即使使用哨兵，此时的 Redis 集群的每个数据库依然存有集群中的所有数据，从而导致集群的总数据存储量受限于所有节点中，内存最小的数据库节点，形成木桶效应。

对 Redis 进行水平扩容，在旧版 Redis 中通常使用客户端分片来解决这个问题，即启动多个 Redis 数据库节点，由客户端决定每个键交由哪个数据库节点存储，下次客户端读取该键时直接到该节点读取。

这种实现将整个数据分布存储在N个数据库节点中，每个节点只存放总数据量的 1/N。但对于需要扩容的场景来说，在客户端分片后，如果想增加更多的节点，就需要对数据进行手工迁移，同时在迁移的过程中为了保证数据的一致性，还需要将集群暂时下线，相对比较复杂。

考虑到Redis实例非常轻量的特点，可以采用预分片技术(presharding)在一定程度上避免此问题，具体来说是在节点部署初期，就提前考虑日后的存储规模，建立足够多的实例（如128个节点）。初期数据很少，所以每个节点存储的数据也非常少，但由于节点轻量的特性，数据之外的内存开销并不大，这使得只需要很少的服务器即可运行这些实例。日后存储规模扩大后，所要做的不过是将某些实例迁移到其他服务器上，而不需要对所有数据进行重新分片并进行集群下线和数据迁移了。

无论如何，客户端分片终归是有非常多的缺点，比如维护成本高，增加、移除节点较繁琐等。Redis 3.0版的一大特性就是支持集群功能。集群的特点在于拥有和单机实例同样的性能。集群支持几乎所有的单机实例支持的命令，对于涉及多键的命令（如MGET），如果每个键都位于同一个节点中，则可以正常支持，否则会提示错误。

除此之外集群还有一个限制是只能使用默认的0号数据库，如果执行 SELECT切换数据库则会提示错误。

**一：配置集群**

使用集群，只需要将每个数据库节点的”cluster-enabled”配置选项打开即可。每个集群中至少需要3个主库才能正常运行。这里以配置一个3主3从的集群系统为例。

首先启动 6 个 Redis 实例，假设6个实例的端口分别是 6380、6381、6382、6383、6384和 6385。集群会将当前节点记录的集群状态保存在指定文件中，这个文件默认为当前工作目录下的nodes.conf文件。每个节点对应的文件必须不同，否则会造成启动失败，所以启动节点时，注意每个节点使用不同的工作目录，或者通过”cluster-config-file”选项修改文件的名称。启动后的效果如下：

# ps -ef|grep redis

root 2747 2544 0 12**:**24 pts**/**4 00**:**00**:**00 redis**-**server **\*:**6380 **[**cluster**]**

root 2819 2803 0 12**:**25 pts**/**1 00**:**00**:**00 redis**-**server **\*:**6381 **[**cluster**]**

root 2905 2875 0 12**:**26 pts**/**3 00**:**00**:**00 redis**-**server **\*:**6382 **[**cluster**]**

root 3010 2961 0 12**:**27 pts**/**6 00**:**00**:**00 redis**-**server **\*:**6383 **[**cluster**]**

root 3099 3066 0 12**:**28 pts**/**8 00**:**00**:**00 redis**-**server **\*:**6384 **[**cluster**]**

root 3181 3155 0 12**:**28 pts**/**9 00**:**00**:**00 redis**-**server **\*:**6385 **[**cluster**]**

每个节点启动后都会输出类似下面的内容：

No cluster configuration found**,** I'm d07331b6438ceb0c5a710c188c01c84548f3f5e9

其中d07331b643...表示该节点的运行ID，运行ID是节点在集群中的唯一标识。启动后，可以使用redis-cli连接任一节点使用info命令来判断集群是否正常启用了：

127.0.0.1**:**6380**>** info cluster

# Cluster

cluster\_enabled**:**1

其中”cluster\_enabled”为1表示集群正常启用了。

现在每个节点都是完全独立的，要将它们加入同一个集群里还需要几个步骤。Redis源代码中提供了一个辅助工具 redis-trib.rb可以方便地完成这一任务。安装运行redis-trib.rb所需要的环境，参考文章《安装运行redis-trib.rb所需的环境》。使用redis-trib.rb来初始化集群，执行一下命令：

**/**root**/**redis**/**redis**-**trib**.**rb create **--**replicas 1 127.0.0.1**:**6380 127.0.0.1**:**6381 127.0.0.1**:**6382 127.0.0.1**:**6383 127.0.0.1**:**6384 127.0.0.1**:**6385

其中 create参数表示要初始化集群，”--replicas 1”表示每个主库拥有的从库个数为1，所以整个集群共有3个主库以及3个从库。执行完后，redis-trib.rb输出如下内容：

# /root/redis/redis-trib.rb create --replicas 1 127.0.0.1:6380 127.0.0.1:6381 127.0.0.1:6382 127.0.0.1:6383 127.0.0.1:6384 127.0.0.1:6385

**>>>** Creating cluster

Connecting to node 127.0.0.1**:**6380**:** OK

Connecting to node 127.0.0.1**:**6381**:** OK

Connecting to node 127.0.0.1**:**6382**:** OK

Connecting to node 127.0.0.1**:**6383**:** OK

Connecting to node 127.0.0.1**:**6384**:** OK

Connecting to node 127.0.0.1**:**6385**:** OK

**>>>** Performing hash slots allocation on 6 nodes**...**

Using 3 masters**:**

127.0.0.1**:**6380

127.0.0.1**:**6381

127.0.0.1**:**6382

Adding replica 127.0.0.1**:**6383 to 127.0.0.1**:**6380

Adding replica 127.0.0.1**:**6384 to 127.0.0.1**:**6381

Adding replica 127.0.0.1**:**6385 to 127.0.0.1**:**6382

M**:** b5a45d4563559c356a4cefff28b8327cdb6a0a0e 127.0.0.1**:**6380

slots**:**0**-**5460 **(**5461 slots**)** master

M**:** 284246d556ca09339a30a5b5b12fadbbdc52b81a 127.0.0.1**:**6381

slots**:**5461**-**10922 **(**5462 slots**)** master

M**:** 88188805dfb1bb3cfa2abbe2dba47a2b6dd7158d 127.0.0.1**:**6382

slots**:**10923**-**16383 **(**5461 slots**)** master

S**:** 1844736ca7cc73bbf9ccdd3b814ad7b9e2b0c36e 127.0.0.1**:**6383

replicates b5a45d4563559c356a4cefff28b8327cdb6a0a0e

S**:** 95eced38f441d7421f438c44729fa92c3040b939 127.0.0.1**:**6384

replicates 284246d556ca09339a30a5b5b12fadbbdc52b81a

S**:** 8ee6175b50b086ba19264f198660c18b40994b64 127.0.0.1**:**6385

replicates 88188805dfb1bb3cfa2abbe2dba47a2b6dd7158d

Can I set the above configuration**?** **(**type 'yes' to accept**):**

内容包括集群具体的分配方案，如果觉得没问题则输入”yes”来开始创建。输入”yes”后，输出内容为：

>>> Nodes configuration updated

>>> Assign a different config epoch to each node

>>> Sending CLUSTER MEET messages to join the cluster

Waiting for the cluster to join......

>>> Performing Cluster Check (using node 127.0.0.1:6380)

M: b5a45d4563559c356a4cefff28b8327cdb6a0a0e 127.0.0.1:6380

slots:0-5460 (5461 slots) master

M: 284246d556ca09339a30a5b5b12fadbbdc52b81a 127.0.0.1:6381

slots:5461-10922 (5462 slots) master

M: 88188805dfb1bb3cfa2abbe2dba47a2b6dd7158d 127.0.0.1:6382

slots:10923-16383 (5461 slots) master

M: 1844736ca7cc73bbf9ccdd3b814ad7b9e2b0c36e 127.0.0.1:6383

slots: (0 slots) master

replicates b5a45d4563559c356a4cefff28b8327cdb6a0a0e

M: 95eced38f441d7421f438c44729fa92c3040b939 127.0.0.1:6384

slots: (0 slots) master

replicates 284246d556ca09339a30a5b5b12fadbbdc52b81a

M: 8ee6175b50b086ba19264f198660c18b40994b64 127.0.0.1:6385

slots: (0 slots) master

replicates 88188805dfb1bb3cfa2abbe2dba47a2b6dd7158d

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

下面根据上面的输出详细介绍集群创建的过程。

首先redis-trib.rb会以客户端的形式尝试连接所有的节点，并发送ping命令以确定节点能够正常服务。如果有任何节点无法连接，则创建失败。然后发送info命令获取每个节点的运行ID以及是否开启了集群功能（即cluster\_enabled为 1）。

准备就绪后，集群会向每个节点发送“cluster meet”命令，格式为：

“cluster meet ip port”，这个命令用来告诉当前节点，在指定ip和port上运行的节点也是集群的一部分，从而使得6个节点最终可以归入一个集群。这一过程会在下面具体介绍。

然后redis-trib.rb会分配主从数据库节点，分配的原则是尽量保证每个主库运行在不同的IP地址上，同时每个从库和主库也不运行在同一IP地址上，以保证系统的容灾能力。分配结果如下：

Using 3 masters**:**

127.0.0.1**:**6380

127.0.0.1**:**6381

127.0.0.1**:**6382

Adding replica 127.0.0.1**:**6383 to 127.0.0.1**:**6380

Adding replica 127.0.0.1**:**6384 to 127.0.0.1**:**6381

Adding replica 127.0.0.1**:**6385 to 127.0.0.1**:**6382

其中主库是 6380、6381 和 6382 端口上的节点，6383是6380的从库，6384 是6381的从库，6385是6382的从库。

分配完成后，会为每个主库分配插槽，分配插槽的过程其实就是分配哪些键归哪些节点负责，这部分会在下面介绍。之后对每个要成为子库的节点发送：

”cluster replicate 主库的运行ID”，来将当前节点转换成从库，并复制指定运行ID 的节点（主库）。

此时整个集群的过程即创建完成，使用redis-cli连接任一节点执行:

“cluster nodes”命令，即可获得集群中的所有节点信息。如在6380执行：

127.0.0.1**:**6380**>** cluster nodes

284246d556ca09339a30a5b5b12fadbbdc52b81a 127.0.0.1**:**6381 master **-** 0 1449387703763 2 connected 5461**-**10922

88188805dfb1bb3cfa2abbe2dba47a2b6dd7158d 127.0.0.1**:**6382 master **-** 0 1449387702753 3 connected 10923**-**16383

1844736ca7cc73bbf9ccdd3b814ad7b9e2b0c36e 127.0.0.1**:**6383 slave b5a45d4563559c356a4cefff28b8327cdb6a0a0e 0 1449387701743 4 connected

8ee6175b50b086ba19264f198660c18b40994b64 127.0.0.1**:**6385 slave 88188805dfb1bb3cfa2abbe2dba47a2b6dd7158d 0 1449387704773 6 connected

b5a45d4563559c356a4cefff28b8327cdb6a0a0e 127.0.0.1**:**6380 myself**,**master **-** 0 0 1 connected 0**-**5460

95eced38f441d7421f438c44729fa92c3040b939 127.0.0.1**:**6384 slave 284246d556ca09339a30a5b5b12fadbbdc52b81a 0 1449387705783 5 connected

从上面的输出中可以看到所有节点的运行ID、地址和端口、角色、状态以及负责的插槽等信息，后文会进行解读。

**二：增加节点**

redis-trib.rb 是使用“cluster meet”命令来使每个节点认识集群中的其他节点的，因此，向集群中加入新的节点，也需要使用该命令实现。

加入新节点非常简单，只需要向新节点（以下记作A）发送如下命令即可：

“cluster meet ip port”，ip和port是集群中任一节点的地址和端口号，A接收到客户端发来的命令后，会与该地址和端口号的节点B进行握手，使B将A认作当前集群中的一员。当 B与A握手成功后，B会使用”Gossip”协议将节点A的信息通知给集群中的每一个节点。

通过这一方式，即使集群中有多个节点，也只需要选择 MEET 其中任一节点，即可使新节点最终加入整个集群中。

**三：插槽的分配**

新的节点加入集群后有两种选择，要么使用”cluster replicate”命令复制某个主库来以从库的形式运行，要么向集群申请分配插槽(slot)来以主库的形式运行。

在一个集群中，所有的键会被分配给16384（16k）个插槽，而每个主库会负责处理其中的一部分插槽。现在再看一下上面创建集群时的输出：

M**:** b5a45d4563559c356a4cefff28b8327cdb6a0a0e 127.0.0.1**:**6380

slots**:**0**-**5460 **(**5461 slots**)** master

M**:** 284246d556ca09339a30a5b5b12fadbbdc52b81a 127.0.0.1**:**6381

slots**:**5461**-**10922 **(**5462 slots**)** master

M**:** 88188805dfb1bb3cfa2abbe2dba47a2b6dd7158d 127.0.0.1**:**6382

slots**:**10923**-**16383 **(**5461 slots**)** master

上面的每一行表示一个主库的信息，其中可以看到6380负责处理[0, 5460]这5461个插槽，6381负责处理[5461, 10922]这5462个插槽，6382则负责处理[10923, 16383]这5461个插槽。

虽然redis-trib.rb初始化集群时分配给每个节点的插槽都是连续的，但是实际上Redis并没有此限制，可以将任意的几个插槽分配给任意的节点负责。

在介绍如何将插槽分配给指定的节点前，先来介绍键与插槽的对应关系。Redis 将每个键的键名的有效部分使用CRC16算法计算出散列值，然后对16384的取余。这样使得每个键都可以分配到16384个插槽中，进而分配到指定的一个节点中处理。

CRC16的具体实现参阅相关资料。这里键名的有效部分是指：

a：如果键名包含”{“符号，且在”{“符号后面存在”}”符号，且”{“和”}”之间有至少一个字符，则有效部分是指”{“和”}”之间的内容；

b：如果不满足上一条规则，那么整个键名为有效部分。

例如，键”hello.world”的有效部分为"hello.world"，键”{user102}:last.name”的有效部分为"user102"。

之前说过，如果命令涉及多个键（如MGET），只有当所有键位于同一个节点时 Redis才能支持。因此，利用键的分配规则，可以将所有相关的键的有效部分设置成同样的值，使得相关键都能分配到同一个插槽以支持多键操作。比如，”{user102}:first.name”和”{user102}:last.name”就会被分配到同一个节点。

接下来介绍如何将插槽分配给指定节点。可以通过命令”cluster slots”来查看插槽的分配情况，如：

127.0.0.1**:**6380**>** cluster slots

1**)** 1**)** **(**integer**)** 5461

2**)** **(**integer**)** 10922

3**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6381

4**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6384

2**)** 1**)** **(**integer**)** 10923

2**)** **(**integer**)** 16383

3**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6382

4**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6385

3**)** 1**)** **(**integer**)** 0

2**)** **(**integer**)** 5460

3**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6380

4**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6383

返回结果的格式很容易理解，一共3条记录，每条记录的前两个值表示插槽的开始号码和结束号码，后面的值则为负责该插槽的节点，包括主库和所有的从库，主库始终在第一位。

插槽的分配分为如下几种情况：

a：插槽之前没有被分配过，现在想分配给指定节点。

b：插槽之前被分配过，现在想移动到指定节点。

其中第一种情况使用”cluster addslots”命令来实现，redis-trib.rb也是通过该命令在创建集群时为新节点分配插槽的。该命令的用法为：

“cluster addslots slot1 [slot2] ... [slotN]”。如想将100和101两个插槽分配给某个节点，只需要在该节点执行：“cluster addslots 100 101”即可。

如果指定插槽已经分配过了，则会提示：

**(**error**)** ERR Slot 100 is already busy

第二种情况处理起来就相对复杂一些，不过redis-trib.rb提供了比较方便的方式来对插槽进行迁移。下面首先使用redis-trib.rb将一个插槽从6380迁移到6381，然后再介绍如何不使用redistrib.rb来完成迁移。

首先执行如下命令：

# /root/redis/redis-trib.rb reshard 127.0.0.1:6380

其中reshard表示告诉redis-trib.rb要重新分片，127.0.0.1:6380是集群中的任一节点的地址和端口，redis-trib.rb会自动获取集群信息：

root@localhost**:~**# **/root/redis/**redis**-**trib**.**rb reshard 127.0.0.1**:**6380

Connecting to node 127.0.0.1**:**6380**:** OK

Connecting to node 127.0.0.1**:**6381**:** OK

Connecting to node 127.0.0.1**:**6382**:** OK

Connecting to node 127.0.0.1**:**6383**:** OK

Connecting to node 127.0.0.1**:**6385**:** OK

Connecting to node 127.0.0.1**:**6384**:** OK

**>>>** Performing Cluster Check **(**using node 127.0.0.1**:**6380**)**

M**:** b5a45d4563559c356a4cefff28b8327cdb6a0a0e 127.0.0.1**:**6380

slots**:**0**-**5460 **(**5461 slots**)** master

1 additional replica**(**s**)**

M**:** 284246d556ca09339a30a5b5b12fadbbdc52b81a 127.0.0.1**:**6381

slots**:**5461**-**10922 **(**5462 slots**)** master

1 additional replica**(**s**)**

M**:** 88188805dfb1bb3cfa2abbe2dba47a2b6dd7158d 127.0.0.1**:**6382

slots**:**10923**-**16383 **(**5461 slots**)** master

1 additional replica**(**s**)**

S**:** 1844736ca7cc73bbf9ccdd3b814ad7b9e2b0c36e 127.0.0.1**:**6383

slots**:** **(**0 slots**)** slave

replicates b5a45d4563559c356a4cefff28b8327cdb6a0a0e

S**:** 8ee6175b50b086ba19264f198660c18b40994b64 127.0.0.1**:**6385

slots**:** **(**0 slots**)** slave

replicates 88188805dfb1bb3cfa2abbe2dba47a2b6dd7158d

S**:** 95eced38f441d7421f438c44729fa92c3040b939 127.0.0.1**:**6384

slots**:** **(**0 slots**)** slave

replicates 284246d556ca09339a30a5b5b12fadbbdc52b81a

**[**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**)?**

接下来，redistrib.rb将会询问具体如何进行重新分片，首先会询问想要迁移多少个插槽，我们只需要迁移一个，所以输入1后回车。

接下来redis-trib.rb会询问要把插槽迁移到哪个节点，可以通过”cluster nodes”命令获取6381的运行ID，这里是284246d556ca09339a30a5b5b12fadbbdc52b81a，输入并回车。

接着最后一步是询问从哪个节点移出插槽，输入6380对应的运行ID，这里是b5a45d4563559c356a4cefff28b8327cdb6a0a0e，回车后输入”done”再按回车确认即可。

接下来输入yes来确认重新分片方案，重新分片即告成功：

How many slots **do** you want to move **(**from 1 to 16384**)?** 1

What is the receiving node ID**?** 284246d556ca09339a30a5b5b12fadbbdc52b81a

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**:**b5a45d4563559c356a4cefff28b8327cdb6a0a0e

Source node #2**:**done

Ready to move 1 slots**.**

Source nodes**:**

M**:** b5a45d4563559c356a4cefff28b8327cdb6a0a0e 127.0.0.1**:**6380

slots**:**0**-**5460 **(**5461 slots**)** master

1 additional replica**(**s**)**

Destination node**:**

M**:** 284246d556ca09339a30a5b5b12fadbbdc52b81a 127.0.0.1**:**6381

slots**:**5461**-**10922 **(**5462 slots**)** master

1 additional replica**(**s**)**

Resharding plan**:**

Moving slot 0 from b5a45d4563559c356a4cefff28b8327cdb6a0a0e

Do you want to proceed with the proposed reshard plan **(**yes**/**no**)?** yes

Moving slot 0 from 127.0.0.1**:**6380 to 127.0.0.1**:**6381**:**

然后用”cluster slots”命令，查看当前插槽的分配情况如下：

127.0.0.1**:**6380**>** cluster slots

1**)** 1**)** **(**integer**)** 0

2**)** **(**integer**)** 0

3**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6381

4**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6384

2**)** 1**)** **(**integer**)** 5461

2**)** **(**integer**)** 10922

3**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6381

4**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6384

3**)** 1**)** **(**integer**)** 10923

2**)** **(**integer**)** 16383

3**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6382

4**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6385

4**)** 1**)** **(**integer**)** 1

2**)** **(**integer**)** 5460

3**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6380

4**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6383

可见，现在比之前多了一条记录，第0号插槽已经由6381负责，此时重新分片成功。

那么redis-trib.rb实现重新分片的原理是什么，如何不借助redis-trib.rb手工进行重新分片呢？使用如下命令即可：

”cluster setslot 插槽号 node 新节点的运行ID”，比如要把0号插槽迁移回6380：

127.0.0.1**:**6380**>** cluster setslot 0 node b5a45d4563559c356a4cefff28b8327cdb6a0a0e

OK

然而这样迁移插槽的前提是插槽中并没有任何键（实际上，即使该slot上没有键，也需要分别在源节点和目标节点上都运行一次上面的命令，才能生效。），因为使用” cluster setslot”命令迁移插槽时，并不会连同相应的键一起迁移，为此需要手工获取插槽中存在哪些键，然后将每个键迁移到新的节点中才行。

手工获取某个插槽存在哪些键的方法是：

”cluster getkeysinslot 插槽号 要返回的键的数量”，之后对每个键，使用”migrate”命令将其迁移到目标节点：

”migrate ip port key 数据库号码 超时时间 [COPY] [REPLACE]”，用ip和port表示目标节点，key表示要迁移的键名，因为集群模式只能使用0号数据库，所以数据库号码始终为0，超时时间表示与目标交互过程中，最大的阻塞时间（并非意味着非得在timeout时间内完成，只表示交互时，阻塞时间的最大值），单位是毫秒。copy选项表示不将键从当前数据库中删除，而是复制一份，replace表示如果目标节点存在同名键，则覆盖。比如，要把键abc从当前节点（如6381）迁移到6380：

127.0.0.1**:**6381**>** migrate 127.0.0.1 6380 abc 0 15999 REPLACE

至此，已经知道如何将插槽委派给其他节点，并同时将当前节点中插槽下所有的键迁移到目标节点中。

然而，上面的命令并不能执行成功（redis-3.0.5），这是因为还有一个问题，如果slot中的数据量比较大，整个过程会花费较长时间，那么究竟在什么时候执行”cluster setslot”命令来完成插槽的交接呢？

如果在键迁移未完成时执行，那么客户端就会尝试在新的节点读取键值，此时还没有迁移完成，自然有可能读不到键值，从而造成相关键的临时“丢失”。相反，如果在键迁移完成后再执行，那么在迁移时客户端会在旧的节点读取键值，然而有些键已经迁移到新的节点上了，同样也会造成键的临时“丢失”。

Redis提供了如下两个命令用来实现在集群不下线的情况下迁移数据：

“cluster setslot 插槽号 migrating 新节点的运行ID”

“cluster setslot 插槽号 importing 原节点的运行ID”

假设要把0号插槽从A迁移到B，此时redis-trib.rb会依次执行如下操作：

a：在B执行” cluster setslot 0 importing A”；

b：在A 执行” cluster setslot 0 migrating B”；

c：在A 执行 “cluster getkeysinslot 0 1”获取0号插槽的键列表；

d：在A中，对第3步获取的每个键执行migrate命令，将其从A迁移到B；

e：在A或B中，执行”cluster setslot 0 node B”来完成迁移。

从上面的步骤来看， redis-trib.rb多了a和 b两个步骤，这两个步骤就是为了解决迁移过程中键的临时“丢失”问题。首先执行完前两步后，当客户端向 A 请求插槽 0 中的键时，如果键存在（即尚未被迁移），则正常处理，如果不存在，则返回一个 ASK跳转请求，告诉客户端这个键在 B里，如下图所示：



客户端接收到ask跳转请求后，首先向B发送asking命令，然后再重新发送之前的命令。

当客户端向 B请求插槽 0 中的键时，如果前面执行了 asking命令，则返回键值内容，否则返回 moved跳转请求（后续介绍），如下图所示：



这样一来客户端只要能够处理ask跳转，则可以在数据库迁移时自动从正确的节点获取到相应的键值，避免了键在迁移过程中临时“丢失”的问题。

实际的例子如下：在6381节点有一个名为”hehe”的key，它对应的slot是4618，现在，想将其迁移到6380节点中。首先直接在6381节点执行migrate命令，得到错误结果：

127.0.0.1**:**6381**>** migrate 127.0.0.1 6380 hehe 0 15999 replace

**(**error**)** ERR Target instance replied with error**:** MOVED 4618 127.0.0.1**:**6381

可见，直接这样迁移是不行的。所以，按照上面介绍的步骤来，首先是在6380节点执行importing，然后分别在6380和6381节点执行”get hehe”：

127.0.0.1**:**6380**>** cluster setslot 4618 importing 284246d556ca09339a30a5b5b12fadbbdc52b81a

OK

127.0.0.1**:**6380**>** get hehe

**(**error**)** MOVED 4618 127.0.0.1**:**6381

127.0.0.1**:**6381**>** get hehe

"hello"

其中，284246d556ca09339a30a5b5b12fadbbdc52b81a就是6381节点的运行ID。此时执行”get hehe”命令，得到的还是往常的结果。

然后，在6381节点执行migrating，然后分别在6380和6381节点执行”get hehe”：

127.0.0.1**:**6381**>** cluster setslot 4618 migrating b5a45d4563559c356a4cefff28b8327cdb6a0a0e

OK

127.0.0.1**:**6381**>** get hehe

"hello"

127.0.0.1**:**6380**>** get hehe

**(**error**)** MOVED 4618 127.0.0.1**:**6381

其中b5a45d4563559c356a4cefff28b8327cdb6a0a0e就是6380节点的运行ID。此时执行”get hehe”命令，得到的还是往常的结果。

然后，在6381节点上执行” migrate”命令：

127.0.0.1**:**6381**>** migrate 127.0.0.1 6380 hehe 0 15999 replace

OK

127.0.0.1**:**6381**>** get hehe

**(**error**)** ASK 4618 127.0.0.1**:**6380

127.0.0.1**:**6380**>** get hehe

**(**error**)** MOVED 4618 127.0.0.1**:**6381

127.0.0.1**:**6380**>** asking

OK

127.0.0.1**:**6380**>** get hehe

"hello"

在6381上执行完”migrate”后，此时要获取”hehe”，就会返回ASK，说明”hehe”已经不在本节点中了。在6380上直接执行”get hehe”，返回的还是MOVED错误，只有执行了”asking”命令后，”get hehe”才会返回正确的结果。

最后，在6381或者6380节点上执行”node”命令，完成迁移：

127.0.0.1**:**6380**>** cluster setslot 4618 node b5a45d4563559c356a4cefff28b8327cdb6a0a0e

OK

127.0.0.1**:**6381**>** cluster slots

1**)** 1**)** **(**integer**)** 5461

2**)** **(**integer**)** 10922

3**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6381

4**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6384

2**)** 1**)** **(**integer**)** 10923

2**)** **(**integer**)** 16383

3**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6382

4**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6385

3**)** 1**)** **(**integer**)** 0

2**)** **(**integer**)** 5460

3**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6380

4**)** 1**)** "127.0.0.1"

2**)** **(**integer**)** 6383

可见，4618这个slot已经在6380中了。

**四：获取与插槽对应的节点**

对于指定的键，可以根据前文所述的算法来计算其属于哪个插槽，但是如何获取某一个键由哪个节点负责呢？

当客户端向集群中的任意一个节点发送命令后，该节点会判断相应的键是否在当前节点中，如果键在该节点中，则会像单机实例一样正常处理该命令；如果键不在该节点中，就会返回一个move 重定向请求，告诉客户端这个键目前由哪个节点负责，然后客户端再将同样的请求向目标节点重新发送一次以获得结果。

一些语言的 Redis 库支持代理move请求，所以对于开发者而言命令重定向的过程是透明的，使用集群与使用单机实例并没有什么不同。然而也有些语言的 Redis 库并不支持集群，这时就需要在客户端编码处理了。

以上面的集群配置为例，键foo应该由6382节点负责，如果尝试在6380节点执行与键foo相关的命令，就会有如下输出：

127.0.0.1**:**6380**>** set foo bar

**(**error**)** MOVED 12182 127.0.0.1**:**6382

返回的是一个move重定向请求，12182表示foo所属的插槽号，127.0.0.1:6382则是负责该插槽的节点地址和端口，客户端收到重定向请求后，应该将命令重新向 6382节点发送一次：

127.0.0.1**:**6382**>** set foo bar

OK

redis-cli提供了集群模式来支持自动重定向，使用”-c”参数来启用：

# redis-cli -p 6380 -c

127.0.0.1**:**6380**>** get foo

**->** Redirected to slot **[**12182**]** located at 127.0.0.1**:**6382

"hello"

可见加入了-c参数后，如果当前节点并不负责要处理的键，redis-cli会进行自动命令重定向。而这一过程正是每个支持集群的客户端应该实现的。

相比单机实例，集群的命令重定向也增加了命令的请求次数，原先只需要执行一次的命令现在有可能需要依次发向两个节点，算上往返时延，可以说请求重定向对性能的还是有些影响的。

为了解决这一问题，当发现新的重定向请求时，客户端应该在重新向正确节点发送命令的同时，缓存插槽的路由信息，即记录下当前插槽是由哪个节点负责的。这样每次发起命令时，客户端首先计算相关键是属于哪个插槽的，然后根据缓存的路由判断插槽由哪个节点负责。考虑到插槽总数相对较少（16384个），缓存所有插槽的路由信息后，每次命令将均只发向正确的节点，从而达到和单机实例同样的性能。

**五：故障恢复**

在一个集群中，每个节点都会定期向其他节点发送 ping 命令，并通过有没有收到回复来判断目标节点是否已经下线了。

具体来说，集群中的每个节点每隔1秒钟就会随机选择5个节点，然后选择其中最久没有响应的节点发送ping命令。如果一定时间内目标节点没有响应回复，则发起ping命令的节点会认为目标节点疑似下线(PFAIL)。疑似下线可以与哨兵的主观下线类比，两者都表示，某一节点从自身的角度认为目标节点是下线的状态。与哨兵的模式类似，如果要使在整个集群中的所有节点都认为某一节点已经下线，需要一定数量的节点都认为该节点疑似下线才可以，这一过程具体为：

a：一旦节点A认为节点B是疑似下线状态，就会在集群中传播该消息，所有其他节点收到消息后都会记录下这一信息；

b：当集群中的某一节点C收集到半数以上的节点认为B是疑似下线的状态时，就会将B标记为下线（FAIL），并且向集群中的其他节点传播该消息，从而使得B在整个集群中下线。

在集群中，当一个主库下线时，就会出现一部分插槽无法写入的问题。这时如果该主库拥有至少一个从库，集群就进行故障恢复操作，来将其中一个从库转变成主库，来保证集群的完整。选择哪个从库来作为主库的过程与在哨兵中选择领头哨兵的过程一样，都是基于Raft算法，过程如下。

a：发现其复制的主库下线的从库（下面称作A）向每个集群中的节点发送请求，要求对方选自己成为主库。

b：如果收到请求的节点没有选过其他人，则会同意将A设置成主库。

c：如果A发现有超过集群中节点总数一半的节点同意选自己成为主库，则A成功成为主库。

d：当有多个从库节点同时参选主库，则会出现没有任何节点当选的可能。此时每个参选节点将等待一个随机时间重新发起参选请求，进行下一轮选举，直到选举成功。

当某个从库当选为主库后，会通过命令”slaveof no one”将自己转换成主库，并将旧的主库的插槽转换给自己负责。如果一个至少负责一个插槽的主库下线且没有相应的从库可以进行故障恢复，则整个集群默认会进入下线状态无法继续工作。如果想在这种情况下使集群仍能正常工作，可以修改配置”cluster-require-full-coverage”为”no”（默认为”yes”）。