**Redis**

**问：Redis 有哪些优势？**

1. 速度快，因为数据存在内存中。
2. 支持丰富数据类型，支持 string、list、set 、sorted set、hash。
3. 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行。
4. 丰富的特性：可用于缓存，消息，按 key 设置过期时间，过期后将会自动删除。
5. 单线程，单进程，采用 IO 多路复用技术。

**问：Redis 的存储结构是怎样的？**

key-value 键值对。

**问：Redis 支持哪些数据结构？**

string（[字符串](https://www.zhihu.com/search?q=%E5%AD%97%E7%AC%A6%E4%B8%B2&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2273872574}" \t "https://www.zhihu.com/_blank)

）, hash（哈希）, list（队列）, set（集合）及 zset（sorted set 有序集合）。

**问：Redis 的数据结构，有哪些应用场景？**

* string：简单地 get / set 缓存。
* hash：可以缓存用户资料。比如命令：hmset user1 name "lin" sex "male" age "25" ，缓存用户 user1 的资料，姓名为 lin ，性别为男，年龄 25。
* list：可以做队列。往 list 队列里面 push 数据，然后再 pop 出来。
* zset：可以用来做排行榜。

**问：Redis 的数据结构，底层分别是由什么实现的？**

* Redis 字符串，却不是 C 语言中的字符串（即以空字符 ’\0’ 结尾的字符数组），它是自己构建了一种名为 简单动态字符串（simple dynamic string , SDS）的抽象类型，并将 SDS 作为 Redis 的默认字符串表示。
* Redi List ，底层是 ZipList ，不满足 ZipList 就使用[双向链表](https://www.zhihu.com/search?q=%E5%8F%8C%E5%90%91%E9%93%BE%E8%A1%A8&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2273872574}" \t "https://www.zhihu.com/_blank)
* 。ZipList 是为了节约内存而开发的。和各种语言的数组类似，它是由连续的内存块组成的，这样一来，由于内存是连续的，就减少了很多内存碎片和指针的内存占用，进而节约了内存。

**问：Redis 怎么保证可靠性？Redis 的持久化方式有哪些？有哪些优缺点？**

一个可靠安全的系统，肯定要考虑数据的可靠性，尤其对于内存为主的 Redis ，就要考虑一旦服务器挂掉，启动之后，如何恢复数据的问题，也就是说数据如何持久化的问题。

AOF 就是备份操作记录。AOF 由于是备份操作命令，备份快、恢复慢。

AOF 的优点：AOF 更好保证数据不会被丢失，最多只丢失一秒内的数据。另外重写操作保证了数据的有效性，即使日志文件过大也会进行重写。AOF 的日志文件的记录可读性非常的高。

AOF 的缺点：对于相同数量的数据集而言， AOF 文件通常要大于 RDB 文件。

RDB 就是备份所有数据，使用了快照。RDB 恢复数据比较快。

**问：AOF 文件过大，怎么处理？**

会进行 AOF 文件重写。

* 随着 AOF 文件越来越大，里面会有大部分是重复命令或者可以合并的命令。
* 重写的好处：减少 AOF 日志尺寸，减少内存占用，加快数据库恢复时间。

执行一个 AOF 文件重写操作，重写会创建一个当前 AOF 文件的体积优化版本。

**问：讲一下 Redis 的事务。**

先以 MULTI 开始一个事务， 然后将多个命令入队到事务中， 最后由 EXEC 命令触发事务， 一并执行事务中的所有命令。如果想放弃这个事务，可以使用 DISCARD 命令。

**问：Redis 事务无法回滚，那怎么处理？**

**问：怎么设置 Redis 的 key 过期时间？**

key 的的过期时间通过 EXPIRE key seconds 命令来设置数据的过期时间。返回 1 表明设置成功，返回 0 表明 key 不存在或者不能成功设置过期时间。

**问：Redis 的过期策略有哪些？**

惰性删除：当读/写一个已经过期的 key 时，会触发惰性删除策略，直接删除掉这个过期 key ，并按照 key 不存在去处理。惰性删除，对内存不太好，已经过期的 key 会占用太多的内存。

定期删除：每隔一段时间，就会对 Redis 进行检查，主动删除一批已过期的 key。

**问：为什么 Redis 不使用定时删除？**

定时删除，就是在设置 key 的过期时间的同时，创建一个定时器，让定时器在过期时间来临时，立即执行对 key 的删除操作。

定时删会占用 CPU ，影响服务器的响应时间和性能。

**问：Redis 的内存回收机制都有哪些？**

当前已用内存超过 maxmemory 限定时，会触发主动清理策略，也就是 Redis 的内存回收策略。

LRU 、TTL。

noeviction ：默认策略，不会删除任何数据，拒绝所有写入操作并返回客户端错误信息，此时 Redis 只响应读操作。

* volatitle - lru ：根据 LRU 算法删除设置了超时属性的键，知道腾出足够空间为止。如果没有可删除的键对象，回退到 noeviction 策略。
* allkeys - lru ：根据 LRU 算法删除键，不管数据有没有设置超时属性，直到腾出足够空间为止。
* allkeys - random ：随机删除所有键，知道腾出足够空间为止。
* volatitle - random ：随机删除过期键，知道腾出足够空间为止。
* volatitle - ttl ：根据键值对象的 ttl 属性，删除最近将要过期数据。如果没有，回退到 noeviction 策略。

**问：手写一下 LRU 算法。**

**问：Redis 的搭建有哪些模式？**

主从模式、哨兵模式、Cluster（集群）模式。最好是用集群模式。

**问：你用过的 Redis 是多主多从的，还是一主多从的？集群用到了多少节点？用到了多少个哨兵？**

集群模式。三主三从。

**问：Redis 采用多主多从的集群模式，各个主节点的数据是否一致？**

**问：Redis 集群有哪些特性**

master 和 slaver。主从复制。读写分离。哨兵模式。

**问：Redis 是怎么进行水平扩容的？**

**问：Redis 集群数据分片的原理是什么？**

Redis 数据分片原理是哈希槽（hash slot）。

Redis 集群有 16384 个哈希槽。每一个 Redis 集群中的节点都承担一个哈希槽的子集。

哈希槽让在集群中添加和移除节点非常容易。例如，如果我想添加一个新节点 D ，我需要从节点 A 、B、C 移动一些哈希槽到节点 D。同样地，如果我想从集群中移除节点 A ，我只需要移动 A 的哈希槽到 B 和 C。当节点 A 变成空的以后，我就可以从集群中彻底删除它。因为从一个节点向另一个节点移动哈希槽并不需要停止操作，所以添加和移除节点，或者改变节点持有的哈希槽百分比，都不需要任何停机时间（downtime）。

**问：讲一下一致性 Hash 算法。**

一致性 Hash 算法将整个哈希值空间组织成一个虚拟的圆环, 我们对 key 进行哈希计算，使用哈希后的结果对 2 ^ 32 取模，hash 环上必定有一个点与这个整数对应。依此确定此数据在环上的位置，从此位置沿环顺时针“行走”，第一台遇到的服务器就是其应该定位到的服务器。

一致性 Hash 算法对于节点的增减都只需重定位环空间中的一小部分数据，具有较好的容错性和可扩展性。

比如，集群有四个节点 Node A 、B 、C 、D ，增加一台节点 Node X。Node X 的位置在 Node B 到 Node C 直接，那么受到影响的仅仅是 Node B 到 Node X 间的数据，它们要重新落到 Node X 上。

所以一致性哈希算法对于容错性和扩展性有非常好的支持。

**问：为什么 Redis Cluster 分片不使用 Redis 一致性 Hash 算法？**

一致性哈希算法也有一个严重的问题，就是数据倾斜。

如果在分片的集群中，节点太少，并且分布不均，一致性哈希算法就会出现部分节点数据太多，部分节点数据太少。也就是说无法控制节点存储数据的分配。

**问：集群的拓扑结构有没有了解过？集群是怎么连接的？**

无中心结构。Redis-Cluster 采用无中心结构，每个节点保存数据和整个集群状态,每个节点都和其他所有节点连接。

**问：讲一下 Redis 主从复制的过程。**

从机发送 SYNC（同步）命令，主机接收后会执行 BGSAVE（异步保存）命令备份数据。

主机备份后，就会向从机发送备份文件。主机之后还会发送缓冲区内的写命令给从机。 当[缓冲区](https://www.zhihu.com/search?q=%E7%BC%93%E5%86%B2%E5%8C%BA&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2273872574}" \t "https://www.zhihu.com/_blank)

命令发送完成后，主机执行一条写命令，就会往从机发送同步写入命令。

**问：讲一下 Redis [哨兵机制](https://www.zhihu.com/search?q=%E5%93%A8%E5%85%B5%E6%9C%BA%E5%88%B6&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2273872574}" \t "https://www.zhihu.com/_blank)**

**。**

下面是 Redis 官方文档对于哨兵功能的描述：

* 监控（Monitoring）：哨兵会不断地检查[主节点](https://www.zhihu.com/search?q=%E4%B8%BB%E8%8A%82%E7%82%B9&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2273872574}" \t "https://www.zhihu.com/_blank)

· 和从节点是否运作正常。

· 自动故障转移（Automatic Failover）：当主节点不能正常工作时，哨兵会开始自动故障转移操作，它会将失效主节点的其中一个从节点升级为新的主节点，并让其他从节点改为复制新的主节点。

· 配置提供者（Configuration Provider）：客户端在初始化时，通过连接哨兵来获得当前 Redis 服务的主节点地址。

· 通知（Notification）：哨兵可以将故障转移的结果发送给[客户端](https://www.zhihu.com/search?q=%E5%AE%A2%E6%88%B7%E7%AB%AF&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2273872574}" \t "https://www.zhihu.com/_blank)

* 。

**问：讲一下布隆过滤器。**

布隆过滤器的主要是由一个很长的二进制向量和若干个（k 个）散列映射函数组成。因为每个元数据的存储信息值固定，而且总的二进制向量固定。所以在内存占用和查询时间上都远远超过一般的算法。当然存在一定的不准确率（可以控制）和不容易删除样本数据。

[布隆过滤器](https://www.zhihu.com/search?q=%E5%B8%83%E9%9A%86%E8%BF%87%E6%BB%A4%E5%99%A8&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2273872574}" \t "https://www.zhihu.com/_blank)

的优点：大批量数据去重，特别的占用内存。但是用布隆过滤器（Bloom Filter）会非常的省内存。

布隆过滤器的特点：当布隆过滤器说某个值存在时，那可能就不存在，如果说某个值不存在时，那肯定就是不存在了。

布隆过滤器的应用场景：新闻推送（不重复推送）。解决缓存穿透的问题。

**缓存**

**问：缓存雪崩是什么？**

如果缓存数据设置的过期时间是相同的，并且 Redis 恰好将这部分数据全部删光了。这就会导致在这段时间内，这些缓存同时失效，全部请求到数据库中。这就是缓存雪崩。

**问：怎么解决缓存雪崩？**

解决方法：在缓存的时候给过期时间加上一个随机值，这样就会大幅度的减少缓存在同一时间过期。

**问：缓存穿透是什么？**

缓存穿透是指查询一个一定不存在的数据。由于缓存不命中，并且出于容错考虑，如果从数据库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数据库去查询，失去了缓存的意义。

**问：怎么解决缓存穿透？**

**问：什么是缓存与数据库双写一致问题？**

**问：如何保证缓存与数据库的一致性？**

读的时候，先读缓存，缓存没有的话，就读数据库，然后取出数据后放入缓存，同时返回响应。

先删除缓存，再更新数据库。

**问：为什么是先删除缓存，而不是先更新缓存？**

**问：先更新数据库，再删除缓存，会有什么问题？**

先更新数据库，再删除缓存。可能出现以下情况：

* 如果更新完数据库， Java 服务提交了事务，然后挂掉了，那 Redis 还是会执行，这样也会不一致。
* 如果更新数据库成功，删除缓存失败了，那么会导致数据库中是新数据，缓存中是旧数据，数据就出现了不一致。

先删除缓存，再更新数据库。

* 如果删除缓存失败，那就不更新数据库，缓存和数据库的数据都是旧数据，数据是一致的。
* 如果删除缓存成功，而数据库更新失败了，那么数据库中是旧数据，缓存中是空的，数据不会不一致。因为读的时候缓存没有，所以去读了数据库中的旧数据，然后更新到缓存中。

**问：先删除缓存，在写数据库成功之前，如果有读请求发生，可能导致旧数据入缓存，引发数据不一致，怎么处理？**

**分布式锁**

**问：Redis 如何实现分布式锁？**

使用 set key value ex nx 命令。

* 当 key 不存在时，将 key 的值设为 value ，返回 1。若给定的 key 已经存在，则 setnx 不做任何动作，返回 0。
* 当 setnx 返回 1 时，表示获取锁，做完操作以后 del key ，表示释放锁，如果 setnx 返回 0 表示获取锁失败。

详细的命令如下：

set key value [EX seconds] [PX milliseconds] [NX|XX]EX

seconds：设置失效时长，单位秒PX

milliseconds：设置失效时长，单位毫秒

NX：key不存在时设置value，成功返回OK，失败返回(nil)

XX：key存在时设置value，成功返回OK，失败返回(nil)。

示例如下：

set name fenglin ex 100 nx

**问：为什么不先 set nx ，然后再使用 expire 设置超时时间？**

我们需要保证 setnx 命令和 expire 命令以原子的方式执行，否则如果客户端执行 setnx 获得锁后，这时客户端宕机了，那么这把锁没有设置过期时间，导致其他客户端永远无法获得锁了。

**问：使用 Redis 分布式锁， key 和 value 分别设置成什么？**

value 可以使用 json 格式的字符串，示例：

{ "count":1, "expireAt":147506817232, "jvmPid":22224, "mac":"28-D2-44-0E-0D-9A", "threadId":14}

**问：Redis 实现的分布式锁，如果某个系统获取锁后，宕机了怎么办？**

系统模块宕机的话，可以通过设置过期时间（就是设置缓存失效时间）解决。系统宕机时锁阻塞，过期后锁释放。

**问：设置缓存失效时间，那如果前一个线程把这个锁给删除了呢？**

**问：如果加锁和解锁之间的业务逻辑执行的时间比较长，超过了锁过期的时间，执行完了，又删除了锁，就会把别人的锁给删了。怎么办？**

这两个属于锁超时的问题。

可以将锁的 value 设置为 Json 字符串，在其中加入线程的 id 或者请求的 id ，在删除之前， get 一下这个 key ，判断 key 对应的 value 是不是当前线程的。只有是当前线程获取的锁，当前线程才可以删除。

**问：Redis 分布式锁，怎么保证可重入性？**

可以将锁的 value 设置为 Json 字符串，在其中加入线程的 id 和 count 变量。

* 当 count 变量的值为 0 时，表示当前分布式锁没有被线程占用。
* 如果 count 变量的值大于 0 ，线程 id 不是当前线程，表示当前分布式锁已经被其他线程占用。
* 如果 count 变量的值大于 0 ，线程 id 是当前线程的 id ，表示当前线程已经拿到了锁，不必阻塞，可以直接重入，并将 count 变量的值加一即可。

这种思路，其实就是参考了 ReentrantLock 可重入锁的机制。

**问：Redis 做分布式锁， Redis 做了主从，如果设置锁之后，主机在传输到从机的时候挂掉了，从机还没有加锁信息，如何处理？**

可以使用开源框架 Redisson ，采用了 redLock。

**问：讲一下 Redis 的 redLock。**

**问：Zookeeper 是怎么实现分布式锁的？**

分布式锁：基于 Zookeeper 一致性文件系统,实现锁服务。锁服务分为保存独占及时序控制两类。

* 保存独占：将 Zookeeper 上的一个 znode 看作是一把锁，通过 createznode 的方式来实现。所有客户端都去创建 / distribute \_ lock 节点，最终成功创建的那个客户端也即拥有了这把锁。用完删除自己创建的 distribute \_ lock 节点就释放锁。
* [时序控制](https://www.zhihu.com/search?q=%E6%97%B6%E5%BA%8F%E6%8E%A7%E5%88%B6&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"answer","sourceId":2273872574}" \t "https://www.zhihu.com/_blank)
* ：基于/ distribute \_ lock 锁，所有客户端在它下面创建临时顺序编号目录节点，和选 master 一样，编号最小的获得锁，用完删除，依次方便。

更详细的回答如下：

其实基于 Zookeeper ，就是使用它的临时有序节点来实现的分布式锁。

原理就是：当某客户端要进行逻辑的加锁时，就在 Zookeeper 上的某个指定节点的目录下，去生成一个唯一的临时有序节点， 然后判断自己是否是这些有序节点中序号最小的一个，如果是，则算是获取了锁。如果不是，则说明没有获取到锁，那么就需要在序列中找到比自己小的那个节点，并对其调用 exist() 方法，对其注册事件监听，当监听到这个节点被删除了，那就再去判断一次自己当初创建的节点是否变成了序列中最小的。如果是，则获取锁，如果不是，则重复上述步骤。

当释放锁的时候，只需将这个临时节点删除即可。