链滴

个人整理 - Java 后端面试题 - 缓存篇

作者：valarchie
原文链接：https://ld246.com/article/1584005501825
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

```
<h2 id="-redis的主从复制怎么做的-">★redis 的主从复制怎么做的？</h2>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">Redis主从复制可以根据是否是全量分为全量同步和增量同步。以下对其相应的同步过程及原理
下简要说明。
</span></span><span class="highlight-line"><span class="highlight-cl">增量同步
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">Redis增量同步主
指Slave完成初始化后开始正常工作时，Master发生的写操作同步到Slave的过程。通常情况下，
</span></span><span class="highlight-line"><span class="highlight-cl">Master每执行一
写命令就会向Slave发送相同的写命令，然后Slave接收并执行。
</span></span><span class="highlight-line"><span class="highlight-cl">全量同步
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">Redis的全量同步
程主要分三个阶段：
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">同步快照阶段：Ma
ter创建并发送快照给Slave,Slave载入并解析快照。Master同时将此阶段所产生的新的写命令存储到
冲区。
</span></span><span class="highlight-line"><span class="highlight-cl">同步写缓冲阶段：
aster向Slave同步存储在缓冲区的写操作命令。
</span></span><span class="highlight-line"><span class="highlight-cl">同步增量阶段：Ma
ter向Slave同步写操作命令。
</span></span></code></pre>
<h2 id="-redis为什么读写速率快性能好-">★redis 为什么读写速率快性能好？</h2>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">纯内存操作。
</span></span><span class="highlight-line"><span class="highlight-cl">核心是基于非阻塞
 IO 多路复用机制。
</span></span><span class="highlight-line"><span class="highlight-cl">C 语言实现，一般
说，C 语言实现的程序“距离”操作系统更近，执行速度相对会更快。
</span></span><span class="highlight-line"><span class="highlight-cl">单线程反而避免了
线程的频繁上下文切换问题，预防了多线程可能产生的竞争问题。
</span></span></code></pre>
<h2 id="redis为什么是单线程的-">redis 为什么是单线程的？</h2>
<ul>
<li>我们要去理解为什么多线程好，多线程好在 IO 慢的时候才能凸显出高性能，因为在一个线程等待
O 的时候马上切换线程运行。</li>
<li>而在 Redis 中数据都在内存当中，IO 非常快，如果还是采用多线程的话，反而会浪费时间在线
的上下文切换。</li>
</ul>
<h2 id="缓存的优点-">缓存的优点？</h2>
<ol>
<li>减少了对数据库的读操作，数据库的压力降低</li>
<li>加快了响应速度</li>
</ol>
<h2 id="-aof与rdb的优点和区别-">★aof 与 rdb 的优点和区别？</h2>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">持久化RDB（redis database）和AOF(append only file)的区别
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">RDB持久化是指在
定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是fork一个子进程，先将数据集写入
时文件，
</span></span><span class="highlight-line"><span class="highlight-cl">写入成功后，再替
之前的文件，用二进制压缩存储。 RDB持久化保存键空间的所有键值对(包括过期字典中的数据),并以
```

进制
形式保存，符合rd
文件规范，根据不同数据类型会有不同处理。

AOF持久化以日志
形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文
看到详细
的操作记录。AOF
久化保存redis服务器所执行的所有写命令来记录数据库状态,在写入之前命令存储在aof_buf缓冲区。

持久化RDB和AOF
优缺点:

RDB存在哪些优势
?
(1) 一旦采用该方
，那么你的整个Redis数据库将只包含一个文件，这对于文件备份而言是非常完美的。比如，你可能
算每个小时
归档一次最近24小
的数据，同时还要每天归档一次最近30天的数据。通过这样的备份策略，一旦系统出现灾难性故障，
们可以非常
容易的进行恢复。
(2) 对于灾难恢复
言，RDB是非常不错的选择。因为我们可以非常轻松的将一个单独的文件压缩后再转移到其它存储介
上。
(3) 性能最大化。
于Redis的服务进程而言，在开始持久化时，它唯一需要做的只是fork出子进程，之后再由子进程完
这些持久化
的工作，这样就可
极大的避免服务进程执行IO操作了。
(4) 相比于AOF机
，如果数据集很大，RDB的启动效率会更高。

RDB又存在哪些劣
呢?

(1) 如果你想保证
据的高可用性，即最大限度的避免数据丢失，那么RDB将不是一个很好的选择。因为系统一旦在定时
久化之前出现
宕机现象，此前没
来得及写入磁盘的数据都将丢失。
(2). 由于RDB是通过
ork子进程来协助完成数据持久化工作的，因此，如果当数据集较大时，可能会导致整个服务器停止
务几百毫秒，
甚至是1秒钟。

AOF的优势有哪些
?
(1) 该机制可以带
更高的数据安全性，即数据持久性。Redis中提供了3种同步策略，即每秒同步、每个命令同步和不同
。事实上，
每秒同步也是异步
成的，其效率也是非常高的，所差的是一旦系统出现宕机现象，那么这一秒钟之内修改的数据将会丢
。而每个命令

同步，我们可以将视为同步持久化，即每次发生的数据变化都会被立即记录到磁盘中。可以预见，这种方式在效率上是低的。至于无
同步，由操作系统定任何将缓冲区里面的命令写入磁盘里面,在这种模式写,服务器遭遇意外停机时,丢失命令的数据是不定的
(2) 由于该机制对志文件的写入操作采用的是append模式，因此在写入过程中即使出现宕机现象，也不会破坏日志文中已经存在的
内容。然而如果我本次操作只是写入了一半数据就出现了系统崩溃问题，不用担心，在Redis下一次启动之前，我们可通过redis-
check-aof工具来助我们解决数据一致性的问题。
(3) 如果日志过大，edis可以自动启用rewrite机制。主要是为了解决aof文件不断增长减少重复键读写操作的日志, 通过定触发,
重新根据实际内存值情况写入新的aof文件, 再进新旧替换文件,实现aof文件最小化。
(4) AOF包含一个式清晰、易于理解的日志文件用于记录所有的修改操作。事实上，我们也可以通过该文件完成数据的建。

AOF的劣势有哪些?
(1) 对于相同数量数据集而言，AOF文件通常要大于RDB文件。RDB 在恢复大数据集时的速度比 AOF 的恢复速度要快

(2) 根据同步策略不同，AOF在运行效率上往往会慢于RDB。总之，每秒同步策略的效率是比较高的，同步禁用策略的率和RDB
一样高效。

二者选择的标准，是看系统是愿意牺牲一些性能，换取更高的缓存一致性（aof），还是愿意写操作频繁的时候，不启备份来
换取更高的性能，手动运行save的时候，再做备份（rdb）。rdb这个就更有些 eventually consistent的意思了。不过产环境
其实更多都是二者合使用的。
</code></pre>
<h2 id="redis的List能用做什么场景-">redis 的 List 能用做什么场景？</h2>
1.消息队列。2.排行榜。3.最新列表。
<h2 id="用java自己实现一个LRU-">用 java 自己实现一个 LRU。</h2>
继承 LinkHashMap 并重写 removeEldestEntry 方法。
<h2 id="-为什么用缓存-用过哪些缓存-redis和memcache的区别">★ 为什么用缓存，用过哪些缓，redis 和 memcache 的区别</h2>
使用缓存,最主要的目的是提升系统的响应速度和对高负载的承受能力。
redis，memcached 都有用过。

数据操作不同

与 Memcached 仅支持简单的 key-value 结构的数据记录不同，Redis 支持的数据类型要丰富多。Memcached 基本只支持简单的 key-value 存储，不支持枚举，不支持持久化和复制等功能。Re is 支持服务器端的数据操作相比 Memcached 来说，拥有更多的数据结构和并支持更丰富的数据操，支持 list、set、sorted set、hash 等众多数据结构，还同时提供了持久化和复制等功能。
而通常在 Memcached 里，使用者需要将数据拿到客户端来进行类似的修改再 set 回去，这大增加了网络 IO 的次数和数据体积。在 Redis 中，这些复杂的操作通常和一般的 GET/SET 一样高效所以，如果需要缓存能够支持更复杂的结构和操作， Redis 会是更好的选择。

<ol start="2">
内存管理机制不同

在 Redis 中，并不是所有的数据都一直存储在内存中的。这是和 Memcached 相比一个最大的别。当物理内存用完时，Redis 可以将一些很久没用到的 value 交换到磁盘。Redis 只会缓存所有的 k y 的信息，如果 Redis 发现内存的使用量超过了某一个阀值，将触发 swap 的操作，Redis 根据“swa pability = age*log(size_in_memory)”计算出哪些 key 对应的 value 需要 swap 到磁盘。然后再将些 key 对应的 value 持久化到磁盘中，同时在内存中清除。这种特性使得 Redis 可以保持超过其机本身内存大小的数据。
而 Memcached 默认使用 Slab Allocation 机制管理内存，其主要思想是按照预先规定的大小，分配的内存分割成特定长度的块以存储相应长度的 key-value 数据记录，以完全解决内存碎片问题。
从内存利用率来讲，使用简单的 key-value 存储的话，Memcached 的内存利用率更高。而如果edis 采用 hash 结构来做 key-value 存储，由于其组合式的压缩，其内存利用率会高于 Memcache。

<ol start="3">
性能不同

由于 Redis 只使用单核，而 Memcached 可以使用多核，所以平均每一个核上 Redis 在存储小据时比 Memcached 性能更高。而在 100k 以上的数据中，Memcached 性能要高于 Redis，虽然 R dis 也在存储大数据的性能上进行了优化，但是比起 Memcached，还是稍有逊色。

<ol start="4">
集群管理不同

Memcached 是全内存的数据缓冲系统，Redis 虽然支持数据的持久化，但是全内存毕竟才是其性能的本质。作为基于内存的存储系统来说，机器物理内存的大小就是系统能够容纳的最大数据量。果需要处理的数据量超过了单台机器的物理内存大小，就需要构建分布式集群来扩展存储能力。
Memcached 本身并不支持分布式，因此只能在客户端通过像一致性哈希这样的分布式算法来实 Memcached 的分布式存储。相较于 Memcached 只能采用客户端实现分布式存储，Redis 更偏向在服务器端构建分布式存储。

<h2 id="redis的持久化方式-以及项目中用的哪种-为什么">redis 的持久化方式，以及项目中用的哪，为什么</h2>

RDB 持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是 fork 一子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。

AOF 持久化以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文的方式记录，可以打开文件看到详细的操作记录。

二者优缺点

RDB 存在哪些优势呢？

1). 一旦采用该方式，那么你的整个 Redis 数据库将只包含一个文件，这对于文件备份而言是非完美的。比如，你可能打算每个小时归档一次最近 24 小时的数据，同时还要每天归档一次最近 30 的数据。通过这样的备份策略，一旦系统出现灾难性故障，我们可以非常容易的进行恢复。

2). 对于灾难恢复而言，RDB 是非常不错的选择。因为我们可以非常轻松的将一个单独的文件压后再转移到其它存储介质上。

3). 性能最大化。对于 Redis 的服务进程而言，在开始持久化时，它唯一需要做的只是 fork 出子程，之后再由子进程完成这些持久化的工作，这样就可以极大的避免服务进程执行 IO 操作了。

4). 相比于 AOF 机制，如果数据集很大，RDB 的启动效率会更高。

RDB 又存在哪些劣势呢？

1). 如果你想保证数据的高可用性，即最大限度的避免数据丢失，那么 RDB 将不是一个很好的选。因为系统一旦在定时持久化之前出现宕机现象，此前没有来得及写入磁盘的数据都将丢失。

2). 由于 RDB 是通过 fork 子进程来协助完成数据持久化工作的，因此，如果当数据集较大时，能会导致整个服务器停止服务几百毫秒，甚至是 1 秒钟。

AOF 的优势有哪些呢？

1). 该机制可以带来更高的数据安全性，即数据持久性。Redis 中提供了 3 中同步策略，即每秒步、每修改同步和不同步。事实上，每秒同步也是异步完成的，其效率也是非常高的，所差的是一旦统出现宕机现象，那么这一秒钟之内修改的数据将会丢失。而每修改同步，我们可以将其视为同步持化，即每次发生的数据变化都会被立即记录到磁盘中。可以预见，这种方式在效率上是最低的。至于同步，无需多言，我想大家都能正确的理解它。

2). 由于该机制对日志文件的写入操作采用的是 append 模式，因此在写入过程中即使出现宕机象，也不会破坏日志文件中已经存在的内容。然而如果我们本次操作只是写入了一半数据就出现了系崩溃问题，不用担心，在 Redis 下一次启动之前，我们可以通过 redis-check-aof 工具来帮助我们解数据一致性的问题。

3). 如果日志过大，Redis 可以自动启用 rewrite 机制。即 Redis 以 append 模式不断的将修改据写入到老的磁盘文件中，同时 Redis 还会创建一个新的文件用于记录此期间有哪些修改命令被执行因此在进行 rewrite 切换时可以更好的保证数据安全性。

4). AOF 包含一个格式清晰、易于理解的日志文件用于记录所有的修改操作。事实上，我们也可通过该文件完成数据的重建。

AOF 的劣势有哪些呢？

1). 对于相同数量的数据集而言，AOF 文件通常要大于 RDB 文件。RDB 在恢复大数据集时的速比 AOF 的恢复速度要快。

2). 根据同步策略的不同，AOF 在运行效率上往往会慢于 RDB。总之，每秒同步策略的效率是比高的，同步禁用策略的效率和 RDB 一样高效。

二者选择的标准，就是看系统是愿意牺牲一些性能，换取更高的缓存一致性（aof），还是愿意操作频繁的时候，不启用备份来换取更高的性能，待手动运行 save 的时候，再做备份（rdb）。rdb 个就更有些 eventually consistent 的意思了。

<h2 id="-redis集群的理解-怎么动态增加或者删除一个节点-而保证数据不丢失--一致性哈希问题-">redis 集群的理解，怎么动态增加或者删除一个节点，而保证数据不丢失。（一致性哈希问题）</h2>
先添加删除节点，再移动槽位 reshard，利用一致性哈希的原理保证移动槽位的时候不影响其他点。
<h2 id="-Redis的缓存失效策略">★Redis 的缓存失效策略</h2>
FIFO：First In First Out，先进先出。判断被存储的时间，离目前最远的数据优先被淘汰。
LRU：Least Recently Used，最近最少使用。判断最近被使用的时间，目前最远的数据优先被淘汰。br>
LFU：Least Frequently Used，最不经常使用。在一段时间内，数据被使用次数最少的，优先被淘汰
<h2 id="缓存穿透的解决办法">缓存穿透的解决办法</h2>
导致缓存穿透的原因：

恶意攻击，猜测你的 key 命名方式，然后估计使用一个你缓存中不会有的 key 进行访问。
第一次数据访问，这时缓存中还没有数据，则并发场景下，所有的请求都会压到数据库。
数据库的数据也是空，这样即使访问了数据库，也是获取不到数据，那么缓存中肯定也没有对应数据。这样也会导致穿透。
解决缓存穿透：
再 web 服务器启动时，提前将有可能被频繁并发访问的数据写入缓存。—这样就规避大量的请在第 3 步出现排队阻塞。
规范 key 的命名，并且统一缓存查询和写入的入口。这样，在入口处，对 key 的规范进行检测。这样保存恶意的 key 被拦截。
Synchronized 双重检测机制，这时我们就需要使用同步（Synchronized）机制，在同步代码块查询一下缓存是否存在对应的 key，然后同步代码块里面再次查询缓存里是否有要查询的 key。 这样双重检测”的目的，还是避免并发场景下导致的没有意义的数据库的访问（也是一种严格避免穿透的案）。这一步会导致排队，但是第一步中我们说过，为了避免大量的排队，可以提前将可以预知的大请求提前写入缓存。
不管数据库中是否有数据，都在缓存中保存对应的 key，值为空就行。–这样是为了避免数据库没有这个数据，导致的平凡穿透缓存对数据库进行访问。
第 4 步中的空值如果太多，也会导致内存耗尽。导致不必要的内存消耗。这样就要定期的清理空的 key。避免内存被恶意占满。导致正常的功能不能缓存数据。

<h2 id="-redis集群-高可用-原理">★redis 集群，高可用，原理</h2>
Redis的高可用详解：Redi
哨兵、复制、集群的设计原理，以及区别 - 疯子110 - 博客园
<h2 id="mySQL里有2000w数据-redis中只存20w的数据-如何保证redis中的数据都是热点数据">m
SQL 里有 2000w 数据，redis 中只存 20w 的数据，如何保证 redis 中的数据都是热点数据</h2>
即 redis 的缓存每命中一次,就给命中的缓存增加一定 ttl(过期时间)(根据具体情况来设定, 比如 10
分钟).
一段时间后, 热数据的 ttl 都会较大, 不会自动失效, 而冷数据基本上过了设定的 ttl 就马上失效了.
<h2 id="用Redis和任意语言实现一段恶意登录保护的代码-限制1小时内每用户Id最多只能登录5次"
用 Redis 和任意语言实现一段恶意登录保护的代码，限制 1 小时内每用户 Id 最多只能登录 5 次</h2

使用 userId 为 key
ttl 为一小时的数字值表示登录次数
超过 5 次即限制
<h2 id="-如何做到缓存数据一致性-">★ 如何做到缓存数据一致性。</h2>
设置过期时间 加 双删
<h2 id="如何防止缓存穿透-雪崩-击穿-">如何防止缓存穿透、雪崩、击穿。</h2>
一、缓存穿透
概念：缓存穿透是指缓存和数据库中都没有的数据，而用户不断发起请求，如发起为 id 为“-1
的数据或 id 为特别大不存在的数据。这时的用户很可能是攻击者，攻击会导致数据库压力过大。</p

解决方案：

接口校验请求参数的有效性，如校验 id 是否合法
从缓存取不到的数据，在数据库中也没有取到，这时也可以将 key-value 对写为 key-null，缓存
效时间可以设置短点，如 30 秒。

二、缓存雪崩
概念：指缓存中数据大批量到过期时间，而查询数据量巨大，引起数据库压力过大甚至 down
。侧重很多的缓存数据都失效了。
解决方案：

设置不同的缓存到期时间，如热门数据有效期长一点，冷门数据有效期短一点。
针对长期有效，数据变化基本不变的数据进行永久性缓存。
缓存服务集群部署

三、缓存击穿
概念：
指一个 key 非常热点，在不停的扛着大并发，大并发集中对这一个点进行访问，当这个 key 在
效的瞬间，持续的大并发就穿破缓存，直接请求数据库，就像在一个屏障上凿开了一个洞。
解决方案：

热点数据实现缓存永久有效
采用互斥锁来缓存数据
缓存击穿的重点就是，同一时间点上，大量请求没有在缓存中命中数据，而对数据库服务产生巨
的压力，所以我们可以在请求数据库之前加上一个锁，持有锁的线程才能去请求数据库，并缓存到缓
数据库，其他的线程等待，最后从缓存获获取数据。

<h2 id="redis的list结构相关的操作-">redis 的 list 结构相关的操作。</h2>
list 是双向链表。列表常用来作为异步队列使用
通过使用 rpush、rpop、lpush、lpop 四条指令，在链表的表头和表尾追加或移除元素，可以将链
作为队列或堆栈使用；
<h2 id="-redis2和redis3的区别-redis3内部通讯机制-">★redis2 和 redis3 的区别，redis3 内部通

机制。</h2>

哨兵的作用就是监控 Redis 系统的运行状况。它的功能包括以下两个。
监控主数据库和从数据库是否正常运行。
主数据库出现故障时自动将从数据库转换为主数据库。
sentinel 发现 master 挂了后，就会从 slave 中重新选举一个 master。
哨兵模式强调高可用
Sentinel 系统用于管理多个 Redis 服务器（instance）， 该系统执行以下三个任务：
监控（Monitoring）： Sentinel 会不断地检查你的主服务器和从服务器是否运作正常。
提醒（Notification）： 当被监控的某个 Redis 服务器出现问题时， Sentinel 可以通过 API 向
理员或者其他应用程序发送通知。
自动故障迁移（Automatic failover）： 当一个主服务器不能正常工作时， Sentinel 会开始一次自
故障迁移操作， 它会将失效主服务器的其中一个从服务器升级为新的主服务器， 并让失效主服务器
其他从服务器改为复制新的主服务器； 当客户端试图连接失效的主服务器时， 集群也会向客户端返
新主服务器的地址， 使得集群可以使用新主服务器代替失效服务器。
客户端中不会记录 redis 的地址（某个 IP），而是记录 sentinel 的地址，这样我们可以直接从 s
ntinel 获取的 redis 地址，因为 sentinel 会对所有的 master、slave 进行监控，它是知道到底谁才
真正的 master 的，例如我们故障转移，这时候对于 sentinel 来说，master 是变了的，然后通知客
端。而客户端根本不用关心到底谁才是真正的 master，只关心 sentinel 告知的 master。
集群
即使使用哨兵，redis 每个实例也是全量存储，每个 redis 存储的内容都是完整的数据，浪费内
且有木桶效应。为了最大化利用内存，可以采用集群，就是分布式存储。即每台 redis 存储不同的内
，共有 16384 个 slot。每个 redis 分得一些 slot，hash_slot = crc16(key) mod 16384 找到对应 slo
，键是可用键，如果有{}则取{}内的作为可用键，否则整个键是可用键
集群至少需要 3 主 3 从，且每个实例使用不同的配置文件，主从不用配置，集群会自己选。
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。
集群模式提高并发量。

<h2 id="redis和memcached-的内存管理的区别-">redis 和 memcached 的内存管理的区别。</h
>

在 Redis 中，并不是所有的数据都一直存储在内存中的。这是和 Memcached 相比一个最大的
别。当物理内存用完时，Redis 可以将一些很久没用到的 value 交换到磁盘。
Memcached 默认使用 Slab Allocation 机制管理内存，其主要思想是按照预先规定的大小，将
配的内存分割成特定长度的块以存储相应长度的 key-value 数据记录，以完全解决内存碎片问题。</l
>

<h2 id="-Redis的并发竞争问题如何解决-了解Redis事务的CAS操作吗-">★Redis 的并发竞争问题
何解决，了解 Redis 事务的 CAS 操作吗。</h2>

使用 incr
使用 watch
分布式锁

<h2 id="Redis的选举算法和流程是怎样的-">Redis 的选举算法和流程是怎样的。</h2>
<pre><code class="highlight-chroma"><span class="highlight
cl">监视这个挂了的主节点的所有Sentinel都有被选举为领头的资格
每进行一次选举，
论是否成功，配置纪元+1，配置纪元就是个计数器
每个Sentinel在每
配置纪元中有且仅有一次选举机会，一旦选好了该节点认为的主节点，在这个纪元内，不可以再更改
每个发现服务器挂
的Sentinel都会配置纪元+1并投自己一票，接着发消息要求其他Sentinel设置自己为领头人1，每个S
ntinel都想成为领头的

每个Sentinel会将先发来请求领头的节点设为自己的领头节点并发送回复，谁先来我选谁
当源Sentinel收到复，并且回复中的配置纪元和自己的一致且领头Id是自己的Sentinel Id时，表明目标Sentinel已经将己设为领头
在一个配置纪元内当某个Sentinel收到半数以上的同意回复时，它就是领头的了
如果在给定时间内没有被成功选举的Sentinel，那么过段时间发起新的选举
选举领头Sentinel过程和规则大概就如上所述，需要注意的是只有集群出现节点挂了才需要选举出领头Sentinel，平时个Sentinel还是平等身份~
这里的选举其实是rat算法的一个应用.
</code></pre>
<h2 id="-redis的集群怎么同步的数据的-Reids的主从复制机制原理--">★redis 的集群怎么同步的据的(Reids 的主从复制机制原理)。</h2>

redis 的复制功能是支持多个数据库之间的数据同步。一类是主数据库（master）一类是从数据（slave），主数据库可以进行读写操作，当发生写操作的时候自动将数据同步到从数据库，而从数库一般是只读的，并接收主数据库同步过来的数据，一个主数据库可以有多个从数据库，而一个从数库只能有一个主数据库。
通过 redis 的复制功能可以很好的实现数据库的读写分离，提高服务器的负载能力。主数据库主进行写操作，而从数据库负责读操作。
当一个从数据库启动时，会向主数据库发送 sync 命令，
主数据库接收到 sync 命令后会开始在后台保存快照（执行 rdb 操作），并将保存期间接收到的令缓存起来
当快照完成后，redis 会将快照文件和所有缓存的命令发送给从数据库。
从数据库收到后，会载入快照文件并执行收到的缓存的命令。

<h2 id="redis优化详解">redis 优化详解</h2>
https://blog.csdn.net/qq_39399966/article/details/101211939
https://blog.csdn.net/achuo/article/details/80600170
<h2 id="-Redis的线程模型是什么-">★Redis 的线程模型是什么。</h2>

Redis 基于 Reactor 模式开发了网络事件处理器，这个处理器被称为文件事件处理器。它的组成构为 4 部分：多个套接字、IO 多路复用程序、文件事件分派器、事件处理器。因为文件事件分派器列的消费是单线程的，所以 Redis 才叫单线程模型。
尽管多个文件事件可能会并发地出现，但 I/O 多路复用程序总是会将所有产生事件的套接字都推一个队列里面，然后通过这个队列，以有序（sequentially）、同步（synchronously）、每次一个接字的方式向文件事件分派器传送套接字：当上一个套接字产生的事件被处理完毕之后（该套接字为件所关联的事件处理器执行完毕）， I/O 多路复用程序才会继续向文件事件分派器传送下一个套接字

<h2 id="集中式缓存和分布式缓存-">集中式缓存和分布式缓存？</h2>
集中式：容量，并发瓶颈，但一致性有保证。
分布式：提升容量，并发能力，但会有一致性问题。
<h2 id="elasticsearch-索引数据多了怎么办-如何调优-部署-">elasticsearch 索引数据多了怎么办如何调优，部署。</h2>
<h2 id="Elasticsearch在部署时-对Linux的设置有哪些优化方法-">Elasticsearch 在部署时，对 Linu

的设置有哪些优化方法？</h2>
<h2 id="elasticsearch是如何实现master选举的-">elasticsearch 是如何实现 master 选举的。</h
>
<h2 id="详细描述一下Elasticsearch搜索的过程-">详细描述一下 Elasticsearch 搜索的过程。</h2>
<h2 id="redis和memcache的区别">redis 和 memcache 的区别</h2>

redis 使用单线程模型，数据顺序提交，redis 支持主从模式，mencache 只支持一致性 hash 做布式；redis 支持数据落地，rdb 定时快照和 aof 实时记录操作命令的日志备份，memcache 不支持redis 数据类型丰富，有 string，hash，set，list， sort set，而 memcache 只支持简单数据类型；emcache 使用 cas 乐观锁做一致性。

jedis 操作 Hash：hmset, hmget, hdel, hkeys

jedis 操作 List： lpush，lrange 按照范围取出，rpush， del， sort 等 keyjedis 操作 Set：sad，srem 移除 noname，smembers， sismember， scard 等。summary.note

使用场景例如

Hash：存储读取更新用户多个属性

List：微博 TimeLine，消息列表

Set：共同好友，二度好友，用唯一性可以统计网站所有独立 IP，好友推荐根据 tag 求交集，大于threshold 就可以推荐。

sortset：set 增加 1 个权重 score 参数

其他场景：A 订阅发布系统，redis 对某个 key 消息发布及订阅，当 1 个 key 消息发布后，所订阅它的客户端都会收到相应消息，例如实时消息系统，即时聊天，群聊等。

事务-常用 EX,EC 提交执行的命令，在 server 不出问题，可以保证一连串的命令是顺序执行额；供 1 个 watch 功能，对 1 个 key 作 watch，然后再执行 transation。

<h2 id="redis过期淘汰策略">redis 过期淘汰策略</h2>

redis 内存数据上升到一定大小会执行数据淘汰策略，redis 提供了 6 种数据淘汰策略。

LRU：从已设置过期时间的数据集合中挑选最近最少使用的数据淘汰

random：从已设置过期时间的数据中挑选任意数据淘汰

ttl：从已设置过期时间的数据集合中挑选将要过期的数据淘汰。

notenvision：禁止驱逐数据

如 mysql 中有 2 千万数据，redis 只存储 20 万的热门数据。LRU 或者 TTL 都满足热点数据读较多，不太可能超时特点。

redis 特点：速度块，O(1)，丰富的数据类型，支持事物原子性，可用于缓存，比 memecache 度块，可以持久化数据。

常见问题和解决：Master 最好不做持久化如 RDB 快照和 AOF 日志文件；如果数据比较重要，分 slave 开启 AOF 备份数据，策略为每秒 1 次，为了主从复制速度及稳定，MS 主从在同一局域网；主从复制不要用图状结构，用单向链表更为稳定 M-S-S-S-S。。。。；redis 过期采用懒汉 + 定期懒汉即 get/set 时候检查 key 是否过期，过期则删除 key，定期遍历每个 DB，检查制定个数个 key 结合服务器性能调节并发情况。

过期淘汰，数据写入 redis 会附带 1 个有效时间，这个有效时间内该数据被认为是正确的并不关真实情况，例如对支付等业务采用版本号实现，redis 中每一份数据都维持 1 个版本号，DB 中也维持 份，只有当 redis 的与 DB 中的版本一致时，才会认为 redis 为有效的，不过仍然每次都要访问 DB 只需要查询 version 版本字段即可。

<h2 id="如何将数据分布在redis第几个库-">如何将数据分布在 redis 第几个库？</h2>
答：redis 本身支持 16 个数据库，通过 数据库 id 设置，默认为 0。
例如 jedis 客户端设置。一：JedisPool(org.apache.commons.pool.impl.GenericObjectPool.Confi poolConfig, String host, int port, int timeout, String password, int database);
第一种通过指定构造函数 database 字段选择库，不设置则默认 0 库。二：jedis.select(index);调用 j dis 的 select 方法指定。
<h2 id="Elasticsearch分片使用优化-">Elasticsearch 分片使用优化?</h2>
(1)拆分集群
对于存在明显分界线的业务，可以按照业务、地域使用不同集群，这种拆分集群的思路是非常靠的。对于我们的场景，已经按照地域拆分了集群，且同一地域的子业务间分界线不明显，拆分过多的群维护成本较高。
(2)调整滚动周期
根据保留时长调整 index 滚动周期是最简单有效的思路。例如保留 3 天的数据按天滚动，保留 31 天的数据按周滚动，保留一年的数据按月滚动。合理的滚动周期，可以在存储成本增加不大的情况下大幅降低分片数量。
对于我们的场景，大部分数据保留 31 天，在按周滚动的情况下，集群的总分片数可以下降到 6.5w~。
(3)合理设置分片数和副本数
除个别子业务压力较高外，大部分业务压力较小，合理设置单 Index 的分片数效果也不错。我们经验是单个分片的大小在 10GB<del>30GB 之间比较合适，对于压力非常小的业务可以直接分配 1 分片。其他用户可结合具体场景考虑，同时注意单分片的记录条数不要超过上限 2,147,483,519。
在平衡我们的业务场景对数据可靠性的要求 及 不同副本数对存储成本的开销 两个因素之后，我们选使用一主一从的副本策略。
目前我们集群单 Index 的平均分配数为 3，集群的总分片数下降到 3w</del>个。

(4)分片分配流程优化
默认情况下，ES 在分配分片时会考虑分片 relocation 对磁盘空间的影响。在分片数较少时，这优化处理的副作用不明显。但随着单机分片数量的上升，这个优化处理涉及的多层循环嵌套过程耗时发明显。可通过 cluster.routing.allocation.disk.include_relocations: false 关闭此功能，这对磁盘衡程度影响不明显。
(5)预创建 Index
对于单集群 3w 分片的场景，集中在每周某天 0 点创建 Index，对集群的压力还是较大，且存储间存在波动。考虑到集群的持续扩展能力和可靠性，我们采用预创建方式提前创建分片，并把按 Index 的创建时间均匀打散到每周的每一天。
(6)持续调整分片数
对于集群分片的调整，通常不是一蹴而就的。随着业务的发展，不断新增的子业务 或 原有子业规模发生突变，都需要持续调整分片数量。
默认情况下，新增的子业务会有默认的分片数量，如果不足，会在测试阶段及上线初期及时发现。随业务发展，系统会考虑 Index 近期的数据量、写入速度、集群规模等因素，动态调整分片数量。
<h2 id="ElasticSearch如何解决深度分页的问题-">ElasticSearch 如何解决深度分页的问题？</h2>
使用 scroll（有状态）和 search after（无状态）的游标方式。
<h2 id="lucence倒排索引">lucence 倒排索引</h2>

三个文件：字典文件，频率文件，位置文件。词典文件不仅保存有每个关键词，还保留了指向频文件和位置文件的指针，通过指针可以找到该关键字的频率信息和位置信息。

field 的概念，用于表达信息所在位置（如标题中，文章中，url 中），在建索引中，该 field 信也记录在词典文件中，每个关键词都有一个 field 信息(因为每个关键字一定属于一个或多个 field)。

关键字是按字符顺序排列的（lucene 没有使用 B 树结构），因此 lucene 可以用二元搜索算法速定位关键词。

假设要查询单词 “live”，lucene 先对词典二元查找、找到该词，通过指向频率文件的指针读所有文章号，然后返回结果。词典通常非常小，因而，整个过程的时间是毫秒级的。

对词典文件中的关键词进行了压缩，关键词压缩为 < 前缀长度，后缀 >，例如：当前词为阿拉伯语”，上一个词为“阿拉伯”，那么“阿拉伯语”压缩为 <3，语 >。对数字的压缩，数只保存与上一个值的差值。