redis

是什么?

是完全开源免费的，用c语言编写的，是一个单线程，高性能的（key/value）内存数据库，基于内存运行并支持持久化的nosql数据库

能干嘛?

主要是用来做缓存，但不仅仅只能做缓存，比如：redis的计数器生成分布式唯一主键，redis实现分布式锁，队列，会话缓存。

去哪下?

官网，也可以通过Linux yum直接下载安装

怎么玩?

1.安装 2.redis数据类型（api操作） 3.redis配置文件解析 4.redis的持久化 5.redis的事务 6.redis的发布订阅

7.java客户端操作（jedis）

redis的安装

1.解压 2.make 如果make报错的话 大家就可以看一下是不是报没有gcc的错 如果是报没有gcc的错，那就要先安装一个gcc

yum install gcc-c++ 安装好gcc之后最好执行一下make distclean 因为前面make的时候它执行了一些东西 要先把他清掉 3.make install

查看redis默认安装位置 /usr/local/bin

redis设置外网访问

```
1.注释bind并且把protected-mode no
2.使用bind
3.设置密码
protected-mode它启用的条件有两个，第一是没有使用bind，第二是没有设置访问密码。
```

redis数据类型及api操作(http://redisdoc.com/)

key

keys *

scan 0 match * count 1

exists key 判断某个key是否存在

move key db 当前库就没有了，到指定的库中去了

expire key 为给定的key设置过期时间

ttl key 查看还有多少时间过期 -1表示永不过期 -2表示已过期

type key 查看key是什么类型

1.string

string是redis最基本的类型，你可以理解成与Memcached一模一样的类型，一个key对应一个value。

string类型是二进制安全的。意思是redis的string可以包含任何数据。比如jpg图片或者序列化的对象 。

string类型是Redis最基本的数据类型，一个redis中字符串value最多可以是512M

set key value 设置key value

get key 查看当前key的值

del key 删除key

append key value 如果key存在，则在指定的key末尾添加，如果key存在则类似set

strlen key 返回此key的长度

以下几个命令只有在key值为数字的时候才能正常操作

---

incr key 为执定key的值加一

decr key 为指定key的值减一

incrby key 数值 为指定key的值增加数值

decrby key 数值 为指定key的值减数值

---

getrange key 0(开始位置) -1（结束位置） 获取指定区间范围内的值，类似between……and的关系 （0 -1）表示全部

setrange key 1（开始位置，从哪里开始设置） 具体值 设置（替换）指定区间范围内的值

setex 键 秒值 真实值 设置带过期时间的key，动态设置。

setnx key value 只有在 key 不存在时设置 key 的值。

mset key1 value key2 value 同时设置一个或多个 key-value 对。

mget key1 key 2 获取所有(一个或多个)给定 key 的值。

msetnx key1 value key2 value 同时设置一个或多个 key-value 对，当且仅当所有给定 key 都不存在。

getset key value 将给定 key 的值设为 value ，并返回 key 的旧值(old value)。

2.list

它是一个字符串链表，left、right都可以插入添加； 如果键不存在，创建新的链表； 如果键已存在，新增内容； 如果值全移除，对应的键也就消失了。 链表的操作无论是头和尾效率都极高，但假如是对中间元素进行操作，效率就很惨淡了。

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素导列表的头部（左边）或者尾部（右边）。 它的底层实际是个链表

lpush key value1 value2 将一个或多个值加入到列表头部

rpush key value1 value2 将一个或多个值加入到列表底部

lrange key start end 获取列表指定范围的元素 （0 -1）表示全部

lpop key 移出并获取列表第一个元素

rpop key 移出并获取列表最后一个元素

lindex key index 通过索引获取列表中的元素

llen 获取列表长度

lrem key 0（数量） 值，表示删除全部给定的值。零个就是全部值 从left往right删除指定数量个值等于指定值的元素，返回的值为实际删除的数量

ltrim key start(从哪里开始截) end（结束位置） 截取指定索引区间的元素，格式是ltrim list的key 起始索引 结束索引

3.set

Redis的Set是string类型的无序，不能重复的集合。

sadd key value1 value 2 向集合中添加一个或多个成员

smembers key 返回集合中所有成员

sismembers key member 判断member元素是否是集合key的成员

scard key 获取集合里面的元素个数

srem key value 删除集合中指定元素

srandmember key 数值 从set集合里面随机取出指定数值个元素 如果超过最大数量就全部取出，

spop key 随机移出并返回集合中某个元素

smove key1 key2 value(key1中某个值) 作用是将key1中执定的值移除 加入到key2集合中

sdiff key1 key2 在第一个set里面而不在后面任何一个set里面的项(差集)

sinter key1 key2 在第一个set和第二个set中都有的 （交集）

sunion key1 key2 两个集合所有元素（并集）

4.hash

Redis hash 是一个键值对集合。 Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象。

kv模式不变，但v是一个键值对

类似Java里面的Map

hset key (key value) 向hash表中添加一个元素

hget key key 向hash表中获取一个元素

hmset key key1 value1 key2 value2 key3 value3 向集合中添加一个或多个元素

hmget key key1 key2 key3 向集合中获取一个或多个元素

hgetall key 获取在hash列表中指定key的所有字段和值

hdel key key1 key2 删除一个或多个hash字段

hlen key 获取hash表中字段数量

hexits key key 查看hash表中，指定key（字段）是否存在

hkeys key 获取指定hash表中所有key（字段）

hvals key 获取指定hash表中所有value(值)

hincrdy key key1 数量（整数） 执定hash表中某个字段加 数量 ， 和incr一个意思

hincrdyfloat key key1 数量（浮点数，小数） 执定hash表中某个字段加 数量 ， 和incr一个意思

hsetnx key key1 value1 与hset作用一样，区别是不存在赋值，存在了无效。

5.zset

Redis zset 和 set 一样也是string类型元素的集合,且不允许重复的成员。 不同的是每个元素都会关联一个double类型的分数。 redis正是通过分数来为集合中的成员进行从小到大的排序。zset的成员是唯一的,但分数(score)却可以重复。

zadd key score 值 score 值 向集合中添加一个或多个成员

zrange key 0 -1 表示所有 返回指定集合中所有value

zrange key 0 -1 withscores 返回指定集合中所有value和score

zrangebyscore key 开始score 结束score 返回指定score间的值

zrem key score某个对应值（value），可以是多个值 删除元素

zcard key 获取集合中元素个数

zcount key 开始score 结束score 获取分数区间内元素个数

zrank key vlaue 获取value在zset中的下标位置(根据score排序)

zscore key value 按照值获得对应的分数

redis的持久化机制

说白了，就是在指定的时间间隔内，将内存当中的数据集快照写入磁盘，它恢复时是将快照文件直接读到内存

什么意思呢？我们都知道，内存当中的数据，如果我们一断电，那么数据必然会丢失，但是玩过redis的同学应该都知道，我们一关机之后再启动的时候数据是还在的，所以它必然是在redis启动的时候重新去加载了持久化的文件

redis提供两种方式进行持久化，

一种是RDB持久化默认，

另外一种是AOF（append only file）持久化。

1.RDB

是什么?

原理是redis会单独创建（fork）一个与当前进程一模一样的子进程来进行持久化，这个子线程的所有数据（变量。环境变量，程序程序计数器等）都和原进程一模一样，会先将数据写入到一个临时文件中，待持久化结束了，再用这个临时文件替换上次持久化好的文件，整个过程中，主进程不进行任何的io操作，这就确保了极高的性能

1.这个持久化文件在哪里

2.他什么时候fork子进程，或者什么时候触发rdb持久化机制

shutdown时，如果没有开启aof，会触发 配置文件中默认的快照配置 执行命令save或者bgsave save是只管保存，其他不管，全部阻塞 bgsave： redis会在后台异步进行快照操作，同时可以响应客户端的请求 执行flushall命令 但是里面是空的，无意义

2.aof(--fix)　ls -l --block-size=M

是什么?

原理是将Reids的操作日志以追加的方式写入文件，读操作是不记录的

1.这个持久化文件在哪里

2.触发机制（根据配置文件配置项）

no：表示等操作系统进行数据缓存同步到磁盘（快，持久化没保证） always：同步持久化，每次发生数据变更时，立即记录到磁盘（慢，安全） everysec：表示每秒同步一次（默认值,很快，但可能会丢失一秒以内的数据）

3.aof重写机制

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AOF文件大小的增长率大于该配置项时自动开启重写（这里指超过原大小的100%）。 auto-aof-rewrite-percentage 100

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AOF文件大小大于该配置项时自动开启重写

auto-aof-rewrite-min-size 64mb

4.redis4.0后混合持久化机制

开启混合持久化

4.0版本的混合持久化默认关闭的，通过aof-use-rdb-preamble配置参数控制，yes则表示开启，no表示禁用，5.0之后默认开启。

混合持久化是通过bgrewriteaof完成的，不同的是当开启混合持久化时，fork出的子进程先将共享的内存副本全量的以RDB方式写入aof文件，然后在将重写缓冲区的增量命令以AOF方式写入到文件，写入完成后通知主进程更新统计信息，并将新的含有RDB格式和AOF格式的AOF文件替换旧的的AOF文件。简单的说：新的AOF文件前半段是RDB格式的全量数据后半段是AOF格式的增量数据，

优点：混合持久化结合了RDB持久化 和 AOF 持久化的优点, 由于绝大部分都是RDB格式，加载速度快，同时结合AOF，增量的数据以AOF方式保存了，数据更少的丢失。

缺点：兼容性差，一旦开启了混合持久化，在4.0之前版本都不识别该aof文件，同时由于前部分是RDB格式，阅读性较差

小总结:

1.redis提供了rdb持久化方案，为什么还要aof?

优化数据丢失问题，rdb会丢失最后一次快照后的数据，aof丢失不会超过2秒的数据

2.如果aof和rdb同时存在，听谁的?

aof

3.rdb和aof优势劣势

rdb 适合大规模的数据恢复，对数据完整性和一致性不高， 在一定间隔时间做一次备份，如果redis意外down机的话，就会丢失最后一次快照后的所有操作 aof 根据配置项而定

1.官方建议 两种持久化机制同时开启，如果两个同时开启 优先使用aof持久化机制

性能建议（这里只针对单机版redis持久化做性能建议）：

因为RDB文件只用作后备用途，只要15分钟备份一次就够了，只保留save 900 1这条规则。

如果Enalbe AOF，好处是在最恶劣情况下也只会丢失不超过两秒数据，启动脚本较简单只load自己的AOF文件就可以了。 代价一是带来了持续的IO，二是AOF rewrite的最后将rewrite过程中产生的新数据写到新文件造成的阻塞几乎是不可避免的。 只要硬盘许可，应该尽量减少AOF rewrite的频率，AOF重写的基础大小默认值64M太小了，可以设到5G以上。 默认超过原大小100%大小时重写可以改到适当的数值。

---

redis集群专题

Redis主从复制

1.是什么

1.单机有什么问题：

单机故障

容量瓶颈

qps瓶颈

主机数据更新后根据配置和策略，自动同步到备机的master/slaver机制，mester已写为主，slaver已读为主

2.能干嘛

1.读写分离

2.容灾备份

3.怎么玩

玩法原则:

1.配从不配主

2.使用命令 SLAVEOF 动态指定主从关系 ， 如果设置了密码，关联后使用 config set masterauth 密码

3.配置文件和命令混合使用时，如果混合使用，动态指定了主从，请注意一定要修改对应的配置文件

1.新建redis8000,redis8001,redis8002文件夹

2.将redis.conf文件复制在redis8000下

3.分别修改个目录下的redis.conf文件

redis8000/redis.conf

1.bind 192.168.0.104 指定本机ip

2.port 8000

3.daemonize yes

4.pidfile /var/run/redis_8000.pid

5.dir /myredis/redis8000

6.requirepass 123456

4.把redis8000/redis.conf文件复制到redis8001,redis8002下

redis8001/redis.conf

1. :%s/8000/8001/g 批量替换

2. replicaof 192.168.0.104 8000

3. masterauth 123456

redis8002/redis.conf

```
1.  :%s/8000/8002/g    批量替换
2.  replicaof 192.168.0.104 8000
3.  masterauth 123456
```

5.分别启动8000.8001,8002实例

[root@localhost myredis]# /usr/local/bin/redis-server /myredis/redis8000/redis.conf [root@localhost myredis]# /usr/local/bin/redis-server /myredis/redis8001/redis.conf [root@localhost myredis]# /usr/local/bin/redis-server /myredis/redis8002/redis.conf

6.客户端连接

/usr/local/bin/redis-cli -h 192.168.0.104 -p 8000 -a 123456

/usr/local/bin/redis-cli -h 192.168.0.104 -p 8001 -a 123456

/usr/local/bin/redis-cli -h 192.168.0.104 -p 8002 -a 123456

4.全量复制消耗

1.bgsave时间 2.rdb文件网络传输 3.从节点请求请求数据时间 4.从节点加载rdb的时间 5.可能的aof重写时间

5.缺点

1.由于所有的写操作都是先在Master上操作，然后同步更新到Slave上，所以从Master同步到Slave机器有一定的延迟，当系统很繁忙的时候，延迟问题会更加严重，Slave机器数量的增加也会使这个问题更加严重。

2.当主机宕机之后，将不能进行写操作，需要手动将从机升级为主机，从机需要重新制定master

简单总结:

一个master可以有多个Slave

一个slave只能有一个master

数据流向是单向的，只能从主到从

redis哨兵模式

1.是什么，能干嘛?

在Redis 2.8版本开始引入。哨兵的核心功能是主节点的自动故障转移。

通俗来讲哨兵模式的出现是就是为了解决我们主从复制模式中需要我们人为操作的东西变为自动版，并且它比人为要更及时

2.哨兵主要功能（做了哪些事）

监控（Monitoring）：哨兵会不断地检查主节点和从节点是否运作正常。

自动故障转移（Automatic Failover）：当主节点不能正常工作时，哨兵会开始自动故障转移操作，它会将失效主节点的其中一个从节点升级为新的主节点，并让其他从节点改为复制新的主节点。

配置提供者（Configuration Provider）：客户端在初始化时，通过连接哨兵来获得当前Redis服务的主节点地址。

通知（Notification）：哨兵可以将故障转移的结果发送给客户端。

其中，监控和自动故障转移功能，使得哨兵可以及时发现主节点故障并完成转移；而配置提供者和通知功能，则需要在与客户端的交互中才能体现。

3.架构

哨兵节点：哨兵系统由一个或多个哨兵节点组成，哨兵节点是特殊的Redis节点，不存储数据。

数据节点：主节点和从节点都是数据节点。

4.怎么玩（实战）？

1.部署主从节点

哨兵系统中的主从节点，与普通的主从节点配置是一样的，并不需要做任何额外配置。下面分别是主节点（port=8000）和2个从节点（port=8001/8002）的配置文件；

我们刚才搭建的主从复制就是主从节点

2.部署哨兵节点

哨兵节点本质上是特殊的Redis节点。

3个哨兵节点的配置几乎是完全一样的，主要区别在于端口号的不同（26379 / 26380 / 263 81）下面以26379节点为例介绍节点的配置和启动方式；配置部分尽量简化：

sentinel-26379.conf

port 26379

daemonize yes

logfile "26379.log"

sentinel monitor mymaster 192.168.0.104 6379 2

其中，sentinel monitor mymaster 192.168. 92.128 6379 2配置的含义是：该哨兵节点监92.168.0.104 6379这个主节点，该主节点的名称是mymaster，最后的2的含义与主节点的故障判定有关：至少需要2个哨兵节点同意，才能判定主节点故障并进行故障转移。

哨兵节点的启动有两种方式，二者作用是完全相同的：

redis-sentinel sentinel-26379.conf

redis-server sentinel-26379.conf --sentinel

5.故障转移演示（哨兵的监控和自动故障转移功能）

使用kill命令杀掉主节点

6.客户端（jedis）访问哨兵系统（自动故障转移功能）

```
public static void main(String[] args)  {
        Logger logger= LoggerFactory.getLogger(TestJedisSentinel.class);
        Set<String> set=new HashSet<>();
        set.add("192.168.0.104:28000");
        set.add("192.168.0.104:28001");
        set.add("192.168.0.104:28002");
        JedisSentinelPool jedisSentinelPool=new JedisSentinelPool("mymaster",set,"123456");
        while (true) {
```

```
            Jedis jedis=null;
            try {
                jedis = jedisSentinelPool.getResource();
                String s = UUID.randomUUID().toString();
                jedis.set("k" + s, "v" + s);
                System.out.println(jedis.get("k" + s));
                Thread.sleep(1000);
            }catch (Exception e){
                logger.error(e.getMessage());
            }finally {
                if(jedis!=null){
                    jedis.close();
                }
            }
        }
    }
```

7.基本原理

关于哨兵的原理，关键是了解以下几个概念：

主观下线：在心跳检测的定时任务中，如果其他节点超过一定时间没有回复，哨兵节点就会将其进行主观下线。顾名思义，主观下线的意思是一个哨兵节点“主观地”判断下线；与主观下线相对应的是客观下线。

客观下线：哨兵节点在对主节点进行主观下线后，会通过sentinel is-master-down-by-addr命令询问其他哨兵节点该主节点的状态；如果判断主节点下线的哨兵数量达到一定数值，则对该主节点进行客观下线。

需要特别注意的是，客观下线是主节点才有的概念；如果从节点和哨兵节点发生故障，被哨兵主观下线后，不会再有后续的客观下线和故障转移操作。

定时任务：每个哨兵节点维护了3个定时任务。定时任务的功能分别如下：

1.每10秒通过向主从节点发送info命令获取最新的主从结构；

发现slave节点

确定主从关系

2.每2秒通过发布订阅功能获取其他哨兵节点的信息；SUBSCRIBE c2 PUBLISH c2 hello-redis

交互对节点的“看法”和自身情况

3.每1秒通过向其他节点发送ping命令进行心跳检测，判断是否下线（monitor）。

心跳检测，失败判断依据

选举领导者哨兵节点：当主节点被判断客观下线以后，各个哨兵节点会进行协商，选举出一个领导者哨兵节点，并由该领导者节点对其进行故障转移操作。

监视该主节点的所有哨兵都有可能被选为领导者，选举使用的算法是Raft算法；Raft算法的基本思路是先到先得：即在一轮选举中，哨兵A向B发送成为领导者的申请，如果B没有同意过其他哨兵，则会同意A成为领导者。选举的具体过程这里不做详细描述，一般来说，哨兵选择的过程很快，谁先完成客观下线，一般就能成为领导者。

故障转移：选举出的领导者哨兵，开始进行故障转移操作，该操作大体可以分为3个步骤：

在从节点中选择新的主节点：选择的原则是，

1.首先过滤掉不健康的从节点；

2.然后选择优先级最高的从节点（由replica-priority指定）；如果优先级无法区分，

3.则选择复制偏移量最大的从节点；如果仍无法区分，

4.则选择runid最小的从节点。

更新主从状态：通过slaveof no one命令，让选出来的从节点成为主节点；并通过slaveof命令让其他节点成为其从节点。

将已经下线的主节点（即6379）保持关注，当6379从新上线后设置为新的主节点的从节点

8.实践建议

哨兵节点的数量应不止一个。一方面增加哨兵节点的冗余，避免哨兵本身成为高可用的瓶颈；另一方面减少对下线的误判。此外，这些不同的哨兵节点应部署在不同的物理机上。

哨兵节点的数量应该是奇数，便于哨兵通过投票做出"决策"：领导者选举的决策、客观下线的决策等。

各个哨兵节点的配置应一致，包括硬件、参数等；此外应保证时间准确、一致。

9.总结

在主从复制的基础上，哨兵引入了主节点的自动故障转移，进一步提高了Redis的高可用性；但是哨兵的缺陷同样很明显：哨兵无法对从节点进行自动故障转移，在读写分离场景下，从节点故障会导致读服务不可用，需要我们对从节点做额外的监控、切换操作。此外，哨兵仍然没有解决写操作无法负载均衡、及存储能力受到单机限制的问题

---

redis cluster高可用集群

1.redis cluster集群是什么?

redis cluster集群是一个由多个主从节点群组成的分布式服务器群，它具有复制、高可用和分片特 性。Redis cluster集群不需要sentinel哨兵也能完成节点移除和故障转移的功能。需要将每个节点 设置成集群模式，这种集群模式没有中心节点，可水平扩展，据官方文档称可以线性扩展到 1000节点。redis cluster集群的性能和高可用性均优于之前版本的哨兵模式，且集群配置非常简单

2.redis cluster集群搭建

1.原生搭建

1.配置开启cluster节点

cluster-enabled yes（启动集群模式）

cluster-config-file nodes-8001.conf（这里800x最好和port对应上）

2.meet

cluster meet ip port

3.指派槽

查看crc16 算法算出key的槽位命令 cluster keyslot key

16384/3 0-5461 5462-10922 10923-16383 16384/4 4096

cluster addslots slot（槽位下标）

4.分配主从

cluster replicate node-id

2.使用redis提供的rb脚本

redis cluster集群需要至少要三个master节点，我们这里搭建三个master节点，并且给每个 master再搭建一个slave节点，总共6个redis节点，由于节点数较多，这里采用在一台机器 上创建6个redis实例，并将这6个redis实例配置成集群模式，所以这里搭建的是伪集群模 式，当然真正的分布式集群的配置方法几乎一样，搭建伪集群的步骤如下： 第一步：在/usr/local下创建文件夹redis-cluster，然后在其下面分别创建6个文件夹如下 （1）mkdir -p /usr/local/redis-cluster （2）mkdir 8001、 mkdir 8002、 mkdir 8003、 mkdir 8004、 mkdir 8005、 mkdir 8006 第一步：把之前的redis.conf配置文件copy到8001下，修改如下内容： （1）daemonize yes （2）port 8001（分别对每个机器的端口号进行设置） （3）bind 127.0.0.1（如果只在本机玩则可以指定为127.0.0.1 如果需要外网访问则需要指定本机真实ip） 定可能会出现循环查找集群节点机器的情况） （4）dir /usr/local/redis-cluster/8001/（指定数据文件存放位置，必须要指定不同的目 录位置，不然会丢失数据） （5）cluster-enabled yes（启动集群模式） （6）cluster-config-file nodes-8001.conf（这里800x最好和port对应上） （7）cluster-node-timeout 5000 （8）appendonly yes 第三步：把修改后的配置文件，分别 copy到各个文夹下，注意每个文件要修改第2、4、6 项里的端口号，可以用批量替换：:%s/源字符串/目的字符串/g 第四步：由于 redis集群需要使用 ruby命令，所以我们需要安装 ruby（redis5.0之后省略） （1）yum install ruby （2）yum install rubygems （3）gem install redis --version 3.0.0（安装redis和 ruby的接口） 第五步：分别启动6个redis实例，然后检查是否启动成功 （1）/usr/local/redis/bin/redis-server /usr/local/redis-cluster/800*/redis.conf （2）ps -ef | grep redis 查看是否启动成功

第六步：在redis3的安装目录下执行 redis-trib.rb命令创建整个redis集群 （1）cd /usr/local/redis3/src （2）./redis-trib.rb create --replicas 1 127.0.0.1:9000 127.0.0.1:9001 127.0.0.1:9002 127.0.0.1:9003 127.0.0.1:9004 127.0.0.1:9005

redis5.0使用/usr/local/bin/redis-cli --cluster create 192.168.0.104:7000 192.168.0.104:7001 192.168.0.104:7002 192.168.0.104:7003 192.168.0.104 :7004 192.168.0.104:7005 --cluster-replicas 1

第七步：验证集群： （1）连接任意一个客户端即可：./redis-cli -c -h -p (-c表示集群模式，指定ip地址和端口 号）如：/usr/local/redis/bin/redis-cli -c -h 127.0.0.1 -p 800* （2）进行验证： cluster info（查看集群信息）、cluster nodes（查看节点列表） （3）进行数据操作验证 （4）关闭集群则需要逐个进行关闭，使用命令：/usr/local/redis/bin/redis-cli -c -h 127.0.0.1 -p 800* shutdown

3.集群伸缩

1.扩容集群

1.准备新节点

2.加入集群

使用redis-cli 语法：add-node 新节点ip 端口 已存在节点ip 端口

使用原生命令 语法：cluster meet ip port

指定主从

使用redis-cli 语法（加入时指定）：add-node 新节点ip 端口 已存在节点ip 端口 --cluster-slave --cluster-master-id masterID

使用原生命令 语法：cluster replicate node-id

3.迁移槽和数据

1.槽迁移计划

语法：/redis-cli --cluster reshard 已存在节点ip： 端口

/usr/local/bin/redis-cli --cluster reshard 192.168.0.104:7000

2.迁移数据

执行流程：提示要分配多少槽-》接收节点ID-》all/done

3.添加从节点

2.缩容集群

1.下线迁移槽

语法：redis-cli --cluster reshard --cluster-from 要迁出节点ID --cluster-to 接收槽节点ID --cluster-slots 迁出槽数量 已存在节点ip 端口

/usr/local/bin/redis-cli --cluster reshard --cluster-from a2fdd1359d03acacf2a6e558acbc006639445d53 --cluster-to 1794864d5f8af79e88cfc0f699f02b6341c78b5c --cluster-slots 1366 192.168.0.104 7000

2.忘记节点.关闭节点

语法： redis-cli --cluster del-node 已存在节点IP：端口 要删除的节点ID

/usr/local/bin/redis-cli --cluster del-node 192.168.0.104:7000 8de55e2a7419983184cede9daab5d36ee9da1fa3

4.cluster客户端

1.moved重定向：指我们发送命令时，会对发送的key进行crc16算法，得到一个数字，然而我们连接的客户端并不是管理这个数字的范围，所以会返回错误并告诉你此key应该对应的槽位，然后客户端需要捕获此异常，重新发起请求到对应的槽位

2.asx重定向：指在我们送发命令时，对应的客户端正在迁移槽位中，所以此时我们不能确定这个key是还在旧的节点中还是新的节点中

3.smart客户端

1.从集群中选取一个可运行节点，使用cluster slots初始化槽和节点映射。

2.将cluster slots的结果映射到本地，为每个节点创建jedispool

3.准备执行命令

5.故障转移（与哨兵相似）

1.故障发现： 通过ping/pong消息实现故障发现（不依赖sentinel）

2.故障恢复

1.检查资格

1.每个从节点检查与主节点的断开时间

超过cluster-node-timeout * cluster-replica-validity-factor 时间取消资格

2.选择偏移量最大的

替换主节点

1.当前从节点取消复制变为主节点（slaveof no one）

2.撤销以前主节点的槽位，给新的主节点

3.向集群广播消息，表明已经替换了故障节点

总结

redis集群演变过程

1.单机版

核心技术：持久化

持久化是最简单的高可用方法（有时甚至不被归为高可用的手段），主要作用是数据备份，即将数据存储在硬盘，保证数据不会因进程退出而丢失。

2.主从复制

复制是高可用Redis的基础，哨兵和集群都是在复制基础上实现高可用的。复制主要实现了数据的多机备份，以及对于读操作的负载均衡和简单的故障恢复。缺陷是故障恢复无法自动化；写操作无法负载均衡；存储能力受到单机的限制。

3.哨兵

在复制的基础上，哨兵实现了自动化的故障恢复。缺陷是写操作无法负载均衡；存储能力受到单机的限制。

4.集群

通过集群，Redis解决了写操作无法负载均衡，以及存储能力受到单机限制的问题，实现了较为完善的高可用方案