|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | | |  | |
|  | | | |
|  | | | |
| 方圆ok1 | | | | |
|  | Redis 4.0.1 安装文档  单实例+主从架构+集群架构 | |  | |
|  | |
|  |  | |  | |
|  |
|  | |  | |  | |

目 录

[前 言 i](#_Toc490906544)

[1 初步认识Redis 1](#_Toc490906545)

[1.1 Redis的介绍 1](#_Toc490906546)

[1.2 Redis的官方网址 2](#_Toc490906547)

[1.3 Redis部署环境选择 2](#_Toc490906548)

[1.4 Redis的安装提示 2](#_Toc490906549)

[1.5 Redis安装软件下载 3](#_Toc490906550)

[1.6 环境依赖包检查 4](#_Toc490906551)

[2 配置环境 4](#_Toc490906552)

[2.1 修改linux系统限制 4](#_Toc490906553)

[2.2 内核参数:overcommit\_memory 5](#_Toc490906554)

[2.3 内核参数:somaxconn 5](#_Toc490906555)

[2.4 内核参数:netdev\_max\_backlog 6](#_Toc490906556)

[2.5 关闭Transparent Huge Pages 6](#_Toc490906557)

[2.6 禁用防火墙和Selinux 7](#_Toc490906558)

[3 安装Redis软件 8](#_Toc490906559)

[3.1 上传并解压Redis软件包 8](#_Toc490906560)

[3.2 编译Redis 9](#_Toc490906561)

[3.3 配置全局环境（可选） 10](#_Toc490906562)

[4 配置Redis实例 10](#_Toc490906563)

[4.1 创建Redis数据库所属目录 10](#_Toc490906564)

[4.2 创建Redis配置文件 10](#_Toc490906565)

[5 使用Redis 11](#_Toc490906566)

[5.1 启动Redis数据库 11](#_Toc490906567)

[5.2 登录Redis数据库 11](#_Toc490906568)

[5.3 关闭Redis数据库 11](#_Toc490906569)

[5.4 KEY（键）的使用 12](#_Toc490906570)

[5.5 其它命令的使用和详解 12](#_Toc490906571)

[6 主从复制 12](#_Toc490906572)

[6.1 Redis主从复制的工作原理 12](#_Toc490906573)

[6.2 创建从库和对应的配置文件 13](#_Toc490906574)

[6.3 实现主从复制功能 15](#_Toc490906575)

[6.4 验证主从复制 16](#_Toc490906576)

[7 Redis Cluster 17](#_Toc490906577)

[7.1 Redis集群的介绍与目标 17](#_Toc490906578)

[7.2 Redis集群的工作原理 18](#_Toc490906579)

[7.3 Redis集群的环境准备 19](#_Toc490906580)

[7.3.1 OS环境准备 19](#_Toc490906581)

[7.3.2 Redis节点准备 21](#_Toc490906582)

[7.4 创建Redis集群 23](#_Toc490906583)

[7.5 Redis集群的使用与验证 24](#_Toc490906584)

[7.6 Redis集群的failover 26](#_Toc490906585)

[7.6.1 被动Failover 26](#_Toc490906586)

[7.6.2 手动Failover 27](#_Toc490906587)

[7.7 Redis集群的管理 29](#_Toc490906588)

[7.7.1 添加master节点 29](#_Toc490906589)

[7.7.2 添加slave节点 31](#_Toc490906590)

[7.7.3 Cluster的重新分片 32](#_Toc490906591)

[7.7.4 移除节点 34](#_Toc490906592)

[7.7.5 Slave节点的转移 35](#_Toc490906593)

# 初步认识Redis

## Redis的介绍

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件. 它支持多种类型的数据结构，如 [字符串（strings）](http://www.redis.cn/topics/data-types-intro.html#strings)， [散列（hashes）](http://www.redis.cn/topics/data-types-intro.html#hashes)， [列表（lists）](http://www.redis.cn/topics/data-types-intro.html#lists)， [集合（sets）](http://www.redis.cn/topics/data-types-intro.html#sets)， [有序集合（sorted sets）](http://www.redis.cn/topics/data-types-intro.html#sorted-sets) 与范围查询， [bitmaps](http://www.redis.cn/topics/data-types-intro.html#bitmaps)， [hyperloglogs](http://www.redis.cn/topics/data-types-intro.html#hyperloglogs) 和 [地理空间（geospatial）](http://www.redis.cn/commands/geoadd.html) 索引半径查询. Redis 内置了 [复制（replication）](http://www.redis.cn/topics/replication.html)， [LUA脚本（Lua scripting）](http://www.redis.cn/commands/eval.html)， [LRU驱动事件（LRU eviction）](http://www.redis.cn/topics/lru-cache.html)， [事务（transactions）](http://www.redis.cn/topics/transactions.html) 和不同级别的 [磁盘持久化（persistence）](http://www.redis.cn/topics/persistence.html)， 并通过 [Redis哨兵（Sentinel）](http://www.redis.cn/topics/sentinel.html) 和自动[分区（Cluster）](http://www.redis.cn/topics/cluster-tutorial.html)提供高可用性（high availability）.

你可以对这些类型执行 原子操作 ， 列如：[字符串（strings）的append 命令](http://www.redis.cn/commands/append.html); [散列（hashes）的hincrby命令](http://www.redis.cn/commands/hincrby.html); [列表（lists）的lpush命令](http://www.redis.cn/commands/lpush.html); [集合（sets）计算交集sinter命令](http://www.redis.cn/commands/sinter.html)， [计算并集union命令](http://www.redis.cn/commands/sunion.html) 和 [计算差集sdiff命令](http://www.redis.cn/commands/sdiff.html); 或者[在有序集合（sorted sets）里面获取成员的最高排名zrangebyscore命令](http://www.redis.cn/commands/zrangebyscore.html).

为了实现其卓越的性能， Redis 采用运行在 内存中的数据集工作方式. 根据您的使用情况， 您可以每隔一定时间将 [数据集导出到磁盘](http://www.redis.cn/topics/persistence.html#snapshotting) ，或者[追加到命令日志中](http://www.redis.cn/topics/persistence.html#append-only-file). 您也可以关闭持久化功能，将Redis作为一个高效的网络的缓存数据功能使用.

Redis 同样支持 [主从复制](http://www.redis.cn/topics/replication.html)（能自动重连和网络断开时自动重新同步），并且第一次同步是快速的非阻塞式的同步.

其他功能包括:

* [事务（Transactions）](http://www.redis.cn/topics/transactions.html)
* [订阅分发（Pub/Sub）](http://www.redis.cn/topics/pubsub.html)
* [LUA脚本（Lua scripting）](http://www.redis.cn/commands/eval.html)
* [过期自动删除key](http://www.redis.cn/commands/expire.html)
* [内存回收](http://www.redis.cn/topics/lru-cache.html)
* [自动故障转移](http://www.redis.cn/topics/sentinel.html)

您可以使用[大多数的编程语言](http://www.redis.cn/clients.html)来使用Redis.

Redis 使用 ANSI C 编写并且能在绝大Linux系统上运行，基于BSD协议，对OS X没有外部依赖. 我们支持Linux 和 OS X两种系统的开发和测试，我们推荐使用Linux部署. Redis 可以像SmartOS一样运行在Solaris系统中，但是我们会最大力度的支持它. 官方不支持Windos版本的Redis,但微软开发和维护着[支持win-64 的Redis](https://github.com/MSOpenTech/redis)版本.

## Redis的官方网址

官方网址：https://redis.io/

中文翻译网址：http://www.redis.cn/

## Redis部署环境选择

官方资料如下：

The Redis project does not officially support Windows. However, the Microsoft Open Tech group develops and maintains this Windows port targeting Win64.

**We suggest deploying Redis using the Linux operating system**. Redis is also tested heavily on OS X, and tested from time to time on FreeBSD and OpenBSD systems. However Linux is where we do all the major stress testing, and where most production deployments are working.

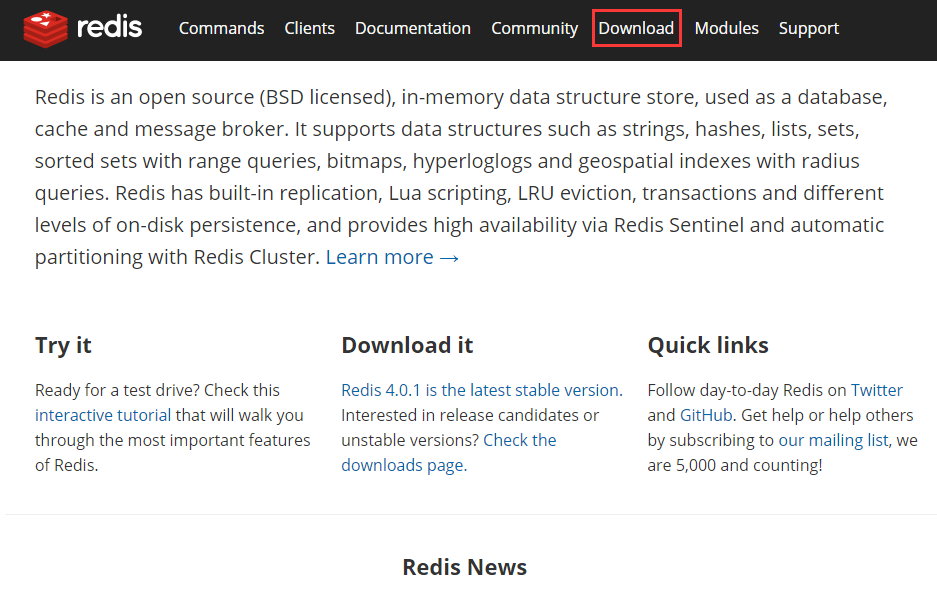
* Redis没有官方的Windows版本；
* Redis在linux系统上进行过所有主要的压力测试；
* 目前实际使用中的产品项目绝大多数是把Redis部署在linux系统上；

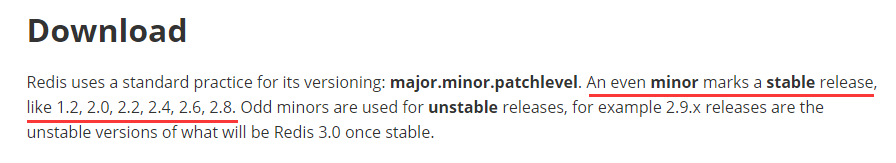
## Redis的安装提示

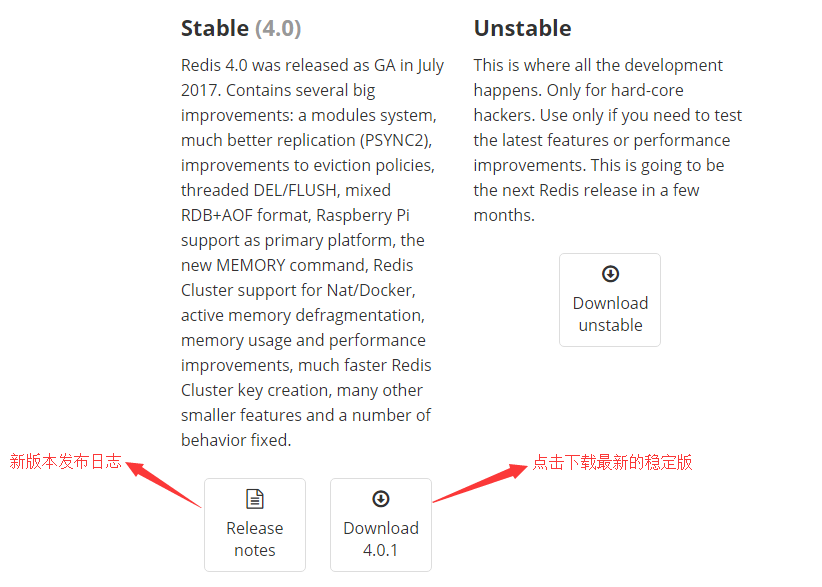
* Make sure to set the Linux kernel **overcommit memory setting to 1**.
* Make sure to **disable Linux kernel feature transparent huge pages**, it will affect greatly both memory usage and latency in a negative way.
* Make sure to **setup some swap** in your system (we suggest as much as swap as memory).
* Set an explicit maxmemory option limit in your instance in order to make sure that the instance will report errors instead of failing when the system memory limit is near to be reached.
* If you are using Redis in a very write-heavy application, while saving an RDB file on disk or rewriting the AOF log **Redis may use up to 2 times the memory normally used**.
* Even if you have persistence disabled, Redis will need to perform RDB saves if you use replication, unless you use the new diskless replication feature, which is currently experimental.
* If you are using replication, **make sure that either your master has persistence enabled, or that it does not automatically restarts on crashes**: slaves will try to be an exact copy of the master, so if a master restarts with an empty data set, slaves will be wiped as well.
* **By default Redis does not require any authentication and listens to all the network interfaces**. This is a big security issue if you leave Redis exposed on the internet or other places where attackers can reach it.

## Redis安装软件下载

登录官方网站下载最新版本的Redis软件（<https://redis.io/>）







## 环境依赖包检查

[root@Redis ~]# rpm -q make

make-3.81-20.el6.x86\_64

# 配置环境

## 修改linux系统限制

编辑/etc/security/limits.conf，添加以下内容：

# use by Redis

\* soft nproc 65536

\* hard nproc 65536

\* soft nofile 65536

\* hard nofile 65536

\* soft stack 10240

编辑/etc/pam.d/login，添加以下内容：

session required /lib/security/pam\_limits.so

## 内核参数:overcommit\_memory

overcommit\_memory：内存分配策略

可选值：0、1、2。

1. 表示内核将检查是否有足够的可用内存供应用进程使用；如果有足够的可用内存，内存申请允许；否则，内存申请失败，并把错误返回给应用进程；
2. 表示内核允许分配所有的物理内存，而不管当前的内存状态如何；
3. 表示内核允许分配超过所有物理内存和交换空间总和的内存；

OOM（out-of-memory）：

[Linux](http://lib.csdn.net/base/linux)对大部分申请内存的请求都回复"yes"，以便能跑更多更大的程序。因为申请内存后，并不会马上使用内存。这种技术叫做Overcommit。当[linux](http://lib.csdn.net/base/linux)发现内存不足时，会发生OOM killer(OOM=out-of-memory)。它会选择杀死一些进程(用户态进程，不是内核线程)，以便释放内存。

当oom-killer发生时，linux会选择杀死哪些进程？选择进程的函数是oom\_badness函数(在mm/oom\_kill.c中)，该函数会计算每个进程的点数(0~1000)。点数越高，这个进程越有可能被杀死。每个进程的点数跟oom\_score\_adj有关，而且oom\_score\_adj可以被设置(-1000最低，1000最高)。

处理方法：

编辑/etc/sysctl.conf ，添加vm.overcommit\_memory=1，然后sysctl -p 使配置文件生效；

## 内核参数:somaxconn

net.core.somaxconn是linux中的一个kernel参数，表示socket监听（listen）的backlog上限。backlog是socket的监听队列，当一个请求（request）尚未被处理或建立时，他会进入backlog。而socket server可以一次性处理backlog中的所有请求，处理后的请求不再位于监听队列中。当server处理请求较慢，以至于监听队列被填满后，新来的请求会被拒绝。所以说net.core.somaxconn限制了接收新 TCP 连接侦听队列的大小。对于一个经常处理新连接的高负载 web服务环境来说，默认的 128 太小了。大多数环境这个值建议增加到 1024 或者更多。

处理方法：

编辑/etc/sysctl.conf ，添加net.core.somaxconn=65536, 然后sysctl -p 使配置文件生效；

## 内核参数:netdev\_max\_backlog

在每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许送到队列的数据包的最大数目。

处理方法：

编辑/etc/sysctl.conf ，添加net.core.netdev\_max\_backlog=65536, 然后sysctl -p 使配置文件生效；

## 关闭Transparent Huge Pages

**To disable THP at boot time:**

Append the following to the kernel command line in grub.conf:

transparent\_hugepage=never

**example:**

# grub.conf generated by anaconda

#

# Note that you do not have to rerun grub after making changes to this file

# NOTICE: You do not have a /boot partition. This means that

# all kernel and initrd paths are relative to /, eg.

# root (hd0,0)

# kernel /boot/vmlinuz-version ro root=/dev/sda1

# initrd /boot/initrd-[generic-]version.img

#boot=/dev/sda

default=0

timeout=5

splashimage=(hd0,0)/boot/grub/splash.xpm.gz

hiddenmenu

title Red Hat Enterprise Linux (2.6.32-431.el6.x86\_64)

root (hd0,0)

kernel /boot/vmlinuz-2.6.32-431.el6.x86\_64 ro root=UUID=f09a9d3b-1b6d-4845-b413-12c8ae2893dd rd\_NO\_LUKS rd\_NO\_LVM LANG=en\_US.UTF-8 rd\_NO\_MD SYSFONT=latarcyrheb-sun16 crashkernel=128M KEYBOARDTYPE=pc KEYTABLE=us rd\_NO\_DM rhgb quiet **transparent\_hugepage=never**

initrd /boot/initramfs-2.6.32-431.el6.x86\_64.img

**OR**

Add the following lines in /etc/rc.local and reboot the server (this still can be done on Redhat 7 although rc.local is being deprecated):

if test -f /sys/kernel/mm/transparent\_hugepage/enabled; then

echo never > /sys/kernel/mm/transparent\_hugepage/enabled

fi

if test -f /sys/kernel/mm/transparent\_hugepage/defrag; then

echo never > /sys/kernel/mm/transparent\_hugepage/defrag

fi

**NOTE:** Please change the file path for RHEL kernel to

/sys/kernel/mm/redhat\_transparent\_hugepage/ accordingly.

**附：更多关于Transparent Huge Pages请查看以下文档：**



## 禁用防火墙和Selinux

编辑/etc/selinux/config Selinux=enable 修改为 selinux=disabled

service iptables stop

chkconfig iptables off

service iptables status

# 安装Redis软件

## 上传并解压Redis软件包

[root@Redis redis]# cd /u01/software/

[root@Redis software]# tar -zxvf redis-4.0.1.tar.gz

[root@Redis software]# ln -s /u01/software/redis-4.0.1 /usr/local/redis

[root@Redis software]# cd /usr/local/redis/

[root@Redis redis]# more README.md

注：README.md文件中包含了Redis的介绍、安装、使用和文件说明等信息，阅读能够进一步加深对Redis软件的理解。

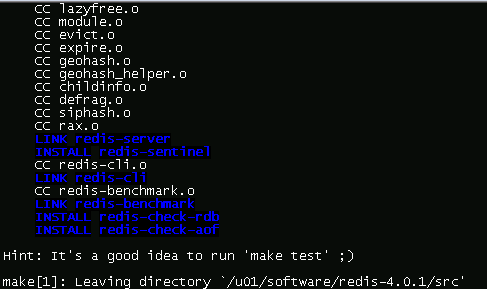


## 编译Redis

[root@Redis redis]# pwd

/usr/local/redis

[root@Redis redis]# make



## 配置全局环境（可选）

[root@Redis ~]# vi /etc/profile

加入如下内容：

export PATH

PATH=$PATH:/usr/local/redis/src

重新加载全局配置文件：

[root@Redis ~]# source /etc/profile

# 配置Redis实例

## 创建Redis数据库所属目录

[root@Redis ~]# mkdir -p /u01/redis\_27000/

[root@Redis ~]# mkdir -p /u01/redis\_config/

## 创建Redis配置文件

[root@Redis redis]# vi /u01/redis\_config/redis\_27000.conf



# 使用Redis

## 启动Redis数据库

命令： # redis-server 配置文件绝对路径

范例： [root@Redis ~]# redis-server /u01/redis\_config/redis\_27000.conf

## 登录Redis数据库

命令： # redis-cli [-h <hostname> -a <password> -p <port> -n <db\_num> ......]

范例： [root@Redis ~]# redis-cli -p 27000 –raw

## 关闭Redis数据库

命令： # redis-cli shutdown

范例： [root@Redis ~]# redis-cli -p 27000 shutdown

或

[root@Redis ~]# redis-cli -p 27000

127.0.0.1:27000> shutdown

## KEY（键）的使用

命令： set key value [EX seconds] [PX milliseconds] [NX|XX]

范例： 127.0.0.1:27000> set name jack

127.0.0.1:27000> get name

jack

## 其它命令的使用和详解

更多命令的使用方法和解析，请登录官方网址进行学习和了解：

https://redis.io/commands

或

http://www.redis.cn/commands.html

# 主从复制

## Redis主从复制的工作原理

Every Redis master has a replication ID: it is a large pseudo random string that marks a given story of the dataset. Each master also takes an offset that increments for every byte of replication stream that it is produced to be sent to slaves, in order to update the state of the slaves with the new changes modifying the dataset. The replication offset is incremented even if no slave is actually connected, so basically every given pair of:

Replication ID, offset

Identifies an exact version of the dataset of a master.

When slaves connects to master, they use the PSYNC command in order to send their old master replication ID and the offsets they processed so far. This way the master can send just the incremental part needed. However if there is not enough backlog in the master buffers, or if the slave is referring to an history (replication ID) which is no longer known, than a full resynchronization happens: in this case the slave will get a full copy of the dataset, from scratch.

This is how a full synchronization works in more details:

The master starts a background saving process in order to produce an RDB file. At the same time it starts to buffer all new write commands received from the clients. When the background saving is complete, the master transfers the database file to the slave, which saves it on disk, and then loads it into memory. The master will then send all buffered commands to the slave. This is done as a stream of commands and is in the same format of the Redis protocol itself.

You can try it yourself via telnet. Connect to the Redis port while the server is doing some work and issue the [SYNC](https://redis.io/commands/sync) command. You'll see a bulk transfer and then every command received by the master will be re-issued in the telnet session. Actually [SYNC](https://redis.io/commands/sync) is an old protocol no longer used by newer Redis instances, but is still there for backward compatibility: it does not allow partial resynchronizations, so now PSYNC is used instead.

As already said, slaves are able to automatically reconnect when the master-slave link goes down for some reason. If the master receives multiple concurrent slave synchronization requests, it performs a single background save in order to serve all of them.

**Diskless replication**

Normally a full resynchronization requires to create an RDB file on disk, then reload the same RDB from disk in order to feed the slaves with the data.

With slow disks this can be a very stressing operation for the master. Redis version 2.8.18 is the first version to have support for diskless replication. In this setup the child process directly sends the RDB over the wire to slaves, without using the disk as intermediate storage.

## 创建从库和对应的配置文件

[root@Redis ~]# mkdir -p /u01/redis\_27001/

[root@Redis ~]# vi /u01/redis\_config/redis\_27001.conf

在标准配置文件下加入如下内容：

# 指定master的IP和PORT

# slaveof <masterip> <masterport>

slaveof 192.168.5.134 27000

# 如果master指定了密码，则在这里输入

# masterauth <master-password>

# 当slave丢失master或者同步正在进行时，如果发生对slave的服务请求：

# yes：依旧正常对外服务

# no：返回报错信息“SYNC with master in progress”，除了INFO and SLAVEOF命令

slave-serve-stale-data yes

# slave是否设置成只读模式（2.6版本开始默认为只读模式）

slave-read-only yes

# 无硬盘复制功能还处于测试阶段；

# master创建一个进程不会将数据存储在磁盘中，而是直接发送给slave，其他操作一致

# 在一个硬盘比较慢，网络比较快的环境下

# 无硬盘复制功能效率更加好

repl-diskless-sync no

# 配置从库开始复制的延迟时间（单位：秒）

repl-diskless-sync-delay 5

# slave发送ping到master的时间间隔（单位：秒）

# 默认是10秒

# repl-ping-slave-period 10

# 从库同步超时时间

# repl-timeout 60

# 在slave和master同步后（发送psync/sync），后续的同步是否设置成TCP\_NODELAY

# 假如设置成yes，则redis会合并小的TCP包从而节省带宽

# 但会增加同步延迟**（40ms）**，造成master与slave数据不一致

# 假如设置成no，则redis master会立即发送同步数据，没有延迟

# **前者关注性能，后者关注一致性**

repl-disable-tcp-nodelay no

# 设置backlog的大小，backlog是一个缓冲区，在slave端失连时存放要同步到slave的数据

# 因此当一个slave要重连时，经常是不需要完全同步的，执行局部同步就足够了

# backlog设置的越大，slave可以失连的时间就越长

# repl-backlog-size 1mb

# 如果一段时间后没有slave连接到master，则backlog size的内存将会被释放

# 如果值为0则表示永远不释放这部份内存

# repl-backlog-ttl 3600

# slave端的优先级设置，值是一个整数，数字越小表示优先级越高

# 当master故障时将会按照优先级来选择slave端进行恢复

# 如果值设置为0，则表示该slave永远不会被选择

slave-priority 100

# 设置当master可用的slave少于N个，或延迟时间大于M秒，不接受写操作

# min-slaves-to-write 3

# min-slaves-max-lag 10

# 通过设置以下参数，slave通过不同的IP和PORT到达master

# slave-announce-ip 5.5.5.5

# slave-announce-port 1234

## 实现主从复制功能

启动Redis从库：

[root@Redis ~]# redis-server /etc/redis\_27001.conf

检查Redis监听端口：

[root@Redis redis]# ps -ef | grep redis

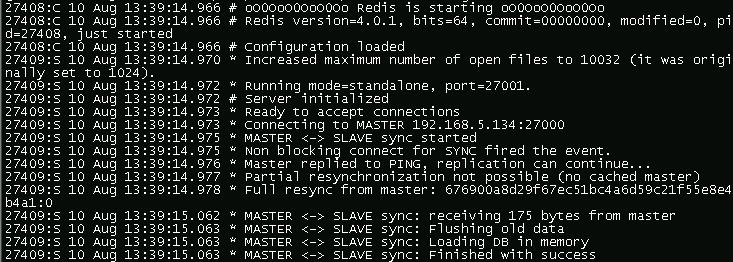
root 27404 1 0 13:39 ? 00:00:08 redis-server \*:27000

root 27409 1 0 13:39 ? 00:00:04 redis-server \*:27001

root 27503 27373 1 13:58 pts/0 00:00:00 grep redis

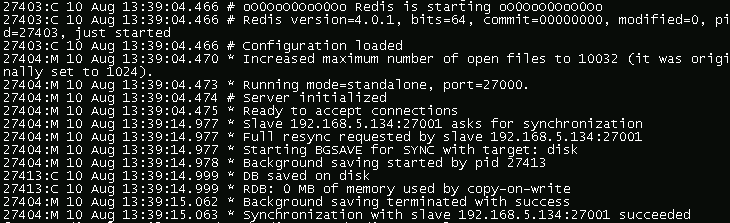
检查从库日志：

[root@Redis ~]# tail -100 /u01/redis\_27001/redis\_27001.log



检查主库日志：

[root@Redis ~]# tail -100 /u01/redis\_27000/redis\_27000.log



## 验证主从复制

主库设置KEY：

[root@Redis ~]# redis-cli -p 27000 --raw

127.0.0.1:27000> set 手机号码 15013135743

OK

127.0.0.1:27000> get 手机号码

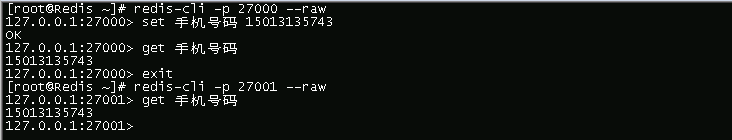
15013135743

从库查询对应KEY：

[root@Redis ~]# redis-cli -p 27001 --raw

127.0.0.1:27001> get 手机号码

15013135743



# Redis Cluster

**参考资料：**

<https://redis.io/topics/cluster-tutorial> （**Redis cluster tutorial**）

https://redis.io/topics/cluster-spec （**Redis Cluster Specification**）

## Redis集群的介绍与目标

Redis 集群是一个提供在**多个Redis间节点间共享数据**的程序集。Redis集群并不支持处理多个keys的命令,因为这需要在不同的节点间移动数据,从而达不到像Redis那样的性能,在高负载的情况下可能会导致不可预料的错误.

Redis 集群通过分区来提供**一定程度的可用性**,在实际环境中当某个节点宕机或者不可达的情况下继续处理命令. Redis 集群的优势:

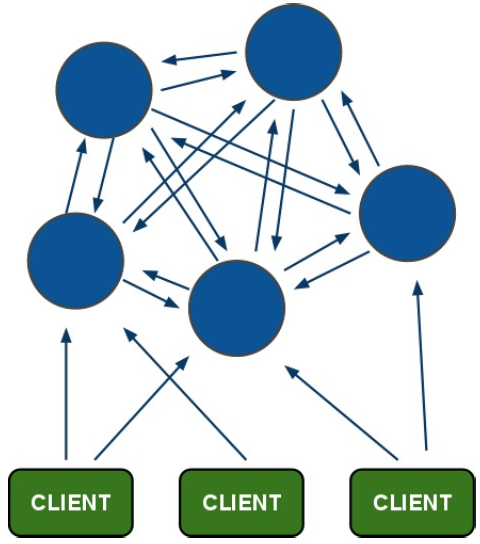
* 自动分割数据到不同的节点上。
* 整个集群的部分节点失败或者不可达的情况下能够继续处理命令。

**Redis Cluster goals**

Redis Cluster is a distributed implementation of Redis with the following goals, in order of importance in the design:

* High performance and linear scalability up to 1000 nodes. There are no proxies, asynchronous replication is used, and no merge operations are performed on values.
* Acceptable degree of write safety: the system tries (in a best-effort way) to retain all the writes originating from clients connected with the majority of the master nodes. Usually there are small windows where acknowledged writes can be lost. Windows to lose acknowledged writes are larger when clients are in a minority partition.
* Availability: Redis Cluster is able to survive partitions where the majority of the master nodes are reachable and there is at least one reachable slave for every master node that is no longer reachable. Moreover using *replicas migration*, masters no longer replicated by any slave will receive one from a master which is covered by multiple slaves.

## Redis集群的工作原理



**Redis 集群的数据分片**

Redis 集群没有使用一致性hash, 而是引入了 **哈希槽**的概念.Redis 集群有16384个哈希槽,每个key通过CRC16校验后对16384取模来决定放置哪个槽.集群的每个节点负责一部分hash槽,举个例子,比如当前集群有3个节点,那么:

* 节点 A 包含 0 到 5500号哈希槽.
* 节点 B 包含5501 到 11000 号哈希槽.
* 节点 C 包含11001 到 16384号哈希槽.

这种结构很容易添加或者删除节点. 比如如果我想新添加个节点D, 我需要从节点 A, B, C中得部分槽到D上. 如果我像移除节点A,需要将A中得槽移到B和C节点上,然后将没有任何槽的A节点从集群中移除即可. 由于从一个节点将哈希槽移动到另一个节点并不会停止服务,所以无论添加删除或者改变某个节点的哈希槽的数量都不会造成集群不可用的状态.

**Redis 集群的主从复制模型**

为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型,每个节点都会有N-1个复制品.

在我们例子中具有A，B，C三个节点的集群,在没有复制模型的情况下,如果节点B失败了，那么整个集群就会以为缺少5501-11000这个范围的槽而不可用.

然而如果在集群创建的时候（或者过一段时间）我们为每个节点添加一个从节点A1，B1，C1,那么整个集群便有三个master节点和三个slave节点组成，这样在节点B失败后，集群便会选举B1为新的主节点继续服务，整个集群便不会因为槽找不到而不可用了, 不过当B和B1 都失败后，集群是不可用的.

**Redis 一致性保证**

Redis 并不能保证数据的**强一致性**. 这意味这在实际中集群在特定的条件下可能会丢失写操作。第一个原因是因为集群是用了异步复制. 写操作过程：

* 客户端向主节点B写入一条命令.
* 主节点B向客户端回复命令状态.
* 主节点将写操作复制给他得从节点 B1, B2 和 B3.

主节点对命令的复制工作发生在返回命令回复之后， 因为如果每次处理命令请求都需要等待复制操作完成的话， 那么主节点处理命令请求的速度将极大地降低 —— 我们必须在性能和一致性之间做出权衡。 注意：Redis 集群可能会在将来提供同步写的方法。 Redis 集群另外一种可能会丢失命令的情况是集群出现了网络分区， 并且一个客户端与至少包括一个主节点在内的少数实例被孤立。

## Redis集群的环境准备

### OS环境准备

1. redis软件：（本文已介绍如何安装Redis软件）；
2. ruby：

redis 4.0.1版本的集群创建需要安装ruby 1.9+版本以上，RHEL6通过yum install ruby\* 安装的ruby版本是1.8.7，后续执行redis-trib.rb脚本的时候会出现以下报错：

undefined method `require\_relative' for main:Object

其中'require\_relative' is only available for ruby 1.9+；

下载最新版本的ruby并安装：

http://www.ruby-lang.org/zh\_cn/downloads/



# cd /u01/software/

# tar -zxvf ruby-2.4.1.tar.gz

# cd ruby-2.4.1

# ./configure --prefix=/usr/local/bin/ruby-2.4.1

# make

# make install

备注：如系统已有旧版本的ruby，则需要设置新版本的环境变量。

1. rubygems：

rubygems包依赖于ruby包，下载后上传到服务器安装。

https://rubygems.org/pages/download



# cd /u01/software/

# unzip rubygems-2.6.12.zip

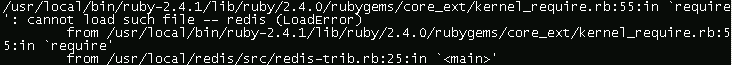


# ruby setup.rb



1. gem对应的redis接口包：

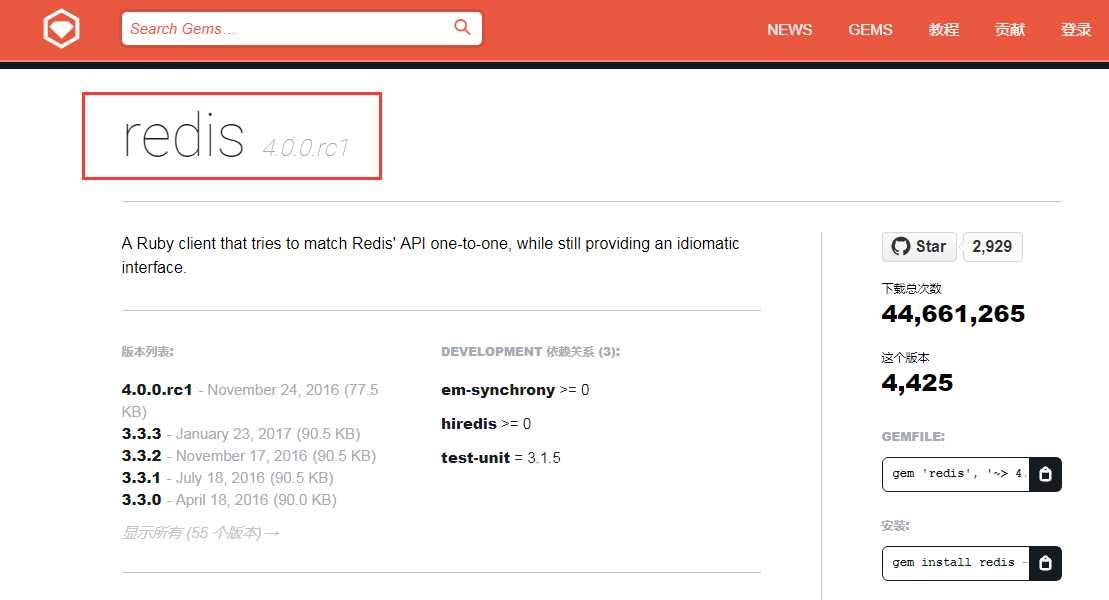
如果不安装对应Redis的接口包，在创建集群的时候会出现以下报错：



1. 打开gem网站下载接口包：https://rubygems.org/
2. 输入redis查找对应接口包：



1. 找到对应的接口包下载：



1. 上传至服务器进行安装：

gem install redis-4.0.0.rc1.gem -l

### Redis节点准备

搭建集群之前我们需要一些开启在集群模式下的实例，要让集群正常运作至少需要三个主节点，为保证集群的高可用性强，烈建议使用六个节点：即其中三个为主节点， 而其余三个则是各个主节点的从节点。

创建节点所属目录和配置文件：

[root@Redis u01]# mkdir redis\_28000 redis\_28001 redis\_28002 redis\_28003 redis\_28004 redis\_28005

[root@Redis ~]# cd redis\_config/

[root@Redis ~]# touch redis\_28000.conf redis\_28001.conf redis\_28002.conf redis\_28003.conf redis\_28004.conf redis\_28005.conf

在各个节点的标准配置文件上添加如下内容：

**Example（redis\_28000.conf）：**

cluster-enabled yes

cluster-node-timeout 15000

cluster-config-file nodes\_28000.conf

cluster-slave-validity-factor 10

cluster-migration-barrier 1

cluster-require-full-coverage yes

**附：集群参数解释**

* **cluster-enabled <yes/no>**: If yes enables Redis Cluster support in a specific Redis instance. Otherwise the instance starts as a stand alone instance as usually.
* **cluster-config-file <filename>**: Note that despite the name of this option, this is not an user editable configuration file, but the file where a Redis Cluster node automatically persists the cluster configuration (the state, basically) every time there is a change, in order to be able to re-read it at startup. The file lists things like the other nodes in the cluster, their state, persistent variables, and so forth. Often this file is rewritten and flushed on disk as a result of some message reception.
* **cluster-node-timeout <milliseconds>**: The maximum amount of time a Redis Cluster node can be unavailable, without it being considered as failing. If a master node is not reachable for more than the specified amount of time, it will be failed over by its slaves. This parameter controls other important things in Redis Cluster. Notably, every node that can't reach the majority of master nodes for the specified amount of time, will stop accepting queries.
* **cluster-slave-validity-factor <factor>**: If set to zero, a slave will always try to failover a master, regardless of the amount of time the link between the master and the slave remained disconnected. If the value is positive, a maximum disconnection time is calculated as the *node timeout* value multiplied by the factor provided with this option, and if the node is a slave, it will not try to start a failover if the master link was disconnected for more than the specified amount of time. For example if the node timeout is set to 5 seconds, and the validity factor is set to 10, a slave disconnected from the master for more than 50 seconds will not try to failover its master. Note that any value different than zero may result in Redis Cluster to be unavailable after a master failure if there is no slave able to failover it. In that case the cluster will return back available only when the original master rejoins the cluster.
* **cluster-migration-barrier <count>**: Minimum number of slaves a master will remain connected with, for another slave to migrate to a master which is no longer covered by any slave. See the appropriate section about replica migration in this tutorial for more information.
* **cluster-require-full-coverage <yes/no>:** If this is set to yes, as it is by default, the cluster stops accepting writes if some percentage of the key space is not covered by any node. If the option is set to no, the cluster will still serve queries even if only requests about a subset of keys can be processed.

启动6个集群模式下的实例：

[root@Redis ~]# redis-server /u01/redis\_config/redis\_28000.conf

[root@Redis ~]# redis-server /u01/redis\_config/redis\_28001.conf

[root@Redis ~]# redis-server /u01/redis\_config/redis\_28002.conf

[root@Redis ~]# redis-server /u01/redis\_config/redis\_28003.conf

[root@Redis ~]# redis-server /u01/redis\_config/redis\_28004.conf

[root@Redis ~]# redis-server /u01/redis\_config/redis\_28005.conf

# ps -ef | grep redis

root 17582 1 0 Aug08 ? 00:06:20 redis-server \*:28000 **[cluster]**

root 17584 1 0 Aug08 ? 00:06:21 redis-server \*:28001 **[cluster]**

root 17592 1 0 Aug08 ? 00:06:19 redis-server \*:28002 **[cluster]**

root 17594 1 0 Aug08 ? 00:06:24 redis-server \*:28003 **[cluster]**

root 17599 1 0 Aug08 ? 00:06:18 redis-server \*:28004 **[cluster]**

root 17604 1 0 Aug08 ? 00:06:18 redis-server \*:28005 **[cluster]**

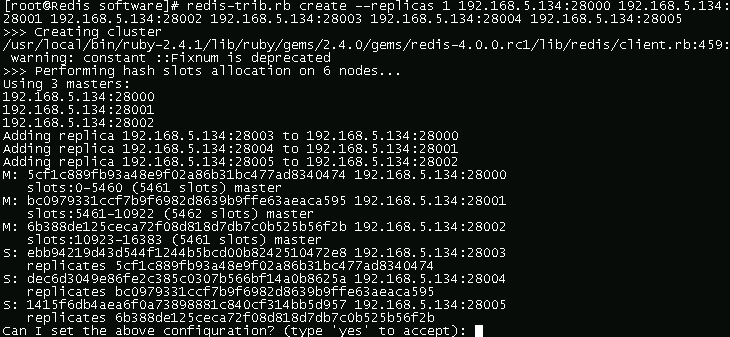
## 创建Redis集群

现在我们已经有了六个正在运行中的 Redis 实例， 接下来我们需要使用这些实例来创建集群， 并为每个节点编写配置文件。

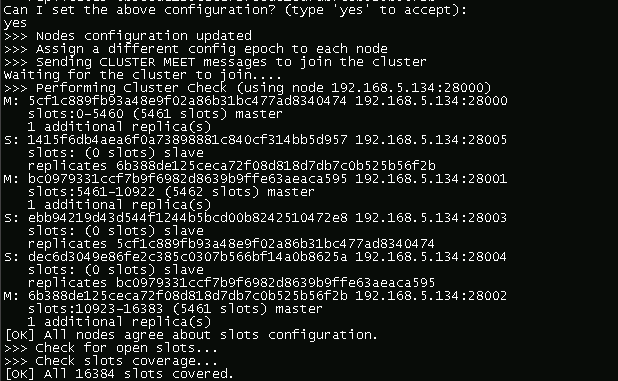
通过使用 Redis 集群命令行工具 redis-trib ， 编写节点配置文件的工作可以非常容易地完成： redis-trib 位于 Redis 源码的 src 文件夹中， 它是一个 Ruby 程序， 这个程序通过向实例发送特殊命令来完成创建新集群，检查集群，或者对集群进行重新分片（reshared）等工作。

通过命令redis-trib.rb对6个实例进行集群创建：

[root@Redis ~]# redis-trib.rb create --replicas 1 192.168.5.134:28000 192.168.5.134:28001 192.168.5.134:28002 192.168.5.134:28003 192.168.5.134:28004 192.168.5.134:28005



键入yes，等待完成配置：



至此，Redis集群创建完毕。

## Redis集群的使用与验证

**我们继续使用redis-cli测试Redis Cluster：**

[root@Redis u01]# redis-cli -p 28000 --raw

127.0.0.1:28000> set name jack

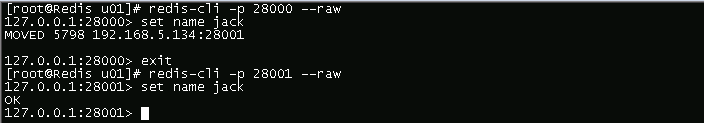
MOVED 5798 192.168.5.134:28001 --提示我们需要到28001节点上进行操作

127.0.0.1:28000> exit

[root@Redis u01]# redis-cli -p 28001 --raw

127.0.0.1:28001> set name jack

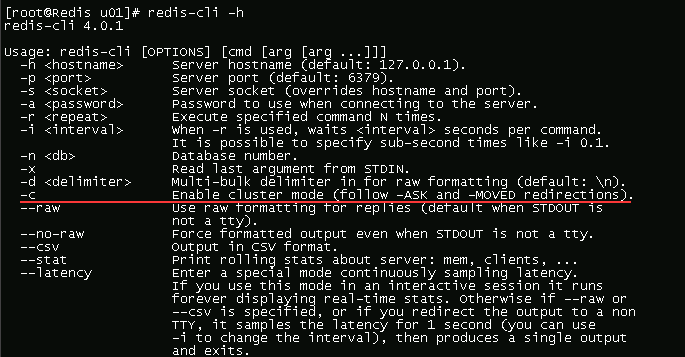
OK



**解释：**

由于KEY：name通过CRC16校验后对16384取模后为：5789，需要放置到节点B上（即28001节点），所以提示我们需要在节点28001上操作。

**命令redis-cli在集群模式下的便捷使用：**



使用-c 参数开启集群模式：

[root@Redis u01]# redis-cli -p 28000 --raw -c

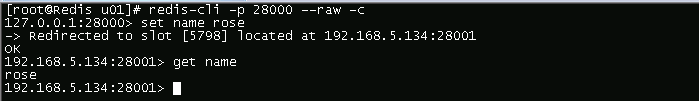
127.0.0.1:28000> set name rose

-> Redirected to slot [5798] located at 192.168.5.134:28001 --自动跳转到28001节点

OK

192.168.5.134:28001> get name

Rose



**通过cluster nodes命令查看节点详细信息：**

192.168.5.134:28001> cluster nodes

1415f6db4aea6f0a73898881c840cf314bb5d957 192.168.5.134:28005@38005 slave 6b388de125ceca72f08d818d7db7c0b525b56f2b 0 1502350563000 6 connected

bc0979331ccf7b9f6982d8639b9ffe63aeaca595 192.168.5.134:28001@38001 myself,master - 0 1502350561000 9 connected 5461-10922

5cf1c889fb93a48e9f02a86b31bc477ad8340474 192.168.5.134:28000@38000 master - 0 1502350563000 1 connected 0-5460

6b388de125ceca72f08d818d7db7c0b525b56f2b 192.168.5.134:28002@38002 master - 0 1502350563599 3 connected 10923-16383

ebb94219d43d544f1244b5bcd00b8242510472e8 192.168.5.134:28003@38003 slave 5cf1c889fb93a48e9f02a86b31bc477ad8340474 0 1502350564619 4 connected

dec6d3049e86fe2c385c0307b566bf14a0b8625a 192.168.5.134:28004@38004 slave bc0979331ccf7b9f6982d8639b9ffe63aeaca595 0 1502350562000 9 connected

CLUSTER NODES 命令的输出看起来有点复杂,其实他非常的简单，含义如下:

* 节点ID
* IP:端口
* 标志: master, slave, myself, fail, …
* 如果是个从节点, 这里是它的主节点的NODE ID
* 集群最近一次向节点发送 PING 命令之后， 过去了多长时间还没接到回复。.
* 节点最近一次返回 PONG 回复的时间。
* 节点的配置纪元（configuration epoch）：详细信息请参考 Redis 集群规范 。
* 本节点的网络连接情况：例如 connected 。
* 节点目前包含的槽：例如 127.0.0.1:7001 目前包含号码为 5960 至 10921 的哈希槽。

## Redis集群的failover

### 被动Failover

通过kill -9 pid 杀死28000主节点进程，模拟master fail的情景：

[root@Redis ~]# ps -ef | grep redis

root 17582 1 0 Aug08 ? 00:06:30 redis-server \*:28000 [cluster]

root 17584 1 0 Aug08 ? 00:06:30 redis-server \*:28001 [cluster]

root 17592 1 0 Aug08 ? 00:06:29 redis-server \*:28002 [cluster]

root 17594 1 0 Aug08 ? 00:06:34 redis-server \*:28003 [cluster]

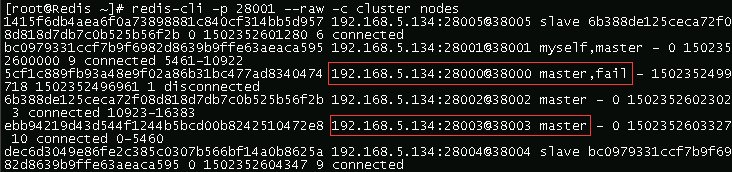
root 17599 1 0 Aug08 ? 00:06:27 redis-server \*:28004 [cluster]

root 17604 1 0 Aug08 ? 00:06:27 redis-server \*:28005 [cluster]

[root@Redis ~]# kill -9 17582

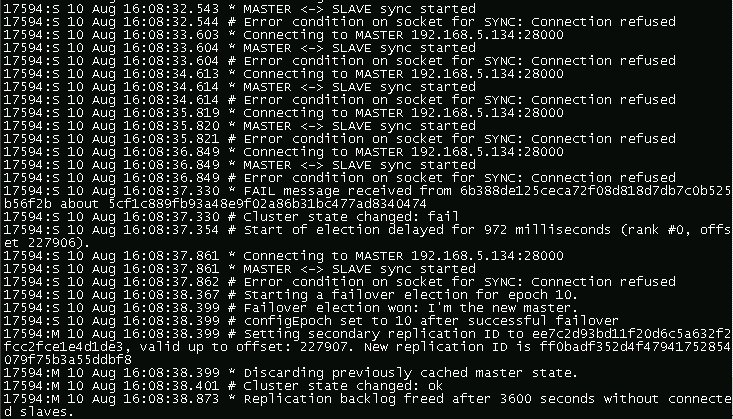
查看Redis Cluster状态：

[root@Redis ~]# redis-cli -p 28001 --raw -c cluster nodes



查看28000主节点的slave日志：

[root@Redis ~]# tail -100 /u01/redis\_28003/redis\_28003.log



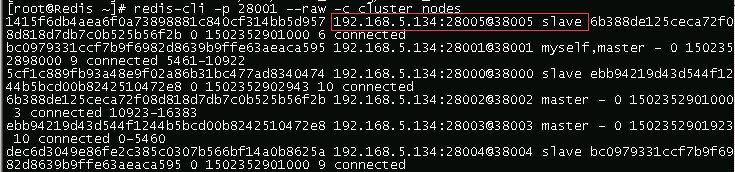
由此可见，kill掉28000主节点后，28003通过connecting fail一定次数后，自动Failover成为master。

### 手动Failover

重新启动28000节点：

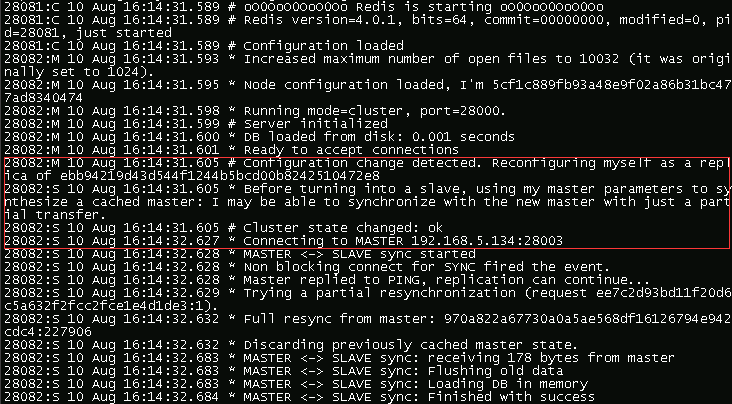
[root@Redis ~]# redis-server /u01/redis\_config/redis\_28000.conf

查看Redis Cluster状态：



查看28000节点的数据库日志：

[root@Redis ~]# tail -100 /u01/redis\_28000/redis\_28000.log



28000节点在重启后自动重新加入到集群，并成为28003的Slave。

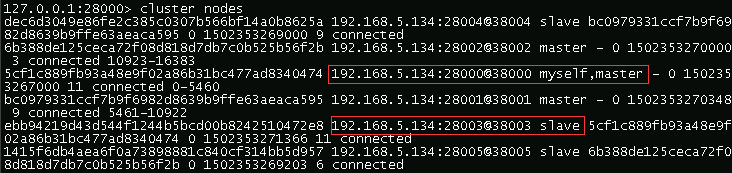
通过命令CLUSTER FAILOVER进行故障转移（将要被转移主节点的从节点上执行该命令）：

[root@Redis ~]# redis-cli -c -p 28000 --raw

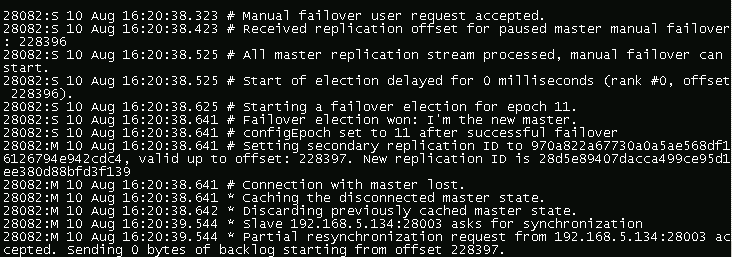
127.0.0.1:28000> cluster failover

OK

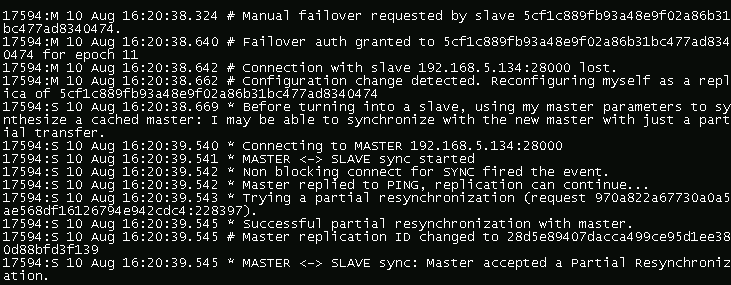
查看Redis Cluster状态：



查看28000节点日志：



查看28003节点日志：



## Redis集群的管理

添加新的节点的基本过程就是添加一个空的节点然后移动一些数据给它，有两种情况，添加一个主节点和添加一个从节点（添加从节点时需要将这个新的节点设置为集群中某个节点的复制）。两种情况第一步都是要添加 一个空的节点.

### 添加master节点

创建一个新的并且启动在集群底下的redis实例：

[root@Redis ~]# cd /u01/

[root@Redis u01]# mkdir redis\_28006

[root@Redis u01]# cd redis\_config/

[root@Redis redis\_config]# cp redis\_28000.conf redis\_28006.conf

[root@Redis redis\_config]# vi redis\_28006.conf **--（修改对应的参数）**

[root@Redis redis\_config]# redis-server /u01/redis\_config/redis\_28006.conf

[root@Redis redis\_config]# ps -ef | grep redis

root 17584 1 0 Aug08 ? 00:06:35 redis-server \*:28001 [cluster]

root 17592 1 0 Aug08 ? 00:06:35 redis-server \*:28002 [cluster]

root 17594 1 0 Aug08 ? 00:06:38 redis-server \*:28003 [cluster]

root 17599 1 0 Aug08 ? 00:06:32 redis-server \*:28004 [cluster]

root 17604 1 0 Aug08 ? 00:06:33 redis-server \*:28005 [cluster]

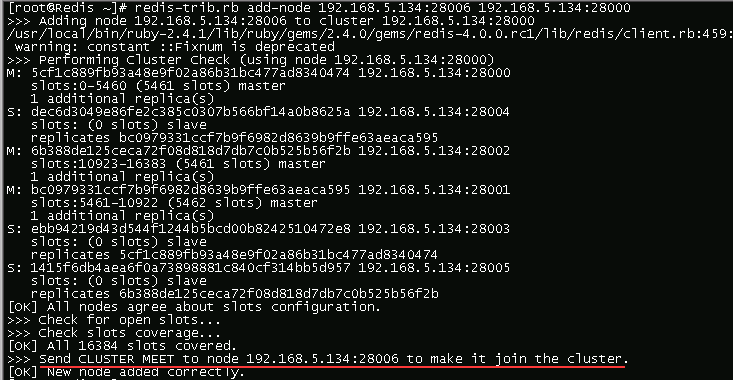
root 28082 1 0 16:14 ? 00:00:03 redis-server \*:28000 [cluster]

root 28175 1 0 16:33 ? 00:00:00 redis-server \*:28006 [cluster]

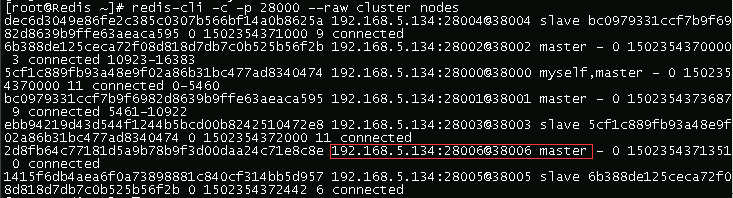
执行redis-trib.rb添加新的master节点：

命令语法：redis-trib.rb add-node **新节点ip:port** **任意一个已经存在的节点的ip:port**

[root@Redis ~]# redis-trib.rb add-node 192.168.5.134:28006 192.168.5.134:28000



查看Redis Cluster状态：



### 添加slave节点

创建一个新的并且启动在集群底下的redis实例：

[root@Redis ~]# cd /u01/

[root@Redis u01]# mkdir redis\_28007

[root@Redis u01]# cd redis\_config/

[root@Redis redis\_config]# cp redis\_28000.conf redis\_28007.conf

[root@Redis redis\_config]# vi redis\_28007.conf **--（修改对应的参数）**

[root@Redis redis\_config]# redis-server /u01/redis\_config/redis\_28007.conf

[root@Redis redis\_config]# ps -ef | grep redis

root 17584 1 0 Aug08 ? 00:06:35 redis-server \*:28001 [cluster]

root 17592 1 0 Aug08 ? 00:06:35 redis-server \*:28002 [cluster]

root 17594 1 0 Aug08 ? 00:06:38 redis-server \*:28003 [cluster]

root 17599 1 0 Aug08 ? 00:06:32 redis-server \*:28004 [cluster]

root 17604 1 0 Aug08 ? 00:06:33 redis-server \*:28005 [cluster]

root 28082 1 0 16:14 ? 00:00:03 redis-server \*:28000 [cluster]

root 28175 1 0 16:33 ? 00:00:00 redis-server \*:28006 [cluster]

root 28224 1 0 16:44 ? 00:00:00 redis-server \*:28007 [cluster]

执行redis-trib.rb添加新的slave节点：

命令语法：

redis-trib.rb add-node --slave --master-id 该从节点的主节点ID **新节点ip:port** **任意一个已经存在的节点的ip:port**

或

redis-trib.rb add-node --slave **新节点ip:port** **任意一个已经存在的节点的ip:port**

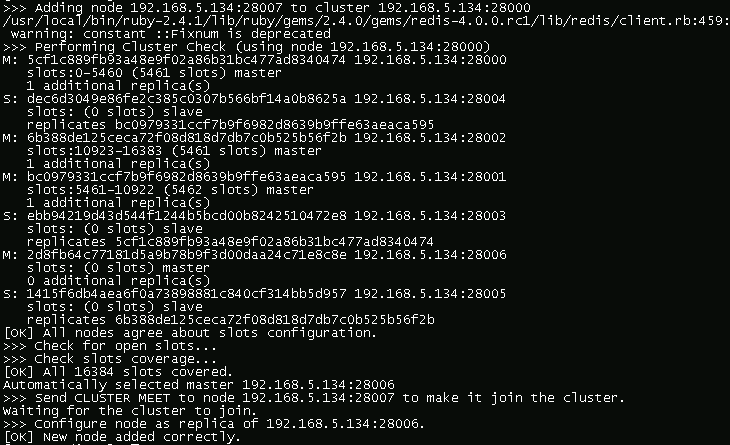
127.0.0.1:7006> cluster replicate 该从节点的主节点ID

[root@Redis ~]# redis-trib.rb add-node --slave 192.168.5.134:28007 192.168.5.134:28000

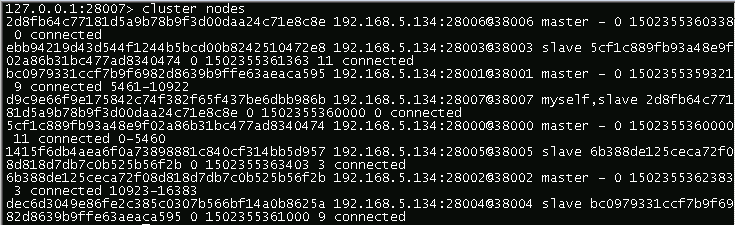
[root@Redis ~]# redis-cli -c -p 28007 --raw

127.0.0.1:28007> cluster replicate 2d8fb64c77181d5a9b78b9f3d00daa24c71e8c8e

OK

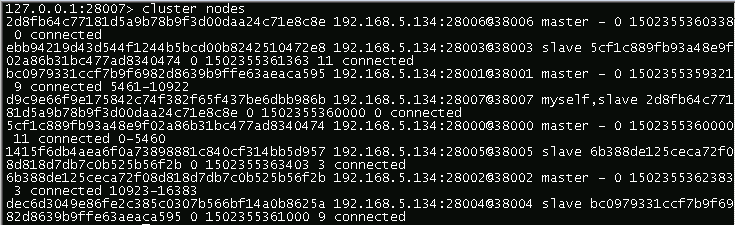


查看Redis Cluster状态：



### Cluster的重新分片

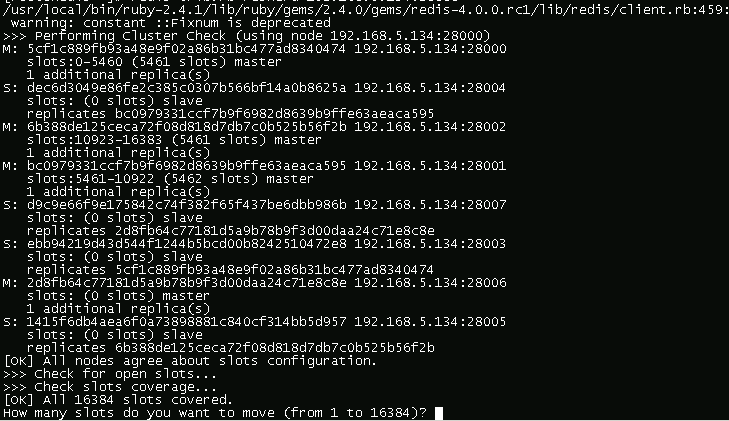
查看Redis Cluster状态：



目前看来刚才新添加的28006主节点，并没有获取到hash slots，此时我们需要重新分片。 重新分片操作基本上就是将某些节点上的哈希槽移动到另外一些节点上面， 和创建集群一样， 重新分片也可以使用 redis-trib 程序来执行 执行以下命令可以开始一次重新分片操作：

语法：redis-trib.rb reshard **任意一个已经存在的节点的ip:port**

[root@Redis ~]# redis-trib.rb reshard 192.168.5.134:28000



依次输入：

1. 想要移动的hash slots数量

1000

1. 接受hash slots的master-id

2d8fb64c77181d5a9b78b9f3d00daa24c71e8c8e --28006主节点

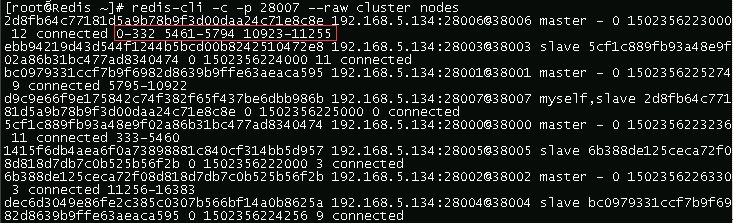
1. 从哪些master上进行移动（全部则输入all）

all

1. 最后确认是否按照计划执行

yes

查看Redis Cluster状态：



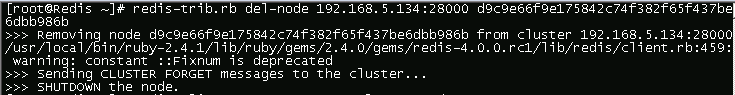
Cluster的hash slots Reshareding成功！

### 移除节点

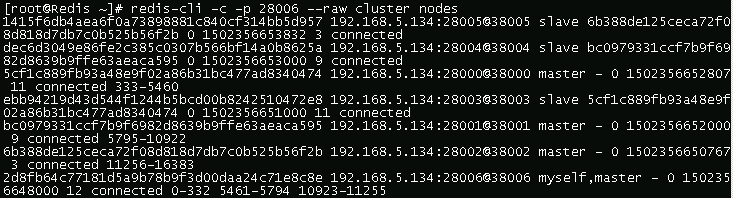
使用命令redis-trib.rb进行节点的移除：

语法：redis-trib.rb del-node **任意一个已经存在的节点的ip:port node-id**

[root@Redis ~]# redis-trib.rb del-node 192.168.5.134:28000 d9c9e66f9e175842c74f382f65f437be6dbb986b



查看Redis Cluster状态：



28007-slave节点已经成功移除！

备注：

使用同样的方法移除主节点,不过在移除主节点前，需要确保这个主节点是空的. 如果不是空的,需要将这个节点的数据重新分片到其他主节点上。

### Slave节点的转移

在Redis集群中会存在改变一个从节点的主节点的情况，需要执行如下命令：

进入slave节点：

[root@Redis ~]# redis-cli -c -p 28007 --raw

查看cluster nodes信息：

127.0.0.1:28007> cluster nodes



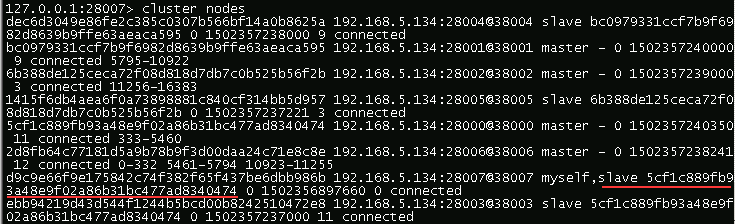
将slave节点28007的master节点28006转移成28000：

127.0.0.1:28007> cluster replicate 5cf1c889fb93a48e9f02a86b31bc477ad8340474

OK

查看cluster nodes信息：

127.0.0.1:28007> cluster nodes



slave 28007的master节点已经从28006转移到28000！