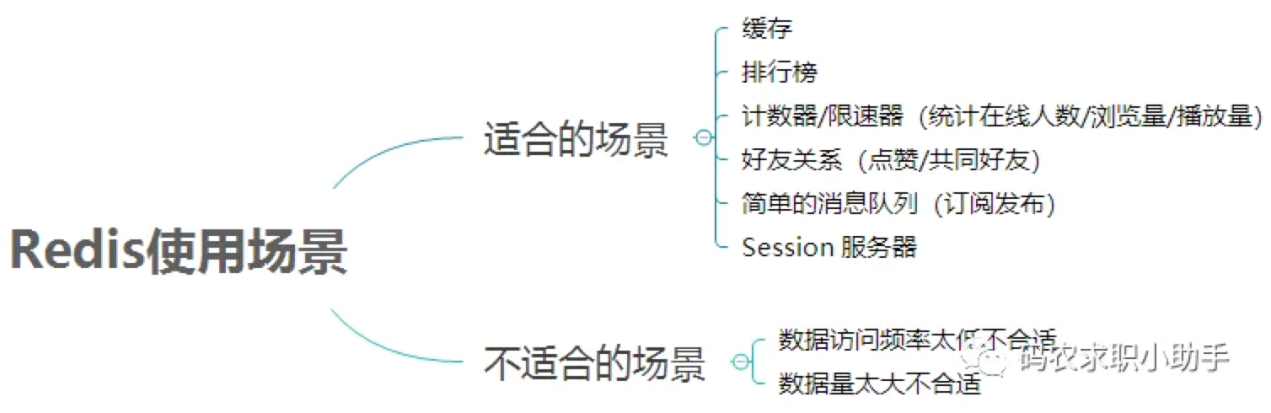
**1、谈下你对 Redis 的了解？**

Redis（全称：Remote Dictionary Server 远程字典服务）是一个开源的使用 ANSI C 语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value 数据库，并提供多种语言的 API。

**2、Redis 一般都有哪些使用场景？**



**Redis 适合的场景**

1. 缓存：减轻 MySQL 的查询压力，提升系统性能；
2. 排行榜：利用 Redis 的 SortSet（有序集合）实现；
3. 计算器/限速器：利用 Redis 中原子性的自增操作，我们可以统计类似用户点赞数、用户访问数等。这类操作如果用 MySQL，频繁的读写会带来相当大的压力；限速器比较典型的使用场景是限制某个用户访问某个 API 的频率，常用的有抢购时，防止用户疯狂点击带来不必要的压力；
4. 好友关系：利用集合的一些命令，比如求交集、并集、差集等。可以方便解决一些共同好友、共同爱好之类的功能；
5. 消息队列：除了 Redis 自身的发布/订阅模式，我们也可以利用 List 来实现一个队列机制，比如：到货通知、邮件发送之类的需求，不需要高可靠，但是会带来非常大的 DB 压力，完全可以用 List 来完成异步解耦；
6. Session 共享：Session 是保存在服务器的文件中，如果是集群服务，同一个用户过来可能落在不同机器上，这就会导致用户频繁登陆；采用 Redis 保存 Session 后，无论用户落在那台机器上都能够获取到对应的 Session 信息。

**Redis 不适合的场景**

数据量太大、数据访问频率非常低的业务都不适合使用 Redis，数据太大会增加成本，访问频率太低，保存在内存中纯属浪费资源。

**3、Redis 有哪些常见的功能？**

1. 数据缓存功能
2. 分布式锁的功能
3. 支持数据持久化
4. 支持事务
5. 支持消息队列

**4、Redis 支持的数据类型有哪些？**

* **1. string 字符串**

字符串类型是 Redis 最基础的数据结构，首先键是字符串类型，而且其他几种结构都是在字符串类型基础上构建的。字符串类型实际上可以是字符串：简单的字符串、XML、JSON；数字：整数、浮点数；二进制：图片、音频、视频。

使用场景：缓存、计数器、共享 Session、限速。

* **2. Hash（哈希）**

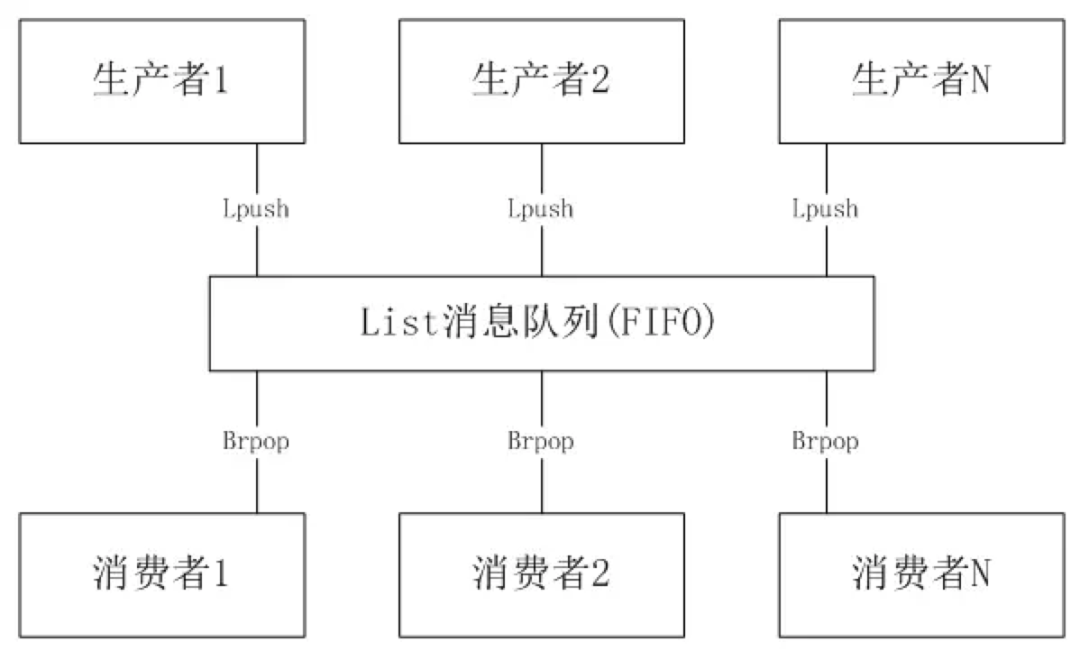
在 Redis中哈希类型是指键本身是一种键值对结构，如 value={{field1,value1},……{fieldN,valueN}}

使用场景：哈希结构相对于字符串序列化缓存信息更加直观，并且在更新操作上更加便捷。所以常常用于用户信息等管理，但是哈希类型和关系型数据库有所不同，哈希类型是稀疏的，而关系型数据库是完全结构化的，关系型数据库可以做复杂的关系查询，而 Redis 去模拟关系型复杂查询开发困难且维护成本高。

* **3. List（列表）**

列表类型是用来储存多个有序的字符串，列表中的每个字符串成为元素，一个列表最多可以储存 2 ^ 32 - 1 个元素，在 Redis 中，可以队列表两端插入和弹出，还可以获取指定范围的元素列表、获取指定索引下的元素等，列表是一种比较灵活的数据结构，它可以充当栈和队列的角色。

使用场景：Redis 的 lpush + brpop 命令组合即可实现阻塞队列，生产者客户端是用 lpush 从列表左侧插入元素，多个消费者客户端使用 brpop 命令阻塞式的“抢”列表尾部的元素，多个客户端保证了消费的负载均衡和高可用性。



* **4. Set（集合）**

集合类型也是用来保存多个字符串的元素，但和列表不同的是集合中不允许有重复的元素，并且集合中的元素是无序的，不能通过索引下标获取元素，Redis 除了支持集合内的增删改查，同时还支持多个集合取交集、并集、差集。合理的使用好集合类型，能在实际开发中解决很多实际问题。

使用场景：如：一个用户对娱乐、体育比较感兴趣，另一个可能对新闻感兴趣，这些兴趣就是标签，有了这些数据就可以得到同一标签的人，以及用户的共同爱好的标签，这些数据对于用户体验以及曾强用户粘度比较重要。

* **5. zset（sorted set：有序集合）**

有序集合和集合有着必然的联系，它保留了集合不能有重复成员的特性，但不同得是，有序集合中的元素是可以排序的，但是它和列表的使用索引下标作为排序依据不同的是：它给每个元素设置一个分数，作为排序的依据。

使用场景：排行榜是有序集合经典的使用场景。例如：视频网站需要对用户上传的文件做排行榜，榜单维护可能是多方面：按照时间、按照播放量、按照获得的赞数等。

**5、Redis 为什么这么快？**

1. 完全基于内存，绝大部分请求是纯粹的内存操作，非常快速；
2. 数据结构简单，对数据操作也简单；
3. 采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；
4. 使用多路 I/O 复用模型，非阻塞 IO。

**6、什么是缓存穿透？怎么解决？**

缓存穿透是指查询一个一定不存在的数据，由于缓存是不命中时需要从数据库查询，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，造成缓存穿透。

**解决办法：**

1、缓存空对象：如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），我们仍然把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

**缓存空对象带来的问题：**

1. 空值做了缓存，意味着缓存中存了更多的键，需要更多的内存空间，比较有效的方法是针对这类数据设置一个较短的过期时间，让其自动剔除。
2. 缓存和存储的数据会有一段时间窗口的不一致，可能会对业务有一定影响。例如：过期时间设置为 5分钟，如果此时存储添加了这个数据，那此段时间就会出现缓存和存储数据的不一致，此时可以利用消息系统或者其他方式清除掉缓存层中的空对象。

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，从而避免了对底层存储系统的查询压力。

**7、什么是缓存雪崩？该如何解决？**

如果缓存集中在一段时间内失效，发生大量的缓存穿透，所有的查询都落在数据库上，造成了缓存雪崩。

**解决办法：**

1. 加锁排队：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. 数据预热：可以通过缓存 reload 机制，预先去更新缓存，再即将发生大并发访问前手动触发加载缓存不同的 key，设置不同的过期时间，让缓存失效的时间点尽量均匀；
3. 做二级缓存，或者双缓存策略：Cache1 为原始缓存，Cache2 为拷贝缓存，Cache1 失效时，可以访问 Cache2，Cache1 缓存失效时间设置为短期，Cache2 设置为长期。
4. 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。

**8、 怎么保证缓存和数据库数据的一致性？**

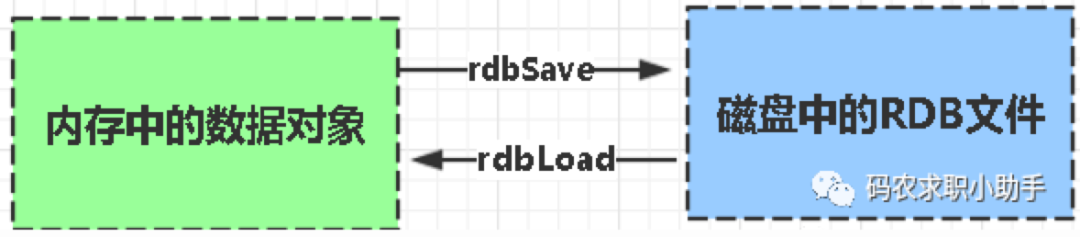
1. 从理论上说，只要我们设置了合理的键的过期时间，我们就能保证缓存和数据库的数据最终是一致的。因为只要缓存数据过期了，就会被删除。随后读的时候，因为缓存里没有，就可以查数据库的数据，然后将数据库查出来的数据写入到缓存中。除了设置过期时间，我们还需要做更多的措施来尽量避免数据库与缓存处于不一致的情况发生。
2. 新增、更改、删除数据库操作时同步更新 Redis，可以使用事物机制来保证数据的一致性。

**9、Redis 持久化有几种方式？**

持久化就是把内存的数据写到磁盘中去，防止服务宕机了内存数据丢失。Redis 提供了两种持久化方式：RDB（默认） 和 AOF。

**RDB**

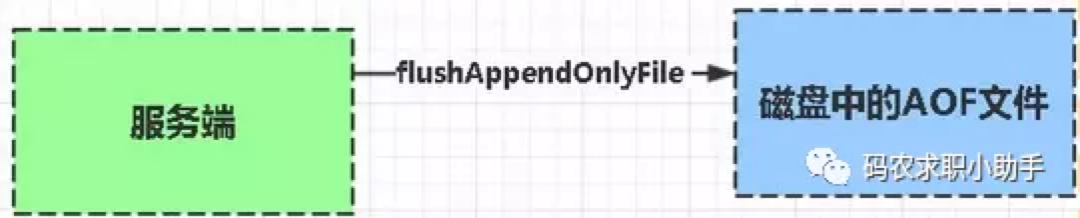
RDB 是 Redis DataBase 的缩写。按照一定的时间周期策略把内存的数据以快照的形式保存到硬盘的二进制文件。即 Snapshot 快照存储，对应产生的数据文件为 dump.rdb，通过配置文件中的 save 参数来定义快照的周期。核心函数：rdbSave（生成 RDB 文件）和 rdbLoad（从文件加载内存）两个函数。



**AOF**

AOF 是 Append-only file 的缩写。Redis会将每一个收到的写命令都通过 Write 函数追加到文件最后，类似于 MySQL 的 binlog。当 Redis 重启是会通过重新执行文件中保存的写命令来在内存中重建整个数据库的内容。每当执行服务器（定时）任务或者函数时，flushAppendOnlyFile 函数都会被调用， 这个函数执行以下两个工作：

* WRITE：根据条件，将 aof\_buf 中的缓存写入到 AOF 文件；
* SAVE：根据条件，调用 fsync 或 fdatasync 函数，将 AOF 文件保存到磁盘中。



**RDB 和 AOF 的区别：**

1. AOF 文件比 RDB 更新频率高，优先使用 AOF 还原数据；
2. AOF比 RDB 更安全也更大；
3. RDB 性能比 AOF 好；
4. 如果两个都配了优先加载 AOF。

**10、Redis 怎么实现分布式锁？**

Redis 为单线程模式，采用队列模式将并发访问变成串行访问，且多客户端对 Redis 的连接并不存在竞争关系。Redis 中可以使用 SETNX 命令实现分布式锁。一般使用 setnx(set if not exists) 指令，只允许被一个程序占有，使用完调用 del 释放锁。

**11、Redis 内存淘汰策略有哪些？**

1. volatile-lru：从已设置过期时间的数据集（server. db[i]. expires）中挑选最近最少使用的数据淘汰；
2. volatile-ttl：从已设置过期时间的数据集（server. db[i]. expires）中挑选将要过期的数据淘汰。
3. volatile-random：从已设置过期时间的数据集（server. db[i]. expires）中任意选择数据淘汰。
4. allkeys-lru：从数据集（server. db[i]. dict）中挑选最近最少使用的数据淘汰。
5. allkeys-random：从数据集（server. db[i]. dict）中任意选择数据淘汰。
6. no-enviction（驱逐）：禁止驱逐数据。

**12、Redis 常见性能问题和解决方案？**

1. Master 最好不要做任何持久化工作，如 RDB 内存快照和 AOF 日志文件。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；
2. 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内；
3. 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…

**13、Redis的过期键的删除策略**

我们都知道，Redis是key-value数据库，我们可以设置Redis中缓存的key的过期时间。Redis的过期策略就是指当Redis中缓存的key过期了，Redis如何处理。

过期策略通常有以下三种：

* 定时过期：每个设置过期时间的key都需要创建一个定时器，到过期时间就会立即清除。该策略可以立即清除过期的数据，对内存很友好；但是会占用大量的CPU资源去处理过期的数据，从而影响缓存的响应时间和吞吐量。
* 惰性过期：只有当访问一个key时，才会判断该key是否已过期，过期则清除。该策略可以最大化地节省CPU资源，却对内存非常不友好。极端情况可能出现大量的过期key没有再次被访问，从而不会被清除，占用大量内存。
* 定期过期：每隔一定的时间，会扫描一定数量的数据库的expires字典中一定数量的key，并清除其中已过期的key。该策略是前两者的一个折中方案。通过调整定时扫描的时间间隔和每次扫描的限定耗时，可以在不同情况下使得CPU和内存资源达到最优的平衡效果。  
  (expires字典会保存所有设置了过期时间的key的过期时间数据，其中，key是指向键空间中的某个键的指针，value是该键的毫秒精度的UNIX时间戳表示的过期时间。键空间是指该Redis集群中保存的所有键。)

Redis中同时使用了惰性过期和定期过期两种过期策略。

**14、我们知道通过expire来设置key 的过期时间，那么对过期的数据怎么处理呢?**

除了缓存服务器自带的缓存失效策略之外（Redis默认的有6中策略可供选择），我们还可以根据具体的业务需求进行自定义的缓存淘汰，常见的策略有两种：

1. 定时去清理过期的缓存；
2. 当有用户请求过来时，再判断这个请求所用到的缓存是否过期，过期的话就去底层系统得到新数据并更新缓存。

两者各有优劣，第一种的缺点是维护大量缓存的key是比较麻烦的，第二种的缺点就是每次用户请求过来都要判断缓存失效，逻辑相对比较复杂！具体用哪种方案，大家可以根据自己的应用场景来权衡。

**15、Hash 冲突怎么办？**

Redis 通过**链式哈希**解决冲突：**也就是同一个 桶里面的元素使用链表保存**。但是当链表过长就会导致查找性能变差可能，所以 Redis 为了追求快，使用了两个全局哈希表。用于 rehash 操作，增加现有的哈希桶数量，减少哈希冲突。

开始默认使用 「hash 表 1 」保存键值对数据，「hash 表 2」 此刻没有分配空间。当数据越来越多触发 rehash 操作，则执行以下操作：

1. 给 「hash 表 2 」分配更大的空间；
2. 将 「hash 表 1 」的数据重新映射拷贝到 「hash 表 2」 中；
3. 释放 「hash 表 1」 的空间。

**值得注意的是，将 hash 表 1 的数据重新映射到 hash 表 2 的过程中并不是一次性的，这样会造成 Redis 阻塞，无法提供服务。**

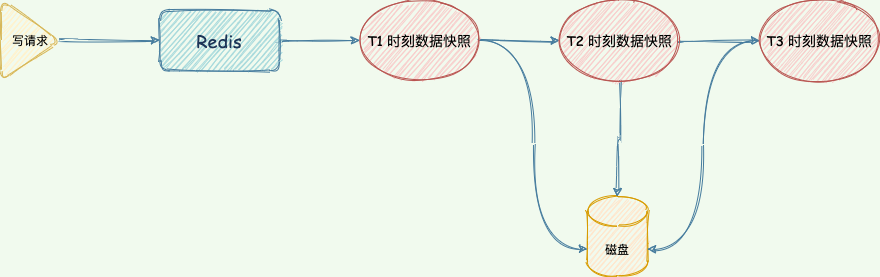
而是采用了**渐进式 rehash**，每次处理客户端请求的时候，先从「 hash 表 1」 中第一个索引开始，将这个位置的 所有数据拷贝到 「hash 表 2」 中，就这样将 rehash 分散到多次请求过程中，避免耗时阻塞。

**16、什么是 RDB 内存快照？**

在 Redis 执行「写」指令过程中，内存数据会一直变化。所谓的内存快照，指的就是 Redis 内存中的数据在某一刻的状态数据。

好比时间定格在某一刻，当我们拍照的，通过照片就能把某一刻的瞬间画面完全记录下来。

Redis 跟这个类似，就是把某一刻的数据以文件的形式拍下来，写到磁盘上。这个快照文件叫做 **RDB 文件，RDB 就是 Redis DataBase 的缩写。**



在做数据恢复时，直接将 RDB 文件读入内存完成恢复。

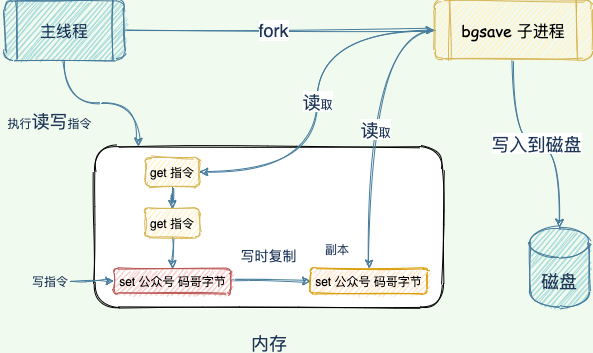
**17、在生成 RDB 期间，Redis 可以同时处理写请求么？**

可以的，Redis 使用操作系统的多进程**写时复制技术 COW(Copy On Write)**来实现快照持久化，保证数据一致性。

Redis 在持久化时会调用 glibc 的函数fork产生一个子进程，快照持久化完全交给子进程来处理，父进程继续处理客户端请求。

当主线程执行写指令修改数据的时候，这个数据就会复制一份副本， bgsave子进程读取这个副本数据写到 RDB 文件。

这既保证了快照的完整性，也允许主线程同时对数据进行修改，避免了对正常业务的影响。



**18、如何实现数据尽可能少丢失又能兼顾性能呢？**

重启 Redis 时，我们很少使用 rdb 来恢复内存状态，因为会丢失大量数据。我们通常使用 AOF 日志重放，但是重放 AOF 日志性能相对 rdb 来说要慢很多，这样在 Redis 实例很大的情况下，启动需要花费很长的时间。

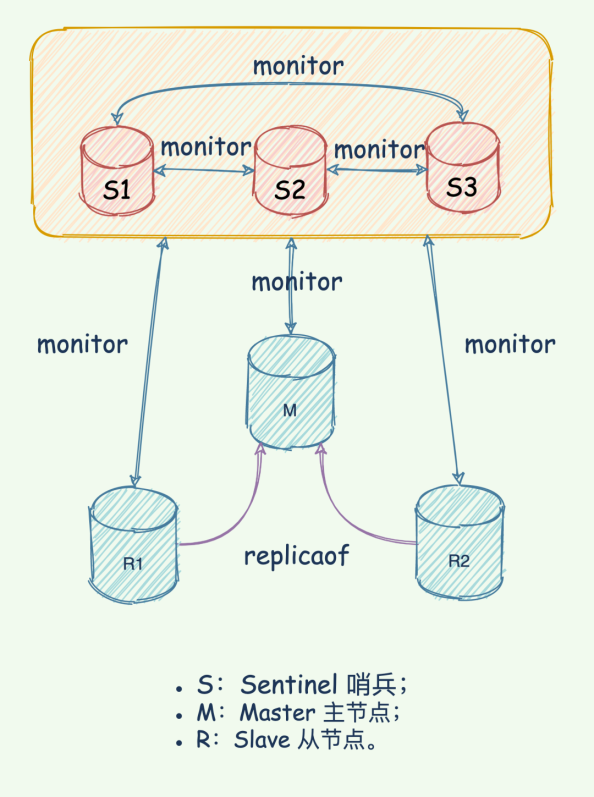
Redis 4.0 为了解决这个问题，带来了一个新的持久化选项——**混合持久化**。将 rdb 文件的内容和增量的 AOF 日志文件存在一起。这里的 AOF 日志不再是全量的日志，而是**自持久化开始到持久化结束的这段时间发生的增量 AOF 日志**，通常这部分 AOF 日志很小。

于是**在 Redis 重启的时候，可以先加载 rdb 的内容，然后再重放增量 AOF 日志就可以完全替代之前的 AOF 全量文件重放，重启效率因此大幅得到提升**。

**19、你知道 哨兵集群原理么？**

哨兵是 Redis 的一种运行模式，它专注于**对 Redis 实例（主节点、从节点）运行状态的监控，并能够在主节点发生故障时通过一系列的机制实现选主及主从切换，实现故障转移，确保整个 Redis 系统的可用性**。

他的架构图如下：



Redis 哨兵具备的能力有如下几个：

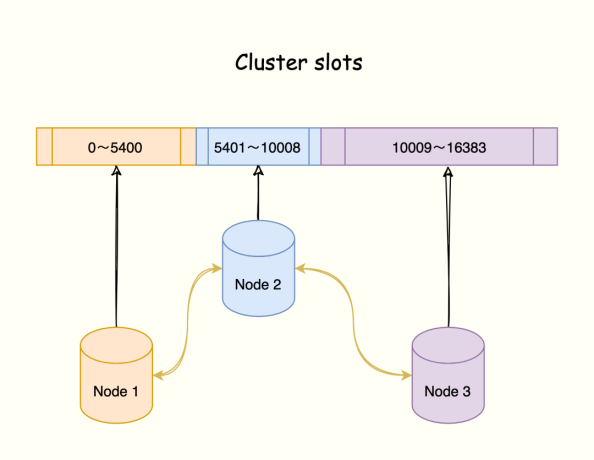
* **监控**：持续监控 master 、slave 是否处于预期工作状态。
* **自动切换主库**：当 Master 运行故障，哨兵启动自动故障恢复流程：从 slave 中选择一台作为新 master。
* **通知**：让 slave 执行 replicaof ，与新的 master 同步；并且通知客户端与新 master 建立连接。

**20、什么是 Cluster 集群？**

Redis 集群是一种分布式数据库方案，集群通过分片（sharding）来进行数据管理（「分治思想」的一种实践），并提供复制和故障转移功能。

将数据划分为 16384 的 slots，每个节点负责一部分槽位。槽位的信息存储于每个节点中。

它是去中心化的，如图所示，该集群由三个 Redis 节点组成，每个节点负责整个集群的一部分数据，每个节点负责的数据多少可能不一样。



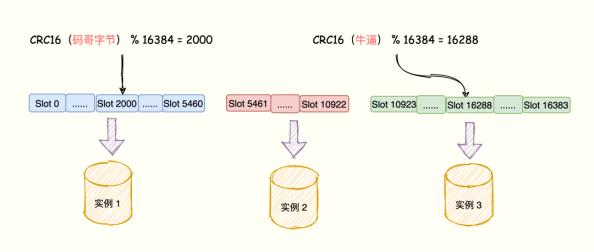
三个节点相互连接组成一个对等的集群，它们之间通过 Gossip协议相互交互集群信息，最后每个节点都保存着其他节点的 slots 分配情况。

**使用 Redis Cluster 集群，主要解决了大数据量存储导致的各种慢问题**

**21、哈希槽又是如何映射到 Redis 实例上呢？**

1. 根据键值对的 key，使用 CRC16 算法，计算出一个 16 bit 的值；
2. 将 16 bit 的值对 16384 执行取模，得到 0 ～ 16383 的数表示 key 对应的哈希槽。
3. 根据该槽信息定位到对应的实例。

键值对数据、哈希槽、Redis 实例之间的映射关系如下：



**22、Cluster 如何实现故障转移？**

Redis 集群节点采用 Gossip 协议来广播自己的状态以及自己对整个集群认知的改变。比如一个节点发现某个节点失联了 (PFail)，它会将这条信息向整个集群广播，其它节点也就可以收到这点失联信息。

如果一个节点收到了某个节点失联的数量 (PFail Count) 已经达到了集群的大多数，就可以标记该节点为确定下线状态 (Fail)，然后向整个集群广播，强迫其它节点也接收该节点已经下线的事实，并立即对该失联节点进行主从切换。

**23、Redis如何做内存优化？**

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可以用更紧凑的方式存放到一起。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

**24、Redis线程模型**

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（file event handler）。它的组成结构为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队列的消费是单线程的，所以Redis才叫单线程模型。

* 文件事件处理器使用 I/O 多路复用（multiplexing）程序来同时监听多个套接字， 并根据套接字目前执行的任务来为套接字关联不同的事件处理器。
* 当被监听的套接字准备好执行连接应答（accept）、读取（read）、写入（write）、关闭（close）等操作时， 与操作相对应的文件事件就会产生， 这时文件事件处理器就会调用套接字之前关联好的事件处理器来处理这些事件。

虽然文件事件处理器以单线程方式运行， 但通过使用 I/O 多路复用程序来监听多个套接字， 文件事件处理器既实现了高性能的网络通信模型， 又可以很好地与 redis 服务器中其他同样以单线程方式运行的模块进行对接， 这保持了 Redis 内部单线程设计的简单性。

**25、Redis事务及其相关面试题**

**什么是事务？**

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。

**Redis事务的概念**

Redis 事务的本质是通过MULTI、EXEC、WATCH等一组命令的集合。事务支持一次执行多个命令，一个事务中所有命令都会被序列化。在事务执行过程，会按照顺序串行化执行队列中的命令，其他客户端提交的命令请求不会插入到事务执行命令序列中。

总结说：redis事务就是一次性、顺序性、排他性的执行一个队列中的一系列命令。

**Redis事务的三个阶段**

1. 事务开始 MULTI
2. 命令入队
3. 事务执行 EXEC

事务执行过程中，如果服务端收到有EXEC、DISCARD、WATCH、MULTI之外的请求，将会把请求放入队列中排

**事务管理（ACID）概述**

* 原子性（Atomicity）

原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

* 一致性（Consistency）

事务前后数据的完整性必须保持一致。

* 隔离性（Isolation）

多个事务并发执行时，一个事务的执行不应影响其他事务的执行

* 持久性（Durability）

持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响

Redis的事务总是具有ACID中的一致性和隔离性，其他特性是不支持的。当服务器运行在*AOF*持久化模式下，并且appendfsync选项的值为always时，事务也具有耐久性。

**Redis事务支持隔离性吗**

Redis 是单进程程序，并且它保证在执行事务时，不会对事务进行中断，事务可以运行直到执行完所有事务队列中的命令为止。因此，Redis 的事务是总是带有隔离性的。

**Redis事务保证原子性吗，支持回滚吗**

Redis中，单条命令是原子性执行的，但事务不保证原子性，且没有回滚。事务中任意命令执行失败，其余的命令仍会被执行。

**Redis事务其他实现**

* 基于Lua脚本，Redis可以保证脚本内的命令一次性、按顺序地执行，  
  其同时也不提供事务运行错误的回滚，执行过程中如果部分命令运行错误，剩下的命令还是会继续运行完
* 基于中间标记变量，通过另外的标记变量来标识事务是否执行完成，读取数据时先读取该标记变量判断是否事务执行完成。但这样会需要额外写代码实现，比较繁琐

**26、Redis是单线程的，如何提高多核CPU的利用率？**

可以在同一个服务器部署多个Redis的实例，并把他们当作不同的服务器来使用，在某些时候，无论如何一个服务器是不够的， 所以，如果你想使用多个CPU，你可以考虑一下分片（shard）。

**27、为什么要做Redis分区？**

分区可以让Redis管理更大的内存，Redis将可以使用所有机器的内存。如果没有分区，你最多只能使用一台机器的内存。分区使Redis的计算能力通过简单地增加计算机得到成倍提升，Redis的网络带宽也会随着计算机和网卡的增加而成倍增长。

**28、你知道有哪些Redis分区实现方案？**

* 客户端分区就是在客户端就已经决定数据会被存储到哪个redis节点或者从哪个redis节点读取。大多数客户端已经实现了客户端分区。
* 代理分区 意味着客户端将请求发送给代理，然后代理决定去哪个节点写数据或者读数据。代理根据分区规则决定请求哪些Redis实例，然后根据Redis的响应结果返回给客户端。redis和memcached的一种代理实现就是Twemproxy
* 查询路由(Query routing) 的意思是客户端随机地请求任意一个redis实例，然后由Redis将请求转发给正确的Redis节点。Redis Cluster实现了一种混合形式的查询路由，但并不是直接将请求从一个redis节点转发到另一个redis节点，而是在客户端的帮助下直接redirected到正确的redis节点。

**29、Redis分区有什么缺点？**

* 涉及多个key的操作通常不会被支持。例如你不能对两个集合求交集，因为他们可能被存储到不同的Redis实例（实际上这种情况也有办法，但是不能直接使用交集指令）。
* 同时操作多个key,则不能使用Redis事务.
* 分区使用的粒度是key，不能使用一个非常长的排序key存储一个数据集（The partitioning granularity is the key, so it is not possible to shard a dataset with a single huge key like a very big sorted set）
* 当使用分区的时候，数据处理会非常复杂，例如为了备份你必须从不同的Redis实例和主机同时收集RDB / AOF文件。
* 分区时动态扩容或缩容可能非常复杂。Redis集群在运行时增加或者删除Redis节点，能做到最大程度对用户透明地数据再平衡，但其他一些客户端分区或者代理分区方法则不支持这种特性。然而，有一种预分片的技术也可以较好的解决这个问题。

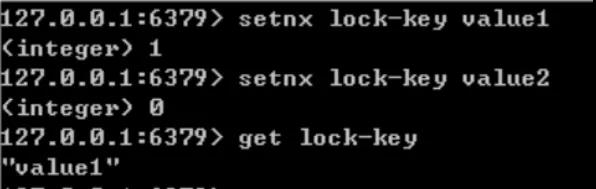
**30、Redis实现分布式锁**

Redis为单进程单线程模式，采用队列模式将并发访问变成串行访问，且多客户端对Redis的连接并不存在竞争关系Redis中可以使用SETNX命令实现分布式锁。

当且仅当 key 不存在，将 key 的值设为 value。若给定的 key 已经存在，则 SETNX 不做任何动作

SETNX 是『SET if Not eXists』(如果不存在，则 SET)的简写。

返回值：设置成功，返回 1 。设置失败，返回 0 。



使用SETNX完成同步锁的流程及事项如下：

使用SETNX命令获取锁，若返回0（key已存在，锁已存在）则获取失败，反之获取成功

为了防止获取锁后程序出现异常，导致其他线程/进程调用SETNX命令总是返回0而进入死锁状态，需要为该key设置一个“合理”的过期时间

释放锁，使用DEL命令将锁数据删除

**31、如何解决 Redis 的并发竞争 Key 问题**

所谓 Redis 的并发竞争 Key 的问题也就是多个系统同时对一个 key 进行操作，但是最后执行的顺序和我们期望的顺序不同，这样也就导致了结果的不同！

推荐一种方案：分布式锁（zookeeper 和 redis 都可以实现分布式锁）。（如果不存在 Redis 的并发竞争 Key 问题，不要使用分布式锁，这样会影响性能）

基于zookeeper临时有序节点可以实现的分布式锁。大致思想为：每个客户端对某个方法加锁时，在zookeeper上的与该方法对应的指定节点的目录下，生成一个唯一的瞬时有序节点。判断是否获取锁的方式很简单，只需要判断有序节点中序号最小的一个。当释放锁的时候，只需将这个瞬时节点删除即可。同时，其可以避免服务宕机导致的锁无法释放，而产生的死锁问题。完成业务流程后，删除对应的子节点释放锁。

在实践中，当然是从以可靠性为主。所以首推Zookeeper。

参考：https://www.jianshu.com/p/8bddd381de06

**32、分布式Redis是前期做还是后期规模上来了再做好？为什么？**

既然Redis是如此的轻量（单实例只使用1M内存），为防止以后的扩容，最好的办法就是一开始就启动较多实例。即便你只有一台服务器，你也可以一开始就让Redis以分布式的方式运行，使用分区，在同一台服务器上启动多个实例。

一开始就多设置几个Redis实例，例如32或者64个实例，对大多数用户来说这操作起来可能比较麻烦，但是从长久来看做这点牺牲是值得的。

这样的话，当你的数据不断增长，需要更多的Redis服务器时，你需要做的就是仅仅将Redis实例从一台服务迁移到另外一台服务器而已（而不用考虑重新分区的问题）。一旦你添加了另一台服务器，你需要将你一半的Redis实例从第一台机器迁移到第二台机器。

**33、什么是 RedLock**

Redis 官方站提出了一种权威的基于 Redis 实现分布式锁的方式名叫 *Redlock*，此种方式比原先的单节点的方法更安全。它可以保证以下特性：

1. 安全特性：互斥访问，即永远只有一个 client 能拿到锁
2. 避免死锁：最终 client 都可能拿到锁，不会出现死锁的情况，即使原本锁住某资源的 client crash 了或者出现了网络分区
3. 容错性：只要大部分 Redis 节点存活就可以正常提供服务

Redis 面试题网站：www.iamshuaidi.com/1864.html