##### Redis 数据持久化

redis 是基于内存的，但是redis也会定时的将这些数据持久化到 dump.rdb文件中，然后也会读取rdb文件

在redis.conf中是可以配置dump.rdb（二进制文件）文件的，如果rdb文件换了 那么就等于redis 的库中数据换了 ，Redis启动时在 加载持久化文件 包括rdb 和 aof 文件

###### RDB

默认数据持久化方式 ： rdb

Rdb持久化方式的特点 ：每隔一段时间持久化一次 当宕机时会导致数据丢失 ， fork线程占用资源

**RDB主从复制集群是关不掉的，如果关掉触发全量复制会自动开启**

1. Redis使用fork函数复制一份当前进程（父进程）的副本（子进程）；

fork分为 save 同步 和 bgsave 异步  
（2）父进程继续接收并处理客户端发来的命令，而子进程开始将内存中的数据写入硬盘中的临时文件；  
（3）当子进程写入完所有数据后会用该临时文件替换旧的 RDB 文件，至此一次快照操作完成。

###### Aof

1. Aof持久化

根据每次操作命令日志将数据追加到aof文件中 ，虽然是日志的形式，但是aof 保存的是重复命令的最后一个命令 ，并不会记录过程，如果修改key的值，那么在下 一次重写aof文件的时候 就会修改这key对应的值

1. Aof持久化特点

宕机时不会导致数据丢失，如果数据丢失可以通过 redis-check-aof工具检查找 回但是由于是日志日志的形式持久化数据会导致aof文件过大、效率降低

(3)Redis 提供了 aof文件重写机制来解决了文件过大的问题

重写机制：对现有的文件压缩，以rdb的形式保存将内存减小

1. 配置配置触发重写时机：auto-aof-rewrite-min-size 64mb(默认，生成环境都是几个G) 当文件大小达到64兆，会自动触发重写
2. 配置如何重写：auto-aof-rewrite-percentge 100(默认) 增长率 当文件第一次到64mb时 ，会触发重写机制，文件只有32mb ，如果文件再增长32时 就再次重写 就会压缩一半到43 ，然后增长43后 又会再次重写

C、手动调用重写bgrewriteaof 命令

原理 ： 当执行aof重写命令时 ，由于另外还有其他；的命令在执行， 主线程会把这时的数据写到缓冲区2，当redis重写并压缩完原来的aof（二进制）文件后 ， 然后把缓冲区的执行命令的日志数据加载到新的aof中 从而保证的数据的一致性

缺点： 兼容性差 ，只支持4.0之后的版本使用 ，4.0之前不认识aof文件

##### Redis数据备份方式

Redis 支持数据的备份，即 master-slave 模式的数据备份，且异步 只要主节点备份完毕 即默认成功（这样可能导致数据不一致问题 ：master节点处理数据完毕 异步还未同步至 从节点，master节点挂了 此时数据就不存在 ）

##### 不同类型的不同使用场景

###### String

1. 分布式锁

service 执行时间 大于 锁的过期时间 可以使用redisssion 进行锁续命。

锁续命原理 ： 在执行业务时新开一个线程定时查看，如果未执行完重新设置过 期时间redission过期时间默认30s ， 锁续命查看时间 1/3

新开一个线程 每隔30s检查一次，如果执行时间大于 过期时间，就重新设置过期时间

1. 计数器

###### Hash ： map 嵌套

###### List ：有序可重复的双向list

Lpush 、Lpop 从左边添加 或者 删除

Rpush、Rpop 从右边添加删除

Blopo ： 从左边 拿出一个元素 ， 没有就等待

Bropo ： 从右边 拿出一个元素 ， 没有就等待

带阻塞的 可以做监听的功能， 有就拿 没有就等待

Lrange 开始位置 结束位置 展示前几条数据

1. 实现 栈的功能 （后进先出）
2. 实现 队列功能 （先进先出） 生产消费
3. 实现延迟队列 消费队列，没有就阻塞等待

如果客户端连接线程一直阻塞，时间一长 为了减少资源暂用服务器会自动断开连接 ，这时 blpop、brpop 会抛异常 ，所以应注意异常的捕获，一单出现异常需进行连接从试

###### SET ： 无序不可重复

1. 抽奖功能

参与抽奖的人 存至集合 sadd key userId

查看抽奖人数 smemers key

随机送集合中抽取 若干名中奖用户 ：

* + 1. srandmemery key count(抽几个人) 人数不从集合中删除
    2. Spop key(抽几个人) 人数从集合中删除

1. 点赞功能

点赞 Sadd key value(userId)

取消点赞 srem key value(userId)

检查用户是否点赞 sismember key value(userId)

获取点赞列表 ： 查询 set 集合

获取点赞用户数 ： scard key

###### Zset : key 、value （value中 key、value）

1. 求交集、并集、差集 功能

##### 一个String 默认最大存储大小 512M

##### redis 常见性能解决方案

什么是内存快照 ？ 内存快照 即 持久化数据的过程

(1)、Master 最好不要写内存快照 (rdb持久化)，如果 Master 写内存快照，save 命令调度 rdbSave

函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性

暂停服务

(2)、主节点也不要开启aof 。如果数据比较重要，某个 Slave 开启 AOF 备份数据，策略设置为每 秒同步一次

(3)、为了主从复制的速度和连接的稳定性，Master 和 Slave 最好在同一个局域网

(4)、主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1

<- Slave2 <- Slave3…这样的结构方便解决单点故障问题，实现 Slave 对 Master

的替换。如果 Master 挂了，可以立刻启用 Slave1 做 Master，其他不变。

生产环境master节点建议不要开启持久化功能 ， 如果触发全量复制 master节点会自动触发rdb持久化，然后进行全量复制

##### redis过期建删除策略设置

(1)、定时删除:在设置键的过期时间的同时，创建一个定时器 timer(). 让定时器在键

的过期时间来临时，立即执行对键的删除操作。

(2)、惰性删除:放任键过期不管，但是每次从键空间中获取键时，都检查取得的键是

否过期，如果过期的话，就删除该键;如果没有过期，就返回该键。

(3)、当内存超过maxMemory时 ，触发主动清理策略

##### redis键回收策略 ： 当内存达到一定的程度触发从内存中淘汰无用的key 提高性能

4种大的策略

Nocviction （默认）

lru 最近最少使用的key

Random 随机

ttl key 剩余时间

**设置key 过期时间**

1. volatile-lru：从设置过期时间的数据集中挑选出最近最少使用的数据淘汰
2. volatile-ttl：从设置过期时间的数据集中挑选将要过期的数据淘汰，ttl值越大越优先被淘汰

(3)volatile-random：从设置过期时间的数据中任意挑选一个key删除，只有在内存达到限制无法写入未设置过期时间的数据时

**未设置key 过期时间(所有的key 包括有过期时间的)**

(1)allkeys-lru：从数据集中挑选最近最少使用的数据淘汰，该策略要淘汰的key面向的是全体key集合，而非过期的key集合

(2) allkeys-random：从数据集(server.db[i].dict）中选择任意数据淘汰

(3)no-enviction：禁止驱逐数据，也就是当内存不足以容纳新入数据时，新写入操作就会报错，请求可以继续进行，线上任务也不能持续进行采用no-enviction策略可以保证数据不被丢失，这也是系统默认的一种淘汰策略。

##### Redis 同步机制 （全量复制、增量复制）

1. 全量复制 场景 ：当redis使用一段时间后产生了数据（作为从机时 会清空本机的数据） ，此时这台redis需要作为从机 连接主机 ，当第一次连接时 会触发全量复制

全量复制的流程 ：

1. slaver向master发送命令 psync ？-1
   * 1. ？连接主机的redis的runID（进程标识）（主机的id从机并不知道所以传问号）
     2. -1 告诉master 全量复制
2. Master接收到slaver的连接命令后，会将本机的redis的runID 和 offset（偏移量：） 传给 slaver 告诉从机 让从机机备份他的数据

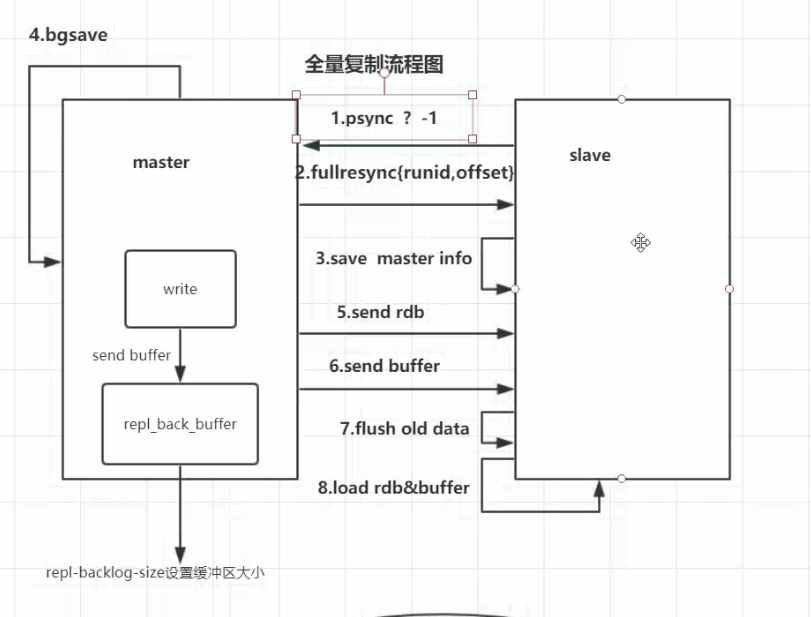
偏移量： 记录各节点的数据是否一致， 偏移量不一致说明数据不一致 （偏移量是实时变化的）

1. 然后从机保存master传来的数据
2. 然后主机在本机进行执行bgsave命令进行RDB持久化 （主机的配置文件中RDB模式是关闭，但 是全量复制也会触发）
3. 在主机本身进行rdb持久化的时候 ，此时也会有新的命令写入 ，会将新的命令写入缓冲 区 （默认只有1m ，当1m满时遵循先进先出的原则 将key进行持久化 并删除key）
4. 1m的配置 ：repl-backlog-size 1mb
5. 将4 缓冲区的文件 发送给 slaver服务器
6. 将4 持久化的rdb文件 ，发送给 slaver服务器
7. 清空 本机上内存中的数据
8. 加载并持久化 master 发来的数据

全量复制的缺点：

1. 执行bgsave 命令
2. 建立网络连接传输文件
3. 从节点加载rdb 写数据
4. 从机可能会造成aof重写 ，由于从机 rdb 和 aof 都是开启的 ，aof的优先级高于rdb ，aof文本重写 造成了 主、从机 持久化文件的不一致

因此性能会降低 ！



当 主机和从机由于网络问题 断开连接，当再次连接时 从机向主机汇报偏移量 发现偏移量不一致时 会触发增量复制

1. 先判断1m的缓冲区是否由于内存满了 ，删除过数据 如果没删除 则会将现在1m缓冲区中的数据 发送数据给从机进行同步 进行增量复制
2. 如果删除过数据 ， 则会进行全量复制

从这里可以看出 1m的缓冲区 跟多是为从机的正能量复制而准备的

主从复制的缺点：

1. 过于依赖master ， 所有写的操作都先经过 master ，然后发送给slaver 因此会有延迟
2. 一旦master down机，salver需要手动执行 从机中的某一台作为新的 master
3. Slaverof no one ：脱离 已经down 的主机 称为独立的机器
4. SLAVEROF I地址 端口 ：然后给另一台从机设置主机（已经脱离的主机）

##### pipeLine 批量执行命令使用

多个命令批量一次执行，提高效率 前提： 批量执行的命令没有因果关系性

##### scan 命令使用

由于redis单线程，在生产环境使用keys 命令查询键 会导致性能下降 可以使用scan 高效的分页显 示数据，并 且是非阻塞的

##### 11、redis 集群主从复制模型是怎样的

为了使在部分节点失败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模型, 每个节点都会有N-1个复制品.

##### Redis集群存在数据丢失吗

节点之间数据的同步是异步的， 因此redis并不能保证数据的强一致性 ，在主节点宕机 ，部分数据还未同步到从节点时 会出现数据丢失的情况

##### 在使用redis时如何尽量使用更少的内存

可以好好利用 Hash,list,sorted set,set

等集合类型数据，因为通常情况下很多小的 Key-Value 可

在存储复杂数据时 ，如java对象 尽量使用 散列表 hash ，而不是将对象转成字符 串 作为value存至value

##### Redis内存用完了 会发生什么

答：默认如果达到设置的上限，Redis 的写命令会返回错误信息（但是读命令还可以正

常返回。）也可以设置内存回收策略

##### Redis如何存热点数据

1. 通过ttl

当每次存放数据时，设置一个过期时间 如10分钟， 当第二次存放数据时 通过ttl 命令检查剩余过期时间 然后再次设置 剩余时间 + 10 分钟 ， 这样经常使用的数据的过期时间就会比较长 一直保存在redis中

##### Redis 可实现的复杂功能

A、队列

B、栈

C、阻塞队列（发布、订阅） ： list中有值就拿 而且可以一次生产多次消费 。

D、延时队列(Zset)

E、监听 ： key 过期自动触发监听

##### 分布式加锁 ， 如果锁冲突（key已存在） 该如何处理

1. 直接抛异常 ，稍后再试
2. 添加至延迟队列
3. Sleep一会重试

##### redis 缓存雪崩

(1)、什么是缓存雪崩 ： redis存的大量数据后 ， 大部分数据失效

**什么原因导致的**

1 ) 往redis中存放数据 往往会设置 过期时间， 当存在大量的数据在 某一时刻 同时 失效，就会造成缓存雪崩的情况

1. 如果redis发生故障 挂掉了 ，导致数据不能正常也是 也属于缓存雪崩

**怎么避免**

* + 1. 同一份多条数据设置过期时间时 ， 要错开设置 (避免过期时间导致的雪崩)
    2. 搭高可用 集群 (避免redis 挂掉导致的雪崩)

单机 ---> 主从复制--->哨兵(主从的升级，将主从切换自动化)--->集群

1. Redis 缓存穿透

当查询数据时 先判断缓存是否存在 ，如果不存在查询mysql

但是恶意的请求 ，会直接查一个id不存在的数据 如id = -1，导致mysql

压力过大

**解决方案** ： 1） 、布隆过滤器

Reddission 、 springboot的redisTemplte整合了布隆过滤器

布隆过滤器

1. 原理 ： 是一个很长的二进制数组保存0 ， 然后存放数据时 会经过几次 hash 运算 ，每次的 hash运算计算的结果都会得出一个 数组下标 然后将对应的下表上0 改为 1 。

当查询值时 会再经历同样的hash运算 计算出下标 然后获取 对应下标上的值是否为1 ，如果都为1 则表示数据存在。

1. 问题 ： 有误算的几率 。 在使用布隆过滤器时可以动态改变 误算的几率 。 这个几率直接关系到 二进制数组的的长度 以及 hash的运算次数 。 因此 几率越小 性能越底 。可以根据项目合理的给一个 误算率。
2. 、将空的数据根据查询的条件作为key缓存一份空的数据到redis中 并指定过期时间，如果有恶意请求，那么每次查询都会到redis中查
3. Redis 缓存击穿

热点key同一时间访问的次数很高 ， 当这个key的过期时间结束的瞬间，这些大 量的请求一瞬间访问mysql ，导致mysql压力过大

***解决方案：***

1. . 使用互斥锁 。 redis如果是单机 可以使用 sync、lock实现 ，集群可以使 用分布式锁 ， 如果对数据一致性要求较高 ， 要使用分布式锁 。 因为本地锁 也会 存在 缓存一致性的问题

18、怎么理解 Redis 事务？

事务是一个单独的隔离操作： 事务中的所有命令都会序列化、按顺序地执行。事务 在执行的过程中， 不会被其他客户端发送来的命令请求所打断。

事务是一个原子操作： 事务中的命令要么全部被执行， 要么全部都不执行。

19、Redis 事务相关的命令有哪几个？

答： MULTI、EXEC、DISCARD、WATCH

20、Redis 集群如何选择数据库？

答： Redis 集群目前无法做数据库选择， 默认在 0 数据库。

1. 如何解决 缓存不一致的问题

Mysql 中的缓存数据在系统中可能存在多个副本 ， 由于网络问题 导致数据缓存更 新失败

1. . 根据 业务 将缓存时间 调整的尽可能短点 ， 让缓存数据及时过期
2. . 如果是 要求数据强一致性的业务 ， 在每次更新缓存失败后 ， **可以进行重试** 重试失败， 加个重试失败的标记 ， 当这个标记存在的情况下 每次取数据时都 先清除缓存 再从数据库取。
3. 如何解决 数据并发竞争 （ 既保证 数据安全 、 又要保证性能 ）

安全 ：

1. .锁 看数据的影响范围是 单个服务还是 多个服务

A . 单个 服务 使用 lock 或者 synchronize

1. 多个服务 分布式锁

性能 ：

1. . 对缓存进行多处备份

在从缓存拿数据时 可以提供性能

1. 操作字符换设置了 过期时间 ， 如果数据更新 ， 时间也会重置

List 、hash 就不会

1. 如何应对 热点key 和 big key 可能带来的问题
2. . 热点 key
3. 对热点key进行分散备份 ， 当请求访问时 随机取一个缓存
4. 也可以对缓存进行 服务本都缓存不使用redis 。或者本地缓存 + redis的方式
5. .big key value较大的key

根据 业务在存储这些key时 ，尽可能使用拆分

1. 如根据根据业务id 分别拆分然后去存

存的时候 尽量不使用String ， 而是 list hash 能更节省内存

1. Zk 实现分布式锁 (**必须使用 临时有序节点** 来实现 锁)
2. 当客户端连接zk时都创建临时有序节点， 并且对前一个节点进行 监听
3. 序号最小的 临时节点 才能获取到锁 。
4. 释放锁时 ， 序号最小的处理完业务后断开zk 连接 ， 临时节点删除 此时 后一个节点会 监听到节点的 变化 ， 然后会判
   1. 先断是否是最小的节点 是的就上锁
   2. 不是 则继续重新监听前一个节点

死锁问题:临时节点不会出现死锁 ，即使网络出现问题 客户端会断开然后释放锁

重入问题 ： 可以将当前的 thread信息 或者标识 存至 临时节点 ， 在获取锁 时先判断 线程信息 ，如果相等 就进行重入

1. Redis 最大内存满了怎么办 ？
2. redis 有自己的 内存拒绝策略 ， 并且在启动时可以设置最大maxmemory内存
3. Redis 内存满了，如果主机内存未达到上线 可以通过 config set maxmemory 设置
4. 也可以 增加节点
5. 如果不想增加节点 ， 可以更改默认的 内存淘汰策略

也可以再不停机的情况下 ，使用命令 更改 内存淘汰策略

1. 分布式 缓存一致性 解决方案
2. . 并发不大 ， 常规解决方案

修改数据库前， 先删缓存 然后修改数据

当有查询操作时 ， 再 查询缓存 如果没有就同步缓存

(不要 修改数据后，立马同步缓存 ， 因为 1分钟修改 100次数据 ， 可能只 有1次查询 操作 。 99 次同步缓存都是无用的)

1. . 数据一致性 要求较高 ， 上面的方式 可能会出现脏数据

A 操作修改，还没修改完 B操作查询 ， 直接查询旧的值 存到缓存。 然后 A修改成功 ， 导致数据不一致

1. . 锁 (分布式锁 、cas 操作)

缺点 颗粒度大 ， 不同业务的所有操作都会阻塞

1. 将同一个业务的 查询 + 更新操作 存放至一个队列中
2. . A 操作数据 ------> 根据业务id 存到一个队列中

然后进行出队操作 ，删除缓存 更行数据库

1. . B 查询数据 ---> 根据业务id 存到 一个队列中

然后出队操作 ， 查询数据库中的数据 然后 存放到缓存中

注意的点 ：

1. 一个 队列 对应一个 线程 ， 如果是多个线程可能出现并行的情况
2. 同一个业务如果查询比较少 ，操作数据比较多 ，队列中 多个数据库操作串行 在一起是没有意义的 反而性能较慢

因此如果同一个业务 更新操作，已经在队列中了 或者 这个业务没有缓存 ， 说明可以放心操作数据 ，不用担心脏数据的问题

就没必要再入队 ， 可以直接处理 后的更新操作

1. 使用 mq 发送有序的 消息来实现
2. .