liangzzz
2017 年 12 月 07 日

Linux运维-搭建高可用Redis缓存

前言

Redis 是一个高性能的 key-value 数据库，现时越来越多企业与应用使用 Redis 作为缓存服务器。楼主是一枚 JAVA 后端程序员，也算是半个运维工程师了。在 Linux 服务器上搭建 Redis ，怎么可以不会呢？下面楼主就带着大家从0开始，依次搭建： Redis 单机服务器 -> Redis 主从复制 -> Redis-Sentinel高可用 。逐步搭建出高可用的Redis缓存服务器。主要参考文章：简书：搭建一个redis高可用系统

楼主基于此文一步步参考着搭建，所以内容较为相似，感谢原文作者，特此申明。

本文同步发布于简书：http://www.jianshu.com/p/d7bc873b8797

搭建Redis

1. 下载并解压

首先从 Redis 官网下载 Redis 并解压，楼主使用的版本是4.0.2。依次执行如下命令：

```
cd /usr/local/
wget http://download.redis.io/releases/redis-4.0.2.tar.gz
tar -zxvf redis-4.0.2.tar.gz
```

如果没有安装 gcc 依赖包，则安装对应依赖包

```
yum install -y gcc-c++ tcl
```

2. 编译并安装

```
cd /usr/local/redis-4.0.2/
make install PREFIX=/usr/local/redis
```

复制 Redis 相关命令到 /usr/sbin 目录下，这样就可以直接执行这些命令，不用写全路径

```
cd /usr/local/redis/bin/
sudo cp redis-cli  redis-server  redis-sentinel   /usr/sbin/
```

3. 建立Redis配置文件

安装完成之后将 Redis 配置文件拷贝到系统配置目录 /etc/ 下， redis.conf 是 Redis 的配置文件， redis.conf 在 Redis 源码目录， port 默认 6379。

```
cp /usr/local/redis-4.0.2/redis.conf  /etc/
```

Redis 配置文件主要参数解析参考

```
daemonize  no                #redis进程是否以守护进程的方式运行,yes为是,no为否(不以守护进程
pidfile /var/run/redis.pid   #指定redis进程的PID文件存放位置
port 6379                    #redis进程的端口号
bind 127.0.0.1               #绑定的主机地址
timeout  300                 #客户端闲置多长时间后关闭连接,默认此参数为0即关闭此功能
loglevel verbose             #redis日志级别,可用的级别有debug.verbose.notice.warning
logfile stdout               #log文件输出位置,如果进程以守护进程的方式运行,此处又将输出文件设
databases 16                 #设置数据库的数量,默认为0可以使用select <dbid>命令在连接上指定
save <seconds> <changes>     #指定在多少时间内刷新次数达到多少的时候会将数据同步到数据文件;
rdbcompression yes           #指定存储至本地数据库时是否压缩文件,默认为yes即启用存储;
dbfilename dump.db           #指定本地数据库文件名
dir ./                       #指定本地数据问就按存放位置;
slaveof <masterip> <masterport>      #指定当本机为slave服务时,设置master服务的IP地址及端口
masterauth <master-password>         #当master设置了密码保护时,slave服务连接master的密码
requirepass footbared        #设置redis连接密码,如果配置了连接密码,客户端在连接redis是需要通
maxclients 128               #设置同一时间最大客户连接数,默认无限制;redis可以同时连接的客户端
maxmemory<bytes>             #指定Redis最大内存限制，Redis在启动时会把数据加载到内存中
```

3.1 设置后端启动：

由于 `Redis` 默认是前端启动，必须保持在当前的窗口中，如果使用 `ctrl + c` 退出，那么 `Redis` 也就退出，不建议使用。

```
vi /etc/redis.conf
```

修改 `Redis` 配置文件把旧值 `daemonize no` 改为 新值 `daemonize yes`

3.2 设置访问：

`Redis` 默认只允许本机访问，可是有时候我们也需要 Redis 被远程访问。

```
vi /etc/redis.conf
```

找到 bind 那行配置，默认是： `# bind 127.0.0.1`

去掉 `#` 注释并改为： `bind 0.0.0.0` 此设置会变成允许所有远程访问。如果想指定限制访问，可设置对应的IP。

3.3 配置Redis日志记录：

找到 `logfile` 那行配置，默认是： `logfile ""` ，改为 `logfile /var/log/redis_6379.log`

3.4 设置 Redis 请求密码：

```
vi /etc/redis.conf
```

找到默认是被注释的这一行： `# requirepass foobared`

修改之后重启下服务

有了密码之后，进入客户端，就得这样访问： `redis-cli -h 127.0.0.1 -p 6379 -a 123456`

4. Redis常用操作

4.1 启动

```
/usr/local/redis/bin/redis-server /etc/redis.conf
```

4.2 关闭

```
/usr/local/redis/bin/redis-cli -h 127.0.0.1 -p 6379 shutdown
```

4.3 查看是否启动

```
ps -ef | grep redis
```

4.4 进入客户端

```
redis-cli
```

4.5 关闭客户端

```
redis-cli shutdown
```

4.6 设置开机自动启动配置

```
echo "/usr/local/redis/bin/redis-server /etc/redis.conf" >> /etc/rc.local
```

添加规则：iptables -I INPUT -p tcp -m tcp --dport 6379 -j ACCEPT

保存规则：service iptables save

重启 iptables：service iptables restart

5. 将Redis注册为系统服务

在/etc/init.d目录下添加Redis服务的启动，暂停和重启脚本：

```
vi /etc/init.d/redis
```

脚本内容如下：

```
#!/bin/sh
#
# redis - this script starts and stops the redis-server daemon
#
# chkconfig:   - 85 15
# description:  Redis is a persistent key-value database
# processname: redis-server
# config:      /usr/local/redis/bin/redis-server
# config:      /etc/redis.conf
# Source function library.
. /etc/rc.d/init.d/functions
# Source networking configuration.
. /etc/sysconfig/network
# Check that networking is up.
[ "$NETWORKING" = "no" ] && exit 0
redis="/usr/local/redis/bin/redis-server"
prog=$(basename $redis)
REDIS_CONF_FILE="/etc/redis.conf"
[ -f /etc/sysconfig/redis ] && . /etc/sysconfig/redis
lockfile=/var/lock/subsys/redis
start() {
    [ -x $redis ] || exit 5
    [ -f $REDIS_CONF_FILE ] || exit 6
    echo -n $"Starting $prog: "
    daemon $redis $REDIS_CONF_FILE
```

```
    return $retval
}
stop() {
    echo -n $"Stopping $prog: "
    killproc $prog -QUIT
    retval=$?
    echo
    [ $retval -eq 0 ] && rm -f $lockfile
    return $retval
}
restart() {
    stop
    start
}
reload() {
    echo -n $"Reloading $prog: "
    killproc $redis -HUP
    RETVAL=$?
    echo
}
force_reload() {
    restart
}
rh_status() {
    status $prog
}
rh_status_q() {
    rh_status >/dev/null 2>&1
}
case "$1" in
    start)
        rh_status_q && exit 0
        $1
        ;;
    stop)
        rh_status_q || exit 0
        $1
```

```
        ;;
    reload)
        rh_status_q || exit 7
        $1
        ;;
    force-reload)
        force_reload
        ;;
    status)
        rh_status
        ;;
    condrestart|try-restart)
        rh_status_q || exit 0
    ;;
    *)
        echo $"Usage: $0 {start|stop|status|restart|condrestart|try-restart| reloa
        exit 2
esac
```

赋予脚本权限

```
chmod 755 /etc/init.d/redis
```

启动、停止和重启：

```
service redis start
service redis stop
service redis restart
```

至此， `Redis` 单机服务器已搭建完毕，下面我们看看主从架构如何搭建。

搭建Redis主从架构

1. redis-server说明

```
172.16.2.185:6379 主
```

2. Redis主从架构配置

- 编辑从机的 `Redis` 配置文件，找到 210 行（大概），默认这一行应该是注释的： `# slaveof <masterip> <masterport>`
- 我们需要去掉该注释，并且填写我们自己的主机的 IP 和 端口，比如： `slaveof 172.16.2.185 6379` ，如果主机设置了密码，还需要找到 `masterauth <master-password>` 这一行，去掉注释，改为 `masterauth 主机密码` 。
- 配置完成后重启从机 `Redis` 服务
- 重启完之后，进入主机的 `redis-cli` 状态下 `redis-cli -h 127.0.0.1 -p 6379 -a 123456` ，输入： `INFO replication` 可以查询到当前主机的 `Redis` 处于什么角色，有哪些从机已经连上主机。

主机信息 `172.16.2.185`

```
# Replication
role:master
connected_slaves:1
slave0:ip=172.16.2.181,port=6379,state=online,offset=28,lag=1
master_replid:625ae9f362643da5337835beaeabfdca426198c7
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:28
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:28
```

从机信息 `172.16.2.181`

```
# Replication
role:slave
master_host:172.16.2.185
```

```
master_sync_in_progress:0
slave_repl_offset:210
slave_priority:100
slave_read_only:1
connected_slaves:0
master_replid:625ae9f362643da5337835beaeabfdca426198c7
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:210
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:210
```

- 此时已经完成了主从配置，我们可以测试下：我们进入主机的 `redis-cli` 状态，然后 `set` 某个值，比如： `set myblog YouMeek.com`
- 我们切换进入从机的 `redis-cli` 的状态下，获取刚刚设置的值看是否存在： `get myblog` ，此时，我们可以发现是可以获取到值的。

3. Redis主从架构总结

- 需要注意的是：从库不具备写入数据能力，不然会报错。 从库只有只读能力。
- 主从架构的优点：除了减少主库连接的压力，还有可以关掉主库的持久化功能，把持久化的功能交给从库进行处理。
- 第一个从库配置的信息是连上主库，后面的第二个从库配置的连接信息是连上第一个从库， 假如还有第三个从库的话，我们可以把第三个从库的配置信息连上第二个从库上，以此类推。

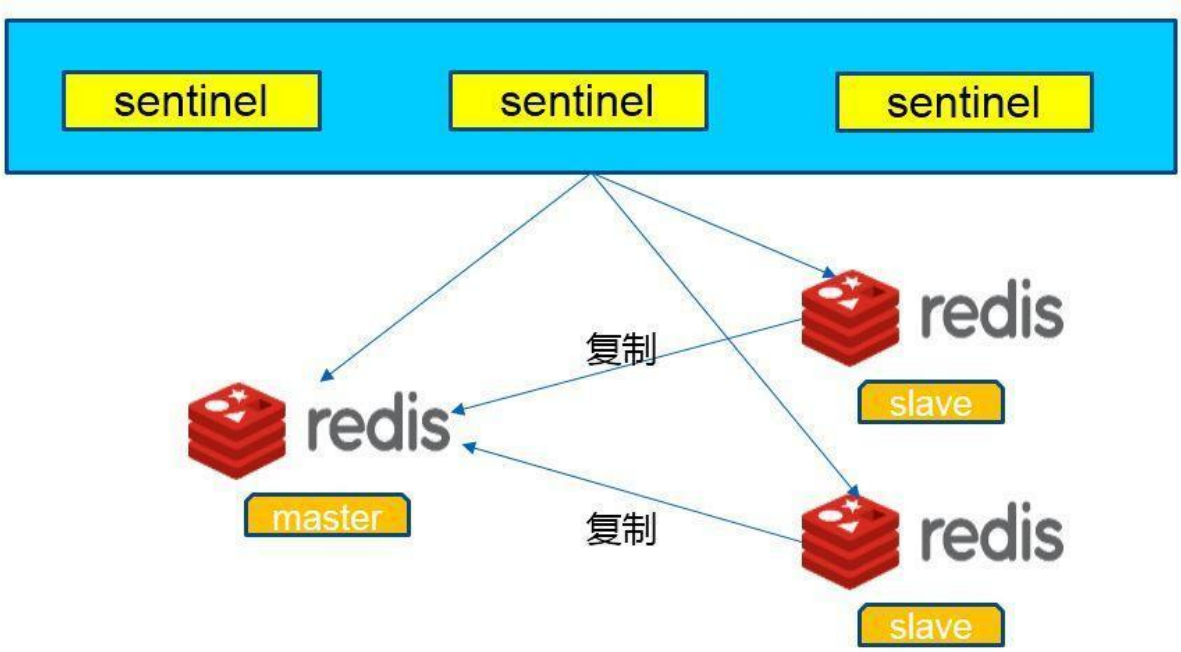
Redis Sentinel高可用架构搭建

1. 自动故障转移

- 虽然使用主从架构配置 `Redis` 做了备份，看上去很完美。但由于 `Redis` 目前

为主 Redis 。

- 这就需要自动故障转移， Redis Sentinel 带有这个功能，当一个主 Redis 不能提供服务时， Redis Sentinel 可以将一个从 Redis 升级为主 Redis ，并对其他从 Redis 进行配置，让它们使用新的主 Redis 进行复制备份。



注意：搭建 Redis Sentinel 推荐至少3台服务器，但由于楼主偷懒，下面用例只用了2台服务器。

Redis Sentinel 的主要功能如下：

1. 监控：哨兵不断的检查 master 和 slave 是否正常的运行。
2. 通知：当监控的某台 Redis 实例发生问题时，可以通过 API 通知系统管理员和其他的应用程序。
3. 自动故障转移：如果一个 master 不正常运行了，哨兵可以启动一个故障转移进程，将一个 slave 升级成为 master ，其他的 slave 被重新配置使用新的 master ，并且应用程序使用 Redis 服务端通知的新地址。
4. 配置提供者：哨兵作为 Redis 客户端发现的权威来源：客户端连接到哨兵请求当

默认情况下，每个 Sentinel 节点会以每秒一次的频率对 Redis 节点和其它的 Sentinel 节点发送 PING 命令，并通过节点的回复来判断节点是否在线。

如果在 down-after-milliseconds 毫秒内，没有收到有效的回复，则会判定该节点为主观下线。

如果该节点为 master ，则该 Sentinel 节点会通过 sentinel is-master-down-by-addr 命令向其它 sentinel 节点询问对该节点的判断，如果超过 <quorum> 个数的节点判定 master 不可达，则该 sentinel 节点会将 master 判断为客观下线。

这个时候，各个 Sentinel 会进行协商，选举出一个领头 Sentinel ，由该领头 Sentinel 对 master 节点进行故障转移操作。

故障转移包含如下三个操作：

1. 在所有的 slave 服务器中，挑选出一个 slave ，并将其转换为 master 。
2. 让其它 slave 服务器，改为复制新的 master 。
3. 将旧 master 设置为新 master 的 slave ，这样，当旧的 master 重新上线时，它会成为新 master 的 slave 。

2. 搭建Redis Sentinel高可用架构

这里使用两台服务器，每台服务器上开启一个 redis-server 和 redis-sentinel 服务。

redis-server说明

```
172.16.2.185:6379 主

172.16.2.181:6379 从
```

redis-sentinel说明

```
172.16.2.185:26379

172.16.2.181:26379
```

如果要做自动故障转移，则建议所有的 `redis.conf` 都设置 `masterauth` ，因为自动故障只会重写主从关系，即 `slaveof` ，不会自动写入 `masterauth` 。如果 `Redis` 原本没有设置密码，则可以忽略。

`Redis` 程序上面已经安装过了，我们只需增加 `redis-sentinel` 的相关配置即可，将 `redis-sentinel` 的配置文件拷贝到系统配置目录 `/etc/` 下， `sentinel.conf` 是 `redis-sentinel` 的配置文件， `sentinel.conf` 在 `Redis` 源码目录。

```
cp /usr/local/redis-4.0.2/sentinel.conf  /etc/
```

修改 `sentinel.conf` 配置文件内容如下：

```
vi /etc/sentinel.conf


protected-mode no
sentinel monitor mymaster 172.16.2.185 6379 2
# redis在搭建时设置了密码，所以要进行密码配置
sentinel auth-pass mymaster “123456“
#5秒内mymaster没有响应，就认为SDOWN
sentinel down-after-milliseconds mymaster 5000
sentinel failover-timeout  mymaster 15000
```

在配置最后加上

```
logfile /var/log/sentinel.log
pidfile  /var/run/sentinel.pid
daemonize yes
```

配置文件说明：

1.port :当前Sentinel服务运行的端口

2.dir : Sentinel服务运行时使用的临时文件夹

3.sentinel monitor master001 192.168.110.10163792:Sentinel去监视一个名为master001的主redis实例，这个主实例的IP地址为本机地址192.168.110.101，端口

4.sentinel down-after-milliseconds master001 30000:指定了Sentinel认为Redis实例已经失效所需的毫秒数。当实例超过该时间没有返回PING，或者直接返回错误，那么Sentinel将这个实例标记为主观下线。只有一个 Sentinel进程将实例标记为主观下线并不一定会引起实例的自动故障迁移：只有在足够数量的Sentinel都将一个实例标记为主观下线之后，实例才会被标记为客观下线，这时自动故障迁移才会执行

5.sentinel parallel-syncs master001 1：指定了在执行故障转移时，最多可以有多少个从Redis实例在同步新的主实例，在从Redis实例较多的情况下这个数字越小，同步的时间越长，完成故障转移所需的时间就越长

6.sentinel failover-timeout master001 180000：如果在该时间（ms）内未能完成failover操作，则认为该failover失败

7.sentinel notification-script ： 指定sentinel检测到该监控的redis实例指向的实例异常时，调用的报警脚本。该配置项可选，但是很常用

2.2 开放防火墙端口

```
添加规则：iptables -I INPUT -p tcp -m tcp --dport 26379 -j ACCEPT
保存规则：service iptables save
重启 iptables：service iptables restart
```

2.3 启动redis-sentinel

```
redis-sentinel  /etc/sentinel.conf
```

在任意一台机子均可查看到相关服务信息

```
redis-cli -h 127.0.0.1 -p 26379

INFO sentinel

# Sentinel
sentinel_masters:1
```

```
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=ok,address=172.16.2.185:6379,slaves=1,sentinels=2
```

3. 自动故障转移测试

3.1 停止主Redis

```
redis-cli -h 172.16.2.185 -p 6379 -a 123456 shutdown
```

3.2 查看redis-sentinel的监控状态

```
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=ok,address=172.16.2.181:6379,slaves=1,sentinels=2
```

发现从库提升为主库。

3.3 注意事项

- 如果停掉 `master` 后， `Sentinel` 显示足够数量的 `sdown` 后，没有出现 `odown` 或 `try-failover` ，则检查密码等配置是否正确
- 如果停掉 `master` 后，试图切换的时候，发现日志出现 `failover-abort-not-elected` ，则分2种情况分别解决：

1. 如果 `Redis` 实例没有配置

```
protected-mode yes
bind 172.16.2.185
```

2. 如果 Redis 实例有配置

```
protected-mode yes
bind 172.16.2.185
```

则在 Sentinel 配置文件加上

```
protected-mode yes
bind 172.16.2.185
```

至此，redis的高可用方案已经搭建完成。

VIP对外提供虚拟IP实现高可用

1. 现有情况概述

客户端程序（如JAVA程序）连接 Redis 时需要 ip 和 port ，但 redis-server 进行故障转移时，主 Redis 是变化的，所以 ip 地址也是变化的。客户端程序如何感知当前主 Redis 的 ip 地址和端口呢？ redis-sentinel 提供了接口，请求任何一个 Sentinel ，发送 SENTINEL get-master-addr-by-name <master name> 就能得到当前主 Redis 的 ip 和 port 。

客户端每次连接 Redis 前，先向 sentinel 发送请求，获得主 Redis 的 ip 和 port ，然后用返回的 ip 和 port 连接 Redis 。

这种方法的缺点是显而易见的，每次操作 Redis 至少需要发送两次连接请求，第一次请求 Sentinel ，第二次请求 Redis 。

更好的办法是使用 VIP ，当然这对配置的环境有一定的要求，比如 Redis 搭建在阿里云服务器上，可能不支持 VIP 。

VIP 方案是， Redis 系统对外始终是同一ip地址，当 Redis 进行故障转移时，需要做的是将 VIP 从之前的 Redis 服务器漂移到现在新的主 Redis 服务器上。

比如：当前 Redis 系统中主 Redis 的 ip 地址是 172.16.2.185 ，那么

当主 Redis 宕机，进行故障转移时， 172.16.2.181 这台服务器上的 Redis 提升为主，这时 VIP（172.16.2.250） 指向 172.16.2.181 ，这样客户端程序不需要修改任何代码，连接的是 172.16.2.181 这台主 Redis 。

2.漂移VIP实现Redis故障转移

那么现在的问题是，如何在进行 Redis 故障转移时，将 VIP 漂移到新的主 Redis 服务器上。

这里可以使用 Redis Sentinel 的一个参数 client-reconfig-script ，这个参数配置执行脚本， Sentinel 在做 failover 的时候会执行这个脚本，并且传递6个参数 <master-name>、 <role>、 <state>、 <from-ip>、 <from-port>、 <to-ip> 、<to-port> ，其中 <to-ip> 是新主 Redis 的 IP 地址，可以在这个脚本里做 VIP 漂移操作。

```
sentinel client-reconfig-script mymaster /opt/notify_mymaster.sh
```

修改两个服务器的 redis-sentinel 配置文件 /etc/sentinel.conf ，增加上面一行。然后在 /opt/ 目录下创建 notify_mymaster.sh 脚本文件，这个脚本做 VIP 漂移操作，内容如下:

```
vi /opt/notify_mymaster.sh

#!/bin/bash
echo "File Name: $0"
echo "Quoted Values: $@"
echo "Quoted Values: $*"
echo "Total Number of Parameters : $#"

MASTER_IP=${6}  #第六个参数是新主redis的ip地址
LOCAL_IP='172.16.2.185'  #当前服务器IP，主机172.16.2.185，从机172.16.2.181
VIP='172.16.2.250'
NETMASK='24'
INTERFACE='eth1'
if [ ${MASTER_IP} = ${LOCAL_IP} ]; then
```

```
else
      sudo /sbin/ip addr del ${VIP}/${NETMASK} dev ${INTERFACE}    #将VIP从该服务器上册
   exit 0
fi
exit 1  #如果返回1，sentinel会一直执行这个脚本
```

赋予脚本权限

```
chmod 755 /opt/notify_mymaster.sh
```

现在当前主 `Redis` 是 `172.16.2.185` ，需要手动绑定 `VIP` 到该服务器上。

```
/sbin/ip  addr add 172.16.2.250/24 dev eth1
/sbin/arping -q   -c 3 -A 172.16.2.250 -I eth1
```

由于VIP只能绑定只有一台机子，所以建议将改为 `bind 0.0.0.0` 添加至 `redis.conf` 中

```
vi /etc/redis.conf
```

设置 `bind 0.0.0.0`

由于VIP只能绑定只有一台机子，所以建议将改为 `bind 0.0.0.0` 添加至 `sentinel.conf` 中

```
vi /etc/sentinel.conf
```

设置 `bind 0.0.0.0`

重启 `Redis`

```
service redis restart`
```

重启 `Sentinel`

```
redis-sentinel /etc/sentinel.conf
```

```
redis-cli -h 172.16.2.250 -p 6379 -a 123456 INFO replication
```

可正常通讯，信息如下：

```
# Replication
role:master
connected_slaves:1
slave0:ip=172.16.2.181,port=6379,state=online,offset=0,lag=0
master_replid:325b0bccab611d329d9c2cd2c35a1fe3c01ae196
master_replid2:c1f7a7d17d2c35575a34b00eb10c8abf32df2243
master_repl_offset:22246293
second_repl_offset:22241024
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:22237293
repl_backlog_histlen:9001
```

访问主机的 `Sentinel`

```
redis-cli -h 172.16.2.250 -p 26379 INFO sentinel
```

可正常通讯，信息如下：

```
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=ok,address=172.16.2.185:6379,slaves=1,sentinels=3
```

下面关闭主机的 `Redis` 服务，看看VIP是否漂移到另一台服务器上。

```
redis-cli -h 172.16.2.185 -p 6379 -a 123456 shutdown
```

查看是否已进行切换

```
# Sentinel
sentinel_masters:1
sentinel_tilt:0
sentinel_running_scripts:0
sentinel_scripts_queue_length:0
sentinel_simulate_failure_flags:0
master0:name=mymaster,status=ok,address=172.16.2.181:6379,slaves=1,sentinels=3
```

通过查询 `Sentinel` 发现从机 `172.16.2.181` 提升为主。

通过访问 `VIP` 的方式连接 `Redis`

```
redis-cli -h 172.16.2.250 -p 6379 -a 123456 INFO replication
```

```
# Replication
role:master
connected_slaves:0
master_replid:cab30a4083f35652053ffcd099d70b9aaf7a80f3
master_replid2:3da856dd33cce4bedd54926df6797b410f1ab9e8
master_repl_offset:74657
second_repl_offset:36065
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:74657
```

从上面信息可知， `VIP` 已经飘移成功。可喜可贺，大吉大利，晚上吃鸡。

总结

至此，高可用 `Redis` 缓存服务已搭建完毕，迟点会再出一篇文章教大家如何通过 `JAVA` 连接 `Redis` 进行相关操作。至于 `Redis Cluster` 集群方案，等有空再搭建然后再和大家一同分享。

参考文章

Redis 复制、Sentinel的搭建和原理说明

Redis 快速入门(官网翻译)

Redis Sentinel机制与用法（一）

Redis哨兵-实现Redis高可用

读懂Redis并配置主从集群及高可用部署

在Redis Sentinel环境下，jedis该如何配置

redis sentinel 主从切换(failover)解决方案，详细配置

Redis-3.2.1主从故障测试实例

Redis Linux Java

88 分享到

liangzzz
JAVA工程师 @ 珠海
个人主页

相关文章

JAVA并发-自问自答学ThreadLocal
liangzzz ♥ 181 💬 2

JAVA容器-自问自答学HashMap
liangzzz ♥ 112 💬 3

JAVA容器-自问自答学LinkedList
liangzzz ♥ 82 💬 5

JAVA基础-自问自答学hashCode和equals
liangzzz ♥ 71 💬 2

评论

说说你的看法

牛盾
写的很详细！很棒

▲ 0 评论 56分钟前