

# 46 道史上最全 REDIS 面试题，面试官能问的都被我找到了（含答案）

## 1、什么是 REDIS？简述它的优缺点？

Redis 的全称是：Remote Dictionary Server，本质上是一个 Key-Value 类型的内存数据库，很像

memcached，整个数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据 flush 到硬盘上进行保存。

因为是纯内存操作，Redis 的性能非常出色，每秒可以处理超过 10 万次读写操作，是已知性能最快的 Key-Value DB。

Redis 的出色之处不仅仅是性能，Redis 最大的魅力是支持保存多种数据结构，此外单个 value 的最大限制是 1GB，不像 memcached 只能保存 1MB 的数据，因此 Redis 可以用来实现很多有用的功能。

比方说用他的 List 来做 FIFO 双向链表，实现一个轻量级的高性能消息队列服务，用他的 Set 可以做高性能的 tag 系统等等。

另外 Redis 也可以对存入的 Key-Value 设置 expire 时间，因此也可以被当作一个功能加强版的 memcached 来用。Redis 的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写，因此 Redis 适合的场景主要局限在较小数据量的高性能操作和运算上。

## 2、REDIS 与 MEMCACHED 相比有哪些优势？

1.memcached 所有的值均是简单的字符串，redis 作为其替代者，支持更为丰富的数据类型

2.redis 的速度比 memcached 快很多 redis 的速度比 memcached 快很多

3.redis 可以持久化其数据 redis 可以持久化其数据

## 3、REDIS 支持哪几种数据类型？

String、List、Set、Sorted Set、hashes

## 4、REDIS 主要消耗什么物理资源？

内存。

## 5、REDIS 有哪几种数据淘汰策略？

1.noeviction: 返回错误当内存限制达到，并且客户端尝试执行会让更多内存被使用的命令。

2.allkeys-lru: 尝试回收最少使用的键（LRU），使得新添加的数据有空间存放。

3.volatile-lru: 尝试回收最少使用的键（LRU），但仅限于在过期集合的键，使得新添加的数据有

空间存放。

4.allkeys-random: 回收随机的键使得新添加的数据有空间存放。

5.volatile-random: 回收随机的键使得新添加的数据有空间存放，但仅限于在过期集合的键。

6.volatile-ttl: 回收在过期集合的键，并且优先回收存活时间（TTL）较短的键，使得新添加的数据有空间存放。

## 6、REDIS 官方为什么不提供 WINDOWS 版本？

因为目前 Linux 版本已经相当稳定，而且用户量很大，无需开发 windows 版本，反而会带来兼容性问题。

## 7、一个字符串类型的值能存储最大容量是多少？

512M

## 8、为什么 REDIS 需要把所有数据放到内存中？

Redis 为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。

所以 redis 具有快速和数据持久化的特征，如果不将数据放在内存中，磁盘 I/O 速度为严重影响 redis 的性能。

在内存越来越便宜的今天，redis 将会越来越受欢迎，如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

## 9、REDIS 集群方案应该怎么做？都有哪些方案？

1.codis

2. 目前用的最多的集群方案，基本和 twemproxy 一致的效果，但它支持在节点数量改变情况下，旧节点数据可恢复到新 hash 节点。

redis cluster3.0 自带的集群，特点在于他的分布式算法不是一致性 hash，而是 hash 槽的概念，以及自身支持节点设置从节点。具体看官方文档介绍。

3. 在业务代码层实现，起几个毫无关联的 redis 实例，在代码层，对 key 进行 hash 计算，然后去对应的 redis 实例操作数据。这种方式对 hash 层代码要求比较高，考虑部分包括，节点失效后的替代算法方案，数据震荡后的自动脚本恢复，实例的监控，等等。

Java 架构学习资料（里面有高可用、高并发、高性能及分布式、Jvm 性能调优、Spring 源码，MyBatis，Netty,Redis,Kafka,MySQL,Zookeeper,Tomcat,Docker,Dubbo,Nginx 等多个知识点的架构资料）合理利用自己每一分每一秒的时间来学习提升自己，不要再用“没有时间”来掩饰自己思想上的懒惰！趁年轻，使劲拼，给未来的自己一个交代！

## 10、REDIS 集群方案什么情况下会导致整个集群不可用？

有 A, B, C 三个节点的集群, 在没有复制模型的情况下, 如果节点 B 失败了, 那么整个集群就会以为缺少 5501-11000 这个范围的槽而不可用。

## 11、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？

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。

其实面试除了考察 Redis，不少公司都很重视高并发高可用的技术，特别是一线互联网公司，分布式、

JVM、spring 源码分析、微服务等知识点已是面试的必考题。文末分享给大家一线互联网公司最新的技术知识（彩蛋）

## 12、REDIS 有哪些适合的场景？

### (1) 会话缓存 (Session Cache)

最常用的一种使用 Redis 的情景是会话缓存 (sessioncache)，用 Redis 缓存会话比其他存储（如 Memcached）的优势在于：Redis 提供持久化。当维护一个不是严格要求一致性的缓存时，如果用户的购物车信息全部丢失，大部分人都会不高兴的，现在，他们还会这样吗？

幸运的是，随着 Redis 这些年的改进，很容易找到怎么恰当的使用 Redis 来缓存会话的文档。甚至广为人知的商业平台 Magento 也提供 Redis 的插件。

### (2) 全页缓存 (FPC)

除基本的会话 token 之外，Redis 还提供很简便的 FPC 平台。回到一致性问题，即使重启了 Redis 实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降，这是一个极大改进，类似 PHP 本地 FPC。

再次以 Magento 为例，Magento 提供一个插件来使用 Redis 作为全页缓存后端。

此外，对 WordPress 的用户来说，Pantheon 有一个非常好的插件 wp-redis，这个插件能帮助你以最快速度加载你曾浏览过的页面。

### (3) 队列

Redis 在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得 Redis 能作为一个很好的消息队列平台来使用。Redis 作为队列使用的操作，就类似于本地程序语言（如 Python）对 list 的 push/pop 操作。

如果你快速的在 Google 中搜索 “Redis queues”，你马上就能找到大量的开源项目，这些项目的目的就是利用 Redis 创建非常好的后端工具，以满足各种队列需求。例如，Celery 有一个后台就是使用 Redis 作为 broker，你可以从这里去查看。

### (4) 排行榜 / 计数器

Redis 在内存中对数字进行递增或递减的操作实现的非常好。集合 (Set) 和有序集合 (SortedSet) 也使得我们在执行这些操作的时候变的非常简单，Redis 只是正好提供了这两种数据结构。

所以，我们要从排序集合中获取到排名最靠前的 10 个用户—我们称之为 “user\_scores”，我们只需要像下面一样执行即可：

当然，这是假定你是根据你用户的分数做递增的排序。如果你想返回用户及用户的分数，你需要这样执行：

ZRANGE user\_scores 0 10 WITHSCORES Agora Games 就是一个很好的例子，用 Ruby 实现的，它的排行榜就是使用 Redis 来存储数据的，你可以在这里看到。

#### (5) 发布 / 订阅

最后（但肯定不是最不重要的）是 Redis 的发布 / 订阅功能。发布 / 订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布 / 订阅的脚本触发器，甚至用 Redis 的发布 / 订阅功能来建立聊天系统！

## 13、REDIS 支持的 JAVA 客户端都有哪些？官方推荐用哪个？

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

## 14、REDIS 和 REDISSON 有什么关系？

Redisson 是一个高级的分布式协调 Redis 客户端，能帮助用户在分布式环境中轻松实现一些 Java 的对象 (Bloom filter, BitSet, Set, SetMultimap, ScoredSortedSet, SortedSet, Map, ConcurrentMap, List, ListMultimap, Queue, BlockingQueue, Deque, BlockingDeque, Semaphore, Lock, ReadWriteLock, AtomicLong, CountDownLatch, Publish / Subscribe, HyperLogLog)。

## 15、JEDIS 与 REDISSON 对比有什么优缺点？

Jedis 是 Redis 的 Java 实现的客户端，其 API 提供了比较全面的 Redis 命令的支持；

Redisson 实现了分布式和可扩展的 Java 数据结构，和 Jedis 相比，功能较为简单，不支持字符串操作，不支持排序、事务、管道、分区等 Redis 特性。Redisson 的宗旨是促进使用者对 Redis 的关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

## 16、说说 REDIS 哈希槽的概念？

Redis 集群没有使用一致性 hash，而是引入了哈希槽的概念，Redis 集群有 16384 个哈希槽，每个 key 通过 CRC16 校验后对 16384 取模来决定放置哪个槽，集群的每个节点负责一部分 hash 槽。

## 17、REDIS 集群的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。

## 18、REDIS 集群会有写操作丢失吗？为什么？

Redis 并不能保证数据的强一致性，这意味这在实际中集群在特定的条件下可能会丢失写操作。

## 19、REDIS 集群之间是如何复制的？

异步复制

## 20、REDIS 集群最大节点个数是多少？

16384 个

## 21、REDIS 集群如何选择数据库？

Redis 集群目前无法做数据库选择，默认在 0 数据库。

## 22、REDIS 中的管道有什么用？

一次请求 / 响应服务器能实现处理新的请求即使旧的请求还未被响应，这样就可以将多个命令发送到服务器，而不用等待回复，最后在一个步骤中读取该答复。

这就是管道（pipelining），是一种几十年来广泛使用的技术。例如许多 POP3 协议已经实现支持这个功能，大大加快了从服务器下载新邮件的过程。

## 23、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。

## 24、REDIS 事务相关的命令有哪几个？

MULTI、EXEC、DISCARD、WATCH

## 25、REDIS KEY 的过期时间和永久有效分别怎么设置？

EXPIRE 和 PERSIST 命令

## 26、REDIS 如何做内存优化？

尽可能使用散列表（hashes），散列表（是说散列表里面存储的数少）使用的内存非常小，所以你应该尽可能的将你的数据模型抽象到一个散列表里面。

比如你的 web 系统中有一个用户对象，不要为这个用户的名称，姓氏，邮箱，密码设置单独的 key，而是应该把这个用户的所有信息存储到一张散列表里面。

## 27、REDIS 回收进程如何工作的？

客户端连接了新的命令，添加了新的数据，Redis 检查内存使用情况，如果大于

！它/ 端运行了新的脚本，添加新的数据。Redis 也是内存使用有限，对未入 maxmemory 的限制，则根据设定好的策略进行回收。一个新的命令被执行，等等。

所以我们不断地穿越内存限制的边界，通过不断达到边界然后不断地回收回到边界以下。

如果一个命令的结果导致大量内存被使用（例如很大的集合的交集保存到一个新的键），不用多久内存限制就会被这个内存使用量超越。

## 28. 加锁机制

咱们来看上面那张图，现在某个客户端要加锁。如果该客户端面对的是一个 redis cluster 集群，他首先会根据 hash 节点选择一台机器。这里注意，仅仅只是选择一台机器！这点很关键！紧接着，就会发送一段 lua 脚本到 redis 上，那段 lua 脚本如下所示：

为啥要用 lua 脚本呢？因为一大坨复杂的业务逻辑，可以通过封装在 lua 脚本中发送给 redis，保证这段复杂业务逻辑执行的原子性。

那么，这段 lua 脚本是什么意思呢？这里 KEYS[1] 代表的是你加锁的那个 key，比如说：RLock lock = redisson.getLock("myLock"); 这里你自己设置了加锁的那个锁 key 就是 “myLock”。

ARGV[1]代表的就是锁 key 的默认生存时间，默认 30 秒。ARGV[2]代表的是加锁的客户端的 ID，类似于下面这样：8743c9c0-0795-4907-87fd-6c719a6b4586:1 给大家解释一下，第一段 if 判断语句，就是用 “exists myLock” 命令判断一下，如果你要加锁的那个锁 key 不存在的话，你就进行加锁。如何加锁呢？很简单，用下面的命令：hset myLock8743c9c0-0795-4907-87fd-6c719a6b4586:1 1，通过这个命令设置一个 hash 数据结构，这行命令执行后，会出现一个类似下面的数据结构：

上述就代表 “8743c9c0-0795-4907-87fd-6c719a6b4586:1” 这个客户端对 “myLock” 这个锁 key 完成了加锁。接着会执行 “pexpire myLock 30000” 命令，设置 myLock 这个锁 key 的生存时间是 30 秒。好了，到此为止，ok，加锁完成了。

## 29. 锁互斥机制

那么在这个时候，如果客户端 2 来尝试加锁，执行了同样的一段 lua 脚本，会咋样呢？很简单，第一个 if 判断会执行 “exists myLock”，发现 myLock 这个锁 key 已经存在了。接着第二个 if 判断，判断一下，myLock 锁 key 的 hash 数据结构中，是否包含客户端 2 的 ID，但是明显不是的，因为那里包含的是客户端 1 的 ID。

所以，客户端 2 会获取到 pttl myLock 返回的一个数字，这个数字代表了 myLock 这个锁 key 的剩余生存时间。比如还剩 15000 毫秒的生存时间。此时客户端 2 会进入一个 while 循环，不停的尝试加锁。

## 30.WATCH DOG 自动延期机制

客户端 1 加锁的锁 key 默认生存时间才 30 秒，如果超过了 30 秒，客户端 1 还想一直持有这把锁，怎么办呢？

**简单！**只要客户端 1 一旦加锁成功，就会启动一个 watch dog 看门狗，他是一个后台线程，会每隔 10 秒检查一下，如果客户端 1 还持有锁 key，那么就会不断的延长锁 key 的生存时间。

## 31. 可重入加锁机制

那如果客户端 1 都已经持有了这把锁了，结果可重入的加锁会怎么样呢？比如下面这种代码：

这时我们分析一下上面那段 lua 脚本。第一个 if 判断肯定不成立，“exists myLock” 会显示

锁 key 已经存在了。第二个 if 判断会成立，因为 myLock 的 hash 数据结构中包含的那个 ID，就是客户端 1 的那个 ID，也就是 “8743c9c0-0795-4907-87fd-6c719a6b4586:1”

此时就会执行可重入加锁的逻辑，他会用：

`incrby myLock 8743c9c0-0795-4907-87fd-6c719a6b4586:1 1`，通过这个命令，对客户端 1 的加锁次数，累加 1。此时 myLock 数据结构变为下面这样：

大家看到了吧，那个 myLock 的 hash 数据结构中的那个客户端 ID，就对应着加锁的次数

## 32. 释放锁机制

如果执行 `lock.unlock()`，就可以释放分布式锁，此时的业务逻辑也是非常简单的。其实说白了，就是每次都对 myLock 数据结构中的那个加锁次数减 1。如果发现加锁次数是 0 了，说明这个客户端已经不再持有锁了，此时就会用：“`del myLock`” 命令，从 redis 里删除这个 key。

然后呢，另外的客户端 2 就可以尝试完成加锁了。这就是所谓的分布式锁的开源 Redisson 框架的实现机制。

一般我们在生产系统中，可以用 Redisson 框架提供的这个类库来基于 redis 进行分布式锁的加锁与释放锁。

## 33. 上述 REDIS 分布式锁的缺点

其实上面那种方案最大的问题，就是如果你对某个 redis master 实例，写入了 myLock 这种锁 key 的 value，此时会异步复制给对应的 master slave 实例。但是这个过程中一旦发生 redis master 宕机，主备切换，redis slave 变为了 redis master。

接着就会导致，客户端 2 来尝试加锁的时候，在新的 redis master 上完成了加锁，而客户端 1 也以为自己成功加了锁。此时就会导致多个客户端对一个分布式锁完成了加锁。这时系统在业务语义上一定会有问题，导致各种脏数据的产生。

所以这个就是 redis cluster，或者是 redis master-slave 架构的主从异步复制导致的 redis 分布式锁的最大缺陷：在 redis master 实例宕机的时候，可能导致多个客户端同时完成加锁。

## 34. 使用过 REDIS 分布式锁么，它是如何实现？

先拿 `setnx` 来争抢锁，抢到之后，再用 `expire` 给锁加一个过期时间防止锁忘记了释放。

如果在 `setnx` 之后执行 `expire` 之前进程意外 crash 或者要重启维护了，那会怎么样？

`set` 指令有非常复杂的参数，这个应该是可以同时把 `setnx` 和 `expire` 合成一条指令来用的！

## 35. 使用过 REDIS 做异步队列么，你是怎么用的？有什么缺点？

般使用 list 结构作为队列，`rpush` 生产消息，`lpop` 消费消息。当 `lpop` 没有消息的时候，要适当 `sleep` 一会再重试。

缺点：

在消费者下线的情况下，生产的消息会丢失，得使用专业的消息队列如 `rabbitmq` 等。

能不能生产一次消费多次呢？

使用 pub/sub 主题订阅者模式，可以实现 1:N 的消息队列。

## 36. 什么是缓存穿透？如何避免？什么是缓存雪崩？何如避免？

### 缓存穿透

一般的缓存系统，都是按照 key 去缓存查询，如果不存在对应的 value，就应该去后端系统查找（比如 DB）。一些恶意的请求会故意查询不存在的 key，请求量很大，就会对后端系统造成很大的压力。这就叫做缓存穿透。

### 如何避免？

- 1: 对查询结果为空的情况也进行缓存，缓存时间设置短一点，或者该 key 对应的数据 insert 了之后清理缓存。
- 2: 对一定不存在的 key 进行过滤。可以把所有的可能存在的 key 放到一个大的 Bitmap 中，查询时通过该 bitmap 过滤。

### 缓存雪崩

当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，会给后端系统带来很大压力。导致系统崩溃。

### 如何避免？

- 1: 在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对某个 key 只允许一个线程查询数据和写缓存，其他线程等待。
- 2: 做二级缓存，A1 为原始缓存，A2 为拷贝缓存，A1 失效时，可以访问 A2，A1 缓存失效时间设置为短期，A2 设置为长期
- 3: 不同的 key，设置不同的过期时间，让缓存失效的时间点尽量均匀

## 37. REDIS 和 MEMCACHED 什么区别？为什么高并发下有时单线程的 REDIS 比多线程的 MEMCACHED 效率要高？

### 区别：

- 1.mc 可缓存图片和视频。rd 支持除 k/v 更多的数据结构；
- 2.rd 可以使用虚拟内存，rd 可持久化和 aof 灾难恢复，rd 通过主从支持数据备份；
- 3.rd 可以做消息队列。

### 原因：

mc 多线程模型引入了缓存一致性和锁，加锁带来了性能损耗。

redis 主从复制如何实现的？redis 的集群模式如何实现？redis 的 key 是如何寻址的？

**主从复制实现：**主节点将自己内存中的数据做一份快照，将快照发给从节点，从节点将数据恢复到内存中。之后再每次增加新数据的时候，主节点以类似于 mysql 的二进制日志方式将语句发送给从节点，从节点拿到主节点发送过来的语句进行重放。



### 分片方式：

- 客户端分片
- 基于代理的分片
- Twemproxy
- codis - 路由查询分片
- Redis-cluster（本身提供了自动将数据分散到 Redis Cluster 不同节点的能力，整个数据集合的某个数据子集存储在哪个节点对于用户来说是透明的）

redis-cluster 分片原理：Cluster 中有一个 16384 长度的槽（虚拟槽），编号分别为 0-16383。

每个 Master 节点都会负责一部分的槽，当有某个 key 被映射到某个 Master 负责的槽，那么这个 Master 负责为这个 key 提供服务，至于哪个 Master 节点负责哪个槽，可以由用户指定，也可以在初始化的时候自动生成，只有 Master 才拥有槽的所有权。Master 节点维护着一个 16384/8 字节的位序列，Master 节点用 bit 来标识对于某个槽自己是否拥有。比如对于编号为 1 的槽，Master 只要判断序列的第二位（索引从 0 开始）是不是为 1 即可。

这种结构很容易添加或者删除节点。比如如果我想新添加个节点 D，我需从节点 A、B、C 中得部分槽到 D 上。

## 38. 使用 REDIS 如何设计分布式锁？说一下实现思路？使用 ZK 可以吗？如何实现？这两种有什么区别？

### redis:

1. 线程 A 使用 `setnx` (上锁的对象, 超时时的时间戳 `t1`)，如果返回 `true`，获得锁。
2. 线程 B 用 `get` 获取 `t1`，与当前时间戳比较，判断是否超时，没超时 `false`，若超时执行第 3 步；
3. 计算新的超时时间 `t2`，使用 `getset` 命令返回 `t3` (该值可能其他线程已经修改过)，如果 `t1==t3`，获得锁，如果 `t1!=t3` 说明锁被其他线程获取了。
4. 获取锁后，处理完业务逻辑，再去判断锁是否超时，如果没超时删除锁，如果已超时，不用处理（防止删除其他线程的锁）。

### zk:

1. 客户端对某个方法加锁时，在 zk 上的与该方法对应的指定节点的目录下，生成一个唯一的临时有序节点 `node1`；
2. 客户端获取该路径下所有已经创建的子节点，如果发现自己创建的 `node1` 的序号是最小的，就认为这个客户端获得了锁。
3. 如果发现 `node1` 不是最小的，则监听比自己创建节点序号小的最大的节点，进入等待。
4. 获取锁后，处理完逻辑，删除自己创建的 `node1` 即可。

区别: zk 性能差一些，开销大，实现简单。

## 39. 知道 REDIS 的持久化吗？底层如何实现的？有什么优点缺点？

RDB(Redis DataBase: 在不同的时间点将 redis 的数据生成的快照同步到磁盘等介质上): 内存到硬盘的快照，定期更新。缺点：耗时，耗性能 (fork+io 操作)，易丢失数据。

AOF(Append Only File: 将 redis 所执行过的所有指令都记录下来，在下次 redis 重启时，只需要执行指令就可以了):

写日志。缺点：体积大，恢复速度慢。bgsave 做镜像全量持久化，aof 做增量持久化。因为 bgsave 会消耗比较长的时间，不够实时，在停机的时候会导致大量的数据丢失，需要 aof 来配合，在 redis 实例重启时，优先使用 aof 来恢复内存的状态，如果没有 aof 日志，就会使用 rdb 文件来恢复。Redis 会定期做 aof 重写，压缩 aof 文件日志大小。Redis4.0 之后有了混合持久化的功能，将 bgsave 的全量和 aof 的增量做了融合处理，这样既保证了恢复的效率又兼顾了数据的安全性。bgsave 的原理，fork 和 cow, fork 是指 redis 通过创建子进程来进行 bgsave 操作，cow 指的是 copy onwrite，子进程创建后，父子进程共享数据段，父进程继续提供读写服务，写脏的页面数据会逐渐和子进程分离开来。

redis 过期策略都有哪些？LRU 算法知道吗？写一下 java 代码实现？

#### 过期策略:

定时过期 (一 key 一定时器)，惰性过期：只有使用 key 时才判断 key 是否已过期，过期则清除。定期过期：前两者折中。

LRU:new LinkedHashMap<K, V>(capacity, DEFAULT\_LOAD\_FACTORY, true);// 第三个参数置为 true，代表 linkedlist 按访问顺序排序，可作为 LRU 缓存；设为 false 代表按插入顺序排序，可作为 FIFO 缓存

#### LRU 算法实现:

1. 通过双向链表来实现，新数据插入到链表头部；
2. 每当缓存命中（即缓存数据被访问），则将数据移到链表头部；
3. 当链表满的时候，将链表尾部的数据丢弃。

LinkedHashMap: HashMap 和双向链表合二为一即是 LinkedHashMap。HashMap 是无序的，LinkedHashMap 通过维护一个额外的双向链表保证了迭代顺序。该迭代顺序可以是插入顺序（默认），也可以是访问顺序。

## 40 缓存穿透、缓存击穿、缓存雪崩解决方案？

#### 缓存穿透:

指查询一个一定不存在的数据，如果从存储层查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到 DB 去查询，可能导致 DB 挂掉。

#### 解决方案:

1. 查询返回的数据为空，仍把这个空结果进行缓存，但过期时间会比较短；
2. 布隆过滤器：将所有可能存在的数据哈希到一个足够大的 bitmap 中，一个一定不存在的数据会被这个 bitmap 拦截掉，从而避免了对 DB 的查询。

#### 缓存击穿:

对于设置了过期时间的 key，缓存在某个时间点过期的时候，恰好这时间点对这个 Key 有大量的并发请求过来，这些请求发现缓存过期一般都会从后端 DB 加载数据并回设到缓存，这个时候大并发的请求可能会瞬间把 DB 压垮。

### 解决方案：

1. 使用互斥锁：当缓存失效时，不立即去 load db，先使用如 Redis 的 setnx 去设置一个互斥锁，当操作成功返回时再进行 load db 的操作并回设缓存，否则重试 get 缓存的方法。
2. 永远不过期：物理不过期，但逻辑过期（后台异步线程去刷新）。

缓存雪崩：设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到 DB，DB 瞬时压力过重雪崩。与缓存击穿的区别：雪崩是很多 key，击穿是某一个 key 缓存。

解决方案：将缓存失效时间分散开，比如可以在原有的失效时间基础上增加一个随机值，比如 1-5 分钟随机，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件。

## 41. 在选择缓存时，什么时候选择 REDIS，什么时候选择 MEMCACHED？

### 选择 redis 的情况：

- 1、复杂数据结构，value 的数据是哈希，列表，集合，有序集合等这种情况下，会选择 redis，因为 memcache 无法满足这些数据结构，最典型的的使用场景是，用户订单列表，用户消息，帖子评论等。
- 2、需要进行数据的持久化功能，但是注意，不要把 redis 当成数据库使用，如果 redis 挂了，内存能够快速恢复热数据，不会将压力瞬间压在数据库上，没有 cache 预热的过程。对于只读和数据一致性要求不高的场景可以采用持久化存储
- 3、高可用，redis 支持集群，可以实现主动复制，读写分离，而对于 memcache 如果想要实现高可用，需要进行二次开发。
- 4、存储的内容比较大，memcache 存储的 value 最大为 1M。

### 选择 memcache 的场景：

- 1、纯 KV，数据量非常大的业务，使用 memcache 更合适，原因是：

a) memcache 的内存分配采用的是预分配内存池的管理方式，能够省去内存分配的时间，redis 是临时申请空间，可能导致碎片化。

b) 虚拟内存使用，memcache 将所有的数据存储在物理内存里，redis 有自己的 vm 机制，理论上能够存储比物理内存更多的数据，当数据超量时，引发 swap，把冷数据刷新到磁盘上，从这点上，数据量大时，memcache 更快

c) 网络模型，memcache 使用非阻塞的 IO 复用模型，redis 也是使用非阻塞的 IO 复用模型，但是 redis 还提供了一些非 KV 存储之外的排序，聚合功能，复杂的 CPU 计算，会阻塞整个 IO 调度，从这点上由于 redis 提供的功能较多，memcache 更快些

d) 线程模型，memcache 使用多线程，主线程监听，worker 子线程接受请求，执行读写，这个过程可能存在锁冲突。redis 使用的单线程，虽然无锁冲突，但是难以利用多核的特性提升吞吐量。

### 缓存与数据库不一致怎么办

假设采用的主存分离，读写分离的数据库，

如果一个线程 A 先删除缓存数据，然后将数据写入到主库当中，这个时候，主库和从库同步没有完成，线程 B 从缓存当中读取数据失败，从从库当中读取到旧数据，然后更新至缓存，这个时候，缓存当中的就是旧的数据。

发生上述不一致的原因在于，主从库数据不一致问题，加入了缓存之后，主从不一致的时间被拉长了

处理思路：在从库有数据更新之后，将缓存当中的数据也同时进行更新，即当从库发生了数据更新之后，向缓存发出删除，淘汰这段时间写入的旧数据。

主从数据库不一致如何解决场景描述，对于主从库，读写分离，如果主从库更新同步有时差，就会导致主从库数据的不一致

- 1、忽略这个数据不一致，在数据一致性要求不高的业务下，未必需要时时一致性
- 2、强制读主库，使用一个高可用的主库，数据库读写都在主库，添加一个缓存，提升数据读取的性能。
- 3、选择性读主库，添加一个缓存，用来记录必须读主库的数据，将哪个库，哪个表，哪个主键，作为缓存的 key，设置缓存失效的时间为主从库同步的时间，如果缓存当中有这个数据，直接读取主库，如果缓存当中没有这个主键，就到对应的从库中读取。

## 42.REDIS 常见的性能问题和解决方案

- 1、master 最好不要做持久化工作，如 RDB 内存快照和 AOF 日志文件
- 2、如果数据比较重要，某个 slave 开启 AOF 备份，策略设置成每秒同步一次
- 3、为了主从复制的速度和连接的稳定性，master 和 Slave 最好在一个局域网内
- 4、尽量避免在压力大得主库上增加从库
- 5、主从复制不要采用网状结构，尽量是线性结构，Master<--Slave1<----Slave2 ....

## 43.REDIS 的数据淘汰策略有哪些

volatile-lru 从已经设置过期时间的数据集中挑选最近最少使用的数据淘汰

volatile-ttl 从已经设置过期时间的数据库集中挑选将要过期的数据

volatile-random 从已经设置过期时间的数据集任意选择淘汰数据

allkeys-lru 从数据集中挑选最近最少使用的数据淘汰

allkeys-random 从数据集中任意选择淘汰的数据

no-eviction 禁止驱逐数据

## 44.REDIS 当中有哪些数据结构

字符串 String、字典 Hash、列表 List、集合 Set、有序集合 SortedSet。如果是高级用户，那么还会有，如果你是 Redis 中高级用户，还需要加上下面几种数据结构 HyperLogLog、Geo、Pub/Sub。

假如 Redis 里面有 1 亿个 key，其中有 10w 个 key 是以某个固定的已知的前缀开头的，如果将它们全部找出来？

使用 keys 指令可以扫出指定模式的 key 列表。

对方接着追问：如果这个 redis 正在给线上的业务提供服务，那使用 keys 指令会有什么问题？

这个时候你要回答 redis 关键的一个特性：redis 的单线程的。keys 指令会导致线程阻塞一段时间，线上服务会停顿，直到指令执行完毕，服务才能恢复。这个时候可以使用 scan 指令，scan 指令可以无阻塞的提取出指定模式的 key 列表，但是会有一定的重复概率，在客户端做一次去重就可以了，但是整体所花费的时间会比直接用 keys 指令长。

## 45. 使用 REDIS 做过异步队列吗，是如何实现的

















使用 list 类型保存数据信息，rpush 生产消息，lpop 消费消息，当 lpop 没有消息时，可以 sleep 一段时间，然后再检查有没有信息，如果不想 sleep 的话，可以使用 blpop, 在没有信息的时候，会一直阻塞，直到信息的到来。redis 可以通过 pub/sub 主题订阅模式实现一个生产者，多个消费者，当然也存在一定的缺点，当消费者下线时，生产的消息会丢失。

## 46.Redis 如何实现延时队列

使用 sortedset，使用时间戳做 score，消息内容作为 key，调用 zadd 来生产消息，消费者使用 zrangbyscore 获取 n 秒之前的数据做轮询处理。

关注我后台私信回复【java 架构】领取面试资料（助你金三银四能跳槽涨薪）

## 收集了还有你不知道的其它面试题 (SPRINGBOOT、MYBATIS、并发、 JAVA 中高级面试总结等)

 消息中间件面试专题及答案.pdf	2019/2/25 20:59	WPS PDF 文档
 微服务面试专题及答案.pdf	2019/2/25 20:59	WPS PDF 文档
 数据库面试专题及答案.pdf	2019/2/25 20:59	WPS PDF 文档
 设计模式面试专题及答案.pdf	2019/2/25 20:59	WPS PDF 文档
 面试必备之乐观锁与悲观锁.pdf	2019/2/25 21:22	WPS PDF 文档
 开源框架面试专题及答案.pdf	2019/2/25 20:59	WPS PDF 文档
 多线程面试专题及答案.pdf	2019/2/25 20:58	WPS PDF 文档
 并发编程面试专题及答案（下）.pdf	2019/2/25 20:58	WPS PDF 文档
 并发编程及答案（上）.pdf	2019/2/25 20:58	WPS PDF 文档
 zookeeper面试专题及答案.pdf	2019/2/25 20:54	WPS PDF 文档
 Tomcat面试专题及答案.pdf	2019/2/25 20:54	WPS PDF 文档
 SQL优化面试专题及答案.pdf	2019/2/25 20:54	WPS PDF 文档
 Spring面试专题及答案.pdf	2019/2/25 20:54	WPS PDF 文档
 SpringMVC面试专题及答案.pdf	2019/2/25 20:54	WPS PDF 文档
 SpringCloud面试专题及答案.pdf	2019/2/25 20:54	WPS PDF 文档
 SpringBoot面试专题及答案.pdf	2019/2/25 20:54	WPS PDF 文档

 RabbitMQ消息中间件面试专题及答案.p...	2019/2/25 20:53	WPS PDF 文档
 Nginx面试专题及答案.pdf	2019/2/25 20:53	WPS PDF 文档

MySQL性能优化的21个最佳实践.pdf	2019/2/25 20:53	WPS PDF 文档
mysql面试专题及答案.pdf	2019/2/25 20:53	WPS PDF 文档
MySQL55题及答案.pdf	2019/2/25 20:43	WPS PDF 文档
MyBatis面试专题及答案.pdf	2019/2/25 20:53	WPS PDF 文档
MongoDB面试专题及答案.pdf	2019/2/25 20:50	WPS PDF 文档
memcached面试专题及答案.pdf	2019/2/25 20:49	WPS PDF 文档
Linux面试专题及答案.pdf	2019/2/25 20:48	WPS PDF 文档
Kafka面试专题及答案.pdf	2019/2/25 20:47	WPS PDF 文档
JVM面试专题及答案.pdf	2019/2/25 20:46	WPS PDF 文档
Java基础面试题.pdf	2019/2/25 21:22	WPS PDF 文档
java后端面试题答案.pdf	2019/2/24 19:07	WPS PDF 文档
Dubbo面试专题及答案（下）.pdf	2019/2/25 20:45	WPS PDF 文档
Dubbo面试及答案（上）.pdf	2019/2/25 20:45	WPS PDF 文档
ActiveMQ消息中间件面试专题.pdf	2019/2/25 20:35	WPS PDF 文档

写下你的评论...

不错

全文完

本文由 简悦 SimpRead 优化，用以提升阅读体验

使用了 全新的简悦词法分析引擎 <sup>beta</sup>，[点击查看详细说明](#)

