redis 集群部署、维护以及 115 项目 redis 集群需求测试说明

redis 集群相关说明

redis 支持的 cluster 特性:

1):节点自动发现

2):slave->master 选举,集群容错

3):Hot resharding:在线分片

4):进群管理:cluster xxx

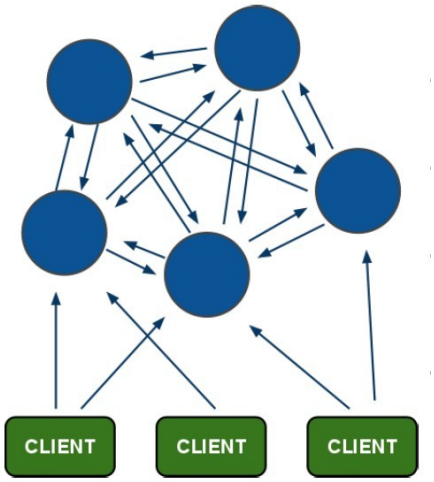
5):基于配置(nodes-port.conf)的集群管理

6):ASK 转向/MOVED 转向机制.

redis cluster 架构

redis-cluster 架构图架构细节:

1) redis-cluster 架构图



架构细节:

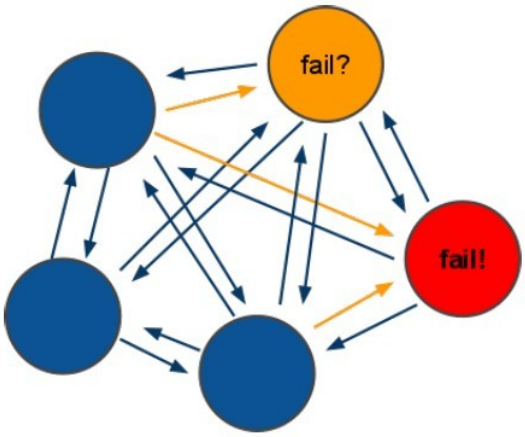
(1)所有的 redis 节点彼此互联(PING-PONG 机制),内部使用二进制协议优化传输速度和带宽.

(2)节点的 fail 是通过集群中超过半数的节点检测失效时才生效.

(3)客户端与 redis 节点直连,不需要中间 proxy 层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可

(4)redis-cluster 把所有的物理节点映射到[0-16383]slot 上,cluster 负责维护 node<->slot<->value

2）redis-cluster 选举:容错



(1)领着选举过程是集群中所有 master 参与,如果半数以上 master 节点与 master 节点通信超过(cluster-node-timeout),认为当前 master 节点挂掉.

(2):什么时候整个集群不可用(cluster_state:fail),当集群不可用时,所有对集群的操作做都不可用，收到((error) CLUSTERDOWN The cluster is down)错误

a:如果集群任意 master 挂掉,且当前 master 没有 slave.集群进入 fail 状态,也可以理解成进群的 slot 映射[0-16383]不完成时进入 fail 状态.

b:如果进群超过半数以上 master 挂掉，无论是否有 slave 集群进入 fail 状态.

redis 集群部署

部署前的关于部署的说明：

首先明确，redis 集群可以在集群内任意一台服务器设备上启动;

其次明确，启动 redis 集群需要一些辅助类工具，鉴于此，在你想在某个终端上启动 redis 集群的时候必须要确认相应的辅助工具是否已经成功安装，并能够正常使用;

最后明确部署需要准备软件包（包含辅助类工具安装包）。

开始部署：

安装

1):安装 redis-cluster 依赖;

纵然 redis-cluster 的依赖库在使用时有兼容问题,安装过程也不必太拘泥于版本，但前提是：注意所安装依赖库版本的要求，一般情况下，首选稳定版。

(1)确保系统安装 zlib,否则 gem install 会报(no such file to load -- zlib)

```
#下载 zlib-1.2.6.tar
tar xf  zlib-1.2.6.tar
cd zlib-1.2.6
./configure
make
make install
```

(2)安装 ruby（如果系统已经默认安装了 ruby，该步骤可以省略）

```
# 下载 ruby 源码安装包
tar xf ruby.tar.gz
cd ruby
./configure -prefix=/usr/local/ruby
make
make install
sudo cp ruby /usr/local/bin
```

(3)安装 rubygem

```
# 下载 rubygems.tgz
tar xf rubygems.tar.gz
cd rubygems
sudo ruby setup.rb
sudo cp bin/gem /usr/local/bin
```

(3)安装 jscon_pure.gem、redis.gem

有两种安装方式，分别是：

a.离线安装：

到该网址（http://rubygems.org/gems/）下，搜索、下载合适的 gem 文件

cd 到下载好的 gem 目录下，执行如下命令：

```
gem install -l jscon_pure.gem
gem install -l redis.gem
```

b.在线安装：

```
gem install redis --version 版本号
gem install jscon_pure --version   版本号
```

2)安装 redis

```
tar xf redis.tar.gz
```

```
cd redis
make
sudo make install
sudo cp src/redis-trib.rb /usr/local/bin
```

配置：

1)配置说明：

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

按照需求编辑 redis 配置文件，所以有必要先介绍一下 redis 的配置文件。

#是否在后台执行，yes：后台运行；no：不是后台运行（老版本默认）

daemonize yes

#3.2 里的参数，是否开启保护模式，默认开启。要是配置里没有指定 bind 和密码。开启该参数后，redis 只会本地进行访问，拒绝外部访问。要是开启了密码　和 bind，可以开启。否　则最好关闭，设置为 no。

protected-mode yes

#redis 的进程文件

pidfile /var/run/redis/redis-server.pid

#redis 监听的端口号。

port 6379

#此参数确定了 TCP 连接中已完成队列(完成三次握手之后)的长度，　当然此值必须不大于 Linux 系统定义的/proc/sys/net/core/somaxconn 值，默认是 511，而 Linux 的默认参数值是 128。当系统并发量大并且客户端速度缓慢的时候，可以将这二个参数一起参考设定。该内核参数默认值一般是 128，对于负载很大的服务程序来说大大的不够。一般会将它修改为 2048 或者更大。在/etc/sysctl.conf 中添加:net.core.somaxconn = 2048，然后在终端中执行 sysctl -p。

tcp-backlog 511

#指定 redis 只接收来自于该 IP 地址的请求，如果不进行设置，那么将处理所有请求

bind 127.0.0.1

#配置 unix socket 来让 redis 支持监听本地连接。

unixsocket /var/run/redis/redis.sock

#配置 unix socket 使用文件的权限

unixsocketperm 700

此参数为设置客户端空闲超过 timeout，服务端会断开连接，为 0 则服务端不会主动断开连接，不能小于 0。

timeout 0

#tcp keepalive 参数。如果设置不为 0，就使用配置 tcp 的 SO_KEEPALIVE 值，使用 keepalive 有两个好处:检测挂掉的对端。降低中间设备出问题而导致网络看似连接却已经与对端端口的问题。在 Linux 内核中，设置了 keepalive，redis 会定时给对端发送 ack。检测到对端关闭需要两倍的设置值。

tcp-keepalive 0

#指定了服务端日志的级别。级别包括：debug（很多信息，方便开发、测试），verbose（许多有用的信息，但是没有 debug 级别信息多），notice（适当的日志级别，适合生产环境），warn（只有非常重要的信息）

loglevel notice

#指定了记录日志的文件。空字符串的话，日志会打印到标准输出设备。后台运行的 redis 标准输出是/dev/null。

logfile /var/log/redis/redis-server.log

#是否打开记录 syslog 功能

syslog-enabled no

#syslog 的标识符。

syslog-ident redis

#日志的来源、设备

syslog-facility local0

#数据库的数量，默认使用的数据库是 DB 0。可以通过”SELECT “命令选择一个 db

databases 16

################################ SNAPSHOTTING ################################

快照配置

注释掉“save”这一行配置项就可以让保存数据库功能失效

设置 sedis 进行数据库镜像的频率。

900 秒（15 分钟）内至少 1 个 key 值改变（则进行数据库保存--持久化）

300 秒（5 分钟）内至少 10 个 key 值改变（则进行数据库保存--持久化）

60 秒（1 分钟）内至少 10000 个 key 值改变（则进行数据库保存--持久化）

save 900 1

save 300 10

save 60 10000

#当 RDB 持久化出现错误后，是否依然进行继续进行工作，yes：不能进行工作，no：可以继续进行工作，可以通过 info 中的 rdb_last_bgsave_status 了解 RDB 持久化是否有错误

stop-writes-on-bgsave-error yes

#使用压缩 rdb 文件，rdb 文件压缩使用 LZF 压缩算法，yes：压缩，但是需要一些 cpu 的消耗。no：不压缩，需要更多的磁盘空间

rdbcompression yes

#是否校验 rdb 文件。从 rdb 格式的第五个版本开始，在 rdb 文件的末尾会带上 CRC64 的校验和。这跟有利于文件的容错性，但是在保存 rdb 文件的时候，会有大概 10%的性能损耗，所以如果你追求高性能，可以关闭该配置。

rdbchecksum yes

#rdb 文件的名称

dbfilename dump.rdb

#数据目录，数据库的写入会在这个目录。rdb、aof 文件也会写在这个目录

dir /var/lib/redis

################################## REPLICATION #################################

#复制选项，slave 复制对应的 master。

slaveof <masterip> <masterport>

#如果 master 设置了 requirepass，那么 slave 要连上 master，需要有 master 的密码才行。masterauth 就是用来配置 master 的密码，这样可以在连上 master 后进行认证。

masterauth <master-password>

#当从库同主机失去连接或者复制正在进行，从机库有两种运行方式：1）如果 slave-serve-stale-data 设置为 yes(默认设置)，从库会继续响应客户端的请求。2）如果 slave-serve-stale-data 设置为 no，除去 INFO 和 SLAVOF 命令之外的任何请求都会返回一个错误”SYNC with master in progress”。

slave-serve-stale-data yes

#作为从服务器，默认情况下是只读的（yes），可以修改成 NO，用于写（不建议）。

slave-read-only yes

#是否使用 socket 方式复制数据。目前 redis 复制提供两种方式，disk 和 socket。如果新的 slave 连上来或者重连的 slave 无法部分同步，就会执行全量同步，master 会生成 rdb 文件。有 2 种方式：disk 方式是 master 创建一个新的进程把 rdb 文件保存到磁盘，再把磁盘上的 rdb 文件传递给 slave。socket 是 master 创建一个新的进程，直接把 rdb 文件以 socket 的方式发给 slave。disk 方式的时候，当一个 rdb 保存的过程中，多个 slave 都能共享这个 rdb 文件。socket 的方式就的一个个 slave 顺序复制。在磁盘速度缓慢，网速快的情况下推荐用 socket 方式。

repl-diskless-sync no

#diskless 复制的延迟时间，防止设置为 0。一旦复制开始，节点不会再接收新 slave 的复制请求直到下一个 rdb 传输。所以最好等待一段时间，等更多的 slave 连上来。

repl-diskless-sync-delay 5

#slave 根据指定的时间间隔向服务器发送 ping 请求。时间间隔可以通过 repl_ping_slave_period 来设置，默认 10 秒。

repl-ping-slave-period 10

#复制连接超时时间。master 和 slave 都有超时时间的设置。master 检测到 slave 上次发送的时间超过 repl-timeout，即认为 slave 离线，清除该 slave 信息。slave 检测到上次和 master 交互的时间超过 repl-timeout，则认为 master 离线。需要注意的是 repl-timeout 需要设置一个比 repl-ping-slave-period 更大的值，不然会经常检测到超时。

repl-timeout 60

#是否禁止复制 tcp 链接的 tcp nodelay 参数，可传递 yes 或者 no。默认是 no，即使用 tcp nodelay。如果 master 设置了 yes 来禁止 tcp nodelay 设置，在把数据复制给 slave 的时候，会减少包的数量和更小的网络带宽。但是这也可能带来数据的延迟。默认我们推荐更小的延迟，但是在数据量传输很大的场景下，建议选择 yes。

repl-disable-tcp-nodelay no

#复制缓冲区大小，这是一个环形复制缓冲区，用来保存最新复制的命令。这样在 slave 离线的时候，不需要完全复制 master 的数据，如果可以执行部分同步，只需要把缓冲区的部分数据复制给 slave，就能恢复正常复制状态。缓冲区的大小越大，slave 离线的时间可以更长，复制缓冲区只有在有 slave 连接的时候才分配内存。没有 slave 的一段时间，内存会被释放出来，默认 1m。

repl-backlog-size 5mb

#master 没有 slave 一段时间会释放复制缓冲区的内存，repl-backlog-ttl 用来设置该时间长度。单位为秒。

repl-backlog-ttl 3600

#当 master 不可用，Sentinel 会根据 slave 的优先级选举一个 master。最低的优先级的 slave，当选 master。而配置成 0，永远不会被选举。

slave-priority 100

#redis 提供了可以让 master 停止写入的方式，如果配置了 min-slaves-to-write，健康的 slave 的个数小于 N，mater 就禁止写入。master 最少得有多少个健康的 slave 存活才能执行写命令。这个配置虽然不能保证 N 个 slave 都一定能接收到 master 的写操作，但是能避免没有足够健康的 slave 的时候，master 不能写入来避免数据丢失。设置为 0 是关闭该功能。

min-slaves-to-write 3

#延迟小于 min-slaves-max-lag 秒的 slave 才认为是健康的 slave。

min-slaves-max-lag 10

设置 1 或另一个设置为 0 禁用这个特性。

Setting one or the other to 0 disables the feature.

By default min-slaves-to-write is set to 0 (feature disabled) and

min-slaves-max-lag is set to 10.

################################## SECURITY ###################################

#requirepass 配置可以让用户使用 AUTH 命令来认证密码，才能使用其他命令。这让 redis 可以使用在不受信任的网络中。为了保持向后的兼容性，可以注释该命令，因为大部分用户也不需要认证。使用 requirepass 的时候需要注意，因为 redis 太快了，每秒可以认证 15w 次密码，简单的密码很容易被攻破，所以最好使用一个更复杂的密码。

requirepass foobared

#把危险的命令给修改成其他名称。比如 CONFIG 命令可以重命名为一个很难被猜到的命令，这样用户不能使用，而内部工具还能接着使用。

rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52

#设置成一个空的值，可以禁止一个命令

rename-command CONFIG ""

################################### LIMITS ####################################

设置能连上 redis 的最大客户端连接数量。默认是 10000 个客户端连接。由于 redis 不区分连接是客户端连接还是内部打开文件或者和 slave 连接等，所以 maxclients 最小建议设置到 32。如果超过了 maxclients，redis 会给新的连接发送’max number of clients reached’，并关闭连接。

maxclients 10000

#redis 配置的最大内存容量。当内存满了，需要配合 maxmemory-policy 策略进行处理。注意 slave 的输出缓冲区是不计算在 maxmemory 内的。所以为了防止主机内存使用完，建议设置的 maxmemory 需要更小一些。

maxmemory <bytes>

#内存容量超过 maxmemory 后的处理策略。

#volatile-lru：利用 LRU 算法移除设置过过期时间的 key。

#volatile-random：随机移除设置过过期时间的 key。

#volatile-ttl：移除即将过期的 key，根据最近过期时间来删除（辅以 TTL）

#allkeys-lru：利用 LRU 算法移除任何 key。

#allkeys-random：随机移除任何 key。

#noeviction：不移除任何 key，只是返回一个写错误。

#上面的这些驱逐策略，如果 redis 没有合适的 key 驱逐，对于写命令，还是会返回错误。redis 将不再接收写请求，只接收 get 请求。写命令包括：set setnx setex append incr decr rpush lpush rpushx lpushx linsert lset rpoplpush sadd sinter sinterstore sunion sunionstore sdiff sdiffstore zadd zincrby zunionstore zinterstore hset hsetnx hmset hincrby incrby decrby getset mset msetnx exec sort。

maxmemory-policy noeviction

#lru 检测的样本数。使用 lru 或者 ttl 淘汰算法，从需要淘汰的列表中随机选择 sample 个 key，选出闲置时间最长的 key 移除。

maxmemory-samples 5

############################## APPEND ONLY MODE ###############################

#默认 redis 使用的是 rdb 方式持久化，这种方式在许多应用中已经足够用了。但是 redis 如果中途宕机，会导致可能有几分钟的数据丢失，根据 save 来策略进行持久化，Append Only File 是另一种持久化方式，可以提供更好的持久化特性。Redis 会把每次写入的数据在接收后都写入 appendonly.aof 文件，每次启动时 Redis 都会先把这个文件的数据读入内存里，先忽略 RDB 文件。

appendonly no

#aof 文件名

appendfilename "appendonly.aof"

#aof 持久化策略的配置

#no 表示不执行 fsync，由操作系统保证数据同步到磁盘，速度最快。

#always 表示每次写入都执行 fsync，以保证数据同步到磁盘。

#everysec 表示每秒执行一次 fsync，可能会导致丢失这 1s 数据。

appendfsync everysec

在aof重写或者写入rdb文件的时候，会执行大量IO，此时对于everysec和always的aof模式来说，执行fsync会造成阻塞过长时间，no-appendfsync-on-rewrite字段设置为默认设置为no。如果对延迟要求很高的应用，这个字段可以设置为yes，否则还是设置为no，这样对持久化特性来说这是更安全的选择。设置为yes表示rewrite期间对新写操作不fsync,暂时存在内存中,等rewrite完成后再写入，默认为no，建议yes。Linux的默认fsync策略是30秒。可能丢失30秒数据。

no-appendfsync-on-rewrite no

#aof自动重写配置。当目前aof文件大小超过上一次重写的aof文件大小的百分之多少进行重写，即当aof文件增长到一定大小的时候Redis能够调用bgrewriteaof对日志文件进行重写。当前AOF文件大小是上次日志重写得到AOF文件大小的二倍（设置为100）时，自动启动新的日志重写过程。

auto-aof-rewrite-percentage 100

#设置允许重写的最小aof文件大小，避免了达到约定百分比但尺寸仍然很小的情况还要重写

auto-aof-rewrite-min-size 64mb

#aof文件可能在尾部是不完整的，当redis启动的时候，aof文件的数据被载入内存。重启可能发生在redis所在的主机操作系统宕机后，尤其在ext4文件系统没有加上data=ordered选项（redis宕机或者异常终止不会造成尾部不完整现象。）出现这种现象，可以选择让redis退出，或者导入尽可能多的数据。如果选择的是yes，当截断的aof文件被导入的时候，会自动发布一个log给客户端然后load。如果是no，用户必须手动redis-check-aof修复AOF文件才可以。

aof-load-truncated yes

################################ LUA SCRIPTING ###############################

如果达到最大时间限制（毫秒），redis会记个log，然后返回error。当一个脚本超过了最大时限。只有SCRIPT KILL和SHUTDOWN NOSAVE可以用。第一个可以杀没有调write命令的东西。要是已经调用了write，只能用第二个命令杀。

lua-time-limit 5000

################################ REDIS CLUSTER ###############################

#集群开关，默认是不开启集群模式。

cluster-enabled yes

#集群配置文件的名称，每个节点都有一个集群相关的配置文件，持久化保存集群的信息。这个文件并不需要手动配置，这个配置文件有Redis生成并更新，每个Redis集群节点需要一个单独的配置文件，请确保与实例运行的系统中配置文件名称不冲突

cluster-config-file nodes-6379.conf

#节点互连超时的阀值。集群节点超时毫秒数

cluster-node-timeout 15000

#在进行故障转移的时候，全部 slave 都会请求申请为 master，但是有些 slave 可能与 master 断开连接一段时间了，导致数据过于陈旧，这样的 slave 不应该被提升为 master。该参数就是用来判断 slave 节点与 master 断线的时间是否过长。判断方法是：

#比较 slave 断开连接的时间和(node-timeout * slave-validity-factor) + repl-ping-slave-period

#如果节点超时时间为三十秒，并且 slave-validity-factor 为 10,假设默认的 repl-ping-slave-period 是 10 秒，即如果超过 310 秒 slave 将不会尝试进行故障转移

cluster-slave-validity-factor 10

#master 的 slave 数量大于该值，slave 才能迁移到其他孤立 master 上，如这个参数若被设为 2，那么只有当一个主节点拥有 2 个可工作的从节点时，它的一个从节点会尝试迁移。

cluster-migration-barrier 1

#默认情况下，集群全部的 slot 有节点负责，集群状态才为 ok，才能提供服务。设置为 no，可以在 slot 没有全部分配的时候提供服务。不建议打开该配置，这样会造成分区的时候，小分区的 master 一直在接受写请求，而造成很长时间数据不一致。

cluster-require-full-coverage yes

################################## SLOW LOG ###################################

###slog log 是用来记录 redis 运行中执行比较慢的命令耗时。当命令的执行超过了指定时间，就记录在 slow log 中，slog log 保存在内存中，所以没有 IO 操作。

#执行时间比 slowlog-log-slower-than 大的请求记录到 slowlog 里面，单位是微秒，所以 1000000 就是 1 秒。注意，负数时间会禁用慢查询日志，而 0 则会强制记录所有命令。

slowlog-log-slower-than 10000

#慢查询日志长度。当一个新的命令被写进日志的时候，最老的那个记录会被删掉。这个长度没有限制。只要有足够的内存就行。你可以通过 SLOWLOG RESET 来释放内存。

slowlog-max-len 128

################################ LATENCY MONITOR ##############################

#延迟监控功能是用来监控 redis 中执行比较缓慢的一些操作，用 LATENCY 打印 redis 实例在跑命令时的耗时图表。只记录大于等于下边设置的值的操作。0 的话，就是关闭监视。默认延迟监控功能是关闭的，如果你需要打开，也可以通过 CONFIG SET 命令动态设置。

latency-monitor-threshold 0

############################# EVENT NOTIFICATION ##############################

#键空间通知使得客户端可以通过订阅频道或模式，来接收那些以某种方式改动了 Redis 数据集的事件。因为开启键空间通知功能需要消耗一些 CPU ，所以在默认配置下，该功能处于关闭状态。

#notify-keyspace-events 的参数可以是以下字符的任意组合，它指定了服务器该发送哪些类型的通知：

##K 键空间通知，所有通知以 __keyspace@__ 为前缀

##E 键事件通知，所有通知以 __keyevent@__ 为前缀

##g DEL 、 EXPIRE 、 RENAME 等类型无关的通用命令的通知

##S 字符串命令的通知

##l 列表命令的通知

##s 集合命令的通知

##h 哈希命令的通知

##z 有序集合命令的通知

##x 过期事件：每当有过期键被删除时发送

##e 驱逐(evict)事件：每当有键因为 maxmemory 政策而被删除时发送

##A 参数 g$lshzxe 的别名

#输入的参数中至少要有一个 K 或者 E，否则的话，不管其余的参数是什么，都不会有任何 通知被分发。详细使用可以参考 http://redis.io/topics/notifications

notify-keyspace-events ""

################################ ADVANCED CONFIG ###############################

#数据量小于等于 hash-max-ziplist-entries 的用 ziplist，大于 hash-max-ziplist-entries 用 hash

hash-max-ziplist-entries 512

#value 大小小于等于 hash-max-ziplist-value 的用 ziplist，大于 hash-max-ziplist-value 用 hash。

hash-max-ziplist-value 64

#数据量小于等于 list-max-ziplist-entries 用 ziplist，大于 list-max-ziplist-entries 用 list。

list-max-ziplist-entries 512

#value 大小小于等于 list-max-ziplist-value 的用 ziplist，大于 list-max-ziplist-value 用 list。

list-max-ziplist-value 64

#数据量小于等于 set-max-intset-entries 用 iniset，大于 set-max-intset-entries 用 set。

set-max-intset-entries 512

#数据量小于等于 zset-max-ziplist-entries 用 ziplist，大于 zset-max-ziplist-entries 用 zset。

zset-max-ziplist-entries 128

#value 大小小于等于 zset-max-ziplist-value 用 ziplist，大于 zset-max-ziplist-value 用 zset。

zset-max-ziplist-value 64

#value 大小小于等于 hll-sparse-max-bytes 使用稀疏数据结构（sparse），大于 hll-sparse-max-bytes 使用稠密的数据结构（dense）。一个比 16000 大的 value 是几乎没用的，建议的 value 大概为 3000。如果对 CPU 要求不高，对空间要求较高的，建议设置到 10000 左右。

hll-sparse-max-bytes 3000

#Redis 将在每 100 毫秒时使用 1 毫秒的 CPU 时间来对 redis 的 hash 表进行重新 hash，可以降低内存的使用。当你的使用场景中，有非常严格的实时性需要，不能够接受 Redis 时不时的对请求有 2 毫秒的延迟的话，把这项配置为 no。如果没有这么严格的实时性要求，可以设置为 yes，以便能够尽可能快的释放内存。

activerehashing yes

##对客户端输出缓冲进行限制可以强迫那些不从服务器读取数据的客户端断开连接，用来强制关闭传输缓慢的客户端。

#对于 normal client，第一个 0 表示取消 hard limit，第二个 0 和第三个 0 表示取消 soft limit，normal client 默认取消限制，因为如果没有寻问，他们是不会接收数据的。

client-output-buffer-limit normal 0 0 0

#对于 slave client 和 MONITER client，如果 client-output-buffer 一旦超过 256mb，又或者超过 64mb 持续 60 秒，那么服务器就会立即断开客户端连接。

client-output-buffer-limit slave 256mb 64mb 60

#对于 pubsub client，如果 client-output-buffer 一旦超过 32mb，又或者超过 8mb 持续 60 秒，那么服务器就会立即断开客户端连接。

client-output-buffer-limit pubsub 32mb 8mb 60

#redis 执行任务的频率为 1s 除以 hz。

hz 10

#在 aof 重写的时候，如果打开了 aof-rewrite-incremental-fsync 开关，系统会每 32MB 执行一次 fsync。这对于把文件写入磁盘是有帮助的，可以避免过大的延迟峰值。

aof-rewrite-incremental-fsync yes

++
++
++

2)集群当下配置(以 IP 地址 172.26.3.82 使用端口 7000 开启 redis 服务为例)：

daemonize yes

pidfile /var/run/redis_7000.pid

port 7000

tcp-backlog 511

bind 192.168.1.100 10.0.0.1

bind 172.26.3.82

timeout 0

tcp-keepalive 0

loglevel notice

```
logfile ""
databases 16
save 900 1
save 300 10
save 60 10000
stop-writes-on-bgsave-error yes
rdbcompression yes
rdbchecksum yes
dbfilename dump.rdb
dir /etc/redis/cluster_redis/7000
slave-serve-stale-data yes
slave-read-only yes
repl-diskless-sync no
repl-disable-tcp-nodelay no
slave-priority 100
appendonly no
appendfilename "appendonly.aof"
appendfsync everysec
no-appendfsync-on-rewrite no
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
aof-load-truncated yes
lua-time-limit 5000
cluster-enabled yes
cluster-config-file nodes-7000.conf
cluster-node-timeout 15000
slowlog-log-slower-than 10000
latency-monitor-threshold 0
hash-max-ziplist-entries 512
hash-max-ziplist-value 64
list-max-ziplist-entries 512
list-max-ziplist-value 64
set-max-intset-entries 512
zset-max-ziplist-entries 128
zset-max-ziplist-value 64
hll-sparse-max-bytes 3000
```

activerehashing yes

client-output-buffer-limit normal 0 0 0

client-output-buffer-limit slave 256mb 64mb 60

client-output-buffer-limit pubsub 32mb 8mb 60

hz 10

aof-rewrite-incremental-fsync yes

举例规划集群：

cd /etc/redis

mkdir 7000 7001 7002 7003 7004

#计划在启动5个redis节点，每个redis节点对应一个redis配置文件。按照需要修改源码目录下redis.conf文件，

#将对应配置文件分别存放到对应目录下（此处我们计划使用7000、7001、7002、7003、7004端口）

启动redis集群：

启动集群相关节点（必须是空节点），指定配置文件

redis-server /etc/redis/cluster/7000/redis.conf

redis-server /etc/redis/cluster/7001/redis.conf

redis-server /etc/redis/cluster/7002/redis.conf

redis-server /etc/redis/cluster/7003/redis.conf

redis-server /etc/redis/cluster/7004/redis.conf

使用自带的ruby工具(redis-trib.rb)构建集群

#redis-trib.rb的create子命令构建

#--replicas 则指定了为Redis Cluster中的每个Master节点配备几个Slave节点

redis-trib.rb create --replicas 1 172.26.3.82:7000 172.26.3.82:7001 172.26.3.82:7002 172.26.3.82:7003 172.26.3.82:7004

关闭redis集群：

使用pkill redis-server

这种关闭集群的方式有待商榷。

redis 维护说明

Redis 集群添加、删除节点

添加节点

1：首先把需要添加的节点启动

cd /etc/redis/cluster_redis/

mkdir 7006

cp /修改好的配置文件/redis.conf /etc/redis/cluster_redis/7006/

cd /etc/redis/cluster_redis/7006/

redis-server redis.conf

2：执行以下命令，将这个新节点添加到集群中；

cd /usr/local/redis3.0/src/

redis-trib.rb add-node 172.26.3.82:7006 172.26.3.82:7000

3：执行命令 redis-cli -c -h 172.26.3.82 -p 7000 cluster nodes 查看刚才新增的节点；

4：增加了新的节点之后，这个新的节点可以成为主节点或者是从节点

4.1 把这个节点变成主节点，使用 redis-trib 程序，将集群中的某些哈希槽移动到新节点里面， 这个新节点就成为真正的主节点了。

执行下面的命令对集群中的哈希槽进行移动

redis-trib.rb reshard 172.26.3.82:7000

系统会提示我们要移动多少哈希槽，这里移动 1000 个(提示输入数字) ；

然后还需要指定把这些哈希槽转移到哪个节点上，输入我们刚才新增的节点的 ID：（172.26.3.82:7006 的节点 ID）；

然后需要我们指定转移哪几个几点的哈希槽 输入 all 表示从所有的主节点中随机转移，凑够 1000 个哈希槽；

然后再输入 yes，redis 集群就开始分配哈希槽了。

至此，一个新的主节点就添加完成了，执行命令查看现在的集群中节点的状态

redis-cli -c -h 172.26.3.82 -p 7000 cluster nodes

4.2：把这个节点变成从节点

前面我们已经把这个新节点添加到集群中了，现在我们要让新节点成为 172.26.3.82:7001 的从节点，只需要执行下面的命令就可以了，命令后面的节点 ID 就是 127.0.0.1:7001 的节点 ID

redis-cli -c -h 172.26.3.82 -p 7006 cluster replicate 节点（172.26.3.82:7001 的节点 ID）

使用下面命令来确认一下 172.26.3.82:7006 是否已经成为 172.26.3.82:7001 的从节点

```
redis-cli -h 172.26.3.82 -p 7000 cluster nodes | grep slave | grep 节点（172.26.3.82:7001 的节点 ID）
```

删除节点

1：如果删除的节点是主节点，这里我们删除 172.26.3.82:7006 节点，这个节点有 1000 个哈希槽

首先要把节点中的哈希槽转移到其他节点中，执行下面的命令

```
redis-trib.rb reshard 172.26.3.82:7000
```

系统会提示我们要移动多少哈希槽，这里移动 1000 个，因为 172.26.3.82:7006 节点有 1000 个哈希槽

然后系统提示我们输入要接收这些哈希槽的节点的 ID，这里使用 172.26.3.82:7001 的节点 ID

然后要我们选择从那些节点中转出哈希槽，这里一定要输入 172.26.3.82:7006 这个节点的 ID，最后输入 done 表示输入完毕

最后一步，使用下面的命令把这个节点删除

```
redis-trib.rb del-node 172.26.3.82:7006 172.26.3.82:7006
```

2：如果节点是从节点的，直接使用下面的命令删除即可。

```
redis-trib.rb del-node 172.26.3.82:7006 172.26.3.82:7006
```

数据备份和恢复：

备份：

使用 save 或者 bgsave（后台运行备份命令）

该命令将在 Redis 安装目录中创建 dump.rdb 文件。

恢复：

将备份文件（dump.rdb）移动到 redis 安装目录并启动服务即可。

启动常见问题：

1、集群启动前，记得将配置文件 dir 项所配置的目录下的文件备份或者删除！！！！！！

以 172.26.3.82：7000 为例：

就是将目录：/etc/redis/cluster_redis/7000/目录下的 redis.conf 以外的文件全部移除或备份到其他路径下！！！！！！！！否则集群启动不会成功。

2、集群启动时，如果集群命令一直停留在 join……状态，那么需要检查，是否是由于配置文件 bind 项目，配置绑定了 127.0.0.1，如果集群启动不应该使用该 IP 进行绑定！！！！！！！

需求测试说明

启动集群后，开始数据入库，在数据入库过程中，将配置连接的特定 IP 端口的服务 kill 掉，查看入库是否能够继续进行，如果能够正常进行，则集群功能测试成功。

ACL 框架 C++集群 redis 客户端 API 介绍

源码

```
 * @brief-ACL 框架 redis 集群客户端访问 API 调用举例
 * @author wangchenxi
 * @version 0.1.00
 * @date 2017-01-10
 */

#include "acl_cpp/lib_acl.hpp"
#include "lib_acl.h"

static acl::string __keypre("wangchenxi_test_key");

static bool test_set(acl::redis& cmd, int i)
{
|       cmd.clear();

|       acl::string key;
|       key.format("%s_%d", __keypre.c_str(), i);

|       acl::string value;
|       value.format("value_%s", key.c_str());

|       bool ret = cmd.set(key.c_str(), value.c_str());
|       if (i < 10)
|       |       printf("set key: %s, value: %s %s\r\n", key.c_str(),
|       |       |       value.c_str(), ret ? "ok" : "error");
|       return ret;
}



int main(int argc, char* argv[])
{
    int ch, n = 1, connect_timeout = 10, rw_timeout = 10;
```

```
    int  max_threads = 10, nsleep = 500, nretry = 10;
    acl::string addrs("172.26.3.81:7000");
    acl::acl_cpp_init();
    acl::redis_client_cluster cluster;
    cluster.set(addrs.c_str(), 10000);
    // 当某个连接池结点出问题，设置探测该连接结点是否恢复的时间间隔(秒)，当该值
    // 为 0 时，则不检测
    // 是否需要将所有哈希槽的对应关系提前设置好，这样可以去掉运行时动态添加
    // 哈希槽的过程，从而可以提高运行时的效率
    bool ret;
    acl::redis cmd;
    cmd.set_cluster(&cluster, 10000);
    ret = test_set(cmd, 100);
}
```

编译

```
g++ *.cpp -L../.. -l_acl -I../../lib_acl/ -I. -I../../lib_acl_cpp/ -I../../lib_protocol/ -g
-Wall  -Wshadow -Wno-long-long -Wpointer-arith -D_REENTRANT -D_POSIX_PTHREAD_SEMANTICS
-D_USE_FAST_MACRO -DLINUX2 -rdynamic -lpthread
```