为什么要有 cluster 集群？ | 小林coding

为什么要有 cluster 集群？

> 原文地址：字节二面：你知道 Redis Cluster 集群工作原理吗？

有位伙伴面试了字节，说有道题，他当时答不上，大家一起来看看：**redis的cluster集群原理，客户端是怎样知道该访问哪个分片的**。

我们应该怎样更好回答呢？可以分这几个维度

- 为什么需要Redis Cluster？哨兵模式不香吗？
- 客户端是怎样知道该访问哪个分片的? (哈希槽)
- redis实例上并没有相应的数据，会怎么样？(MOVED重定向和ASK重定向)
- 各个节点之间是怎么通信的呢(Gossip协议)
- 集群内节点出现故障怎么办（故障转移）
- 加餐：Redis Cluster的Hash Slot 为什么是16384？

1. 为什么需要Redis Cluster？

哨兵模式基于主从模式，实现读写分离，它还可以自动切换，系统可用性更高。但是它**每个节点存储的数据是一样的，浪费内存，并且不好在线扩容**。

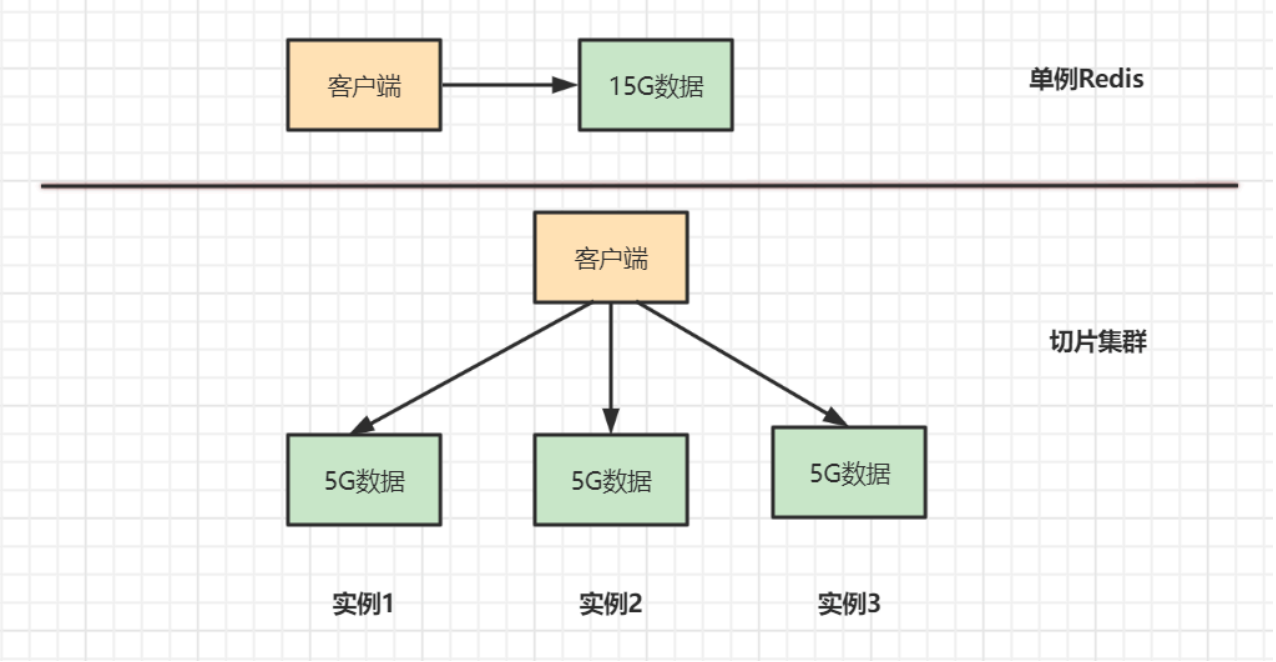
因此，Reids Cluster集群（切片集群的实现方案）应运而生，它在Redis3.0加入的，实现了Redis的分布式存储。对数据进行分片，也就是说**每台Redis节点上存储不同的内容，来解决在线扩容的问题。并且，它可以保存大量数据，即分散数据到各个Redis实例，还提供复制和故障转移的功能。**

> 比如你一个Redis实例保存15G甚至更大的数据，响应就会很慢，这是因为Redis RDB 持久化机制导致的，Redis会fork子进程完成 RDB 持久化操作，fork执行的耗时与 Redis 数据量成正相关。

这时候你很容易想到，把15G数据分散来存储就好了嘛。这就是Redis切片集群的初衷。

切片集群是啥呢？来看个例子，如果你要用Redis保存15G的数据，可以用单实例Redis，或者3台Redis实例组成切片集群，对比如下：

> 切片集群和Redis Cluster的区别：Redis Cluster是从Redis3.0版本开始，官方提供的一种实现切片集群的方案。



既然数据是分片分布到不同Redis实例的，**那客户端到底是怎么确定想要访问的数据在哪个实例上呢**？我们一起来看下Reids Cluster是怎么做的哈。

2. 客户端是怎样知道该访问哪个分片的?哈希槽

Redis Cluster方案采用哈希槽（Hash Slot），来处理数据和实例之间的映射关系。

> 一个切片集群被分为16384个slot（槽），每个进入Redis的键值对，根据key进行散列，分配到这16384插槽中的一个。使用的哈希映射也比较简单，用CRC16算法计算出一个16bit的值，再对16384取模。数据库中的每个键都属于这16384个槽的其中一个，集群中的每个节点都可以处理这16384个槽。

集群中的每个节点负责一部分的哈希槽，假设当前集群有A、B、C3个节点，每个节点上负责的哈希槽数 =16384/3，那么可能存在的一种分配：

- 节点A负责0~5460号哈希槽
- 节点B负责5461~10922号哈希槽
- 节点C负责10923~16383号哈希槽

客户端给一个Redis实例发送数据读写操作时，如果这个实例上并没有相应的数据，会怎么样呢？**MOVED重定向和ASK重定向了解一下哈**

3. 实例上并没有相应的数据，会怎么样？(MOVED重定向和ASK重定向)

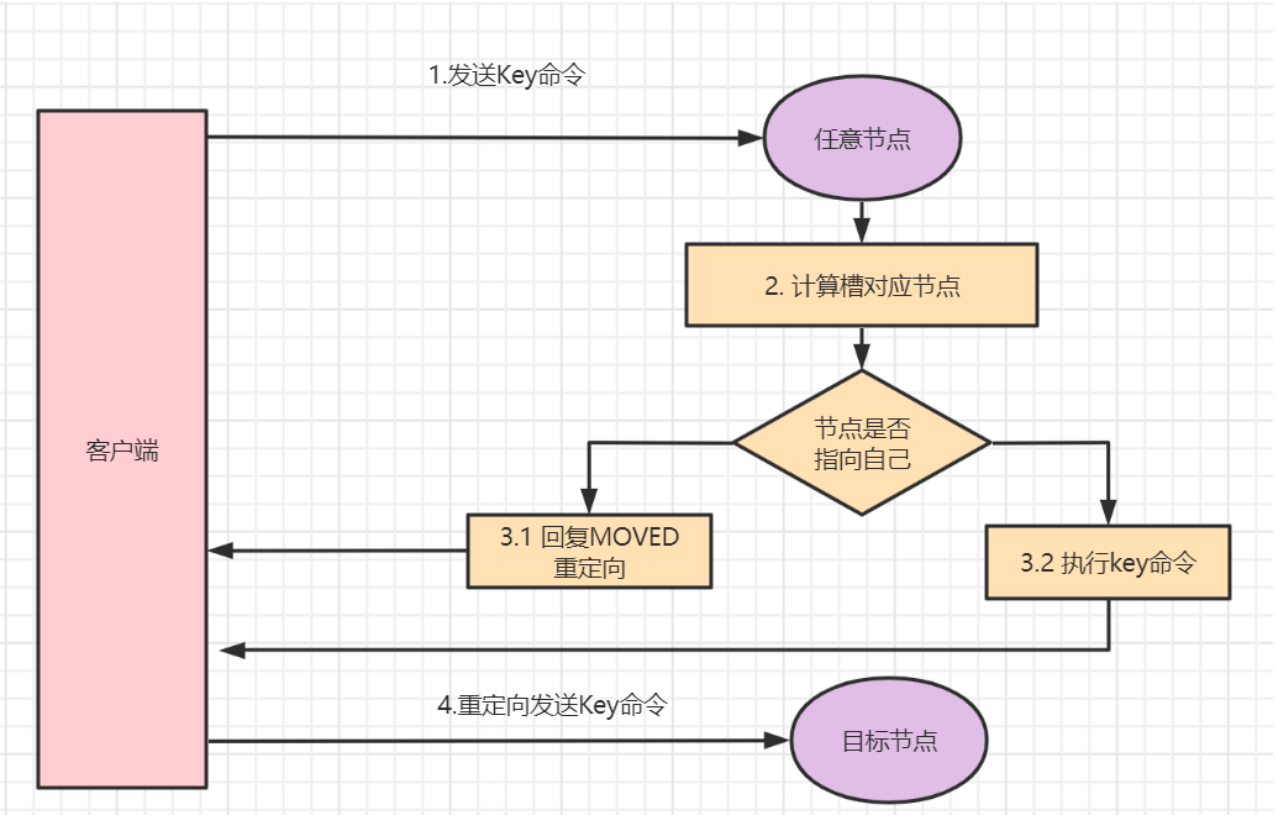
在Redis cluster模式下，节点对请求的处理过程如下：

- 通过哈希槽映射，检查当前Redis key是否存在当前节点
- 若哈希槽不是由自身节点负责，就返回MOVED重定向
- 若哈希槽确实由自身负责，且key在slot中，则返回该key对应结果
- 若Redis key不存在此哈希槽中，检查该哈希槽是否正在迁出（MIGRATING）？

- 若Redis key正在迁出，返回ASK错误重定向客户端到迁移的目的服务器上
- 若哈希槽未迁出，检查哈希槽是否导入中？
- 若哈希槽导入中且有ASKING标记，则直接操作，否则返回MOVED重定向

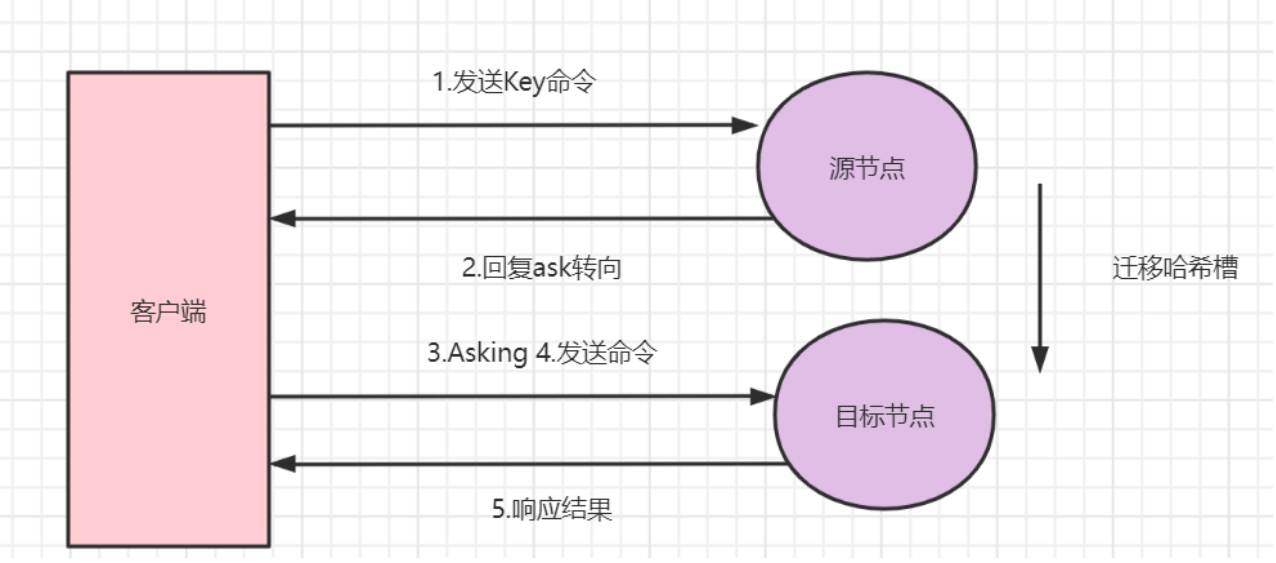
3.1 Moved 重定向

客户端给一个Redis实例发送数据读写操作时，如果计算出来的槽不是在该节点上，这时候它会返回MOVED重定向错误，MOVED重定向错误中，会将哈希槽所在的新实例的IP和port端口带回去。这就是Redis Cluster的MOVED重定向机制。流程图如下：



3.2 ASK 重定向

Ask重定向一般发生于集群伸缩的时候。集群伸缩会导致槽迁移，当我们去源节点访问时，此时数据已经可能已经迁移到了目标节点，使用Ask重定向可以解决此种情况。

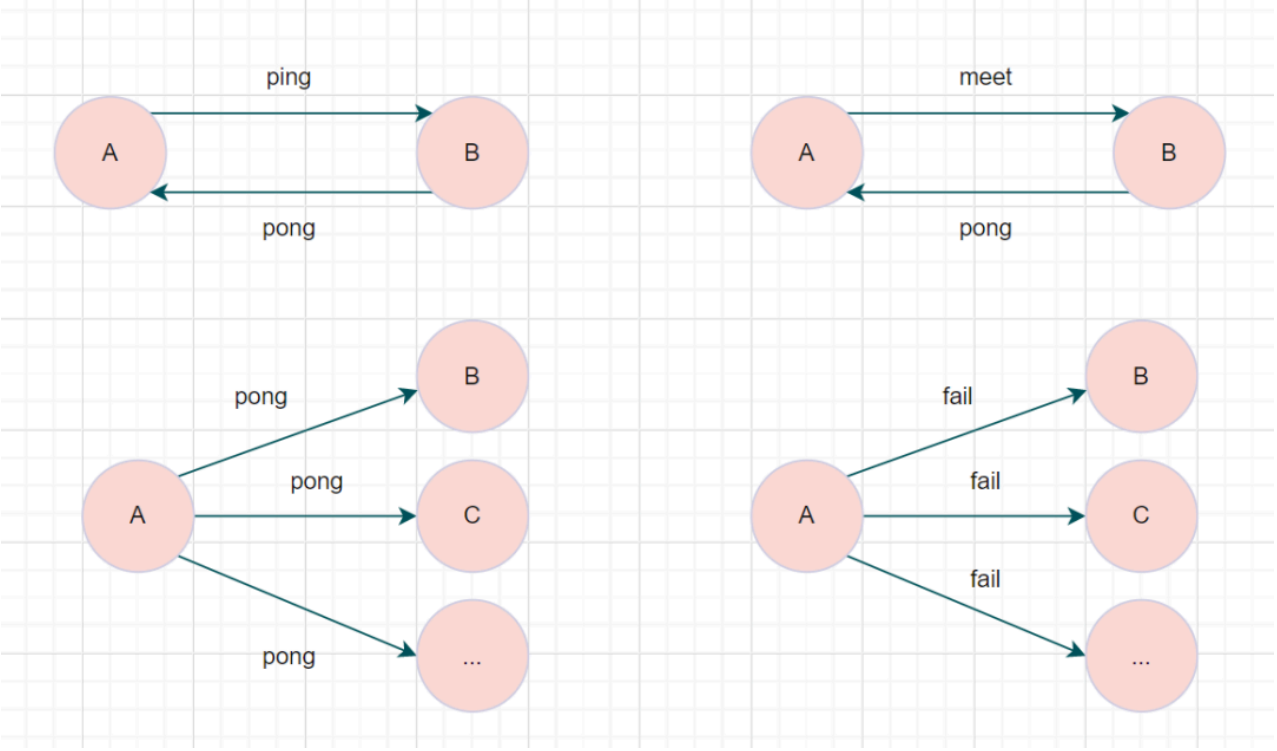


4. 各个节点之间是怎么通信的呢(Gossip)

一个Redis集群由多个节点组成，各个节点之间是怎么通信的呢？通过Gossip协议！Gossip是一种谣言传播协议，每个节点周期性地从节点列表中选择 k 个节点，将本节点存储的信息传播出去，直到所有节点信息一致，即算法收敛了。

> Gossip协议基本思想：一个节点想要分享一些信息给网络中的其他的一些节点。于是，它周期性的随机选择一些节点，并把信息传递给这些节点。这些收到信息的节点接下来会做同样的事情，即把这些信息传递给其他一些随机选择的节点。一般而言，信息会周期性的传递给N个目标节点，而不只是一个。这个N被称为fanout

Redis Cluster集群通过Gossip协议进行通信，节点之前不断交换信息，交换的信息内容包括节点出现故障、新节点加入、主从节点变更信息、slot信息等等。gossip协议包含多种消息类型，包括ping，pong，meet，fail等等



- meet消息：通知新节点加入。消息发送者通知接收者加入到当前集群，meet消息通信正常完成后，接收节点会加入到集群中并进行周期性的ping、pong消息交换。
- ping消息：节点每秒会向集群中其他节点发送 ping 消息，消息中带有自己已知的两个节点的地址、槽、状态信息、最后一次通信时间等
- pong消息：当接收到ping、meet消息时，作为响应消息回复给发送方确认消息正常通信。消息中同样带有自己已知的两个节点信息。
- fail消息：当节点判定集群内另一个节点下线时，会向集群内广播一个fail消息，其他节点接收到fail消息之后把对应节点更新为下线状态。

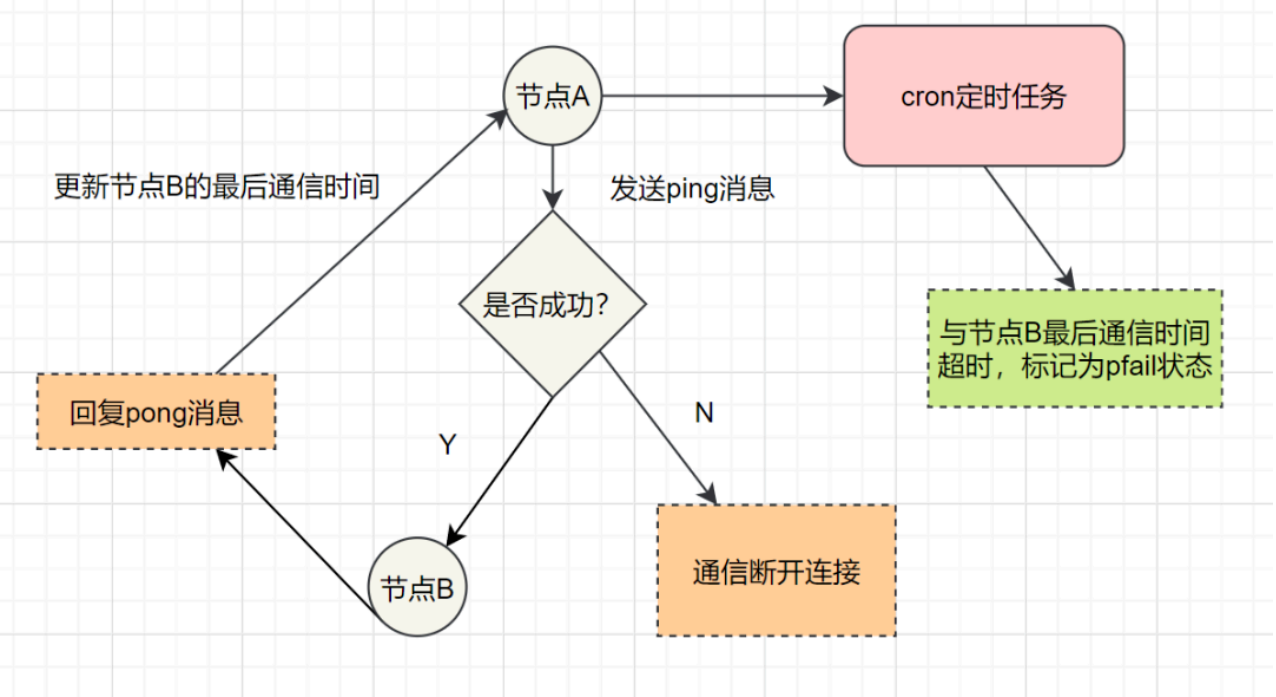
> 特别的，每个节点是通过集群总线(cluster bus) 与其他的节点进行通信的。通讯时，使用特殊的端口号，即对外服务端口号加10000。例如如果某个node的端口号是6379，那么它与其它nodes通信的端口号是 16379。nodes 之间的通信采用特殊的二进制协议。

5. 集群内节点出现故障怎么办（故障转移）

Redis集群实现了高可用，当集群内节点出现故障时，通过**故障转移**，以保证集群正常对外提供服务。

redis集群通过ping/pong消息，实现故障发现。这个环境包括**主观下线和客观下线。**

> **主观下线**：某个节点认为另一个节点不可用，即下线状态，这个状态并不是最终的故障判定，只能代表一个节点的意见，可能存在误判情况。

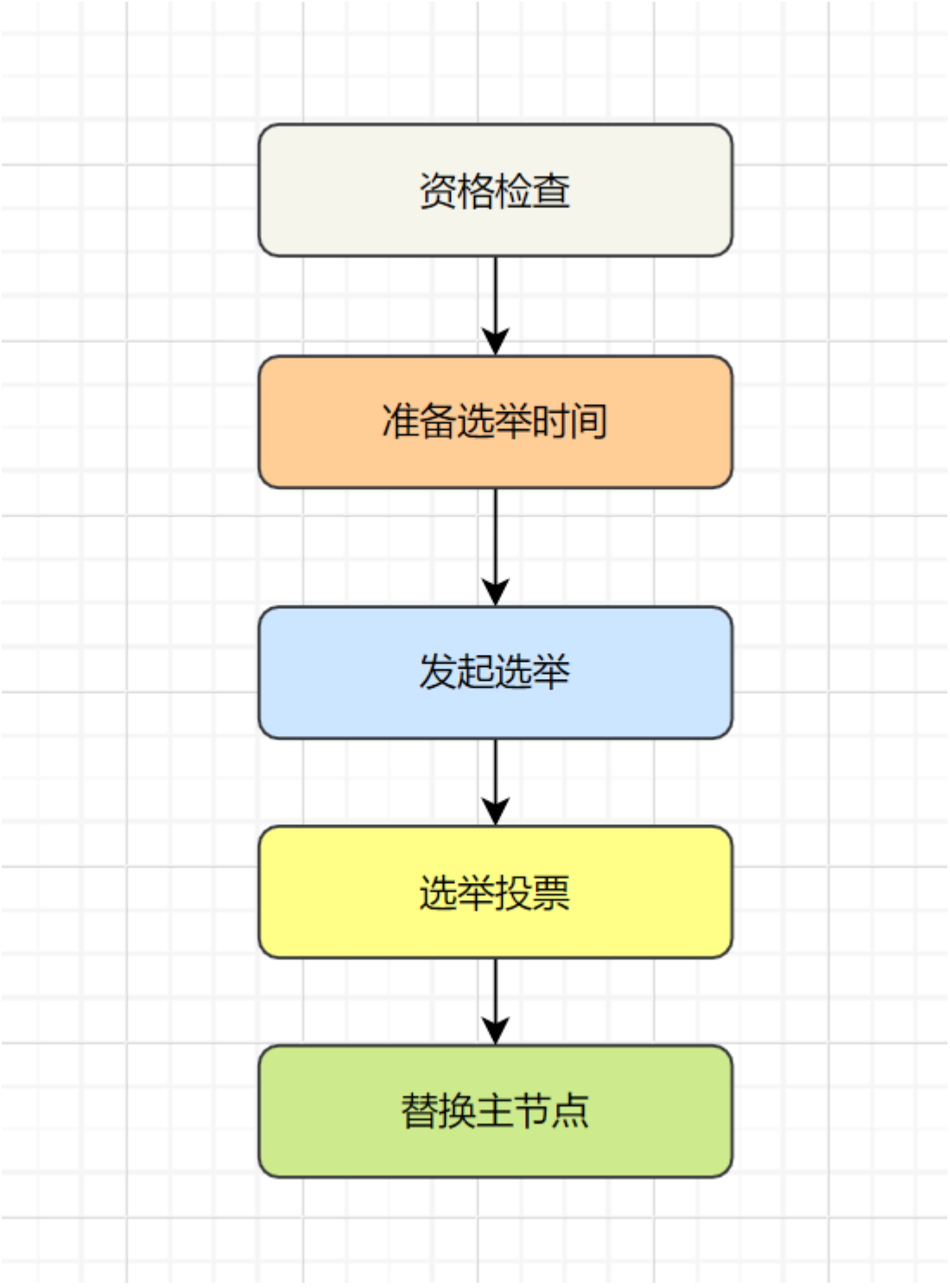


> **客观下线**：指标记一个节点真正的下线，集群内多个节点都认为该节点不可用，从而达成共识的结果。如果是持有槽的主节点故障，需要为该节点进行故障转移。

- 假如节点A标记节点B为主观下线，一段时间后，节点A通过消息把节点B的状态发到其它节点，当节点C接受到消息并解析出消息体时，如果发现节点B的pfail状态时，会触发客观下线流程；
- 当下线为主节点时，此时Redis Cluster集群为统计持有槽的主节点投票，看投票数是否达到一半，当下线报告统计数大于一半时，被标记为客观下线状态。

流程如下：

> 故障恢复：故障发现后，如果下线节点的是主节点，则需要在它的从节点中选一个替换它，以保证集群的高可用。流程如下：

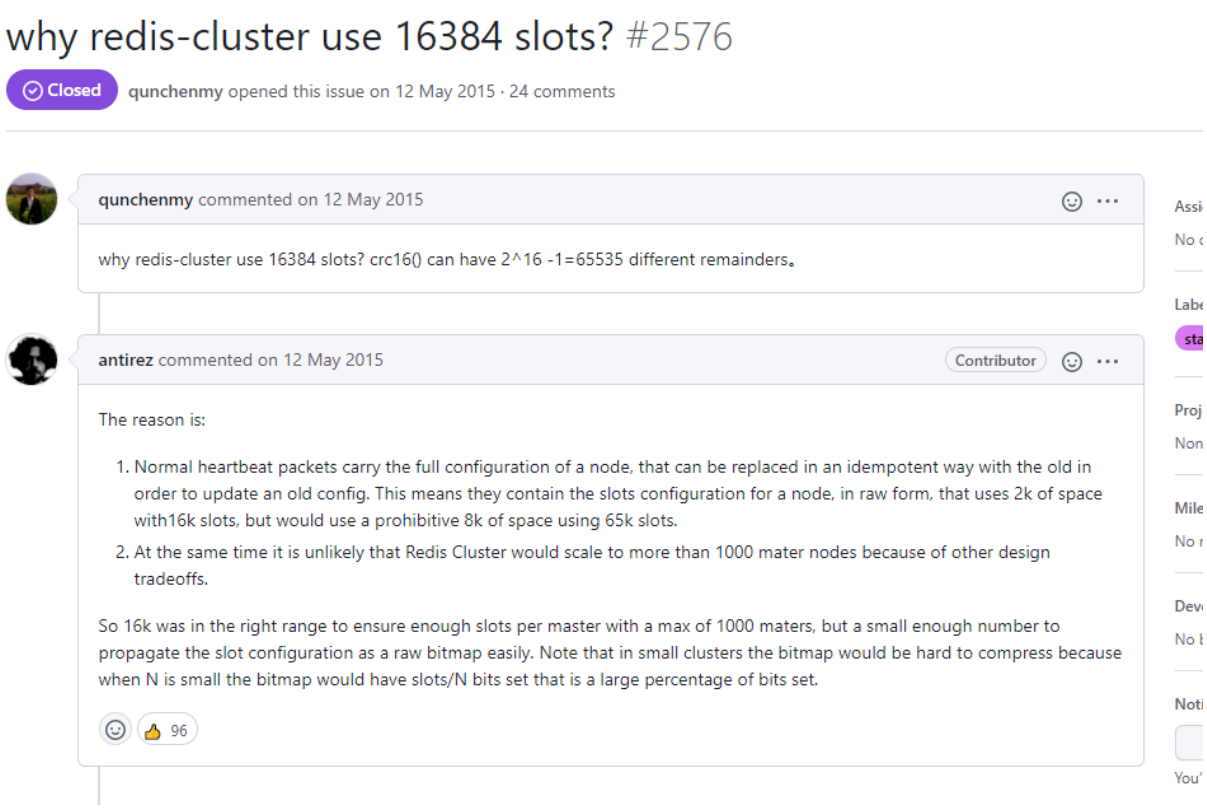


> - 资格检查：检查从节点是否具备替换故障主节点的条件。
> - 准备选举时间：资格检查通过后，更新触发故障选举时间。
> - 发起选举：到了故障选举时间，进行选举。
> - 选举投票：只有持有槽的主节点才有票，从节点收集到足够的选票（大于一半），触发替换主节点操

6. 加餐：为什么Redis Cluster的Hash Slot 是16384？

对于客户端请求过来的键值key，哈希槽=CRC16(key) % 16384，CRC16算法产生的哈希值是16bit的，按道理该算法是可以产生2^16=65536个值，**为什么不用65536，用的是16384（2^14）**呢？

大家可以看下作者的原始回答：



why redis-cluster use 16384 slots? #2576

Closed qunchenmy opened this issue on 12 May 2015 · 24 comments

qunchenmy commented on 12 May 2015

why redis-cluster use 16384 slots? crc16() can have 2^16 -1=65535 different remainders。

antirez commented on 12 May 2015 Contributor

The reason is:

1. Normal heartbeat packets carry the full configuration of a node, that can be replaced in an idempotent way with the old in order to update an old config. This means they contain the slots configuration for a node, in raw form, that uses 2k of space with16k slots, but would use a prohibitive 8k of space using 65k slots.
2. At the same time it is unlikely that Redis Cluster would scale to more than 1000 mater nodes because of other design tradeoffs.

So 16k was in the right range to ensure enough slots per master with a max of 1000 maters, but a small enough number to propagate the slot configuration as a raw bitmap easily. Note that in small clusters the bitmap would be hard to compress because when N is small the bitmap would have slots/N bits set that is a large percentage of bits set.

96

Redis 每个实例节点上都保存对应有哪些slots，它是一个`unsigned char slots[REDIS_CLUSTER_SLOTS/8]`类型

```
typedef struct clusterNode {
    mstime_t ctime; /* Node object creation time. */
    char name[REDIS_CLUSTER_NAMELEN]; /* Node name, hex string, sha1-size */
    int flags;      /* REDIS_NODE_... */
    uint64_t configEpoch; /* Last configEpoch observed for this node */
    unsigned char slots[REDIS_CLUSTER_SLOTS/8]; /* slots handled by this node */
    int numslots;   /* Number of slots handled by this node */
    int numslaves;  /* Number of slave nodes, if this is a master */
    struct clusterNode **slaves; /* pointers to slave nodes */
    struct clusterNode *slaveof; /* pointer to the master node */
    mstime_t ping_sent;      /* Unix time we sent latest ping */
    mstime_t pong_received;  /* Unix time we received the pong */
    mstime_t fail_time;      /* Unix time when FAIL flag was set */
    mstime_t voted_time;     /* Last time we voted for a slave of this master */
    mstime_t repl_offset_time;  /* Unix time we received offset for this node */
    long long repl_offset;      /* Last known repl offset for this node. */
    char ip[REDIS_IP_STR_LEN];  /* Latest known IP address of this node */
    int port;                   /* Latest known port of this node */
    clusterLink *link;          /* TCP/IP link with this node */
    list *fail_reports;         /* List of nodes signaling this as failing */
} clusterNode;
```

- 在redis节点发送心跳包时需要把所有的槽放到这个心跳包里，如果slots数量是65536，占空间= `65536 / 8(一个字节8bit) / 1024(1024个字节1kB)` =8kB，如果使用slots数量是16384，所占空间 =`16384 / 8(每个字节8bit) / 1024(1024个字节1kB)` = 2kB，可见16384个slots比 65536省 6kB内存左右，假如一个集群有100个节点,那每个实例里就省了600kB啦

- 一般情况下Redis cluster集群主节点数量基本不可能超过1000个，超过1000会导致网络拥堵。对于节点数在1000以内的Redis cluster集群，16384个槽位其实够用了。

既然为了节省内存网络开销，为什么 slots不选择用8192（即16384/2）呢?

> `8192 / 8(每个字节8bit) / 1024(1024个字节1kB)` = 1kB,只需要1KB！可以先看下 Redis 把 Key 换算成所属 slots 的方法

```
unsignedintkeyHashSlot(char*key,intkeylen){
ints,e;/*start-endindexesof{and}*/

for(s=0;s<keylen;s++)
if(key[s]=='{')break;

/*No'{'?Hashthewholekey.Thisisthebasecase.*/
if(s==keylen)returncrc16(key,keylen)&0x3FFF;

/*'{'found?Checkifwehavethecorresponding'}'.*/
for(e=s+1;e<keylen;e++)
if(key[e]=='}')break;

/*No'}'ornothingbetweeen{}?Hashthewholekey.*/
if(e==keylen||e==s+1)returncrc16(key,keylen)&0x3FFF;

/*Ifweareherethereisbotha{anda}onitsright.Hash
*whatisinthemiddlebetween{and}.*/
returncrc16(key+s+1,e-s-1)&0x3FFF;
}
```

Redis 将key换算成slots 的方法：其实就是是将`crc16(key)` 之后再和slots的数量进行与计算

这里为什么用`0x3FFF(16383)`来计算,而不是`16384`呢？因为**在不产生溢出的情况下`x % (2^n)等价于x & (2^n - 1)`，即`x % 16384 == x & 16383`**

那到底为什么不用8192呢？

> crc16 出来结果,理论上出现重复的概率为 `1/65536`,但实际结果重复概率可能比这个大不少，就像crc32 结果 理论上 1/40亿 分之一，但实际有人测下来10万碰撞的概率就比较大了。
>
> 假如 slots 设置成 8192，200个实例的节点情况下，理论值是 每40个不同key请求，命中就会失效一次，假如节点数增加到400,那就是20个请求。并且1kb 并不会比 2k 省太多,性价比不是特别高，所以可能选16384会更为通用一点。

7. 总结

- 哨兵模式已经实现了故障自动转移的能力，但业务规模的不断扩展，用户量膨胀，并发量持续提升，会出现了 Redis 响应慢的情况。
- 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，同时也便于横向拓展。在面对千万级甚至亿级别的流量的时候，很多大厂的做法是在千百台的实例节点组成的集群上进行流量调度、服务治理的。
- 整个Redis数据库划分为16384个哈希槽，Redis集群可能有n个实例节点，每个节点可以处理0个 到至多 16384 个槽点，这些节点把 16384个槽位瓜分完成。
- Cluster 是具备Master 和 Slave模式，Redis 集群中的每个实例节点都负责一些槽位，节点之间保持TCP通信，当Master发生了宕机， Redis Cluster自动会将对应的Slave节点选为Master，来继续提供服务。
- 客户端能够快捷的连接到服务端，主要是将slots与实例节点的映射关系存储在本地，当需要访问的时候，对key进行CRC16计算后，再对16384 取模得到对应的 Slot 索引，再定位到相应的实例上。实现高效的连接。