NSD NOSQL DAY03

1 案例1：配置redis主从复制

1.1 问题

- 具体要求如下：
- 将主机192.168.4.51作为主库
- 将主机192.168.4.52作为从库
- 测试配置

1.

1.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：配置redis主从复制

1）配置主从，4.51为主，4.52为从

若主机做过redis集群，需要在配置文件里面把开启集群，存储集群信息的配置文件都关闭，新主机则不用，这里用之前的redis集群做主从，需要还原redis服务器，4.51和4.52都需要还原（以4.51为例）

```
[root@redisA ~]# redis-cli -c -h 192.168.4.51 -p 6351 shutdown
//先关闭redis集群
[root@redisA ~]# vim /etc/redis/6379.conf
bind 192.168.4.51
port 6379
# cluster-enabled yes
# cluster-config-file nodes-6351.conf

[root@redisA ~]# /etc/init.d/redis_6379 start
Starting Redis server...

 [root@redisA ~]# ss -antlp | grep 6379
LISTEN     0      511    192.168.4.51:6379                  *:*                   users:(("redis-serv

[root@redisA ~]# redis-cli -h 192.168.4.51
192.168.4.51:6379> info replication       //查看主从配置信息
```

Top

```
# Replication
role:master          //默认是master 服务器
connected_slaves:0
master_replid:eaa14478158a71c41f947eaea036658c2087e8f2
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:0
second_repl_offset:-1
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
192.168.4.51:6379>
```

2）配置从库192.168.4.52/24

```
192.168.4.52:6379> SLAVEOF  192.168.4.51  6379        //把52配置为51的从库
OK
```

从库查看

```
192.168.4.52:6379> INFO replication
# Replication
role:slave
master_host:192.168.4.51       //主库为4.51
master_port:6379
master_link_status:up
master_last_io_seconds_ago:3
master_sync_in_progress:0
```

3）主库查看

```
[root@redisA ~]# redis-cli -h 192.168.4.51
192.168.4.51:6379> info replication
# Replication
role:master                                                    Top
connected_slaves:1
slave0:ip=192.168.4.52,port=6379,state=online,offset=14,lag=1   //从库为4.52
```

```
master_replid: db7932eb0ea4302bddbebd395efa174fb079319f
master_replid2: 0000000000000000000000000000000000000000
master_repl_offset: 14
second_repl_offset: -1
repl_backlog_active: 1
repl_backlog_size: 1048576
repl_backlog_first_byte_offset: 1
repl_backlog_histlen: 14
192.168.4.51:6379>
```

4）反客为主，主库宕机后，手动将从库设置为主库

```
[root@redisA ~]# redis-cli -h 192.168.4.51 shutdown          //关闭主库

192.168.4.52:6379> SLAVEOF no one      //手动设为主库
OK
192.168.4.52:6379> INFO replication
# Replication
role: master
connected_slaves: 0
master_replid: 00e35c62d2b673ec48d3c8c7d9c7ea3366eac33a
master_replid2: db7932eb0ea4302bddbebd395efa174fb079319f
master_repl_offset: 420
second_repl_offset: 421
repl_backlog_active: 1
repl_backlog_size: 1048576
repl_backlog_first_byte_offset: 1
repl_backlog_histlen: 420
192.168.4.52:6379>
```

5）哨兵模式

主库宕机后，从库自动升级为主库

在slave主机编辑sentinel.conf文件

在slave主机运行哨兵程序

```
[root@redisB ~]# redis-cli -h 192.168.4.52
192.168.4.52:6379> SLAVEOF 192.168.4.51 6379
OK
```

Top

```
192.168.4.52:6379> INFO replication
# Replication
role:slave
master_host:192.168.4.51
master_port:6379
...

[root@redisA ~]# /etc/init.d/redis_6379 start
Starting Redis server...

[root@redisA ~]# redis-cli -h 192.168.4.51
192.168.4.51:6379> info replication
# Replication
role:master
connected_slaves:1
slave0:ip=192.168.4.52,port=6379,state=online,offset=451,lag=1
master_replid:4dfa0877c740507ac7844f8dd996445d368d6d0f
master_replid2:0000000000000000000000000000000000000000
...


[root@redisB ~]# vim /etc/sentinel.conf
sentinel  monitor  redisA      192.168.4.51  6379  1
关键字     关键字   主机名自定义      ip          端口  票数
sentinel  auth-pass  redis51 密码  //连接主库密码，若主库有密码加上这一行
[root@redisB ~]# redis-sentinel /etc/sentinel.conf    //执行，之后把主库宕机
...
25371:X 28 Sep 11:16:54.993 # +sdown master redis51 192.168.4.51 6379
25371:X 28 Sep 11:16:54.993 # +odown master redis51 192.168.4.51 6379 #quorum 1/1
25371:X 28 Sep 11:16:54.993 # +new-epoch 3
25371:X 28 Sep 11:16:54.993 # +try-failover master redis51 192.168.4.51 6379
25371:X 28 Sep 11:16:54.994 # +vote-for-leader be035801d4d48eb63d8420a72796f52fc5
...
25371:X 28 Sep 11:16:55.287 * +slave slave 192.168.4.51:6379 192.168.4.51 6379 @ redis
25371:X 28 Sep 11:17:25.316 # +sdown slave 192.168.4.51:6379 192.168.4.51 6379 @ redi
```

6）配置带验证的主从复制

关闭4.51和4.52，启动之后用info replication查看，各自为主

主库设置密码，在51上面操作

Top

```
[root@redisA ~]# redis-cli -h 192.168.4.51 shutdown

[root@redisA ~]# vim /etc/redis/6379.conf
requirepass 123456

[root@redisA ~]# /etc/init.d/redis_6379 start
Starting Redis server...

[root@redisA ~]# redis-cli -h 192.168.4.51 -a 123456
192.168.4.51:6379> ping
PONG
192.168.4.51:6379>
```

7）配置从库主机

```
[root@redisB ~]# redis-cli -h 192.168.4.52 shutdown

[root@redisB ~]# vim /etc/redis/6352.conf
slaveof 192.168.4.51 6379
masterauth 123456

[root@redisB ~]# /etc/init.d/redis_6352 start
Starting Redis server...
```

52上面查看 52从主库变为从库

```
[root@redisB ~]# redis-cli -h 192.168.4.52 -a 123456
192.168.4.52:6379> info replication
# Replication
role:slave
master_host:192.168.4.51
master_port:6379
master_link_status:up
```

51上面查看 51的从库为52

Top

```
[root@redisA ~]# redis-cli -h 192.168.4.51 -a 123456
```

```
192.168.4.51:6379> info replication
# Replication
role:master
connected_slaves:1
slave0:ip=192.168.4.52,port=6379,state=online,offset=98,lag=0
```

2 案例2 ：使用RDB文件恢复数据

2.1 问题

- 要求如下：
- 启用RDB
- 设置存盘间隔为120秒 10个key改变存盘
- 备份RDB文件
- 删除数据
- 使用RDB文件恢复数据

1.
2.

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：使用RDB文件恢复数据

RDB介绍：

Redis数据库文件，全称Reids DataBase

数据持久化方式之一

在指定时间间隔内，将内存中的数据集快照写入硬盘

术语叫Snapshot快照

恢复时，将快照文件直接读到内存里

相关配置参数

文件名

dbfilename “dump.rdb” 文件名

save “” 禁用RDB

数据从内存保存到硬盘的频率

save 900 1 900秒内且有1次修改

save 300 10 300秒内且有10次修改

save 60 10000 60秒内且有10000修改

```
[root@redisA ~]# redis-cli -h 192.168.4.51 -a 123456 shutdown
[root@redisA ~]# vim /etc/redis/6379.conf
dbfilename dump.rdb
```

Top

```
04.  #  save ""       //启用RDB，去掉#号为禁用RDB
05.  save 120 10      //120秒内且有1次修改（满足三个条件中的任意一个都会保存）
06.  save 300 10
07.  save 60 10000
08.
09.  [root@redisA ~]# /etc/init.d/redis_6379 start
10.  Starting Redis server...
11.  [root@redisA ~]# redis-cli -h 192.168.4.51 -a 123456
12.  192.168.4.51:6379>
13.  [root@redisA ~]# redis-cli -h 192.168.4.51 -a 123456
14.  192.168.4.51:6379>
15.  192.168.4.51:6379> set v1 k1
16.  OK
17.  192.168.4.51:6379> set v2 k1
18.  OK
19.  192.168.4.51:6379> set v3 k1
20.  OK
21.  192.168.4.51:6379> set v4 k1
22.  OK
23.  192.168.4.51:6379> set v45 k1
24.  OK
25.  192.168.4.51:6379> set v46 k1
26.  OK
27.  192.168.4.51:6379> set v7 k1
28.  OK
29.  192.168.4.51:6379> set v8 k1
30.  OK
31.  192.168.4.51:6379> set v9 k1
32.  OK
33.  192.168.4.51:6379> set v10 k1
34.  OK
35.  192.168.4.51:6379> keys *
36.   1) "v2"
37.   2) "v9"
38.   3) "v10"
39.   4) "v45"
40.   5) "v4"
41.   6) "v1"
42.   7) "v46"
43.   8) "v8"
44.   9) "v7"
```

Top

```
45.  10) "v3"
46.  192.168.4.51:6379>exit
```

备份数据

```
01.  [root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456 shutdown       //停止服务
02.  [root@redisA ~]# cd /var/lib/redis/6379/
03.  [root@redisA 6379]# ls
04.  dump.rdb  nodes-6351.conf
05.  [root@redisA 6379]# cp dump.rdb dump.rdb.bak    //备份dump.rdb，之后删除
```

删除数据

```
01.  [root@redisA 6379]# rm -rf dump.rdb
02.  [root@redisA 6379]# /etc/init.d/redis_6379 start
03.  Starting Redis server...
04.  [root@redisA 6379]# ls
05.  dump.rdb  dump.rdb.bak  nodes-6351.conf
06.  [root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456
07.  192.168.4.51:6379> keys *       //已经没有数据
08.  (empty list or set)
09.  192.168.4.51:6379>
```

恢复数据

```
01.  [root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456 shutdown
02.  [root@redisA 6379]# mv dump.rdb.bak dump.rdb
03.  mv: overwrite ‘dump.rdb’? y
04.  [root@redisA 6379]# /etc/init.d/redis_6379 start
05.  Starting Redis server...
06.  [root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456
07.  192.168.4.51:6379> keys *
08.   1) "v7"
09.   2) "v46"
10.   3) "v45"
11.   4) "v8"
12.   5) "v4"
```

Top

```
13.      6) "v2"
14.      7) "v1"
15.      8) "v3"
16.      9) "v9"
17.     10) "v10"
18.    192.168.4.51:6379>
```

RDB优点：

高性能的持久化实现：创建一个子进程来执行持久化，先将数据写入临时文件，持久化过程结束后，再用这个临时文件替换上次持久化好的文件；过程中主进程不做任何IO操作

比较适合大规模数据恢复，且对数据完整性要求不是非常高的场合

RDB的缺点：

意外宕机时，最后一次持久化的数据会丢失

3 案例3 ：使用AOF文件恢复数据

3.1 问题

- 要求如下：
- 启用AOF
- 备份AOF文件
- 删除数据
- 使用AOF文件恢复数据

3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：使用AOF文件恢复数据

1）AOF介绍

只做追加操作的文件，Append Only File

记录redis服务所有写操作

不断的将新的写操作，追加到文件的末尾

使用cat命令可以查看文件内容

2）参数配置

文件名

appendfilename "appendonly.aof" 指定文件名

appendonly yes 启用aof ，默认no

AOF文件记录写操作的方式

appendfsync always 有新写操作立即记录

appendfsync everysec 每秒记录一次

appendfsync no 从不记录

Top

```
[root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456 shutdown
[root@redisA 6379]# rm -rf dump.rdb
[root@redisA 6379]# vim /etc/redis/6379.conf
appendonly yes              //启用aof ，默认no
appendfilename "appendonly.aof"    //文件名
appendfsync everysec              //每秒记录一次
[root@redisA 6379]# vim /etc/redis/6379.conf
[root@redisA 6379]# /etc/init.d/redis_6379 start
Starting Redis server...
[root@redisA 6379]# ls              //会出现appendonly.aof文件
appendonly.aof  dump.rdb  nodes-6351.conf

[root@redisA 6379]# cat appendonly.aof        //无内容
[root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456
192.168.4.51:6379> set v1 a1
OK
192.168.4.51:6379> set v2 a2
OK
192.168.4.51:6379> set v3 a3
OK
192.168.4.51:6379> set v4 a4
OK
192.168.4.51:6379> set v5 a5
OK
192.168.4.51:6379> set v6 a6
OK
192.168.4.51:6379> set v7 a7
OK
192.168.4.51:6379> set v8 a7
OK
192.168.4.51:6379> set v9 a9
OK
192.168.4.51:6379> set v10 a10
OK
192.168.4.51:6379> keys *
 1) "v2"
 2) "v5"
 3) "v10"
 4) "v9"
 5) "v6"
```

Top

```
41.     6) "v8"
42.     7) "v3"
43.     8) "v7"
44.     9) "v1"
45.     10) "v4"
46.     192.168.4.51:6379> exit
47.     [root@redisA 6379]# cat appendonly.aof          //有数据
```

3）使用AOF恢复数据

备份数据

```
01.     [root@redisA 6379]# cp appendonly.aof appendonly.aof.bak
02.     [root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456 shutdown
```

删除数据

```
01.     [root@redisA 6379]# rm -rf appendonly.aof
02.     [root@redisA 6379]# /etc/init.d/redis_6379 start
03.     Starting Redis server...
04.     [root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456
05.     192.168.4.51:6379> keys *
06.     (empty list or set)
07.     192.168.4.51:6379> exit
```

恢复数据

```
01.     [root@redisA 6379]# mv appendonly.aof.bak appendonly.aof
02.     mv: overwrite ‘appendonly.aof’? y
03.     [root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456 shutdown
04.     [root@redisA 6379]# /etc/init.d/redis_6379 start
05.     Starting Redis server...
06.     [root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456
07.     192.168.4.51:6379> keys *
08.     1) "v9"
09.     2) "v5"
10.     3) "v8"
11.     4) "v2"
```

Top

```
12.      5) "v1"
13.      6) "v4"
14.      7) "v10"
15.      8) "v6"
16.      9) "v7"
17.     10) "v3"
18.     192.168.4.51:6379>
```

修复AOF文件，把文件恢复到最后一次的正确操作

```
01.     [root@redisA 6379]#vim appendonly.aof
02.     *2      //可以把这一行删除
03.     $6
04.     SELECT
05.     $1
06.     0
07.     *3
08.     $3
09.     set
10.     $2
11.     v1
12.     $2
13.     a1
14.     *3
15.     $3
16.     ...
17.     [root@redisA 6379]#redis-check-aof --fix appendonly.aof       //恢复文件
18.     0x             0: Expected prefix '*', got: '$'
19.     AOF analyzed: size=311, ok_up_to=0, diff=311
20.     This will shrink the AOF from 311 bytes, with 311 bytes, to 0 bytes
21.     Continue? [y/N]: y
22.     Successfully truncated AOF
```

RDB优点:

可以灵活的设置同步持久化appendfsync always或异步持久化appendfsync verysec

宕机时，仅可能丢失1秒的数据

RDB的缺点:

Top

AOF文件的体积通常会大于RDB文件的体积

执行fsync策略时的速度可能会比RDB慢

4 案例4：Redis数据库常用操作

4.1 问题

- 对Redis数据库各数据类型进行增删改查操作
- 数据类型分别为strings、hash表、list列表
- 设置数据缓存时间
- 清空所有数据
- 对数据库操作

4.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：redis数据类型

1）String字符串

set key value [ex seconds] [px milliseconds] [nx|xx]

设置key及值，过期时间可以使用秒或毫秒为单位

setrange key offset value

从偏移量开始复写key的特定位的值

```
[root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456
192.168.4.51:6379> set first "hello world"
OK
192.168.4.51:6379> setrange first 6 "Redis"    //改写为hello Redis
(integer) 11
192.168.4.51:6379> get first
"hello Redis"
```

strlen key，统计字串长度

```
192.168.4.51:6379> strlen first
(integer) 11
```

append key value 存在则追加，不存在则创建key及value，返回key长度

```
192.168.4.51:6379> append myname jacob
(integer) 5
```

Top

setbit key offset value 对key所存储字串，设置或清除特定偏移量上的位(bit)，value值可以为1或0，offset为0~2^32之间，key不存在，则创建新key

```
192.168.4.51:6379> setbit bit 0 1        //设置bit第0位为1
(integer) 0
192.168.4.51:6379> setbit bit 1 0        //设置bit第1位为0
(integer) 0
```

bitcount key 统计字串中被设置为1的比特位数量

```
192.168.4.51:6379> setbit bits 0 1       //0001
(integer) 0
192.168.4.51:6379> setbit bits 3 1       //1001
(integer) 0
192.168.4.51:6379> bitcount bits         //结果为2
(integer) 2
```

记录网站用户上线频率，如用户A上线了多少天等类似的数据，如用户在某天上线，则使用setbit，以用户名为key，将网站上线日为offset，并在该offset上设置1，最后计算用户总上线次数时，使用bitcount用户名即可，这样即使网站运行10年，每个用户仅占用10*365比特位即456字节

```
192.168.4.51:6379> setbit peter 100 1    //网站上线100天用户登录了一次
(integer) 0
192.168.4.51:6379> setbit peter 105 1    //网站上线105天用户登录了一次
(integer) 0
192.168.4.51:6379> bitcount peter
(integer) 2
```

decr key 将key中的值减1，key不存在则先初始化为0，再减1

```
192.168.4.51:6379> set z 10
OK
192.168.4.51:6379> decr z
(integer) 9
192.168.4.51:6379> decr z
(integer) 8
```

Top

```

192.168.4.51:6379> decr bb
```

```
10.  (integer) -1
11.  192.168.4.51:6379> decr bb
12.  (integer) -2
```

decrby key decrement 将key中的值，减去decrement

```
01.  192.168.4.51:6379> set count 100
02.  OK
03.  192.168.4.51:6379> DECRBY cc 20   //定义每次减少20（步长）
04.  (integer) -20
05.  192.168.4.51:6379> DECRBY cc 20
06.  (integer) -40
```

get key 返回key存储的字符串值，若key不存在则返回nil，若key的值不是字串，则返回错误，get只能处理字串

```
01.  192.168.4.51:6379> get a
02.  (nil)
```

getrange key start end 返回字串值中的子字串，截取范围为start和end，负数偏移量表示从末尾开始计数，-1表示最后一个字符，-2表示倒数第二个字符

```
01.  192.168.4.51:6379> set x 123456789
02.  OK
03.  192.168.4.51:6379> getrange x -5 -1
04.  "56789"
05.  192.168.4.51:6379> getrange x 0 4
06.  "12345"
```

incr key 将key的值加1，如果key不存在，则初始为0后再加1，主要应用为计数器

```
01.  192.168.4.51:6379> set page 20
02.  OK
03.  192.168.4.51:6379> incr page
04.  (integer) 21
```

Top

incrby key increment 将key的值增加increment

```
192.168.4.51:6379> set x 10
OK
192.168.4.51:6379> incr x
(integer) 11
192.168.4.51:6379> incr x
(integer) 12
```

incrbyfloat key increment 为key中所储存的值加上浮点数增量 increment

```
192.168.4.51:6379> set num 16.1
OK
192.168.4.51:6379> incrbyfloat num 1.1
"17.2"
```

mset key value [key value ...] 设置多个key及值，空格分隔，具有原子性

```
192.168.4.51:6379> mset j 9 k 29
OK
```

mget key [key...] 获取一个或多个key的值，空格分隔，具有原子性

```
192.168.4.51:6379> mget j k
1) "9"
2) "29"
```

2）list列表

Redis的list是一个字符队列，先进后出，一个key可以有多个值

lpush key value [value...] 将一个或多个值value插入到列表key的表头，Key不存在，则创建key

```
192.168.4.51:6379> lpush list a b c      //list值依次为c b a
(integer) 3
```

Top

lrange key start stop 从开始位置读取key的值到stop结束

```
192.168.4.51:6379> lrange list 0 2      //从0位开始，读到2位为止
1) "c"
2) "b"
3) "a"
192.168.4.51:6379> lrange list 0 -1   //从开始读到结束为止
1) "c"
2) "b"
3) "a"
192.168.4.51:6379> lrange list 0 -2      //从开始读到倒数第2位值
1) "c"
2) "b"
```

lpop key 移除并返回列表头元素数据，key不存在则返回nil

```
192.168.4.51:6379> lpop list      //删除表头元素，可以多次执行
"c"
192.168.4.51:6379> LPOP list
"b"
```

llen key 返回列表key的长度

```
192.168.4.51:6379> llen list
(integer) 1
```

lindex key index 返回列表中第index个值

```
192.168.4.51:6379> lindex list 1
"c"
```

lset key index value 将key中index位置的值修改为value

```
192.168.4.51:6379> lpush list a b c d
(integer) 5
```

Top

```
03.  192.168.4.51:6379> lset list 3 test      //将list中第3个值修改为test
04.  OK
```

rpush key value [value...] 将value插入到key的末尾

```
01.  192.168.4.51:6379> rpush list3  a b c    //list3值为a b c
02.  (integer) 3
03.  192.168.4.51:6379> rpush list3 d    //末尾插入d
04.  (integer) 4
```

rpop key 删除并返回key末尾的值

```
01.  192.168.4.51:6379> RPOP list3
02.  "d"
```

3）hash表

hset key field value 将hash表中field值设置为value

```
01.  192.168.4.51:6379> hset site google 'www.g.cn'
02.  (integer) 1
03.  192.168.4.51:6379> hset site baidu 'www.baidu.com'
04.  (integer) 1
```

hget key filed 获取hash表中field的值

```
01.  192.168.4.51:6379> hget site google
02.  "www.g.cn"
```

hmset key field value [field value...] 同时给hash表中的多个field赋值

```
01.  192.168.4.51:6379> hmset site google www.g.cn  baidu www.baidu.com
02.  OK
```

Top

hmget key field [field...] 返回hash表中多个field的值

```
192.168.4.51:6379> hmget site google baidu
1) "www.g.cn"
2) "www.baidu.com"
```

hkeys key 返回hash表中所有field名称

```
192.168.4.51:6379> hmset site google www.g.cn baidu www.baidu.com
OK
192.168.4.51:6379> hkeys site
1) "google"
2) "baidu"
```

hgetall key 返回hash表中所有key名和对应的值列表

```
192.168.4.51:6379> hgetall site
1) "google"
2) "www.g.cn"
3) "baidu"
4) "www.baidu.com"
```

hvals key 返回hash表中所有key的值

```
192.168.4.51:6379> hvals site
1) "www.g.cn"
2) "www.baidu.com"
```

hdel key field [field...] 删除hash表中多个field的值，不存在则忽略

```
192.168.4.51:6379> hdel site google baidu
(integer) 2
```

Top