

Redis 集群

一. 单节点实例

单节点实例还是比较简单的，平时做个测试，写个小程序如果需要用到缓存的话，启动一个 Redis 还是很轻松的，做为一个 key/value 数据库也是可以胜任的。

二. 主从模式 (master/slaver)

2.1 搭建方式

主从模式的简介和配置，以下面搭建一个 master 与两个 slave 为例
创建一个 master-slave 文件夹

```
$ mkdir master-slave
```

进入 master-slave

```
$ cd master-slave
```

创建三个文件夹 redis11, redis12, redis13。其中 redis11 为 master 文件，redis12/redis13 为 slave 文件夹。准备需要的 redis 执行文件

```
$ mkdir redis11 redis12 redis13
```

复制 redis-server 到每一个文件夹。

```
$ cp redis-5.0.7/src/redis-server ./redis11
```

```
$ cp redis-5.0.7/src/redis-server ./redis12
```

```
$ cp redis-5.0.7/src/redis-server ./redis13
```

复制 redis-cli 到 master-slave 文件夹

```
$ cp redis-5.0.7/src/redis-cli ./
```

在 master 的 redis11 里面添加配置文件

```
$ vim redis.conf
```

```
bind 0.0.0.0
port 8001

logfile "8001.log"

dbfilename "dump-8001.rdb"

daemonize yes

rdbcompression yes
```

在 slave 的 redis12 里面配置文件

```
bind 0.0.0.0
port 8002

logfile "8002.log"

dbfilename "dump-8002.rdb"

daemonize yes

rdbcompression yes

slaveof 192.168.199.131 8001
```

在 slave 的 redis13 里面配置文件

```
bind 0.0.0.0
port 8003

logfile "8003.log"

dbfilename "dump-8003.rdb"

daemonize yes

rdbcompression yes

slaveof 192.168.199.131 8001
```

创建 start-all.sh 的 shell 脚本文件

```
#!/bin/bash

cd redis11/
./redis-server redis.conf &
cd ..
cd redis12/
./redis-server redis.conf &
cd ..
cd redis13/
./redis-server redis.conf &
cd ..
```

查看 redis 启动情况

```
$ netstat -anop | grep redis
```

```
tcp        0      0 0.0.0.0:8001          0.0.0.0:*            LISTEN      2340/redis-server 0 off (0.00/0/0)
tcp        0      0 0.0.0.0:8002          0.0.0.0:*            LISTEN      2342/redis-server 0 off (0.00/0/0)
tcp        0      0 0.0.0.0:8003          0.0.0.0:*            LISTEN      2341/redis-server 0 off (0.00/0/0)
tcp        0      0 0.0.0.0:17001         0.0.0.0:*            LISTEN      1000/redis-server 0 off (0.00/0/0)
```

主从模式的搭建还是很简单的，官网的介绍就更加简单了，只需要在配置文件中加上一行配置：

```
slaveof 192.168.1.1 6379
```

指明 master 的 ip 和端口号就可以了，实际上真的这么简单。

2.2 测试主从模式

```
wangbojing@ubuntu:~/share/redis-cluster/master-slave$ ./redis-cli -p 8001
127.0.0.1:8001>
127.0.0.1:8001>
127.0.0.1:8001> get name
(nil)
127.0.0.1:8001> exit
wangbojing@ubuntu:~/share/redis-cluster/master-slave$ ./redis-cli -p 8002
127.0.0.1:8002> get name
(nil)
127.0.0.1:8002> exit
wangbojing@ubuntu:~/share/redis-cluster/master-slave$ ./redis-cli -p 8003
127.0.0.1:8003> get name
(nil)
127.0.0.1:8003> exit
wangbojing@ubuntu:~/share/redis-cluster/master-slave$
wangbojing@ubuntu:~/share/redis-cluster/master-slave$ ./redis-cli -p 8001
127.0.0.1:8001> set name king
OK
127.0.0.1:8001> get name
"king"
127.0.0.1:8001> exit
wangbojing@ubuntu:~/share/redis-cluster/master-slave$ ./redis-cli -p 8002
127.0.0.1:8002> get name
"king"
127.0.0.1:8002> exit
wangbojing@ubuntu:~/share/redis-cluster/master-slave$ ./redis-cli -p 8003
127.0.0.1:8003> get name
"king"
127.0.0.1:8003> █
```

2.3 其他配置

在 `redis.conf` 中，还有一些关于主从的其他配置，按需配置即可。

首先谈谈我对主从模式的必要性：

主从模式的一个作用是备份数据，这样当一个节点损坏（指不可恢复的硬件损坏）时，数据因为有备份，可以方便恢复。

另一个作用是负载均衡，所有客户端都访问一个节点肯定会影响 Redis 工作效率，有了主从以后，查询操作就可以通过查询从节点来完成。

对主从模式必须的理解（结论已经验证过，可以自行验证）：

一个 Master 可以有多个 Slaves

默认配置下，master 节点可以进行读和写，slave 节点只能进行读操作，写操作被禁止。不要修改配置让 slave 节点支持写操作，没有意义，原因一，写入的数据不会被同步到其他节点；原因二，当 master 节点修改同一条数据后，slave 节点的数据会被覆盖掉。slave 节点挂了不影响其他 slave 节点的读和 master 节点的读和写，重新启动后会将数据从 master 节点同步过来。

master 节点挂了以后，不影响 slave 节点的读，Redis 将不再提供写服务，master 节点启动后 Redis 将重新对外提供写服务。

master 节点挂了以后，不会 slave 节点重新选一个 master

对有密码的情况说明一下，当 **master** 节点设置密码时：
客户端访问 **master** 需要密码
启动 **slave** 需要密码，在配置中进行配置即可
客户端访问 **slave** 不需要密码

2.4 主从节点的缺点

主从模式的缺点其实从上面的描述中可以得出：

master 节点挂了以后，**redis** 就不能对外提供写服务了，因为剩下的 **slave** 不能成为 **master**

这个缺点影响是很大的，尤其是对生产环境来说，是一刻都不能停止服务的，所以一般的生产环境是不会单单只有主从模式的。所以有了下面的 **sentinel** 模式。

三. sentinel 模式

3.1 搭建方式

在之前的主从模式基础上面，将端口从 8001 改为 9001，8002 改为 9002，8003 改为 9003。

```
wangbojing@ubuntu: ~/share/redis-cluster/sentinel$ ls
redis21 redis22 redis23 redis-cli start-all.sh
wangbojing@ubuntu:~/share/redis-cluster/sentinel$
```

复制 **redis-sentinel** 到 **redis21/redis22/redis23** 的文件夹中

```
wangbojing@ubuntu:~/share/redis-cluster/sentinel$ cp ../redis-5.0.7/src/redis-sentinel ./redis21/
```

复制 **sentinel.conf** 到 **redis21/redis22/redis23**

```
wangbojing@ubuntu:~/share/redis-cluster/sentinel$ cp ../redis-5.0.7/sentinel.conf ./redis21/
```

修改 **sentinel.conf** 文件

```
# port <sentinel-port>
# The port that this sentinel instance will
port 19001

# By default Redis Sentinel does not run as
# Note that Redis will write a pid file in /
# daemonized.

# time while performing the synchronization with the m
sentinel monitor mymaster 192.168.199.131 9002 2
```

3.2 测试 sentinel 模式

```
127.0.0.1:19001> sentinel master mymaster
1) "name"
2) "mymaster"
3) "ip"
4) "192.168.199.131"
5) "port"
6) "9001"
7) "runid"
8) "c2b38d2e5bdbbb222627b83a2845e4a3135605f0"
9) "flags"
10) "master"
11) "link-pending-commands"
12) "0"
13) "link-refcount"
14) "1"
15) "last-ping-sent"
16) "0"
17) "last-ok-ping-reply"
18) "244"
19) "last-ping-reply"
20) "244"
21) "down-after-milliseconds"
22) "30000"
23) "info-refresh"
24) "4507"
25) "role-reported"
26) "master"
27) "role-reported-time"
28) "64807"
29) "config-epoch"
30) "0"
31) "num-slaves"
```

```
wangbojing@ubuntu:~/share/redis-cluster/sentinel$ netstat -anop | grep redis
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:19001        0.0.0.0:*            LISTEN     2460/redis-sentinel off (0.00/0/0)
tcp        0      0 0.0.0.0:19002        0.0.0.0:*            LISTEN     2461/redis-sentinel off (0.00/0/0)
tcp        0      0 0.0.0.0:19003        0.0.0.0:*            LISTEN     2462/redis-sentinel off (0.00/0/0)
tcp        0      0 0.0.0.0:9001         0.0.0.0:*            LISTEN     2464/redis-server 0 off (0.00/0/0)
tcp        0      0 0.0.0.0:9002         0.0.0.0:*            LISTEN     2463/redis-server 0 off (0.00/0/0)
tcp        0      0 0.0.0.0:9003         0.0.0.0:*            LISTEN     2465/redis-server 0 off (0.00/0/0)
tcp        0      0 192.168.199.131:19001 192.168.199.131:39534 ESTABLISHED 2460/redis-sentinel keepalive (201.09/0/0)
```

```
wangbojing@ubuntu:~/share/redis-cluster/sentinel$ kill 2464
```

```
127.0.0.1:19001> sentinel master mymaster
1) "name"
2) "mymaster"
3) "ip"
4) "192.168.199.131"
5) "port"
6) "9003"
7) "runid"
8) "11cacbc452836c56a007b34e7a2cbcae98102d2f"
9) "flags"
10) "master"
11) "link-pending-commands"
12) "0"
13) "link-refcount"
14) "1"
15) "last-ping-sent"
16) "0"
17) "last-ok-ping-reply"
18) "115"
19) "last-ping-reply"
20) "115"
21) "down-after-milliseconds"
22) "30000"
23) "info-refresh"
24) "7065"
25) "role-reported"
26) "master"
27) "role-reported-time"
28) "68140"
29) "config-epoch"
30) "1"
31) "num-slaves"
```

3.3 sentinel 原理

1、Redis sentinel 介绍

Redis Sentinel 是 Redis 高可用的实现方案。Sentinel 是一个管理多个 Redis 实例的工具，它可以实现对 Redis 的监控、通知、自动故障转移。

2、Redis Sentinel 的主要功能

Sentinel 的主要功能包括主节点存活检测、主从运行情况检测、自动故障转移（failover）、主从切换。Redis 的 Sentinel 最小配置是一主一从。Redis 的 Sentinel 系统可以用来管理多个 Redis 服务器，该系统可以执行以下四个任务：

监控

Sentinel 会不断的检查主服务器和从服务器是否正常运行。

通知

当被监控的某个 Redis 服务器出现问题，Sentinel 通过 API 脚本向管理员或者其他的应用程

序发送通知。

自动故障转移

当主节点不能正常工作时，Sentinel 会开始一次自动的故障转移操作，它会将与失效主节点是主从关系的其中一个从节点升级为新的主节点，并且将其他的从节点指向新的主节点。

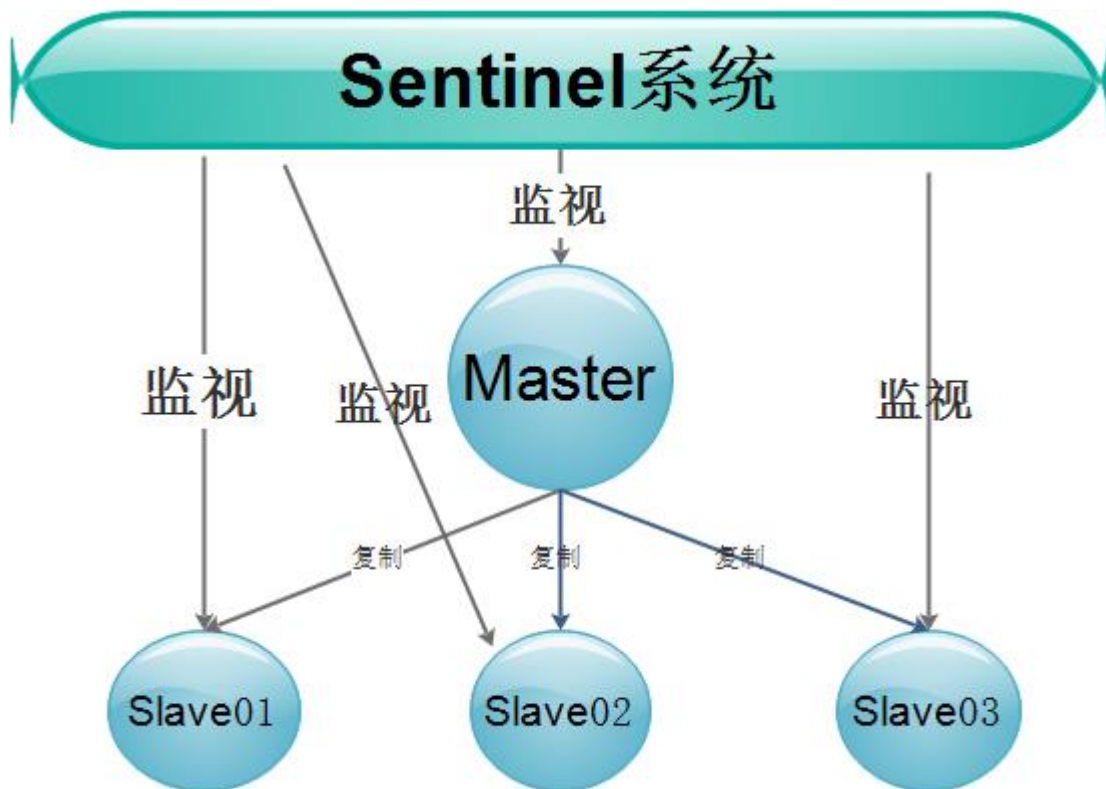
配置提供者

在 Redis Sentinel 模式下，客户端应用在初始化时连接的是 Sentinel 节点集合，从中获取主节点的信息。

3、Redis Sentinel 的工作流程

Sentinel 是 Redis 的高可用性解决方案：

由一个或多个 Sentinel 实例组成的 Sentinel 系统可以监视任意多个主服务器，以及所有从服务器，并在被监视的主服务器进入下线状态时，自动将下线主服务器属下的某个从服务器升级为新的主服务器，然后由新的主服务器代替已下线的主服务器继续处理命令请求。如下图：



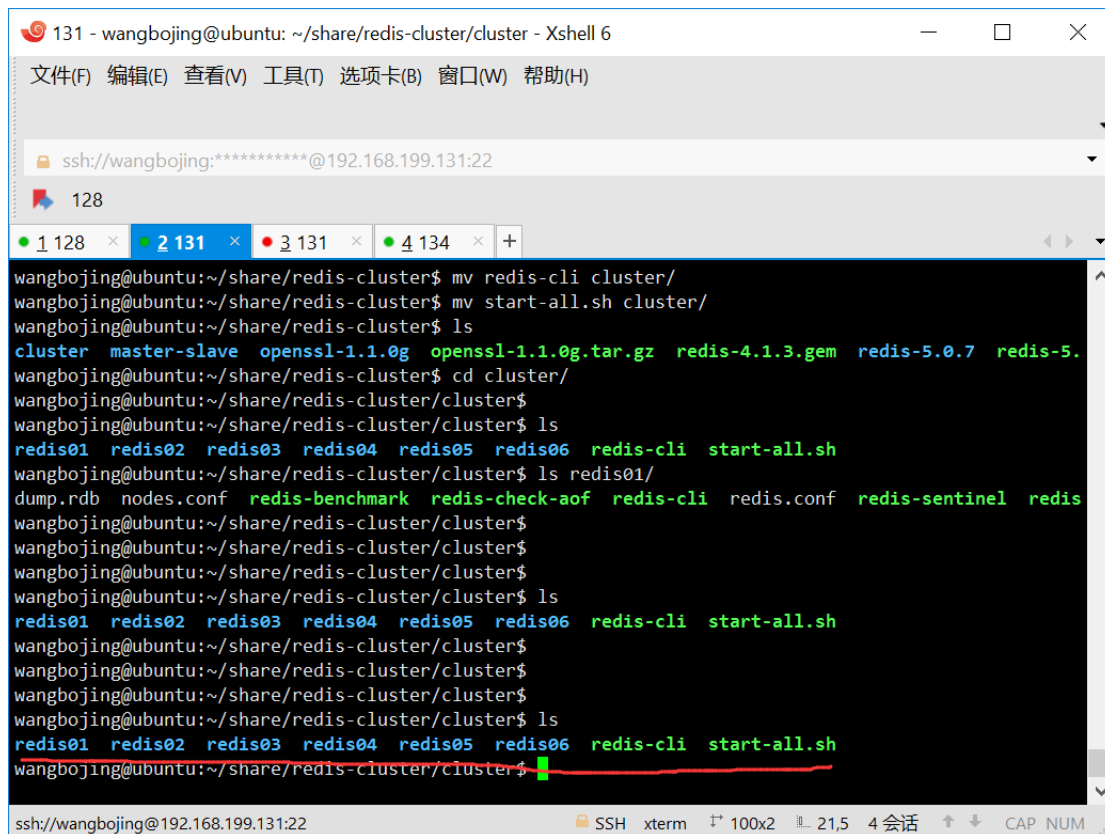
Sentinel 负责监控集群中的所有主、从 Redis，当发现主故障时，Sentinel 会在所有的从中选一个成为新的主。并且会把其余的从变为新主的从。同时那台有问题的旧主也会变为新主的从，也就是说当旧的主即使恢复时，并不会恢复原来的主身份，而是作为新主的一个从。在 Redis 高可用架构中，Sentinel 往往不是只有一个，而是有 3 个或者以上。目的是为了使其更加可靠，毕竟主和从切换角色这个过程还是蛮复杂的。

sentinel 模式基本可以满足一般生产的需求，具备高可用性。但是当数据量过大到一台服务器存放不下的情况时，主从模式或 sentinel 模式就不能满足需求了，这个时候需要对存储的数据进行分片，将数据存储到多个 Redis 实例中，就是下面要讲的。

四. cluster 模式

4.1 搭建方法

创建 redis01/redis02/redis03/redis04/redis05/redis06。创建六个节点。

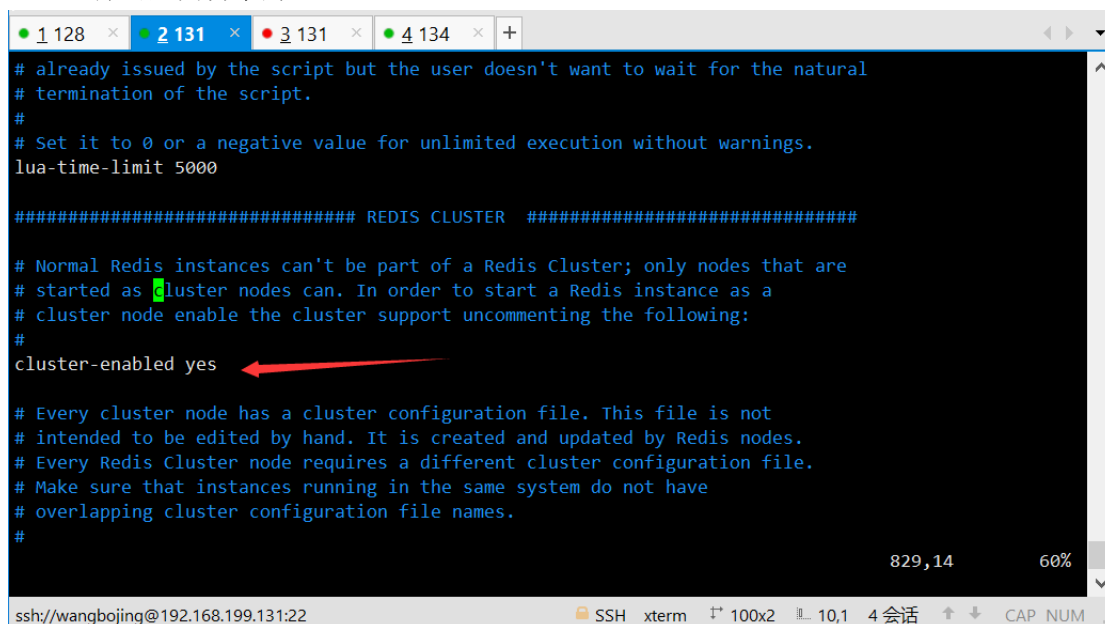


```
131 - wangbojing@ubuntu: ~/share/redis-cluster/cluster - Xshell 6
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H)

ssh://wangbojing:*****@192.168.199.131:22
128

wangbojing@ubuntu:~/share/redis-cluster$ mv redis-cli cluster/
wangbojing@ubuntu:~/share/redis-cluster$ mv start-all.sh cluster/
wangbojing@ubuntu:~/share/redis-cluster$ ls
cluster master-slave openssl-1.1.0g openssl-1.1.0g.tar.gz redis-4.1.3.gem redis-5.0.7 redis-5.
wangbojing@ubuntu:~/share/redis-cluster$ cd cluster/
wangbojing@ubuntu:~/share/redis-cluster/cluster$ ls
redis01 redis02 redis03 redis04 redis05 redis06 redis-cli start-all.sh
wangbojing@ubuntu:~/share/redis-cluster/cluster$ ls redis01/
dump.rdb nodes.conf redis-benchmark redis-check-aof redis-cli redis.conf redis-sentinel redis
wangbojing@ubuntu:~/share/redis-cluster/cluster$
wangbojing@ubuntu:~/share/redis-cluster/cluster$
wangbojing@ubuntu:~/share/redis-cluster/cluster$ ls
redis01 redis02 redis03 redis04 redis05 redis06 redis-cli start-all.sh
wangbojing@ubuntu:~/share/redis-cluster/cluster$
wangbojing@ubuntu:~/share/redis-cluster/cluster$
wangbojing@ubuntu:~/share/redis-cluster/cluster$ ls
redis01 redis02 redis03 redis04 redis05 redis06 redis-cli start-all.sh
wangbojing@ubuntu:~/share/redis-cluster/cluster$
```

为 redis01..redis06 每一个文件夹里面添加 redis.conf。redis.conf 可以直接使用 redis 源码里面自带的



```
# already issued by the script but the user doesn't want to wait for the natural
# termination of the script.
#
# Set it to 0 or a negative value for unlimited execution without warnings.
lua-time-limit 5000

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

# Normal Redis instances can't be part of a Redis Cluster; only nodes that are
# started as cluster nodes can. In order to start a Redis instance as a
# cluster node enable the cluster support uncommenting the following:
#
cluster-enabled yes

# Every cluster node has a cluster configuration file. This file is not
# intended to be edited by hand. It is created and updated by Redis nodes.
# Every Redis Cluster node requires a different cluster configuration file.
# Make sure that instances running in the same system do not have
# overlapping cluster configuration file names.
#
```



```
# is running).
#
# IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO ALL THE INTERFACES
# JUST COMMENT THE FOLLOWING LINE.
# ~~~~~
bind 0.0.0.0

# Protected mode is a layer of security protection, in order to avoid that
# Redis instances left open on the internet are accessed and exploited.
#
# When protected mode is on and if:
#
# 1) The server is not binding explicitly to a set of addresses using the
#    "bind" directive.
# 2) No password is configured.
#
# The server only accepts connections from clients connecting from the
# IPv4 and IPv6 loopback addresses 127.0.0.1 and ::1, and from Unix domain
# sockets.
```

83,1

4%

```
# even if no authentication is configured, nor a specific set of interfaces
# are explicitly listed using the "bind" directive.
protected-mode yes

# Accept connections on the specified port, default is 6379 (IANA #815344).
# If port 0 is specified Redis will not listen on a TCP socket.
port 7001

# TCP listen() backlog.
#
# In high requests-per-second environments you need an high backlog in order
# to avoid slow clients connections issues. Note that the Linux kernel
# will silently truncate it to the value of /proc/sys/net/core/somaxconn so
# make sure to raise both the value of somaxconn and tcp_max_syn_backlog
# in order to get the desired effect.
tcp-backlog 511

# Unix socket.
#
# Specify the path for the Unix socket that will be used to listen for
```

105,14

6%

跟之前配置 sentinel 模式与主从模式一样。

```
wangbojing@ubuntu:~/share/redis-cluster/cluster/redis01$ ls
dump.rdb  redis-benchmark  redis-cli  redis-sentinel
nodes.conf  redis-check-aof  redis.conf  redis-server
wangbojing@ubuntu:~/share/redis-cluster/cluster/redis01$
```

创建 start-all.sh 的 shell 文件

```
#!/bin/bash

cd redis01
./redis-server redis.conf &
cd ..
cd redis02
./redis-server redis.conf &
cd ..
cd redis03
./redis-server redis.conf &
cd ..
cd redis04
./redis-server redis.conf &
cd ..
cd redis05
./redis-server redis.conf &
cd ..
cd redis06
./redis-server redis.conf &
cd ..
```

```
wangbojing@ubuntu:~/share/redis-cluster/cluster$ ./redis-cli --cluster create 192.168.199.131:7001 192.168.199.131:7002 192.168.199.131:7003 192.168.199.131:7004 192.168.199.131:7005 192.168.199.131:7006 --cluster-replicas 1
```

```
$ ./redis-cli --cluster create 192.168.199.131:7001 192.168.199.131:7002 192.168.199.131:7003 192.168.199.131:7004 192.168.199.131:7005 192.168.199.131:7006 --cluster-replicas 1
```

```
wangbojing@ubuntu:~/share/redis-cluster/cluster$ ./redis-cli --cluster create 192.168.199.131:7001 192.168.199.131:7002 192.168.199.131:7003 192.168.199.131:7004 192.168.199.131:7005 192.168.199.131:7006 --cluster-replicas 1
[ERR] Node 192.168.199.131:7001 is not empty. Either the node already knows other nodes (check with CLUSTER NODES) or contains some keys in database 0.
```

表示节点已经自己构建了集群。

cluster 的出现是为了解决单机 Redis 容量有限的问题，将 Redis 的数据根据一定的规则分配到多台机器。对 **cluster** 的一些理解：

cluster 可以说是 **sentinel** 和主从模式的结合体，通过 **cluster** 可以实现主从和 **master** 重选功能，所以如果配置两个副本三个分片的话，就需要六个 Redis 实例。

因为 Redis 的数据是根据一定规则分配到 **cluster** 的不同机器的，当数据量过大时，可以新增机器进行扩容

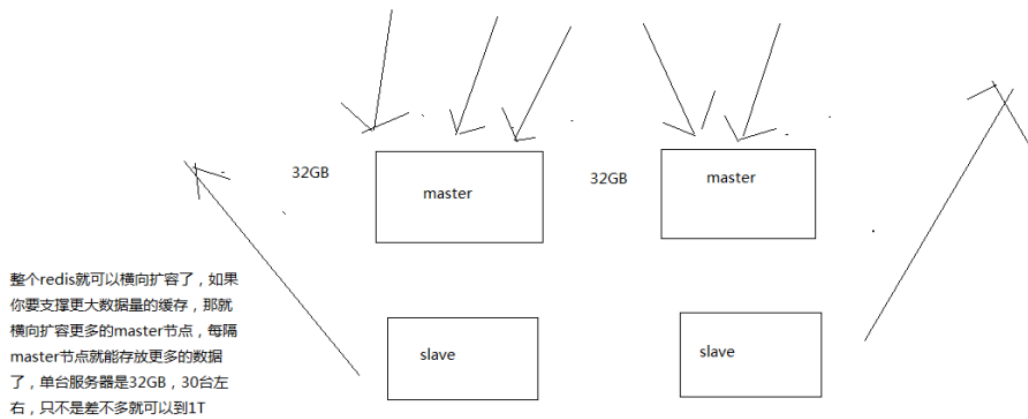
这种模式适合数据量巨大的缓存要求，当数据量不是很大使用 **sentinel** 即可。

4.2 cluster 原理

redis cluster 是 Redis 的分布式解决方案，在 3.0 版本推出后有效地解决了 redis 分布式方面的需求。自动将数据进行分片，每个 **master** 上放一部分数据。提供内置的高可用支持，部分 **master** 不可用时，还是可以继续工作的

支撑 N 个 redis master node，每个 master node 都可以挂载多个 slave node 高可用，因为每个 master 都有 slave 节点，那么如果 mater 挂掉，redis cluster 这套机制，就会自动将某个

slave 切换成 master

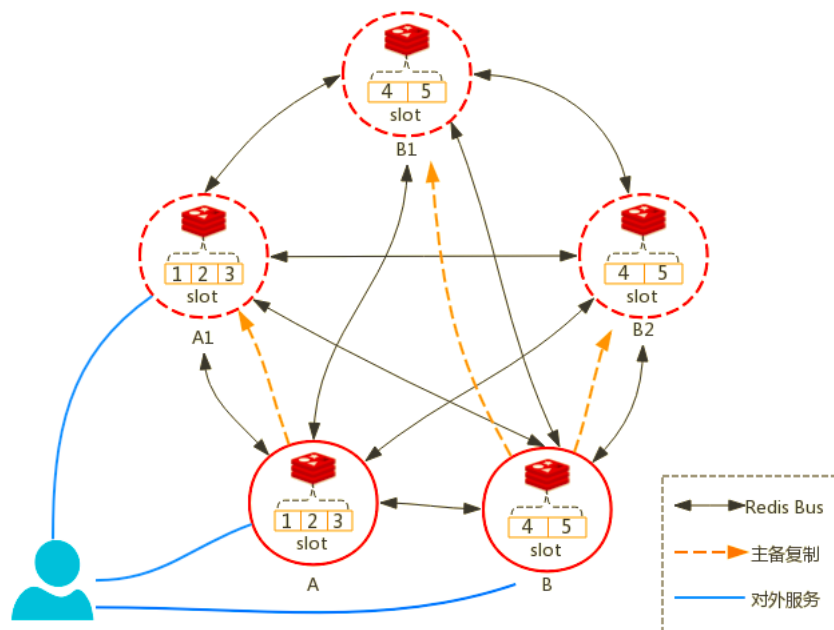


redis cluster vs. replication + sentinel

如果你的数据量很少, 主要是承载高并发高性能的场景, 比如你的缓存一般就几个 G, 单机足够了

replication, 一个 mater, 多个 slave, 要几个 slave 跟你的要求的读吞吐量有关系, 然后自己搭建一个 **sentinal** 集群, 去保证 **redis** 主从架构的高可用性, 就可以了

redis cluster, 主要是针对海量数据+高并发+高可用的场景, 海量数据, 如果你的数据量很大, 那么建议就用 **redis cluster**



数据分布算法

hash 算法

比如你有 N 个 redis 实例，那么如何将一个 key 映射到 redis 上呢，你很可能会采用类似下面的通用方法计算 key 的 hash 值，然后均匀的映射到到 N 个 redis 上：

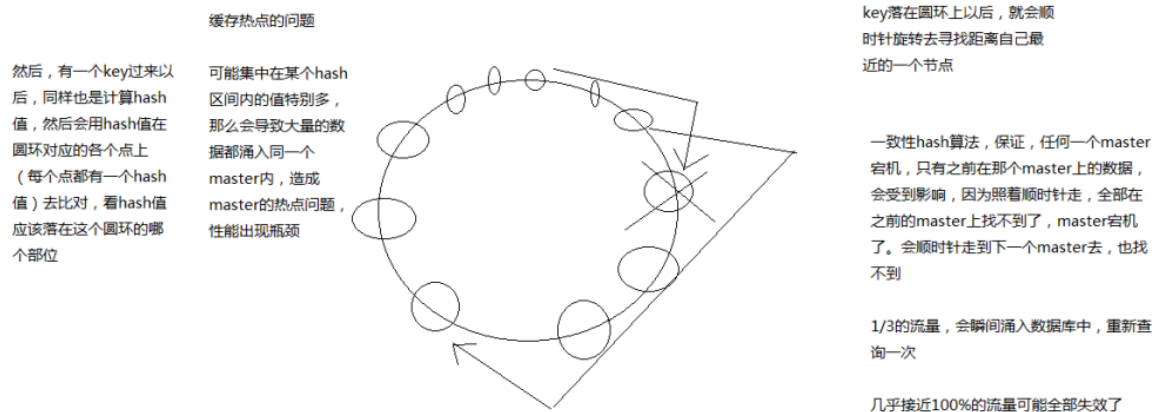
$\text{hash}(\text{key}) \% N$

如果增加一个 redis，映射公式变成了 $\text{hash}(\text{key}) \% (N+1)$

如果一个 redis 宕机了，映射公式变成了 $\text{hash}(\text{key}) \% (N-1)$

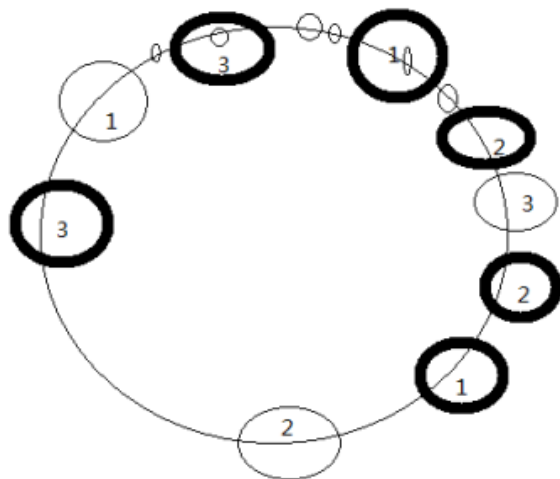
在这两种情况下，几乎所有的缓存都失效了。会导致数据库访问的压力陡增，严重情况，还可能导致数据库宕机。

一致性 hash 算法



一个 master 宕机不会导致大部分缓存失效，可能存在缓存热点问题

用虚拟节点改进



给每隔master都做了均匀分布的虚拟节点

然后这样的话，在每隔区间内，大量的数据，都会均匀地分布到不同的节点内，而不是按照顺时针的顺序去走，全部涌入同一个master内

hash slot 算法

redis cluster 有固定的 16384 个 hash slot，对每个 key 计算 CRC16 值，然后对 16384 取模，可以获取 key 对应的 hash slot

redis cluster 中每个 master 都会持有部分 slot，比如有 3 个 master，那么可能每个 master 持有 5000 多个 hash slot

hash slot 让 node 的增加和移除很简单，增加一个 master，就将其他 master 的 hash slot 移动部分过去，减少一个 master，就将它的 hash slot 移动到其他 master 上去

移动 hash slot 的成本是非常低的

客户端的 api，可以对指定的数据，让他们走同一个 hash slot，通过 hash tag 来实现

127.0.0.1:7000>CLUSTER ADDSLOTS 0 1 2 3 4 ... 5000 可以将槽 0-5000 指派给节点 7000 负责。

每个节点都会记录哪些槽指派给了自己，哪些槽指派给了其他节点。

客户端向节点发送键命令，节点要计算这个键属于哪个槽。

如果是自己负责这个槽，那么直接执行命令，如果不是，向客户端返回一个 MOVED 错误，指引客户端转向正确的节点。

多 master 的写入

在 redis cluster 写入数据的时候，其实是可以将请求发送到任意一个 master 上去执行，但是，每个 master 都会计算这个 key 对应的 CRC16 值，然后对 16384 个 hashslot 取模，找到 key 对应的 hashslot，找到 hashslot 对应的 master

如果对应的 master 就在自己本地的话，set mykey1 v1, mykey1 这个 key 对应的 hashslot 就在自己本地，那么自己就处理掉了

但是如果计算出来的 hashslot 在其他 master 上，那么就会给客户端返回一个 moved error，告诉你，你得到哪个 master 上去执行这条写入的命令

什么叫做多 master 的写入，就是每条数据只能存在于一个 master 上，不同的 master 负责存储不同的数据，分布式的数据存储

100w 条数据，5 个 master，每个 master 就负责存储 20w 条数据，分布式数据存储

默认情况下，redis cluster 的核心理念，主要是用 slave 做高可用的，每个 master 挂两个 slave，主要是做数据的热备，还有 master 故障时的主备切换，实现高可用的

redis cluster 默认是不支持 slave 节点读或者写的，跟我们手动基于 replication 搭建的主从架构不一样的

jedis 客户端，对 redis cluster 的读写分离支持不太好的

默认的话就是读和写都到 master 上去执行的

如果你要让最流行的 jedis 做 redis cluster 的读写分离的访问，那可能还得自己修改一点 jedis 的源码，成本比较高

读写分离，是为了什么，主要是因为要建立一主多从的架构，才能横向任意扩展 slave node 去支撑更大的读吞吐量

redis cluster 的架构下，实际上本身 master 就是可以任意扩展的，你如果要支撑更大的读吞吐量，或者写吞吐量，或者数据量，都可以直接对 master 进行横向扩展就可以了

节点间的内部通信机制

1. 基础通信原理

(1) redis cluster 节点间采取 gossip 协议进行通信

跟集中式不同，不是将集群元数据（节点信息，故障，等等）集中存储在某个节点上，而是互相之间不断通信，保持整个集群所有节点的数据是完整的

集中式：好处在于，元数据的更新和读取，时效性非常好，一旦元数据出现了变更，立即就更新到集中式的存储中，其他节点读取的时候立即就可以感知到；不好在于，所有的元数据的更新压力全部集中在一个地方，可能会导致元数据的存储有压力

gossip：好处在于，元数据的更新比较分散，不是集中在一个地方，更新请求会陆陆续

续，打到所有节点上去更新，有一定的延时，降低了压力；缺点，元数据更新有延时，可能导致集群的一些操作会有一些滞后

（2）10000 端口

每个节点都有一个专门用于节点间通信的端口，就是自己提供服务的端口号+10000，比如 7001，那么用于节点间通信的就是 17001 端口

每隔节点每隔一段时间都会往另外几个节点发送 ping 消息，同时其他节点接收到 ping 之后返回 pong

（3）交换的信息

故障信息，节点的增加和移除，hash slot 信息，等等

2. gossip 协议

gossip 协议包含多种消息，包括 ping, pong, meet, fail, 等等

meet: 某个节点发送 meet 给新加入的节点，让新节点加入集群中，然后新节点就会开始与其他节点进行通信

`redis-trib.rb add-node`

其实内部就是发送了一个 gossip meet 消息，给新加入的节点，通知那个节点去加入我们的集群

ping: 每个节点都会频繁给其他节点发送 ping，其中包含自己的状态还有自己维护的集群元数据，互相通过 ping 交换元数据

每个节点每秒都会频繁发送 ping 给其他的集群，ping，频繁的互相之间交换数据，互相进行元数据的更新

pong: 返回 ping 和 meet，包含自己的状态和其他信息，也可以用于信息广播和更新

fail: 某个节点判断另一个节点 fail 之后，就发送 fail 给其他节点，通知其他节点，指定的节点宕机了

3. ping 消息深入

ping 很频繁，而且要携带一些元数据，所以可能会加重网络负担

每个节点每秒会执行 10 次 ping，每次会选择 5 个最久没有通信的其他节点

当然如果发现某个节点通信延时达到了 $\text{cluster_node_timeout} / 2$ ，那么立即发送 ping，避免数据交换延时过长，落后的时间太长了

比如说，两个节点之间都 10 分钟没有交换数据了，那么整个集群处于严重的元数据不一致的情况，就会有问题

所以 `cluster_node_timeout` 可以调节，如果调节比较大，那么会降低发送的频率

每次 ping，一个是带上自己节点的信息，还有就是带上 1/10 其他节点的信息，发送出去，进行数据交换

至少包含 3 个其他节点的信息，最多包含总节点-2 个其他节点的信息

基于重定向的客户端

1. 请求重定向

客户端可能会挑选任意一个 redis 实例去发送命令，每个 redis 实例接收到命令，都会计算 key 对应的 hash slot

如果在本地就在本地处理，否则返回 moved 给客户端，让客户端进行重定向

cluster keyslot mykey，可以查看一个 key 对应的 hash slot 是什么

用 redis-cli 的时候，可以加入 -c 参数，支持自动的请求重定向，redis-cli 接收到 moved 之后，会自动重定向到对应的节点执行命令

2. 计算 hash slot

计算 hash slot 的算法，就是根据 key 计算 CRC16 值，然后对 16384 取模，拿到对应的 hash slot

用 hash tag 可以手动指定 key 对应的 slot，同一个 hash tag 下的 key，都会在一个 hash slot 中，比如 set mykey1:{100} 和 set mykey2:{100}

3. hash slot 查找

节点间通过 gossip 协议进行数据交换，就知道每个 hash slot 在哪个节点上

高可用性与主备切换原理

redis cluster 的高可用的原理，几乎跟哨兵是类似的

1、判断节点宕机

如果一个节点认为另外一个节点宕机，那么就是 pfail，主观宕机

如果多个节点都认为另外一个节点宕机了，那么就是 fail，客观宕机，跟哨兵的原理几乎一样，sdown，odown

在 cluster-node-timeout 内，某个节点一直没有返回 pong，那么就被认为 pfail

如果一个节点认为某个节点 pfail 了，那么会在 gossip ping 消息中，ping 给其他节点，如果超过半数的节点都认为 pfail 了，那么就会变成 fail

2、从节点过滤

对宕机的 master node，从其所有的 slave node 中，选择一个切换成 master node
检查每个 slave node 与 master node 断开连接的时间，如果超过了 cluster-node-timeout
* cluster-slave-validity-factor，那么就没有资格切换成 master
这个也是跟哨兵是一样的，从节点超时过滤的步骤

3、从节点选举

哨兵：对所有从节点进行排序，slave priority, offset, run id
每个从节点，都根据自己对 master 复制数据的 offset，来设置一个选举时间，offset 越大（复制数据越多）的从节点，选举时间越靠前，优先进行选举
所有的 master node 开始 slave 选举投票，给要进行选举的 slave 进行投票，如果大部分 master node ($N/2 + 1$) 都投票给了某个从节点，那么选举通过，那个从节点可以切换成 master
从节点执行主备切换，从节点切换为主节点

4、与哨兵比较

整个流程跟哨兵相比，非常类似，所以说，redis cluster 功能强大，直接集成了 replication 和 sentinel 的功能