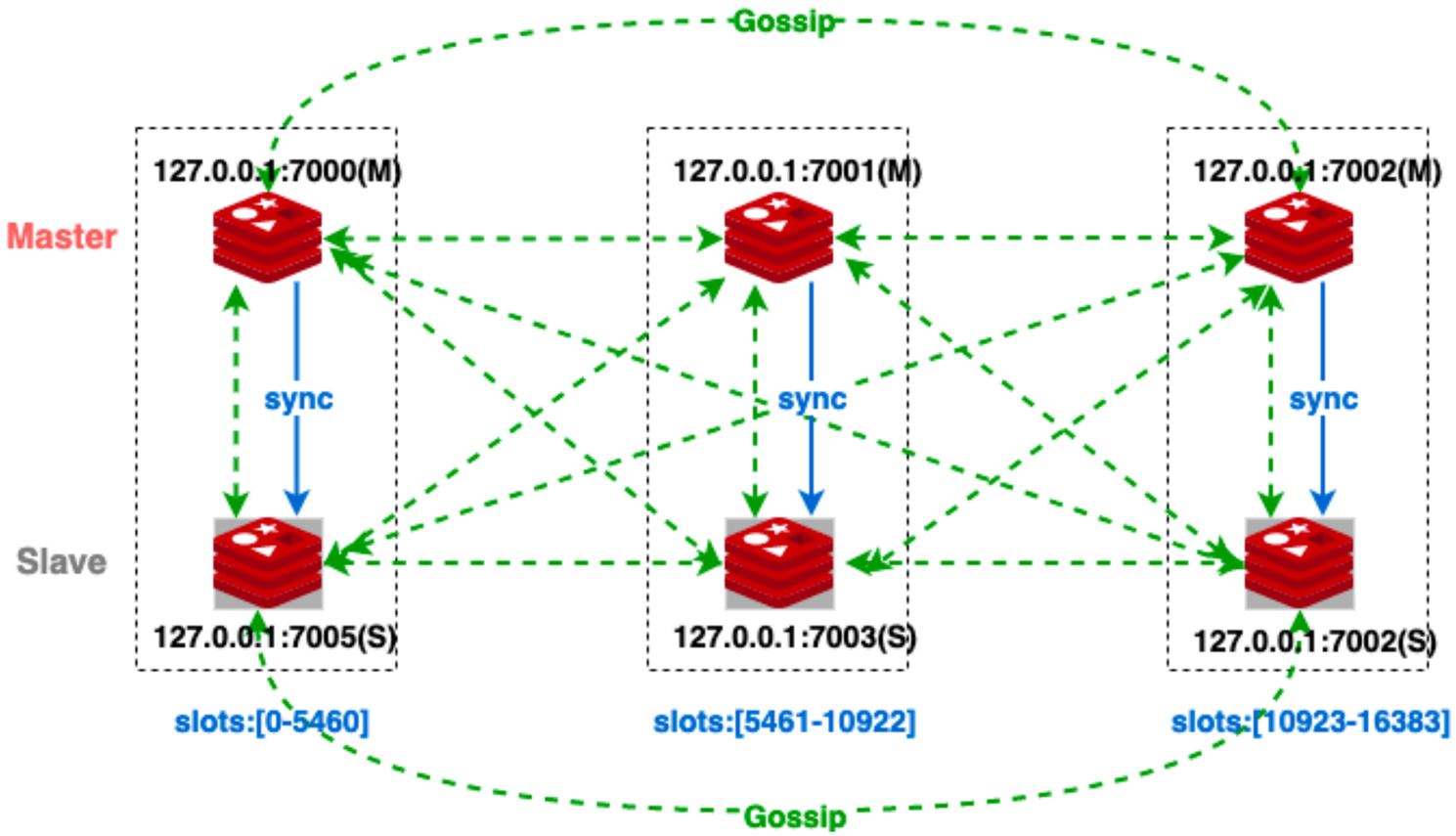
Redis Cluster分享

朱伟 2022-06-22

大纲

- 1.Redis Cluster架构
- 2.Cluster集群创建
- 3.Slot迁移过程
- 4.主从切换

1.Redis Cluster架构



① 主从关系

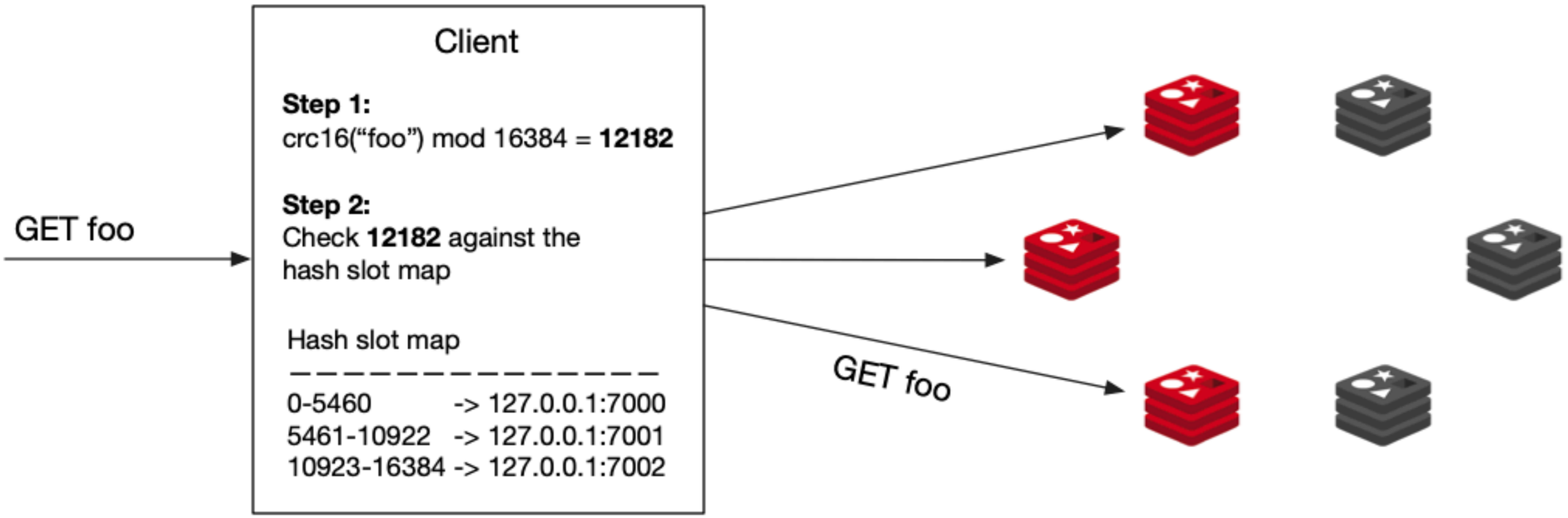
② 数据同步

③ 内部和外部端口

④ Gossip

⑤ Slot

1.Redis Cluster架构



1.Redis Cluster架构

Hash Slot

- Slot数量:16384
- Slot的分布决定了key在Cluster中的分布
- 计算方法 crc16(key) % 16384

```
> CLUSTER KEYSLOT somekey
(integer) 11058
> CLUSTER KEYSLOT foo{hash_tag}
(integer) 2515
> CLUSTER KEYSLOT bar{hash_tag}
(integer) 2515
```

1.Redis Cluster架构

Hash Slot

```
192.168.132.114:6403> cluster slots
1) 1) (integer) 0         // Start slot range
   2) (integer) 2730      // End slot range
   3) 1) "192.168.134.95"  // Redis实例-IP(主)
      2) (integer) 6403   // Redis实例-端口
      3) "702608b2556413c6f8927ef06e86e5c0e869a07d" // node ID
   4) 1) "192.168.134.109"  // Redis实例-IP(从)
      2) (integer) 6403     // Redis实例-端口
      3) "d950357fd1e0010646b32d3397ced3e52b1322fd"
2) 1) (integer) 8192
   2) (integer) 10922
   3) 1) "192.168.134.95"
      2) (integer) 6403
      3) "702608b2556413c6f8927ef06e86e5c0e869a07d"
   4) 1) "192.168.134.109"
      2) (integer) 6403
      3) "d950357fd1e0010646b32d3397ced3e52b1322fd"
3) 1) (integer) 2731
   2) (integer) 8191
   3) 1) "192.168.132.114"
      2) (integer) 6403
      3) "fccdf478a66f2a11713111b5567e3afa2be6c3ab"
   4) 1) "192.168.132.123"
      2) (integer) 6403
      3) "83600e144b488ad324235d84dca9ae6f02928715"
```

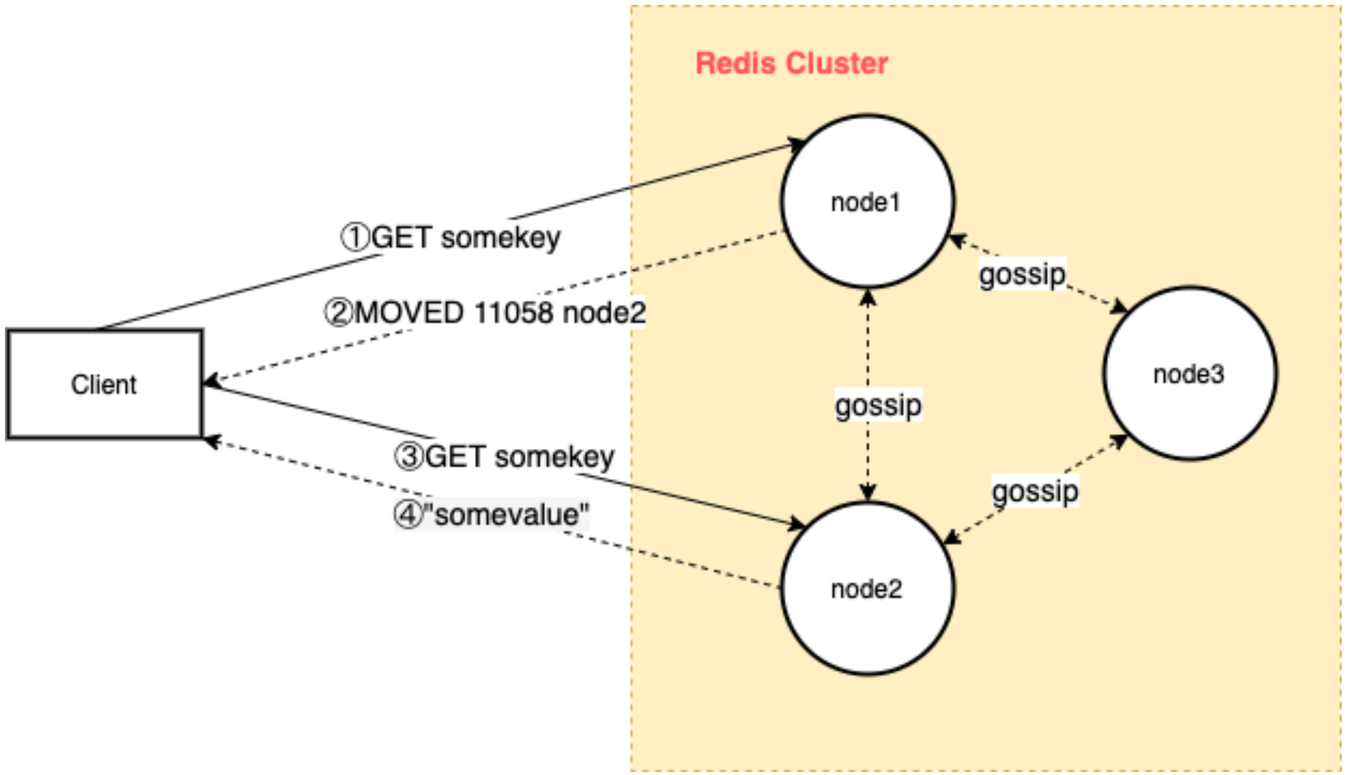
1.Redis Cluster架构

Hash Slot

start	end	node
0	2730	192.168.134.95/192.168.134.109
2731	8191	192.168.132.114/192.168.132.123
8192	10922	192.168.134.95/192.168.134.109
10923	16383	192.168.132.124/192.168.132.128

1.Redis Cluster架构

Hash Slot

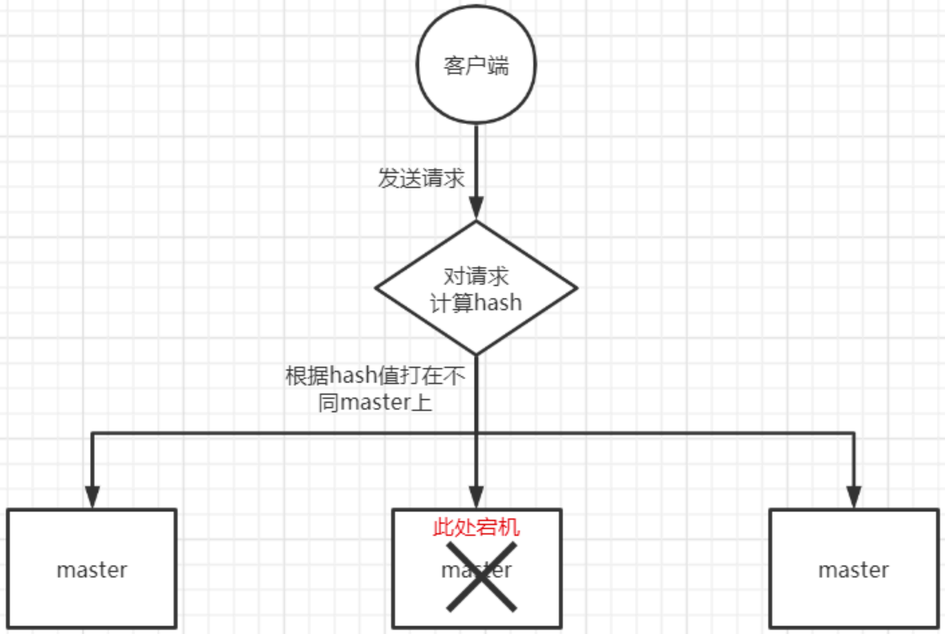



```
192.168.132.114:6403> get a
(error) MOVED 15495 192.168.132.124:6403
```

1.Redis Cluster架构

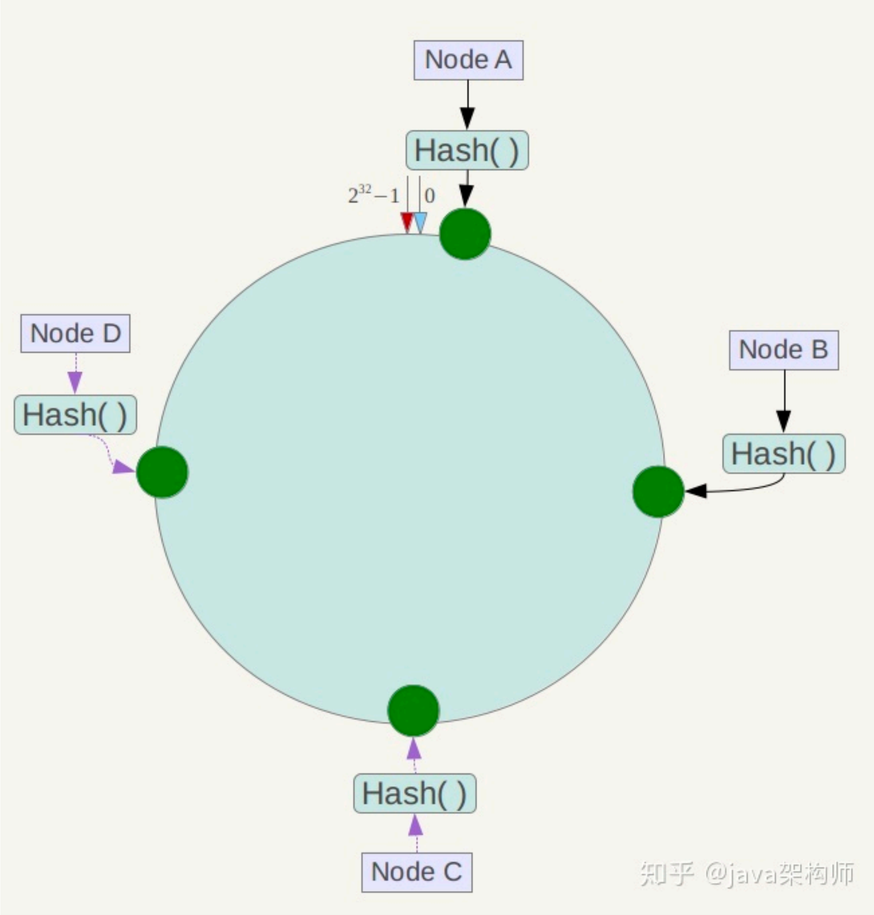
Hash Slot VS 普通Hash

Hash算法：



1.Redis Cluster架构

Hash Slot VS 一致性Hash



可扩展性好。

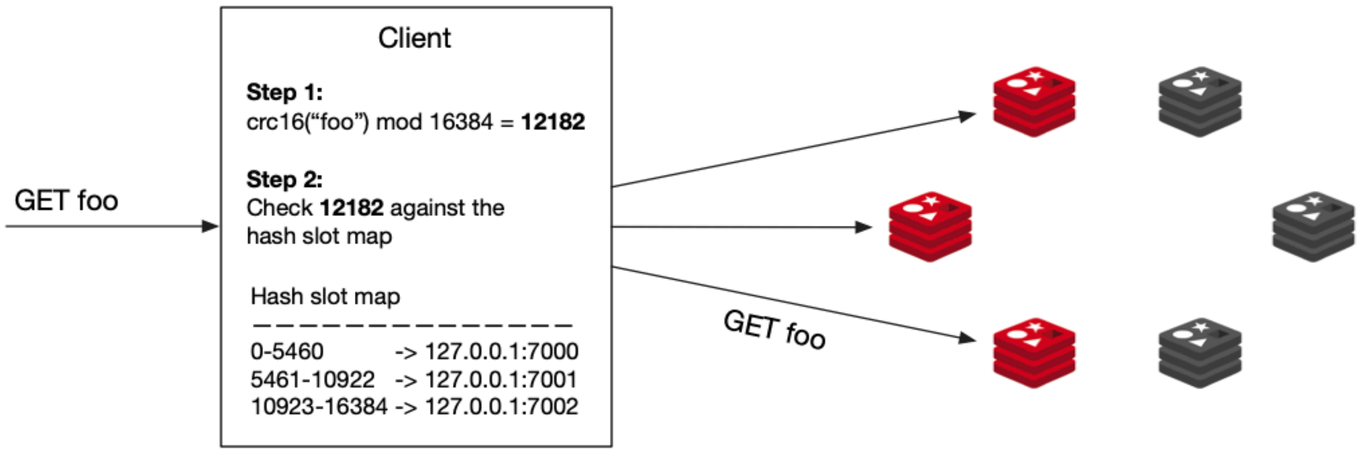
一致性哈希算法保证了**增加或减少服务器**时，

数据存储的改变最少，相比传统哈希算法大大

节省了数据移动的开销。

1.Redis Cluster架构

Hash Slot VS 一致性Hash



Hash Slot的优点&特性

- 1.可扩展性好
- 2.元数据(路由信息)维护成本低
- 3.便于人工管理

注意:添加新节点后，不会自动负载均衡

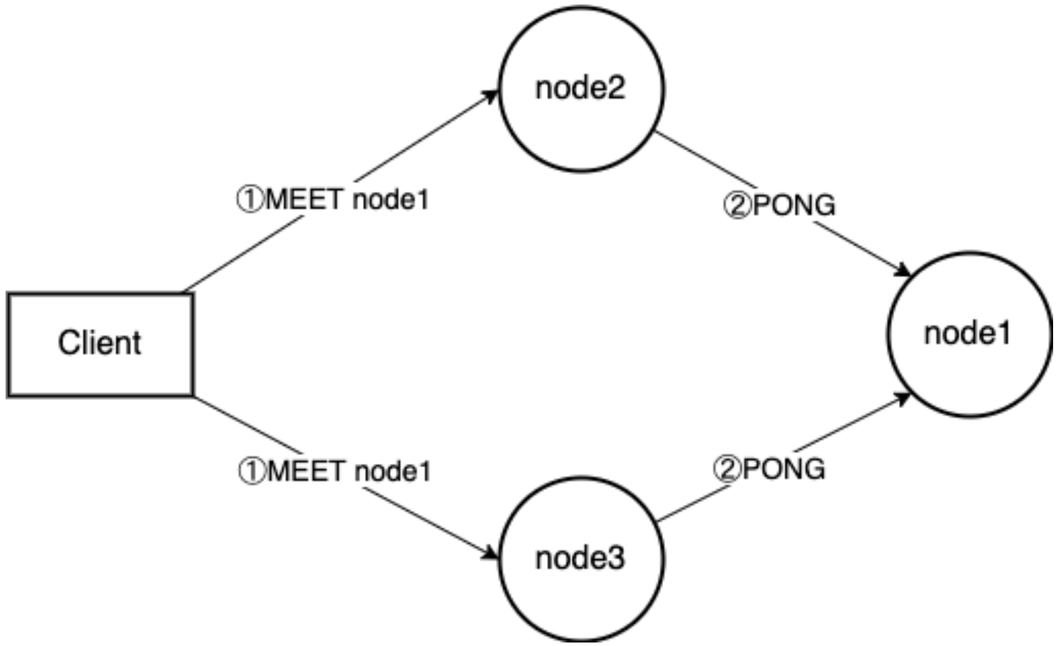
2.Cluster集群创建

```
redis-cli --cluster create 127.0.0.1:7000 127.0.0.1:7001 \
127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005 \
--cluster-replicas 1
```

- 实现节点的相互发现
- 指定主从关系
- 确定slot在集群中的分布

2.Cluster集群创建

2.1相互发现




```
typedef struct {
    uint64_t currentEpoch;  /* The epoch accordingly to the sending node. */
    uint64_t configEpoch;   /* The config epoch if it's a master, or the last
                               epoch advertised by its master if it is a
                               slave. */
    uint64_t offset;    /* Master replication offset if node is a master or
                           processed replication offset if node is a slave. */
    unsigned char myslots[CLUSTER_SLOTS/8];
    char slaveof[CLUSTER_NAMELEN];  // 如果是从节点，此字段为对应主节点的ID
    char myip[NET_IP_STR_LEN];    /* Sender IP, if not all zeroed. */
    union clusterMsgData data;    // 集群中的节点信息
} clusterMsg;
```

2.Cluster集群创建

2.2指定主从关系

客户端下发指令CLUSTER REPLICATE，要求Slave与对应的Master进行数据同步

```
CLUSTER REPLICATE node-id
```

2.Cluster集群创建

2.2确定slot在集群中的分布

客户端下发指令CLUSTER ADDSLOTS, 告知每个Master节点，它需要负责的slot

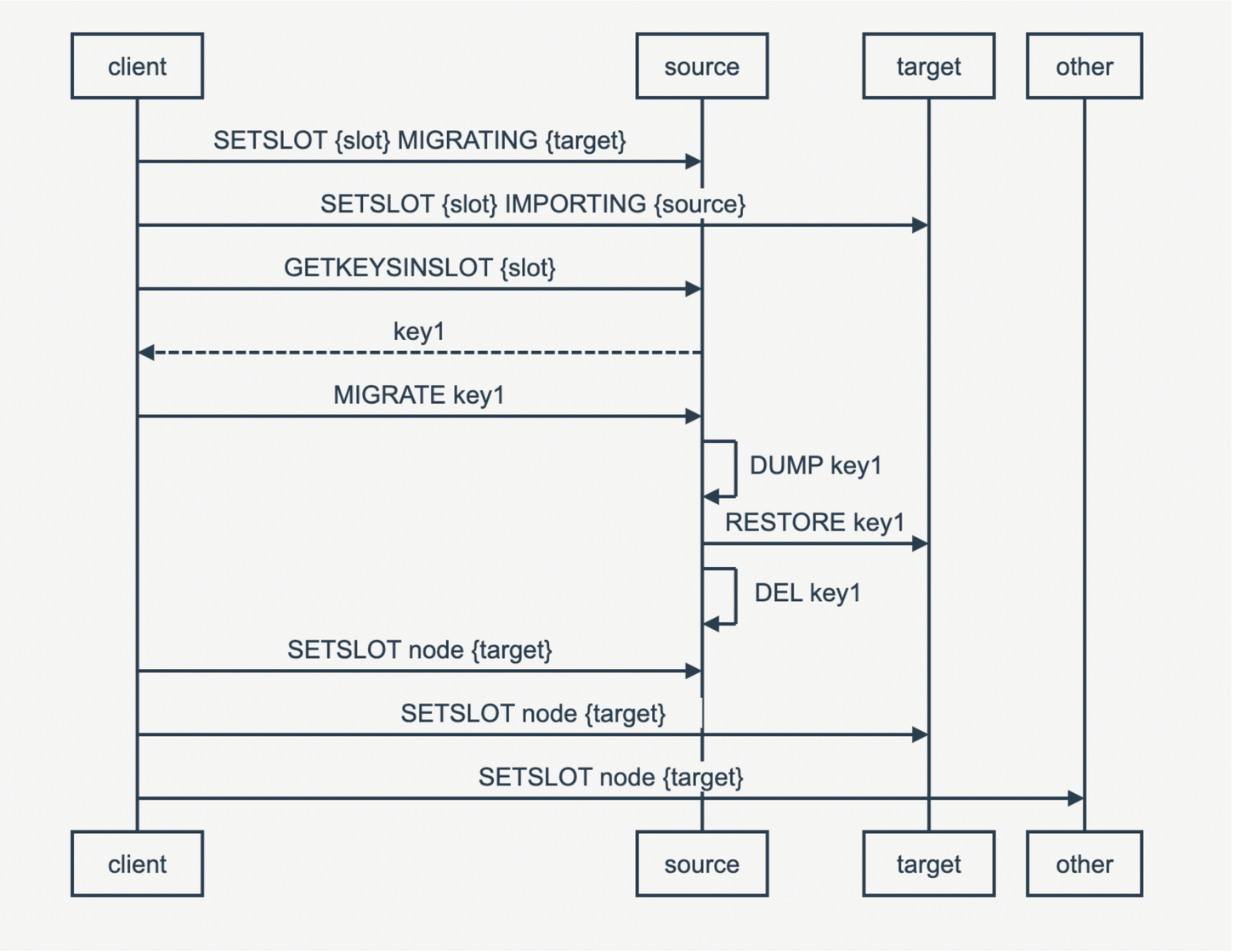
```
CLUSTER ADDSLOTS slot [slot ...]
```

3.Slot迁移过程

```
root@BJ-02:~/cluster-test/7007# redis-cli --cluster reshard
127.0.0.1:7000
>>> Performing Cluster Check (using node 127.0.0.1:7000)
S: 7eb7ceb4d886580c6d122e7fd92e436594cc105e 127.0.0.1:7000
   slots: (0 slots) slave
   replicates be905740b96469fc6f20339fc9898e153c06d497
M: 86f3cb72813a2d07711b56b3143ff727911f4e1e 127.0.0.1:7006
   slots: (0 slots) master
   1 additional replica(s)
M: be905740b96469fc6f20339fc9898e153c06d497 127.0.0.1:7005
   slots:[0-5460] (5461 slots) master
   1 additional replica(s)
S: 9e29dd4b2a7318e0e29a48ae4b37a7bd5ea0a828 127.0.0.1:7007
   slots: (0 slots) slave
   replicates 86f3cb72813a2d07711b56b3143ff727911f4e1e
S: 603a8a403536f625f53467881f5f78def9bd46e5 127.0.0.1:7003
   slots: (0 slots) slave
   replicates 784fa4b720213b0e2b51a4542469f5e318e8658b
M: 4e0e4be1b4afd2cd1d10166a6788449dd812a4c0 127.0.0.1:7002
   slots:[10923-16383] (5461 slots) master
   1 additional replica(s)
S: 585c7df69fb267941a40611bbd8ed90349b49175 127.0.0.1:7004
   slots: (0 slots) slave
   replicates 4e0e4be1b4afd2cd1d10166a6788449dd812a4c0
M: 784fa4b720213b0e2b51a4542469f5e318e8658b 127.0.0.1:7001
   slots:[5461-10922] (5462 slots) master
   1 additional replica(s)
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
How many slots do you want to move (from 1 to 16384)? 0
How many slots do you want to move (from 1 to 16384)? 1
What is the receiving node ID?
86f3cb72813a2d07711b56b3143ff727911f4e1e
Please enter all the source node IDs.
  Type 'all' to use all the nodes as source nodes for the hash slots.
  Type 'done' once you entered all the source nodes IDs.
Source node #1: all

Ready to move 1 slots.
  Source nodes:
    M: be905740b96469fc6f20339fc9898e153c06d497 127.0.0.1:7005
       slots:[0-5460] (5461 slots) master
       1 additional replica(s)
    M: 4e0e4be1b4afd2cd1d10166a6788449dd812a4c0 127.0.0.1:7002
       slots:[10923-16383] (5461 slots) master
       1 additional replica(s)
    M: 784fa4b720213b0e2b51a4542469f5e318e8658b 127.0.0.1:7001
       slots:[5461-10922] (5462 slots) master
       1 additional replica(s)
  Destination node:
    M: 86f3cb72813a2d07711b56b3143ff727911f4e1e 127.0.0.1:7006
       slots: (0 slots) master
       1 additional replica(s)
  Resharding plan:
    Moving slot 5461 from 784fa4b720213b0e2b51a4542469f5e318e8658b
Do you want to proceed with the proposed reshard plan (yes/no)? yes
Moving slot 5461 from 127.0.0.1:7001 to 127.0.0.1:7006:
```

3.Slot迁移过程



- MIGRATE <Key> 在source 上是原子的
- RESTORE <key>在target上是原子的
- 广播状态：CLUSTER SETSLOT <slot> node <destination-node-id>

3.Slot迁移过程

Q:在Slot迁移过程中，如果有key需要写入/读取, 将是怎么样的过程?

如果有新key写入，是不是永远都迁移不完?

3.Slot迁移过程

“somekey”属于Slot[11058]

当前 Slot[11058]正在迁移

127.0.0.1:7002 —> Slot[11058] —> 127.0.0.1:7006

类别	地址
`source-node`	127.0.0.1:7002
`destination-node`	127.0.0.1:7006
无关节点	127.0.0.1:7000

Case 1

发往某个无关节点的请求

无论读写只会MOVED重定向到 `source-node`

```
127.0.0.1:7000> get somekey
(error) MOVED 11058 127.0.0.1:7002
127.0.0.1:7000> set somekey xxx
(error) MOVED 11058 127.0.0.1:7002
```

3.Slot迁移过程

Case 2

对于发往 `source-node` 的请求，分为2种情况

如果key当前在 `source-node` 上存在则处理并正常返回，

否则使用ASK error重定向到 `destination-node`

```
127.0.0.1:7002> get somekey
(error) ASK 11058 127.0.0.1:7006
127.0.0.1:7002> set somekey abc
(error) ASK 11058 127.0.0.1:7006
```

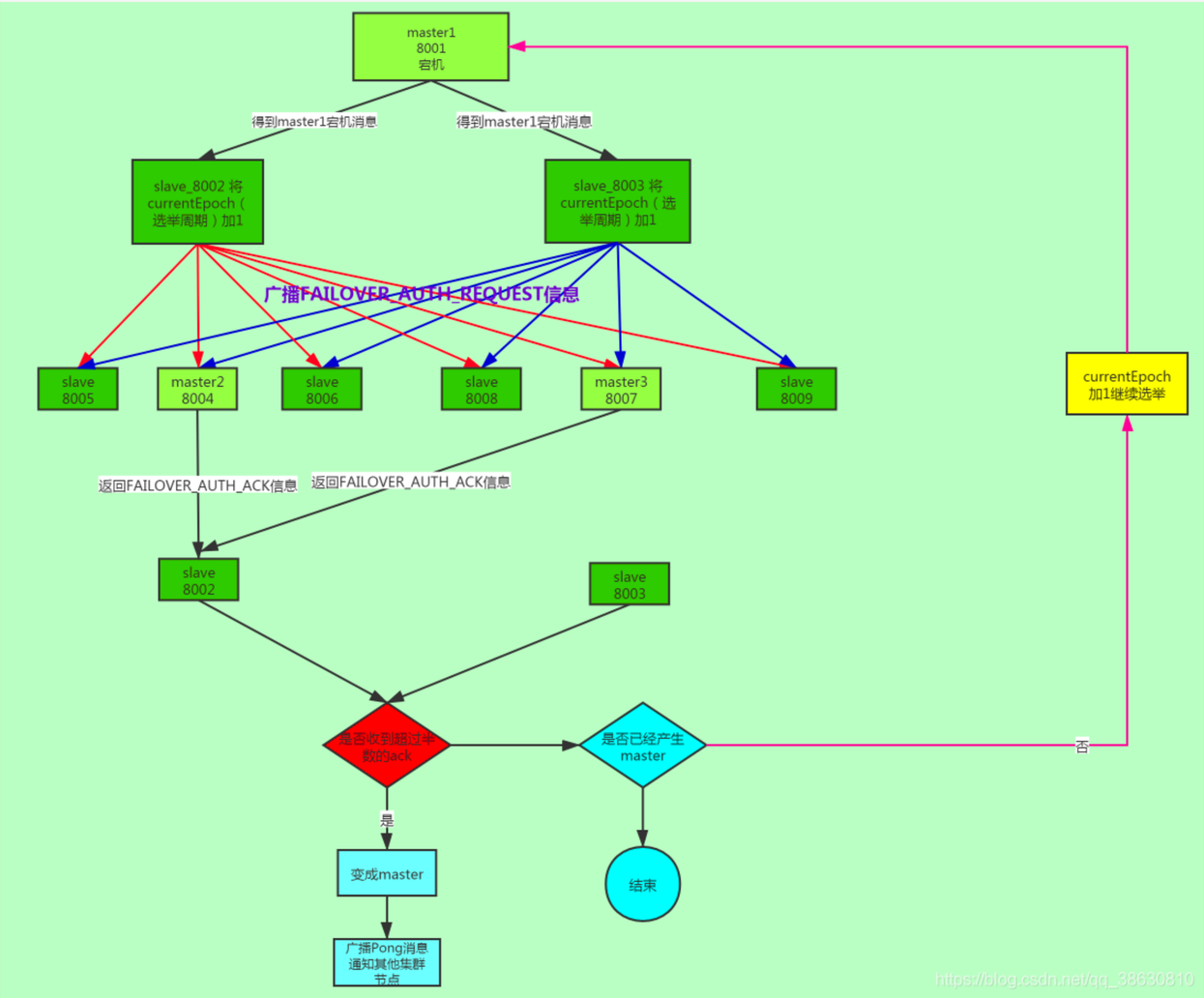
3.Slot迁移过程

Case 3

对于发往 `destination-node` 的请求，它只处理ASKING开头的请求，其余仍然MOVED重定向到 `source-node`

```
 127.0.0.1:7006> get somekey
(error) MOVED 11058 127.0.0.1:7002
127.0.0.1:7006> set somekey "abc"
(error) MOVED 11058 127.0.0.1:7002
127.0.0.1:7006> ASKING
OK
127.0.0.1:7006> set somekey "abc"
OK
127.0.0.1:7006> get somekey
(error) MOVED 11058 127.0.0.1:7002
127.0.0.1:7006> ASKING
OK
127.0.0.1:7006> get somekey
"abc"
```

4.主从切换(failover)



- 主观宕机和客观宕机
- 客观Fail的判定:MasterCount/2 +1
- 获得过半数投票(Master)则当选为Master

4.主从切换(failover)

延迟发起选举投票

延迟计算公式：DELAY = 500ms + random(0 ~ 500ms) + SLAVE_RANK * 1000ms

SLAVE_RANK表示此slave已经从master复制数据的总量的rank。Rank越小代表已复制的数据越新。这种方式下，持有最新数据的slave将会首先发起选举（理论上）。

4.主从切换(failover)

Q:Redis集群为什么至少需要三个master节点

Q&A