1. 简述一下什么是redis?

redis本质上是一个key-value的内存数据库, 整个数据库都是在内存中操作的,redis会定时将数据刷到磁盘中进行持久化保存, 因为是内存操作, redis的性能非常强

1. redis支持哪几种数据类型？

redis支持多种数据类型

string 最基本的数据结构 二进制的安全字符串 最大为512MB

list 按照添加顺序保持顺序的字符串列表 最大为2 ^ 32 -1

hash key-value的一种数据结构

set 无需字符串的集合, 不存在重复数据

zset 在set类型上多一个score字段用于排序

1. redis的主要功能有哪些?
   1. 哨兵机制(Sentinel) 与主从复制(Replication)

Sentinel 可以管理多个redis服务器, 提供了监控, 提醒, 故障自动转移等功能

Replication 负责一台服务器进行配备多个备份服务器

Redis也是通过这两个功能保证高可用的

* 1. 支持事务

redis事务支持一次性执行多条命令, 并保证其原子性

* 1. 支持LUA脚本

在事务的基础上, 如果想对命令做出更复杂的判断就可与用到LUA脚本

* 1. 持久化

redis的持久化是将内存中的数据持久化到本地硬盘中, 当下次启动的时候就会重新加载数 据, 从而降低缓存丢失带来的影响

* 1. 集群

单台服务器的资源是有限的, 可以通过主从复制进行读写分离, 将一部分压力转到其他服 务器上, 集群可以做到 读写分离 及 数据容灾

1. redis是单线程的吗?

是单线程的, redis用队列技术将传统的并发访问变为穿行访问, 消除了传统的串行化的开销

1. redis为什么是单线程的?

因为多线程涉及到了锁以及线程的切换从而浪费CPU降低性能, redis的瓶颈不是CPU而是机 器内存和网络带宽, 所以将redis的模型设计为单线程的

1. redis的数据为什么存在内存中?

这个问题和为什么是单线程差不多, 因为要追求性能

将数据放在内存中可以达到最快的读写性能

如果放到磁盘中 I/O会严重影响redis的性能

1. redis的优势?

速度快: 因为在内存中查找 时间复杂度为O(1)

数据类型丰富: string list hash set zset

支持事务: 开启事务后操作都是原子性的

特性丰富: 可以给key设置过期时间, 到期自动失效

1. redis的劣势?

redis的主要缺点是数据库容量受内存的限制, 不能用于海量数据的高性能读写

因此redis的使用场景在小数据量的高性能操作上

1. 类似的缓存有哪些?

memcached：单进程多线程模型

mongodb: 支持丰富的数据表达和索引的非关系型数据库

1. redis的单点吞吐量?

单点TPS: 8w/s 单点QPS: 10w/s

TPS: 应用系统每秒最大的处理请求数量

每秒钟可以处理的事务次数, 应用系统一秒钟能完成多少个事务的处理

QPS: 应用系统每秒最大能接受的访问量

每秒钟处理完的请求次数, 具体指发出请求到响应的返回结果, 可以理解为有一个变量 count, 每次请求都会 count++, QPS是指一秒钟后count的值

1. redis中一个value的最大限制是多少

String最大512MB 其他类型最大1GB (2 ^ 32 -1),

但是string最好不要超过10KB, 其他的最好不要超过5000个元素

1. redis相比memcached有什么优势?

memcached的类型单一 只有string, 而redis的数据类型有很多

redis的速度要比memcached快

redis支持持久化

redis支持数据的备份, 及master-slave模式

1. redis的数据淘汰策略

在redis的配置文件中允许用户配置最大使用内存的大小 server.maxmemory 当内存达到一定 阀值的时候, 会启用数据的淘汰策略

1. volatile-lru 从设置过期时间的数据集淘汰最近最少使用的数据
2. volatile-ttl 从设置过期时间的数据集淘汰将要过期的数据
3. volatile-random 从设置过期时间的数据集淘汰任意数据
4. allkeys-lru 从数据集淘汰最近最少使用的数据
5. allkeys-random 从数据集淘汰任意数据
6. noenviction 禁止数据淘汰, 会出现OOM, 可读不可写
7. redis的集群方案
8. twemproxy
9. codis

目前用的最多的方案, 基本与twemproxy一致, 他支持节点数量改变的情况下

旧节点的数据恢复到新节点

1. redis cluster3.0 自带的集

他的特点是分布式算法不是一致性的hash, 而是hash槽的概念 支持节点设置从节点

1. redis的读写分离模型

可以增加Slave的数量, 读的性能会线行增长, 为了避免Master的单点故障, 集群一般都有两个

Master进行双机热备, 所以集群的读写效率都十分的高

Salve负责读, 所以Salve的数量决定读的效率

Master负责写, 所以Master的数量决定写的效率

读写分离架构的缺陷在于, 不管是Master还是Salve, 他们都要保存完整的数据

如果在数据量很庞大的情况下, 集群的拓展能力还是受限于节点的存储能力

对于写操作密集的程序 (Write-intensive), 读写分离并不适合

1. redis的分片模型

为了解决读写分离的缺陷, 可以运用数据的分片模型

将每个节点看做为一个Master, 通过业务来实现数据的分片

1. redis哨兵有什么作用?

集群监控: 负责监控集群Master和Slave是否正常工作

消息通知: 如果某个redis实例故障, 哨兵负责发送消息作为警报通知管理员

故障转移: 如果Master Node挂了, 会自动转移到Slave Node上

配置中心: 如果发生故障转移, 哨兵会通知client新的master地址

1. redis哨兵的高可用?

当主节点发生故障时, 由哨兵自动完成故障的发现和转移, 并通知应用方, 实现高可用

1. redis哨兵的工作流程?

简述版:

1. 哨兵机制建立了多个哨兵节点(进程), 共同监控数据节点的运行状况
2. 同时哨兵节点之间也会互相通信, 交换对主从节点的监控状况
3. 每隔一秒每个哨兵都会向整个集群 (Master节点+Slave节点+其他Sentinel) 进程发送一次 ping指令做一次心跳检测

这就是哨兵节点判断节点是否正常工作的重要依据

完整版:

1. 每个哨兵以每秒的频率向发所知的master、slave、sentinel发送一个PING命令
2. 如果节点响应PONG的时间如果超过了down-after-milliseconds指定的值, 那么这个节点就会被当前哨兵标记为主观下线状态
3. 如果master被标记为主观下线状态, 那么所有正在监控他的哨兵都会以每秒的速度发送PING确定是否进入了主观下线的状态
4. 如果有足够多的sentinel (大于等于配置文件的值) 确定master为主观下线状态, 则master会被标记为客观下线状态
5. 在一般情况下, sentinel每十秒向他监控的master、slave发送INFO命令
6. 当master被sentinel标记为客观下线状态时, sentinel回想已下线的master的所有slave发送INFO命令, 并且频率从10秒一次变为1秒一次
7. 若没有足够多的sentinel同意master下线, master的客观下线状态会被移除

若master重新向sentinel发送的PING命令返回有效, master的主观下线状态会被移除

1. redis 的 INFO replication的作用是什么?

列出从服务器的列表, 可以看出最后一次向从服务器发送命令距离现在多少秒

1. redis哨兵的主观下线和客观下线是什么?

主观下线: 一个哨兵节点判定主节点挂掉是主观下线 (单方面)

客观下线: 如果有半数哨兵节点都判断主节点主观下线, 此时哨兵节点会交换判定结果, 才会

判断主节点客观下线 (真的下线了)

原理: 哪个哨兵节点最先判断出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会去完成主从自动化切换的过程

1. redis如何进行故障转移?
2. 由sentinel节点监控主节点是否出现故障, sentinel会向master发送心跳信息PING 来确定

master是否存活, 如果master在一定时间没没有回复PONG或者回复了一个错误的信息,

那么当前哨兵会单方面(主观)认为master挂了, 会要求其他sentinel进行确认这个master是否真的挂掉, 如果确认挂掉, 则认为客观下线

1. 当master客观下线, sentinel节点之间会进行选举领导者进行失败转移 (领导者一般是主观下线的发现者)
2. 由哨兵领导者进行故障转移, 过程和主从复制一样, 但是是自动的
3. 故障转移后会生成新的拓扑结构图
4. 过滤掉不健康的(下线或断线)、没有回复哨兵ping响应的从节点
5. 选择slave-priority优先级最高的节点
6. 如果没选到, 则会选择复制偏移量最大的(复制最完整)的从节点
7. redis故障转移的流程?
8. 将slave1脱离原节点, 升级为主节点 slaveof no one
9. 将slave2指向新主节点的从节点 slaveof new master
10. 通知客户端主节点已改变
11. 将原来的主节点指向新主节点的从节点 slaveof new master
12. redis领导者哨兵节点是如何选举的?
13. 每个在线的哨兵节点都可以成为领导者, 比如哨兵3确认了主节点主观下线时, 他会向其他哨兵节点发送is-master-down-by-addr命令, 征求将自己设置为领导者, 由领导者 (他) 进行失败转移
14. 当其他节点收到选举命令时可以同意也可以拒绝
15. 如果哨兵3发现自己的票数大于等于 (sentinels) / 2+1时, 就会成为领导者, 如果没有达到则继续选举
16. redis如何防止master不在安全的情况下执行写命令?

min-slaves-to-write 3

min-slaves-max-lag 10 默认关闭

该配置表示: 从服务器数量小于三个或者从服务器的延迟大于等于10s时, master将拒绝执行写命令, lag的值应该是0或1 如果高于1则证明主从连接有问题, slave的lag值可以通过 INFO replication命令查看

在分布式的环境下可以进行配置, 合理的配置可以降低主从架构之间因为网络区间造成的数据不一致问题

1. redis如何进行命令丢失检测?

如果发生网络故障, 主服务器发送给从服务器的命令丢失, 那么从服务器向主服务器发送

REPLCONF ACK命令后, 主服务器的复制偏移量就会大于从服务器的复制偏移量, 然后主服务器会根据从服务器小于自己的复制偏移量在 积压缓冲区 里寻找从服务器丢失的数据, 并重新发送给从服务区

主服务器向从服务器补发缺失数据的操作原理和部分重同步操作的原理相似, 区别在于: 补发缺失数据的操作在主从服务器没有断线的情况下执行, 而重同步操作则在主服务器和从服务器断开后 重新连接完毕后执行

1. redis心跳检测的作用是什么?

心跳检测流程: 在命令传播阶段, 从服务器默认以每秒的速度向主服务器发送命令

(REPLCONF ACK <offset>) offset是从服务器的复制偏移量

作用: 检测主服务器的网络连接状态, 实现辅助min-slave选项, 检测命令的丢失

1. 简述一下redis的主从复制流程?
2. 从库向主库发送sync (同步)命令
3. 主库接收到同步命令后会fork子进程进行创建快照 (RDB)文件, 在此之间的所有写操作都会 缓存到积压缓冲区
4. 当主库生成完快照后会向从库发送快照和积压缓冲区的数据, 从库接收到后会将数据载入
5. 当主从复制完毕后, 主库接收到写命令都会发送给从库
6. 简述一下redis的主从断连后恢复连接怎么进行数据同步?

在redis2.8以后 当主从断开后会根据最近命令的偏移量进行增量复制

1. 当主从重连后, 从库会向主库发送 psync(id) 这个id是最近一条记录的ID
2. 如果主库id和发来的从库id相同就证明没有新数据 完成同步操作
3. 如果id不相同, 则会去找积压缓冲区的数据进行复制初始化
4. 简述一下什么是redis的乐观复制?

在redis主从复制的情况下, redis采用了乐观复制策略

乐观复制就是容忍在一段时间内主从数据是不同的, 但是数据最终还是保持一致性的

具体来说就是, redis主从复制之间复制的过程本来就是异步的, 这就证明主程序执行完写操作立马将执行结果告诉客户端, 则不会等待从库收到命令之后返回给客户端, 这一特性保证了性能不受影响, 但可能会产生数据的不一致, 当主服务器执行完一条命令后, 主机的数据已经改变, 主机数据同步到从机之前, 如果主从之间断开连接, 则会造成数据的不一致

1. 简述一下redis主从复制的作用?

读写分离: 不仅可以提高服务器的效率, 而且可以根据请求的规模增加或减少从库

数据备份: 数据备份到从机, 就算主机机器故障, 也会有其他从机顶上来

1. 简述一下redis集群的作用?

负载均衡: 解决redis容量问题, 降低服务器负载, 降低单节点的读写能力, 将读写 分离到不同的服务器上, 提高数据的可用性

1. redis集群在什么情况下会导致数据的不可用?

假设当前集群有三个节点A,B,C 并没有进行主从复制, 这时如果B 宕机了, 那么集群就会缺少 5501-11000这个范围的槽, 导致集群的不可用

1. redis的持久化策略有哪些?

redis提供了两种持久化策略: AOF 和 RDB

RDB: 默认开启, 会按照指定时间将内存中的数据快照备份到本地磁盘中, 当Redis再次启动的 时候会将数据再次恢复到内存中

RBD工作流程: redis会单独创建一个子进程 (fork()), 将当前父进程的数据复制到子进程中, 子 进程会将数据写到一个临时文件中, 当持久化过程完毕后, 子进程会将临时文件覆盖掉之前的 持久化文件, 之后退出子进程并释放内存

注意: 每次持久化都会将主程序的数据复制一遍, 导致内存与CPU的开销过大, 如果此时内存 不足, 服务器会阻塞, 等待持久化执行完毕后才回释放内存 每次持久化都会将内存的数据全 部写入磁盘一次, 如果数据量大, 并且频繁地进行写操作, 必然会引起大量I/O操作, 严重影响 性能, 最后一次持久化还有可能导致数据的丢失

AOF: 默认关闭, 以日志的形式记录所有的写操作, 只需要追加文件, 但不需要修改文件, redis 启动时, 会将日志文件从头到尾执行一遍以完成恢复操作 (包括flush也会执行)

AOF触发方式: 有写操作就添加 每秒定时写(也可能丢失数据)

AOF工作流程: 因为AOF默认是拼接的方式, 所以文件会越来越大, 针对这个问题, 增加了 重写机制, 当日志大到一定程度时, 会fork出一个新的进程, 用来遍历内存中的数据, 每条数据 对应一条set语句, 写到临时文件中, 再替换到旧的日志文件中, 类似于RDB持久化, 触发重写 机制的条件是: 当 AOF文件大小是上次重写后大小的1倍且大于64MB时才会触发

当两种持久化方式都开启后, redis会优先使用AOF进行数据的恢复, 一般情况下使用默认的 RDB 已经足够了, 相比AOF, RDB更适合数据的备份, 并且回复数据也比较快

开启持久化缓存会对性能有一定的影响, 特别是当内存满的时候, TPS会下降不止一点, 因此:

如果是做缓存可以不用开启持久化

1. redis常见的性能问题和解决方案?
2. Master最好不要做任何持久化工作 AOF 和 RDB
3. 如果数据重要, 让从机开启AOF持久化, 设定每秒同步一次
4. 为了保证主从复制的稳定性与速度, Master 和 Slave最好在同一个局域网
5. 尽量避免在压力很大的Master 添加Slave
6. 主从复制不要用树结构, 最好用单向链表结构

如:Master1 < Slave1 < Slave2 < Slave3...

这样的架构可以解决单点故障, 实现Master和Slave的切换, 如果Master挂了, 可以立即启用Slave1作为主机, 其他不变

1. redis支持的Java客户端有什么?

Redisson、Jedis、lettuce等等, 官方推荐使用Redisson

1. redis哈希槽的概念?

redis集群没有使用一致性hash, 而是采用桶的概念, 当需要redis集群添加数据的时候, 根据CRC16(key) mod 16384 (对key的值取模), 决定数据放到哪个桶中

1. redis最大节点数是多少?

集群预分好16384和桶 (哈希槽)

1. redis主从复制模型是什么?

为了使部分节点挂掉或者大部分节点无法通信的情况下集群仍然可用, 所以集群使用了主从复 制的模型, 每个节点都会有N-1个复制品

1. redis集群写操作会丢失吗?

redis的写操作并不能保证强一致性, 这就意味着在集群下一些特殊的条件下数据可能会导致写 操作的丢失

1. redis集群之间是如何复制的?

主从之间异步复制

1. redis如何内存调优?

尽可能使用散列表 (hash), 因为hash在内存中占用的资源非常的少, 可以将实体对象抽象到 hash中

1. redis回收进程是如何工作的?

当客户端执行了写操作, redis会检查内存信息, 如果超过了maxmemory会执行对应的回收策略

1. redis回收进程的算法是什么?

LRU策略, 回收最近最少使用的数据

1. redis的管道的作用?

管道(pipelining) 一次请求/响应能实现处理新的请求, 即使原来的请求还未被响应, 这样就可以 将多个命令发送到服务器, 而不用等待回复, 在最后一个步骤中取得该回复

1. redis的事务是什么?

事务是一个单独的隔离操作: 开启事务后所有的命令都会序列化, 按顺序的执行, 在事务执行中, 不会被其他客户端发送来的命令打断, 这保证了命令是作为一个独立的单位执行的, redis的事 务是原子操作

1. 事务的相关命令有哪些?

MULTI 开启事务

EXEC 执行事务

DISCARD 回滚事务

WATCH 对某个key加锁 如果这个key被其他客户端修改了 则导致事务失败

UNWATCH 释放锁

1. redis事务可能发生错误?

命令无法进入队列:

如 incrs id 立即报错但不会停止, exec会报错, 事务不会执行

调用EXEC命令执行失败:

如 incr name 不会立即报错, exec执行到那一条时报错, 事务最终不会执行

1. redis的过期和永久有效怎么设置?

EXPIRE key 10 设置key10秒过期

PERSIST key 将key设为永久有效

1. redis应用场景有哪些?

Session共享 (单点登录) 可以对key设置过期时间

页面缓存

队列 list或set的pop\push

排行榜 zset的score

计数器 incr或decr

订阅/发布

分布式锁

1. 如果mysql中有2000w数据, redis有20w数据, 怎么保证redis的数据是热点数据?

redis内存数据集大小上升到一定大小的时候, 就会施行数据淘汰策略, 指定淘汰策略为

allkeys-lru就可以了

1. 简述一下什么是缓存雪崩?

当缓存中的大量数据在同一时间失效, 这时所有的请求都会到数据库上, 对DB及CPU造成巨大的压力

1. 怎么解决缓存雪崩?

方案1: 排队加锁, 如使用MQ 避免key失效时大量请求到数据库

方案2: 建立备份缓存, A缓存和B缓存, A设置过期时间, B不设置, 先从A读, 没有读到就读B, 并且更新A缓存和B缓存

方案3: 设置缓存时再加一个随机时间, 如缓存为5分钟, 再给他加随机的分钟, 这是为降低群体失效概率, 一定程度上避免了缓存雪崩

方案4: 如果是分布式情况下, 可以用redis的setnx来加锁

方案5: 设置集群的高可用, 如果其他机器宕机, 可以有别的机器顶上来, 可以防止一定的风险

方案6: 使用netfix的hystrix, 做资源的线程池的隔离, 从而达到保护主线程

1. 简述一下什么是缓存穿透?

当请求数据库不存在的数据, 缓存也肯定没有, 请求会到数据库上, 造成数据库压力过大

1. 怎么解决缓存穿透?

方案1: 使用布隆过滤器, 使用一个足够大的bigmap, 用于存储可以访问的key, 不存在的key直接被过滤

方案2: 访问DB后, 当没有获取到数据时, 将空值也写入缓存, 可以设置较短的过期时间, 当其他用户再次查询这个不存在的数据, 那么不会再去数据库查询直接返回空

1. 简述一下什么是缓存击穿 (缓存并发)?

一个存在的key, 在缓存过期的那一瞬间, 同时有大量的请求, 这些请求都会击穿到DB, 对数据库造成压力 (热点key在高峰期失效, 瞬间大量请求落到数据库), 有可能造成缓存的频繁更新

1. 怎么解决缓存击穿?

方案1: 设置互斥锁 mutex: 单机使用 lock 分布式使用setnx (set if not exists)

方案2: 异步构建缓存

1. 简述一下什么是缓存预热?

缓存预热就是在系统上线前将相关的缓存数据直接加载到redis, 避免了缓存首次未命中,

大型活动或秒杀活动都需要提前预热

1. 怎么解决缓存预热?

方案1: 直接写个单元测试, 上线的时候手动操作一下

方案2: 如果数据量不大, 可以在项目上线之后直接加载

方案3: 定时器进行缓存刷新

1. 简述一下什么是缓存更新?

除了redis自带的6中缓存数据淘汰策略外, 我们需要根据业务需求定义自己的缓存淘汰策略

1. 缓存更新策略有什么?

常见的策略有两种:

策略1: 定时清理过期的缓存

策略2: 当有请求进来时, 先判断缓存中的数据时候过期, 过期了就去得到新数据并更新

策略1的缺点是维护大量key非常麻烦

策略2的缺点是每次请求都要判断失效, 逻辑上麻烦

1. 简述一下什么是缓存降级?

当访问量剧增、服务器出现问题 (响应慢或不响应的时候)、非核心服务影响到核心服务性能的时候, 仍要保证服务的可用性, 系统会根据一些关键数据进行自动降级, 也可以通过配置进行手动降级, 进行缓存降级前需要考虑是否可以丢卒保帅, 那些可以降级

降级的主要目的是保证核心服务的可用性, 有些服务是不能降级的 如:购物车

可以查看日志级别来判断是否需要降级:

一般: 比如有些服务因为网络波动或正在上线而超时, 可以自动降级

警告: 有些服务在一段时间内成功率有波动(95% - 100%), 可以自动降级或人工降级, 并 发送警告

错误: 服务的可用率低于90%, 或者连接池崩溃, 或者访问量猛增到系统能承受的最大阀 值, 此时可以根据情况自动或人工降级

严重错误: 因为特殊原因数据发生错误, 需要人工紧急降级

1. 简述分布式缓存会遇到的问题?

缓存一致性问题:

1: 缓存系统与底层数据的一致性

2: 有继承关系的缓存之间的一致性(二级缓存)

3: 多个缓存副本之间的一致性

缓存雪崩和缓存穿透的问题

1. redis为什么要禁用keys ?

因为redis是单线程的, 导致redis被锁住, 如果数据很多的话, 会导致CPU飙升,引起核心服务卡主 如(支付), 所有请求流量都堆积到了DB中, 造成雪崩效应, 引起数据库宕机

例子: 当QPS为20000w的时候使用keys \*操作:

1. redis是单线程的, 用keys \*命令查询所有的key, 导致redis被锁住
2. 此时QPS非常高, 在keys \* 检索的时候又来了很多个请求, 因为redis被锁住, 所以

所有的请求都被hang住了

1. 因为很多请求被hang住了, CPU又飙升, 导致redis宕机
2. 因为redis宕机, 请求全部并发到数据库, 导致数据库宕机

使用高耗时的命令是很危险的, 会占用唯一一个线程的大量处理时间, 导致所有请求都被延迟

keys 命令的时间复杂度为O(n), 严禁在生产环境使用!

还有很多时间复杂度为O(n)的命令如:hgetall lrange smemebers zrange sinter等命令, 并非不

能使用, 需要指定N的值, 负责也可能出现缓存宕机!

1. redis怎么禁用一些命令?

redis.conf中的SECURITY这一项新增 rename-command 命令 ""

如: rename-command FLUSHALL ""

rename-command FLUSHDB ""

rename-command KEYS ""

rename-command CONFIG ""

对于FLUSHALL命令, 需要配置appendonly no, 否则服务无法运行

1. redis使用scan命令的好处?

在redis2.8以后增加了scan命令, 可以分批扫描redis记录, 会导致查询的时间很长, 但是不会导致redis阻塞, 影响服务器的使用

SCAN index [reg] [count] 迭代当前数据库的key

SSCAN key index [reg] [count] 迭代集合的元素

HSCAN key index [reg] [count] 迭代hash的键值对

ZSCAN key index [reg] [count] 迭代zset的值和分数

index 指针偏移量

count默认10 也就是默认只查询10条

1. ?

设