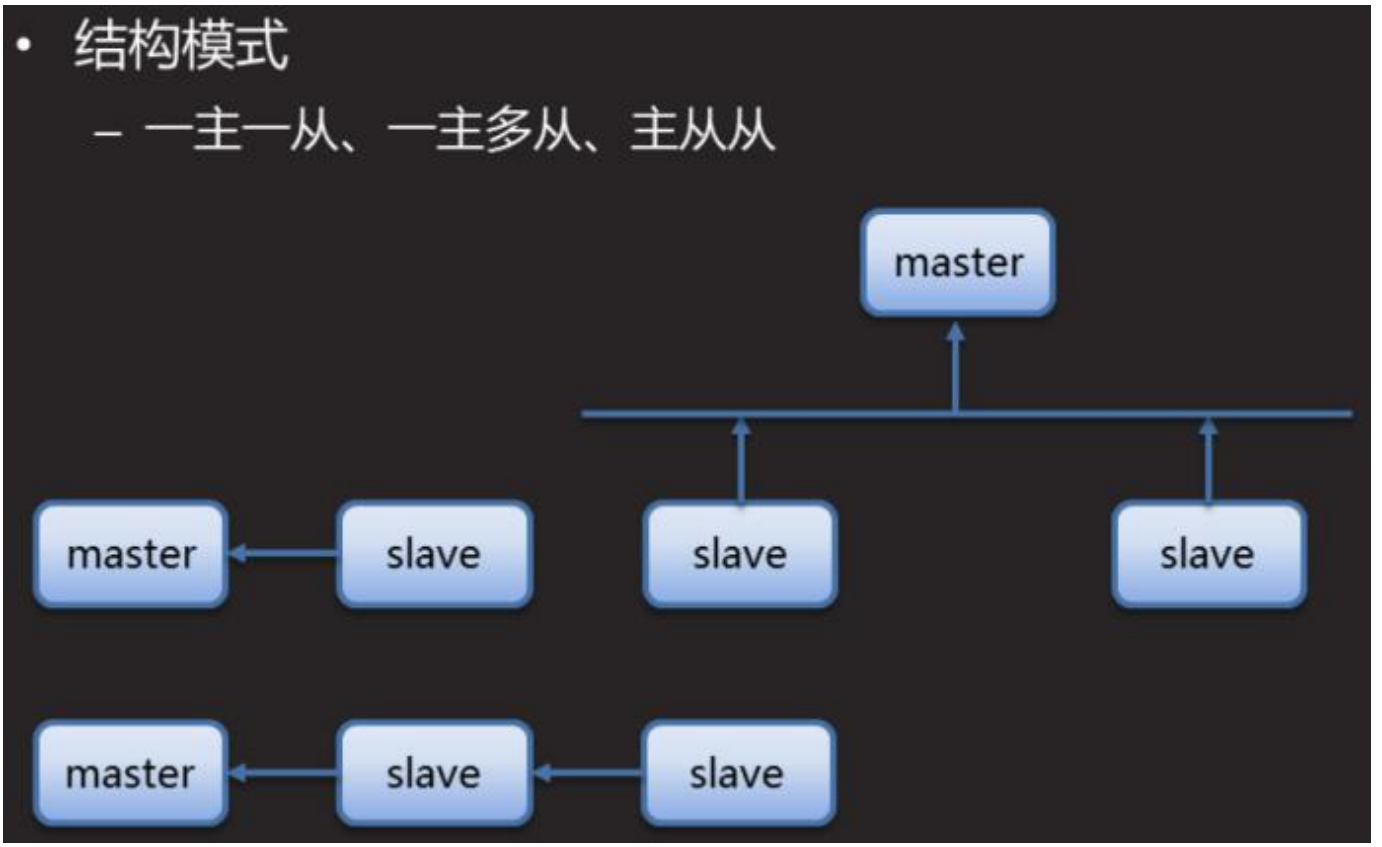
11_nosql03 主从复制+持久化+数据类型

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

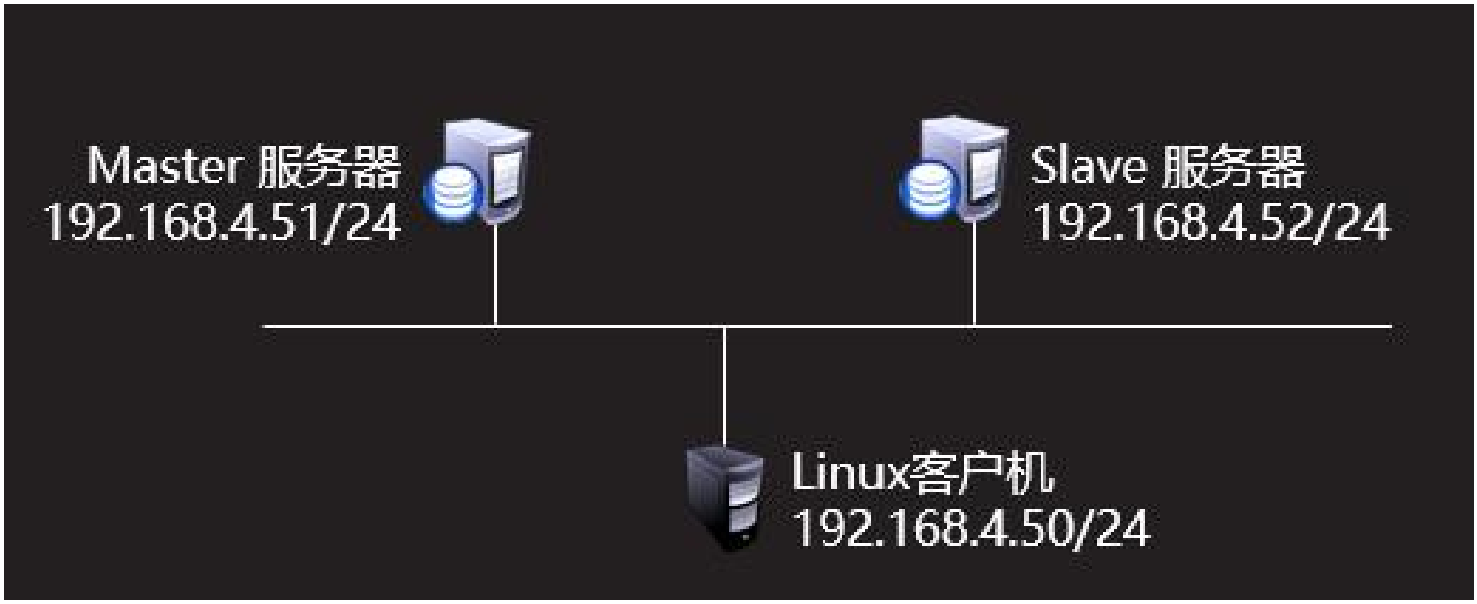
master 完成后台存盘后,传送整个数据文件到 slave

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['sentɪnl]

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 Append Only File

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