链滴

Redis：集群（cluster）

作者：function001

原文链接：https://ld246.com/article/1637725764039

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20210119.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h2 id="1-概述">1 概述</h2>
集群，即 Redis Cluster，是 Redis 3.0 开始引入的分布式存储方案。
集群由多个节点(Node)组成，Redis 的数据分布在这些节点中。集群中的节点分为主节点和从节
：只有主节点负责读写请求和集群信息的维护；从节点只进行主节点数据和状态信息的复制。
集群的作用，可以归纳为两点：

数据分区：数据分区(或称数据分片)是集群最核心的功能。

集群将数据分散到多个节点，一方面突破了 Redis 单机内存大小的限制，存储容量大大增加；另
方面每个主节点都可以对外提供读服务和写服务，大大提高了集群的响应能力。
<ol start="2">
高可用：集群支持主从复制和主节点的自动故障转移（与哨兵类似）；当任一节点发生故障时，
群仍然可以对外提供服务。

<h2 id="2-部署集群">2 部署集群</h2>
搭建一个简单的集群：所有节点在同一台服务器上，以端口号进行区分；配置从简。3 个主节点
口号：7000/7001/7002
<h2 id="2-1-redis命令搭建">2.1 redis 命令搭建</h2>
<h3 id="2-1-1-启动节点">2.1.1 启动节点</h3>
节点配置文件，以端口为 7002 的为例：
<pre><code class="language-java highlight-chroma">
配置文件进行了精简</spa
>，完整配置可自行和官方提
的完整conf文件进行对照。<span class="highlig
t-n">端口号自行对应修改
#后台启动的意思
daemonize yes
#端口号
port 7002
日志文件
logfile "cluster-node-7002.log

开启集群
cluster-<span class=
highlight-n">enabled yes
集群持久化配置文件</spa
>，内容包含其它节点的状态<
span>，持久化变量等</spa
>，会自动生成在上面配置的d
r目录下
cluster-<span class=
highlight-n">config-
ile cluster-<
pan class="highlight-n">node-<span class="hi
hlight-n">7002.con

集群节点不可用的最大时间
/span>(毫秒<span cl
ss="highlight-o">)，<span class="highlight-n
>如果主节点在指定时间内不可达，<span class=
highlight-n">那么会进行故障转移
cluster-<span class=

highlight-n">node-timeout 5000
云服务器上部署需指定公网ip
cluster-announce-ip 公网ip地址
允许外部连接
protected-mode no
定义生成rdb文件的名称
dbfilename "dump-7002.rdb"

</code></pre>

cluster-enabled ：一个节点就是一个运行在集群模式下的 Redis 服务器，Redis 服务器在启动时会根据 cluster-enabled 配置选项是否为 yes 来决定是否开启服务器的集群模式。
cluster-config-file：该参数指定了集群配置文件的配置。每个节点在运行过程中，会维护一份集群配置文件；每当集群信息发生变化时（如增减节点），集群内所有节点会将最新信息更新到该配置文件；当节点重启后，会重新读取该配置文件，获取集群信息，可以方便的重新加入到集群中。也就是说，当Redis节点以集群模式启动时，会首先寻找是否有集群配置文件，如果有则使用文件中的配置启动，如果没有，则初始化配置并将配置保存到文件中。集群配置文件由 Redis 节点维护，不需要人工修改。
之后就可以启动服务：例：<code>redis-server redis-7000.conf</code>
然后可以通过 cluster nodes 命令查看节点的状态：

其中返回值第一项表示节点 id，由 40 个 16 进制字符串组成，节点 id 与 主从复制 一文中提到的 runId 不同：Redis 每次启动 runId 都会重新创建，但是节点 id 只在集群初始化时创建一次，然后保存到集群配置文件中，以后节点重新启动时会直接在集群配置文件中读取。
其他节点使用相同办法启动，不再赘述。需要特别注意，在启动节点阶段，节点是没有主从关系的，因此从节点不需要加 slaveof 配置。

<h3 id="2-1-2-创建集群">2.1.2 创建集群</h3>
节点启动之后都是一个独立的集群，并不知道其他节点的存在，需要进行节点握手，将独立的节点组成一个网络。
节点加入集群命令：
<pre><code class="language-java highlight-chroma">cluster meet ip port
</code></pre>
创建集群命令：
<pre><code class="language-java highlight-chroma">redis-cli --cluster

create 172.17.0.13:6381 172.17.0.13:6382 --cluster-replicas 1

</code></pre>

运行上面命令结果：

查看集群的节点，以 7000 为例：

至此集群就创建成功了！！！
<h3 id="2-1-3-创建过程遇到的问题">2.1.3 创建过程遇到的问题</h3>
问题 1：
在实际操作中发现 cluster meet ip port 中 ip 地址如果使用公网 ip，那么加入集群的节点就会手失败。
解决方法：
在配置文件中加入：
<pre><code class="language-java highlight-chroma"># 云服务器上部署需指定公网ip
cluster-announce-ip 公网ip地址
</code></pre>
同时注意在云服务器上需要开放端口和安全组，能够外部连接。
问题 2：
搭建 Redis 集群的过程中，执行到 cluster create : ... 的时候，发现程序在阻塞，显示：Waiting or the cluster to join 的字样，然后就无休无尽的等待... 如图：

解决方法：
开放 Redis 服务的两个 TCP 端口。譬如 Redis 客户端连接端口为 6379，而 Redis 服务在集群还有一个叫集群总线端口，其端口为客户端连接端口加上 10000，即 6379 + 10000 = 16379。所开放每个集群节点的客户端端口和集群总线端口才能成功创建集群！
<blockquote>
客户端端口：客户端访问 Redis 服务器的端口
集群总线端口：用二进制协议(gossip 协议)的点对点集群通信的端口。用于节点的失败侦测、配更新、故障转移授权，等等。
总而言之，客户端端口提供的是外部客户端访问服务的端口；而集群总线端口是提供集群内部各 Redis 服务之间的通信。
</blockquote>
<h2 id="2-2-集群方案设计">2.2 集群方案设计</h2>
设计集群方案时，至少要考虑以下因素：

（1）高可用要求：根据故障转移的原理，至少需要 3 个主节点才能完成故障转移，且 3 个主节不应在同一台物理机上；每个主节点至少需要 1 个从节点，且主从节点不应在一台物理机上；因此高用集群至少包含 6 个节点。
（2）数据量和访问量：估算应用需要的数据量和总访问量(考虑业务发展，留有冗余)，结合每主节点的容量和能承受的访问量(可以通过 benchmark 得到较准确估计)，计算需要的主节点数量。
（3）节点数量限制：Redis 官方给出的节点数量限制为 1000，主要是考虑节点间通信带来的消。在实际应用中应尽量避免大集群；如果节点数量不足以满足应用对 Redis 数据量和访问量的要求，以考虑：(1)业务分割，大集群分为多个小集群；(2)减少不必要的数据；(3)调整数据过期策略等。</p

（4）适度冗余：Redis 可以在不影响集群服务的情况下增加节点，因此节点数量适当冗余即可不用太大。
<h2 id="3-集群原理">3 集群原理</h2>
<h2 id="3-1-集群的数据结构">3.1 集群的数据结构</h2>
clusterNode 结构保存了一个节点的当前状态，比如节点的创建时间、节点的名字、节点当前的置纪元、节点的 IP 地址和端口号等等。
每个节点都会使用-一个 clusterNode 结构来记录自己的状态，并为集群中的所有其他节点(包括节点和从节点)都创建-一个相应的 clusterNode 结构，以此来记录其他节点的状态：

</p

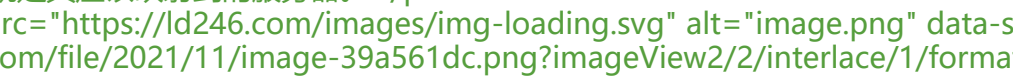
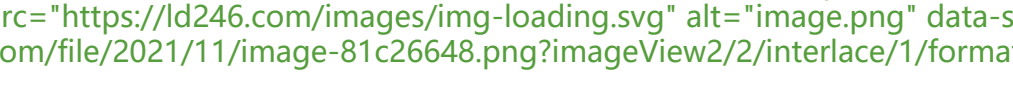
clusterNode 结构的 link 属性是一个 clusterLink 结构，该结构保存了连接节点所需的有关信息套接字描述符，输入缓冲区，输出缓存区：

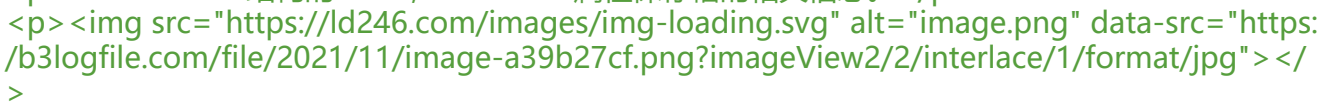
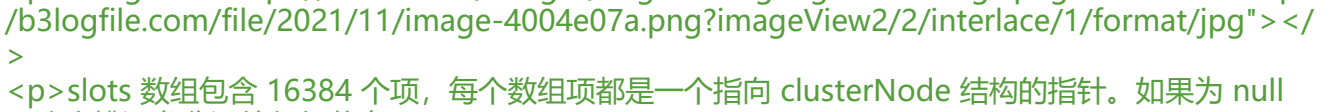
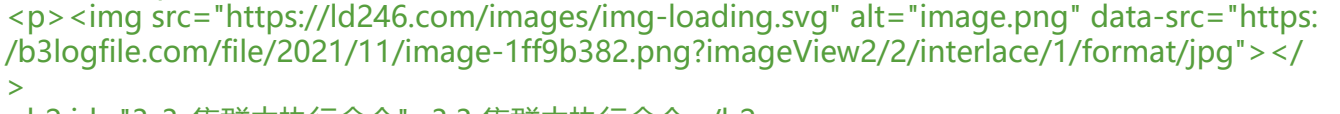
</
>
最后，每个节点都保存着-一个 clusterState 结构，这个结构记录了在当前节点的视
角下，集群目前所处的状态，例如集群是在线还是下线，集群包含多少个节点，集群当前的
配置纪元，诸如此类: .

</
>
<h2 id="3-2-cluster-meet-命令实现">3.2 cluster meet 命令实现</h2>
通过向节点 A 发送 cluster meet ip port 命令，可以让另一个节点 B 加入 A 节点的集群中。</
>
收到命令的节点 A 将于节点 B 进行握手（handShake）以此来确定彼此的存在，握手流程：</
>

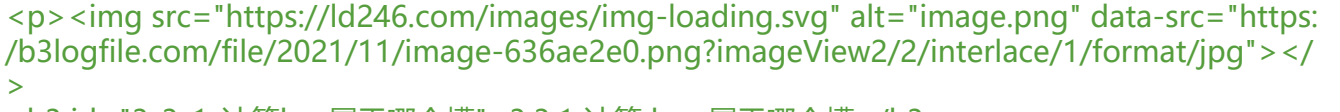
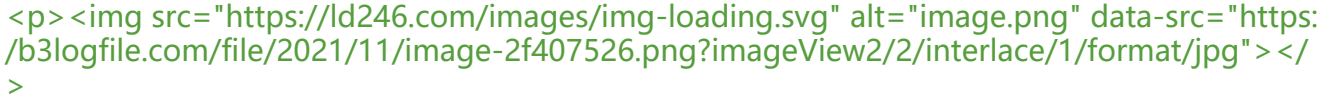
节点 A 会为节点 B 创建一个 ClusterNode 结构，并将该结构添加到自己的 clusterState.nodes 字典里面。
节点 A 根据 Cluster meet 命令给定的 IP 地址和端口号，向节点 B 发送一条 MEET 消息。
节点 B 将接受到节点 A 发送的 MEET 消息，节点 B 会为节点 A 创建一个 clusterNode 结构，将该结构添加到自己的 clusterState.nodes 字典中。
节点 B 将向节点 A 返回一条 PONG 消息。
节点 A 接收到节点 B 返回的 PONG 消息，通过这条 PONG 消息节点 A 可以知道节点 B 已经成地接收到自己发送的 MEET 消息。
节点 A 向节点 B 返回一条 PING 消息
节点 B 接收到节点 A 返回的 PING 消息，通过这条 PING 消息节点 B 可以知道节点 A 已经成功收到自己返回的 PONG 消息，握手完成。

</
>
最后，节点 A 会将节点 B 的信息通过 Gossip 协议传播给集群中的其他节点，让其他节点也与

点 B 进行握手，最终，节点 B 会被集群中的所有节点认识。
<h2 id="3-2-集群的数据分区">3.2 集群的数据分区</h2>
<h3 id="3-2-1-数据分区方案">3.2.1 数据分区方案</h3>
数据分区有顺序分区、哈希分区等，其中哈希分区由于其天然的随机性，用广泛；集群的分区方案便是哈希分区的一种。
哈希分区的基本思路是：对数据的特征值（如 key）进行哈希，然后根据哈希值决定数据落在哪节点。常见的哈希分区包括：哈希取余分区、一致性哈希分区、带虚拟节点的一致性哈希区等。
衡量数据分区方法好坏的标准有很多，其中比较重要的两个因素是(1)数据分布是否均匀(2)增加删减节点对数据分布的影响。由于哈希的随机性，哈希分区基本可以保证数据分布均匀；因此在比较希分区方案时，重点要看增减节点对数据分布的影响。
（1）哈希取余分区
哈希取余分区思路非常简单：计算 key 的 hash 值，然后对节点数量进行取余，从而决定数据映到哪个节点上。该方案最大的问题是，当新增或删减节点时，节点数量发生变化，系统中所有的数据需要重新计算映射关系，引发大规模数据迁移。
（2）一致性哈希分区
一致性哈希算法将整个哈希值空间组织成一个虚拟的圆环，如下图所示，范围为 0-2^32-1；对每个数据，根据 key 计算 hash 值，确定数据在环上的位置，然后从此位置沿环顺时针行走，找到的一台服务器就是其应该映射到的服务器。

/b3logfile.com/file/2021/11/image-39a561dc.png?imageView2/2/interlace/1/format/jpg"></
>
与哈希取余分区相比，一致性哈希分区将增减节点的影响限制在相邻节点。以上图为例，如果在ode1 和 node2 之间增加 node5，则只有 node2 中的一部分数据会迁移到 node5；如果去掉 node，则原 node2 中的数据只会迁移到 node4 中，只有 node4 会受影响。
一致性哈希分区的主要问题在于，当节点数量较少时，增加或删减节点，对单个节点影响可能很大，造成数据的严重不平衡。还是以上图为例，如果去掉 node2，node4 中数据由总数据的 1/4 左右变为 1/2 左右，与其他节点相比负载过高。
（3）带虚拟节点的一致性哈希分区
该方案在一致性哈希分区的基础上，引入了虚拟节点的概念。Redis<stron>集群使用的便是该方案，其中的虚拟节点称为槽（slot）。槽是于数据和实际节点之间的虚拟概念；每个实际节点包含一定数量的槽，每个槽包含哈希值在一定范围的数据。引入槽以后，数据的映射关系由数据 hash-> 实际节点，变成了数据 hash-> 槽-> 实际节点。
在使用了槽的一致性哈希分区中，槽是数据管理和迁移的基本单位。槽解耦了数据和际节点之间的关系，增加或删除节点对系统的影响很小。仍以上图为例，系统中有 4 个实节点，假设为其分配 16 个槽(0-15)； 槽 0-3 位于 node1，4-7 位于 node2，以此类推。如果此时除 node2，只需要将槽 4-7 重新分配即可，例如槽 4-5 分配给 node1，槽 6 分配给 node3，槽 7 配给 node4；可以看出删除 node2 后，数据在其他节点的分布仍然较为均衡。
槽的数量一般远小于 2^32，远大于实际节点的数量；在 Redis 集群中，槽的数量为 16384。</
>
下面这张图很好的总结了 Redis 集群将数据映射到实际节点的过程：

/b3logfile.com/file/2021/11/image-81c26648.png?imageView2/2/interlace/1/format/jpg"></
>
<h3 id="3-2-2-数据管理基本单位槽-slot-">3.2.2 数据管理基本单位槽（slot）</h3>
Redis 集群通过分片的方式来保存数据库中的键值对:集群的整个数据库被分为 16384 个槽(slot)数据库中的每个键都属于这 16384 个槽的其中-一个，集群中的每个节点可以处理 0 个或最多 16384 个槽。
当数据库中的 16384 个槽都有节点在处理时,集群处于上线状态(ok);相反地，如果数据库中有任一个槽没有得到处理，那么集群处于下线状态(fail)。
（1）指派槽
通过向节点发送 CLUSTER ADDSLOTS 命令，我们可以将一个或多个槽指派（assign）给节点：
/p>

CLUSTER ADDSLOTS < slot > [slot ...]
例：
127.0.0.1:8000> CLUSTER ADDSLOTS 0 1 2 ... 10000
（2）保存槽信息
clusterNode 结构的 slots，numslots 属性保存槽的相关信息。

slots 属性是一个二进制位数组，这个数组的大小是 16284/8=2048 个字节，共包含 16384 个进制位。
redis 以 0 为起始索引，16383 为终止索引，slots[i] = 1 表示该节点负责槽 i，=0 表示不负责槽
numslots 属性则记录节点负责处理槽的数量，即 slots 数组中值为 1 的数量。
（3）传播槽信息
一个节点除了会将自己负责处理的槽记录在 clusterNode 结构的 slots 属性和 numslots 属性外他还会将自己的 slots 数组通过消息发送给集群中的其他节点，以此来告知其它节点自己目前负责处哪些槽。
当节点 A 通过消息从节点 B 那里接收到节点 B 的 slots 数组时，节点 A 会在自己的 clusterState.nodes 字典中查找节点 B 对应的 clusterNode 结构，并对结构中的 slots 数组进行保存或者更新。
因为集群中的每个节点都会将自己的 slots 数组通过消息发送给集群中的其他节点，并且每个接到 slots 数组的节点都会将数组保存到相应节点的 clusterNode 结构里面。因此，集群中的每个节点会知道数据库中的 16384 个槽分别被指派给了集群中的哪些节点。
（4）保存集群中所有槽信息
clusterState 结构中 slots 数组记录集群中所有 16384 个槽的指派信息：

slots 数组包含 16384 个项，每个数组项都是一个指向 clusterNode 结构的指针。如果为 null 示这个槽没有分派给任何节点。
<h3 id="3-2-3-cluster-addslots-命令实现">3.2.3 cluster addslots 命令实现</h3>
CLUSTER ADDSLOTS 命令接受一个或多个槽作为参数，并将所有输入的槽指派给接收该命令的点负责。

<h2 id="3-3-集群中执行命令">3.3 集群中执行命令</h2>
在对数据库中的 16384 个槽都进行了指派之后，集群就会进入上线状态，这时客户端就
可以向集群中的节点发送数据命令了。
客户端向节点发送与数据库键有关命令时，接收命令的节点会计算出命令要处理键属于哪个槽，检查这个槽是否指派给自己：

如果键所在槽就是当前节点，那么直接执行命令。
如果不在当前节点，那么节点会向客户端返回个亿 MOVEN 错误，指引客户端跳转（redirect）正确节点，并再次发送之前想要执行的命令。


<h3 id="3-3-1-计算key属于哪个槽">3.3.1 计算 key 属于哪个槽</h3>

以上代码就是计算 key 属于按个槽的代码，其中 CRC16(key)语句计算 key 的 CRC-16 校验和。

/p>
使用 cluster keyslot < key > 命令可以查看 key 属于哪个槽。
<h3 id="3-3-2-判断槽">3.3.2 判断槽</h3>
找到槽之后，根据 clusterState 中保存的 slots 数组判断当前节点是否是管理该槽的。如果不是向客户端返回 Moved 错误，指引客户端转向管理该槽的节点，然后重新发送命令。
<h3 id="3-3-3-MOVED错误">3.3.3 MOVED 错误</h3>
节点发现要处理的命令的 key 的槽不在自己的节点上，就会返回 MOVED 错误。
格式：
MOVED slot ip port
slot 是处理键的槽，ip，port 是这个管理这个槽的节点。
一个集群客户端通常会与集群中的多个节点创建套接字连接，而所谓的节点转向就是换一个套接来发送命令。
如果客户端尚未与想要转向的节点创键套接字连接，那么客户端会先根据 MOVED 错误提供的 IP 地址和端口号来连接节点，然后再进行转向，之后再重新发送命令。
<h3 id="3-3-4-集群下的节点数据库">3.3.4 集群下的节点数据库</h3>
集群下的节点和单机服务器在存储键值对，对键值对过期时间处理方面都相同。
它们之间的一个区别就是集群下的节点只能使用一个数据库，而单机服务器没有这个限制。
节点除了将键值对保存到数据库中之外，节点还会用 clusterState 结构中的 slots_to_keys 跳跃表（有序的按分值排序，分值相同的按对象排序，对象不可以重复，分值可以重复）来保存和键之间的关系。
slots_to_keys 跳跃表每个节点的分值（score）都是一个槽号，而每个节点的成员对象都是一个据库键：

每当节点往数据库中添加一个新的键值对时，节点就会将这个键以及键的槽号关联到 slots_to_ke s 跳跃表。
当节点删除数据库中某个键值对时，节点就会在 slots_to_keys 跳跃表解除被删除键与槽号的关 。

<h2 id="3-4-集群重新分片">3.4 集群重新分片</h2>
Redis 集群的重新分片操作可以将任意数量已经指派给某个节点(源节点)的槽改为指派给另一个点(目标节点),并且相关槽所属的键值对也会从源节点被移动到目标节点。
重新分片操作可以在线(online)进行，在重新分片的过程中，集群不需要下线，并且源节点和目节点都可以继续处理命令请求。
<h3 id="3-4-1-分片原理">3.4.1 分片原理</h3>
Redis 集群的重新分片操作由 redis 的集群管理软件 redis-trib 负责执行的，redis 提供了进行新分片所需的所有命令，而 redis-trib 则通过向源节点和目标节点发送命令来进行重新分片操作。</ >
redis-trib 对集群的单个槽 slot 进行重新分片的步骤如下：

redis-trib 对目标节点发送 cluster setslot < slot > importing <source_id> 命令，目标节点准备好从源节点导入属于槽 slot 的键值对。
redis-trib 对源节点发送 cluster setslot < slot > migating <target_id> 命令，让节点准备好将属于槽 slot 的键值对迁移至目标节点。
redis-trib 向源节点发送 cluster getkeysinslot < slot> < count> 命令，获得最多 c unt 个属于槽 slot 的键值对的键名。
对于步骤 3 获取的每个键名，redis-trib 都向源节点发送一个 migrate <target_ip> <tar et_port> <key_name> 0 < timeout> 命令，将被选中的键原子地从源节点迁移至目节点。
重复执行步骤 3 和步骤 4，直到源节点保存的所有属于槽 slot 的键值对都被迁移至目标节点为 。每次迁移键的过程如图所示。
redis-trib 向集群中任意一个节点发送 cluster setslot < slot> node <target_id> 命 ，将槽 slot 指派给目标节点，这一指派信息会通过消息发送至整个集群，最终集群中的所有节点都会道槽 slot 已经指派给了目标节点。


如果分片涉及多个槽，那么 redis-trib 会对每一个给的的槽执行以上步骤。

<h3 id="3-4-2-cluster-setslot-importing-命令实现">3.4.2 cluster setslot importing 命令实现</h3>
clusterState 结构的 importing_slots_from 数组记录了当前节点正在从其他节点导入的槽。

如果 importing_solts_from[i] 的值不为 null，而是指向一个 clusterNode 结构，那么表示当前节点正在从 clusterNode 所代表的节点导入槽 i。
<h3 id="3-4-3-cluster-setslot-migrating-命令实现">3.4.3 cluster setslot migrating 命令实现</h3>
clusterState 结构的 migrating_slots_to 数组记录了当前节点正在迁移至其他节点的槽：

如果 migrating_slots_to[i]的值不为 null，而是指向一个 clusterNode 结构，那么表示当前节点正在将槽 i 迁移至 clusterNode 所代表的节点。
<h2 id="3-5-ASK错误">3.5 ASK 错误</h2>
在重新分片期间，源节点目标节点迁移一个槽的过程中，可能出现一中情况：属于被迁移槽的一分键值对保存在源节点，一部分保存在目标节点中。
当客户端向源节点发送一个与数据库键有关的命令，并且命令要处理的数据库键恰好就属于正在迁移的槽时：

如上图所示，当出现问题时，节点向客户端返回 ASK 错误，接到 ASK 错误的客户端会根据错误供的 IP 地址和端口号，转向至正在导入槽的目标节点，然后首先向目标节点发送一个 ASKING 命令之后再重新发送原本想要执行的命令。
<h3 id="3-5-1-ASKING命令">3.5.1 ASKING 命令</h3>
ASKING 命令唯一做的就是打开发送改命令的客户端的 REDIS_ASKING 标识。
为什么要首先发送一个 ASKING 命令，然后在发送想要执行的命令呢？
在一般情况下，如果客户端向节点发送一个关于槽 i 的命令，而槽 i 又没有指派给这个节点的话那么节点将向客户端返回一个 MOVED 错误（然后转向）。但是，如果节点的 clusterState.importin_slots_from[i] 显示节点正在导入槽 i，并且发送命令的客户端带有 REDIS_ASKING 标识，那么节点破例执行这个关于槽 i 的命令一次。

<h3 id="3-5-2-ASK错误和MOVED错误的区别">3.5.2 ASK 错误和 MOVED 错误的区别</h3>
ASK 错误和 MOVED 错误都会导致客户端转向，它们区别在于：

MOVED 错误标识槽的负责权在另一个节点，客户端收到关于槽 i 的 MOVED 错误之后，直接连的 MOVED 给定的节点，以后每次遇到关于槽 i 的命令请求时，都可以直接将该命令请求发送到 MOED 错误指向的节点。
与 MOVED 相反，ASK 错误只是两个节点在迁移槽的过程中使用的是一种临时措施，在客户端

到关于槽 i 的 ASK 错误之后，客户端只会在接下来的一次命令请求中将关于槽 i 的命令请求发送至 A
K 错误所指示的节点，但这种转向不会对客户端今后发送关于槽 i 的命令请求产生任何影响，客户端
然会将关于槽 i 的命令请求发送至目前负责处理槽 i 的节点，除非 ASK 错误再次出现。

<h2 id="3-6-复制与故障转移">3.6 复制与故障转移</h2>
Redis 集群中的节点分为主节点(master)和从节点(slave),其中主节点用于处理槽，而从节点则
于复制某个主节点，并在被复制的主节点下线时，代替下线主节点继续处理命令请求。
<h3 id="3-6-1-设置从节点">3.6.1 设置从节点</h3>
向一个节点发送命令：
cluster replicate < node_id>
可以让接受命令的节点成为 node_id 所指定节点的从节点，并开始对主节点进行复制：

接受到该命令的节点首先会在自己的 clusterState.nodes 字典中找到 node_id 所对应节点的 clus
erNode 结构，并将自己的 clusterState.myself.slaveof 指针指向这个结构，以此来记录这个节点正
复制的主节点：<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-s
c="https://b3logfile.com/file/2021/11/image-fd1df2f1.png?imageView2/2/interlace/1/format/
pg">
节点会修改自己在 clusterState.myself.flags 中的属性，关闭原本的 REDIS_NODE_MASTER 标
，打开 REDIS_NODE_SLAVE 标识，表示这个节点已经从原来的主节点变成了从节点。
最后，节点会调用复制代码，并根据 clusterState.myself.slaveof 指向的 clusterNode 结构所
存的 IP 地址和端口号，对主节点进行复制。节点的复制功能和单机 redis 服务器的复制功能使用了
同的代码，所以让从节点复制主节点相当于向从节点发送命令 slaveof < master_ip> <maste
_port>。

一个节点成为从节点，并开始复制某个主节点这一信息会通过消息发送给集群中的其他节点，最
集群中的所有节点都会知道某个从节点正在复制某个主节点。
集群中所有节点都会在代表主节点的 clusterNode 结构的 slaves 属性和 numslaves 属性中记
正在复制这个主节点的从节点名单：
<pre><code class="language-csharp highlight-chroma">struct</
pan> clusterNode{
//正在复制这个主节点的从节点数量
 int <sp
n class="highlight-n">numslaves
//一个数组，每个数组项指向一个正在复制这个主节点的从节点的cl
sterNode结构
 struct <
pan class="highlight-nc">clusterNode **<span c
ass="highlight-n">slaves;
}
</code></pre>
<h3 id="3-6-2-故障检测">3.6.2 故障检测</h3>
集群中的每个节点都会定期地向集群中的其他节点发送 PING 消息，以此来检测对方是否在线，
果接收 PING 消息的节点没有在规定的时间内，向发送 PING 消息的节点返回 PONG 消息，那么发送
PING 消息的节点就会将接收 PING 消息的节点标记为疑似下线(probable fail, PFAIL)。
集群中的各个节点会通过互相发送消息的方式来交换集群中各个节点的状态信息，节点的状态：
线，疑似下线（PFAIL），已下线（FAIL）。
当一个主节点 A 通过消息得知主节点 B 认为主节点 C 进入了疑似下线状态时，主节点 A 会在自
的 clusterState.nodes 字典中找到主节点 C 所对应的 clusterNode 结构，并将主节点 B 的下线报告
加到 cluster 结构的 fail_reports 链表里面：
<pre><code class="language-c highlight-chroma">struct</span
clusterNode{
//一个链表记录了所有其他节点对该节点的下线报告
 list <spa
class="highlight-o">*fail_reports<span class="h

ghlight-p">;
}
</code></pre>
下线报告由一个 clusterNodeFailReport 结构表示：
<pre><code class="language-csharp highlight-chroma">struct clusterNodeFailReport

    //报告目标节点以及下线的节点
    struct clusterNode *node;
    //最后一次从node节点收到下线报告的时间
    //程序使用这个时间戳来检查下线报告是否过期，与当前时间相差太大的下线报告会被删除
    mstime_t time;
}
</code></pre>
如果在一个集群里面，半数以上负责处理槽的主节点都将某个主节点 x 报为疑似下线，那么这个主节点 x 将被标记为已下线(FAIL),将主节点 x 标记为已下线的节点会向集群广一条关于主节点 x 的 FAIL 消息，所有收到这条 FAIL 消息的节点都会立即将主节点 x 标记为已下线。/p>
<h3 id="3-6-3-故障转移">3.6.3 故障转移</h3>
当一个从节点发现自己正在复制的主节点进人了已下线状态时，从节点将开始对下线主节点进行障转移，以下是故障转移的执行步骤:

复制下线主节点的所有从节点里面，会有一个从节点被选中。
被选中的从节点会执行 SLAVEOF no one 命令，成为新的主节点。
新的主节点会撤销所有对已下线主节点的槽指派，并将这些槽全部指派给自己。
新的主节点向集群广播一条 PONG 消息,这条 PONG 消息可以让集群中的其他节点立即知道这节点已经由从节点变成了主节点,并且这个主节点已经接管了原本由已下线节点负
责处理的槽。
新的主节点开始接收和自己负责处理的槽有关的命令请求，故障转移完成。

<h3 id="3-6-4-选举新节点">3.6.4 选举新节点</h3>
集群选举新的主节点的方法：

集群的配置纪元是一个自增计数器，他的初始值为 0。
对集群里的某个节点开始一次故障转移操作时，集群配置纪元的值会被加一。
对于每个配置纪元，集群里每个负责处理槽的主节点都有一次投票的机会，而第一个向主节点要投票的从节点将获得主节点的投票。
当从节点发现自己正在复制的主节点进入已下线状态时，从节点会向集群广播一条 cluster_type_ailover_auth_request 消息，要求所有收到这条消息，并且具投票权的主节点向这个从节点投票。</l>
如果一个主节点具有投票权（它正在负责处理槽）并且这个节点没有投票给其他从节点，那么主点将向要求投票的从节点返回一条 clustermsg_type_failover_auth_ask 消息，表示这个主节点支持节点称为新的主节点。
每个参与选举的从节点都会接收 clustermsg_type_failover_auth_ack 消息，根据收到了多少条种消息，来统计自己获得了多少主节点的支持。
如果集群中有 N 个具有投票权的主节点，那么当一个从节点收集到大于等于 N/2+1 张支持票时这个从节点就会被当选为新的主节点。
在每一个配置纪元中，具有投票权的主节点只能投一次票，那么具有超过一半票数的从节点只会一个，确保了新的主节点只会有一个。
如果在一个配置纪元里面没有从节点能收集到足够多的支持票，那么集群进入一个新的配置纪元

并再次进行选举，直到选出新的主节点为止。

这种选举新主节点的方法和选举领头 Sentinel 的方法非常相似，两者都是基于 Raft 算法的领头举方法实现的。
<h2 id="3-7集群的通信机制">3.7 集群的通信机制</h2>
<h3 id="3-7-1-两个端口">3.7.1 两个端口</h3>
在哨兵系统中，节点分为数据节点和哨兵节点：前者存储数据，后者实现额外的控制功能。在集中，没有数据节点与非数据节点之分：所有的节点都存储数据，也都参与集群状态的维护。为此，集中的每个节点，都提供了两个 TCP 端口：

普通端口：即我们在前面指定的端口(7000 等)。普通端口主要用于为客户端提供服务（与单机节类似）；但在节点间数据迁移时也会使用。
集群端口：端口号是普通端口 +10000（10000 是固定值，无法改变），如 7000 节点的集群端为 17000。集群端口只用于节点之间的通信，如搭建集群、增减节点、故障转移等操作时节点间的通；不要使用客户端连接集群接口。为了保证集群可以正常工作，在配置防火墙时，要同时开启普通端和集群端口。

<h3 id="3-7-2-Gossip协议">3.7.2 Gossip 协议</h3>
节点间通信，按照通信协议可以分为几种类型：单对单、广播、Gossip 协议等。重点是广播和 Gssip 的对比。
广播是指向集群内所有节点发送消息；优点是集群的收敛速度快(集群收敛是指集群内所有节点得的集群信息是一致的)，缺点是每条消息都要发送给所有节点，CPU、带宽等消耗较大。
Gossip 协议的特点是：在节点数量有限的网络中，每个节点都“随机”的与部分节点通信（并是真正的随机，而是根据特定的规则选择通信的节点），经过一番杂乱无章的通信，每个节点的状态快会达到一致。Gossip 协议的优点有负载(比广播)低、去中心化、容错性高(因为通信有冗余)等；缺主要是集群的收敛速度慢。
<h3 id="3-7-3-消息">3.7.3 消息</h3>
集群中的各个节点通过发送和接收消息来进行通信，发送消息的节点称为 sender（发送者），收消息的称为 receiver（接收者）。
节点发送的消息主要有五种：

MEET 消息：当发送者接到客户端发送的 cluster 命令时，发送者会向接收者发送 MEET 消息，求接受者加入到发送者当前所处的集群里面。
PING 消息：集群里的每个节点默认每隔一秒钟就会从已知节点列表中随机选出五个节点，然后这五个节点中最长时间没有发送过 PING 消息的节点发送 PING 消息，以此来检测被选中的节点是否线。除此之外，如果节点 A 最后一次收到节点 B 发送的 PONG 消息的时间，距离当前时间已经超过节点 A 的 cluster-node-timeout 选项设置时长的一半，那么节点 A 也会向节点 B 发送 PING 消息这可以防止节点 A 因为长时间没有随机选中节点 B 作为 PING 消息的发送对象而导致对节点 B 的信更新滞后。
PONG 消息：当接收者收到发送者发来的 MEET 消息或者 PING 消息时，为了向发送者确认这条 MEET 消息或者 PING 消息已到达，接收者会向发送者返回一条 PONG 消息。另外，一个节点也可通过向集群广播自己的 PONG 消息来让集群中的其他节点立即刷新关于这个节点的认识，例如当一故障转移操作成功执行之后，新的主节点会向集群广播一条 PONG 消息，以此来让集群中的其他节立即知道这个节点已经变成了主节点，并且接管了已下线节点负责的槽。
FAIL 消息:当一个主节点 A 判断另-一个主节点 B 已经进人 FAIL 状态时，节点 A 会向集群广播条关于节点 B 的 FAIL 消息，所有收到这条消息的节点都会立即将节点 B 标记为已下线。
PUBLISH 消息:当节点接收到-一个 PUBLISH 命令时，节点会执行这个命令，并向集群广播一条 UBLISH 消息，所有接收到这条 PUBLISH 消息的节点都会执行相同的 PUBLISH 命令。

所有的消息都由：消息头（header）+ 消息正文（data） 组成。
<h2 id="4常用命令">4 常用命令</h2>
集群
cluster info ： 打印集群的信息
cluster nodes ： 列出集群当前已知的所有节点（ node），以及这些节点的相关信息。

节点
cluster meet ： 将 ip 和 port 所指定的节点添加到集群当中，让它成为集群的一份子。
cluster forget <node_id>： 从集群中移除 node_id 指定的节点。
cluster replicate <node_id>： 将当前节点设置为 node_id 指定的节点的从节点。
cluster saveconfig ： 将节点的配置文件保存到硬盘里面。
槽(slot)
cluster addslots [slot ...]： 将一个或多个槽（ slot）指派（ assign）给当前节点。
cluster delslots [slot ...]： 移除一个或多个槽对当前节点的指派。
cluster flushslots： 移除指派给当前节点的所有槽，让当前节点变成一个没有指派任何槽的节点。<b
>
cluster setslot node <node_id>： 将槽 slot 指派给 node_id 指定的节点，如果槽已经指派
另一个节点，那么先让另一个节点删除该槽 >，然后再进行指派。
cluster setslot migrating <node_id>： 将本节点的槽 slot 迁移到 node_id 指定的节点
。
cluster setslot importing <node_id>： 从 node_id 指定的节点中导入槽 slot 到本节点。<b
>
cluster setslot stable： 取消对槽 slot 的导入（ import）或者迁移（ migrate）。
键
cluster keyslot ： 计算键 key 应该被放置在哪个槽上。
cluster countkeysinslot ： 返回槽 slot 目前包含的键值对数量。
cluster getkeysinslot ： 返回 count 个 slot 槽中的键
<h2 id="5-参考文档">5 参考文档</h2>
集群相关命令 1<
p>
集群相关命令 2
云服务器 Redis 集群部署及客户端通过公网 IP 连接问题
<a href="https://link.ld246.com/forward?goto=https%3A%2F%2Fwww.cnblogs.com%2Fksmetv%2Fp%2F9853040.html%23t21" target="_blank" rel="nofollow ugc">编程迷思---集群</
>
《redis 的设计与实现》