Redis

> 缓存三兄弟 击穿 雪崩 穿透

击穿：缓存过期

1. 设置逻辑不过期（异步更新缓存过期时间 其他线程先返回过期的数据 只有一个线程更新缓存（锁）） 高可用
2. 强一致 互斥锁 直到更新完毕

雪崩：大量KEY失效，随机设置过期时间 或者 热点数据分布存储 热点数据永不过期 多级缓存 限流策略。。。

穿透：访问不存在的key 缓存没有 sql返回NUll后也不会存在redis 大量访问DB 导致DB负荷大

1.返回null值 并存在缓存里 设置一个过期时间防止负荷大

2.布隆过滤器 先查有没有 hash求值存在 01位图里面 因为是hash所以有误判率 可以设置为5%左右 redisson和guava有实现

> 双写一致

延迟双删 和读写锁 异步（canal和MQ通知）

> 持久化 *RDB*和*AOP*

一个存快照 ---> COPY ON WRITE

一个存写命令（这个更好 有三种设置 always everysecond和系统自己决定）

> 数据过期策略

因为不能一直存着 空间有限和性能有限

惰性删除 设置TTL 用到的时候会检查是否过期 检查到过期了再删除 对CPU友好但对内存不友好 因为没及时删

定期删除，定期对KEY检查（一定数量的KEY） 但是难以确定时长和频率 难以优化

SLOW模式 定时任务 默认10hz 每次不超过25ms (为了不影响主进程)

FAST模式：执行频率不固定，但两次间隔不低于2ms 每次耗时不超过1ms

两种策略配合用最好

> 数据淘汰策略

redis内存不够 要删除了 怎么办？ 淘汰策略

八种策略 volatile:对设置了TTL的key来操作 allkeys 对所有keys操作

random lru lfu 三种策略 3*2 还有前两种策略

1. noeviction:默认策略 满了不写新数据
2. volatile-ttl: 对设置TTL的key，比较key的剩余TTL值，TTL越小越先淘汰
3. allkeys-random: 对全体key，随机进行淘汰
4. volatile-random:对设置了TTL的key 随机进行淘汰
5. allkeys-lru:对全体key,基于LRU算法进行淘汰。
6. volatile-lru:对设置ttl的key，基于LRU算法进行淘汰。
7. allkeys-lfu:对全体key，基于LFU算法进行淘汰。
8. volatile-lfu:对设置了TTL的key 基于LFU算法进行淘汰

有冷热区分：allkeys-lru LFU不准确

没有冷热区分：allkeys-random

有置顶需求：volatile-lru （置顶数据不过期）

短时高频：lfu策略

allkeys-lru淘汰数据 留下来的都是经常访问的热点数据

默认是**noeviction** 满了会保存

> 分布式锁

不同数据在不同的服务器 集群部署 锁的服务器 不同服务器不同线程共享一把锁

setnx命令 要加失效时间 防止死锁

可以使用 不同的锁 **Key** 来锁定不同的业务，确保多个业务逻辑之间互不干扰。

锁的失效时间可能会比业务执行时间短 怎么办？

给锁续期

redisson实现的分布式锁 已经实现了给锁续期

看门狗机制

释放锁 同时要通知看门狗 不需要再监听了

其他线程 while循环尝试获取锁 但是会设置循环次数限制（获取时间限制） 不会一直循环导致死循环

基于lua脚本

redisson锁也是可重入的 用hash结构记录线程id key id 和重入次数 重入次数变0就删除信息

主从一致性

RedLock红锁 多个redis实例上创建锁 n/2+1 防止主节点宕机

但是实现复杂性能差

非要实现强一致性 应该使用zookeeper实现的分布式锁 用临时节点来管理锁 搞完就自动删除节点（其他线程监听） 让当前最小的节点来获取锁 （排队）

> 其他面试问题

集群方案

主从复制 哨兵模式 分片集群

> 主从复制

读写分离 写在主节点 同步到从节点 来读

实际上

还有

用replid id判断是否一致 来判断是否第一次同步 因为服务器变了的话 就要重新从0复制（全量同步）

执行bgsave用RDB文件来同步 RDB期间的命令（部分的AOP）(通过 Replication Buffer) 发到从节点来执行 防止漏了

offset判断是否落后 要不要更新 如果 `repl_offset` 发生偏移，则进行 增量同步。

> 哨兵模式

监控节点是否正常工作

master故障 会将一个slave升级

发生了故障转移 也要通知到客户端 写操作转移到新的master

基于心跳机制来监控 每隔1秒发Ping命令

主观下线和客观下线

前者是 没响应就主观下线 后者是超过指定数量的哨兵都发现下线了 那就客观下线

> 哨兵选择顺序

判断主和从节点断开时间长短 超过指定值就排除该从节点-》priority值（越小优先越高）---》**offset**值（越大数据越多 优先级越高）**---**》运行ID（越小越高）

> 脑裂问题

网络原因 主节点没寄 哨兵以为寄了 去选一个新的主节点 但是写入数据还在旧的主节点在写 网络恢复后 旧主节点被强制降为从节点 数据被清空

设置最少的slave节点 有slave才能写

设置同步延迟不超过设置时间

达不到要求就拒绝请求

1主1从就够了

> 分片集群

多个master 每个master保存不同数据 多个slave

master之间通过ping监测（心跳）彼此健康状态

访问任意节点 会转发到正确节点 --->哈希槽 CRC16 %16384 （像一致性哈希 但是一致性哈希有虚拟节点 负载均衡）

> *Redis*是单线程的

线程安全问题 避免不必要的上下文切换　使用非阻塞IO （NIO）

NIO主要就是实现了高效的网络请求

内核空间和用户空间

NIO的一种 **IO**多路复用

BIO 和AIO

BIO是阻塞IO 一直阻塞直到完成拿到数据

NIO不会阻塞 但是要等 反复read　此时非阻塞 （忙等）　数据就绪后 用户进程阻塞 拿数据

> *IO*多路复用

IO多路复用（也是NIO） 调用select 监听多个sockets　有socket准备好就返回readable 此过程用户进程阻塞　第二阶段 非阻塞 调用recvfrom从就绪socket读取数据 内核拷贝数据到用户数据 也是阻塞

监听的三种方式：select poll epoll

select和poll基本相似 但是Poll没有数量限制　都不知道哪个socket就绪 需要遍历

epoll 同时会通知socket是哪个　放入用户空间