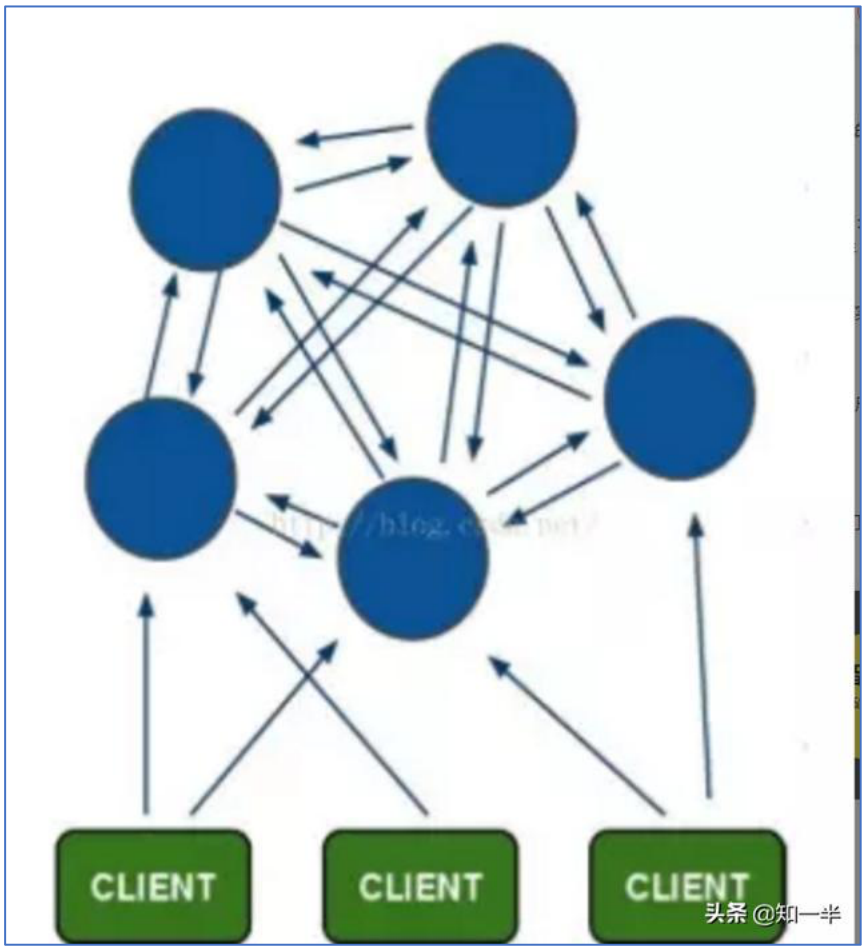
2. Redis 的 Hash 实现：

Redis 散列/哈希是键值对的集合。Redis 散列/哈希是字符串字段和字符串值之间的映射，但字段值只能是字符串，不支持其他类型。因此，他们用于表示对象。应用场景：存储用户的基本信息，等等。

3. Redis cluster：

Redis 最开始使用主从模式做集群，若 master 宕机需要手动配置 slave 转为 master；后来为了高可用提出来哨兵模式，该模式下有一个哨兵监视 master 和 slave，若 master 宕机可自动将 slave 转为 master，但它也有一个问题，就是不能动态扩充；所以在 3.x 提出 cluster 集群模式。

Redis-Cluster 采用无中心结构，每个节点保存数据和整个集群状态，每个节点都和其他所有节点连接。



4. 其结构特点：

- 所有的 Redis 节点彼此互联(PING-PONG 机制),内部使用二进制协议优化传输速度和带宽。
- 节点的 fail 是通过集群中超过半数的节点检测失效时才生效。
- 客户端与 Redis 节点直连,不需要中间 proxy 层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可。
- Redis-cluster 把所有的物理节点映射到[0-16383]slot 上（不一定是平均分配）,cluster 负责维护 node<->slot<->value。
- Redis 集群预分好 16384 个桶，当需要在 Redis 集群中放置一个 key-value 时，根据 CRC16(key) mod 16384 的值，决定将一个 key 放到哪个桶中。

二：Redis 集群和哨兵机制有什么区别？Redis 的持久化机制了解吗？你们在项目中是怎么做持久化的？遇到过 Redis 的 hotkey 吗？怎么处理的？

1. Redis 集群和哨兵机制有什么区别？

谈到 Redis 服务器的高可用，如何保证备份的机器是原始服务器的完整备份呢？这时候就需要哨兵和复制。

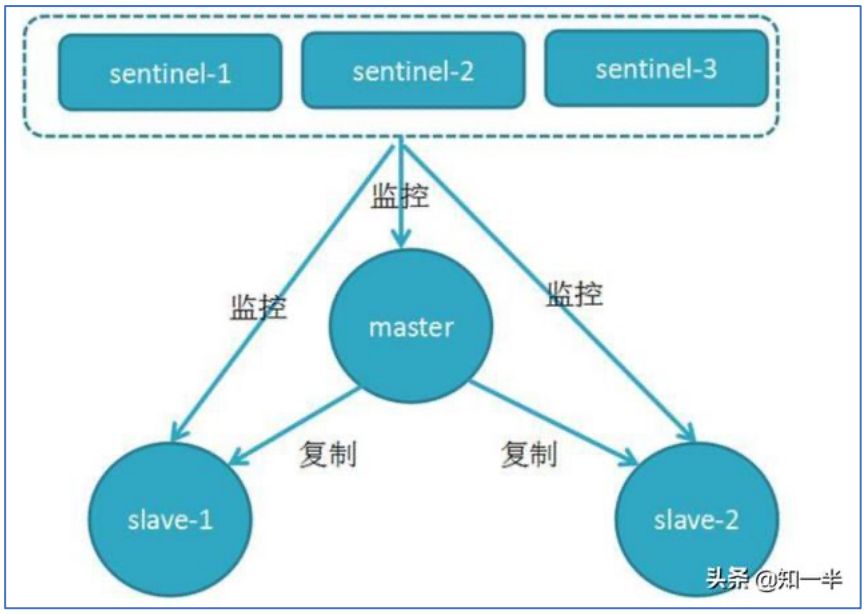
- 哨兵(Sentinel)：可以管理多个 Redis 服务器，它提供了监控，提醒以及自动的故障转移的功能。
- 复制(Replication)：则是负责让一个 Redis 服务器可以配备多个备份的服务器。
- Redis 正是利用这两个功能来保证 Redis 的高可用。

2. Redis 哨兵主要功能

- 集群监控：负责监控 Redis master 和 slave 进程是否正常工作。
- 消息通知：如果某个 Redis 实例有故障，那么哨兵负责发送消息作为报警通知给管理员。
- 故障转移：如果 master node 挂掉了，会自动转移到 slave node 上。
- 配置中心：如果故障转移发生了，通知 client 客户端新的 master 地址。

3. Redis 哨兵的高可用

原理：当主节点出现故障时，由 Redis Sentinel 自动完成故障发现和转移，并通知应用方，实现高可用性。



这个就是哨兵用来判断节点是否正常的重要依据，涉及两个新的概念：主观下线和客观下线。

- 主观下线：一个哨兵节点判定主节点 down 掉是主观下线。
- 客观下线：只有半数哨兵节点都主观判定主节点 down 掉，此时多个哨兵节点交换主观判定结果，才会判定主节点客观下线。

原理：基本上哪个哨兵节点最先判断出这个主节点客观下线，就会在各个哨兵节点中发起投票机制 Raft 算法（选举算法），最终被投为领导者的哨兵节点完成主从自动化切换的过程。

4. Redis 主从复制、哨兵和集群这三个有什么区别？

主从复制是为了数据备份，哨兵是为了高可用，Redis 主服务器挂了哨兵可以切换，集群则是因为单实例能力有限，搞多个分散压力，简短总结如下：

- 主从模式：备份数据、负载均衡，一个 Master 可以有多个 Slaves。
- sentinel 发现 master 挂了后，就会从 slave 中重新选举一个 master。

- cluster 是为了解决单机 Redis 容量有限的问题，将数据按一定的规则分配到多台机器。
- sentinel 着眼于高可用，Cluster 提高并发量。

1. 主从模式：读写分离，备份，一个 Master 可以有多个 Slaves。

2. 哨兵 sentinel：监控，自动转移，哨兵发现主服务器挂了后，就会从 slave 中重新选举一个主服务器。

3. 集群：为了解决单机 Redis 容量有限的问题，将数据按一定的规则分配到多台机器，内存/QPS 不受限于单机，可受益于分布式集群高扩展性。

三：Redis 是单线程的吗？单线程为什么还这么快？讲一讲 Redis 的内存模型？

Redis 内存划分：

Redis 作为内存数据库，在内存中存储的内容主要是数据（键值对）；通过前面的叙述可以知道，除了数据以外，Redis 的其他部分也会占用内存。

Redis 的内存占用主要可以划分为以下几个部分：

1. 数据

作为数据库，数据是最主要的部分；这部分占用的内存会统计在 used_memory 中。

Redis 使用键值对存储数据，其中的值（对象）包括 5 种类型，即字符串、哈希、列表、集合、有序集合。这 5 种类型是 Redis 对外提供的，实际上，在 Redis 内部，每种类型可能有 2 种或更多的内部编码实现；此外，Redis 在存储对象时，并不是直接将数据扔进内存，而是会对对象进行各种包装：如 redisObject、SDS 等；这篇文章后面将重点介绍

Redis 中数据存储的细节。

2. 进程本身运行需要的内存

Redis 主进程本身运行肯定需要占用内存，如代码、常量池等等；这部分内存大约几兆，在大多数生产环境中与 Redis 数据占用的内存相比可以忽略。这部分内存不是由 jemalloc 分配，因此不会统计在 used_memory 中。

补充说明：除了主进程外，Redis 创建的子进程运行也会占用内存，如 Redis 执行 AOF、RDB 重写时创建的子进程。当然，这部分内存不属于 Redis 进程，也不会统计在 used_memory 和 used_memory_rss 中。

3. 缓冲内存

缓冲内存包括客户端缓冲区、复制积压缓冲区、AOF 缓冲区等；其中，客户端缓冲存储客户端连接的输入输出缓冲；复制积压缓冲用于部分复制功能；AOF 缓冲区用于在进行 AOF 重写时，保存最近的写入命令。在了解相应功能之前，不需要知道这些缓冲的细节；这部分内存由 jemalloc 分配，因此会统计在 used_memory 中。

4. 内存碎片

内存碎片是 Redis 在分配、回收物理内存过程中产生的。例如，如果对数据的更改频繁，而且数据之间的大小相差很大，可能导致 Redis 释放的空间在物理内存中并没有释放，但 Redis 又无法有效利用，这就形成了内存碎片。内存碎片不会统计在 used_memory 中。

内存碎片的产生与对数据进行的操作、数据的特点等都有关；此外，与使用的内存分配器也有关系：如果内存分配器设计合理，可以尽可能的减少内存碎片的产生。后面将要说到

的 jemalloc 便在控制内存碎片方面做的很好。

如果 Redis 服务器中的内存碎片已经很大，可以通过安全重启的方式减小内存碎片：因为重启之后，Redis 重新从备份文件中读取数据，在内存中进行重排，为每个数据重新选择合适的内存单元，减小内存碎片。

四：我看你还用了 RabbitMQ，简单说一下 RabbitMQ 的工作原理？如何保证消息的顺序执行？Kafka 了解吗？和 RabbitMQ 有什么区别？你为啥不用 kafka 来做，当时怎么考虑的？

组成部分说明如下：

- Broker：消息队列服务进程，此进程包括两个部分：Exchange 和 Queue。
- Exchange：消息队列交换机，按一定的规则将消息路由转发到某个队列，对消息进行过虑。
- Queue：消息队列，存储消息的队列，消息到达队列并转发给指定的消费方。
- Producer：消息生产者，即生产方客户端，生产方客户端将消息发送到 MQ。
- Consumer：消息消费者，即消费方客户端，接收 MQ 转发的消息。

消息发布接收流程：

-----发送消息-----

1. 生产者和 Broker 建立 TCP 连接。
2. 生产者和 Broker 建立通道。
3. 生产者通过通道消息发送给 Broker，由 Exchange 将消息进行转发。
4. Exchange 将消息转发到指定的 Queue（队列）

----接收消息-----

1. 消费者和 Broker 建立 TCP 连接。
2. 消费者和 Broker 建立通道。
3. 消费者监听指定的 Queue（队列）。
4. 当有消息到达 Queue 时 Broker 默认将消息推送给消费者。
5. 消费者接收到消息。

五：我看你简历里说熟悉计算机网络，来聊一聊计算机网络吧。了不了解 tcp/udp，简单说下两者的区别？

1. 区别：

- TCP 面向连接（如打电话要先拨号建立连接）;UDP 是无连接的，即发送数据之前不需要建立连接。
- TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错、不丢失、不重复，且按序到达，UDP 尽最大努力交付，即不保证可靠交付。
- TCP 面向字节流，实际上是 TCP 把数据看成一连串无结构的字节流;UDP 是面向报文的
- UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如 IP 电话，实时视频会议等）。
- 每一条 TCP 连接只能是点到点的，UDP 支持一对一、一对多、多对一和多对多的交互通信。
- TCP 首部开销 20 字节;UDP 的首部开销小，只有 8 个字节。
- TCP 的逻辑通信信道是全双工的可靠信道，UDP 则是不可靠信道。

六：两次握手为什么不可以？为什么要四次挥手？

采用三次握手是为了防止失效的连接请求报文段突然又传送到主机 B，因而产生错误。失效的连接请求报文段是指：主机 A 发出的连接请求没有收到主机 B 的确认，于是经过一段时间后，主机 A 又重新向主机 B 发送连接请求，且建立成功，顺序完成数据传输。考虑这样一种特殊情况，主机 A 第一次发送的连接请求并没有丢失，而是因为网络节点导致延迟达到主机 B，主机 B 以为是主机 A 又发起的新连接，于是主机 B 同意连接，并向主机 A 发回确认，但是此时主机 A 根本不会理会，主机 B 就一直在等待主机 A 发送数据，导致主机 B 的资源浪费。

TCP 关闭链接四次握手原因在于 TCP 链接是全双工通道，需要双向关闭。client 向 server 发送关闭请求，表示 client 不再发送数据，server 响应。此时 server 端仍然可以向 client 发送数据，待 server 端发送数据结束后，就向 client 发送关闭请求，然后 client 确认。

七：TCP 怎么保证有序传输的？

- 应用数据被分割成 TCP 认为最适合发送的数据块。
- 超时重传：当 TCP 发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段。
- TCP 给发送的每一个包进行编号，接收方对数据包进行排序，把有序数据传送给应用层
- 校验和：TCP 将保持它首部和数据的检验和。这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到段的检验和有差错，TCP 将丢弃这个报文段和不确认收到此报文段。

- TCP 的接收端会丢弃重复的数据。
- 流量控制：TCP 连接的每一方都有固定大小的缓冲空间，TCP 的接收端只允许发送端发送接收端缓冲区能接纳的我数据。当接收方来不及处理发送方的数据，能提示发送方降低发送的速率，防止包丢失。TCP 使用的流量控制协议是可变大小的滑动窗口协议。
- 拥塞控制：当网络拥塞时，减少数据的发送。

八：time_wait 状态，这个状态出现在什么地方，有什么用？

1. 为实现 TCP 全双工连接的可靠释放

当服务器先关闭连接,如果不在一定时间内维护一个这样的 TIME_WAIT 状态,那么当被动关闭的一方的 FIN 到达时，服务器的 TCP 传输层会用 RST 包响应对方，这样被对方认为是有错误发生，事实上这只是正常的关闭连接工程，并没有异常。

2. 为使过期的数据包在网络因过期而消失

在这条连接上，客户端发送了数据给服务器，但是在服务器没有收到数据的时候服务器就断开了连接。

现在数据到了，服务器无法识别这是新连接还是上一条连接要传输的数据，一个处理不当就会导致诡异的情况发生。

九：HTTP 与 HTTPS 有啥区别？HTTPS 是怎么做到安全的？

HTTPS 安全的原因：

HTTPS 把 HTTP 消息进行加密之后再传送，这样就算坏人拦截到了，得到消息之后也看不

懂，这样就做到了安全，具体来说，HTTPS 是通过对称加密和非对称加密和 hash 算法共同作用，来在性能和安全性上达到一个平衡，加密是会影响性能的，尤其是非对称加密，因为它的算法比较复杂，那么加密了就安全了吗？不是的，HTTPS 除了对消息进行了加密以外还会对通信的对象进行身份验证。

HTTPS 并不是一种新的协议，而是使用了一种叫做 TLS（Transport layer secure）的安全层，这个安全层提供了数据加密的支持，让 HTTP 消息运行在这个安全层上，就达到了安全，而运行在这个安全层上的 HTTP 就叫做 HTTPS。

十、还有点时间，写个题吧

leetcode406. 根据身高重建队列

假设有打乱顺序的一群人站成一个队列。 每个人由一个整数对(h, k)表示，其中 h 是这个人的身高，k 是排在这个人前面且身高大于或等于 h 的人数。 编写一个算法来重建这个队列。

注意：

总人数少于 1100 人。

示例

输入:

[[7,0], [4,4], [7,1], [5,0], [6,1], [5,2]]

输出:

[[5,0], [7,0], [5,2], [6,1], [4,4], [7,1]]

思路：

第一步排序：

people：

[[7,0], [4,4], [7,1], [5,0], [6,1], [5,2]]

排序后：

[[7,0], [7,1], [6,1], [5,0], [5,2], [4,4]]

然后从数组 people 第一个元素开始，放入到数组 result 中，放入的位置就是离 result 开始位置偏移了元素第二个数字后的位置。如下：

1. people: [7,0]

插入到离开始位置偏移了 0 个距离的位置。

result: [[7,0]]

2. people: [7,1]

插入到离开始位置偏移了 1 个距离的位置，即插入到[7,0]的后面。

result: [[7,0], [7,1]]

3. people: [6,1]

插入到离开始位置偏移了 1 个距离的位置，即插入到[7,0]的后面。

result: [[7,0], [6,1], [7,1]]

4. people: [5,0]

插入到离开始位置偏移了 0 个距离的位置，即插入到[7,0]的前面。

result: [[5,0], [7,0], [6,1], [7,1]]

5. people: [5,2]

插入到离开始位置偏移了 2 个距离的位置，即插入到[7,0]的后面。

result: [[5,0], [7,0], [5,2], [6,1], [7,1]]

6. people: [4,4]

插入到离开始位置偏移了 4 个距离的位置，即插入到[6,1]的后面。

result: [[5,0], [7,0], [5,2], [6,1], [4,4], [7,1]]

这种算法体现了元素第二个数字与其插入位置的关系，所以通过简单的一个 for 循环就可以搞定。

面经资料来自网络