Redis-4.0.11 集群配置

一见 2015/12/1

版本：redis-3.0.5 redis-3.2.0 redis-3.2.9 redis-4.0.11

参考：http://redis.io/topics/cluster-tutorial。

集群部署交互式命令行工具：https://github.com/eyjian/redis-tools/tree/master/deploy

集群运维命令行工具：https://github.com/eyjian/redis-tools/tree/master

目录

1. 前言

本文参考官方文档而成：http://redis.io/topics/cluster-tutorial。经测试，安装过程也适用于 redis-3.2.0、redis-4.0.11 等。

Redis 运维工具和部署工具：https://github.com/eyjian/redis-tools。

2. 部署计划

依据官网介绍，部署 6 个 redis 节点，为 3 主 3 从。3 台物理机每台都创建 2 个 redis 节点：

服务端口	IP 地址	配置文件名
6379	192.168.0.251	redis-6379.conf
6379	192.168.0.252	redis-6379.conf
6379	192.168.0.253	redis-6379.conf
6380	192.168.0.251	redis-6380.conf
6380	192.168.0.252	redis-6380.conf
6380	192.168.0.253	redis-6380.conf

疑问：3 台物理机，会不会主和从节点分布在同一个物理机上？

3. 目录结构

redis.conf 为从 https://raw.githubusercontent.com/antirez/redis/3.0/redis.conf 下载的配置文件。redis-6379.conf 和 redis-6380.conf 指定了服务端口，两者均通过 include 复用（包含）了 redis.conf。

本文将 redis 安装在**/data/redis**（每台机器完全相同，同一台机器上的多个节点对应相同的目录和文件，并建议将 bin 目录加入到环境变量 PATH 中，以简化后续的使用）：

```
/data/redis
|-- bin
|     |-- redis-benchmark
|     |-- redis-check-aof
|     |-- redis-check-dump
|     |-- redis-cli
|     |-- redis-sentinel -> redis-server
|     `-- redis-server
|-- conf
|     |-- redis-6379.conf
|     |-- redis-6380.conf
|     `-- redis.conf
`-- log

3 directories, 9 files
```

4. 编译安装

打开 redis 的 Makefile 文件，可以看到如下内容：

```
PREFIX?=/usr/local
INSTALL_BIN=$(PREFIX)/bin
INSTALL=install
```

Makefile 中的“?=”表示，如果该变量之前没有定义过，则赋值为/usr/local，否则什么也不做。

如果不设置环境变量 PREFIX 或不修改 Makefile 中的值，则默认安装到/usr/local/bin 目录下。建议不要使用默认配置，而是指定安装目录，如/data/redis-3.0.5：

```
$ make
$ make install PREFIX=/data/redis-3.0.5
$ ln -s /data/redis-3.0.5 /data/redis
$ mkdir /data/redis/conf
$ mkdir /data/redis/log
$ mkdir /data/redis/data
```

5. 修改系统参数

5.1. 修改最大可打开文件数

修改文件/etc/security/limits.conf，加入以下两行：

```
* soft nofile 102400
```

```
* hard nofile 102400

# End of file
```

其中 **102400** 为一个进程最大可以打开的文件个数，当与 RedisServer 的连接数多时，需要设定为合适的值。

有些环境修改后，root 用户需要重启机器才生效，而普通用户重新登录后即生效。如果是 crontab，则需要重启 crontab，如：service crond restart，有些平台可能是 service cron restart。

有些环境下列设置即可让 root 重新登录即生效，而不用重启机器：

```
root soft nofile 102400
root hard nofile 102400

# End of file
```

但是要**小心**，有些环境上面这样做，可能导致无法 ssh 登录，所以在修改时最好打开两个窗口，万一登录不了还可自救。

如何确认更改对一个进程生效？按下列方法（其中$PID 为被查的进程 ID）：

```
$ cat /proc/$PID/limits
```

系统关于/etc/security/limits.conf 文件的说明：

```
#This file sets the resource limits for the users logged in via PAM.
#It does not affect resource limits of the system services.
```

PAM：全称“Pluggable Authentication Modules”，中文名“插入式认证模块”。/etc/security/limits.conf 实际为 **pam_limits.so**（位置：/lib/security/pam_limits.so）的配置文件，只针对单个会话。要使用 limits.conf 生效，必须保证 pam_limits.so 被加入到了启动文件中。

注释说明只对通过 **PAM** 登录的用户生效，与 PAM 相关的文件（均位于/etc/**pam.d** 目录下）：

```
/etc/pam.d/login
/etc/pam.d/sshd
/etc/pam.d/crond
```

如果需要设置 Linux 用户的密码策略，可以修改文件/etc/login.defs，但这个只对新增的用户有效，如果要影响已有用户，可使用命令 chage。

5.2. TCP 监听队列大小

即 TCP listen 的 backlog 大小，“/proc/sys/net/core/**somaxconn**”的默认值一般较小如 128，需要修改大一点，比如改成 32767。立即生效还可以使用命令：**sysctl -w** net.core.somaxconn=32767。

要想永久生效，需要在文件/etc/sysctl.conf 中增加一行：net.core.somaxconn = 32767，然后执行命令“sysctl -p”以生效。

Redis 配置项 **tcp-backlog** 的值不能超过 **somaxconn** 的大小。

5.3. OOM 相关：vm.overcommit_memory

如果“/proc/sys/vm/overcommit_memory”的值为 0，则会表示开启了 OOM。可以设置为 1 关闭 OOM，设置方法请参照 net.core.somaxconn 完成。

5.4. /sys/kernel/mm/transparent_hugepage/enabled

默认值为“[always] madvise never”，建议设置为 never，以开启内核的“Transparent Huge Pages (**THP**)”特性，设置后 redis 进程需要重启。为了永久生效，请将“echo never > /sys/kernel/mm/transparent_hugepage/enabled”加入到文件/etc/rc.local 中。

什么是 Transparent Huge Pages？为提升性能，通过大内存页来替代传统的 4K 页，使用得管理虚拟地址数变少，加快从虚拟地址到物理地址的映射，以及摒弃内存页面的换入换出以提高内存的整体性能。内核 Kernel 将程序缓存内存中，每页内存以 2M 为单位。相应的系统进程为 khugepaged。

在 Linux 中，有两种方式使用 Huge Pages，一种是 2.6 内核引入的 HugeTLBFS，另一种是 2.6.36 内核引入的 THP。HugeTLBFS 主要用于数据库，THP 广泛应用于应用程序。

一般可以在 rc.local 或/etc/default/grub 中对 Huge Pages 进行设置。

6. 配置 redis

从 https://raw.githubusercontent.com/antirez/redis/3.0/redis.conf 下载配置文件（也可直接复制源代码包中的 redis.conf，然后在它的基础上进行修改），在这个基础上，进行如下表所示的修改（配置文件名 redis-**6379**.conf 中的 **6379** 建议设置为实际使用的端口号）：

配置项	值	配置文件	说明
include	redis.conf	redis-6379.conf	引用公共的配置文件，建议为全路径值
port	6379		客户端连接端口，并且总有一个刚好大于 10000 的端口，这个大的端口用于主从复制和集群内部通讯。
cluster-config-file	nodes-6379.conf		**默认放在 dir 指定的目录**
pidfile	/var/run/redis-6379.pid		只有当 daemonize 值为 yes 时，才有意义；并且这个要求对目录/var/run 有写权限，否则可以考虑设置为/tmp/redis-6379.pid。
dir	/data/redis/data/6379		
dbfilename	dump-6379.rdb		位于 dir 指定的目录下
appendonly	yes		
appendfilename	"appendonly-6379.aof"		

logfile	/data/redis/log/redis-6379.log		日志文件，包含目录和文件名，注意 redis 不会自动滚动日志文件
include	redis.conf	redis-6380.conf	引用公共的配置文件
port	6380		
cluster-config-file	nodes-6380.conf		**默认放在 dir 指定的目录**
pidfile	/var/run/redis-6380.pid		
dir	/data/redis/data/6380		AOF 和 RDB 文件存放目录
dbfilename	dump-6380.rdb		RDB 文件名
appendfilename	appendonly-6380.aof		AOF 文件名
logfile	/data/redis/log/redis-6380.log		
loglevel	verbose	redis.conf（公共配置文件）	日志级别，建议为 notice，另外注意 redis 是不会滚动日志文件的，每次写日志都是先打开日志文件再写日志再关闭方式
maxclients	**10000**		最大连接数
timeout	**0**		客户端多长（秒）时间没发包过来关闭它，0 表示永不关闭
cluster-enabled	**yes**		表示以集群方式运行，为 no 表示以非集群方式运行
cluster-node-timeout	15000		单位为毫秒：repl-ping-slave-period+(**cluster-node-timeout***cluster-slave-validity-factor) 判断节点失效（fail）之前，允许不可用的最大时长（毫秒），如果 master 不可用时长超过此值，则会被 failover。不能太小，建议默认值 15000
cluster-slave-validity-factor	0		如果要最大的可用性，值设置为 0。定义 slave 和 master 失联时长的倍数，如果值为 0，则只要失联 slave 总是尝试 failover，而不管与 master 失联多久。失联最大时长：(cluster-slave-validity-factor*cluster-node-timeout)
repl-timeout	10		该配置项的值要求大于 repl-ping-slave-period 的值
repl-ping-slave-period	1		定义 slave 多久（秒）ping 一次 master，如果超过 repl-timeout 指定的时长都没有收到响应，则

			认为 master 挂了
slave-read-only	**yes**		slave 是否只读
slave-serve-stale-data	**yes**		当 slave 与 master 断开连接，slave 是否继续提供服务
slave-priority	**100**		slave 权重值，当 master 挂掉，只有权重最大的 slave 接替 master
aof-use-rdb-preamble			4.0 新增配置项，用于控制是否启用 RDB-AOF 混用，值为 no 表示关闭
appendonly	yes		当同时写 AOF 或 RDB，则 redis 启动时只会加载 AOF，AOF 包含了全量数据。如果当队列使用，入队压力又很大，建议设置为 no
appendfsync	no		可取值 everysec，其中 no 表示由系统自动，当写压力很大时，建议设置为 no，否则容易造成整个集群不可用
daemonize	yes		相关配置项 pidfile
protected-mode	no		3.2.0 新增的配置项，默认值为 yes，限制从其它机器登录 Redis server，而只能从 127.0.0.1 登录。为保证 redis-trib.rb 工具的正常运行，需要设置为 no，完成后可以改回 yes，但每次使用 redis-trib.rb 都需要改回为 no。要想从非 127.0.0.1 访问也需要改为 no。
tcp-backlog	**32767**		取值不能超过系统的 /proc/sys/net/core/somaxconn
auto-aof-rewrite-percentage	100		设置自动 rewite AOF 文件（手工 rewrite 只需要调用命令 BGREWRITEAOF）
auto-aof-rewrite-min-size	64mb		触发 rewrite 的 AOF 文件大小，只有大于此大小时才会触发 rewrite
no-appendfsync-on-rewrite	yes		子进程在做 rewrite 时，主进程不调用 fsync（由内核默认调度）
stop-writes-on-bgsave-error	yes		如果因为磁盘故障等导致保存 rdb 失败，停止写操作，可设置为 NO。
cluster-require-full-coverage	no		为 no 表示有 slots 不可服务时其它 slots 仍然继续服务
maxmemory	26843545600		设置最大的内存，单位为字节

maxmemory-policy	volatile-lru		设置达到最大内存时的淘汰策略
client-output-buffer-limit			设置 master 端的客户端缓存，三种：normal、slave 和 pubsub
cluster-migration-barrier	1		最少 slave 数，用来保证集群中不会有裸奔的 master。当某个 master 节点的 slave 节点挂掉裸奔后，会从其他富余的 master 节点分配一个 slave 节点过来，确保每个 master 节点都有至少一个 slave 节点，不至于因为 master 节点挂掉而没有相应 slave 节点替换为 master 节点导致集群崩溃不可用。
repl-backlog-size	1mb		当 slave 失联时的，环形复制缓区大小，值越大可容忍更长的 slave 失联时长
repl-backlog-ttl			slave 失联的时长达到该值时，释放 backlog 缓冲区
save	save 900 1 save 300 10 save 60 10000		刷新快照（RDB）到磁盘的策略，根据实际调整值，“save 900 1”表示 900 秒后至少有 1 个 key 被修改才触发 save 操作,其它类推。 注意执行 flushall 命令也会产生 RDB 文件，不过是空文件。 如果不想生成 RDB 文件，可以将 save 全注释掉。

7. 启动 redis 实例

登录 3 台物理机，启动两个 redis 实例（启动之前，需要创建好配置中的各目录）：

1) redis-server redis-**6379**.conf
2) redis-server redis-**6380**.conf

可以写一个启动脚本 start-redis-cluster.sh：

```
#!/bin/sh

REDIS_HOME=/data/redis
$REDIS_HOME/bin/redis-server $REDIS_HOME/conf/redis-6379.conf
$REDIS_HOME/bin/redis-server $REDIS_HOME/conf/redis-6380.conf
```

8. 创建和启动 redis cluster 前的准备工作

上一步启动的 redis 只是单机版本，在启动 redis cluster 之前，需要完成如下一些依赖的安装。在此之后，才可以创建和启动 redis cluster。

8.1. 安装 ruby

安装命令：yum install ruby
安装过程中，如提示“[y/d/N]”，请选“y”然后回车。

查看版本：

```
$ ruby --version
ruby 2.0.0p353 (2013-11-22) [x86_64-linux]
```

也可以从 Ruby 官网 https://www.ruby-lang.org 下载安装包(如 ruby-2.3.1.tar.gz)来安装 Ruby。截至 2016/5/13，Ruby 的最新稳定版本为 Ruby 2.3.1。

8.2. 安装 rubygems

安装命令：yum install rubygems
如果不使用 yum 安装，也可以手动安装 RubyGems，RubyGems 是一个 Ruby 包管理框架，它的下载网址：https://rubygems.org/pages/download。
比如下载安装包 rubygems-2.6.4.zip 后解压，然后进入解压生成的目录，里面有个 setup.rb 文件，以 root 用户执行：ruby setup.rb 安装 RubyGems。

8.3. 安装 redis-3.0.0.gem

安装命令：gem install -l redis-3.0.0.gem

安装之前，需要先下载好 redis-3.0.0.gem。
redis-3.0.0.gem 官网：https://rubygems.org/gems/redis/versions/3.0.0
redis-3.0.0.gem 下载网址：https://rubygems.org/downloads/redis-3.0.0.gem
redis-3.3.0.gem 官网：https://rubygems.org/gems/redis/versions/3.3.0
redis-3.3.3.gem 官网：https://rubygems.org/gems/redis/versions/3.3.3
redis-4.0.1.gem 官网：https://rubygems.org/gems/redis/versions/4.0.1

集群的创建只需要在一个节点上操作，所以只需要在一个节点上安装 redis-X.X.X.gem 即可。

```
# gem install -l redis-3.3.3.gem
Successfully installed redis-3.3.3
Parsing documentation for redis-3.3.3
Installing ri documentation for redis-3.3.3
```

```
Done installing documentation for redis after 1 seconds
1 gem installed
```

9. redis-trib.rb

redis-trib.rb 是 redis 官方提供的 redis cluster 管理工具，使用 ruby 实现。

10. 创建和启动 redis 集群

10.1. 复制 redis-trib.rb

将 redis 源代码的 src 目录下的**集群管理程序 redis-trib.rb** 复制到/data/redis/bin 目录，并将 bin 目录加入到环境变量 PATH 中，以简化后续的操作。

redis-trib.rb 用法（不带任何参数执行 redis-trib.rb 即显示用法）：

```
$ ./redis-trib.rb
Usage: redis-trib <command> <options> <arguments ...>

  rebalance       host:port
                  --auto-weights
                  --timeout <arg>
                  --pipeline <arg>
                  --use-empty-masters
                  --weight <arg>
                  --threshold <arg>
                  --simulate
  add-node        new_host:new_port existing_host:existing_port
                  --slave
                  --master-id <arg>
  reshard         host:port
                  --timeout <arg>
                  --pipeline <arg>
                  --yes
                  --slots <arg>
                  --to <arg>
                  --from <arg>
  check           host:port
  set-timeout     host:port milliseconds
  call            host:port command arg arg .. arg
  fix             host:port
                  --timeout <arg>
```

```
  info            host:port
  create          host1:port1 ... hostN:portN
                  --replicas <arg>
  import          host:port
                  --replace
                  --copy
                  --from <arg>
  help            (show this help)
  del-node        host:port node_id

For check, fix, reshard, del-node, set-timeout you can specify the host and port of any working node in the cluster.
```

10.2. 创建 redis cluster

创建命令（3 主 3 从）：

```
redis-trib.rb create --replicas 1 192.168.0.251:6379 192.168.0.252:6379 192.168.0.253:6379 192.168.0.251:6380 192.168.0.252:6380 192.168.0.253:6380
```

➢ **参数说明：**

1) **create**

表示创建一个 redis cluster 集群。

2) **--replicas 1**

表示为集群中的每一个主节点指定一个从节点，即一比一的复制。\

运行过程中，会有个提示，输入 **yes** 回车即可。从屏幕输出，可以很容易地看出哪些是主（master）节点，哪些是从（slave）节点：

```
>>> Creating cluster
Connecting to node 192.168.0.251:6379: OK
/usr/local/share/gems/gems/redis-3.0.0/lib/redis.rb:182: warning: wrong element type nil at 0 (expected array)
/usr/local/share/gems/gems/redis-3.0.0/lib/redis.rb:182: warning: ignoring wrong elements is deprecated, remove them explicitly
/usr/local/share/gems/gems/redis-3.0.0/lib/redis.rb:182: warning: this causes ArgumentError in the next release
>>> Performing hash slots allocation on 6 nodes...
Using 3 masters:
192.168.0.251:6379
192.168.0.252:6379
192.168.0.253:6379
Adding replica 192.168.0.252:6380 to 192.168.0.251:6379
Adding replica 192.168.0.251:6380 to 192.168.0.252:6379
Adding replica 192.168.0.253:6380 to 192.168.0.253:6379
```

```
M: 150f77d1000003811fb3c38c3768526a0b25ec31 192.168.0.251:6379
   slots:0-5460 (5461 slots) master
M: de461d3337b17d2119b79024d57d8b119e7320a6 192.168.0.252:6379
   slots:5461-10922 (5462 slots) master
M: faf50658fb7b0bae64cee5371da782e0f4919eee 192.168.0.253:6379
   slots:10923-16383 (5461 slots) master
S: c567db02cc40eebf577f71f703214dd2f4f26dfb 192.168.0.251:6380
   replicates de461d3337b17d2119b79024d57d8b119e7320a6
S: 284f8196b250ad9ac272316db84a07bebf661ab7 192.168.0.252:6380
   replicates 150f77d1000003811fb3c38c3768526a0b25ec31
S: 39fdef9fd5778dc94d8add819789d7d73ca06899 192.168.0.253:6380
   replicates faf50658fb7b0bae64cee5371da782e0f4919eee
Can I set the above configuration? (type 'yes' to accept): yes
>>> 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 192.168.0.251:6379)
M: 150f77d1000003811fb3c38c3768526a0b25ec31 192.168.0.251:6379
   slots:0-5460 (5461 slots) master
M: de461d3337b17d2119b79024d57d8b119e7320a6 192.168.0.252:6379
   slots:5461-10922 (5462 slots) master
M: faf50658fb7b0bae64cee5371da782e0f4919eee 192.168.0.253:6379
   slots:10923-16383 (5461 slots) master
M: c567db02cc40eebf577f71f703214dd2f4f26dfb 192.168.0.251:6380
   slots: (0 slots) master
   replicates de461d3337b17d2119b79024d57d8b119e7320a6
M: 284f8196b250ad9ac272316db84a07bebf661ab7 192.168.0.252:6380
   slots: (0 slots) master
   replicates 150f77d1000003811fb3c38c3768526a0b25ec31
M: 39fdef9fd5778dc94d8add819789d7d73ca06899 192.168.0.253:6380
   slots: (0 slots) master
   replicates faf50658fb7b0bae64cee5371da782e0f4919eee
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

10.3. ps aux|grep redis

```
[test@test-168-251 ~]$ ps aux|grep redis
test   3824  0.7  5.9 6742404 3885144 ?    Ssl   2015 1639:13 /data/redis/bin/redis-server *:6379 [cluster]
test   3831  0.5  3.9 6709636 2618536 ?    Ssl   2015 1235:43 /data/redis/bin/redis-server *:6380 [cluster]
```

停止 redis 实例，直接使用 kill 命令即可，如：kill **3831**，重启和单机版相同，经过上述一系列操作后，重启会自动转换成 cluster 模式。。

11. redis cluster client

11.1. 命令行工具 redis-cli

官方提供的命令行客户端工具，在单机版 redis 基础上指定参数“**-c**”即可。以下是在 192.168.0.251 上执行 redis-cli 的记录：

```
$ ./redis-cli -c -p 6379
127.0.0.1:6379> set foo bar
-> Redirected to slot [12182] located at 192.168.0.253:6379
OK
192.168.0.253:6379> set hello world
-> Redirected to slot [866] located at 192.168.0.251:6379
OK
192.168.0.251:6379> get foo
-> Redirected to slot [12182] located at 192.168.0.253:6379
"bar"
192.168.0.253:6379> get hello
-> Redirected to slot [866] located at 192.168.0.251:6379
"world"

查看集群中的节点：
192.168.0.251:6379> cluster nodes
```

11.2. 从 slaves 读数据

默认不能从 slaves 读取数据，但建立连接后，执行一次命令 READONLY ，即可从 slaves 读取数据。如果想再次恢复不能从 slaves 读取数据，可以执行下命令 READWRITE。

11.3. jedis（java cluster client）

官网：https://github.com/xetorthio/jedis

编程示例：

```
Set<HostAndPort> jedisClusterNodes = new HashSet<HostAndPort>();
//Jedis Cluster will attempt to discover cluster nodes automatically
jedisClusterNodes.add(new HostAndPort("127.0.0.1", 7379));
JedisCluster jc = new JedisCluster(jedisClusterNodes);
jc.set("foo", "bar");
String value = jc.get("foo");
```

11.4. r3c（C++ cluster client）

官网：https://github.com/eyjian/r3c

12. 新增节点

12.1. 添加一个新主（master）节点

先以单机版配置和启动好 redis-server，然后执行命令：

```
./redis-trib.rb add-node 127.0.0.1:7006 127.0.0.1:7000
```

执行上面这条命令时，可能遇到错误“[ERR] Sorry, can't connect to node 127.0.0.1:7006”。引起该问题的原因可能是因为 ruby 的版本过低（运行 ruby -v 可以查看 ruby 的版本），可以尝试升级 ruby 再尝试，比如 ruby 1.8.7 版本就需要升级。对于 Redis 3.0.5 和 Redis 3.2.0，使用 Ruby 2.3.1 操作正常。请注意升级到最新版本的 ruby 也可能遇到这个错误。

另一个会引起这个问题的原因是从 Redis 3.2.0 版本开始引入了“保护模式（protected mode），防止 redis-cli 远程访问”，仅限 redis-cli 绑定到 127.0.0.1 才可以连接 Redis server。

为了完成添加新主节点，可以暂时性的关闭保护模式，使用 redis-cli，不指定-h 参数（但可以指定-p 参数，或者-h 参数值为 127.0.0.1）进入操作界面：**CONFIG SET protected-mode no**。

注意 7006 是新增的节点，而 7000 是已存在的节点（可为 master 或 slave）。如果需要将 7006 变成某 master 的 slave 节点，执行命令：

```
cluster replicate 3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e
```

新加入的 master 节点上没有任何数据（slots，运行 redis 命令 cluster nodes 可以看到这个情况）。当一个 slave 想成为 master 时，由于这个新的 master 节点不管理任何 slots，它不参与选举。

可以使用工具 redis-trib.rb 的 resharding 特性为这个新 master 节点分配 slots，如：**redis-trib.rb reshard** 127.0.0.1:**7000**，其中 **7000** 为集群中任意一个节点即可，redis-trib.rb 将自动发现其它节点。

在 reshard 过程中，将会询问 reshard 多少 slots：

How many **slots** do you want to move (from 1 to 16384)?，取值范围为 1~16384，其中 16384 为 redis cluster 的拥有的 slots 总数，比如想只移动 100 个，输入 100 即可。如果迁移的 slots 数量多，应当设置 redis-trib.rb 的超时参数--timeout 值大一点。否则，迁移过程中易遇到超时错误“[ERR] Calling MIGRATE: IOERR error or timeout reading to target instance”，导致只完成部分，可能会造成数据丢失。

接着，会提示“What is the receiving node ID?”，输入新加入的 master 节点 ID。过程中如果遇到错误“Sorry, can't connect to node 10.225.168.253:6380”，则可能需要暂时先关闭相应的保护模式。

如果在迁移过程遇到下面这样的错误：

```
>>> Check for open slots...
[WARNING] Node 192.168.0.3:6379 has slots in importing state (5461).
[WARNING] Node 192.168.0.5:6380 has slots in migrating state (5461).
[WARNING] The following slots are open: 5461
```

可以考虑使用命令“redis-trib.rb fix 192.168.0.3:6379”尝试修复。需要显示有节点处于 migrating 或 importing 状态，可以登录到相应的节点，使用命令“cluster setslot **5461** stable”修改，参数 **5461** 为问题显示的 slot 的 ID。

12.2. 添加一个新从（slave）节点

```
./redis-trib.rb add-node --slave 127.0.0.1:7006 127.0.0.1:7000
```

注意这种方式，如果添加了多个 slave 节点，可能导致 master 的 slaves 不均衡，比如一些有 3 个 slave，其它只 1 个 slave。可以在 slave 节点上执行 redis 命令“CLUSTER REPLICATE”进行调整，让它成为其它 master 的 slave。“CLUSTER REPLICATE”带一个参数，即 master ID，注意使用 redis-cli -c 登录到 slave 上执行。

上面方法没有指定 7006 的 master，而是随机指定。下面方法可以明确指定为哪个 master 的 slave：

```
./redis-trib.rb add-node --slave --master-id 3c3a0c74aae0b56170ccb03a76b60cfe7dc1912e 127.0.0.1:7006 127.0.0.1:7000
```

其中“127.0.0.1:7006”是新节点，“127.0.0.1:7000”是集群中已有的节点。

13. 删除节点

从集群中删除一个节点：

```
./redis-trib.rb del-node 127.0.0.1:7000 <node-id>
```

第一个参数为集群中任意一个节点，第二个参数为需要删除节点的 ID。

成功删除后，提示：

```
$./redis-trib.rb del-node 127.0.0.1:6380 f49a2bda05e81aa343adb9924775ba95a1f4236e
>>> Removing node f49a2bda05e81aa343adb9924775ba95a1f4236e from cluster 127.0.0.1:6379
/usr/local/share/gems/gems/redis-3.0.0/lib/redis.rb:182: warning: wrong element type nil at 0 (expected array)
。。。 。。。
/usr/local/share/gems/gems/redis-3.0.0/lib/redis.rb:182: warning: this causes ArgumentError in the next release
>>> Sending CLUSTER FORGET messages to the cluster...
>>> SHUTDOWN the node. 在这里会停顿几分钟，通知并等待被删除节点退出（exit），被删除节点在将数
```

据写到 RDB 文件中后退出，所以停顿时长和写 RDB 文件时长有关，数据量越大时间就越长。5~6G 的 RAID1 的 SATA 盘数据大概需要 45 秒左右。

被删除节点日志：

```
15577:S 06 Sep 20:06:37.774 - Accepted 10.49.126.98:14669
15577:S 06 Sep 20:06:38.741 # User requested shutdown...
15577:S 06 Sep 20:06:38.741 * Calling fsync() on the AOF file.
15577:S 06 Sep 20:06:38.742 * Saving the final RDB snapshot before exiting.
15577:S 06 Sep 20:07:19.683 * DB saved on disk
15577:S 06 Sep 20:07:19.683 * Removing the pid file.
15577:S 06 Sep 20:07:19.683 # Redis is now ready to exit, bye bye...
```

成功后不用再调用“CLUSTER FORGET”，否则报错：

```
$ redis-cli -c CLUSTER FORGET aa6754a093ea4047f92cc0ea77f1859553bc5c57
(error) ERR Unknown node aa6754a093ea4047f92cc0ea77f1859553bc5c57
```

如果待删除节点已经不能连接，则调用 CLUSTER FORGET 剔除（可能需要在所有机器上执行一次 FORGET）：

```
CLUSTER FORGET <node-id>
```

注意如果是删除一个 master 节点，则需要先将它管理的 slots 的迁走，然后才可以删除它。

如果是 master 或 slave 机器不能连接，比如硬件故障导致无法启动，这个时候做不了 **del-node**，只需要直接做 CLUSTER 即可，在 FORGET 后，节点状态变成 handshake。

!!! 请注意，需要在**所有 node** 上执行一次“CLUSTER FORGET”，否则可能遇到被剔除 node 的总是处于 handshake 状态。

如果有部分 node 没有执行到 FORGET，导致有部分 node 还处于 fail 状态，则在一些 node 将看到待剔除节点仍然处于 handshake 状态，并且 nodeid 在不断变化，所以需要在所有 node 上执行“CLUSTER FORGET”。

如果一个节点处于“**:0 master,fail,noaddr**”状态，执行“del-node”会报错：
[ERR] No such node ID 80560d0d97a0b3fa975203350516437b58251745
这种情况下，只需要执行“CLUSTER FORGET”将其剔除即可（注意，需要在所有节点上执行一次，不然未执行的节点上可能仍然看得到“**:0 master,fail,noaddr**”）：

```
# redis-cli -c -p 1383 cluster nodes
80560d0d97a0b3fa975203350516437b58251745 :0 master,fail,noaddr - 1528947205054 1528947203553 0 disconnected
fa7bbbf7d48389409ce05d303272078c3a6fd44f 127.0.0.1:1379 slave 689f7c1ae71ea294c4ad7c5d1b32ae4e78e27915 0
1535871825187 138 connected
c1a9d1d23438241803ec97fbd765737df80f402a 127.0.0.1:1381 slave f03b1008988acbb0f69d96252decda9adf747be9 0
1535871826189 143 connected
50003ccd5885771196e717e27011140e7d6c94e0 127.0.0.1:1385 slave f03b1008988acbb0f69d96252decda9adf747be9 0
1535871825688 143 connected
f6080015129eada3261925cc1b466f1824263358 127.0.0.1:1380 slave 4e932f2a3d80de29798660c5ea62e473e63a6630 0
1535871825388 145 connected
```

```
4e932f2a3d80de29798660c5ea62e473e63a6630 127.0.0.1:1383 myself,master - 0 0 145 connected 5458-10922
689f7c1ae71ea294c4ad7c5d1b32ae4e78e27915 127.0.0.1:1382 master - 0 1535871826490 138 connected 0-1986 1988-5457
f03b1008988acbb0f69d96252decda9adf747be9 127.0.0.1:1384 master - 0 1535871825187 143 connected 1987 10923-16383
```

14. master 机器硬件故障

这种情况下，master 机器可能无法启动，导致其上的 master 无法连接，master 将一直处于“master,fail”状态，如果是 slave 则处于“slave,fail”状态。

如果是 master，则会它的 slave 变成了 master，因此只需要添加一个新的从节点作为原 slave（已变成 master）的 slave 节点。完成后，通过 CLUSTER **FORGET** 将故障的 master 或 slave 从集群中剔除即可。

!!! 请注意，需要在**所有 node** 上执行一次“CLUSTER **FORGET**”，否则可能遇到被剔除 node 的总是处于 handshake 状态。

15. 检查节点状态

redis-trib.rb check 127.0.0.1:6380

如发现如下这样的错误：
[WARNING] Node **192.168.0.11:6380** has slots in migrating state (5461).
[WARNING] The following slots are open: **5461**

可以使用 redis 命令取消 slots 迁移（**5461** 为 slot 的 ID）：
cluster **setslot** **5461** stable
需要注意，须登录到 **192.168.0.11:6380** 上执行 redis 的 **setslot** 子命令。

16. 变更主从关系

使用命令 cluster replicate，参数为 master 节点 ID，注意不是 IP 和端口，在被迁移的 slave 上执行该命令。

17. slots 相关命令

```
CLUSTER ADDSLOTS slot1 [slot2] ... [slotN]
CLUSTER DELSLOTS slot1 [slot2] ... [slotN]
CLUSTER SETSLOT slot NODE node
CLUSTER SETSLOT slot MIGRATING node
CLUSTER SETSLOT slot IMPORTING node
```

17.1. 迁移 slosts

官方参考：https://redis.io/commands/cluster-setslot。

示例：将值为 8 的 slot 从源节点 A 迁移到目标节点 B，有如下两种方法：

```
在目标节点 B 上执行：CLUSTER SETSLOT 8 IMPORTING src-A-node-id
或
在源节点 A 上执行：CLUSTER SETSLOT 8 MIGRATING dst-B-node-id
```

上述操作只是将 slot 标记为迁移状态，完成迁移还需要执行（在目标 node 上执行）：

```
CLUSTER SETSLOT <slot> NODE <dst-node-id>
```

其中 node-id 为目标的 Node ID，**取消迁移**使用“CLUSTER SETSLOT <slot> **STABLE**”。

操作示例：

```
# 将值为 11677 的 slot 迁到 192.168.31.3:6379
$ redis-cli -c -h 192.168.31.3 -p 6379 CLUSTER SETSLOT 11677 IMPORTING 216e0069af11eca91465394b2ad7bf1c27f5f7fe
OK
$ redis-cli -c -h 192.168.31.3 -p 6379 CLUSTER SETSLOT 11677 NODE 4e149c72aff2b6651370ead476dd70c8cf9e3e3c
OK
```

17.2. redis-trib.rb rebalance

当有增减节点时，可以使用命令：
redis-trib.rb rebalance 192.168.0.31:6379 --auto-weights
做一次均衡，简单点可以只指定两个参数：“192.168.0.31:6379”为集群中已知的任何一个节点，参数“-auto-weights”表示自动权重。

18. 人工主备切换

在需要的 slaves 节点上执行命令：CLUSTER **FAILOVER**。如果人工发起 failover，则其它 master 会收到“Failover auth **granted** to 4291f18b5e9729e832ed15ceb6324ce5dfc2ffbe for **epoch** 31”，每次 **epoch** 值增一。

```
23038:M 06 Sep 20:31:24.815 # Failover auth granted to 4291f18b5e9729e832ed15ceb6324ce5dfc2ffbe for epoch 31

当出现下面两条日志时，表示 failover 完成：
23038:M 06 Sep 20:32:44.019 * FAIL message received from ea28f68438e5bb79c26a9cb2135241f11d7a50ba about 5e6ffacb2c5d5761e39aba5270fbf48f296cb5ee
23038:M 06 Sep 20:32:58.487 * Clear FAIL state for node 5e6ffacb2c5d5761e39aba5270fbf48f296cb5ee: slave is reachable again.
```

原 master 收到 failover 后的日志：

```
35475:M 06 Sep 20:35:43.396 - DB 0: 16870482 keys (7931571 volatile) in 50331648 slots HT.
35475:M 06 Sep 20:35:43.396 - 1954 clients connected (1 slaves), 5756515544 bytes in use
35475:M 06 Sep 20:35:48.083 # Manual failover requested by slave 58a40dbe01e1563773724803854406df04c62724.
35475:M 06 Sep 20:35:48.261 # Failover auth granted to 58a40dbe01e1563773724803854406df04c62724 for epoch 32
35475:M 06 Sep 20:35:48.261 - Client closed connection

10.51.147.216:7388 为 failover 前的 slave，
10.51.147.216:7388 的 ID 为 58a40dbe01e1563773724803854406df04c62724
35475:M 06 Sep 20:35:48.261 # Connection with slave 10.51.147.216:7388 lost.
35475:M 06 Sep 20:35:48.278 # Configuration change detected. Reconfiguring myself as a replica of 58a40dbe01e1563773724803854406df04c62724
35475:S 06 Sep 20:35:48.280 - Client closed connection
35475:S 06 Sep 20:35:48.408 - DB 0: 16870296 keys (7931385 volatile) in 50331648 slots HT.
35475:S 06 Sep 20:35:48.408 - 1953 clients connected (0 slaves), 5722753736 bytes in use
35475:S 06 Sep 20:35:48.408 * Connecting to MASTER 10.51.147.216:7388
35475:S 06 Sep 20:35:48.408 * MASTER <-> SLAVE sync started
35475:S 06 Sep 20:35:48.408 * Non blocking connect for SYNC fired the event.
35475:S 06 Sep 20:35:48.408 * Master replied to PING, replication can continue...
35475:S 06 Sep 20:35:48.408 * Partial resynchronization not possible (no cached master)
35475:S 06 Sep 20:35:48.459 * Full resync from master: 36beb63d32b3809039518bf4f3e4e10de227f3ee:16454238619
35475:S 06 Sep 20:35:48.493 - Client closed connection
35475:S 06 Sep 20:35:48.880 - Client closed connection
```

19. 查看集群信息

对应的 redis 命令为：cluster info，示例：

```
127.0.0.1:6381> cluster info
cluster_state:ok 所有 slots 正常则显示为 OK，否则为 error
cluster_slots_assigned:16384 多少 slots 被分配了，即多少被 master 管理了，16384 为全部 slots
cluster_slots_ok:16384 有多少 slots 是正常的
cluster_slots_pfail:0 有多少 slots 可能处于异常状态，处于这个状态并不表示有问题，仍能继续提供服务
cluster_slots_fail:0 有多少 slots 处于异常状态，需要修复才能服务
cluster_known_nodes:10 集群中的节点数
cluster_size:3 集群中 master 个数
cluster_current_epoch:11 本地的当前时间变量，用于故障切换时生成独一无二的增量版本号
cluster_my_epoch:0
cluster_stats_messages_sent:4049 通过集群消息总线发送的消息总数
cluster_stats_messages_received:4051 通过过集通过群消息总线收到的消息总数
```

20. 禁止指定命令

KEYS 命令很耗时，FLUSHDB 和 FLUSHALL 命令可能导致误删除数据，所以线上环境最好禁止使用，可以在 Redis 配置文件增加如下配置：

```
rename-command KEYS ""
rename-command FLUSHDB ""
rename-command FLUSHALL ""
```

21. 各版本配置文件

https://raw.githubusercontent.com/antirez/redis/3.0/redis.conf
https://raw.githubusercontent.com/antirez/redis/3.2.9/redis.conf
https://raw.githubusercontent.com/antirez/redis/4.0/redis.conf
https://raw.githubusercontent.com/antirez/redis/4.0.1/redis.conf
https://raw.githubusercontent.com/antirez/redis/4.0.3/redis.conf
https://raw.githubusercontent.com/antirez/redis/4.0.5/redis.conf
https://raw.githubusercontent.com/antirez/redis/4.0.9/redis.conf
https://raw.githubusercontent.com/antirez/redis/4.0.11/redis.conf

22. 大压力下 Redis 参数调整要点

参数	建议最小值	说明
repl-ping-slave-period	10	每 10 秒 ping 一次
repl-timeout	60	60 秒超时，也就是 ping 十次
cluster-node-timeout	15000	
repl-backlog-size	1GB	Master 对 slave 的队列大小
appendfsync	no	让系统自动刷
save		大压力下，调大参数值，以减少写 RDB 带来的压力："900 **20** 300 **200** 60 **200000**"
appendonly		对于队列，建议单独建立集群，并且设置该值为 no

为何大压力下要这样调整？
最重要的原因之一 Redis 的主从复制，两者复制共享同一线程，虽然是异步复制的，但因为是单线程，所以也十分有限。如果主从间的网络延迟不是在 0.05 左右，比如达到 0.6，甚至 1.2 等，那么情况是非常糟糕的，因此同一 Redis 集群一定要部署在同一机房内。

这些参数的具体值，要视具体的压力而定，而且和消息的大小相关，比如一条 200~500KB 的流水数据可能比较大，主从复制的压力也会相应增大，而 10 字节左右的消息，则压力要小一些。大压力环境中开启 appendfsync 是十分不可取的，容易导致整个集群不可用，在不可用之前的典型表现是 QPS 毛刺明显。

这么做的目的是让 Redis 集群尽可能的避免 master 正常时触发主从切换，特别是容纳的数据量很大时，和大压力结合在一起，集群会雪崩。

当 Redis 日志中，出现大量如下信息，即可能意味着相关的参数需要调整了：

22135:M 06 Sep 14:17:05.388 * FAIL message received from 1d07e208db56cfd7395950ca66e03589278b8e12 about e438a338e9d9834a6745c12931950da87e360ca2

22135:M 06 Sep 14:17:07.551 * FAIL message received from ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467 about d6eb06e9d118c120d3961a659972a1d0191a8652

22135:M 06 Sep 14:17:08.438 # Failover auth **granted** to f7d6b2c72fa3b801e7dcfe0219e73383d143dd0f for epoch 285 （**We can vote for this slave**）

有投票资格的 node：

1）为 master

2）至少有一个 slot

3）投票 node 的 epoch 不能小于 node 自己当前的 epoch（reqEpoch < curEpoch）

4）node 没有投票过该 epoch（already voted for epoch）

5）投票 node 不能为 master（it is a master node）

6）投票 node 必须有一个 master（I don't know its master）

7）投票 node 的 master 处于 fail 状态（its master is up）

22135:M 06 Sep 14:17:19.844 # Failover auth **denied** to 534b93af6ba45a7033dbf38c8f47cd688514125a: already voted for epoch 285

如果一个 node 又联系上了，则它当是一个 slave，或者无 slots 的 master 时，直接清除 FAIL 标志；但如果是一个 master，则当“(now - node->fail_time) > (server.cluster_node_timeout * CLUSTER_FAIL_UNDO_TIME_MULT)”时，也清除 FAIL 标志，定义在 cluster.h 中（cluster.h:#define CLUSTER_FAIL_UNDO_TIME_MULT **2** /* Undo fail if master is back. */**）**

22135:M 06 Sep 14:17:29.243 * **Clear FAIL state** for node d6eb06e9d118c120d3961a659972a1d0191a8652: master without slots is reachable again.

如果消息类型为 fail。

22135:M 06 Sep 14:17:31.995 * FAIL message received from f7d6b2c72fa3b801e7dcfe0219e73383d143dd0f about 1ba437fa1683a8caafd38ff977e5fbabdaf84fd6

22135:M 06 Sep 14:17:32.496 * FAIL message received from 1d07e208db56cfd7395950ca66e03589278b8e12 about d7942cfe636b25219c6d56aa72828fcfde2ee261

22135:M 06 Sep 14:17:32.968 # Failover auth granted to 938d9ae2de278938beda1d39185608b02d3b31ec for epoch 286

22135:M 06 Sep 14:17:33.177 # Failover auth granted to d9dadf3342006e2c92def3071ca0a76390be62b0 for epoch 287

22135:M 06 Sep 14:17:36.336 * Clear FAIL state for node 1ba437fa1683a8caafd38ff977e5fbabdaf84fd6: master without slots is reachable again.

22135:M 06 Sep 14:17:36.855 * Clear FAIL state for node d7942cfe636b25219c6d56aa72828fcfde2ee261: master without slots is reachable again.

22135:M 06 Sep 14:17:38.419 * Clear FAIL state for node e438a338e9d9834a6745c12931950da87e360ca2: is reachable again and nobody is serving its slots after some time.

```
22135:M 06 Sep 14:17:54.954 * FAIL message received from ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467 about 7990d146cece7dc83eaf08b3e12cbebb2223f5f8
22135:M 06 Sep 14:17:56.697 * FAIL message received from 1d07e208db56cfd7395950ca66e03589278b8e12 about fbe774cdbd2acd24f9f5ea90d61c607bdf800eb5
22135:M 06 Sep 14:17:57.705 # Failover auth granted to e1c202d89ffe1c61b682e28071627635974c84a7 for epoch 288
22135:M 06 Sep 14:17:57.890 * Clear FAIL state for node 7990d146cece7dc83eaf08b3e12cbebb2223f5f8: slave is reachable again.
22135:M 06 Sep 14:17:57.892 * Clear FAIL state for node fbe774cdbd2acd24f9f5ea90d61c607bdf800eb5: master without slots is reachable again.
```

23. 问题排查

1) 如果最后一条日志为“16367:M 08 Jun 14:48:15.560 # Server started, Redis version 3.2.0”，节点状态始终终于 fail 状态，则可能是 aof 文件损坏了，这时可以使用工具 edis-check-aof --fix 进行修改，如：

../../bin/redis-check-aof --fix appendonly-6380.aof

0x a1492b9b: **Expected prefix '**

AOF analyzed: size=2705928192, ok_up_to=2705927067, diff=1125

This will shrink the AOF from 2705928192 bytes, with 1125 bytes, to 2705927067 bytes

Continue? [y/N]: **y**

2) in `call': ERR Slot 16011 is already busy (Redis::CommandError)

将所有节点上的配置项 cluster-config-file 指定的文件删除，然后重新启；或者在所有节点上执行下 FLUSHALL 命令。

另外，如果使用主机名而不是 IP，也可能遇到这个错误，如：“redis-trib.rb create --replicas 1 redis1:6379 redis2:6379 redis3:6379 redis4:6379 redis5:6379 redis6:6379”，可能也会得到错误“ERR Slot 16011 is already busy (Redis::CommandError)”。

3) **for lack of backlog** (Slave request was: 51875158284)

默认值：

redis-cli config get repl-timeout

A) "repl-timeout"

B) "10"

redis-cli config get client-output-buffer-limit

A) "client-output-buffer-limit"

B) "normal 0 0 0 slave **268435456 67108864** 60 pubsub 33554432 8388608 60"

增大：

redis-cli config set "client-output-buffer-limit" "normal 0 0 0 slave **2684354560 671088640 60** pubsub 33554432 8388608 60"

4) 复制中断场景

A) master 的 slave 缓冲区达到限制的硬或软限制大小，与参数 client-output-buffer-limit 相关；

B) 复制时间超过 repl-timeout 指定的值，与参数 repl-timeout 相关。

slave 反复循环从 master 复制，如果调整以上参数仍然解决不了，可以尝试删除 slave 上的 aof 和 rdb 文件，然后再重启进程复制，这个时候可能能正常完成复制。

5) 日志文件出现：Asynchronous AOF fsync is taking too long (disk is busy?). Writing the AOF buffer without waiting for fsync to complete, this may slow down Redis.

考虑优化以下配置项：

no-appendfsync-on-rewrite 值设为 yes

repl-backlog-size 和 client-output-buffer-limit 调大一点

6) 日志文件出现：MISCONF Redis is configured to save RDB snapshots, but is currently not able to persist on disk. Commands that may modify the data set are disabled. Please check Redis logs for details about the error.

考虑设置 stop-writes-on-bgsave-error 值为“no”。

7) Failover auth granted to

当日志大量反反复复出现下列内容时，很可能表示 master 和 slave 间同步和通讯不顺畅，导致无效的 failover 和状态变更，这个时候需要调大相关参数值，容忍更长的延迟，因此也特别注意集群内所有节点间的网络延迟要尽可能的小，最好达到 0.02ms 左右的水平，调大参数的代价是主备切换变迟钝。

Slave 日志：

```
31019:S 06 Sep 11:07:24.169 * Connecting to MASTER 10.5.14.8:6379
31019:S 06 Sep 11:07:24.169 * MASTER <-> SLAVE sync started
31019:S 06 Sep 11:07:24.169 # Start of election delayed for 854 milliseconds (rank #0, offset 5127277817).
31019:S 06 Sep 11:07:24.169 * Non blocking connect for SYNC fired the event.
31019:S 06 Sep 11:07:25.069 # Starting a failover election for epoch 266.
31019:S 06 Sep 11:07:29.190 * Clear FAIL state for node ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467: is reachable again and nobody is serving its slots after some time.
31019:S 06 Sep 11:07:29.191 * Master replied to PING, replication can continue...
31019:S 06 Sep 11:07:29.191 * Clear FAIL state for node f7d6b2c72fa3b801e7dcfe0219e73383d143dd0f: is reachable again and nobody is serving its slots after some time.
31019:S 06 Sep 11:07:29.192 * Trying a partial resynchronization (request ea2261c827fbc54135a95f707046581a55dff133:5127277818).
31019:S 06 Sep 11:07:29.192 * Successful partial resynchronization with master.
31019:S 06 Sep 11:07:29.192 * MASTER <-> SLAVE sync: Master accepted a Partial Resynchronization.
31019:S 06 Sep 11:07:29.811 * Clear FAIL state for node e438a338e9d9834a6745c12931950da87e360ca2: is reachable again and nobody is serving its slots after some time.
31019:S 06 Sep 11:07:37.680 * FAIL message received from 5b41f7860cc800e65932e92d1d97c6c188138e56 about 3114cec541c5bcd36d712cd6c9f4c5055510e386
31019:S 06 Sep 11:07:43.710 * Clear FAIL state for node 3114cec541c5bcd36d712cd6c9f4c5055510e386: slave is reachable again.
31019:S 06 Sep 11:07:48.119 * FAIL message received from 7d61af127c17d9c19dbf9af0ac8f7307f1c96c4b about e1c202d89ffe1c61b682e28071627635974c84a7
31019:S 06 Sep 11:07:49.410 * FAIL message received from 5b41f7860cc800e65932e92d1d97c6c188138e56 about d9dadf3342006e2c92def3071ca0a76390be62b0
31019:S 06 Sep 11:07:53.352 * Clear FAIL state for node d9dadf3342006e2c92def3071ca0a76390be62b0: slave is reachable again.
31019:S 06 Sep 11:07:57.147 * Clear FAIL state for node e1c202d89ffe1c61b682e28071627635974c84a7: slave is reachable again.
31019:S 06 Sep 11:08:36.516 * FAIL message received from ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467 about 938d9ae2de278938beda1d39185608b02d3b31ec
```

31019:S 06 Sep 11:08:41.900 * Clear FAIL state for node 938d9ae2de278938beda1d39185608b02d3b31ec: slave is reachable again.

31019:S 06 Sep 11:08:46.380 * FAIL message received from d7942cfe636b25219c6d56aa72828fcfde2ee261 about fbe774cdbd2acd24f9f5ea90d61c607bdf800eb5

31019:S 06 Sep 11:08:46.531 * Marking node 7990d146cece7dc83eaf08b3e12cbebb2223f5f8 as failing (quorum reached).

31019:S 06 Sep 11:09:01.882 * Clear FAIL state for node 7990d146cece7dc83eaf08b3e12cbebb2223f5f8: master without slots is reachable again.

31019:S 06 Sep 11:09:01.883 * Clear FAIL state for node fbe774cdbd2acd24f9f5ea90d61c607bdf800eb5: master without slots is reachable again.

31019:S 06 Sep 11:09:06.538 * FAIL message received from e438a338e9d9834a6745c12931950da87e360ca2 about d7942cfe636b25219c6d56aa72828fcfde2ee261

31019:S 06 Sep 11:09:06.538 * FAIL message received from e438a338e9d9834a6745c12931950da87e360ca2 about 1ba437fa1683a8caafd38ff977e5fbabdaf84fd6

31019:S 06 Sep 11:09:12.555 * Clear FAIL state for node 1ba437fa1683a8caafd38ff977e5fbabdaf84fd6: is reachable again and nobody is serving its slots after some time.

31019:S 06 Sep 11:09:12.555 * Clear FAIL state for node d7942cfe636b25219c6d56aa72828fcfde2ee261: master without slots is reachable again.

31019:S 06 Sep 11:09:15.565 * Marking node 534b93af6ba45a7033dbf38c8f47cd688514125a as failing (quorum reached).

31019:S 06 Sep 11:09:16.599 * FAIL message received from 0a92bd7472c9af3e52f9185eac1bd1bbf36146e6 about e1c202d89ffe1c61b682e28071627635974c84a7

31019:S 06 Sep 11:09:22.262 * Clear FAIL state for node 534b93af6ba45a7033dbf38c8f47cd688514125a: slave is reachable again.

31019:S 06 Sep 11:09:27.906 * Clear FAIL state for node e1c202d89ffe1c61b682e28071627635974c84a7: is reachable again and nobody is serving its slots after some time.

31019:S 06 Sep 11:09:50.744 * FAIL message received from ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467 about e1c202d89ffe1c61b682e28071627635974c84a7

31019:S 06 Sep 11:09:55.141 * FAIL message received from 5b41f7860cc800e65932e92d1d97c6c188138e56 about d9dadf3342006e2c92def3071ca0a76390be62b0

31019:S 06 Sep 11:09:55.362 * FAIL message received from 7d61af127c17d9c19dbf9af0ac8f7307f1c96c4b about 938d9ae2de278938beda1d39185608b02d3b31ec

31019:S 06 Sep 11:09:55.557 * FAIL message received from ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467 about 1d07e208db56cfd7395950ca66e03589278b8e12

31019:S 06 Sep 11:09:55.578 * FAIL message received from ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467 about 144347d5a51acf047887fe81f22e8f7705c911ec

31019:S 06 Sep 11:09:56.521 * Marking node 534b93af6ba45a7033dbf38c8f47cd688514125a as failing (quorum reached).

31019:S 06 Sep 11:09:57.996 * Clear FAIL state for node 1d07e208db56cfd7395950ca66e03589278b8e12: slave is reachable again.

31019:S 06 Sep 11:09:58.329 * FAIL message received from 5b41f7860cc800e65932e92d1d97c6c188138e56 about 0a92bd7472c9af3e52f9185eac1bd1bbf36146e6

31019:S 06 Sep 11:10:09.239 * Clear FAIL state for node 144347d5a51acf047887fe81f22e8f7705c911ec: slave is reachable again.

31019:S 06 Sep 11:10:09.812 * Clear FAIL state for node d9dadf3342006e2c92def3071ca0a76390be62b0: is reachable again and nobody is serving its slots after some time.

31019:S 06 Sep 11:10:13.549 * Clear FAIL state for node 534b93af6ba45a7033dbf38c8f47cd688514125a: slave is reachable again.

31019:S 06 Sep 11:10:13.590 * FAIL message received from 716f2e2dd9792eaf4ee486794c9797fa6e1c9650 about 1ba437fa1683a8caafd38ff977e5fbabdaf84fd6

31019:S 06 Sep 11:10:13.591 * FAIL message received from f7d6b2c72fa3b801e7dcfe0219e73383d143dd0f about d7942cfe636b25219c6d56aa72828fcfde2ee261

```
31019:S 06 Sep 11:10:14.316 * Clear FAIL state for node e1c202d89ffe1c61b682e28071627635974c84a7: is reachable again and nobody is serving its slots after some time.
31019:S 06 Sep 11:10:15.108 * Clear FAIL state for node d7942cfe636b25219c6d56aa72828fcfde2ee261: slave is reachable again.
31019:S 06 Sep 11:10:17.588 * Clear FAIL state for node 938d9ae2de278938beda1d39185608b02d3b31ec: slave is reachable again.
31019:S 06 Sep 11:10:32.622 * Clear FAIL state for node 0a92bd7472c9af3e52f9185eac1bd1bbf36146e6: slave is reachable again.
31019:S 06 Sep 11:10:32.623 * FAIL message received from 5b41f7860cc800e65932e92d1d97c6c188138e56 about 3114cec541c5bcd36d712cd6c9f4c5055510e386
31019:S 06 Sep 11:10:32.623 * Clear FAIL state for node 3114cec541c5bcd36d712cd6c9f4c5055510e386: slave is reachable again.
```

Master 日志：

```
31014:M 06 Sep 14:08:54.083 * Background saving terminated with success
31014:M 06 Sep 14:09:55.093 * 10000 changes in 60 seconds. Saving...
31014:M 06 Sep 14:09:55.185 * Background saving started by pid 41395
31014:M 06 Sep 14:11:00.269 # Disconnecting timedout slave: 10.15.40.9:6018
31014:M 06 Sep 14:11:00.269 # Connection with slave 10.15.40.9:6018 lost.
41395:C 06 Sep 14:11:01.141 * DB saved on disk
41395:C 06 Sep 14:11:01.259 * RDB: 5 MB of memory used by copy-on-write
31014:M 06 Sep 14:11:01.472 * Background saving terminated with success
31014:M 06 Sep 14:11:11.525 * FAIL message received from 1d07e208db56cfd7395950ca66e03589278b8e12 about 534b93af6ba45a7033dbf38c8f47cd688514125a
31014:M 06 Sep 14:11:23.039 * FAIL message received from 1ba437fa1683a8caafd38ff977e5fbabdaf84fd6 about d78845370c98b3ce4cfc02e8d3e233a9a1d84a83
31014:M 06 Sep 14:11:23.541 * Clear FAIL state for node 534b93af6ba45a7033dbf38c8f47cd688514125a: slave is reachable again.
31014:M 06 Sep 14:11:23.813 * Slave 10.15.40.9:6018 asks for synchronization
31014:M 06 Sep 14:11:23.813 * Partial resynchronization request from 10.15.40.9:6018 accepted. Sending 46668 bytes of backlog starting from offset 5502672944.
31014:M 06 Sep 14:11:23.888 # Failover auth granted to 7d61af127c17d9c19dbf9af0ac8f7307f1c96c4b for epoch 283
31014:M 06 Sep 14:11:32.464 * FAIL message received from d6eb06e9d118c120d3961a659972a1d0191a8652 about 3114cec541c5bcd36d712cd6c9f4c5055510e386
31014:M 06 Sep 14:11:47.616 * Clear FAIL state for node d78845370c98b3ce4cfc02e8d3e233a9a1d84a83: master without slots is reachable again.
31014:M 06 Sep 14:11:55.515 * FAIL message received from d6eb06e9d118c120d3961a659972a1d0191a8652 about ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467
31014:M 06 Sep 14:11:57.135 # Failover auth granted to ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467 for epoch 284
31014:M 06 Sep 14:12:01.766 * Clear FAIL state for node ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467: slave is reachable again.
31014:M 06 Sep 14:12:08.753 * Clear FAIL state for node 3114cec541c5bcd36d712cd6c9f4c5055510e386: master without slots is reachable again.
31014:M 06 Sep 14:16:02.070 * 10 changes in 300 seconds. Saving...
31014:M 06 Sep 14:16:02.163 * Background saving started by pid 13832
31014:M 06 Sep 14:17:18.443 * FAIL message received from ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467 about d6eb06e9d118c120d3961a659972a1d0191a8652
31014:M 06 Sep 14:17:18.443 # Failover auth granted to f7d6b2c72fa3b801e7dcfe0219e73383d143dd0f for epoch 285
```

31014:M 06 Sep 14:17:29.272 # Connection with slave client id #40662 lost.

31014:M 06 Sep 14:17:29.273 # Failover auth denied to 534b93af6ba45a7033dbf38c8f47cd688514125a: already voted for epoch 285

31014:M 06 Sep 14:17:29.278 * Slave 10.15.40.9:6018 asks for synchronization

31014:M 06 Sep 14:17:29.278 * Partial resynchronization request from 10.15.40.9:6018 accepted. Sending 117106 bytes of backlog starting from offset 5502756264.

13832:C 06 Sep 14:17:29.850 * DB saved on disk

13832:C 06 Sep 14:17:29.970 * RDB: 7 MB of memory used by copy-on-write

31014:M 06 Sep 14:17:38.449 * FAIL message received from f7d6b2c72fa3b801e7dcfe0219e73383d143dd0f about 1ba437fa1683a8caafd38ff977e5fbabdaf84fd6

31014:M 06 Sep 14:17:38.449 * FAIL message received from 1d07e208db56cfd7395950ca66e03589278b8e12 about d7942cfe636b25219c6d56aa72828fcfde2ee261

31014:M 06 Sep 14:17:38.449 # Failover auth denied to 938d9ae2de278938beda1d39185608b02d3b31ec: reqEpoch (286) < curEpoch(287)

31014:M 06 Sep 14:17:38.449 # Failover auth granted to d9dadf3342006e2c92def3071ca0a76390be62b0 for epoch 287

31014:M 06 Sep 14:17:38.449 * Background saving terminated with success

31014:M 06 Sep 14:17:38.450 * Clear FAIL state for node d7942cfe636b25219c6d56aa72828fcfde2ee261: master without slots is reachable again.

31014:M 06 Sep 14:17:38.450 * Clear FAIL state for node 1ba437fa1683a8caafd38ff977e5fbabdaf84fd6: master without slots is reachable again.

31014:M 06 Sep 14:17:38.452 * Clear FAIL state for node d6eb06e9d118c120d3961a659972a1d0191a8652: slave is reachable again.

31014:M 06 Sep 14:17:54.985 * FAIL message received from ae8f6e7e0ab16b04414c8f3d08b58c0aa268b467 about 7990d146cece7dc83eaf08b3e12cbebb2223f5f8

31014:M 06 Sep 14:17:56.729 * FAIL message received from 1d07e208db56cfd7395950ca66e03589278b8e12 about fbe774cdbd2acd24f9f5ea90d61c607bdf800eb5

31014:M 06 Sep 14:17:57.737 # Failover auth granted to e1c202d89ffe1c61b682e28071627635974c84a7 for epoch 288

31014:M 06 Sep 14:17:57.922 * Clear FAIL state for node fbe774cdbd2acd24f9f5ea90d61c607bdf800eb5: master without slots is reachable again.

31014:M 06 Sep 14:17:57.923 * Clear FAIL state for node 7990d146cece7dc83eaf08b3e12cbebb2223f5f8: slave is reachable again.