redis 记录

1.1.1 基本概念

1.1.1.1 关系型数据库的性能

关系型数据库，每秒只能插入、更新或者上拿出 200-2000 条记录（数据行）

1.1.1.2 centos7 上，安装访问 redis 的 python 客户端

```
# yum install epel-release
# yum install -y python-pip
# pip install redis
```

1.1.1.3 centos7 使用 python 访问 redis

```
# python
>>> import redis
>>> conn = redis.Redis(host='127.0.0.1',port=6379)
>>> conn.set('name1','test')
True
>>> conn.get('name')
'junxi'
```

1.1.2 源码安装单节点 redis

```
# yum install -y gcc make
# curl -O http://download.redis.io/releases/redis-4.0.6.tar.gz
# tar -zxvf redis-4.0.6.tar.gz
# cd redis-4.0.6
# make install
```

1.1.3 yum 安装单节点 redis

1.1.3.1 安装

安装 epel 源

```
# yum install -y epel-release
```

安装 redis

```
# yum install -y redis
```

启动 redis 服务

```
# systemctl start redis && systemctl enable redis
# systemctl status redis
```

检查 redis 的状态

```
# redis-cli ping
PONG
# redis-benchmark -q -n 1000 -c 10 -P 5
PING_INLINE: 499999.97 requests per second
PING_BULK: 999999.94 requests per second
SET: 499999.97 requests per second
GET: 999999.94 requests per second
INCR: 333333.34 requests per second
LPUSH: 499999.97 requests per second
RPUSH: 499999.97 requests per second
LPOP: 999999.94 requests per second
RPOP: 999999.94 requests per second
SADD: 499999.97 requests per second
HSET: 499999.97 requests per second
SPOP: 999999.94 requests per second
LPUSH (needed to benchmark LRANGE): 499999.97 requests per second
LRANGE_100 (first 100 elements): 83333.34 requests per second
LRANGE_300 (first 300 elements): 23255.81 requests per second
LRANGE_500 (first 450 elements): 13333.33 requests per second
LRANGE_600 (first 600 elements): 9090.91 requests per second
MSET (10 keys): 249999.98 requests per second
```

1.1.3.2 配置 redis

```
# sed -i 's/tcp-keepalive.*/tcp-keepalive 60/' /etc/redis.conf
# sed -i 's/bind 127.0.0.1$/#bind 127.0.0.1/' /etc/redis.conf
# sed -i 's/# requirepass.*/requirepass password/' /etc/redis.conf
# sed -i 's/# maxmemory-policy noeviction/maxmemory-policy noeviction/' /etc/redis.conf
# sed -i 's/appendonly no/appendonly yes/' /etc/redis.conf
```

重启服务

```
# systemctl restart redis
```

1.1.3.3 验证

```
# redis-cli -h localhost -p 6379
localhost:30001>auth password
localhost:30001> set hello world
OK
localhost:30001> get hello
"world"
localhost:30001> set foo bar
-> Redirected to slot [12182] located at 127.0.0.1:30003
```

```
OK
127.0.0.1:30003> get foo
"bar"
```

1.1.4 centos7 部署主从 redis

1.1.4.1 在所有的节点，安装 redis

安装 epel 源

```
# yum install -y epel-release
```

安装 redis

```
# yum install -y redis
```

启动 redis 服务

```
# systemctl start redis && systemctl enable redis
# systemctl status redis
```

检查 redis 的状态

```
# redis-cli ping
PONG
# redis-benchmark -q -n 1000 -c 10 -P 5
PING_INLINE: 499999.97 requests per second
PING_BULK: 999999.94 requests per second
SET: 499999.97 requests per second
GET: 999999.94 requests per second
INCR: 333333.34 requests per second
LPUSH: 499999.97 requests per second
RPUSH: 499999.97 requests per second
LPOP: 999999.94 requests per second
RPOP: 999999.94 requests per second
SADD: 499999.97 requests per second
HSET: 499999.97 requests per second
SPOP: 999999.94 requests per second
LPUSH (needed to benchmark LRANGE): 499999.97 requests per second
LRANGE_100 (first 100 elements): 83333.34 requests per second
LRANGE_300 (first 300 elements): 23255.81 requests per second
LRANGE_500 (first 450 elements): 13333.33 requests per second
LRANGE_600 (first 600 elements): 9090.91 requests per second
MSET (10 keys): 249999.98 requests per second
```

1.1.4.2 在主节点，配置 redis

```
# sed -i 's/tcp-keepalive.*/tcp-keepalive 60/' /etc/redis.conf
# sed -i 's/bind 127.0.0.1$/#bind 127.0.0.1/' /etc/redis.conf
# sed -i 's/# requirepass.*/requirepass password/' /etc/redis.conf
# sed -i 's/# maxmemory-policy noeviction/maxmemory-policy noeviction/' /etc/redis.conf
# sed -i 's/appendonly no/appendonly yes/' /etc/redis.conf
```

重启服务

```
# systemctl restart redis
```

1.1.4.3 在所有的 slave 节点，配置 redis

```
# sed -i 's/bind 127.0.0.1$/#bind 127.0.0.1/' /etc/redis.conf
# sed -i 's/# requirepass.*/requirepass password/' /etc/redis.conf
# sed -i 's/# slaveof.*/slaveof 192.168.6.44 6379/' /etc/redis.conf
# sed -i 's/# masterauth.*/masterauth password/' /etc/redis.conf
```

注意：这里的 192.168.6.44 是 redis master 的主节点，password 是 redis 主节点的密码

重启服务

```
# systemctl restart redis
```

1.1.4.4 验证主从复制

1.1.4.4.1 在主节点上验证

这里的 192.168.6.44 是 redis master 的 ip

```
# redis-cli -h 192.168.6.44 -p 6379
192.168.6.44:6379> auth password
OK
192.168.6.44:6379> info replication
# Replication
role:master
connected_slaves:2
slave0:ip=192.168.6.45,port=6379,state=online,offset=43,lag=0
slave1:ip=192.168.6.46,port=6379,state=online,offset=43,lag=0
master_repl_offset:43
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:2
repl_backlog_histlen:42
```

1.1.4.4.2 在 slave 节点上，验证

这里的 192.168.6.45 是 redis slave 的 ip

```
# redis-cli -h 192.168.6.45 -p 6379
192.168.6.45:6379> auth password
OK
192.168.6.45:6379> info replication
# Replication
role:slave
```

```
master_host:192.168.6.44
master_port:6379
master_link_status:up
master_last_io_seconds_ago:10
master_sync_in_progress:0
slave_repl_offset:183
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

1.1.4.5 将 master 切换到 slave 节点

1.1.4.5.1 在 redis 的第一个 slave 上，将 slave 改为 master

```
# redis-cli -h 127.0.0.1 -p 6379
127.0.0.1:6379> auth password
OK
127.0.0.1:6379> slaveof no one
OK
127.0.0.1:6379> info replication
# Replication
role:master
connected_slaves:0
master_repl_offset:435
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

此时，这个 redis slave 变为 master 节点

1.1.4.5.2 在其他的 redis slave 节点上，将 master 地址改为新的 slave 地址

```
# redis-cli -h 127.0.0.1 -p 6379
127.0.0.1:6379> auth password
OK
127.0.0.1:6379> info replication
# Replication
role:slave
master_host:192.168.6.44
master_port:6379
```

```
master_link_status:up
master_last_io_seconds_ago:5
master_sync_in_progress:0
slave_repl_offset:603
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
127.0.0.1:6379> slaveof 192.168.6.45 6379
OK
127.0.0.1:6379> info replication
# Replication
role:slave
master_host:192.168.6.45
master_port:6379
master_link_status:up
master_last_io_seconds_ago:5
master_sync_in_progress:0
slave_repl_offset:436
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

1.1.4.5.3 在原来的 master 上，将原来的 master 改为 slave

```
# redis-cli -h 127.0.0.1 -p 6379
127.0.0.1:6379> auth password
OK
192.168.6.44:6379> slaveof 192.168.6.45 6379
OK
192.168.6.44:6379> info replication
# Replication
role:slave
master_host:192.168.6.45
master_port:6379
master_link_status:down
master_last_io_seconds_ago:-1
master_sync_in_progress:0
```

```
slave_repl_offset:1
master_link_down_since_seconds:1513682244
slave_priority:100
slave_read_only:1
connected_slaves:0
master_repl_offset:0
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:2
repl_backlog_histlen:826
```

1.1.5 centos7 在一台机器上，源码安装 redis cluster 测试系统

1.1.5.1 安装 redis

```
# yum install -y gcc make
# curl -O http://download.redis.io/releases/redis-4.0.6.tar.gz
# tar -zxvf redis-4.0.6.tar.gz
# cd redis-4.0.6
# make install
```

1.1.5.2 安装 ruby 软件包—目前在线安装

```
# gpg2 --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3
# curl -L get.rvm.io | bash -s stable
# source /usr/local/rvm/scripts/rvm
```

查看 rvm 中的 ruby 版本，需要安装 2.2.2 以上的版本

```
# rvm list known
# rvm install 2.3.4
```

查看 ruby 版本

```
# ruby --version
# gem install redis
```

1.1.5.3 安装基本的 cluster

```
# cd /root/redis-4.0.6/utils/create-cluster
./create-cluster start
./create-cluster create
```

1.1.5.4 验证

```
# redis-cli -c -h localhost -p 30001
localhost:30001> set hello world
OK
```

```
localhost:30001> get hello
"world"
localhost:30001> set foo bar
-> Redirected to slot [12182] located at 127.0.0.1:30003
OK
127.0.0.1:30003> get foo
"bar"
```

1.1.6 centos7 上部署生产环境的 redis cluster

1.1.6.1 创建 redis cluster

1.1.6.1.1 yum 安装 redis

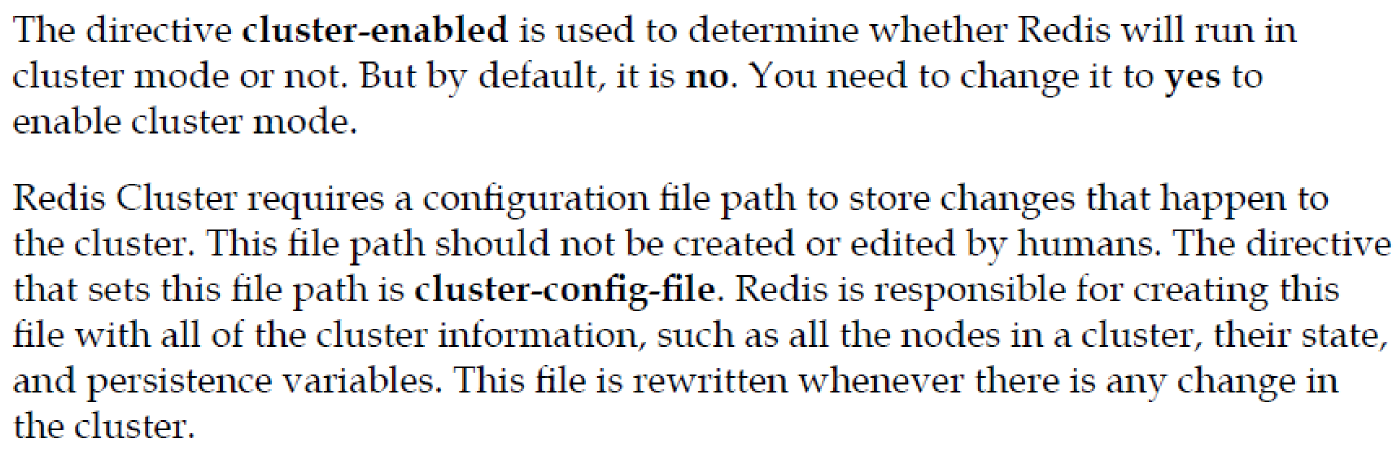
```
# yum install -y epel-release
# yum install -y redis
```

1.1.6.1.2 修改/etc/redis.conf 配置文件

```
# sed -i 's/^bind 127.0.0.1/#bind 127.0.0.1/' /etc/redis.conf
# sed -i 's/# cluster-enabled yes/cluster-enabled yes/' /etc/redis.conf
# sed -i 's/# cluster-config-file nodes-6379.conf/cluster-config-file nodes-6379.conf/' /etc/redis.conf
# sed -i 's/# cluster-node-timeout.*/cluster-node-timeout 5000/' /etc/redis.conf
# sed -i 's/# cluster-slave-validity-factor.*/cluster-slave-validity-factor 10/' /etc/redis.conf
# sed -i 's/# cluster-migration-barrier.*/cluster-migration-barrier 1/' /etc/redis.conf
# sed -i 's/# cluster-require-full-coverage yes/cluster-require-full-coverage yes/' /etc/redis.conf
# sed -i 's/^daemonize no/daemonize yes/' /etc/redis.conf
# sed -i 's/# requirepass.*/requirepass 123456/' /etc/redis.conf
# sed -i 's/# masterauth.*/masterauth 123456/' /etc/redis.conf
# sed -i 's/appendonly no/appendonly yes/' /etc/redis.conf
```

这里的 requirepass 和 masterauth 密码要保持一致，主从复制时使用

参数解释：



The directive **cluster-enabled** is used to determine whether Redis will run in cluster mode or not. But by default, it is **no**. You need to change it to **yes** to enable cluster mode.

Redis Cluster requires a configuration file path to store changes that happen to the cluster. This file path should not be created or edited by humans. The directive that sets this file path is **cluster-config-file**. Redis is responsible for creating this file with all of the cluster information, such as all the nodes in a cluster, their state, and persistence variables. This file is rewritten whenever there is any change in the cluster.

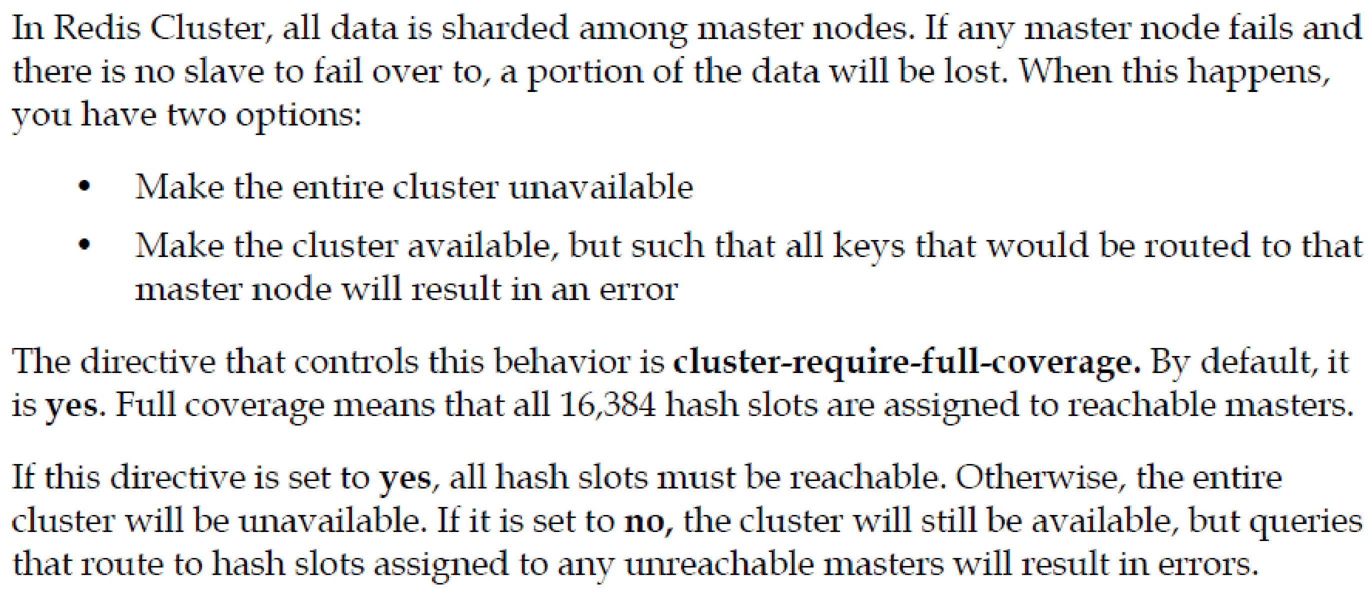
The maximum amount of time for which a node can be unavailable without being considered as failing is specified by the directive **cluster-node-timeout** (this value is in milliseconds). If a node is not reachable for the specified amount of time by the majority of master nodes, it will be considered as failing. If that node is a master, a failover to one of its slaves will occur. If it is a slave, it will stop accepting queries.

Sometimes, network failures happen, and when they happen, it is always a good idea to minimize problems. If network issues are happening and nodes cannot communicate well, it is possible that the majority of nodes think that a given master is down and so a failover procedure should start. If the network only had a hiccup, the failover procedure might have been unnecessary. There is a configuration directive that helps minimize these kinds of problems. The directive is **cluster-slave-validity-factor**, and it expects a factor. By default, the factor is **10**. If there is a network issue and a master node cannot communicate well with other nodes for a certain amount of time (**cluster-node-timeout** multiplied by **cluster-slave-validity-factor**), no slaves will be promoted to replace that master. When the connection issues go away and the master node is able to communicate well with others again, if it becomes unreachable a failover will happen.

When the factor is set to zero, no failovers will be prevented. If any network connectivity issues occur and the factor is zero, a slave will always perform the failover.

It is possible to specify the minimum number of slaves that must be connected to a master through the directive **cluster-migration-barrier**, which has a default value of *1*. This directive is useful if you need to ensure a minimum number of slaves per master. Otherwise, masters without slaves will borrow spare slaves from other masters.

Take the following example: master A has A1 and A2 as slaves, master B has B1 as a slave, master C has C1 as a slave, and the directive **cluster-migration-barrier** is set to *2*. If master C fails and C1 gets promoted to master, master A will keep all of its slaves (because the minimum is *2*), and master C1 will have zero slaves. If you never want to have masters borrowing slaves from other masters, set this configuration to a high number. The value of *0* is discouraged, since it should be used for debugging only.


In Redis Cluster, all data is sharded among master nodes. If any master node fails and there is no slave to fail over to, a portion of the data will be lost. When this happens, you have two options:

- Make the entire cluster unavailable
- Make the cluster available, but such that all keys that would be routed to that master node will result in an error

The directive that controls this behavior is **cluster-require-full-coverage.** By default, it is **yes**. Full coverage means that all 16,384 hash slots are assigned to reachable masters.

If this directive is set to **yes**, all hash slots must be reachable. Otherwise, the entire cluster will be unavailable. If it is set to **no,** the cluster will still be available, but queries that route to hash slots assigned to any unreachable masters will result in errors.

1.1.6.1.3 以集群模式启动 redis 实例

```
# systemctl start redis
```
或者
```
# redis-server /etc/redis.conf
# ps -ef |grep -v grep |grep redis
root         1166         1    0 19:21 ?            00:00:00 redis-server *:6379 [cluster]
```
注意：如果先以 redis-server /etc/redis.conf 启动了 redis，需要确保/var/log/redis 和/var/lib/redis 的权限为 redis;因为 systemctl 的方式，是以 redis 账户启动的，具体查看/lib/systemd/system/redis.service 中的配置

1.1.6.1.4 查看每个 redis 节点的 cluster 信息

这里以第一个节点为例：
```
# redis-cli -c -h 192.168.6.44 -a 123456 cluster info
cluster_state:fail
cluster_slots_assigned:0
cluster_slots_ok:0
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:1
cluster_size:0
cluster_current_epoch:0
cluster_my_epoch:0
cluster_stats_messages_sent:0
cluster_stats_messages_received:0
```
此时集群的状态是 fail，节点 1 知道的节点只有 1 个（也就是自己）

1.1.6.1.5 在第一个节点上，为每个节点分配 slot

redis 共有 16384 个 slot；这里第一个节点是 0-5460，第二个节点是 5461-10922，第三个节点是 10923-16383；

```
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster addslots {0..5460}
OK
# redis-cli -c -h 192.168.6.45 -p 6379 -a 123456 cluster addslots {5461..10922}
OK
# redis-cli -c -h 192.168.6.46 -p 6379 -a 123456 cluster addslots {10923..16383}
OK
```

1.1.6.1.6 在第一个节点上，为每个节点分配 epoch

这个只在第一次创建 redis cluster 用到

```
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster set-config-epoch 1
OK
# redis-cli -c -h 192.168.6.45 -p 6379 -a 123456 cluster set-config-epoch 2
OK
# redis-cli -c -h 192.168.6.46 -p 6379 -a 123456 cluster set-config-epoch 3
OK
```

1.1.6.1.7 在第一个节点上，执行 cluster meet 让第一个 node 与其他 node 互联

其他节点之间的互联，通过 cluster 的通讯来实现

```
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster meet 192.168.6.45 6379
OK
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster meet 192.168.6.46 6379
OK
```

1.1.6.1.8 在第一个节点上，查看 cluster 的状态

```
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:3
cluster_size:3
cluster_current_epoch:3
cluster_my_epoch:1
cluster_stats_messages_sent:350
cluster_stats_messages_received:350
```

此时，集群的状态为 ok，已知节点为 3；此时的 redis 几点运行正常

1.1.6.1.9 在第一个节点上，查看集群节点的状态

```
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes
536cfb2e8b95e19ccee78abaf1ecc3b9e55b74a0 192.168.6.46:6379 master - 0 1513792722316 3 connected 10923-16383
cc1fd3db2218df64267e0cb7e5979513c958bc6b 192.168.6.45:6379 master - 0 1513792723319 2 connected 5461-10922
b5874559e7f308c0eaf62ed439e90e59eec1b720 192.168.6.44:6379 myself,master - 0 0 1 connected 0-5460
```

1.1.6.2 添加 slave/replicas 节点→主节点挂了，以致整个 redis 不可用

1.1.6.2.1 yum 安装 redis

```
# yum install -y epel-release
# yum install -y redis
```

1.1.6.2.2 修改/etc/redis.conf 配置文件

```
# sed -i 's/^bind 127.0.0.1/#bind 127.0.0.1/' /etc/redis.conf
# sed -i 's/# cluster-enabled yes/cluster-enabled yes/' /etc/redis.conf
# sed -i 's/# cluster-config-file nodes-6379.conf/cluster-config-file nodes-6379.conf/' /etc/redis.conf
# sed -i 's/# cluster-node-timeout.*/cluster-node-timeout 5000/' /etc/redis.conf
# sed -i 's/# cluster-slave-validity-factor.*/cluster-slave-validity-factor 10/' /etc/redis.conf
# sed -i 's/# cluster-migration-barrier.*/cluster-migration-barrier 1/' /etc/redis.conf
# sed -i 's/# cluster-require-full-coverage yes/cluster-require-full-coverage yes/' /etc/redis.conf
# sed -i 's/^daemonize no/daemonize yes/' /etc/redis.conf
# sed -i 's/# requirepass.*/requirepass 123456/' /etc/redis.conf
# sed -i 's/# masterauth.*/masterauth 123456/' /etc/redis.conf
# sed -i 's/appendonly no/appendonly yes/' /etc/redis.conf
```

这里的 requirepass 和 masterauth 密码要保持一致，主从复制时使用

1.1.6.2.3 以集群模式启动 redis 实例

```
# systemctl start redis
```

或者

```
# redis-server /etc/redis.conf
# ps -ef |grep -v grep |grep redis
root      1166     1  0 19:21 ?        00:00:00 redis-server *:6379 [cluster]
```

注意：如果先以 redis-server /etc/redis.conf 启动了 redis，需要确保/var/log/redis 和/var/lib/redis 的权限为 redis;因为 systemctl 的方式，是以 redis 账户启动的，具体查看/lib/systemd/system/redis.service 中的配置

1.1.6.2.4 将此节点加入现有 redis 集群

```
# redis-cli -c -h 192.168.6.47 -p 6379 -a 123456 cluster meet 192.168.6.44 6379
```

```
OK
```

解释：192.168.6.47 是新增节点的 ip；192.168.6.44 是现有 redis 集群的一个 ip 地址，6379 是两个 redis 运行的端口，123456 是 192.168.6.47 的 redis 密码

1.1.6.2.5 将 6.47 节点作为 6.44 节点的 replicas

1. 获取 6.44 的 node id

```
# nodeid=$(redis-cli -h 192.168.6.44 -p 6379 -a 123456 cluster nodes |grep 192.168.6.44 |awk '{print $1}')
```

2. 设置 6.47 作为 6.44 的 replicas 节点

```
# redis-cli -c -h 192.168.6.47 -p 6379 -a 123456 cluster replicate $nodeid
OK
```

3. 查看 redis 的 cluster 节点信息

```
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes
c742745c3b2f006520b57fc660119aed3fd31aab 192.168.6.46:6379 master - 0 1513768010530 3 connected 10923-16383
725b62c8213ab71b0fb4836899268b09ecb2b6f3 192.168.6.44:6379 myself,master - 0 0 1 connected 0-5460
f7b08e536860b90e00cbf0380cecb057f078dbb7 192.168.6.47:6379 slave 725b62c8213ab71b0fb4836899268b09ecb2b6f3 0 1513768010027 1 connected
6be83760f07dda5ec326cfa280308a95c3b2ba02 192.168.6.45:6379 master - 0 1513768011032 2 connected 5461-10922
```

解释：此时 192.168.6.47 是 725b62c8213ab71b0fb4836899268b09ecb2b6f3 的 replicas 节点了

4. 查看 redis 主从复制状态

在主 redis（6.44）上查看复制信息

```
# redis-cli -c -h localhost -a 123456 info replication
# Replication
role:master
connected_slaves:1
slave0:ip=192.168.6.47,port=6379,state=online,offset=57,lag=1
master_repl_offset:57
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:2
repl_backlog_histlen:56
```

此时说明，6.47 已经做为 6.44 的 slave，如有问题，查看/var/log/redis/redis.log 日志

1.1.6.3 新增 redis node

1.1.6.3.1 yum 安装 redis

```
# yum install -y epel-release
# yum install -y redis
```

1.1.6.3.2 修改/etc/redis.conf 配置文件

```
# sed -i 's/^bind 127.0.0.1/#bind 127.0.0.1/' /etc/redis.conf
# sed -i 's/# cluster-enabled yes/cluster-enabled yes/' /etc/redis.conf
# sed -i 's/# cluster-config-file nodes-6379.conf/cluster-config-file nodes-6379.conf/' /etc/redis.conf
# sed -i 's/# cluster-node-timeout.*/cluster-node-timeout 5000/' /etc/redis.conf
# sed -i 's/# cluster-slave-validity-factor.*/cluster-slave-validity-factor 10/' /etc/redis.conf
# sed -i 's/# cluster-migration-barrier.*/cluster-migration-barrier 1/' /etc/redis.conf
# sed -i 's/# cluster-require-full-coverage yes/cluster-require-full-coverage yes/' /etc/redis.conf
# sed -i 's/^daemonize no/daemonize yes/' /etc/redis.conf
# sed -i 's/# requirepass.*/requirepass 123456/' /etc/redis.conf
# sed -i 's/# masterauth.*/masterauth 123456/' /etc/redis.conf
# sed -i 's/appendonly no/appendonly yes/' /etc/redis.conf
```

这里的 requirepass 和 masterauth 密码要保持一致，主从复制时使用

1.1.6.3.3 以集群模式启动 redis 实例

```
# systemctl start redis
```

或者

```
# redis-server /etc/redis.conf
# ps -ef |grep -v grep |grep redis
root      1166     1  0 19:21 ?        00:00:00 redis-server *:6379 [cluster]
```

注意：如果先以 redis-server /etc/redis.conf 启动了 redis，需要确保/var/log/redis 和/var/lib/redis 的权限为 redis;因为 systemctl 的方式，是以 redis 账户启动的，具体查看/lib/systemd/system/redis.service 中的配置

1.1.6.3.4 将此节点加入现有 redis 集群

```
# redis-cli -c -h 192.168.6.48 -p 6379 -a 123456 cluster meet 192.168.6.44 6379
OK
```

解释：192.168.6.48 是新增节点的 ip；192.168.6.44 是现有 redis 集群的一个 ip 地址，6379 是两个 redis 运行的端口，123456 是 192.168.6.48 的 redis 密码

此时 192.168.6.48 无 hash slot

```
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes
725b62c8213ab71b0fb4836899268b09ecb2b6f3 192.168.6.44:6379 myself,master - 0 0 1 connected 0-5460
c742745c3b2f006520b57fc660119aed3fd31aab 192.168.6.46:6379 master - 0 1513769014362 3 connected 10923-16383
6be83760f07dda5ec326cfa280308a95c3b2ba02 192.168.6.45:6379 master - 0 1513769015366 2 connected 5461-10922
f7b08e536860b90e00cbf0380cecb057f078dbb7 192.168.6.47:6379 slave 725b62c8213ab71b0fb4836899268b09ecb2b6f3 0 1513769015366 1 connected
8edce63d08111a66428b5fc02505e6d1e4bad445 192.168.6.48:6379 master - 0 1513769013354 0 connected
```

测试，设置的 key 被转到其他节点

```
# redis-cli -c -h localhost -a 123456
localhost:6379> set hello1 world
-> Redirected to slot [11613] located at 192.168.6.46:6379
OK
```

解释：hello1 的 key 位于 slot:11613,后期将其迁移到新的节点

1.1.6.3.5 迁移 hash slot

计划将 hash slot 为 11613 迁移到新节点

查看 11613 的 slot 位于哪个节点，获取其 nodeid

```
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes
725b62c8213ab71b0fb4836899268b09ecb2b6f3 192.168.6.44:6379 myself,master - 0 0 1 connected 0-5460
c742745c3b2f006520b57fc660119aed3fd31aab 192.168.6.46:6379 master - 0 1513769873520 3 connected 10923-16383
6be83760f07dda5ec326cfa280308a95c3b2ba02 192.168.6.45:6379 master - 0 1513769872515 2 connected 5461-10922
f7b08e536860b90e00cbf0380cecb057f078dbb7 192.168.6.47:6379 slave 725b62c8213ab71b0fb4836899268b09ecb2b6f3 0 1513769874024 1 connected
8edce63d08111a66428b5fc02505e6d1e4bad445 192.168.6.48:6379 master - 0 1513769874527 0 connected
```

获取 11613 的 hash slot 的源 nodeid

```
# sourcenodeid=$(redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes |grep 192.168.6.46 |awk '{print $1}')
```

在新增的节点上，执行 hash slot 的 importing 操作

```
# redis-cli -c -h 192.168.6.48 -p 6379 -a 123456 cluster setslot 11613 importing $sourcenodeid
OK
```

获取新建节点的 nodeid

```
# destinationnodeid=$(redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes |grep 192.168.6.48 |awk '{print $1}')
```

在 11613 slot 的源节点上，执行 hash slot 的 migrating 操作

```
# redis-cli -c -h 192.168.6.46 -p 6379 -a 123456 cluster setslot 11613 migrating $destinationnodeid
OK
```

将 11613 的 slot 中的 key 迁移到新节点上

关闭新节点的 protected-mode

```
[root@redis05 ~]# redis-cli -c -h localhost
localhost:6379> config set protected-mode no
OK
localhost:6379> config rewrite
OK
```

查看 11613 的 slot 中，有多少个 key

```
# redis-cli -c -h 192.168.6.46 -p 6379 -a 123456 cluster countkeysinslot 11613
(integer) 1
```

这里表示只有 1 个 key

查看 key 的名称

```
# redis-cli -c -h 192.168.6.46 -p 6379 -a 123456 cluster getkeysinslot 11613 1
1) "hello1"
```

在 11613 的源节点上，将 key 迁移到新节点

```
# redis-cli -c -h 192.168.6.46 -a 123456 migrate 192.168.6.48 6379 hello1 0 2000
OK
```

在所有的 redis 主节点上，更新 11613 的 slot 已迁移到新节点上

```
# destinationnodeid=$(redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes |grep 192.168.6.48 |awk '{print $1}')
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster setslot 11613 node $destinationnodeid
OK
# redis-cli -c -h 192.168.6.45 -p 6379 -a 123456 cluster setslot 11613 node $destinationnodeid
OK
# redis-cli -c -h 192.168.6.46 -p 6379 -a 123456 cluster setslot 11613 node $destinationnodeid
OK
# redis-cli -c -h 192.168.6.48 -p 6379 -a 123456 cluster setslot 11613 node $destinationnodeid
OK
```

更改后的 hash slot 分布情况

```
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes
725b62c8213ab71b0fb4836899268b09ecb2b6f3 192.168.6.44:6379 myself,master - 0 0 1 connected 0-5460
c742745c3b2f006520b57fc660119aed3fd31aab 192.168.6.46:6379 master - 0 1513782487124 3 connected 10923-11612 11614-16383
6be83760f07dda5ec326cfa280308a95c3b2ba02 192.168.6.45:6379 master - 0 1513782486623 2 connected 5461-10922
f7b08e536860b90e00cbf0380cecb057f078dbb7 192.168.6.47:6379 slave 725b62c8213ab71b0fb4836899268b09ecb2b6f3 0 1513782487124 1 connected
8edce63d08111a66428b5fc02505e6d1e4bad445 192.168.6.48:6379 master - 0 1513782486623 4 connected 11613
```

1.1.6.4 删除 redis node

1.1.6.4.1 查看要删除的 node 下有哪些 slot

```
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes |grep '192.168.6.48'
8edce63d08111a66428b5fc02505e6d1e4bad445 192.168.6.48:6379 master - 0 1513783044433 4 connected 11613
```

这里 192.168.6.48 只有 1 个 11613 的 hash slot

1.1.6.4.2 迁移 node 下所有的 hash slot

这里以 11613 的 slot 为例：

获取 11613 的 hash slot 的源 nodeid

```
# sourcenodeid=$(redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes |grep 192.168.6.48 |awk '{print $1}')
```

在要迁移到的目标节点（6.46）上，执行 hash slot 的 importing 操作

```
# redis-cli -c -h 192.168.6.46 -p 6379 -a 123456 cluster setslot 11613 importing $sourcenodeid
OK
```

获取 11613 的 slot 目标节点 nodeid

```
# destinationnodeid=$(redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes |grep 192.168.6.46 |awk '{print $1}')
```

在 11613 slot 的源节点（6.48）上，执行 hash slot 的 migrating 操作

```
# redis-cli -c -h 192.168.6.48 -p 6379 cluster setslot 11613 migrating $destinationnodeid
OK
```

这里的 6.68 上，redis 没有配置密码，所以没有-a 参数

将 11613 的 slot 中的 key 迁移到新节点上

关闭 192.168.6.46 的 protected-mode，并禁用密码

```
# redis-cli -c -h localhost -p 6379
localhost:6379> auth 123456
OK
```

关闭 protected-mode，允许远程无密码登录

```
localhost:6379> config set protected-mode no
OK
```

禁用密码

```
localhost:6379> config set requirepass ""
OK
```

在 11613 的源节点（6.48）上，将 key 迁移到新节点（6.46）

```
# redis-cli -c -h 192.168.6.48 migrate 192.168.6.46 6379 hello1 0 2000
OK
```

在所有的 redis 主节点上，更新 11613 的 slot 已迁移到新节点上

```
# destinationnodeid=$(redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes |grep 192.168.6.46 |awk '{print $1}')
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster setslot 11613 node $destinationnodeid
OK
# redis-cli -c -h 192.168.6.45 -p 6379 -a 123456 cluster setslot 11613 node $destinationnodeid
OK
# redis-cli -c -h 192.168.6.46 -p 6379 -a 123456 cluster setslot 11613 node $destinationnodeid
OK
# redis-cli -c -h 192.168.6.48 -p 6379 -a 123456 cluster setslot 11613 node $destinationnodeid
```

这个返回好像有报错

更改后的 hash slot 分布情况

```
# redis-cli -c -h 192.168.6.48 -p 6379 cluster nodes
725b62c8213ab71b0fb4836899268b09ecb2b6f3 192.168.6.44:6379 master - 0 1513785151644 1 connected 0-5460
8edce63d08111a66428b5fc02505e6d1e4bad445 192.168.6.48:6379 myself,slave c742745c3b2f006520b57fc660119aed3fd31aab 0 0 4 connected
c742745c3b2f006520b57fc660119aed3fd31aab 192.168.6.46:6379 master - 0 1513785151143 5 connected 10923-16383
6be83760f07dda5ec326cfa280308a95c3b2ba02 192.168.6.45:6379 master - 0 1513785151143 2 connected
```

```
5461-10922
f7b08e536860b90e00cbf0380cecb057f078dbb7 192.168.6.47:6379 slave
725b62c8213ab71b0fb4836899268b09ecb2b6f3 0 1513785152147 1 connected
```

此时，6.48 已经没有任何的 hash slot，11613 已经在 6.46 上了。

1.1.6.4.3 在所有的节点上，删除此 nodes

注意：被删除的节点上不需执行 cluster forget <nodeid>，其他的所有节点都需要，包括 master 和 slave 节点；否则删除后又会被同步，以致无法删除

```
# deletenodeid=$(redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster nodes |grep 192.168.6.48 |awk '{print $1}')
# redis-cli -c -h 192.168.6.44 -p 6379 -a 123456 cluster forget $deletenodeid
OK
# redis-cli -c -h 192.168.6.45 -p 6379 -a 123456 cluster forget $deletenodeid
OK
# redis-cli -c -h 192.168.6.46 -p 6379 -a 123456 cluster forget $deletenodeid
OK
# redis-cli -c -h 192.168.6.47 -p 6379 -a 123456 cluster forget $deletenodeid
OK
```

1.1.6.4.4 查看 redis cluster 是否删除了 6.48

```
# redis-cli -c -h 192.168.6.46 -p 6379 -a 123456 cluster nodes
725b62c8213ab71b0fb4836899268b09ecb2b6f3 192.168.6.44:6379 master - 0 1513787466598 1 connected
0-5460
f7b08e536860b90e00cbf0380cecb057f078dbb7 192.168.6.47:6379 slave
725b62c8213ab71b0fb4836899268b09ecb2b6f3 0 1513787466598 1 connected
c742745c3b2f006520b57fc660119aed3fd31aab 192.168.6.46:6379 myself,master - 0 0 5 connected
10923-16383
6be83760f07dda5ec326cfa280308a95c3b2ba02 192.168.6.45:6379 master - 0 1513787465592 2 connected
5461-10922
```

1.1.6.5 安装 keepalived，提供 vip →生产环境使用 haproxy 或 lvs

1.1.6.5.1 在所有的 redis 节点上安装 keepalived

```
# yum install -y keepalived
# systemctl start keepalived && systemctl enable keepalived
```

1.1.6.5.2 第一个节点的 keepalived 配置和脚本

配置文件

```
# cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived
```

```
global_defs {
    router_id redis44
}

vrrp_script chk_redis
{
     script "/etc/keepalived/scripts/redis_check.sh 127.0.0.1 6379 123456"
     interval 2
     timeout 2
     fall 3
}

vrrp_instance redis {
     state MASTER
     interface ens32
     virtual_router_id 51
     priority 100
#       nopreempt
     advert_int 1
     authentication {
          auth_type PASS
          auth_pass 1111
     }
     virtual_ipaddress {
          192.168.6.40
     }
     track_script {
          chk_redis
     }
}
```

解释：
192.168.6.40 是 vip 地址
state 设置为 master
ens32 是网卡名称
127.0.0.1 6379 123456：127.0.0.1 是本机地址，6379 是 redis 端口，123456 是 redis 的密码

脚本

```
# mkdir -p /etc/keepalived/scripts
# cat /etc/keepalived/scripts/redis_check.sh
#!/bin/bash
ALIVE=`/usr/bin/redis-cli -h $1 -p $2 -a $3 PING`
LOGFILE="/var/log/redis/keepalived-redis-check.log"
echo "[CHECK]" >> $LOGFILE
date >> $LOGFILE
if [ $ALIVE == "PONG" ]; then :
    echo "Success: redis-cli -h $1 -p $2 -a no PING $ALIVE" >> $LOGFILE 2>&1
    exit 0
```

```
else
    echo "Failed:redis-cli -h $1 -p $2 -a no PING $ALIVE " >> $LOGFILE 2>&1
    exit 1
fi
```

赋值可执行权限

```
# chmod +x /etc/keepalived/scripts/redis_check.sh
```

1.1.6.5.3 其他所有节点的 keepalived 配置和脚本

配置文件

```
# cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
    router_id redis44
}

vrrp_script chk_redis
{
     script "/etc/keepalived/scripts/redis_check.sh 127.0.0.1 6379 123456"
     interval 2
     timeout 2
     fall 3
}

vrrp_instance redis {
     state slave
     interface ens32
     virtual_router_id 51
     priority 101
#      nopreempt
     advert_int 1
     authentication {
          auth_type PASS
          auth_pass 1111
     }
     virtual_ipaddress {
          192.168.6.40
     }
     track_script {
          chk_redis
     }
}
```

脚本

```
# mkdir -p /etc/keepalived/scripts
# cat /etc/keepalived/scripts/redis_check.sh
```

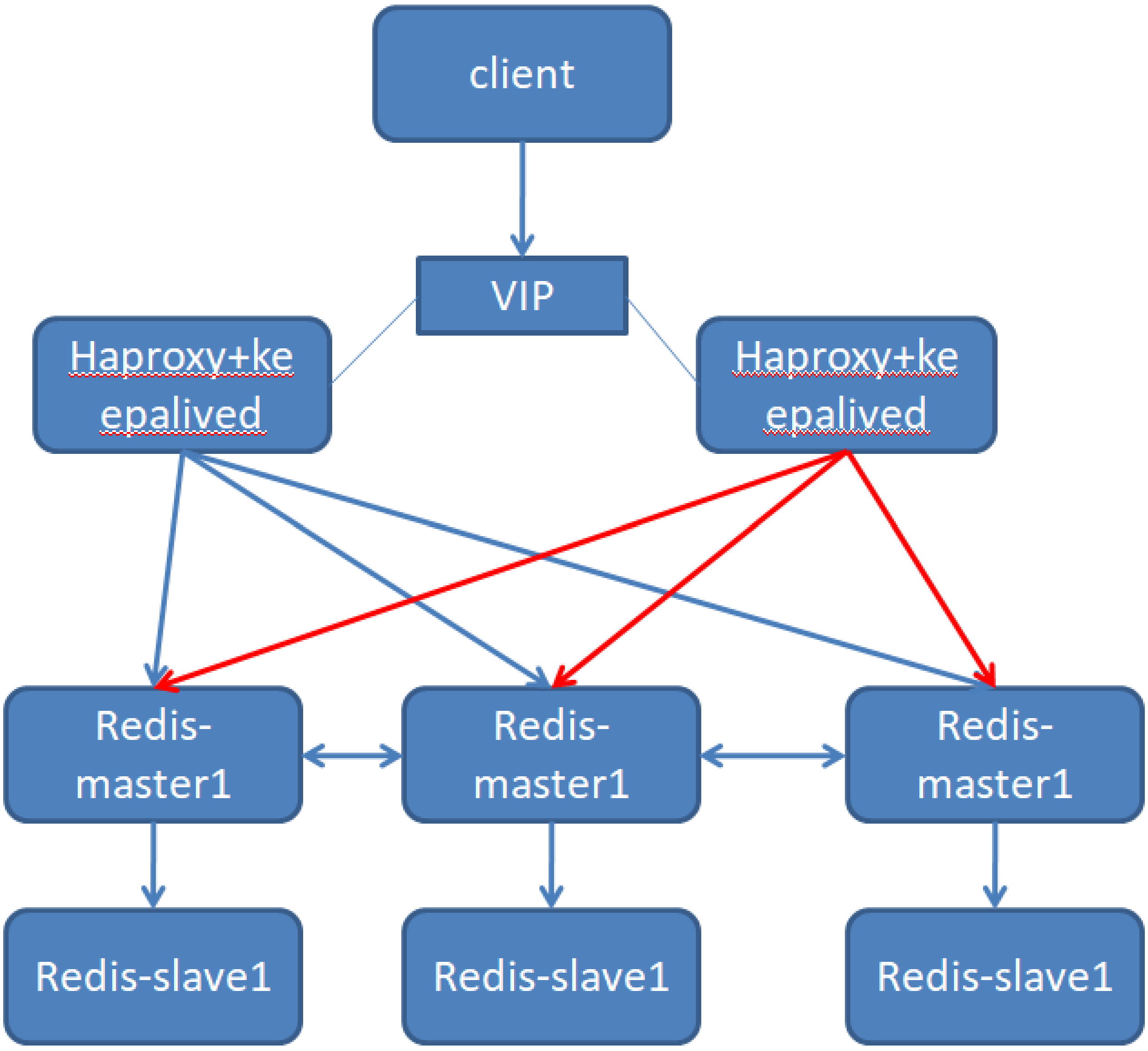
```
#!/bin/bash
ALIVE=`/usr/bin/redis-cli -h $1 -p $2 -a $3 PING`
LOGFILE="/var/log/redis/keepalived-redis-check.log"
echo "[CHECK]" >> $LOGFILE
date >> $LOGFILE
if [ $ALIVE == "PONG" ]; then :
    echo "Success: redis-cli -h $1 -p $2 -a no PING $ALIVE" >> $LOGFILE 2>&1
    exit 0
else
    echo "Failed:redis-cli -h $1 -p $2 -a no PING $ALIVE " >> $LOGFILE 2>&1
    exit 1
fi
```

赋值可执行权限

```
# chmod +x /etc/keepalived/scripts/redis_check.sh
```

1.1.7 haproxy+keepalived 三节点的 redis 负载均衡

1.1.7.1 拓扑图



1.1.7.2 在两个负载均衡节点上，安装 haproxy

```
# yum install -y haproxy
```

1.1.7.3 在两个负载均衡节点上，编辑配置文件

1.1.7.3.1 配置文件内容如下

```
# cat /etc/haproxy/haproxy.cfg
global
    log         127.0.0.1 local2
    chroot        /var/lib/haproxy
    pidfile       /var/run/haproxy.pid
    maxconn       4000
    user          haproxy
    group         haproxy
    daemon
    stats socket /var/lib/haproxy/stats
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 4096

listen redis
    bind 0.0.0.0:6379
    mode tcp
    balance roundrobin
    server etcd01 192.168.6.44 check port 6379 inter 2s rise 3 fall 3 weight 1
    server etcd02 192.168.6.45 check port 6379 inter 2s rise 3 fall 3 weight 1
    server etcd03 192.168.6.46 check port 6379 inter 2s rise 3 fall 3 weight 1
    server redis04 192.168.6.47 check port 6379 inter 2s rise 3 fall 3 weight 1
```

1.1.7.3.2 配置文件解释

bind 0.0.0.0:6379：绑定的地址需要为 0.0.0.0，也就是访问 6379 端口的任意 ip 地址
mode tcp：使用的四层（tcp）负载均衡
balance roundrobin：使用的是轮询
server node01 192.168.6.44 check port 6379 inter 2s rise 3 fall 3 weight 1：
node01 是节点的主机名；
192.168.6.44 是节点的 ip 地址；
check port 6379 是检查节点的 6379 端口是否处于启动状态；
inter 2s 指监控状态检查间隔为 2 秒；
rise 3：表示健康状态检查中，节点从离线状态转换至正常状态需要成功检查的次数；

fall 3：节点从正常状态转换为不可用状态需要检查的次数；

1.1.7.4 在两个负载均衡节点上，启动服务

```
# systemctl start haproxy && systemctl enable haproxy
```

1.1.7.5 在两个负载均衡节点上，安装 keepalived

```
# yum install -y keepalived
```

1.1.7.6 在两个负载均衡节点上，创建 keepalived 监控 haproxy 状态的脚本

创建监控 haproxy 服务的脚本；也就是如果 keepalived 检测到 haproxy 服务 down 了，则先重启，如果重启失败，则关掉 keepalived，以便切换到另外一个 haproxy 服务器上。

```
# cat /etc/keepalived/check_haproxy.sh
#!/bin/bash
#haproxy_num=$(ps -C haproxy --no-header |wc -l)

if [ $(ps -C haproxy --no-header |wc -l) -eq 0 ];then
        systemctl restart haproxy
        echo "restart haproxy"
        sleep 5
fi

if [ $(ps -C haproxy --no-header |wc -l) -eq 0 ];then
        systemctl stop keepalived
        echo "stop keepalived"
fi
```

赋予脚本可执行权限

```
# chmod +x /etc/keepalived/check_haproxy.sh
```

1.1.7.7 在第一个负载均衡节点上，配置 keepalived

```
# cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
    router_id redis1
}

vrrp_script chk_haproxy_status {
        script "/etc/keepalived/check_haproxy.sh"
        interval 2
        weight     2
}


vrrp_instance VIP {
        state MASTER
        interface ens32
        virtual_router_id 51
        priority 100
        advert_int 1
        authentication {
                auth_type PASS
                auth_pass 1111
        }
        track_script {
                chk_haproxy_status
        }
        virtual_ipaddress {
                192.168.6.50
        }
}
```

启动 keepalived 服务

```
# systemctl start keepalived && systemctl enable keepalived
```

参数解释

router_id redis1：router_id 不一致
virtual_ipaddress：是 vip
interface ens32：指定 vip 绑定到哪个接口
state MASTER：keepalive 是主还是从

1.1.7.8 在第二个负载均衡节点上，配置 keepalived

```
# cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
      router_id redis2
}

vrrp_script chk_haproxy_status {
        script "/etc/keepalived/check_haproxy.sh"
        interval 2
        weight    2

}
```

```
vrrp_instance VIP {
    state BACKUP
    interface ens32
    virtual_router_id 51
    priority 80
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    track_script {
        chk_haproxy_status
    }
    virtual_ipaddress {
        192.168.6.50
    }
}
```

启动 keepalived 服务

```
# systemctl start keepalived && systemctl enable keepalived
```

1.1.7.9 验证负载均衡

```
# redis-cli -h 192.168.6.50 -p 6379 -a 123456 cluster nodes
cc1fd3db2218df64267e0cb7e5979513c958bc6b 192.168.6.45:6379 master - 0 1514397406648 2 connected 5461-10922
b5874559e7f308c0eaf62ed439e90e59eec1b720 192.168.6.44:6379 myself,master - 0 0 11 connected 0-5460
a777511afd342deae48c1a16781ca47fcfeaf9dd 192.168.6.47:6379 slave b5874559e7f308c0eaf62ed439e90e59eec1b720 0 1514397405141 11 connected
536cfb2e8b95e19ccee78abaf1ecc3b9e55b74a0 192.168.6.46:6379 master - 0 1514397405643 3 connected 10923-16383
```

1.1.8 日常管理

1.1.8.1 查看集群节点的信息

```
<node-id> <ip:port> <flags> <master> <ping-sent> <pong-recv> <config-epoch> <link-state> <slots>
# redis-cli -h 192.168.6.44 -p 6379 -a 123456 cluster nodes
c742745c3b2f006520b57fc660119aed3fd31aab 192.168.6.46:6379 master - 0 1513767319722 3 connected 10923-16383
```

1.1.8.2 禁用 redis 的密码

```
localhost:6379> config set requirepass ""
OK
localhost:6379> config rewrite
OK
```

1.1.8.3 启用 redis 密码

```
localhost:6379> config set requirepass password1
OK
localhost:6379> config rewrite
OK
```