TTS 11.0 COOKBOOK

(NSD NOSQL DAY03)

版本编号 11.0

2019-06

达内 IT 培训集团

NSD NOSQL DAY03

1. 案例 1：redis 主从复制

- **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

- **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-server",pid=22274,fd=6))

[root@redisA ~]# redis-cli -h 192.168.4.51
192.168.4.51:6379> info replication        //查看主从配置信息
# 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
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
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
be035801d4d48eb63d8420a72796f52fc5cec047 3
...
25371:X 28 Sep 11:16:55.287 * +slave slave 192.168.4.51:6379 192.168.4.51 6379 @
redis51 192.168.4.52 6379
25371:X 28 Sep 11:17:25.316 # +sdown slave 192.168.4.51:6379 192.168.4.51 6379 @
redis51 192.168.4.52 6379
```

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

```
[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

```
[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

- **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

- **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
#   save ""       //启用 RDB，去掉#号为禁用 RDB
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）
save 300 10
save 60 10000

[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>
[root@redisA ~]# redis-cli -h 192.168.4.51 -a 123456
192.168.4.51:6379>
192.168.4.51:6379> set v1 k1
OK
192.168.4.51:6379> set v2 k1
OK
192.168.4.51:6379> set v3 k1
OK
192.168.4.51:6379> set v4 k1
OK
192.168.4.51:6379> set v45 k1
OK
192.168.4.51:6379> set v46 k1
OK
192.168.4.51:6379> set v7 k1
OK
192.168.4.51:6379> set v8 k1
OK
192.168.4.51:6379> set v9 k1
OK
```

```
192.168.4.51:6379> set v10 k1
OK
192.168.4.51:6379> keys *
 1) "v2"
 2) "v9"
 3) "v10"
 4) "v45"
 5) "v4"
 6) "v1"
 7) "v46"
 8) "v8"
 9) "v7"
10) "v3"
192.168.4.51:6379>exit
```

备份数据

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

删除数据

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

恢复数据

```
[root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456 shutdown
[root@redisA 6379]# mv dump.rdb.bak dump.rdb
mv: overwrite ‘dump.rdb’? y
[root@redisA 6379]# /etc/init.d/redis_6379  start
Starting Redis server...
[root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456
192.168.4.51:6379> keys *
 1) "v7"
 2) "v46"
 3) "v45"
 4) "v8"
 5) "v4"
 6) "v2"
 7) "v1"
 8) "v3"
 9) "v9"
10) "v10"
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

- **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
- **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

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

3）使用 AOF 恢复数据

备份数据

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

删除数据

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

恢复数据

```
[root@redisA 6379]# mv appendonly.aof.bak appendonly.aof
mv: overwrite ‘appendonly.aof’? y
[root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456 shutdown
[root@redisA 6379]# /etc/init.d/redis_6379  start
Starting Redis server...
[root@redisA 6379]# redis-cli -h 192.168.4.51 -a 123456
```

```
192.168.4.51:6379> keys *
 1) "v9"
 2) "v5"
 3) "v8"
 4) "v2"
 5) "v1"
 6) "v4"
 7) "v10"
 8) "v6"
 9) "v7"
10) "v3"
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

```
[root@redisA 6379]# vim appendonly.aof
*2        //可以把这一行删除
$6
SELECT
$1
0
*3
$3
set
$2
v1
$2
a1
*3
$3
...
[root@redisA 6379]# redis-check-aof --fix appendonly.aof        //恢复文件
0x              0: Expected prefix '*', got: '$'
AOF analyzed: size=311, ok_up_to=0, diff=311
This will shrink the AOF from 311 bytes, with 311 bytes, to 0 bytes
Continue? [y/N]: y
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：

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

- **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

- **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
```

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


192.168.4.51:6379> decr bb
(integer) -1
192.168.4.51:6379> decr bb
(integer) -2
```

decrby key decrement 将 key 中的值，减去 decrement

```
192.168.4.51:6379> set count 100
OK
192.168.4.51:6379> DECRBY cc 20    //定义每次减少 20（步长）
(integer) -20
192.168.4.51:6379> DECRBY cc 20
(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

```
192.168.4.51:6379> get a
(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

```
192.168.4.51:6379> set x 123456789
OK
192.168.4.51:6379> getrange x -5 -1
"56789"
192.168.4.51:6379> getrange x 0 4
"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

```
192.168.4.51:6379> set page 20
OK
192.168.4.51:6379> incr page
(integer) 21
```

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
```

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

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

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

rpop key　　删除并返回 key 末尾的值

```
192.168.4.51:6379> RPOP list3
"d"
```

3) hash 表

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

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

hget key filed　　获取 hash 表中 field 的值

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

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

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

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
```