Redis 高频面试题及答案

目录

1.redis 是什么？

redis 是 nosql(也是个巨大的 map) 单线程，但是可处理 1 秒 10w 的并发（数据都在内存中）

使用 java 对 redis 进行操作类似 jdbc 接口标准对 mysql，有各类实现他的实现

类，我们常用的是 druid

其中对 redis，我们通常用 Jedis(也为我们提供了连接池 JedisPool)

在 redis 中,key 就是 byte[](string)

redis 的数据结构(value):

String,list,set,orderset,hash

每种数据结构对应不同的命令语句~

2.redis 怎么使用？

先安装好 redis，然后运行，在 pom 文件中引入依赖，在要使用 redis 缓存的类的 mapper.xml 文件配置 redis 的全限定名。引入 redis 的 redis.properties 文件（如果要更改配置就可以使用）

3.应用场景:

String：

1 存储 json 类型对象,2 计数器,3 优酷视频点赞等

list(双向链表)

1 可以使用 redis 的 list 模拟队列,堆,栈

2 朋友圈点赞(一条朋友圈内容语句,若干点赞语句)

规定:朋友圈内容的格式:

1,内容: user:x:post:x content 来存储;

2,点赞: post:x:good list 来存储;(把相应头像取出来显示)

hash(hashmap)

1 保存对象

2 分组

4.为什么 redis 是单线程的都那么快？

1.数据存于内存

2.用了多路复用 I/O

3.单线程

5.redis 也可以进行发布订阅消息吗？

可以，(然后可以引出哨兵模式（后面会讲）怎么互相监督的，就是因为每隔 2 秒哨兵节点会发布对某节点的判断和自身的信息到某频道，每个哨兵订阅该频道获取其他哨兵节点和主从节点的信息，以达到哨兵间互相监控和对主从节点的监控）和很多专业的消息队列系统（例如 Kafka、RocketMQ）相比，Redis 的发布订阅略显粗糙，例如无法实现消息堆积和回溯。但胜在足够简单。

6.redis 能否将数据持久化，如何实现？

能，将内存中的数据异步写入硬盘中，两种方式：RDB（默认）和 AOF

RDB 持久化原理：通过 bgsave 命令触发，然后父进程执行 fork 操作创建子进程，子进程创建 RDB 文件，根据父进程内存生成临时快照文件，完成后对原有文件进行原子替换(定时一次性将所有数据进行快照生成一份副本存储在硬盘中）

优点：是一个紧凑压缩的二进制文件，Redis 加载 RDB 恢复数据远远快于 AOF 的方式。

缺点：由于每次生成 RDB 开销较大，非实时持久化，

AOF 持久化原理：开启后，Redis 每执行一个修改数据的命令，都会把这个命令添加到 AOF 文件中。

优点：实时持久化。

缺点：所以 AOF 文件体积逐渐变大，需要定期执行重写操作来降低文件体积，加载慢

7.主从复制模式下，主挂了怎么办？redis 提供了哨兵模式（高可用）

何谓哨兵模式？就是通过哨兵节点进行自主监控主从节点以及其他哨兵节点 ,发现主节点故障时自主进行故障转移。

8.哨兵模式实现原理？（2.8 版本或更高才有）

1.三个定时监控任务：

1.1 每隔 10s，每个 S 节点（哨兵节点）会向主节点和从节点发送 info 命令获取最新的拓扑结构

1.2 每隔 2s，每个 S 节点会向某频道上发送该 S 节点对于主节点的判断以及当前 Sl 节点的信息，

同时每个 Sentinel 节点也会订阅该频道，来了解其他 S 节点以及它们对主节点的判断（做客观下线依据）

1.3 每隔 1s，每个 S 节点会向主节点、从节点、其余 S 节点发送一条 ping 命令做一次心跳检测(心跳检测机制)，来确认这些节点当前是否可达

2.主客观下线：

2.1 主观下线：根据第三个定时任务对没有有效回复的节点做主观下线处理

2.2 客观下线：若主观下线的是主节点，会咨询其他 S 节点对该主节点的判断，超过半数，对该主节点做客观下线

3.选举出某一哨兵节点作为领导者，来进行故障转移。选举方式：raft 算法。每个 S 节点有一票同意权，哪个 S 节点做出主观下线的时候，就会询问其他 S 节点是否同意其为领导者。获得半数选票的则成为领导者。基本谁先做出客观下线，谁成为领导者。

4.故障转移（选举新主节点流程）：

9.redis 集群（采用虚拟槽方式，高可用）原理（和哨兵模式原理类似，3.0 版本或以上才有）？

1.Redis 集群内节点通过 ping/pong 消息实现节点通信，消息不但可以传播节点槽信息，还可以传播其他状态如：主从状态、节点故障等。因此故障发现也是通过消息传播机制实现的，主要环节包括：主观下线（pfail）和客观下线（fail）

2.主客观下线：

2.1 主观下线：集群中每个节点都会定期向其他节点发送 ping 消息，接收节点回复 pong 消息作为响应。如果通信一直失败，则发送节点会把接收节点标记为主观下线（pfail）状态。

2.2 客观下线：超过半数，对该主节点做客观下线

3.主节点选举出某一主节点作为领导者，来进行故障转移。

4.故障转移（选举从节点作为新主节点）

10.缓存更新策略(即如何让缓存和 mysql 保持一致性）？

10.1 key 过期清除（超时剔除）策略

惰性过期（类比懒加载，这是懒过期）：只有当访问一个 key 时，才会判断该 key 是否已过期，过期则清除。该策略可以最大化地节省 CPU 资源，却对内存非常

不友好。极端情况可能出现大量的过期 key 没有再次被访问，从而不会被清除，占用大量内存。

定期过期：每隔一定的时间，会扫描一定数量的数据库的 expires 字典中一定数量的 key，并清除其中已过期的 key。该策略是前两者的一个折中方案。通过调整定时扫描的时间间隔和每次扫描的限定耗时，可以在不同情况下使得 CPU 和内存资源达到最优的平衡效果。

(expires 字典会保存所有设置了过期时间的 key 的过期时间数据，其中，key 是指向键空间中的某个键的指针，value 是该键的毫秒精度的 UNIX 时间戳表示的过期时间。键空间是指该 Redis 集群中保存的所有键。)

问：比如这么个场景，我设计了很多 key，过期时间是 5 分钟，当前内存占用率是 50%。但是 5 分钟到了，内存占用率还是很高，请问为什么？

Redis 中同时使用了惰性过期和定期过期两种过期策略，即使过期时间到了，但是有部分并没有真正删除，等待惰性删除。

为什么有定期还要有惰性呢？其实很简单，比如 10 万个 key 就要过期了，Redis 默认是 100ms 检查一波。如果他检查出 10 万个即将要清除，那他接下来的时间基本都是在干这些清空内存的事了，那肯定影响性能，所以他只会部分删除，剩下的等惰性。

10.2 Redis 的内存淘汰策略

Redis 的内存淘汰策略是指在 Redis 的用于缓存的内存不足时，怎么处理需要新写入且需要申请额外空间的数据。

noeviction：当内存不足以容纳新写入数据时，新写入操作会报错。

allkeys-lru：当内存不足以容纳新写入数据时，在键空间中，移除最近最少使用的 key。

allkeys-random：当内存不足以容纳新写入数据时，在键空间中，随机移除某个 key。

volatile-lru：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，移除最近最少使用的 key。

volatile-random：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，随机移除某个 key。

volatile-ttl：当内存不足以容纳新写入数据时，在设置了过期时间的键空间中，有更早过期时间的 key 优先移除。

11.缓存粒度控制？

12.如何防止缓存穿透？

（缓存穿透指的是查询一个根本不存在的数据，缓存层不命中，又去查存储层，又不命中。但如果有大量这种查询不存在的数据的请求过来，会对存储层有较大压力，若是恶意攻击，后果就）

12.1：缓存空值存在的问题：

12.2：布隆过滤器：

布隆过滤器存在的问题：相对来说布隆过滤器搞起来代码还是比较复杂的，现阶段我们暂时还不需要，后面实在需要再考虑去做，什么阶段做什么样的事情，不是说这个系统一下子就能做的各种完美。

13.无底洞优化？

造成原因：redis 分布式越来越多，导致性能反而下降，因为键值分布到更多的节点上，所以无论是 Memcache 还是 Redis 的分布式，批量操作通常需要从不同节点上获取，相比于单机批量操作只涉及一次网络操作，分布式批量操作 会涉及多次网络时间。 即分布式过犹不及。

14.雪崩优化

如果缓存层由于某些原因不能提供服务，于是所有的请求都会达到存储层，存储层的调用量会暴增，造成存储层也会级联宕机的情况。

15.热点 key 优化

当前 key 是一个热点 key（例如一个热门的娱乐新闻），并发量非常大。