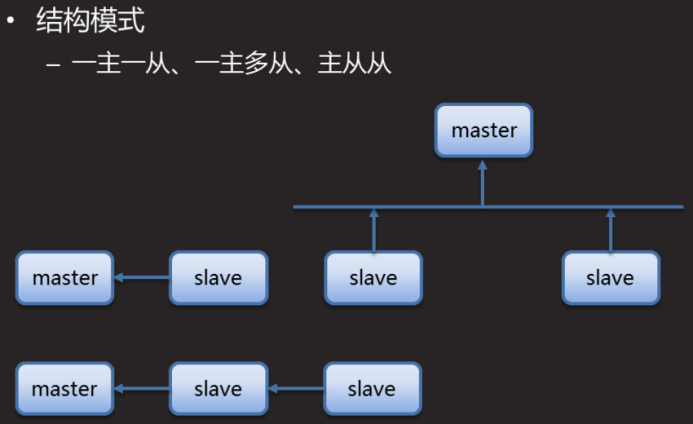
**11\_nosql03主从复制+持久化+数据类型**

**一 主从复制**

**1.1 主从复制结构模式**



**1.2 主从复制工作原理**

slave向master发送sync命令

master启动后台存盘进程,并收集所有修改数据命令

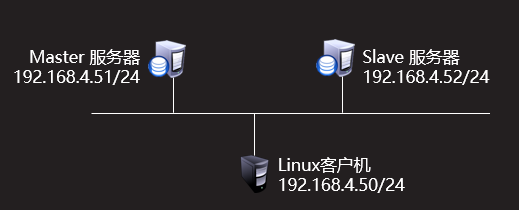
master完成后台存盘后,传送整个数据文件到slave

slave接收数据文件,加载到内存中,完成首次完全同步

后续有新数据产生时,master继续收集数据修改命令,依次传给salve,完成同步

**二 配置主从复制 命令:slaveof 主服务器ip 端口**

**2.1 拓扑结构**



**2.2 配置从库-一主一从,配置51为主,52为从**

**redis服务运行后,默认都是master服务器**

info replication #查看复制信息

#Replication

role: 角色

master\_host: 主库ip地址

master\_port: 主库端口号

master\_link\_status: 与主库连接状态

**2.2.1 51 52 登录redis,查看复制信息**

redis51 ~]# redis-cli -h 192.168.4.51 -p 6351

192.168.4.51:6351> info replication

默认配置role都为master

**2.2.2 命令行临时配置52为51的从,马上生效**

**slaveof 主服务器ip地址 端口**

redis52 ~]# redis-cli -h 192.168.4.52 -p 6352

192.168.4.52:6352> slaveof 192.168.4.51 6351

OK

192.168.4.52:6352> info replication

# Replication

role:slave

master\_host:192.168.4.51 #再次查看本机复制信息,52已成为51的从

master\_port:6351

...

**slave\_read\_only:1 #从服务器默认只读**

**2.2.3 修改配置文件永久配置52为51的从,重启生效**

redis52 ~]# vim /etc/redis/6379.conf

**282 # slaveof <masterip> <masterport> #解除注释该行,并修改**

282 slaveof 192.168.4.51 6351 #slaveof 主服务器ip 端口

**2.2.4 52重启redis服务**

redis52 ~]# /etc/init.d/redis\_6379 stop

redis52 ~]# /etc/init.d/redis\_6379 start

**2.2.5 52登录redis,查看本机复制信息**

redis52 ~]# redis-cli -h 192.168.4.52 -p 6352

role:slave

**2.3 反客为主**

**2.3.1 命令行临时配置将从库恢复为主库,马上生效**

192.168.4.52:6352> **slave no one**

**2.3.2 修改配置文件永久配置将从库恢复为主库,重启生效**

vim /etc/redis/6379.conf

282 #slaveof 192.168.4.51 6351 #注释该行

**2.4 配置51为主,52 53为从的一主多从结构**

按步骤2.2修改53的配置文件并重启即可

**2.5 配置51 53 54 主从从结构**

**2.5.1 临时配置54为53从**

192.168.4.54:6354> slaveof 192.168.4.53 6353

# Replication

role:slave

master\_host:192.168.4.53

master\_port:6353

**2.5.2 查看53的复制信息,仍然是51的从,并且有1个slave**

# Replication

role:slave

master\_host:192.168.4.51

slave0:ip=192.168.4.54,port=6354,state=online,offset=2236,lag=1

**2.6 配置带验证的主从复制**

**2.6.1 修改master配置文件,设置连接密码,重启服务并登录**

redis51 ~]# vim /etc/redis/6379.conf

501 # requirepass foobared #解除该行注释并修改

501 requirepass 123456 #定义连接密码

redis51 ~]# /etc/init.d/redis\_6379 stop

redis51 ~]# /etc/init.d/redis\_6379 start

redis51 ~]# redis-cli -h 192.168.4.51 -p 6351 -a 123456

**2.6.2 修该slave配置文件,永久设置连接密码,重启服务并登录**

redis52 ~]# vim +289 /etc/redis/6379.conf

289 masterauth 123456 #解除该行注释并修改

redis52 ~]# /etc/init.d/redis\_6379 stop

redis52 ~]# /etc/init.d/redis\_6379 start

redis52 ~]# redis-cli -h 192.168.4.52 -p 6352 -a 123456

**2.6.3 命令行配置slave,临时设置连接密码**

192.168.4.53:6353> **config set masterauth** 123456

**2.6.4 命令行配置slave,永久设置连接密码**

192.168.4.53:6353> **config rewrite**

vim /etc/redis/6379.conf,最下面一行 masterauth "123456"

**2.6.5 命令行给53永久设置主服务器连接密码**

192.168.4.53:6353> **config set requirepass** 123546

192.168.4.53:6353> **config rewrite**

提示需要密码时,输入: auth 密码

**2.6.6 给54设置从服务器连接密码**

192.168.4.54:6354> **config set masterauth** 123456

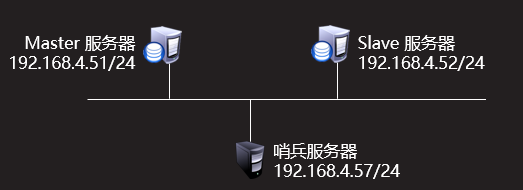
192.168.4.54:6354> **config rewrite**

**2.7 哨兵服务 sentinel[ˈsentɪnl]**

哨兵服务可部署在redis**从**服务器上,最好单独部署

监视master服务器,发现master宕机后,将从服务器升级为主服务器

主配置文件 sentinel.conf 模板文件:redis-4.0.8/sentinel.conf



在57上配置哨兵服务,监视51-53-54的主-从(主)-从结构[关闭52从服务功能]

192.168.4.52:6352> slaveof no one

**2.7.1 配置哨兵服务-创建主配置文件**

redis57 ~]# cp redis-4.0.8/sentinel.conf /etc/sentinel.conf

redis57 ~]# vim /etc/sentinel.conf

52 # sentinel monitor <master-name> <ip> <redis-port> <quorum>

52 sentinel monitor redis51 192.168.4.51 6351 1 #监视的主服务器

<quorum>赋值为1表示有1台哨兵服务器监控到51down了就执行53由从转主

15 # bind 127.0.0.1 192.168.1.1

15 bind 0.0.0.0 #表示本机所有网络接口用于哨兵服务

71 # sentinel auth-pass <master-name> <password>

71 sentinel auth-pass redis51 123456 #监视的主服务器redis连接密码

21 port 26379 #默认

**2.7.2 配置哨兵服务-启动服务**

redis57 ~]# redis-sentinel /etc/sentinel.conf #启动占用一个终端

**2.7.3 配置哨兵服务-测试**

**2.7.3.1 停止主服务器51的redis服务**

redis51 ~]# /etc/init.d/redis\_6379 stop

**2.7.3.2 57哨兵给出提示信息后,在服务器53查看复制信息**

redis53 ~]# redis-cli -h 192.168.4.53 -p 6353 -a 123456

192.168.4.53:6353> info replication

# Replication

role:master #此时53已经成为了master

connected\_slaves:1

slave0:ip=192.168.4.54,port=6354,state=online,offset=43348,lag=0

**2.7.3.3 57另开终端查看哨兵配置文件,监控的主服务器ip地址已变**

redis57 ~]# vim /etc/sentinel.conf

sentinel monitor redis51 192.168.4.53 6353 1

**2.7.3.4** 重启51的redis服务,查看复制信息,此时51为53的从,状态为down,手动设置51连接53的连接密码,设置好后51自动同步宕机期间的数据

redis51 ~]# /etc/init.d/redis\_6379 start #重启redis服务

redis51 ~]# redis-cli -h 192.168.4.51 -p 6351 -a 123456 #登录

192.168.4.51:6351> info replication #查看复制信息

# Replication

role:slave

master\_host:192.168.4.53 #此时51为53从

master\_port:6353

master\_link\_status:down #状态为down

192.168.4.51:6351> config set masterauth 123456

#配置与53的连接密码

192.168.4.51:6351> info replication #再次查看复制信息

# Replication

role:slave

master\_host:192.168.4.53 #此时51为53从

master\_port:6353

master\_link\_status:up #状态为up

192.168.4.51:6351> keys \* #自动同步了宕机期间的数据

192.168.4.51:6351> config rewrite #永久配置与53的连接密码

**三 持久化**

**3.1 RDB** 默认[/var/lib/redis/6379/dump.rdb文件]

**3.1.1 RDB介绍**

redis数据库文件,全称**R**edis **D**ata**B**ase

数据持久化方式之一

数据持久化默认方式

按照指定时间间隔,将内存中的数据集快照写入硬盘

快照术语叫snapshot

恢复时,将快照文件直接读入内存

定义RDB文件名

dbfilename “dump.rdb”

**3.1.2 使用RDB恢复数据**

**3.1.2.1 备份数据**

备份dump.rdb文件到其他位置

cp 数据库目录/dump.rdb 备份目录/

redis50 ~]# cp /var/lib/redis/6379/dump.rdb /root

redis50 ~]# scp /root/dump.rdb root@192.168.4.56:/root

**2.1.2.2 恢复数据(将51的数据在56上恢复)**

拷贝备份文件到数据库目录,重启redis服务

cp 备份目录/dump.rdb 数据库目录/

redis56 ~]# /etc/init.d/redis\_6379 stop #关闭redis服务

redis56 ~]# rm -rf /var/lib/redis/6379/\* #清空数据库目录

redis56 ~]# mv dump.rdb /var/lib/redis/6379/ #移动备份文件

redis56 ~]# /etc/init.d/redis\_6379 start #启动redis服务

redis56 ~]# redis-cli -h 192.168.4.56 -p 6356 #登录查看

**3.1.3 RDB优化设置**

redis50 ~]# vim /etc/redis/6379.conf

219 save 900 1 15分钟且有1个key改变

220 save 300 10 5分钟且有10个key改变

221 save 60 10000 1分钟且有10000个key改变

手动存盘

save 阻塞写存盘,存盘时客户端不能对数据进行读写

bgsave 不阻塞写存盘,存盘时单独生成一个进程去存盘,不影响客户端读写

**3.1.4 RDB优点与缺点**

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传持久化好的文件;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很高

缺点:

意外宕机时,丢失最后一次持久化的所有数据

**3.2 AOF A**ppend **O**nly **F**ile

**3.2.1 AOF介绍**

追加方式记录写操作的文件

记录redis服务所有写操作

不断地将新的写操作,追加到文件的末尾

默认没有启用

使用cat命令可以查看文件内容

**启用AOF**

config set appendonly yes #在redis命令行临时启用

config rewrite #将临时设置写入配置文件,实现永久配置

或

修改配置文件

redis50 ~]# vim /etc/redis/6379.conf

673 appendnoly yes #将no修改为yes

重启redis服务后,在数据库目录下生成appendonly.aof文件,此时此文件为空;重启时优先加载appendonly.aof文件,而不是优先加载dump.rdb文件.从而造成清空数据.

方法1适合旧的正在运行的redis服务器,方法2适合新架设好的redis服务器

**3.2.2 使用AOF文件恢复数据**

**备份数据**

cp 数据库目录/appendonly.aof 备份目录

**恢复数据**

停止当前redis服务

清空数据库目录数据,修改配置文件开启AOF

拷贝备份文件到数据库目录: cp 备份目录/appendonly.aof 数据库目录/

重启redis服务: /etc/init.d/redis\_6379 start

**3.2.3 优化设置**

**定义文件名**

appendonly yes #启用aof,默认no

appendfilename “appendonly.aof” #指定文件名

**AOF文件记录写操作的方式**

appendfsync always 实时记录,并完成磁盘同步

appendfsync everysec 每秒记录一次,并完成磁盘同步

appendfsync no 写入aof文件,不执行磁盘同步

**日志文件会不断增大,何时触发日志重写?**

auto-aof-rewrite-min-size 64mb #首次重写触发值

auto-aof-rewrite-percentage 100

#再次重写,增长百分比,以重写后的文件大小为基数

redis56 ~]# grep -i "auto-aof" /etc/redis/6379.conf

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

**修复AOF文件**:把文件恢复到最后一次的正确操作

**redis-check-aof --fix appendonly.aof** #此命令不是万能的

redis50 ~]# redis-check-aof --fix

/var/lib/redis/6379/appendonly.aof

出现提示时输入y

Successfully truncated AOF

**3.2.4 AOF优点与缺点**

优点:

可以灵活设置持久化方式

出现意外宕机时,仅可能丢失1秒的数据

缺点:

持久化方式的文件体积通常会大于RBD方式的文件体积

执行fsync策略时的速度可能会比RBD方式慢

**四 数据类型**

**type key 查看数据类型**

**4.1 string字符串**

字符串操作

**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;nx表示key不存在则赋值,xx表示key存在则赋值,默认为xx

**mset key value [key value ...]**

设置多个key及值,空格分隔,具有原子性

**setrange key offset value** 从偏移量开始修改key的特定位的值

set first “hello world”

setrange first 6 “Redis” #修改后的结果为hello Redis

**strlen key** 统计字符串长度

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

**setbit key offset value**

对key所存储字符串,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

**bitcount key** 计算key里面1个个数 [key需为二进制]

**decr key** 将key中的值减1,key不存在则先初始化为0,再减1

**incr key** 将key中的值加1,key不存在则先初始化为0,再加1,主要用于计数器

**decrby key decrement** 将key中的值,每次减去decrement

**incrby key increment** 将key中的值,每次增加increment

**incrbyfloat key increment** 将key中的值,每次增加浮点类型的decrement

**get key** 返回key存储的字符串值,若key不存在则返回null

若key的值不是字符串,则返回错误,get只能处理字符串

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

**getrange key start end**

返回字符串值中的子字符串,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

**4.2 list列表**

**4.2.1 list列表简介**

redis的list时一个字符队列,先进后出,一个key可以有多个值

**4.2.2 list列表操作**

**lpush key value [value ...]**

将一个或多个value插入到列表key的表头,key不存在则创建并赋值

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

lrange key 0 2 从0位开始,读到2位为止

lrange key 0 -1 从开始读到结束为止

lrange key 0 -2 从开始读到倒数第2位为止

**lpop key** 移除并返回列表头元素数据,key不存在则返回nil

**llen key** 返回列表key的长度

**lindex key index** 返回列表key中第index个值

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

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

**rpop key** 删除并返回key末尾的值

**4.3 hash表**

**4.3.1 hash表简介**

redis hash 是一个string类型的field和value的映射表

一个key可对应多个field,一个field对应一个value

将一个对象存储为hash类型,比将每个字段都存储成string类型更节省内存

**4.3.2 hash表操作**

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

**hget key field** 获取hash表中field的值

**hmset key filed value [filed value ...]**

同时给hash表中的多个field赋值

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

**hkeys key** 返回hash表中所有field名称

**hgetall key** 返回hash表中所有field的值及对应的value

**hvals key** 返回hash表中所有field对应的value值

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