Redis3-持久化+集群模式

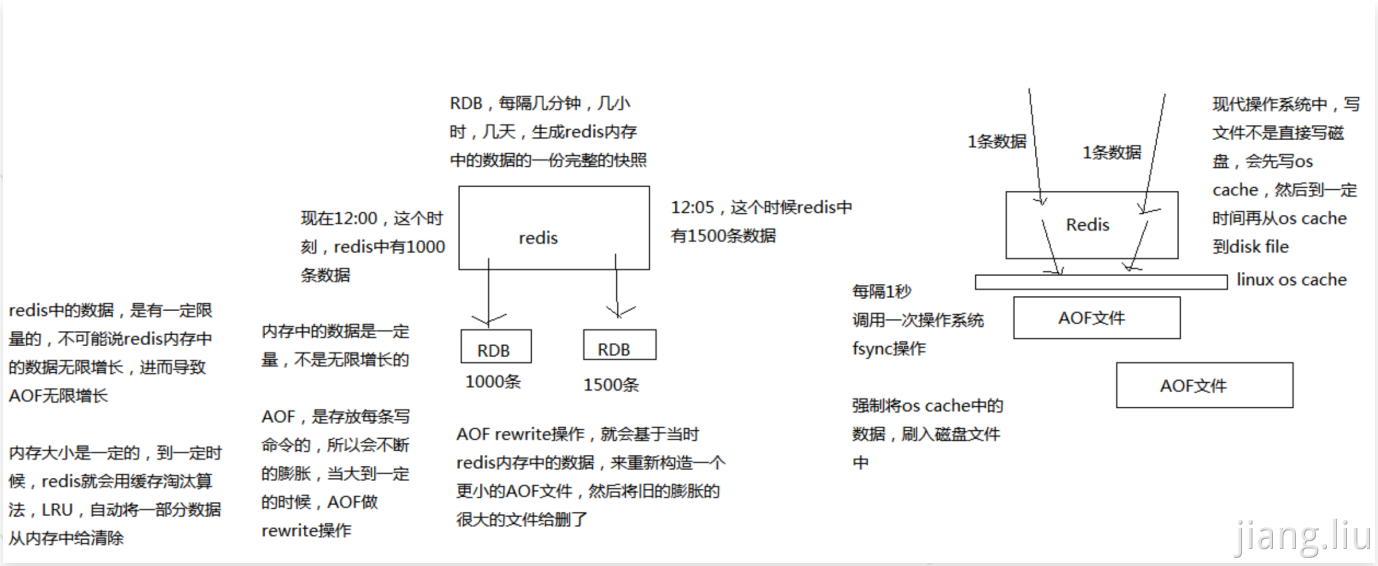
11.Redis持久化有哪几种方式？优缺点？底层实现？

redis持久化的意义，在于故障恢复。
比如部署了一个redis，作为缓存服务，当然也可以保存一些较为重要的数据。如果没有持久化的话，redis遇到灾难性故障的时候，就会丢失所有的数据。如果通过持久化将数据写到磁盘上了，然后定期同步和备份到一些云存储服务上去，这样就可以保证数据不丢失全部，并且还可以恢复一部分数据回来。

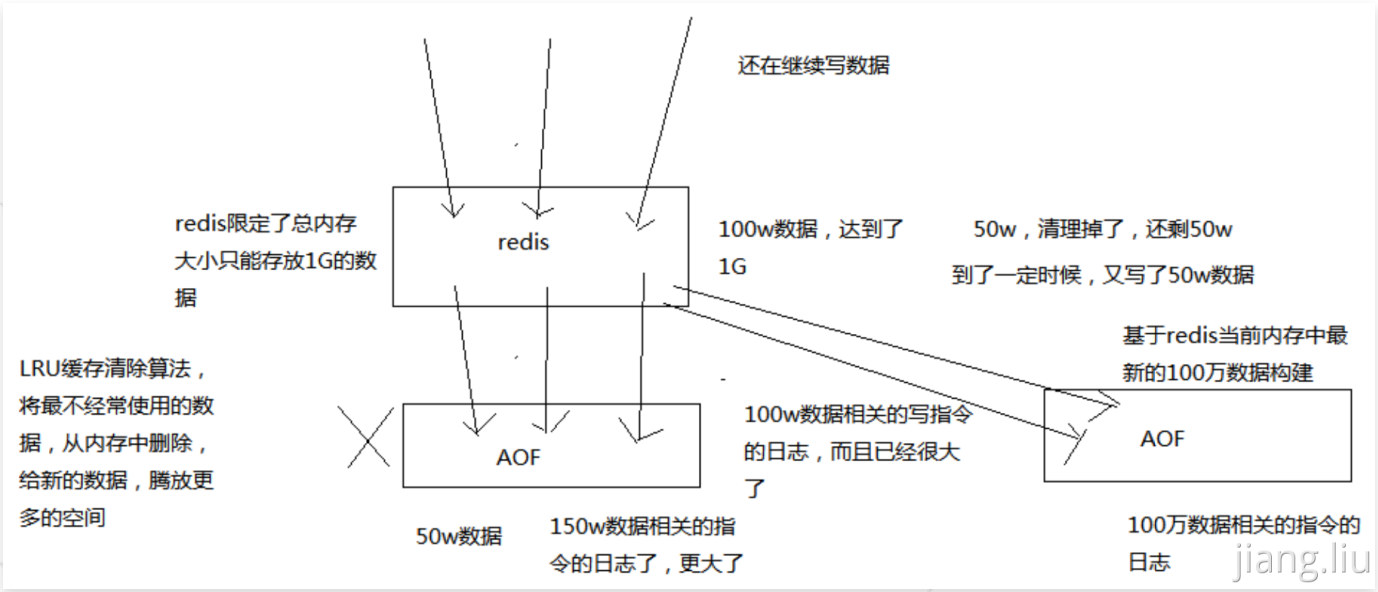
企业级redis集群架构：海量数据、高并发、高可用

RDB和AOF的持久化机制

- RDB：对redis中的数据执行周期性的持久化。Redis默认持久化方式为RDB。
- AOF：对每条写入命令作为日志，以append-only的模式写入一个日志文件中，在redis重启的时候，可以通过回放AOF日志中的写入指令来重新构建整个数据集。



AOF rewrite流程：



如果我们想要redis仅仅作为纯内存的缓存来用，那么可以禁止RDB和AOF所有的持久化机制。

> RDB持久化机制的工作流程
>
> （1）redis根据配置自己尝试去生成rdb快照文件
>
> （2）fork一个子进程出来
>
> （3）子进程尝试将数据dump到临时的rdb快照文件中
>
> （4）完成rdb快照文件的生成之后，就替换之前的旧的快照文件，dump.rdb，每次生成一个新的快照，都会覆盖之前的老快照。

通过RDB或AOF，都可以将redis内存中的数据给持久化到磁盘上面来，然后可以将这些数据备份到别的地方去，比如说阿里云，云服务。
如果redis挂了，服务器上的内存和磁盘上的数据都丢了，可以从云服务上拷贝回来之前的数据，放到指定的目录中，然后重新启动redis，redis就会自动根据持久化数据文件中的数据，去恢复内存中的数据，继续对外提供服务。

如果同时使用RDB和AOF两种持久化机制，那么在redis重启的时候，会使用AOF来重新构建数据，因为AOF中的数据更加完整。

RDB持久化机制的优点

1）RDB会生成多个数据文件，每个数据文件都代表了某一时刻redis中的数据，这种多个数据文件的方式，非常适合做冷备，可以将这种完整的数据文件发送到一些远程的安全存储上去，比如说Amazon的S3云服务上去，在国内可以是阿里云的ODPS分布式存储上，以预定好的备份策略来定期备份redis中的数据。

> RDB可以做冷备，生成多个文件，每个文件都代表了某一个时刻的完整的数据快照。
> AOF也可以做冷备，只有一个文件，但是你可以，每隔一定时间，去copy一份这个文件出来。
>
> RDB做冷备，优势在哪儿呢？由redis去控制固定时长生成快照文件的事情，比较方便; AOF，还需要自己写一些脚本去做这个事情，各种定时
> RDB数据做冷备，在最坏的情况下，提供数据恢复的时候，速度比AOF快

2）RDB对redis对外提供的读写服务，影响非常小，可以让redis保持高性能，因为redis主进程只需要fork一个子进程，让子进程执行磁盘IO操作来进行RDB持久化即可。

> RDB，每次写，都是直接写redis内存，只是在一定的时候，才会将数据写入磁盘中。
> AOF，每次都是要写文件的，虽然可以快速写入os cache中，但是还是有一定的时间开销的,速度肯定比RDB略慢一些。

3）相对于AOF持久化机制来说，直接基于RDB数据文件来重启和恢复redis进程，更加快速。

> AOF，存放的指令日志，做数据恢复的时候，其实是要回放和执行所有的指令日志，来恢复出来内存中的所有数据的。
> RDB，就是一份数据文件，恢复的时候，直接加载到内存中即可。

RDB持久化机制的缺点

1）如果想要在redis故障时，尽可能少的丢失数据，那么RDB没有AOF好。一般来说，RDB数据快照文件，都是每隔5分钟，或者更长时间生成一次，这个时候就得接受一旦redis进程宕机，那么会丢失最近5分钟的数据

> 这个问题，也是rdb最大的缺点，就是不适合做第一优先的恢复方案，如果你依赖RDB做第一优先恢复方案，会导致数据丢失的比较多。

2）RDB每次在fork子进程来执行RDB快照数据文件生成的时候，如果数据文件特别大，可能会导致对客户端提供的服务暂停数毫秒，或者甚至数秒

> 一般不要让RDB的间隔太长，否则每次生成的RDB文件太大了，对redis本身的性能可能会有影响的

AOF持久化机制的优点

1）AOF可以更好的保护数据不丢失，一般AOF会先将**写命令**先写入**磁盘缓存**中，然后在同步到磁盘上，这同步操作会每隔1秒，通过一个后台线程执行一次fsync操作，最多丢失1秒钟的数据。

> 每隔1秒，执行一次fsync操作，保证os cache中的数据写入磁盘中。redis进程挂了，最多丢掉1秒钟的数据

2）AOF日志文件以append-only模式写入，所以没有任何磁盘寻址的开销，写入性能非常高，而且文件不容易破损，即使文件尾部破损，也很容易修复。

3）AOF日志文件即使过大的时候，出现后台rewrite重写操作，也不会影响客户端的读写。因为在rewrite log的时候，会对其中的指令进行压缩，创建出一份需要恢复数据的最小日志出来。再创建新日志文件的时候，老的日志文件还是照常写入。当新的merge后的日志文件ready的时候，再交换新老日志文件即可。

4）AOF日志文件的命令通过非常可读的方式进行记录，这个特性非常适合做灾难性的误删除的紧急恢复。比如某人不小心用flushall命令清空了所有数据，只要这个时候后台rewrite还没有发生，那么就可以立即拷贝AOF文件，将最后一条flushall命令给删了，然后再将该AOF文件放回去，就可以通过恢复机制，自动恢复所有数据。

AOF持久化机制的缺点

1）对于同一份数据来说，AOF日志文件通常比RDB数据快照文件更大。
2）AOF开启后，支持的写QPS会比RDB支持的写QPS低，因为AOF一般会配置成每秒fsync一次日志文件，当然，每秒一次fsync，性能也还是很高的。

3）以前AOF发生过bug，就是通过AOF记录的日志，进行数据恢复的时候，没有恢复一模一样的数据出来。所以说，类似AOF这种较为复杂的基于命令日志/merge/回放的方式，比基于RDB每次持久化一份完整的数据快照文件的方式，更加脆弱一些，容易有bug。不过AOF就是为了避免rewrite过程导致的bug，因此每次rewrite并不是基于旧的指令日志进行merge的，而是基于当时内存中的数据进行指令的重新构建，这样健壮性会好很多。
4）唯一的比较大的缺点，其实就是做数据恢复的时候，会比较慢，还有做冷备，定期的备份，不太方便，可能要自己手写复杂的脚本去做，做冷备不太合适。

RDB和AOF到底该如何选择

（1）不要仅仅使用RDB，因为那样会导致你丢失很多数据
（2）也不要仅仅使用AOF，因为那样有两个问题，第一，你通过AOF做冷备，没有RDB做冷备，来的恢复速度更快; 第二，RDB每次简单粗暴生成数据快照，更加健壮，可以避免AOF这种复杂的备份和恢复机制的bug。
（3）综合使用AOF和RDB两种持久化机制，用AOF来保证数据不丢失，作为数据恢复的第一选择; 用RDB来做不同程度的冷备，在AOF文件都丢失或损坏不可用的时候，还可以使用RDB来进行快速的数据恢复。

12. Redis cluster集群模式

redis cluster：Redis集群模式，支撑N个redis master节点，每个master node都可以挂载多个slave node。

读写分离：对于每个master来说，写就写到master，然后读就从mater对应的slave去读。
高可用性：每个master都有salve节点，那么如果mater挂掉，redis cluster这套机制，就会自动将某个slave切换成master。

> redis cluster（多master + 读写分离 + 高可用）

我们只要基于redis cluster去搭建redis集群即可，不需要手工去搭建 主从+哨兵 实现高可用。

redis cluster 对比 **replication + sentinal**（主从+哨兵）
如果你的数据量很少，主要是承载高并发高性能的场景，比如你的缓存一般就几个G，单机足够了。
replication，一个mater，多个slave，要几个slave跟你的要求的读吞吐量有关系，然后自己搭建一个sentinal集群，去保证redis主从架构的高可用性，就可以了。
redis cluster主要是针对海量数据+高并发+高可用的场景，海量数据，如果你的数据量很大，那么建议就用redis cluster。

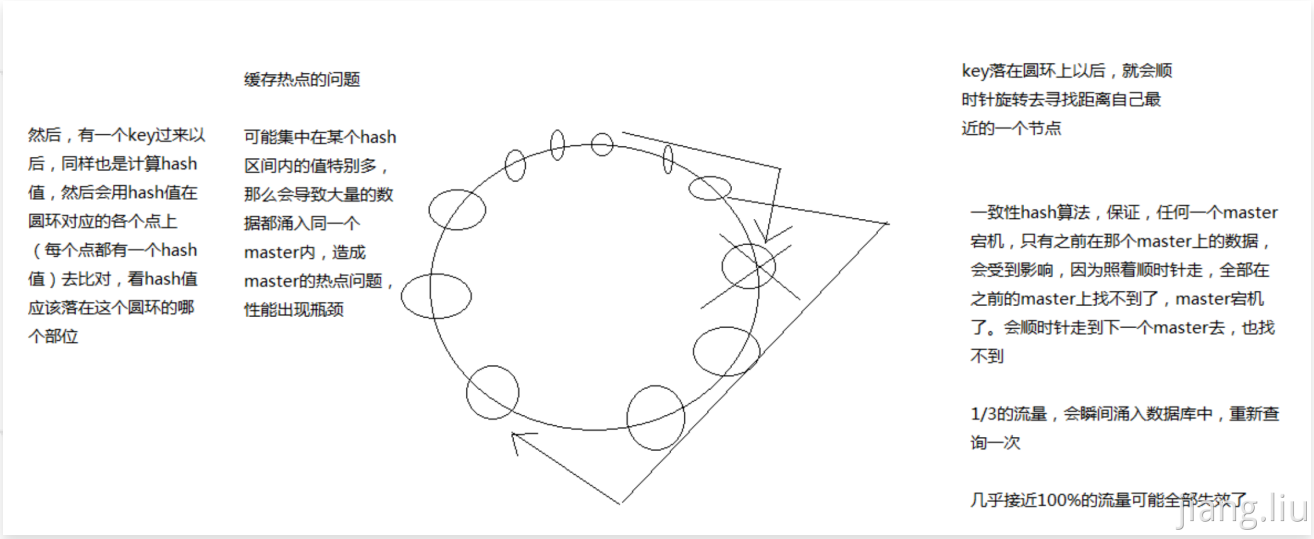
13.hash+一致性hash+redis cluster的hash slot

分布式数据存储的核心算法:

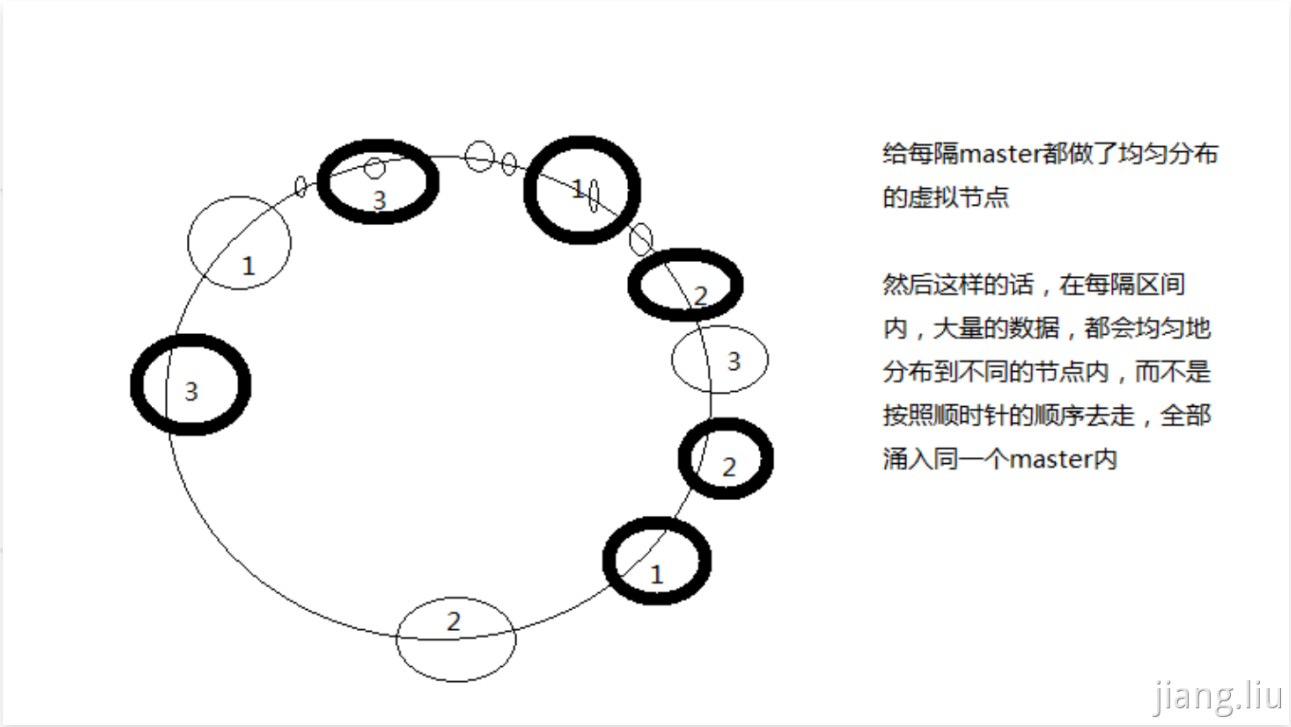
hash算法 -> 一致性hash算法（memcached） -> redis cluster，hash slot算法

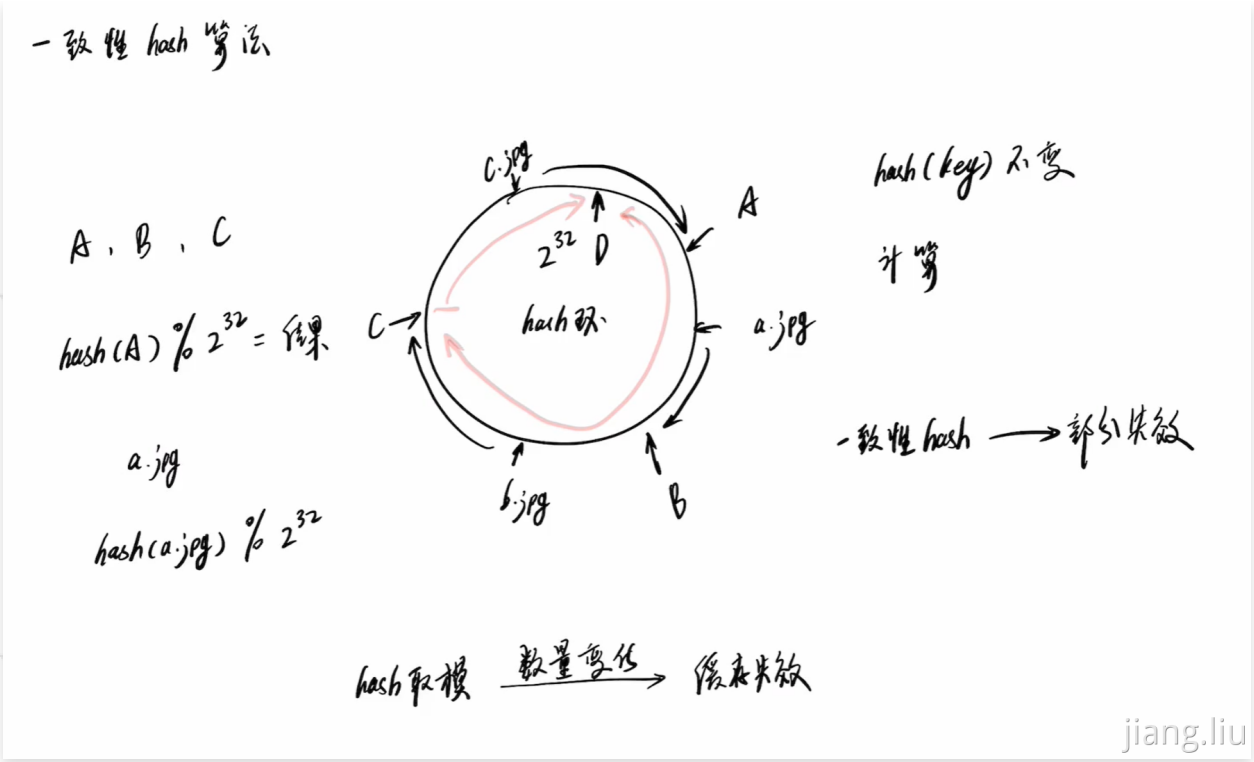
一致性hash算法（自动缓存迁移）+虚拟节点（自动负载均衡）

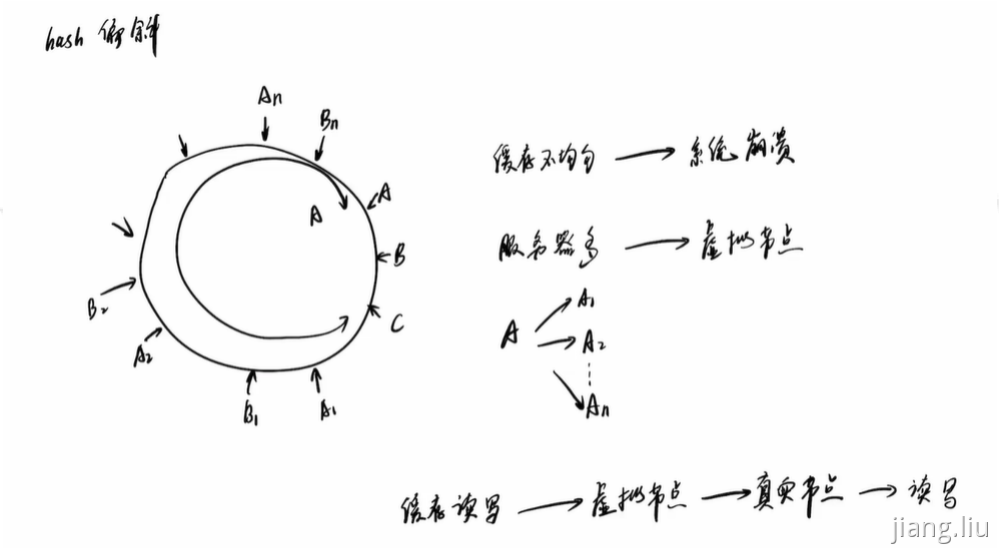
一致性hash算法:



一致性hash算法虚拟节点实现负载均衡:







redis cluster

（1）自动将数据进行分片，每个master上放一部分数据。

（2）提供内置的高可用支持，部分master不可用时，还是可以继续工作的。

在redis cluster架构下，每个redis要放开两个端口号，比如一个是6379的话，另外一个就是6379+10000的端口号，也就是16379，16379端口号是用来进行节点间通信的，也就是cluster bus的东西，集群总线。cluster bus的通信，用来进行故障检测，配置更新，故障转移授权。

cluster bus用了另外一种二进制的协议，主要用于节点间进行高效的数据交换，占用更少的网络带宽和处理时间。

redis cluster的hash slot算法

redis cluster有固定的16384个hash slot（哈希槽），对每个key计算CRC16值，然后对16384取模，可以获取key对应的hash slot。redis cluster中每个master都会持有部分slot，比如有3个master，那么可能每个master持有5000多个hash slot。

> slot=CRC16（key）/16384

hash slot让node的增加和移除很简单，增加一个master，就将其他master的hash slot移动部分过去，减少一个master，就将它的hash slot移动到其他master上去。Redis底层机制保证移动hash slot的成本是非常低的。

客户端的api，可以对指定的数据，让他们走同一个hash slot，通过hash tag来实现。

14. redis cluster的核心原理分析：gossip通信、jedis smart定位、主备切换

节点间的内部通信机制

1、基础通信原理

1）redis cluster节点间采取gossip协议进行通信

跟集中式不同，不是将集群元数据（节点信息，故障，等等）集中存储在某个节点上，而是互相之间不断通信，保持整个集群所有节点的数据是完整的。

> 维护集群的元数据的两种方式：集中式、gossip。
> zookeeper就是采用集中式存储。 大数据领域的storm（分布式大数据实时计算引擎）底层就是基于zookeeper进行元数据的维护。

集中式：好处在于，元数据的更新和读取，时效性非常好，一旦元数据出现了变更，立即就更新到集中式的存储中，其他节点读取的时候立即就可以感知到；缺点在于，所有的元数据的跟新压力全部集中在一个地方，可能会导致元数据的存储有压力。

gossip：好处在于，元数据的更新比较分散，不是集中在一个地方，更新请求会陆陆续续，打到所有节点上去更新，有一定的延时，降低了压力；缺点，元数据更新有延时，可能导致集群的一些操作会有一些滞后。

2）10000端口

每个节点都有一个专门用于节点间通信的端口，就是自己提供服务的端口号+10000，比如7001，那么用于节点间通信的就是17001端口。每个节点每隔一段时间都会往另外几个节点发送ping消息，同时其他几点接收到ping之后返回pong。

3）交换的信息

故障信息，节点的增加和移除，hash slot信息，等等。

2、gossip协议

gossip协议包含多种消息，包括ping，pong，meet，fail，等等。

- **meet**: 某个节点发送meet给新加入的节点，让新节点加入集群中，然后新节点就会开始与其他节点进行通信。发送 `redis-trib.rb add-node` 命令，其实内部就是发送了一个gossip meet消息，给新加入的节点，通知那个节点去加入我们的集群。
- **ping**: 每个节点每秒都会频繁给其他节点发送ping，其中包含自己的状态还有自己维护的集群元数据，互相通过ping交换元数据，互相进行元数据的更新。
- **pong**: 返回ping和meet，包含自己的状态和其他信息，也可以用于信息广播和更新。
- **fail**: 某个节点判断另一个节点fail之后，就发送fail给其他节点，通知其他节点，指定的节点宕机了。

3、ping消息深入

ping很频繁，而且要携带一些元数据，所以可能会加重网络负担。每个节点每秒会执行10次ping，每次会选择5个最久没有通信的其他节点。当然如果发现某个节点通信延时达到了cluster_node_timeout / 2，那么立即发送ping，避免数据交换延时过长，落后的时间太长了。比如说，两个节点之间都10分钟没有交换数据了，那么整个集群处于严重的元数据不一致的情况，就会有问题。所以cluster_node_timeout可以调节，如果调节比较大，那么会降低发送的频率。

每次ping，一个是带上自己节点的信息，还有就是带上1/10其他节点的信息，发送出去，进行数据交换。至少包含3个其他节点的信息，最多包含总节点-2个其他节点的信息。

面向集群的jedis内部实现原理

开发，jedis，redis的java client客户端，redis cluster，jedis cluster api

jedis cluster api与redis cluster集群交互的一些基本原理

1、基于重定向的客户端

redis-cli -c，自动重定向

（1）请求重定向

客户端可能会挑选任意一个redis实例去发送命令，每个redis实例接收到命令，都会计算key对应的hash slot

如果在本地就在本地处理，否则返回moved给客户端，让客户端进行重定向

cluster keyslot mykey，可以查看一个key对应的hash slot是什么

用redis-cli的时候，可以加入-c参数，支持自动的请求重定向，redis-cli接收到moved之后，会自动重定向到对应的节点执行命令。

（2）计算hash slot

计算hash slot的算法，就是根据key计算CRC16值，然后对16384取模，拿到对应的hash slot

用hash tag可以手动指定key对应的slot，同一个hash tag下的key，都会在一个hash slot中，比如set mykey1:{100}和set mykey2:{100}

（3）hash slot查找

节点间通过gossip协议进行数据交换，就知道每个hash slot在哪个节点上

2、smart jedis

1）什么是smart jedis

基于重定向的客户端，很消耗网络IO，因为大部分情况下，可能都会出现一次请求重定向，才能找到正确的节点

所以大部分的客户端，比如java redis客户端，就是jedis，都是smart的

本地维护一份hashslot -> node的映射表，缓存，大部分情况下，直接走本地缓存就可以找到hashslot -> node，不需要通过节点进行moved重定向

2）JedisCluster的工作原理

在JedisCluster初始化的时候，就会随机选择一个node，初始化hashslot -> node映射表，同时为每个节点创建一个JedisPool连接池

每次基于JedisCluster执行操作，首先JedisCluster都会在本地计算key的hashslot，然后在本地映射表找到对应的节点

如果那个node正好还是持有那个hashslot，那么就ok; 如果说进行了reshard这样的操作，可能hashslot已经不在那个node上了，就会返回moved

如果JedisCluter API发现对应的节点返回moved，那么利用该节点的元数据，更新本地的hashslot -> node映射表缓存

重复上面几个步骤，直到找到对应的节点，如果重试超过5次，那么就报错，JedisClusterMaxRedirectionException

jedis老版本，可能会出现在集群某个节点故障还没完成自动切换恢复时，频繁更新hash slot，频繁ping节点检查活跃，导致大量网络IO开销

jedis最新版本，对于这些过度的hash slot更新和ping，都进行了优化，避免了类似问题

3）hashslot迁移和ask重定向

如果hash slot正在迁移，那么会返回ask重定向给jedis

jedis接收到ask重定向之后，会重新定位到目标节点去执行，但是因为ask发生在hash slot迁移过程中，所以JedisCluster API收到ask是不会更新hashslot本地缓存。

已经可以确定说，hashslot已经迁移完了，moved是会更新本地hashslot->node映射表缓存的。

高可用性与主备切换原理

redis cluster的高可用的原理，几乎跟哨兵是类似的

1、判断节点宕机

如果一个节点认为另外一个节点宕机，那么就是pfail，主观宕机

如果多个节点都认为另外一个节点宕机了，那么就是fail，客观宕机，跟哨兵的原理几乎一样，sdown，odown

在cluster-node-timeout内，某个节点一直没有返回pong，那么就被认为pfail

如果一个节点认为某个节点pfail了，那么会在gossip ping消息中，ping给其他节点，如果超过半数的节点都认为pfail了，那么就会变成fail

2、从节点过滤

对宕机的master node，从其所有的slave node中，选择一个切换成master node

检查每个slave node与master node断开连接的时间，如果超过了cluster-node-timeout * cluster-slave-validity-factor，那么就没有资格切换成master

这个也是跟哨兵是一样的，从节点超时过滤的步骤

3、从节点选举

哨兵：对所有从节点进行排序，slave priority，offset，run id

每个从节点，都根据自己对master复制数据的offset，来设置一个选举时间，offset越大（复制数据越多）的从节点，选举时间越靠前，优先进行选举

所有的master node开始slave选举投票，给要进行选举的slave进行投票，如果大部分master node（N/2 + 1）都投票给了某个从节点，那么选举通过，那个从节点可以切换成master

从节点执行主备切换，从节点切换为主节点

4、与哨兵比较

整个流程跟哨兵相比，非常类似，所以说，redis cluster功能强大，直接集成了replication和sentinal的功能。