# NOSQL01------------------------

# 一redis介绍

RDBMS关系数据库系统管理

* Relational database management system
* 按照预先设置的组织结构.将数据存储在物理介质上
* 数据之间可以做关联操作

主流的RDBMS服务软件

* oracle
* db2
* ms sqlserver
* Mysql mariadb

Nosql (not only sql)

* 意思是”不仅仅是sql”
* 泛指非关系型数据库
* 不需要预先定义数据存储结构
* 表的每条记录都可以有不同类型和结构

主流的Nosql服务软件

* Redis
* MongoDB
* Memcached
* CouchDB
* Neo4j
* FlockDB

Redis简介

* Remote Dictionary Server(远程字典服务器)
* 是一款高性能的(Key/values)分布式内存数据库
* 支持数据持久化,可以把内存里的数据保存到硬盘中
* 也支持list hash set zset 数据类型
* 支持master-salve模式数据备份
* 中文网站www.redis.cn

Redis是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型、高性能的Key-Value数据库，并提供多种语言的API(应用程序接口)。

Redis支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。

# 二:搭建redis服务器

## 2.1装包

[root@client-50 ~]# rpm -q gcc || yum -y install gcc

[root@client-50 ~]# cd redis-4.0.8/

[root@client-50 redis-4.0.8]# make

[root@client-50 redis-4.0.8]# make install

## 2.2初始化配置

[root@client-50 redis-4.0.8]# cd utils #提供了初始化脚本在此目录

[root@client-50 utils]# ./install\_server.sh #运行初始化脚本

Welcome to the redis service installer

This script will help you easily set up a running redis server

Please select the redis port for this instance: [6379]

Selecting default: 6379

Please select the redis config file name [/etc/redis/6379.conf]

Selected default - /etc/redis/6379.conf

Please select the redis log file name [/var/log/redis\_6379.log]

Selected default - /var/log/redis\_6379.log

Please select the data directory for this instance [/var/lib/redis/6379]

Selected default - /var/lib/redis/6379

Please select the redis executable path [/usr/local/bin/redis-server]

Selected config:

Port : 6379

Config file : /etc/redis/6379.conf #配置文件

Log file : /var/log/redis\_6379.log

Data dir : /var/lib/redis/6379 #数据库目录

Executable : /usr/local/bin/redis-server

Cli Executable : /usr/local/bin/redis-cli #命令工具目录

Is this ok? Then press ENTER to go on or Ctrl-C to abort. #使用以上配置回车确认, 不使用则ctrl+c结束

Copied /tmp/6379.conf => /etc/init.d/redis\_6379

Installing service...

Successfully added to chkconfig!

Successfully added to runlevels 345!

Starting Redis server...

Installation successful!

## 2.3查看服务状态信息

[root@client-50 utils]# netstat -unltp | grep :6379

tcp 0 0 127.0.0.1:6379 0.0.0.0:\* LISTEN 10041/redis-server

[root@client-50 utils]# ps -C redis-server

PID TTY TIME CMD

10041 ? 00:00:00 redis-server

## 2.4访问redis服务存取数据

[root@client-50 utils]# redis-cli

127.0.0.1:6379> set name bob

OK

127.0.0.1:6379> get name

"bob"

127.0.0.1:6379> exit

## 2.5服务管理

### 停止服务

使用提供的脚本

[root@client-50 utils]# /etc/init.d/redis\_6379 stop

Stopping ...

Waiting for Redis to shutdown ...

Redis stopped

[root@client-50 utils]# netstat -unltp | grep :6379

[root@client-50 utils]# redis-cli

Could not connect to Redis at 127.0.0.1:6379: Connection refused

Could not connect to Redis at 127.0.0.1:6379: Connection refused

not connected> exit

### 启动服务

使用脚本

[root@client-50 utils]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@client-50 utils]# netstat -unltp | grep :6379

tcp 0 0 127.0.0.1:6379 0.0.0.0:\* LISTEN 10162/redis-server

[root@client-50 utils]# redis-cli

### 设置服务连接密码

(设置了密码后,停止服务是需要输入密码)

[root@client-50 utils]# /etc/init.d/redis\_6379 stop

Stopping ...

Waiting for Redis to shutdown ...

Redis stopped

[root@client-50 utils]# vim /etc/redis/6379.conf

501 requirepass 123456 #修改密码为123456

[root@client-50 utils]# /etc/init.d/redis\_6379 start

[root@client-50 utils]# redis-cli

127.0.0.1:6379> set x 99 #设置密码后,登录不输入密码则无法做任务操作

(error) NOAUTH Authentication required.

127.0.0.1:6379> auth 123456 #输入密码方式一:先登录再输入密码

OK

[root@client-50 utils]# redis-cli -a 123456 #输入密码方式二:登录时输入密码

127.0.0.1:6379> set y 100

OK

127.0.0.1:6379> get y

"100"

### 设置密码后停止服务

[root@client-50 utils]# /etc/init.d/redis\_6379 stop

Stopping ...

(error) NOAUTH Authentication required. #报错

Waiting for Redis to shutdown ... #无法停止

#### 停止服务方法一:

使用命令停止有密码的redis服务

[root@client-50 utils]# redis-cli -a 123456 shutdown

[root@client-50 utils]# netstat -unltp | grep :6379

[root@client-50 utils]#

#### 停止服务方法二:

修改脚本

[root@client-50 utils]# vim /etc/init.d/redis\_6379

43 $CLIEXEC -p $REDISPORT -a 123456 shutdown

[root@client-50 utils]# /etc/init.d/redis\_6379 stop

Stopping ...

Waiting for Redis to shutdown ...

Redis stopped

[root@client-50 utils]# netstat -unltp | grep :6379

## 2.6数据管理

### 常用操作指令集合

Set keyname keyvalue 定义数据给变量

get keyname 输出获取变量数据

Select 数据库编号0-15 切换库

Keys \* 查看所有变量

exists keyname 检查变量是否存在,再次赋值会覆盖

ttl keyname 查看生存时间

type keyname 查看类型

move keyname dbname 移动变量

expire keyname 10 设置有效时间 --(10s)

del keyname 删除变量

flushall 删除所有变量(慎用,内存内数据全部清除)

flushdb 仅清除当前库的数据

save 保存所有变量(正常默认是5分钟自动保存到硬盘,执行后立即保存)

Shutdown 关闭redis 服务 #停止服务时,自动将数据存入dump.rdb文件中

### 常用操作指令实操

**Set keyname keyvalue 定义数据给变量**

127.0.0.1:6379> set y 100 #赋值给变量

OK

**get keyname 输出获取变量数据**

127.0.0.1:6379> get y #输出变量

"100"

**切换库Select 数据库编号0-15**

127.0.0.1:6379> select 1

OK

127.0.0.1:6379[1]> select 2

OK

127.0.0.1:6379[2]> select 0

OK

127.0.0.1:6379>

**Keys \* 查看所有变量**

127.0.0.1:6379> keys \*

1) "y"

2) "name"

127.0.0.1:6379> select 1

OK

127.0.0.1:6379[1]> keys \*

(empty list or set)

**Keys a? 查看已有变量名, ?代表任意一个字符**

127.0.0.1:6379> set a1 200

OK

127.0.0.1:6379> set a2 300

OK

127.0.0.1:6379> keys a?

1) "a1"

2) "a2"

**exists keyname 检查变量是否存在,再次赋值会覆盖**

127.0.0.1:6379> exists a1

(integer) 1

127.0.0.1:6379> exists a2

(integer) 1

127.0.0.1:6379> exists a3

(integer) 0

**ttl keyname 查看生存时间**

127.0.0.1:6379> get name

"bob"

127.0.0.1:6379> ttl name

(integer) -1 # 为-1永不过期

**expire keyname 10 设置有效时间**

127.0.0.1:6379> set a1 200

OK

127.0.0.1:6379> get a1

"200"

127.0.0.1:6379> expire a1 6

(integer) 1

127.0.0.1:6379> get a1

"200"

127.0.0.1:6379> get a1

(nil)

127.0.0.1:6379> set a1 200

OK

127.0.0.1:6379> expire a1 20

(integer) 1

127.0.0.1:6379> ttl a1

(integer) 17

127.0.0.1:6379> ttl a1

(integer) 7

127.0.0.1:6379> ttl a1

(integer) 2

127.0.0.1:6379> ttl a1

(integer) 1

127.0.0.1:6379> ttl a1

(integer) 0

127.0.0.1:6379> ttl a1

(integer) -2

127.0.0.1:6379> get a1

(nil)

**type keyname 查看类型**

Set 定义的变量都是字符串类型

127.0.0.1:6379> set a1 200

OK

127.0.0.1:6379> type a1

string

**move keyname dbname 移动变量从一个库移动到另一个库**

127.0.0.1:6379> keys \*

1) "a"

2) "c"

3) "b"

127.0.0.1:6379> move a 1

(integer) 1

127.0.0.1:6379> keys \*

1) "c"

2) "b"

127.0.0.1:6379> select 1

OK

127.0.0.1:6379[1]> keys \*

1) "a"

**del keyname 删除变量**

127.0.0.1:6379> keys \*

1) "a1"

2) "a2"

3) "y"

4) "name"

127.0.0.1:6379> del a1 a2 y

(integer) 3

127.0.0.1:6379> keys \*

1) "name"

**flushall 删除所有变量(慎用,内存内数据全部清除)**

127.0.0.1:6379[1]> keys \*

1) "a"

127.0.0.1:6379[1]> flushall

OK

127.0.0.1:6379[1]> keys \*

(empty list or set)

127.0.0.1:6379[1]> select 0

OK

127.0.0.1:6379> keys \*

(empty list or set)

**flushdb 仅清除当前库的数据**

127.0.0.1:6379[1]> keys \*

1) "b"

127.0.0.1:6379[1]> flushdb

OK

127.0.0.1:6379[1]> keys \*

(empty list or set)

127.0.0.1:6379[1]> select 0

OK

127.0.0.1:6379> keys \*

1) "a"

sav**e 保存所有变量**(正常默认是5分钟自动保存到硬盘,执行后立即保存)

[root@client-50 utils]# ls /var/lib/redis/6379/

dump.rdb #内存数据保存在此文件里面

127.0.0.1:6379> save

OK

Shutdown 关闭redis 服务 #停止服务时,自动将数据存入dump.rdb文件中

127.0.0.1:6379> shutdown

not connected>

not connected> exit

[root@client-50 utils]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@client-50 utils]# redis-cli -a 123456

127.0.0.1:6379>

# 三 配置文件解析:

## 数据单位

1k=1000 bytes

1kb=1024bytes

1M=1000000bytes(100万)

1MB=1024\*1024bytes

1g=1000000000bytes(10亿)

1gb=1024\*1024\*1024bytes

## nn常用配置选项:

Port 6379 端口

bind 127.0.0.1 IP地址

tcp-backlog 511 tcp连接总数(同时接受tcp连接为511个)

timeout 0 连接超时时间(默认为0:没有超时时间)

tcp-keepalive 300 长连接时间(在这个时间内一次连接可多次请求)

daemonize yes 守护进程方式运行(服务启动后,进程一直在内存运行,等待客户连

databases 16 数据库个数(切换数据库个数0开始)

logfile /var/log/redis\_6379.log pid文件,日志文件

maxclient 10000 最大并发连接数量,

dir /var/lib/redis/6379 数据库目录

## 内存管理

### 内存清除策略

搜索lru 在565-572

volatile-lru 删除最近最少使用(针对设置了TTL(变量有效期)的key)

allkeys-lru 删除最少使用的key

volatile-random 在设置了TTL的key里随机移除

allkeys-random 随机移除key

volatile-ttl (minor TTL) 移除最近过期的key

noeviction 不删除,写满时报错(默认方式)

102400 东塘现金充值机

### 内存选项默认设置

maxmemory <bytes> 最大内存大小,

maxmemory-policy noeviction 定义内存清除策略

maxmemory-samples 5 选取模板数据的个数(针对内存清除策略含lru和ttl的策略)

模板:就是比对筛选条件模板.越接近模板,就是要被删除的内存变量

### 修改配置后登录

修改了端口为:6351

修改了IP为:192.168.4.51

修改了密码为:123456

[root@sql51 ~]# redis-cli #此命令默认连接”127.0.0.1:6379”

Could not connect to Redis at 127.0.0.1:6379: Connection refused

[root@sql51 ~]# redis-cli -h 192.168.4.51 -p 6351 -a 123456

192.168.4.51:6351>

### 修改配置后停止服务

修改配置后应立即修改脚本,才能使用脚本停止和启动redis

[root@sql51 ~]# vim /etc/init.d/redis\_6379

43 $CLIEXEC -h 192.168.4.51 -p 6351 -a 123456 shutdown

[root@sql51 ~]# /etc/init.d/redis\_6379 stop

[root@sql51 ~]# /etc/init.d/redis\_6379 start

方法一:

[root@sql51 ~]# /etc/init.d/redis\_6379 stop #调用脚本停止

Stopping ...

Could not connect to Redis at 127.0.0.1:6379: Connection refused

Waiting for Redis to shutdown ...

Waiting for Redis to shutdown ...

此命令调用脚本停止的是”127.0.0.1:6379” ,已修改了IP与端口,所以相应脚本也需修改

[root@sql51 ~]# vim /etc/init.d/redis\_6379

43 $CLIEXEC -h 192.168.4.51 -p 6351 -a 123456 shutdown

[root@sql51 ~]# /etc/init.d/redis\_6379 stop

Stopping ...

Redis stopped

方法二:

[root@sql51 ~]# /etc/init.d/redis\_6379 start #因为脚本已修改好,可直接启动

[root@sql51 ~]# redis-cli -h 192.168.4.51 -p 6351 -a 123456 shutdown

# 四 部署LNMP+Redis

## 1 51搭建Redis服务器

装包

初始化配置

修改配置文件.指定端口 服务ip地址 连接密码

重启服务

查看服务信息

## 2-50上装LNMP

给50准备安装包

[root@room9pc01 ~]# scp '/root/桌面/20190201/丁明一/丁明一OPERATION/LNMP.tar.gz' 192.168.4.50:/root

安装

[root@client-50 LNMP]# rpm -q pcre-devel zlib-devel openssl-devel

未安装软件包 pcre-devel

未安装软件包 zlib-devel

未安装软件包 openssl-devel

本虚拟环境中可能需要安装这些依赖,实际中不一定需要

[root@client-50 LNMP]# yum -y install pcre-devel zlib-devel openssl-devel

[root@client-50 LNMP]# tar -zxvf nginx-1.12.2.tar.gz

[root@client-50 LNMP]# cd nginx-1.12.2/

[root@client-50 LNMP]# cd nginx-1.12.2/

[root@client-50 nginx-1.12.2]#

[root@client-50 nginx-1.12.2]# ./configure --prefix=/usr/local/nginx

[root@client-50 nginx-1.12.2]#make

[root@client-50 nginx-1.12.2]# make install

[root@client-50 nginx-1.12.2]# ls /usr/local/nginx

conf html logs sbin

[root@client-50 LNMP]# ls

lnmp\_soft.tar.gz nginx-1.10.3.tar.gz nginx-1.12.2 nginx-1.12.2.tar.gz nginx.sh

[root@client-50 LNMP]# tar -xf lnmp\_soft.tar.gz

[root@client-50 LNMP]# ls

lnmp\_soft lnmp\_soft.tar.gz nginx-1.10.3.tar.gz nginx-1.12.2 nginx-1.12.2.tar.gz nginx.sh

### 配置nginx

[root@client-50 conf]# vim nginx.conf

65 location ~ \.php$ {

66 root html;

67 fastcgi\_pass 127.0.0.1:9000; #接受php结尾请求给9000端口

68 fastcgi\_index index.php;

69 # fastcgi\_param SCRIPT\_FILENAME /scripts$fastcgi\_script\_name;

70 include fastcgi.conf; #修改成fastcgi.conf

71 }

[root@client-50 conf]# /usr/local/nginx/sbin/nginx -t #检测nginx配置文件

nginx: the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok

nginx: configuration file /usr/local/nginx/conf/nginx.conf test is successful

#此状态证明nginx配置文件无错

[root@client-50 conf]# /usr/local/nginx/sbin/nginx #启动nginx

[root@client-50 conf]# netstat -unltp |grep 80

tcp 0 0 0.0.0.0:80 0.0.0.0:\* LISTEN 5665/nginx: master

**安装 php-fpm-5.4.16-42.el7.x86\_64.rpm,解释php脚本**

[root@client-50 ~]# ls

apache-tomcat-8.0.30.tar.gz php-fpm-5.4.16-42.el7.x86\_64.rpm

[root@client-50 lnmp\_soft]# rpm -ivh php-fpm-5.4.16-42.el7.x86\_64.rpm

警告：php-fpm-5.4.16-42.el7.x86\_64.rpm: 头V3 RSA/SHA256 Signature, 密钥 ID f4a80eb5: NOKEY

错误：依赖检测失败：

php-common(x86-64) = 5.4.16-42.el7 被 php-fpm-5.4.16-42.el7.x86\_64 需要

**rpm -ivh安装失败换yum安装自动解决依赖**

[root@client-50 ~]# yum -y install php php-devel

[root@client-50 ~]# yum -y install php-fpm-5.4.16-42.el7.x86\_64.rpm

[root@client-50 ~]# systemctl start php-fpm **#启动php**

[root@client-50 ~]# systemctl enable php-fpm

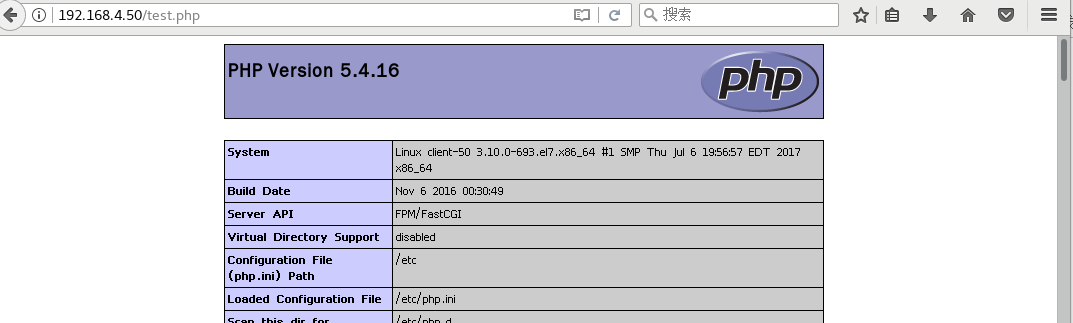
**编写php脚本后可测试**

[root@client-50 conf]# vim /usr/local/nginx/html/test.php

<?php

phpinfo();

?>



### 安装php

[root@nginx utils]# which php

/usr/bin/which: no php in (/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin)

#which指令会在环境变量$PATH设置的目录里查找符合条件的文件。

[root@nginx utils]# php -m #查看php命令

bash: php: command not found... #未发现php命令

[root@nginx utils]# yum -y install php-cli #安装php

[root@nginx utils]# which php

/usr/bin/php

[root@nginx utils]# php -m

[PHP Modules]

bz2

Calendar

Core

ctype

curl

........

[root@client-50 html]# ls /etc/php.ini

/etc/php.ini

### 安装mysql

之前实验已经安装,

## 3 配置php支持redis

### 安装提供redis模块的软件包

[root@client-50 ~]# php -m |grep redis #查看php是否支持redis

[root@client-50 ~]# cd php/

[root@client-50 php]#tar -xf php-redis-2.2.4.tar.gz #装提供redis模块的软件包

[root@client-50 php]# ls

php-devel-5.4.16-42.el7.x86\_64.rpm phpredis-2.2.4 php-redis-2.2.4.tar.gz

[root@client-50 php]# cd phpredis-2.2.4/

[root@client-50 phpredis-2.2.4]# which phpize

/usr/bin/phpize

[root@client-50 phpredis-2.2.4]# phpize

Can't find PHP headers in /usr/include/php

The php-devel package is required for use of this command.

[root@client-50 phpredis-2.2.4]# yum -y install autoconf automake

[root@client-50 phpredis-2.2.4]# yum -y install pcre-devel

[root@client-50 phpredis-2.2.4]# cd ..

root@client-50 php]# yum -y install php php-devel

[root@client-50 php]# yum -y install php-devel-5.4.16-42.el7.x86\_64.rpm

[root@client-50 php]# cd phpredis-2.2.4/

[root@client-50 phpredis-2.2.4]# phpize #生成一个php文件

Configuring for:

PHP Api Version: 20100412

Zend Module Api No: 20100525

Zend Extension Api No: 220100525

[root@client-50 phpredis-2.2.4]# find / -name "php-config"

/usr/bin/php-config

[root@client-50 phpredis-2.2.4]# ls /usr/bin/php-config #安装php而来

/usr/bin/php-config

**开始编译安装**

配置php程序运行时,加载redis模块

[root@client-50 php]# ls

php-devel-5.4.16-42.el7.x86\_64.rpm phpredis-2.2.4 php-redis-2.2.4.tar.gz

[root@client-50 php]# cd phpredis-2.2.4/

[root@client-50 phpredis-2.2.4]# ./configure --with-php-config=/usr/bin/php-config #指定模块编译的路径

[root@client-50 phpredis-2.2.4]# make && make install

Installing shared extensions: /usr/lib64/php/modules/

...

Installing shared extensions: /usr/lib64/php/modules/ #模块文件存放的路径

[root@client-50 phpredis-2.2.4]# ls /usr/lib64/php/modules/ #查看模块存在路径

curl.so fileinfo.so json.so phar.so redis.so zip.so

[root@client-50 phpredis-2.2.4]#vim /etc/php.ini

-----------修改前----------------

728 ; extension\_dir = "./"

729 ; On windows:

730 ; extension\_dir = "ext" #红色为修改地方

-----------修改后---------------

728 extension\_dir = "/usr/lib64/php/modules/" #去掉注释

729 ; On windows:

730 extension = "redis.so" #指定模块文件

[root@client-50 phpredis-2.2.4]# php -m |grep redis

redis

编写连接redis服务的php脚本,存储数据 查询数据

[root@client-50 html]# vim linkredis.php #创建新的php文件

<?php

$redis = new redis();

$redis->connect('192.168.4.51',6351);

$redis ->auth("123456");

$redis->set("school","tarena");

echo $redis->get("redistest");

?>

[root@client-50 phpredis-2.2.4]# php /usr/local/nginx/html/linkredis.php

Tarena

在客户端254(真机) 打开浏览器访问50主机的网站服务

在redis服务器51上查看数据是否存储

NSD NOSQL DAY01

案例1：搭建Redis服务器

案例2：修改Redis服务运行参数

案例3：部署LNMP+Redis

1 案例1：搭建Redis服务器

1.1 问题

具体要求如下：

在主机 192.168.4.51 上安装并启用 redis 服务

设置变量test，值为123

查看变量test的值

1.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：搭建redis服务器

1）安装redis服务器

[root@redis1 ~]# cd redis

[root@redis1 redis]# ls

lnmp redis-4.0.8.tar.gz

[root@redis1 redis]# yum -y install gcc gcc-c++ make

[root@redis1 redis]# tar -zxf redis-4.0.8.tar.gz

[root@redis1 redis]# cd redis-4.0.8/

[root@redis1 redis-4.0.8]# ls

00-RELEASENOTES CONTRIBUTING deps Makefile README.md runtest runtest-sentinel src utils

BUGS COPYING INSTALL MANIFESTO redis.conf runtest-cluster sentinel.conf tests

[root@redis1 redis-4.0.8]# make

[root@redis1 redis-4.0.8]# make install

[root@redis1 redis-4.0.8]# cd utils/

[root@redis1 utils]# ./install\_server.sh

Welcome to the redis service installer

This script will help you easily set up a running redis server

Please select the redis port for this instance: [6379]

Selecting default: 6379

Please select the redis config file name [/etc/redis/6379.conf]

Selected default - /etc/redis/6379.conf

Please select the redis log file name [/var/log/redis\_6379.log]

Selected default - /var/log/redis\_6379.log

Please select the data directory for this instance [/var/lib/redis/6379]

Selected default - /var/lib/redis/6379

Please select the redis executable path [/usr/local/bin/redis-server]

Selected config:

Port : 6379 //端口号

Config file : /etc/redis/6379.conf //配置文件目录

Log file : /var/log/redis\_6379.log //日志目录

Data dir : /var/lib/redis/6379 //数据库目录

Executable : /usr/local/bin/redis-server //启动程序的目录

Cli Executable : /usr/local/bin/redis-cli //命令行的连接工具

Is this ok? Then press ENTER to go on or Ctrl-C to abort. //回车完成配置

Copied /tmp/6379.conf => /etc/init.d/redis\_6379 //服务启动脚本

Installing service...

Successfully added to chkconfig!

Successfully added to runlevels 345!

Starting Redis server...

Installation successful! //安装成功

2）查看状态

[root@redis1 utils]# /etc/init.d/redis\_6379 status

Redis is running (15203)

3）查看监听的端口

[root@redis1 utils]# netstat -antupl |grep :6379

tcp 0 0 127.0.0.1:6379 0.0.0.0:\* LISTEN 15203/redis-server

[root@redis1 utils]# ps -C redis-server

PID TTY TIME CMD

15203 ? 00:00:00 redis-server

4）停止服务

[root@redis1 utils]# /etc/init.d/redis\_6379 stop

Stopping ...

Waiting for Redis to shutdown ...

Redis stopped

[root@redis1 utils]# /etc/init.d/redis\_6379 status

//再次查看，显示 没有那个文件或目录

cat: /var/run/redis\_6379.pid: No such file or directory

Redis is running ()

5）连接redis

[root@redis1 utils]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@redis1 utils]# redis-cli

127.0.0.1:6379> ping

PONG //PONG说明服务正常

6）设置变量test，值为123，查看变量test的值

常用指令操作：

set keyname keyvalue 存储

get keyname 获取

127.0.0.1:6379> set test 123

OK

127.0.0.1:6379> get test

"123"

del keyname 删除变量

127.0.0.1:6379> set k1 v1

OK

127.0.0.1:6379> get k1

"v1"

127.0.0.1:6379> del k1

(integer) 1

keys \* 打印所有变量

127.0.0.1:6379> keys \*

1) "test"

EXISTS keyname 测试是否存在

127.0.0.1:6379> exists k1

(integer) 0

type keyname 查看类型

127.0.0.1:6379> set k2 v1

OK

127.0.0.1:6379> type k2

string

move keyname dbname 移动变量

127.0.0.1:6379> move k2 1 //移动k2到1库

(integer) 1

select 数据库编号0-15 切换库

127.0.0.1:6379> select 1 //切换到1库

OK

127.0.0.1:6379[1]> keys \* //查看有k2

1) "k2"

expire keyname 10 设置有效时间

127.0.0.1:6379[1]> EXPIRE k2 10

(integer) 1

ttl keyname 查看生存时间

127.0.0.1:6379[1]> ttl k2

flushall 删除所有变量

127.0.0.1:6379[1]> FLUSHALL

OK

save 保存所有变量

127.0.0.1:6379[1]> save

OK

shutdown 关闭redis服务

127.0.0.1:6379[1]> SHUTDOWN

2 案例2：修改Redis服务运行参数

2.1 问题

具体要求如下：

端口号 6351

IP地址 192.168.4.51

连接密码 123456

客户端连接Redis服务

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：修改redis运行参数

1）

[root@redis1 utils]# cp /etc/redis/6379.conf /root/6379.conf

//可以先备份一份，防止修改错误没法还原

[root@redis1 utils]# /etc/init.d/redis\_6379 stop

[root@redis1 utils]# vim /etc/redis/6379.conf

...

bind 192.168.4.51 //设置服务使用的ip

port 6351 //更改端口号

requirepass 123456 //设置密码

[root@redis1 utils]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@redis1 utils]# ss -antul | grep 6351 //查看有端口6351

tcp LISTEN 0 128 192.168.4.51:6351 \*:\*

由于修改了配置文件所以在连接的时候需要加上ip和端口

[root@redis1 utils]# redis-cli -h 192.168.4.51 -p 6351

192.168.4.51:6351> ping

(error) NOAUTH Authentication required.

192.168.4.51:6351> auth 123456 //输入密码才能操作（因为之前设置过密码）

OK

192.168.4.51:6351> ping

PONG

还可以直接在命令行输入密码连接

[root@redis1 utils]# redis-cli -h 192.168.4.51 -p 6351 -a 123456

192.168.4.51:6351> ping

PONG

2）停止服务

由于修改Redis服务运行参数，所以在停止服务的时候也不能用默认的方法停止

[root@redis1 utils]# /etc/init.d/redis\_6379 stop //停止失败

Stopping ...

Could not connect to Redis at 127.0.0.1:6379: Connection refused

Waiting for Redis to shutdown ...

Waiting for Redis to shutdown ...

Waiting for Redis to shutdown ...

Waiting for Redis to shutdown ...

...

[root@redis1 utils]# redis-cli -h 192.168.4.51 -p 6351 -a 123456 shutdown

//停止成功

[root@redis1 utils]# ss -antul | grep 6351 //查看没有端口

3 案例3：部署LNMP+Redis

3.1 问题

具体要求如下：

在主机 192.168.4.52 上部署LNMP 环境

把数据存储到本机的redis服务中

3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：部署LNMP+Redis

1）安装redis，（不会搭建的请参考案例1）

2）安装php支持的功能模块（52上面操作）

[root@nginx utils]# which php

/usr/bin/which: no php in (/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin)

[root@nginx utils]# php -m

bash: php: command not found...

[root@nginx utils]# yum -y install php-cli

[root@nginx utils]# which php

/usr/bin/php

[root@nginx utils]# php -m

[PHP Modules]

bz2

calendar

Core

ctype

curl

date

ereg

exif

fileinfo

filter

ftp

gettext

gmp

hash

iconv

json

libxml

mhash

openssl

pcntl

pcre

Phar

readline

Reflection

session

shmop

SimpleXML

sockets

SPL

standard

tokenizer

xml

zip

zlib

[Zend Modules]

3）安装连接redis的功能模块

[root@nginx utils]# php -m | grep -i redis //没有redis模块

[root@nginx redis]# cd lnmp/

[root@nginx lnmp]# ls

nginx-1.12.2.tar.gz

php-devel-5.4.16-42.el7.x86\_64.rpm

php-fpm-5.4.16-42.el7.x86\_64.rpm

php-redis-2.2.4.tar.gz

[root@nginx lnmp]# tar -zxf php-redis-2.2.4.tar.gz

[root@nginx lnmp]# cd phpredis-2.2.4/

[root@nginx phpredis-2.2.4]# which phpize

/usr/bin/phpize

[root@nginx phpredis-2.2.4]# phpize

Can't find PHP headers in /usr/include/php

The php-devel package is required for use of this command.

[root@nginx phpredis-2.2.4]# yum -y install autoconf automake pcre-devel

[root@nginx phpredis-2.2.4]# cd ..

[root@nginx lnmp]# rpm -ivh php-devel-5.4.16-42.el7.x86\_64.rpm

[root@nginx lnmp]# cd phpredis-2.2.4/

[root@nginx phpredis-2.2.4]# phpize //生成一个php的文件

Configuring for:

PHP Api Version: 20100412

Zend Module Api No: 20100525

Zend Extension Api No: 220100525

[root@nginx phpredis-2.2.4]# find / -name "php-config"

/usr/bin/php-config

[root@nginx phpredis-2.2.4]# ./configure --with-php-config=/usr/bin/php-config

//指定模块编译的路径

[root@nginx phpredis-2.2.4]# make && make install

...

Installing shared extensions: /usr/lib64/php/modules/ //模块文件存放的路径

[root@nginx phpredis-2.2.4]# ls /usr/lib64/php/modules/

curl.so fileinfo.so json.so phar.so redis.so zip.so

[root@nginx phpredis-2.2.4]# vim /etc/php.ini

728 extension\_dir = "/usr/lib64/php/modules/"

729 ; On windows:

730 extension = "redis.so"

[root@nginx phpredis-2.2.4]# php -m | grep -i redis

redis //出现redis

4）安装nginx（52上面操作）

[root@nginx ~]# cd redis/lnmp/

[root@nginx lnmp]# ls

nginx-1.12.2.tar.gz

[root@nginx lnmp]# tar -xf nginx-1.12.2.tar.gz

[root@nginx lnmp]# cd nginx-1.12.2/

[root@nginx nginx-1.12.2]# yum -y install gcc pcre-devel openssl-devel

[root@nginx nginx-1.12.2]# useradd -s /sbin/nologin nginx

[root@nginx nginx-1.12.2]# ./configure --user=nginx --group=nginx --with-http\_ssl\_module

[root@nginx nginx-1.12.2]# make && make install

[root@nginx nginx-1.12.2]# ln -s /usr/local/nginx/sbin/nginx /sbin/

[root@nginx nginx-1.12.2]# cd /usr/local/nginx/html/

[root@nginx html]# echo "aa" > text.html

[root@nginx html]# yum -y install mariadb mariadb-server mariadb-devel php php-mysql

[root@nginx html]# cd /root/redis/lnmp/

[root@nginx lnmp]# rpm -ivh php-fpm-5.4.16-42.el7.x86\_64.rpm //安装php

[root@nginx lnmp]# cd /usr/local/nginx/html/

[root@nginx html]# vim test.php

<?php

$i=33;

$j=44;

if($i<$j){

echo "oK";

}

else{

echo "error";

}

#echo $i;

?>

[root@nginx html]# php test.php //在命令行测试

oK

[root@nginx html]# systemctl restart mariadb

[root@nginx html]# systemctl restart php-fpm

[root@nginx html]# vim /usr/local/nginx/conf/nginx.conf

...

location ~ \.php$ {

root html;

fastcgi\_pass 127.0.0.1:9000;

fastcgi\_index index.php;

#fastcgi\_param SCRIPT\_FILENAME /scripts$fastcgi\_script\_name;

include fastcgi.conf;

}

...

[root@nginx html]# nginx //启动nginx

客户端用火狐浏览器访问：

[root@room9pc01 ~]# firefox 192.168.4.56/text.html //成功

[root@room9pc01 ~]# firefox 192.168.4.56/test.php //成功

5）连接redis测试

[root@nginx html]# vim lkredis.php

<?php

$redis = new redis();

$redis->connect('192.168.4.51',6351);

$redis ->auth("123456");

$redis->set("redistest","666666");

echo $redis->get("redistest");

?>

[root@nginx html]# php lkredis.php //命令行测试

666666

火狐浏览器测试，如图-1所示：

图-1

在51上面查看，有数据存入

[root@redis1 lnmp]# redis-cli -h 192.168.4.51 -p 6351 -a 123456

192.168.4.51:6351> ping

PONG

192.168.4.51:6351> keys \*

1) "redistest"

192.168.4.51:6351> get redistest

"666666"

192.168.4.51:6351>

# NOSQL02--------------------

# 一 redis集群

## 准备环境-拓扑结构

客户端:192.168.4.50

sql51 RedisA 192.168.4.51 6351 同时兼任:管理主机

sql52 RedisB 192.168.4.52 6352

sql53 RedisC 192.168.4.53 6353

sql54 RedisD 192.168.4.54 6354

sql55 RedisE 192.168.4.55 6355

sql56 RedisF 192.168.4.56 6356

/etc/init.d/redis\_6379 #redis启动脚本

/etc/redis/6379.conf #redis配置文件

192.168.4.51:6351>flashall #清空所有数据

192.168.4.51:6351>save #保存

# 二 创建redis集群

## 2.1修改redis配置文件

**(以51为例,52-55同样)**

[root@sql51 ~]# /etc/init.d/redis\_6379 stop

[root@sql51 ~]# ls /var/lib/redis/6379/

Dump.rdb #redis数据文件,要删掉,不删掉无法创建集群

[root@sql51 ~]# rm -rf /var/lib/redis/6379/\*

[root@sql51 ~]# vim /etc/redis/6379.conf

70 bind 192.168.4.51 #不用回环地址

93 port 6351 #修改端口

137 daemonize yes #守护进程方式运行

501 #requirepass 123456 #501 注释掉密码

815 cluster-enabled yes #815 行开启集群

823 cluster-config-file nodes-6351.conf #指定集群信息文件

829 cluster-node-timeout 5000 #829 当前主机与其他主机通讯超时时间5s

**修改redis脚本文件**

[root@sql51 ~]# vim /etc/init.d/redis\_6379

43 $CLIEXEC -h 192.168.4.51 -p 6351 shutdown

[root@sql51 ~]# /etc/init.d/redis\_6379 start #检测启动正常

Starting Redis server...

[root@sql51 ~]# netstat -nultp | grep redis

tcp 0 0 192.168.4.51:6351 0.0.0.0:\* LISTEN 5249/redis-server 1

tcp 0 0 192.168.4.51:16351 0.0.0.0:\* LISTEN 5249/redis-server 1

[root@sql51 ~]# ls /var/lib/redis/6379/

dump.rdb nodes-6351.conf

[root@sql52 ~]# /etc/init.d/redis\_6379 stop #检测关闭正常

Stopping ...

Waiting for Redis to shutdown ...

Redis stopped

[root@sql52 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@sql52 ~]# vim /etc/init.d/redis\_6379

[root@sql52 ~]# redis-cli -h 192.168.4.52 -p 6352 #登录正常

192.168.4.52:6352> keys \* #无任何数据

(empty list or set)

192.168.4.52:6352>

## 2.2检测集群

192.168.4.51:6351> cluster info #查看集群状态

cluster\_state:fail #集群状态失败

cluster\_slots\_assigned:0

cluster\_slots\_ok:0

cluster\_slots\_pfail:0

cluster\_slots\_fail:0

cluster\_known\_nodes:1

cluster\_size:0

cluster\_current\_epoch:0

cluster\_my\_epoch:0

cluster\_stats\_messages\_sent:0

cluster\_stats\_messages\_received:0

192.168.4.51:6351> cluster nodes #查看集群节点

412a79829c7d0eed812b24959ca123b58990d60e :6351@16351 myself,master - 0 0 0 connected

## 2.3部署管理主机

sql51 RedisA 192.168.4.51 6351 同时兼任:管理主机

传安装包:

[root@room9pc01 ~]# scp -r '/root/桌面/redis-cluster' 192.168.4.51:/root

root@sql51 ~]# cd redis-cluster/

[root@sql51 redis-cluster]# ls

redis-3.2.1.gem ruby-devel-2.0.0.648-30.el7.x86\_64.rpm

[root@sql51 redis-cluster]# yum -y install ruby rubygems

#脚本文件是ruby语言编写的,所以要安装解释器

[root@sql51 redis-cluster]# yum -y install ruby-devel-2.0.0.648-30.el7.x86\_64.rpm

[root@sql51 redis-cluster]# gem install redis-3.2.1.gem

创建命令工具

[root@sql51 ~]# cd redis-4.0.8

[root@sql51 redis-4.0.8]# ls

00-RELEASENOTES CONTRIBUTING deps Makefile README.md runtest runtest-sentinel src utils

BUGS COPYING INSTALL MANIFESTO redis.conf runtest-cluster sentinel.conf tests

[root@sql51 redis-4.0.8]# cd src/

[root@sql51 src]# echo $PATH

/root/perl5/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin

[root@sql51 src]# mkdir /root/bin

[root@sql51 src]# cp re

redisassert.h redis-check-aof redis-check-rdb.c redis-cli.o redis-trib.rb replication.c

redis-benchmark redis-check-aof.c redis-check-rdb.o redismodule.h release.c replication.o

redis-benchmark.c redis-check-aof.o redis-cli redis-sentinel release.h

redis-benchmark.o redis-check-rdb redis-cli.c redis-server release.o

[root@sql51 src]# cp redis-trib.rb /root/bin/

[root@sql51 src]# chmod +x /root/bin/redis-trib.rb

[root@sql51 src]# redis-trib.rb help

Usage: redis-trib <command> <options> <arguments ...>

create host1:port1 ... hostN:portN

--replicas <arg>

check host:port

.... .......

集群必须至少要有三台主库,

--replicate 1 定义每个主库有1个从库,3台主库,3台从库,总共6台

--replicate 2 定义每个主库有2个从库,3台主库,6台从库,总共9台 以此类推

[root@sql51 src]# redis-trib.rb create --replicas 1 \

> 192.168.4.51:6351 \

> 192.168.4.52:6352 \

> 192.168.4.53:6353 \

> 192.168.4.54:6354 \

> 192.168.4.55:6355 \

> 192.168.4.56:6356

>>> Creating cluster

>>> Performing hash slots allocation on 6 nodes...

Using 3 masters:

192.168.4.51:6351 #主库

192.168.4.52:6352 #主库

192.168.4.53:6353 #主库

Adding replica 192.168.4.55:6355 to 192.168.4.51:6351 #51的从库55

Adding replica 192.168.4.56:6356 to 192.168.4.52:6352 #52的从库56

Adding replica 192.168.4.54:6354 to 192.168.4.53:6353 #53的从库54

M: 412a79829c7d0eed812b24959ca123b58990d60e 192.168.4.51:6351

slots:0-5460 (5461 slots) master

M: d63724ee3c6990a98d41b432ee913796245f4999 192.168.4.52:6352

slots:5461-10922 (5462 slots) master

M: c269f171a53cfbdf3aa5f4f2387731ca7af2f8d8 192.168.4.53:6353

slots:10923-16383 (5461 slots) master

S: 6e4b41e1b1e86faa68fe179aa1ab6ff7c0a6385e 192.168.4.54:6354

replicates c269f171a53cfbdf3aa5f4f2387731ca7af2f8d8

S: 1dc26eb0d3c281ee74a50dc9d8544bc3557f4411 192.168.4.55:6355

replicates 412a79829c7d0eed812b24959ca123b58990d60e

S: 705304b8c3a9187d29430f6d6df6bfaf32138168 192.168.4.56:6356

replicates d63724ee3c6990a98d41b432ee913796245f4999

Can I set the above configuration? (type 'yes' to accept): yes #输入yes

>>> Nodes configuration updated

>>> Assign a different config epoch to each node

>>> Sending CLUSTER MEET messages to join the cluster

Waiting for the cluster to join...

>>> Performing Cluster Check (using node 192.168.4.51:6351)

M: 412a79829c7d0eed812b24959ca123b58990d60e 192.168.4.51:6351

slots:0-5460 (5461 slots) master

1 additional replica(s)

S: 705304b8c3a9187d29430f6d6df6bfaf32138168 192.168.4.56:6356

slots: (0 slots) slave

replicates d63724ee3c6990a98d41b432ee913796245f4999

S: 6e4b41e1b1e86faa68fe179aa1ab6ff7c0a6385e 192.168.4.54:6354

slots: (0 slots) slave

replicates c269f171a53cfbdf3aa5f4f2387731ca7af2f8d8

S: 1dc26eb0d3c281ee74a50dc9d8544bc3557f4411 192.168.4.55:6355

slots: (0 slots) slave

replicates 412a79829c7d0eed812b24959ca123b58990d60e

M: c269f171a53cfbdf3aa5f4f2387731ca7af2f8d8 192.168.4.53:6353

slots:10923-16383 (5461 slots) master

1 additional replica(s)

M: d63724ee3c6990a98d41b432ee913796245f4999 192.168.4.52:6352

slots:5461-10922 (5462 slots) master

1 additional replica(s)

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

## 2.4查看集群状态

192.168.4.51:6351> cluster info

cluster\_state:ok #状态ok

cluster\_slots\_assigned:16384

cluster\_slots\_ok:16384

cluster\_slots\_pfail:0

cluster\_slots\_fail:0

cluster\_known\_nodes:6

cluster\_size:3

cluster\_current\_epoch:6

cluster\_my\_epoch:1

cluster\_stats\_messages\_ping\_sent:615

cluster\_stats\_messages\_pong\_sent:546

cluster\_stats\_messages\_sent:1161

cluster\_stats\_messages\_ping\_received:541

cluster\_stats\_messages\_pong\_received:615

cluster\_stats\_messages\_meet\_received:5

cluster\_stats\_messages\_received:1161

192.168.4.51:6351> cluster nodes

412a79829c7d0eed812b24959ca123b58990d60e 192.168.4.51:6351@16351 myself,master - 0 1551343453000 1 connected 0-5460

705304b8c3a9187d29430f6d6df6bfaf32138168 192.168.4.56:6356@16356 slave d63724ee3c6990a98d41b432ee913796245f4999 0 1551343453000 6 connected

6e4b41e1b1e86faa68fe179aa1ab6ff7c0a6385e 192.168.4.54:6354@16354 slave c269f171a53cfbdf3aa5f4f2387731ca7af2f8d8 0 1551343453000 4 connected

1dc26eb0d3c281ee74a50dc9d8544bc3557f4411 192.168.4.55:6355@16355 slave 412a79829c7d0eed812b24959ca123b58990d60e 0 1551343454502 5 connected

c269f171a53cfbdf3aa5f4f2387731ca7af2f8d8 192.168.4.53:6353@16353 master - 0 1551343452596 3 connected 10923-16383

d63724ee3c6990a98d41b432ee913796245f4999 192.168.4.52:6352@16352 master - 0 1551343453999 2 connected 5461-10922.

[root@sql51 ~]# redis-trib.rb info 192.168.4.51:6351

192.168.4.51:6351 (412a7982...) -> 2 keys | 5461 slots | 1 slaves.

192.168.4.53:6353 (c269f171...) -> 1 keys | 5461 slots | 1 slaves.

192.168.4.52:6352 (d63724ee...) -> 2 keys | 5462 slots | 1 slaves.

[OK] 5 keys in 3 masters.

0.00 keys per slot on average.

## 2.5排错

[root@sql52 ~]# cat /var/lib/redis/6379/nodes-6352.conf #集群信息此文件

如果报错信息提示:原数据库里面有数据和已存在集群中,如下操作

[root@sql52 ~]# ls /var/lib/redis/6379/ #删此处的文件

dump.rdb nodes-6352.conf

[root@sql52 ~]# cat /var/lib/redis/6379/nodes-6352.conf 再删此处的文件

[root@sql58 ~]# netstat -unltp | grep redis

tcp 0 0 192.168.4.58:6358 0.0.0.0:\* LISTEN 7903/redis-server 1

tcp 0 0 192.168.4.58:16358 0.0.0.0:\* LISTEN 7903/redis-server 1

[root@sql58 ~]# ps aux |grep redis

root 7903 0.1 0.9 147308 9656 ? Ssl 19:19 0:04 /usr/local/bin/redis-server 192.168.4.58:6358 [cluster]

root 8418 0.0 0.0 112676 980 pts/0 S+ 20:14 0:00 grep --color=auto redis

[root@sql58 ~]# kill -9 7903

[root@sql58 ~]# /etc/init.d/redis\_6379 start

/var/run/redis\_6379.pid exists, process is already running or crashed #错误

[root@sql58 ~]# ls /var/run/redis\_6379.pid

/var/run/redis\_6379.pid

[root@sql58 ~]# rm /var/run/redis\_6379.pid

rm：是否删除普通文件 "/var/run/redis\_6379.pid"？y

[root@sql58 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

## 2.6管理机上检测集群

[root@sql51 src]# redis-trib.rb check 192.168.4.51:6351

>>> Performing Cluster Check (using node 192.168.4.51:6351)

M: 412a79829c7d0eed812b24959ca123b58990d60e 192.168.4.51:6351

slots:0-5460 (5461 slots) master #主库哈希槽的个数和范围

1 additional replica(s)

S: 705304b8c3a9187d29430f6d6df6bfaf32138168 192.168.4.56:6356

slots: (0 slots) slave

replicates d63724ee3c6990a98d41b432ee913796245f4999

S: 6e4b41e1b1e86faa68fe179aa1ab6ff7c0a6385e 192.168.4.54:6354

slots: (0 slots) slave

replicates c269f171a53cfbdf3aa5f4f2387731ca7af2f8d8

S: 1dc26eb0d3c281ee74a50dc9d8544bc3557f4411 192.168.4.55:6355

slots: (0 slots) slave

replicates 412a79829c7d0eed812b24959ca123b58990d60e

M: c269f171a53cfbdf3aa5f4f2387731ca7af2f8d8 192.168.4.53:6353

slots:10923-16383 (5461 slots) master

1 additional replica(s)

M: d63724ee3c6990a98d41b432ee913796245f4999 192.168.4.52:6352

slots:5461-10922 (5462 slots) master

1 additional replica(s)

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

有三个主库,三个从库

[root@sql51 src]# redis-trib.rb info 192.168.4.51:6351

192.168.4.51:6351 (412a7982...) -> 0 keys | 5461 slots | 1 slaves.

192.168.4.53:6353 (c269f171...) -> 0 keys | 5461 slots | 1 slaves.

192.168.4.52:6352 (d63724ee...) -> 0 keys | 5462 slots | 1 slaves.

[OK] 0 keys in 3 masters.

0.00 keys per slot on average.

## 2.7登录集群 - C

[root@sql51 src]# redis-cli -c -h 192.168.4.51 -p 6351

192.168.4.51:6351> keys \*

(empty list or set)

192.168.4.51:6351> set name bob

-> Redirected to slot [5798] located at 192.168.4.52:6352

OK

192.168.4.52:6352> set age 99

-> Redirected to slot [741] located at 192.168.4.51:6351

OK

192.168.4.51:6351> set school tarena

-> Redirected to slot [8455] located at 192.168.4.52:6352

OK

192.168.4.52:6352> keys \*

1) "name"

2) "school"

192.168.4.52:6352> get name

"bob"

192.168.4.52:6352> get age

-> Redirected to slot [741] located at 192.168.4.51:6351

"99"

## 2.8哈希槽0-16383

192.168.4.51:6351> set x 999

-> Redirected to slot [16287] located at 192.168.4.53:6353

OK

当客户端存数据时候通过crc16算法,计算哈希值

set x 999 计算的结果为16287 (将x crc16算法做 hash计算得到数字,用这个数字%16384(取余) 得到16287

192.168.4.51:6351> get x

999

当客户端取数据时候通过crc16算法,计算哈希值

get x 计算的结果为16287 (将x crc16算法做 hash计算得到数字,用这个数字%16384(取余) 得到16287,集群找16287槽

Redis 集群中内置了 16384 个哈希槽(0-16383)，当需要在 Redis 集群中放置一个 key-value时，redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节点。

Redis 集群没有使用一致性hash, 而是引入了哈希槽的概念。

Redis 集群有16384个哈希槽,每个key通过CRC16校验后对16384取模来决定放置哪个槽.集群的每个节点负责一部分hash槽。这种结构很容易添加或者删除节点，并且无论是添加删除或者修改某一个节点，都不会造成集群不可用的状态

使用哈希槽的好处就在于可以方便的添加或移除节点。

当需要增加节点时，只需要把其他节点的某些哈希槽挪到新节点就可以了；

当需要移除节点时，只需要把移除节点上的哈希槽挪到其他节点就行了；

在这一点上，我们以后新增或移除节点的时候不用先停掉所有的 redis 服务。

## 2.9集群工作过程

## 2.10测试集群的高可用功能

高可用功能.当主库宕机时,对应的从库自动升级为主库

当从库顶替主库成为主库后,又宕机了,那么整个集群也就挂掉了

测试方法:停止任意一台主库的redis服务,查看集群信息(停止52)

# 三 管理redis集群(管理机51上操作)

## 3.1 向集群里添加新redis服务器

使用脚本的add-node 命令

### 3.1.1 添加master 角色

**添加redis服务器**

[root@sql51 ~]# redis-trib.rb del-node 要添加的主机ip:端口 任意主机:端口

[root@sql51 ~]# redis-trib.rb add-node 192.168.4.58:6358 192.168.4.51:6351

#如不指定角色 默认是添加主库

>>> Send CLUSTER MEET to node 192.168.4.58:6358 to make it join the cluster.

[OK] New node added correctly.

**查看集群主库信息**

[root@sql51 ~]# redis-trib.rb info 192.168.4.51:6351 #查看

192.168.4.51:6351 (412a7982...) -> 2 keys | 5461 slots | 1 slaves.

192.168.4.58:6358 (c2e039c3...) -> 0 keys | 0 slots | 0 slaves. #存有0个变量

192.168.4.53:6353 (c269f171...) -> 1 keys | 5461 slots | 1 slaves.

192.168.4.52:6352 (d63724ee...) -> 2 keys | 5462 slots | 1 slaves.

[OK] 5 keys in 4 masters.

0.00 keys per slot on average.

**分配hash slot(哈希槽)**

[root@sql51 ~]# redis-trib.rb reshard 192.168.4.58:6358 #任意集群中ip

>>> Performing Cluster Check (using node 192.168.4.58:6358)

M: c2e039c3a388e62ec6c02a8af3b5a8bd53040a65 192.168.4.58:6358

slots: (0 slots) master

0 additional replica(s)

S: 6e4b41e1b1e86faa68fe179aa1ab6ff7c0a6385e 192.168.4.54:6354

slots: (0 slots) slave

replicates c269f171a53cfbdf3aa5f4f2387731ca7af2f8d8

S: 1dc26eb0d3c281ee74a50dc9d8544bc3557f4411 192.168.4.55:6355

slots: (0 slots) slave

replicates 412a79829c7d0eed812b24959ca123b58990d60e

M: d63724ee3c6990a98d41b432ee913796245f4999 192.168.4.52:6352

slots:5461-10922 (5462 slots) master

1 additional replica(s)

M: c269f171a53cfbdf3aa5f4f2387731ca7af2f8d8 192.168.4.53:6353

slots:10923-16383 (5461 slots) master

1 additional replica(s)

M: 412a79829c7d0eed812b24959ca123b58990d60e 192.168.4.51:6351

slots:0-5460 (5461 slots) master

1 additional replica(s)

S: 705304b8c3a9187d29430f6d6df6bfaf32138168 192.168.4.56:6356

slots: (0 slots) slave

replicates d63724ee3c6990a98d41b432ee913796245f4999

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

How many slots do you want to move (from 1 to 16384)? 4096

What is the receiving node ID? c2e039c3a388e62ec6c02a8af3b5a8bd53040a65

Please enter all the source node IDs.

Type 'all' to use all the nodes as source nodes for the hash slots.

Type 'done' once you entered all the source nodes IDs.

Source node #1:all

Do you want to proceed with the proposed reshard plan (yes/no)?yes

[root@sql51 ~]# redis-trib.rb check 192.168.4.58:6358

>>> Performing Cluster Check (using node 192.168.4.58:6358)

M: c2e039c3a388e62ec6c02a8af3b5a8bd53040a65 192.168.4.58:6358

slots:0-1364,5461-6826,10923-12287 (4096 slots) master

0 additional replica(s) #再次查看发现4.58有4096个hash slot

### 3.1.2 添加slave 角色

没有指定[--master-id id值],默认谁的主库最少,自动添加为谁的从库

Add-node --slave [--master-id id值] ip地址:端口 192.168.4.51:6351

[root@sql51 ~]# redis-trib.rb add-node --slave 192.168.4.57:6357 192.168.4.51:6351

>>> Send CLUSTER MEET to node 192.168.4.57:6357 to make it join the cluster.

Waiting for the cluster to join.

>>> Configure node as replica of 192.168.4.58:6358.

[OK] New node added correctly.

[root@sql51 ~]# redis-trib.rb info 192.168.4.51:6351

192.168.4.51:6351 (412a7982...) -> 0 keys | 4096 slots | 1 slaves.

192.168.4.58:6358 (c2e039c3...) -> 3 keys | 4096 slots | 1 slaves. #有了从库

192.168.4.53:6353 (c269f171...) -> 1 keys | 4096 slots | 1 slaves.

192.168.4.52:6352 (d63724ee...) -> 1 keys | 4096 slots | 1 slaves.

[OK] 5 keys in 4 masters.

0.00 keys per slot on average.

[root@sql51 ~]# redis-trib.rb check 192.168.4.57:6357

>>> Performing Cluster Check (using node 192.168.4.57:6357)

S: f5251bebf750c5beaa48fbbed03e880889e07fcd 192.168.4.57:6357

slots: (0 slots) slave

replicates c2e039c3a388e62ec6c02a8af3b5a8bd53040a65

## 3.2 移除集群中的redis服务器

### 3.2.1 移除slave角色(57)

[root@sql51 ~]# redis-trib.rb del-node 任意主机:端口 被移除的主机id

[root@sql51 ~]# redis-trib.rb del-node 192.168.4.51:6351 f5251bebf750c5beaa48fbbed03e880889e07fcd #57的id

>>> Removing node f5251bebf750c5beaa48fbbed03e880889e07fcd from cluster 192.168.4.51:6351

>>> Sending CLUSTER FORGET messages to the cluster...

>>> SHUTDOWN the node.

[root@sql51 ~]# redis-trib.rb info 192.168.4.51:6351

192.168.4.51:6351 (412a7982...) -> 0 keys | 4096 slots | 1 slaves.

192.168.4.58:6358 (c2e039c3...) -> 3 keys | 4096 slots | 0 slaves. #0个从服务

192.168.4.53:6353 (c269f171...) -> 1 keys | 4096 slots | 1 slaves.

192.168.4.52:6352 (d63724ee...) -> 1 keys | 4096 slots | 1 slaves.

[OK] 5 keys in 4 masters.

0.00 keys per slot on average.

### 3.2.2 移除master角色

#### 重新分片:释放占用的hash槽

[root@sql51 ~]#redis-trib.rb reshard 192.168.4.51:6351

[OK] All 16384 slots covered.

How many slots do you want to move (from 1 to 16384)? 4096

What is the receiving node ID? 412a79829c7d0eed812b24959ca123b58990d60e #接受哈希槽的主机id(51)

Please enter all the source node IDs.

Type 'all' to use all the nodes as source nodes for the hash slots.

Type 'done' once you entered all the source nodes IDs.

Source node #1:c2e039c3a388e62ec6c02a8af3b5a8bd53040a65

#要移除的主机id(58)

Source node #2:done #只移除一个主机的最后写done

[root@sql51 ~]# redis-trib.rb info 192.168.4.51:6351

192.168.4.51:6351 (412a7982...) -> 3 keys | 8192 slots | 1 slaves.

192.168.4.58:6358 (c2e039c3...) -> 0 keys | 0 slots | 0 slaves.

192.168.4.53:6353 (c269f171...) -> 1 keys | 4096 slots | 1 slaves.

192.168.4.52:6352 (d63724ee...) -> 1 keys | 4096 slots | 1 slaves.

#### 再删除

[root@sql51 ~]# redis-trib.rb del-node 任意主机:端口 被移除的主机id

[root@sql51 ~]# redis-trib.rb del-node 192.168.4.51:6351 c2e039c3a388e62ec6c02a8af3b5a8bd53040a65 #被移除的主机id58

>>> Removing node c2e039c3a388e62ec6c02a8af3b5a8bd53040a65 from cluster 192.168.4.51:6351

>>> Sending CLUSTER FORGET messages to the cluster...

>>> SHUTDOWN the node.

[root@sql51 ~]# redis-trib.rb info 192.168.4.51:6351 #没有58了

192.168.4.51:6351 (412a7982...) -> 3 keys | 8192 slots | 1 slaves.

192.168.4.53:6353 (c269f171...) -> 1 keys | 4096 slots | 1 slaves.

192.168.4.52:6352 (d63724ee...) -> 1 keys | 4096 slots | 1 slaves.

[OK] 5 keys in 3 masters.

0.00 keys per slot on average.

#### 扩展:平均分片

[root@sql51 ~]# redis-trib.rb info 192.168.4.51:6351

192.168.4.51:6351 (412a7982...) -> 3 keys | 8192 slots | 1 slaves.

192.168.4.53:6353 (c269f171...) -> 1 keys | 4096 slots | 1 slaves.

192.168.4.52:6352 (d63724ee...) -> 1 keys | 4096 slots | 1 slaves.

[OK] 5 keys in 3 masters.

0.00 keys per slot on average.

[root@sql51 ~]# redis-trib.rb rebalance 192.168.4.51:6351

>>> Performing Cluster Check (using node 192.168.4.51:6351)

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

>>> Rebalancing across 3 nodes. Total weight = 3

Moving 1366 slots from 192.168.4.51:6351 to 192.168.4.53:6353

Moving 1365 slots from 192.168.4.51:6351 to 192.168.4.52:6352

[root@sql51 ~]# redis-trib.rb info 192.168.4.51:6351

192.168.4.51:6351 (412a7982...) -> 1 keys | 5461 slots | 1 slaves.

192.168.4.53:6353 (c269f171...) -> 3 keys | 5462 slots | 1 slaves.

192.168.4.52:6352 (d63724ee...) -> 1 keys | 5461 slots | 1 slaves.

[OK] 5 keys in 3 masters.

0.00 keys per slot on average.

## 3.3 把移除的redis服务器再添加到集群

### 清除原有的集群信息

移除之后的服务器是没有服务了,需要重新启动,

且集群信息文件还存在,还记录着之前的集群信息,必须清除才能新添加

[root@sql58 ~]# netstat -untlp |grep redis

[root@sql58 ~]# ls /var/lib/redis/6379/

dump.rdb nodes-6358.conf

[root@sql58 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@sql58 ~]# redis-cli -h 192.168.4.58 -p 6358 #此处不需要 - c

192.168.4.58:6358> cluster info

cluster\_state:ok

cluster\_slots\_assigned:16384

cluster\_slots\_ok:16384

cluster\_slots\_pfail:0

cluster\_slots\_fail:0

cluster\_known\_nodes:7

cluster\_size:3

cluster\_current\_epoch:12

cluster\_my\_epoch:9

cluster\_stats\_messages\_ping\_sent:192

cluster\_stats\_messages\_sent:192

cluster\_stats\_messages\_pong\_received:192

cluster\_stats\_messages\_received:192

192.168.4.58:6358> claster reset

(error) ERR unknown command 'claster'

192.168.4.58:6358> cluster reset #清除重置集群信息

OK

192.168.4.58:6358> cluster info #查看集群信息

cluster\_state:fail

cluster\_slots\_assigned:0

cluster\_slots\_ok:0

cluster\_slots\_pfail:0

cluster\_slots\_fail:0

cluster\_known\_nodes:1

cluster\_size:0

cluster\_current\_epoch:12

cluster\_my\_epoch:9

cluster\_stats\_messages\_ping\_sent:244

cluster\_stats\_messages\_sent:244

cluster\_stats\_messages\_pong\_received:244

cluster\_stats\_messages\_received:244

### 管理主机再添加master主机

[root@sql51 ~]# redis-trib.rb del-node 要添加的主机ip:端口 集群任意主机:端口

#未指定master | slave 默认是添加为master主

[root@sql51 ~]# redis-trib.rb add-node 192.168.4.58:6358 192.168.4.52:6352

>>> Adding node 192.168.4.58:6358 to cluster 192.168.4.52:6352

>>> Performing Cluster Check (using node 192.168.4.52:6352)

......

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

>>> Send CLUSTER MEET to node 192.168.4.58:6358 to make it join the cluster.

[OK] New node added correctly.

[root@sql51 ~]# redis-trib.rb check 192.168.4.58:6358

>>> Performing Cluster Check (using node 192.168.4.58:6358)

M: c2e039c3a388e62ec6c02a8af3b5a8bd53040a65 192.168.4.58:6358

slots: (0 slots) master

0 additional replica(s)

.......

### 管理主机再添加slave主机

启动服务后清除重置集群信息,再添加slave主机

[root@sql57 ~]# netstat -untlp | grep redis

[root@sql57 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@sql57 ~]# netstat -untlp | grep redis

tcp 0 0 192.168.4.57:6357 0.0.0.0:\* LISTEN 3336/redis-server 1

tcp 0 0 192.168.4.57:16357 0.0.0.0:\* LISTEN 3336/redis-server 1

[root@sql57 ~]# redis-cli -h 192.168.4.47 -p 6357

Could not connect to Redis at 192.168.4.47:6357: No route to host

Could not connect to Redis at 192.168.4.47:6357: No route to host

not connected>

[root@sql57 ~]# redis-cli -h 192.168.4.57 -p 6357

192.168.4.57:6357> cluster reset

OK

192.168.4.57:6357> cluster info

cluster\_state:fail #状态是失败

cluster\_slots\_assigned:0

cluster\_slots\_ok:0

cluster\_slots\_pfail:0

cluster\_slots\_fail:0

cluster\_known\_nodes:1

cluster\_size:0

cluster\_current\_epoch:12

cluster\_my\_epoch:0

cluster\_stats\_messages\_ping\_sent:183

cluster\_stats\_messages\_sent:183

cluster\_stats\_messages\_pong\_received:183

cluster\_stats\_messages\_received:183

[root@sql51 ~]# redis-trib.rb add-node --slave 192.168.4.57:6357 192.168.4.52:6352

>>> Adding node 192.168.4.57:6357 to cluster 192.168.4.52:6352

>>> Performing Cluster Check (using node 192.168.4.52:6352)

...

>>> Send CLUSTER MEET to node 192.168.4.57:6357 to make it join the cluster.

Waiting for the cluster to join.

>>> Configure node as replica of 192.168.4.58:6358.

[OK] New node added correctly.

[root@sql51 ~]#

[root@sql51 ~]# redis-trib.rb check 192.168.4.57:6357

>>> Performing Cluster Check (using node 192.168.4.57:6357)

S: f5251bebf750c5beaa48fbbed03e880889e07fcd 192.168.4.57:6357

slots: (0 slots) slave

replicates c2e039c3a388e62ec6c02a8af3b5a8bd53040a65

M: c2e039c3a388e62ec6c02a8af3b5a8bd53040a65 192.168.4.58:6358

slots: (0 slots) master

1 additional replica(s)

....

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

[root@sql51 ~]#

## 3.4将redis从集群中恢复单个服务

# NOSQL03------------------------

# 1 Redis 主从复制

工作原理:

1. Slave向master发送sync命令
2. Master启动后台存盘进程,并收集所有修改数据命令
3. Master完成后台存盘后,传送整个数据文件到slave
4. Slave接收数据文件(dump.rdb文件).加载到内存中完成首次完全同步
5. 后续有新数据产生,master继续将新的数据收集到的修改命令传送给slave,slave运行命令后完成同步

主从注意事项:

* 启动服务默认为主库
* 从库需要手动配置
* 从库默认是只读的
* 命令操作都是临时,

## 1.1 结构模式

一主一从

一主多从

主从从

## 1.2配置一主一从对应命令

### 查看主从配置信息

192.168.4.51:6351> info replication #查看主从配置信息

# Replication

role:master #主库 开启服务默认是主库

connected\_slaves:0 #从库个数为”0”

master\_replid:f6392ab7acaca07d48824a5d507ba559849edfc2

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

### 临时手动设置为从库

重启服务失效

[root@sql52 ~]# 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 #从库

master\_host:192.168.4.51 #主库地址

master\_port:6351

master\_link\_status:down #down状态是无法通讯,

master\_last\_io\_seconds\_ago:-1

master\_sync\_in\_progress:0

slave\_repl\_offset:0

master\_link\_down\_since\_seconds:1551421058

slave\_priority:100

slave\_read\_only:1

connected\_slaves:0

master\_replid:dbc0e5da85866689df82627203fe344a708933fe

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

主库存数据

192.168.4.51:6351> set x 1

OK

192.168.4.51:6351> set y 2

OK

192.168.4.51:6351> set z 3

OK

从库查看数据

192.168.4.52:6352> keys \*

1) "y"

2) "z"

3) "x"

### 永久配置:修改配置文件

[root@sql52 ~]# vim /etc/redis/6379.conf

282 # slaveof <masterip> <masterport> #修改前

282 slaveof 192.168.4.51 6351 #修改后

[root@sql52 ~]# /etc/init.d/redis\_6379 stop #停止服务

Stopping ...

Redis stopped

[root@sql52 ~]# /etc/init.d/redis\_6379 start #开启服务

Starting Redis server...

[root@sql52 ~]# redis-cli -h 192.168.4.52 -p 6352

192.168.4.52:6352> info replication

# Replication

role:slave

master\_host:192.168.4.51

master\_port:6351

master\_link\_status:up

master\_last\_io\_seconds\_ago:9

master\_sync\_in\_progress:0

slave\_repl\_offset:1014

slave\_priority:100

slave\_read\_only:1

connected\_slaves:0

master\_replid:721fe422b4ca96a06d146d1cb264f52caf085bdf

master\_replid2:0000000000000000000000000000000000000000

master\_repl\_offset:1014

second\_repl\_offset:-1

repl\_backlog\_active:1

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:1001

repl\_backlog\_histlen:14

### 临时手动设置为主库

重启失效,恢复从库

192.168.4.52:6352> slaveof no one

OK

192.168.4.52:6352> info replication

# Replication

role:master

connected\_slaves:0

### 永久配置主库

注释配置文件中

282# slaveof 192.168.4.51 6351 #注释这行

配置有密码的主从复制

### 主库设置连接密码的主从配置

501 requirepass 123456

[root@sql51 ~]# vim /etc/init.d/redis\_6379

43 $CLIEXEC -h 192.168.4.51 -p 6351 -a 123456 shutdown

[root@sql51 ~]# redis-cli -h 192.168.4.51 -p 6351 -a 123456

192.168.4.51:6351>

192.168.4.51:6351> info replication

# Replication

role:master

connected\_slaves:0

2在从库中修改配置文件

192.168.4.52:6352> info replication

# Replication

role:slave

master\_host:192.168.4.51

master\_port:6351

master\_link\_status:down #未指定密码前是down

[root@sql52 ~]# vim /etc/redis/6379.conf

289 # masterauth <master-password>

289 # masterauth 123456

[root@sql52 ~]# /etc/init.d/redis\_6379 stop

Stopping ...

Redis stopped

[root@sql52 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@sql52 ~]# redis-cli -h 192.168.4.52 -p 6352

192.168.4.52:6352> info replication

# Replication

role:slave

master\_host:192.168.4.51

master\_port:6351

master\_link\_status:up #修改后重启 为UP

master\_last\_io\_seconds\_ago:9

### 从库默认是只读的

192.168.4.52:6352> set aa 11

(error) READONLY You can't write against a read only slave.

## 1.2配置一主多从\主从从

51为52,53的主 52为51的从同时为54的主 53为51的从 54为52的从

在上面实验的基础上:51为52的主,,51设置了密码

53配置为

282 slaveof 192.168.4.51 6351

289 masterauth 123456

54配置为

282 slaveof 192.168.4.52 6352

289 masterauth 123456

## 1.3哨兵模式

* 监视当前主库,当主库宕机后,从对应的从库里选举新的主库
* 是一种服务,可以在主库上运行.也可以单独拿一台设备运行
* 使用哨兵模式,如果主服务器设置了密码.各从服务器配置文件中设置了 masterauth 123456(指定连接主服务器密码),因为从服务器没有设置相同连接密码(或者没设置连接密码)的主服务器会漂移,但是所有的从服务器状态都会是down,

### 1 需要安装redis软件包

(本实验用55做,之前已安装完成)

### 2创建服务主配置文件

有配置文件模板

[root@sql51 ~]# ls /root/redis-4.0.8/sentinel.conf

/root/redis-4.0.8/sentinel.conf #模板

直接编辑空白文件,添加内容也可

[root@sql55 ~]# vim /etc/sentinel.conf

bind 0.0.0.0 #实验中未配置

sentinel monitor sql51 192.168.4.51 6351 1

sentinel auth-pass 123456

#sentinel monitor master主机名 masterip地址 master端口 票数

#sentinel auth-pass 连接主库密码

#票数:有几台哨兵主机连接不上主库时切换主库,这里只有一台哨兵服务器主机

2863

### 3启动服务

[root@sql55 ~]# redis-sentinel /etc/sentinel.conf

5886:X 01 Mar 16:09:03.325 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo

5886:X 01 Mar 16:09:03.325 # Redis version=4.0.8, bits=64, commit=00000000, modified=0, pid=5886, just started

5886:X 01 Mar 16:09:03.325 # Configuration loaded

5886:X 01 Mar 16:09:03.327 \* Increased maximum number of open files to 10032 (it was originally set to 1024).

5886:X 01 Mar 16:09:03.379 # Sentinel ID is 23a57eb24521792de0bef4bfbfa054f5b1032c9b

5886:X 01 Mar 16:09:03.379 # +monitor master sql51 192.168.4.51 6351 quorum 1

5886:X 01 Mar 16:09:03.382 \* +slave slave 192.168.4.52:6352 192.168.4.52 6352 @ sql51 192.168.4.51 6351

5886:X 01 Mar 16:09:03.497 \* +slave slave 192.168.4.53:6353 192.168.4.53 6353 @ sql51 192.168.4.51 6351

### 4主库发生宕机之前51 52 53的主从信息

192.168.4.51:6351> info replication

# Replication

role:master

connected\_slaves:2

slave0:ip=192.168.4.52,port=6352,state=online,offset=97270,lag=1

slave1:ip=192.168.4.53,port=6353,state=online,offset=97270,lag=0

master\_replid:0e570019c5576f6196a182ede0d6e4c37adbc09a

master\_replid2:0000000000000000000000000000000000000000

master\_repl\_offset:97270

second\_repl\_offset:-1

repl\_backlog\_active:1

repl\_backlog\_size:1048576

repl\_backlog\_first\_byte\_offset:1

repl\_backlog\_histlen:97270

192.168.4.52:6352> info replication

# Replication

role:slave

master\_host:192.168.4.51

master\_port:6351

master\_link\_status:up

192.168.4.53:6353> info replication

# Replication

role:slave

master\_host:192.168.4.51

master\_port:6351

master\_link\_status:up

### 5主库发生宕机之后51 52 53的主从信息

192.168.4.51:6351> exit

[root@sql51 ~]# /etc/init.d/redis\_6379 stop

Stopping ...

Waiting for Redis to shutdown ...

Redis stopped

192.168.4.52:6352> info replication

# Replication

role:slave

master\_host:192.168.4.53 #52的主库变53

master\_port:6353

master\_link\_status:down

192.168.4.53:6353> info replication

# Replication

role:master #53变主库

connected\_slaves:0

# 2 redis数据持久化(RDB/AOF)

服务在运行的过程中,自动把内存里的数据存储到物理硬盘

## 持久化（Persistence），介绍

* 即把数据（如内存中的对象）保存到可永久保存的存储设备中（如磁盘）。
* 持久化的主要应用是将内存中的对象存储在数据库中，或者存储在磁盘文件中、XML数据文件中等等。
* 持久化是将程序数据在持久状态和瞬时状态间转换的机制。
* JDBC就是一种持久化机制。文件IO也是一种持久化机制。

日常持久化的方法

将鲜肉冷藏，吃的时候再解冻的方法也是。

将水果做成罐头的方法也是。

将人的脏器迅速冷冻，运输，然后解冻给人移植的技术也是。

# 持久化之RDB

## RDB介绍

Redis数据库文件,全程Redis DataBase

* 数据持久化方式之一
* 按照指定时间间隔,将内存中的数据集快照写入硬盘
* 术语叫Snapshot快照
* 恢复时,将快照文件直接读入内存

文件名:

[root@sql57 ~]# vim /etc/redis/6379.conf

254 dbfilename dump.rdb

## 使用RDB文件恢复数据

[root@sql56 6379]# redis-cli -h 192.168.4.56 -p 6356

192.168.4.56:6356> keys \*

1) "y"

2) "x"

3) "a51"

4) "z"

5) "a1"

192.168.4.56:6356> exit

[root@sql56 6379]# scp /var/lib/redis/6379/dump.rdb 192.168.4.57:/root

[root@sql57 ~]# cp dump.rdb /var/lib/redis/6379/

[root@sql57 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@sql57 ~]# redis-cli -h 192.168.4.57 -p 6357

192.168.4.57:6357> keys \*

1) "a51"

2) "x"

3) "y"

4) "a1"

5) "z" #57用了56的dump.rdb文件,两边数据一样

## 数据从内存保存到硬盘的频率

按照配置的时间规律将数据保存到dump文件内

[root@sql57 ~]# vim /etc/redis/6379.conf #配置文件中

217 # save ""

219 save 900 1

220 save 300 10

221 save 60 10000

save "" #禁用RDB

Save 900 1 900秒内且至少有1次修改,进行存盘

Save 300 10 300 秒内且至少有10次修改,则进行存盘

Save 60 10000 60 秒内且至少有10000次修改则进行存盘

[root@sql57 ~]# /etc/init.d/redis\_6379 stop

Stopping ...

Waiting for Redis to shutdown ...

Redis stopped

[root@sql57 ~]# vim /etc/redis/6379.conf

#save 900 1

#save 300 10

#save 60 10000

save 120 10

[root@sql57 ~]# rm /var/lib/redis/6379/\*

rm：是否删除普通文件 "/var/lib/redis/6379/dump.rdb"？y

[root@sql57 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@sql57 ~]# ls /var/lib/redis/6379/

[root@sql57 ~]# redis-cli -h 192.168.4.57 -p 6357

192.168.4.57:6357> set a1 1

OK

192.168.4.57:6357> set a1 2

OK

192.168.4.57:6357> set a1 3

OK

192.168.4.57:6357> set a1 4

OK

192.168.4.57:6357> set a1 5

OK

192.168.4.57:6357> set a1 6

OK

192.168.4.57:6357> set a1 7

OK

192.168.4.57:6357> set a1 8

OK

192.168.4.57:6357> set a1 9

OK

192.168.4.57:6357> set a1 10

OK

192.168.4.57:6357> set a1 11

OK

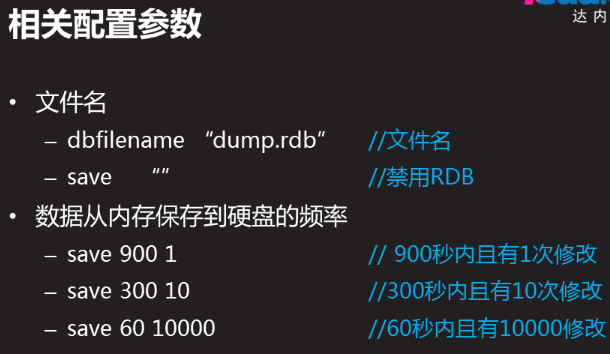
192.168.4.57:6357> exit

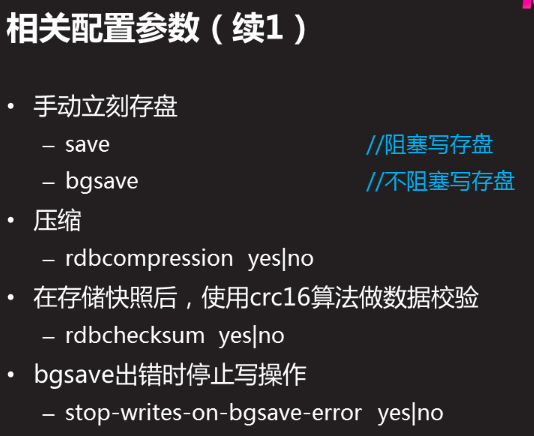
[root@sql57 ~]# ls /var/lib/redis/6379/

[root@sql57 ~]# ls /var/lib/redis/6379/

[root@sql57 ~]# ls /var/lib/redis/6379/

dump.rdb #2分钟后出现dump文件





## RBD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缺点

意外宕机时,最后一次持久化的数据会丢失

# 持久化之AOF

为了弥补RDB的缺点,出现AOF

只做追加操作的文件.Append Only File

* 只记录redis服务所有写的操作
* 不断的将新的写操作追加到文件的末尾
* 就是一个文本文件,使用cat命令可以查看文件内容
* 默认未启用

## 相关配置参数

文件名

* appendfilename “appendonly.aof” #指定文件名
* appendonly on #启用aof ,默认no

[root@sql56 ~]# vim /etc/redis/6379.conf

673 appendonly yes

677 appendfilename "appendonly.aof"

AOF文件记录写操作的方式

* appendfsync always #有新写操作立即记录,记录命令同时把数据写入dump.rdb
* appendfsync everysec #每秒记录一次,记录命令同时把数据写入dump.rdb
* Appendfsync no #从不记录,

[root@sql56 ~]# vim /etc/redis/6379.conf

702 # appendfsync always

703 appendfsync everysec #3选一

704 # appendfsync no

[root@sql56 ~]# ls /var/lib/redis/6379/

dump.rdb

[root@sql56 ~]# /etc/init.d/redis\_6379 stop

Stopping ...

Waiting for Redis to shutdown ...

Redis stopped

[root@sql56 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@sql56 ~]# ls /var/lib/redis/6379/

appendonly.aof dump.rdb

新开终端,tail -f

[root@sql56 ~]# tail -f /var/lib/redis/6379/appendonly.aof #实时查看记录

## 使用AOF文件恢复数据

[root@sql56 ~]# cp /var/lib/redis/6379/appendonly.aof /root

cp：是否覆盖"/root/appendonly.aof"？ y

[root@sql56 ~]# scp /root/appendonly.aof 192.168.4.57:/root

appendonly.aof 100% 180 219.6KB/

[root@sql57 ~]# redis-cli -h 192.168.4.57 -p 6357

192.168.4.57:6357> keys \*

1) "a1"

192.168.4.57:6357> flushall

OK

192.168.4.57:6357> keys \*

(empty list or set)

192.168.4.57:6357> exit

[root@sql57 ~]# /etc/init.d/redis\_6379 stop

Stopping ...

Redis stopped

[root@sql57 ~]# vim /etc/redis/6379.conf

[root@sql57 ~]# rm -rf /var/lib/redis/6379/\*

[root@sql57 ~]# cp appendonly.aof /var/lib/redis/6379/

[root@sql57 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@sql57 ~]# redis-cli -h 192.168.4.57 -p 6357

192.168.4.57:6357> keys \*

1) "a22"

2) "a11"

192.168.4.57:6357>

## 优化配置日志重写

日志文件会不断增大,何时触发日志重写?

redis会记录上次重写时AOF文件的大小

默认配置当aof文件是上次rewrite 后大小的1倍且文件大于64M时触发

[root@sql56 ~]# vim /etc/redis/6379.conf

744 auto-aof-rewrite-percentage 100

#达到按照上次整理后大小的1陪,再次整理,不断反复

745 auto-aof-rewrite-min-size 64mb

#最小大小触发第一次整理的大小

## 修复AOF文件

把文件恢复到最后一次的正确操作.

[root@sql56 ~]# vim /var/lib/redis/6379/appendonly.aof

$3

562

asdf #末尾胡乱添加

asdf #末尾胡乱添加

[root@sql56 ~]# /etc/init.d/redis\_6379 stop

Stopping ...

Waiting for Redis to shutdown ...

Redis stopped

[root@sql56 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@sql56 ~]# redis-cli -h 192.168.4.56 -p 6356

Could not connect to Redis at 192.168.4.56:6356: Connection refused

Could not connect to Redis at 192.168.4.56:6356: Connection refused

not connected> exit

[root@sql56 ~]# netstat -unltp |grep redis #无端口信息

[root@sql56 ~]# redis-check-aof --fix /var/lib/redis/6379/appendonly.aof

0x b4: Expected prefix '\*', got: 'a'

AOF analyzed: size=192, ok\_up\_to=180, diff=12

This will shrink the AOF from 192 bytes, with 12 bytes, to 180 bytes

Continue? [y/N]: y

Successfully truncated AOF

[root@sql56 ~]# vim /var/lib/redis/6379/appendonly.aof #此时查看无胡乱添加

[root@sql56 ~]# /etc/init.d/redis\_6379 start

/var/run/redis\_6379.pid exists, process is already running or crashed

[root@sql56 ~]# redis-cli -h 192.168.4.56 -p 6356

Could not connect to Redis at 192.168.4.56:6356: Connection refused

Could not connect to Redis at 192.168.4.56:6356: Connection refused

[root@sql56 ~]# /etc/init.d/redis\_6379 stop

Stopping ...

Could not connect to Redis at 192.168.4.56:6356: Connection refused

Redis stopped #无法启动也无法停止.登录不进去

[root@sql56 ~]#

[root@sql56 ~]# rm -rf /var/run/redis\_6379.pid #删除pid文件即可

[root@sql56 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@sql56 ~]# redis-cli -h 192.168.4.56 -p 6356

192.168.4.56:6356>

## 报错信息:

检测到有/var/run/redis\_6379.pid 文件是判定启动状态,实际未启动

[root@sql56 ~]# /etc/init.d/redis\_6379 start

/var/run/redis\_6379.pid exists, process is already running or crashed

[root@sql56 ~]# redis-cli -h 192.168.4.56 -p 6356

Could not connect to Redis at 192.168.4.56:6356: Connection refused

Could not connect to Redis at 192.168.4.56:6356: Connection refused

[root@sql56 ~]# /etc/init.d/redis\_6379 stop

Stopping ...

Could not connect to Redis at 192.168.4.56:6356: Connection refused

Redis stopped

解决办法,删除pid文件即可

[root@sql56 ~]# rm -rf /var/run/redis\_6379.pid #删除pid文件即可

[root@sql56 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@sql56 ~]# redis-cli -h 192.168.4.56 -p 6356

192.168.4.56:6356>

AOF优点

* 可以灵活设置持久化方式,同步持久化appendfsyncalways 或异步持久化appendfsyns everysec
* 出现意外宕机时,仅可能丢失1秒的数据

AOF缺点

* 持久化文件体积通常会大于RDB方式
* 执行fsync策略时的速度可能会比RDB的方式慢

# 3 数据类型

## String字符串类型

### set命令

192.168.4.56:6356> set key value [EX seconds] [PX milliseconds] [NX|XX]

# [EX seconds] 有效期以秒计算

# [PX milliseconds] 有效期以毫秒计算

# [NX|XX] NX变量不存在才赋值 XX变量存在时才赋值

192.168.4.56:6356> set k 88 ex 30 以秒为单位计算

OK

192.168.4.56:6356> ttl k

(integer) 23

192.168.4.56:6356> ttl k

(integer) 21

192.168.4.56:6356> ttl k

(integer) -2 #存活值为负

192.168.4.56:6356> get k #不存在变量k

(nil)

192.168.4.56:6356> set a 1

OK

192.168.4.56:6356> set a 2 nx #nx不存在才赋值,赋值不成功

(nil)

192.168.4.56:6356> get a

"1"

192.168.4.56:6356> set a 2 xx #存在才复制,赋值成功

OK

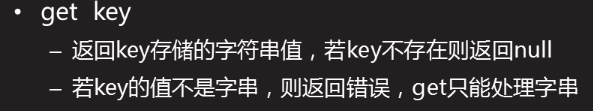
192.168.4.56:6356> get a

"2"

192.168.4.56:6356> set b 2 xx #变量b不存在,赋值不成功

(nil)

### get命令



### del命令(删除key)

192.168.4.56:6356> del key [key ...]

### Setrange复写命令

从偏移量开始复写key的特定位的值

192.168.4.56:6356> setrange key offset value

offset 偏移量位置 第一位是0

value 修改的内容值

192.168.4.56:6356> set dh 13973169942

OK

192.168.4.56:6356> get dh

"13973169942"

192.168.4.56:6356> setrange dh 3 \*\*\*\* #从第四位还是

(integer) 11

192.168.4.56:6356> get dh

"139\*\*\*\*9942"

### Strlen 统计字符串长度命令

统计字符串长度

192.168.4.56:6356> strlen key

192.168.4.56:6356> strlen dh

(integer) 11

### append 命令存(存在追加,不存在创建)

存在则追加,不存在则创建key及value,返回key长度

192.168.4.56:6356> append key value

192.168.4.56:6356> get a

"2"

192.168.4.56:6356> append a 3

(integer) 2

192.168.4.56:6356> get a

"23"

### Setbit 命令(设置位上的值)

* 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

192.168.4.56:6356> setbit key offset value

192.168.4.56:6356> setbit bit 1

(error) ERR wrong number of arguments for 'setbit' command

192.168.4.56:6356> SETBIT zdd 1 1

(integer) 0

192.168.4.56:6356> SETBIT zdd 2 2 #Value值只能为1或0

(error) ERR bit is not an integer or out of range

192.168.4.56:6356> SETBIT zdd 2 0

(integer) 0

192.168.4.56:6356> SETBIT zdd 3 1

(integer) 0

192.168.4.56:6356> SETBIT zdd 4 0

(integer) 0

192.168.4.56:6356> SETBIT zdd 5 1

(integer) 0

192.168.4.56:6356> SETBIT zdd 6 1

(integer) 0

192.168.4.56:6356> SETBIT zdd 7 0

(integer) 0

192.168.4.56:6356> SETBIT zdd 8 1

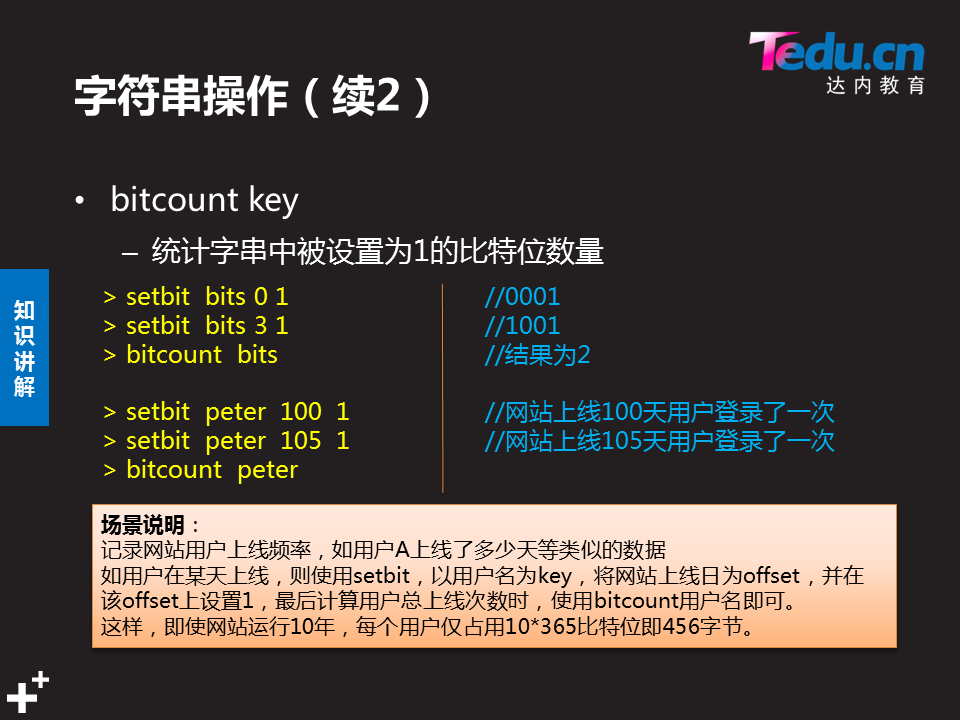
(integer) 0

192.168.4.56:6356> BITCOUNT zdd

(integer) 5



### Bitcount命令(统计1的比特位数量)



统计字串中被设置为1的比特位数量

192.168.4.56:6356> bitcount key [start end]

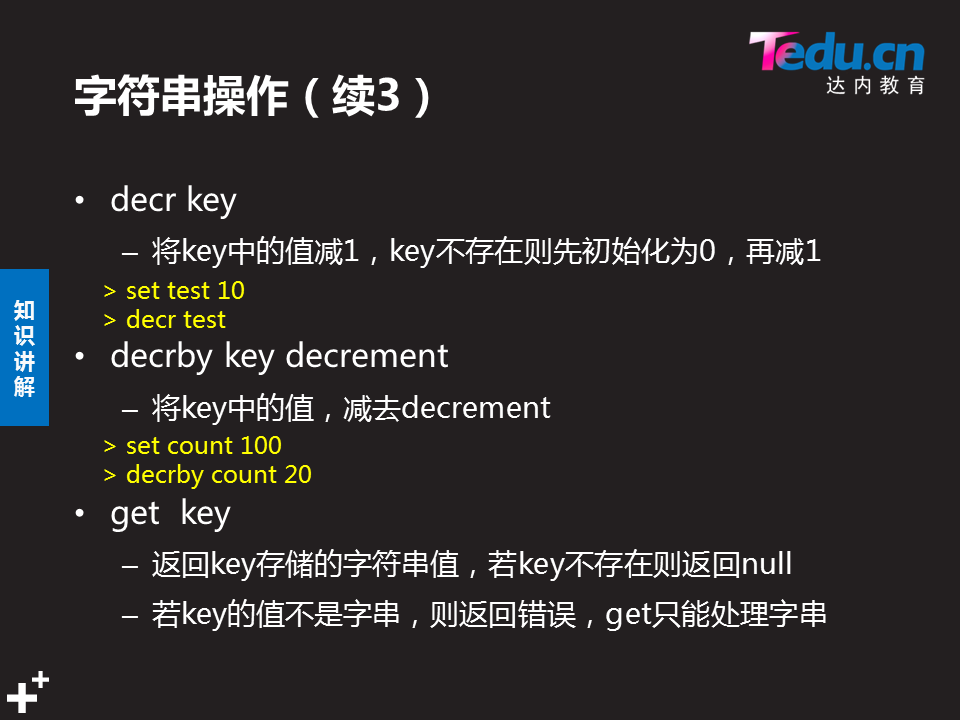
192.168.4.56:6356> setbit zdd 8 0

(integer) 1

192.168.4.56:6356> bitcount zdd

(integer) 4

### decr和decrby命令(自减)



192.168.4.56:6356> decr key

192.168.4.56:6356> set test 10

OK

192.168.4.56:6356> decr test #自减1

(integer) 9

192.168.4.56:6356> get test

"9"

192.168.4.56:6356> decr test

(integer) 8

192.168.4.56:6356> get test

"8"

192.168.4.56:6356> decrby test 2 #自减2

(integer) 6

192.168.4.56:6356> get test

"6"

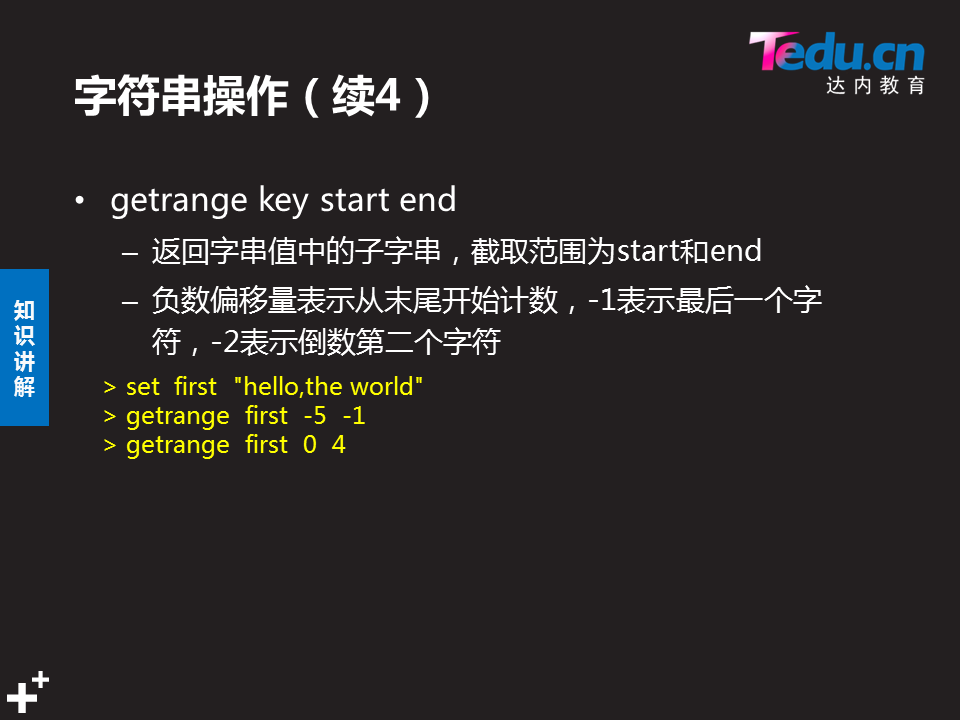
192.168.4.56:6356> decrby test -2 #自减-2(加2)

(integer) 8

192.168.4.56:6356> get test

"8"

### getrange命令(截取字符)



192.168.4.56:6356> set x abcdef

OK

192.168.4.56:6356> get x

"abcdef"

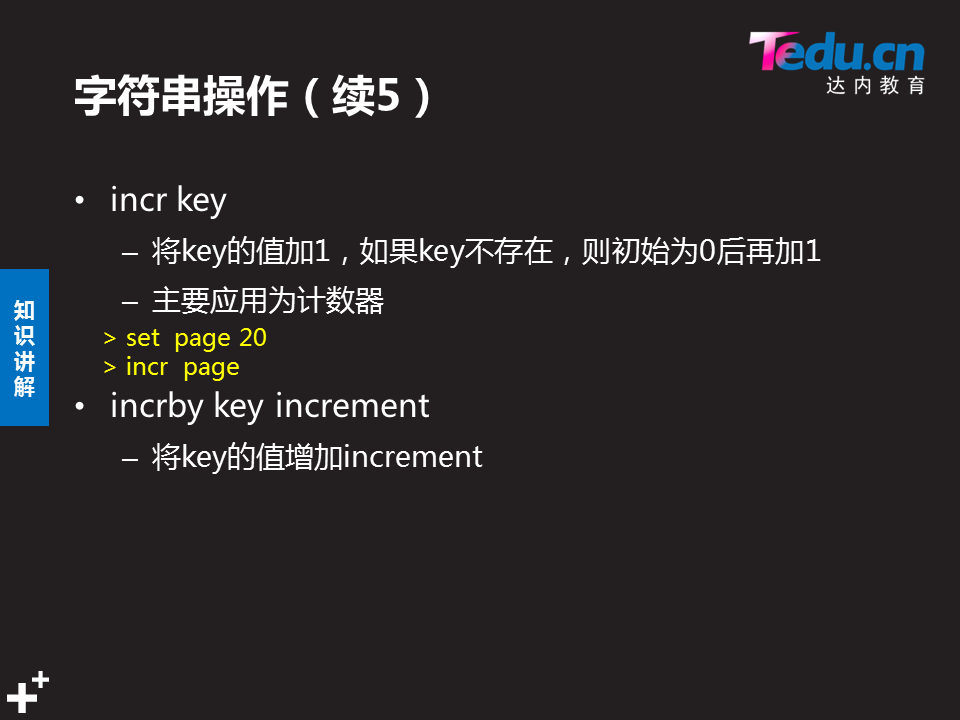
192.168.4.56:6356> getrange x 0 0

"a"

192.168.4.56:6356> getrange x 2 3

"cd"

### Incr和incrby命令(自加)



192.168.4.56:6356> set y 7

OK

192.168.4.56:6356> incr y #自加1

(integer) 8

192.168.4.56:6356> incr y

(integer) 9

192.168.4.56:6356> incr y 3

(error) ERR wrong number of arguments for 'incr' command

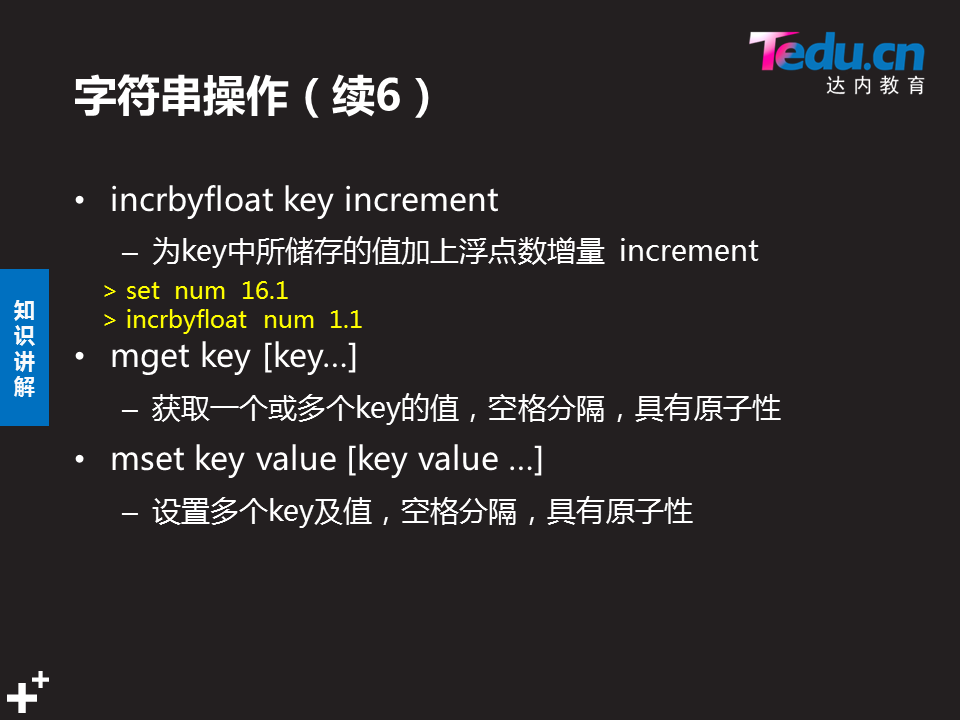
192.168.4.56:6356> incrby y 3 #自加3

(integer) 12

192.168.4.56:6356> incrby y -2

(integer) 10

### Incrbyfloat 命令(自加浮点数)



192.168.4.56:6356> incrbyfloat y 0.2

"10.2"

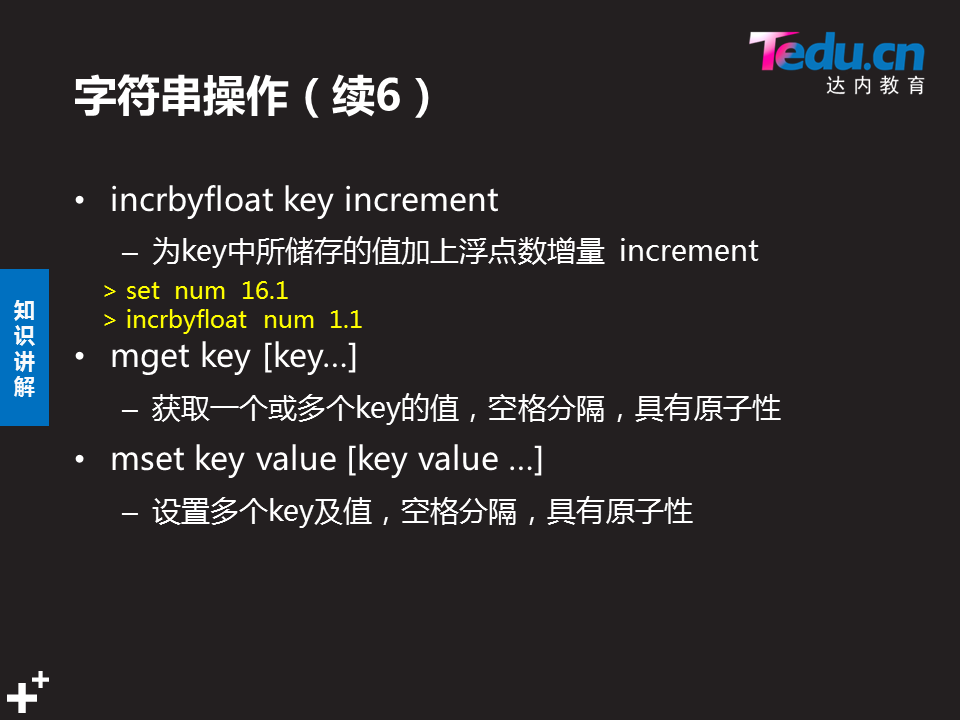
192.168.4.56:6356> get y

"10.2"

192.168.4.56:6356> incrbyfloat y 0.3

"10.5"

### mset(批量赋值)与 mget (批量查看)命令



192.168.4.56:6356> set a1 1

OK

192.168.4.56:6356> set a2 2

OK

192.168.4.56:6356> mget a1 a2

1) "1"

2) "2"

192.168.4.56:6356> mset b1 1 b2 2 b3 3

OK

192.168.4.56:6356> mget b1 b2 b3

1) "1"

2) "2"

3) "3"

## 4 List列表类型

* Redis的list是一个字符队列
* 先进后出
* 一个key可以存多个赋值

### lpush命令(列表赋值)

192.168.4.56:6356> lpush key value [value ...]

192.168.4.56:6356> lpush name a1 a2 a3 a4 a5

(integer) 5

192.168.4.56:6356> lpush name a6 a7

(integer) 7

### lrange命令(输出列表值)

192.168.4.56:6356> lrange key start stop

192.168.4.56:6356> lrange name

(error) ERR wrong number of arguments for 'lrange' command

192.168.4.56:6356> lrange name 0 -1

1) "a7"

2) "a6"

3) "a5"

4) "a4"

5) "a3"

6) "a2"

7) "a1"

192.168.4.56:6356> lrange name 2 3

1) "a5"

2) "a4"

192.168.4.56:6356> lrange name -2 -1

1) "a2"

2) "a1"

### lpop 命令(移除并返回头元素数据)

移除并返回列表头元素数据,key不存在则返回nil

192.168.4.56:6356> lpop bame

(nil)

192.168.4.56:6356> lpop name

"a7"

192.168.4.56:6356> lrange name 0 -1

1) "a6"

2) "a5"

3) "a4"

4) "a3"

5) "a2"

6) "a1"

### llen 命令(返回key长度)

返回列表key的长度

192.168.4.56:6356> llen name

(integer) 6

### lindex 命令(返回指定位置值)

返回列表中第index个值

192.168.4.56:6356> lindex key index

192.168.4.56:6356> lindex name 2

"a4"

192.168.4.56:6356> lindex name 7

(nil)

192.168.4.56:6356> lindex name 6

(nil)

192.168.4.56:6356> lindex name 5

"a1"

192.168.4.56:6356> lindex name 4

"a2"

### lset命令(修改指定位置的值)

192.168.4.56:6356> lset key index value

将key中index位置的值修改为value

192.168.4.56:6356> lset name 5 a111

OK

192.168.4.56:6356> lrange name 0 -1

1) "a6"

2) "a5"

3) "a4"

4) "a3"

5) "a2"

6) "a111"

### rpush 命令(末尾插入值)

将value插入到key的末尾

192.168.4.56:6356> rpush key value [value ...]

192.168.4.56:6356> rpush name a7

(integer) 7

192.168.4.56:6356> rpush name a8

(integer) 8

192.168.4.56:6356> lrange name 0 -1

1) "a6"

2) "a5"

3) "a4"

4) "a3"

5) "a2"

6) "a111"

7) "a7"

8) "a8"

### rpop命令(删除末尾的值)

删除并返回key末尾的值

192.168.4.56:6356> rpop key

192.168.4.56:6356> rpop name

"a8"

192.168.4.56:6356> rpop name

"a7"

192.168.4.56:6356> lrange name 0 -1

1) "a6"

2) "a5"

3) "a4"

4) "a3"

5) "a2"

6) "a111"

## Hash表类型

Redis hash

* 是一个string类型的field(字段,列)和value(值)的映射表
* 一个key可对应多个field(列).一个field对应一个value
* 将一个对象存储为hash类型,较于每个字段都存储成string类型.更能节省内存

### hset命令(设置值)

将hash表中field值设置为value

192.168.4.56:6356> hset key field value

例如存储一个公司信息

公司名 tarena

地址 address beijing

电话 tel 888888

法人 person hsy

注册资金 pay 100000

192.168.4.56:6356> hset tarena address beijing

(integer) 1

192.168.4.56:6356> hset tarena tel 888888

(integer) 1

### hget 命令(输出值)

获取hash表中field值

192.168.4.56:6356> hget key field

192.168.4.56:6356> hget tarena address

"beijing"

192.168.4.56:6356> hget tarena tel

"888888"

### hmset 命令(给多个field赋值)

同时给hash表中的多个field赋值

192.168.4.56:6356> hmset key field value [field value ...]

192.168.4.56:6356> hmset tarena person hsy pay 100000

OK

### hmget命令(同时输出多个值)

返回hash表中多个field的值

192.168.4.56:6356> hmget key field [field ...]

192.168.4.56:6356> hmget tarena address tel person

1) "beijing"

2) "888888"

3) "hsy"

### hkeys 命令(显示所有field名称)

返回hash表中所有field名称

192.168.4.56:6356> hkeys key

192.168.4.56:6356> hkeys tarena

1) "address"

2) "tel"

3) "person"

4) "pay"

### hgetall 命令(显示所有field及对应的值)

返回hash表中所有key名和对应的值列表

192.168.4.56:6356> hgetall key

192.168.4.56:6356> hgetall tarena

1) "address"

2) "beijing"

3) "tel"

4) "888888"

5) "person"

6) "hsy"

7) "pay"

8) "100000

#### hvals 命令(显示key中所有的值)

返回hash表中所有key的值,不显示field

192.168.4.56:6356> hvals key

192.168.4.56:6356> hvals tarena

1) "beijing"

2) "888888"

3) "hsy"

4) "100000"

### hdel 命令(删除多个field)

删除hash表中多个field的值,不存在则忽略

192.168.4.56:6356> hdel key field [field ...]

192.168.4.56:6356> hkeys tarena

1) "address"

2) "tel"

3) "person"

4) "pay"

192.168.4.56:6356> hdel tarena pay person

(integer) 2

192.168.4.56:6356> hkeys tarena

1) "address"

2) "tel"

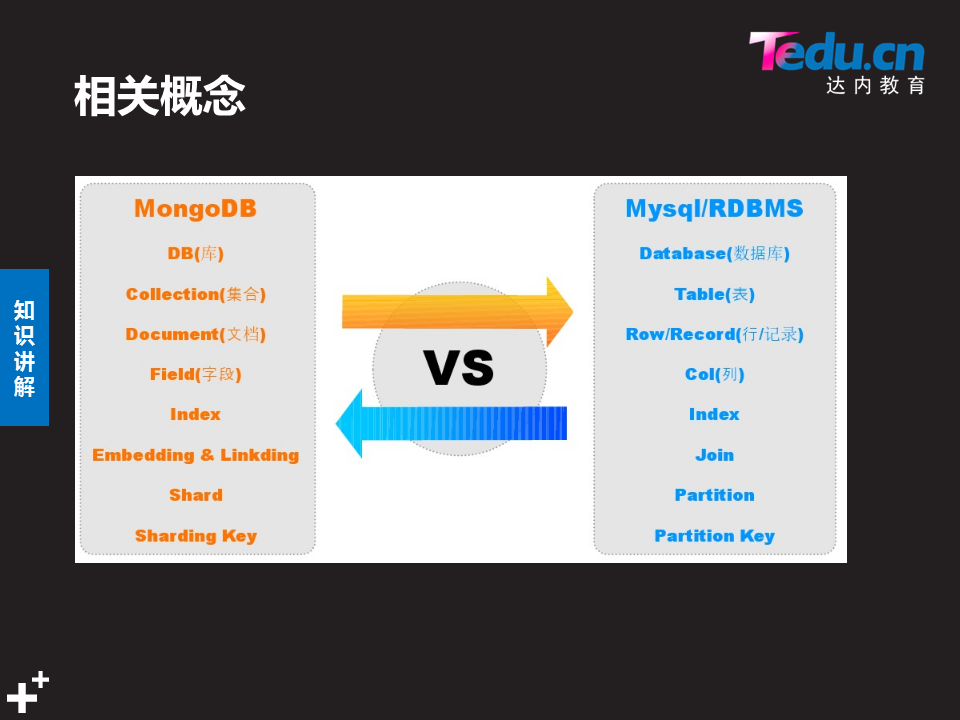
# NOSQL04--------------------------

# MongoDB概述

介于关系型数据库和非关系型数据库之间的产品

* 一款基于分布式文件存储的数据库,旨在为WEB应用提供可扩展的高性能数据存储解决方案
* 将数据存储为一个文档(类似于json对象),数据结构由键值(key=>value)对组成
* 支持丰富的查询表达,可以设置任何属性的索引
* 支持副本集(类似于主从),分片

相关概念



表称为集合

行记录称为文档

列称为字段

# 1搭建MDB服务器

## 1.1装包并创建工作目录

mongodb-linux-x86\_64-rhel70-3.6.3.tgz包是都已编译安装好的,解压即可使用

[root@client-50 ~]# tar -xf mongodb-linux-x86\_64-rhel70-3.6.3.tgz

[root@client-50 ~]# ls

mongodb-linux-x86\_64-rhel70-3.6.3

mongodb-linux-x86\_64-rhel70-3.6.3.tgz

[root@client-50 ~]# cd mongodb-linux-x86\_64-rhel70-3.6.3/

[root@client-50 mongodb-linux-x86\_64-rhel70-3.6.3]# ls

bin GNU-AGPL-3.0 MPL-2 README THIRD-PARTY-NOTICES

[root@client-50 ~]# mkdir /usr/local/mongodb

[root@client-50 ~]# mkdir /usr/local/mongodb/etc

[root@client-50 ~]# mkdir /usr/local/mongodb/log

[root@client-50 ~]# mkdir -p /usr/local/mongodb/data/db

[root@client-50 ~]# cp -r mongodb-linux-x86\_64-rhel70-3.6.3/bin /usr/local/mongodb/

[root@client-50 ~]# ls /usr/local/mongodb/

bin data etc log

[root@client-50 ~]# cd /usr/local/mongodb/bin/

[root@client-50 bin]# ls

bsondump mongod mongofiles mongoreplay mongostat

install\_compass mongodump mongoimport mongorestore mongotop

mongo mongoexport mongoperf mongos

## 1.2创建配置文件

[root@client-50 bin]# vim /usr/local/mongodb/etc/mongodb.conf

dbpath=/usr/local/mongodb/data/db #数据库目录

logpath=/usr/local/mongodb/log #日志目录

logappend=true #追加方式记录日志信息

fork=true #守护进程方式运行(无访问也在后台运行,等待访问)

bind\_ip=192.168.4.50 #指定连接地址

port=27050 #指定连接端口

## 1.3启动服务

[root@client-50 ~]# /usr/local/mongodb/bin/mongod -f /usr/local/mongodb/etc/mongodb.conf

about to fork child process, waiting until server is ready for connections.

forked process: 5411

child process started successfully, parent exiting

## 1.4停止服务

[root@client-50 ~]# /usr/local/mongodb/bin/mongod -f /usr/local/mongodb/etc/mongodb.conf --shutdown

killing process with pid: 5411

## 1.6查看服务

[root@client-50 ~]# netstat -unltp |grep 27017

tcp 0 0 127.0.0.1:27017 0.0.0.0:\* LISTEN 5411/mongod

[root@client-50 ~]# ps -C mongod

PID TTY TIME CMD

5411 ? 00:00:02 mongod

## 1.7设置别名方便启动停止服务

设置临时启动别名

[root@client-50 ~]# alias mdbstart='/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/etc/mongodb.conf'

设置临时停止别名

[root@client-50 ~]#alias mdbstop='/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/etc/mongodb.conf --shutdown'

[root@client-50 ~]# vim /root/.bashrc #编辑别名配置文件永久生效

# .bashrc

# User specific aliases and functions

alias mdbstart='/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/etc/mongodb.conf'

alias mdbstop='/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/etc/mongodb.conf --shutdown'

[root@client-50 ~]# mdbstop

killing process with pid: 5597

[root@client-50 ~]# mdbstart

about to fork child process, waiting until server is ready for connections.

forked process: 6082

child process started successfully, parent exiting

## 1.8连接服务存取数据

[root@client-50 ~]# /usr/local/mongodb/bin/mongo

MongoDB shell version v3.6.3

connecting to: mongodb://127.0.0.1:27017

MongoDB server version: 3.6.3

Welcome to the MongoDB shell.

For interactive help, type "help".

For more comprehensive documentation, see

http://docs.mongodb.org/

Questions? Try the support group

http://groups.google.com/group/mongodb-user

Server has startup warnings:

2019-03-02T15:51:35.634+0800 I CONTROL [initandlisten]

2019-03-02T15:51:35.634+0800 I CONTROL [initandlisten] \*\* WARNING:

........

2019-03-02T15:51:35.635+0800 I CONTROL [initandlisten]

>

> show dbs

admin 0.000GB

config 0.000GB

local 0.000GB

> exit

bye

修改了连接bin\_ip与port参数后连接方式

[root@client-50 ~]# /usr/local/mongodb/bin/mongo --host 192.168.4.50 --port 27050

# 2常用管理命令

## 2.1数据库管理命令

### show dbs查看已有的库

> show tables

> use gamedb

switched to db gamedb

> show tables

t1

t2

### db显示当前所在库

db

> db

test

### use 库名:切换库

库不存在,将延时创建库

use 库名

> db

test

> use gamedb

switched to db gamedb

> db

gamedb

### db.dropDatabase()删除当前所在库

db.dropDatabase()

## 2.2集合管理

### show tables查看库下已有集合

show collections 或者 show tables

> show tables

### db.t1.save创建集合

> db.t1.save({name:"bob",age:19}) #创建t1集合,并插入文档(行数据)

WriteResult({ "nInserted" : 1 })

> show tables #显示集合

t1

> db.t1.find() #查看t1集合

{ "\_id" : ObjectId("5c7a41fb99716aece7440d9b"), "name" : "bob", "age" : 19 }

> db.t1.save({name:"tom",age:19,school:"tarnea"})

WriteResult({ "nInserted" : 1 })

### show tables查看集合

> show tables

t1

### db.t2.drop()删除集合

> db.t2.save({name:"zdd",age:18})

WriteResult({ "nInserted" : 1 })

> show tables

t1

t2

> db.t2.drop()

true

> show tables

t1

## 2.3文档基本管理

> db.t1.find() #查看文档

> db.t1.count() #统计行个数

> db.t1.insert({name:”dachui”,age:18}) #添加插入文档

> db.t1.find({条件}) #按条件查看文档

> db.t1.findOne() #返回一行,查看第一行

> db.t1.remove({}) #删除所有行

> db.t1.remove({条件}) #按条件匹配删除行

### db.t1.save添加/插入文档

一次只能添加一条数据

> db.t1.save({name:"jim",age:90})

2019-03-02T16:58:23.623+0800 E QUERY [thread1] TypeError: db.t1save is not a function :

@(shell):1:1

> db.t1.save({name:"jim",age:90})

WriteResult({ "nInserted" : 1 })

> show tables

t1

### db.t1.insert添加/插入文档

一次只能添加一条

> db.t1.insert({name:"dachui",age:80}) #与db.t1.save功能一样

WriteResult({ "nInserted" : 1 })

### save与insert区别

save

集合不存在时创建集合,然后再插入记录

id字段已存在时,修改文档字段值

id字段不存在时,插入文档

insert

集合不存在时创建集合,然后再插入记录

id字段值已存在时,放弃插入

id字段值不存在时,插入文档

insertMany

### db.t1.insertMany添加/插入多条文档

rs1:PRIMARY> db.t1.insertMany([{name:"zdd2",age:19},{name:"zdd3",email:"zdd@tedu.cn"} ])

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("5c7cf16e2c5c3ca306aa8c93"),

ObjectId("5c7cf16e2c5c3ca306aa8c94")

]

}

rs1:PRIMARY> db.t1.find()

{ "\_id" : ObjectId("5c7cf09e2c5c3ca306aa8c8c"), "name" : "zdd" }

{ "\_id" : ObjectId("5c7cf0ee2c5c3ca306aa8c8d"), "name" : "zdd2", "age" : 19 }

{ "\_id" : ObjectId("5c7cf0ee2c5c3ca306aa8c8e"), "name" : "zdd3", "email" : "zdd@tedu.cn" }

### db.t1.find()查看文档

> db.t1.find()

{ "\_id" : ObjectId("5c7a41fb99716aece7440d9b"), "name" : "bob", "age" : 19 }

{ "\_id" : ObjectId("5c7a44cc99716aece7440d9c"), "name" : "tom", "age" : 19, "school" : "tarnea" }

{ "\_id" : ObjectId("5c7a45b599716aece7440d9e"), "name" : "jim", "age" : 90 }

{ "\_id" : ObjectId("5c7a45ef99716aece7440d9f"), "name" : "dachui", "age" : 80 }

### db.t1.find({条件})按条件查看文档

> db.t1.find({条件})

### db.t1.findOne查看第一行

> db.t1.findOne() #返回一行,查看第一行

### db.t1.count统计行个数

> db.t1.count()

4

### db.t1.remove删除行

> db.t1.remove({}) #删除所有行

> db.t1.remove({条件}) #按条件匹配删除行

# 3数据导入导出

导出:把集合里的文档保存到系统文件里

导入:把系统文件的内容导入到集合里

加--drop删除库中已有数据再导入,加-headrelin,忽略标题

## 3.1导出mongoexport:

把集合里的文档保存到系统文件里

csv格式:必须要-f指定字段导出

json格式:可以-f指定字段导出,也可以不用-f指定字段导出

csv格式一:

mongoexport [--host ip地址 --port 端口 ] -d 库名 -c 集合名 -f 字段名1,字段名2 --type=csv >目录名/文件名.csv

csv格式二:

mongoexport [--host ip地址 --port 端口 ] -d 库名 -c 集合名 -q ‘{条件}’ -f 字段名1,字段名2 --type=csv >目录名/文件名.csv

**导出的数据:**

> db.t1.find()

{ "\_id" : ObjectId("5c7a41fb99716aece7440d9b"), "name" : "bob", "age" : 19 }

{ "\_id" : ObjectId("5c7a44cc99716aece7440d9c"), "name" : "tom", "age" : 19, "school" : "tarnea" }

{ "\_id" : ObjectId("5c7a45b599716aece7440d9e"), "name" : "jim", "age" : 90 }

{ "\_id" : ObjectId("5c7a45ef99716aece7440d9f"), "name" : "dachui", "age" : 80 }

> db.t1.count()

4

### 不重定向到指定文件,将会在终端显示

[root@client-50 ~]# /usr/local/mongodb/bin/mongoexport --host 192.168.4.50 --port 27050 -d gamedb -c t1 -f name,age,school --type csv

2019-03-02T17:33:23.908+0800 connected to: 192.168.4.50:27050

name,age,school

bob,19, #没有school字段的内容就空白

tom,19,tarnea #有school字段的内容就输出,

jim,90,

dachui,80,

2019-03-02T17:33:23.910+0800 exported 4 records

### 创建指定文件目录和文档导出

[root@client-50 ~]# mkdir /mdbdata

[root@client-50 ~]# /usr/local/mongodb/bin/mongoexport --host 192.168.4.50 --port 27050 -d gamedb -c t1 -f name,age,school --type csv > /mdbdata/t1.csv

2019-03-02T17:35:46.201+0800 connected to: 192.168.4.50:27050

2019-03-02T17:35:46.202+0800 exported 4 records

[root@client-50 ~]# cat /mdbdata/t1.csv #查看已导出的数据

name,age,school

bob,19,

tom,19,tarnea

jim,90,

dachui,80,

**导出为json格式:**

json格式:可以-f指定字段导出,也可以不用-f指定字段导出

[root@client-50 ~]# /usr/local/mongodb/bin/mongoexport --host 192.168.4.50 --port 27050 -d gamedb -c t1 > /mdbdata/t1.json

2019-03-02T17:39:31.947+0800 connected to: 192.168.4.50:27050

2019-03-02T17:39:31.949+0800 exported 4 records

## 3.2 导入mongoimport:

把系统文件的内容导入到集合里

1. 导入数据时,若库和集合不存在,则现在创建库和集合后再导入数据;
2. 导入数据时.若库和集合已存在,则以追加的方式导入数据到集合里;
3. 使用--drop选项可以删除原有数据后导入新数据,--headerline 选项忽略标题

命令格式:

mongoimport [--host ip地址 --port 端口 ] -d 库名 -c 集合名

--type=csv 目录名/文件名.csv

> show tables

t1

t2

> db.t1.drop()

true

> show tables

t2

> exit

bye

### 导入json文件

**导入时不能加:**--headerline

[root@client-50 ~]# /usr/local/mongodb/bin/mongoimport --host 192.168.4.50 --port 27050 -d gamedb -c t1 --type=json /mdbdata/t1.json

2019-03-02T18:00:48.432+0800 connected to: 192.168.4.50:27050

2019-03-02T18:00:48.583+0800 imported 4 documents

> show tables

t1

t2

> db.t1.drop()

true

> show tables

t2

### 导入csv文件

**导入时一定要加:**--headerline

[root@client-50 ~]# /usr/local/mongodb/bin/mongoimport --host 192.168.4.50 --port 27050 -d gamedb -c t1 --headerline --type=csv /mdbdata/t1.csv

2019-03-02T17:57:37.630+0800 connected to: 192.168.4.50:27050

2019-03-02T17:57:37.765+0800 imported 4 documents #成功4条

#--headerline 用文件的第一行做字段名

[root@client-50 ~]# /usr/local/mongodb/bin/mongo --host 192.168.4.50 --port 27050

> use gamedb

switched to db gamedb

> show tables;

t1

t2

> db.t1.find()

{ "\_id" : ObjectId("5c7a539150d5efd7024e27e9"), "name" : "bob", "age" : 19, "school" : "" }

{ "\_id" : ObjectId("5c7a539150d5efd7024e27ea"), "name" : "tom", "age" : 19, "school" : "tarnea" }

{ "\_id" : ObjectId("5c7a539150d5efd7024e27eb"), "name" : "jim", "age" : 90, "school" : "" }

{ "\_id" : ObjectId("5c7a539150d5efd7024e27ec"), "name" : "dachui", "age" : 80, "school" : "" }

### 导入数据时加--drop

### 练习:

把/etc/passwd文件的内容存储到mongodb服务器的userdb库下的user集合里

1 把/etc/passwd文件的内容修改为csv格式

用sed工具,将所有的 : 号,换成 , 号,

用sed

# 4 基本数据类型

## 字符string

字符串string

utf-8字符串都可以表示为字符串类型的数据

例如:{name:”张三”}

布尔bool

只有两种值:true和false

例如:{x:true} {y:false}

空null

用于表示空值或者不存在的字段

例如:{x:null}

## 数值/数组array

数值

shell默认使用64位浮点型数值.

{x:3.14} 或者 {x:3}

numberint(4字节整数,32位整型)4个8位,每个8位255,存范围255^4

例如:{x:NumberInt(3)} {x:NumberInt(3.333)} #能插入浮点型数值

numberlong(8字节整数)64为长整型)8个8位,每个8位255,存范围255^8

例如:{x:NumberLong(3)} #不能插入浮点型数值

数组array

数据列表或数据集可以表示数组

例如:{x:[“a”,”b”,”c”]}

## 代码/日期/对象

**代码**

查询和文档中可以包括任何JavaScript代码

例如:{x:function(){/\* 代码 \*/}}

db.t2.save({lname:"php",daima:function(){/\* 代码 \*/}})

> db.t2.save({lname:"php",daima:function(){/\*<?php echo "welcome"; ?> \*/}})

WriteResult({ "nInserted" : 1 })

> db.t2.find({name:"php"})

> db.t2.find({lname:"php"})

{ "\_id" : ObjectId("5c7c932c9d042aa67a8a254c"), "lname" : "php", "daima" : { "code" : "function (){/\*<?php echo \"welcome\"; ?> \*/}" } }

**日期**

日期被存储为自新纪元(1970年1月1日)以来经过的毫秒数,不含时区

例如:{x:new Date()}

> db.t2.save({name:"tom",birthday:new Date()})

WriteResult({ "nInserted" : 1 })

> db.t2.find({name:"tom"})

{ "\_id" : ObjectId("5c7c8eae9d042aa67a8a2549"), "name" : "tom", "birthday" : ISODate("2019-03-04T02:34:22.477Z") }

**对象**

对象id是一个12字节的字符串,是文档的唯一标识

例如:{x:ObjectId()}

> db.t2.save(

... {

... name:"alice",

... x:ObjectId()

... }

... )

WriteResult({ "nInserted" : 1 })

> db.t2.find({name:"alice"})

{ "\_id" : ObjectId("5c7c90159d042aa67a8a254b"), "name" : "alice", "x" : ObjectId("5c7c90159d042aa67a8a254a") }

## 内嵌/正则表达式

**内嵌**

文档可以嵌套其他文档,被嵌套的文档作为值来处理

例如:{tarena:{address:”Beijing”,tel:”888888”,person:”hansy”}}

> db.t2.save({

... tarena:{address:"beijing",tel:"888888",worker:"ksy",url:"www.tedu.cn"}

... })

WriteResult({ "nInserted" : 1 })

> db.t2.find({tel:"888888"})

>

**正则表达式**

查询时,使用正则表达式作为限定条件

例如:{x:/正则表达式/}

> db.t2.save({gn:"匹配空行",bds:/^$/})

WriteResult({ "nInserted" : 1 })

> db.t2.find({gn:"匹配空行"})

{ "\_id" : ObjectId("5c7c943e9d042aa67a8a254e"), "gn" : "匹配空行", "bds" : /^$/ }

# 5 数据备份恢复

[root@client-50 ~]# ls /usr/local/mongodb/bin/

bsondump(查看) mongod mongofiles mongoreplay mongostat

install\_compass mongodump(备份) mongoimport mongorestore(恢复) mongotop

mongo mongoexport mongoperf mongos

## 4.1数据备份mongodump

备份数据所有库到当前目录下的dump目录下

mongodump [--host IP地址 --port 端口] #如果配置文件未指定ip和端口可省略

备份时指定备份的库和备份目录

mongodump [--host IP地址 --port 端口 ] -d 数据库名 -c 集合名 -o 存储目录

## -d 数据库名(不写全部备份) -c 集合名(不写备份全部集合) -o 存储目录

查看bson文件内容

bsondump ./dump/bbs/t1.bson

## 4.2数据恢复

mongorestore --host IP地址 --port 端口 -d 数据库名 -c 集合名 备份目录名

mongorestore --host IP地址 --port 端口 -d 数据库名 -c 集合名 --drop备份目录名

##--drop 删除原有相同名的库和相同名的集合后再在51上恢复

例如:50上备份userdb库,user文档

[root@client-50 ~]# /usr/local/mongodb/bin/mongodump --host 192.168.4.50 --port 27050 -d userdb -c user -o /userdbzdd

[root@client-50 ~]# ls /userdbzdd

userdb

[root@client-50 ~]# ls /userdbzdd/userdb/

user.bson user.metadata.json

[root@client-50 ~]#/usr/local/mongodb/bin/bsondump /userdbzdd/userdb/user.bson

[root@sql51 ~]# /usr/local/mongodb/bin/mongorestore --host 192.168.4.51 --port 27051 -d userdb -c user /root/userdbzdd/userdb/user.bson

测试 --drop

[root@sql51 ~]# /usr/local/mongodb/bin/mongorestore --host 192.168.4.51 --port 27051 -d userdb -c user /root/userdbzdd/userdb/user.bson

2019-03-04T14:10:34.796+0800 checking for collection data in /root/userdbzdd/userdb/user.bson

2019-03-04T14:10:34.799+0800 reading metadata for userdb.user from /root/userdbzdd/userdb/user.metadata.json

2019-03-04T14:10:34.799+0800 restoring userdb.user from /root/userdbzdd/userdb/user.bson

2019-03-04T14:10:34.813+0800 error: multiple errors in bulk operation:

- E11000 duplicate key error collection: userdb.user index: \_id\_ dup key: { : ObjectId('5c7c80bd3345a27478ee24a9') }

- E11000 duplicate key error collection: userdb.user index: \_id\_ dup key: { : ObjectId('5c7c80bd3345a27478ee24aa') }

[root@sql51 ~]# /usr/local/mongodb/bin/mongorestore --host 192.168.4.51 --port 27051 -d userdb -c user --drop /root/userdbzdd/userdb/user.bson

2019-03-04T14:11:10.137+0800 checking for collection data in /root/userdbzdd/userdb/user.bson

2019-03-04T14:11:10.170+0800 reading metadata for userdb.user from /root/userdbzdd/userdb/user.metadata.json

2019-03-04T14:11:10.319+0800 restoring userdb.user from /root/userdbzdd/userdb/user.bson

2019-03-04T14:11:10.383+0800 no indexes to restore

2019-03-04T14:11:10.383+0800 finished restoring userdb.user (42 documents)

2019-03-04T14:11:10.383+0800 done

# NOSQL05--------------------------

# 一MongoDB副本集

* 也称为MongoDB复制
* 指在多个服务器上存储数据副本,并实现数据同步
* 提高数据可用性,安全性,方便数据故障恢复

## MongoDB复制原理

* 也就是副本集工作过程
* 至少需要两个节点,其中一个为主节点,负责处理客户端请求,其余是从节点,负责复制主节点数据
* 常见搭配:一主一从,一主多从
* 主节点记录所有操作oplog,从节点定期轮询主节点获取这些操作,然后对自己的数据副本执行这些操作,从而保证从节点的数据与主节点一致

主从复制的优点

* 从节点可以提供数据查询,降低主节点的访问压力
* 由从节点执行备份,避免锁定主节点数据.
* 当主节点故障时,可快速切换到从节点,实现高可用

## 副本集实现方式

master-slave主从复制

启动一台服务器时加上”-master”参数,作为主节点

启动其他服务器时加上”-slave”和”-source”参数,作为从节点

## 副本集相关命令

查看状态信息

>rs.status()

查看是否是master库

>rs.isMaster()

初始化replica sets环境

>rs.initiate(config)

同步数据验证,允许从库查看数据

# 二 配置replica sets(副本集)

建立集群后,从库是不允许做数据操作

## 1准备环境3台mongodb服务器

192.168.4.51

192.168.4.52

192.168.4.53

## 2启用副本集配置

### 1停止所有服务

**#启动和停止服务都定义了别名:**

~]# alias mdbstart='/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/etc/mongodb.conf'

设置临时停止别名

~]#alias mdbstop='/usr/local/mongodb/bin/mongod -f /usr/local/mongodb/etc/mongodb.conf --shutdown'

[root@sql51 ~]# mdbstop

[root@sql52 ~]# mdbstop

[root@sql53 ~]# mdbstop

### 2所有服务器同步修改配置文件

分别在3台服务器上的配置文件上添加 replSet=rs1

[root@sql51 ~]# vim /usr/local/mongodb/etc/mongodb.conf

replSet=rs1

[root@sql51 ~]# mdbstart

[root@sql51 ~]# /usr/local/mongodb/bin/mongo --host 192.168.4.51 --port 27051

配置之前:

> rs.status()

{

"ok" : 0,

"errmsg" : "not running with --replSet",

"code" : 76,

"codeName" : "NoReplicationEnabled"

}

配置之后

> rs.status()

{

"info" : "run rs.initiate(...) if not yet done for the set",

"ok" : 0,

"errmsg" : "no replset config has been received",

"code" : 94,

"codeName" : "NotYetInitialized",

"$clusterTime" : {

"clusterTime" : Timestamp(0, 0),

"signature" : {

"hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),

"keyId" : NumberLong(0)

}

}

}

[root@sql52 ~]# vim /usr/local/mongodb/etc/mongodb.conf

replSet=rs1

[root@sql52 ~]# mdbstart

[root@sql53 ~]# vim /usr/local/mongodb/etc/mongodb.conf

replSet=rs1

[root@sql53 ~]# mdbstart

## 3定义集群成员(配置节点信息)

进入51(主库)

> config= { \_id:"rs1",members:[ #定义config变量

... {\_id:0,host:"192.168.4.51:27051"},

... {\_id:1,host:"192.168.4.52:27052"},

... {\_id:2,host:"192.168.4.53:27053"}

... ]}

#回车出现如下内容

{

"\_id" : "rs1",

"members" : [

{

"\_id" : 0,

"host" : "192.168.4.51:27051"

},

{

"\_id" : 1,

"host" : "192.168.4.52:27052"

},

{

"\_id" : 2,

"host" : "192.168.4.53:27053"

}

]

}

## 4初始化replica sets环境

**51上操作**

> rs.initiate(config)

{

"ok" : 1,

"operationTime" : Timestamp(1551687278, 1),

"$clusterTime" : {

"clusterTime" : Timestamp(1551687278, 1),

"signature" : {

"hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),

"keyId" : NumberLong(0)

}

}

}

rs1:SECONDARY>

rs1:PRIMARY> #PRIMARY为主库信息

## 5查看主从信息

### 主从状态

rs1:PRIMARY> rs.status #

function () {

return db.\_adminCommand("replSetGetStatus");

}

rs1:PRIMARY> rs.status()

{

"set" : "rs1",

"date" : ISODate("2019-03-04T08:32:58.039Z"),

"myState" : 1,

"term" : NumberLong(1),

"heartbeatIntervalMillis" : NumberLong(2000),

"optimes" : {

"lastCommittedOpTime" : {

"ts" : Timestamp(1551688370, 1),

"t" : NumberLong(1)

},

"readConcernMajorityOpTime" : {

"ts" : Timestamp(1551688370, 1),

"t" : NumberLong(1)

},

"appliedOpTime" : {

"ts" : Timestamp(1551688370, 1),

"t" : NumberLong(1)

},

"durableOpTime" : {

"ts" : Timestamp(1551688370, 1),

"t" : NumberLong(1)

}

},

"members" : [

{

"\_id" : 0,

"name" : "192.168.4.51:27051",

"health" : 1,

"state" : 1,

"stateStr" : "PRIMARY", #主库

"uptime" : 1176,

"optime" : {

"ts" : Timestamp(1551688370, 1),

"t" : NumberLong(1)

},

"optimeDate" : ISODate("2019-03-04T08:32:50Z"),

"electionTime" : Timestamp(1551687289, 1),

"electionDate" : ISODate("2019-03-04T08:14:49Z"),

"configVersion" : 1,

"self" : true

},

{

"\_id" : 1,

"name" : "192.168.4.52:27052",

"health" : 1,

"state" : 2,

"stateStr" : "SECONDARY", #从库

"uptime" : 1099,

"optime" : {

"ts" : Timestamp(1551688370, 1),

"t" : NumberLong(1)

},

"optimeDurable" : {

"ts" : Timestamp(1551688370, 1),

"t" : NumberLong(1)

},

"optimeDate" : ISODate("2019-03-04T08:32:50Z"),

"optimeDurableDate" : ISODate("2019-03-04T08:32:50Z"),

"lastHeartbeat" : ISODate("2019-03-04T08:32:56.625Z"),

"lastHeartbeatRecv" : ISODate("2019-03-04T08:32:56.588Z"),

"pingMs" : NumberLong(0),

"syncingTo" : "192.168.4.51:27051",

"configVersion" : 1

},

{

"\_id" : 2,

"name" : "192.168.4.53:27053",

"health" : 1,

"state" : 2,

"stateStr" : "SECONDARY", #从库

"uptime" : 1099,

"optime" : {

"ts" : Timestamp(1551688370, 1),

"t" : NumberLong(1)

},

"optimeDurable" : {

"ts" : Timestamp(1551688370, 1),

"t" : NumberLong(1)

},

"optimeDate" : ISODate("2019-03-04T08:32:50Z"),

"optimeDurableDate" : ISODate("2019-03-04T08:32:50Z"),

"lastHeartbeat" : ISODate("2019-03-04T08:32:56.619Z"),

"lastHeartbeatRecv" : ISODate("2019-03-04T08:32:56.589Z"),

"pingMs" : NumberLong(0),

"syncingTo" : "192.168.4.51:27051",

"configVersion" : 1

}

],

"ok" : 1,

"operationTime" : Timestamp(1551688370, 1),

"$clusterTime" : {

"clusterTime" : Timestamp(1551688370, 1),

"signature" : {

"hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),

"keyId" : NumberLong(0)

}

}

}

### 查看是否是master库

rs1:SECONDARY> rs.isMaster()

{

"hosts" : [

"192.168.4.51:27051",

"192.168.4.52:27052",

"192.168.4.53:27053"

],

"setName" : "rs1",

"setVersion" : 1,

"ismaster" : false, #false为不是主,true为是主库

"secondary" : true, #为从库

"primary" : "192.168.4.51:27051", #数据来源主库地址

"me" : "192.168.4.52:27052", #自己的地址

"lastWrite" : {

"opTime" : {

"ts" : Timestamp(1551688600, 1),

"t" : NumberLong(1)

},

"lastWriteDate" : ISODate("2019-03-04T08:36:40Z"),

"majorityOpTime" : {

"ts" : Timestamp(1551688600, 1),

"t" : NumberLong(1)

},

"majorityWriteDate" : ISODate("2019-03-04T08:36:40Z")

},

"maxBsonObjectSize" : 16777216,

"maxMessageSizeBytes" : 48000000,

"maxWriteBatchSize" : 100000,

"localTime" : ISODate("2019-03-04T08:36:49.473Z"),

"logicalSessionTimeoutMinutes" : 30,

"minWireVersion" : 0,

"maxWireVersion" : 6,

"readOnly" : false,

"ok" : 1,

"operationTime" : Timestamp(1551688600, 1),

"$clusterTime" : {

"clusterTime" : Timestamp(1551688600, 1),

"signature" : {

"hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),

"keyId" : NumberLong(0)

}

}

}

## 6验证主从配置

### 1.验证数据同步

主库51上操作

rs1:PRIMARY> show databases;

admin 0.000GB

config 0.000GB

local 0.000GB

rs1:PRIMARY> use buydb

switched to db buydb

rs1:PRIMARY> db.atab.save({name:"bob"})

WriteResult({ "nInserted" : 1 })

rs1:PRIMARY> db.atab.find()

{ "\_id" : ObjectId("5c7ce550da9f0fda3bedc1e8"), "name" : "bob" }

rs1:PRIMARY> show databases

admin 0.000GB

buydb 0.000GB

config 0.000GB

local 0.000GB

### 2 同步数据验证,允许从库查看数据

未做此步,无法在从库上查询数据

在50上查看51

[root@client-50 ~]# /usr/local/mongodb/bin/mongo --host 192.168.4.52 --port 27052

rs1:SECONDARY> #从库表示

rs1:SECONDARY> db.getMongo().setSlaveOk()

#同步数据验证,允许从库查看数据,注意大小写Ok:O为大写,k为小写

rs1:SECONDARY> show databases admin 0.000GB

buydb 0.000GB

config 0.000GB

local 0.000GB

rs1:SECONDARY> use buydb

switched to db buydb

rs1:SECONDARY> show tables

atab

rs1:SECONDARY> db.atab.find()

{ "\_id" : ObjectId("5c7ce550da9f0fda3bedc1e8"), "name" : "bob" }

### 3验证高可用

把当前主库51的服务停止,会自动在副本主机里选择一个新主机做主库

51恢复,启动服务自动边从库

# 三 去掉集群

## 成为从库之后

被配置成从库之后,无法对数据进行操作

> show databases

2019-03-04T15:53:35.389+0800 E QUERY [thread1] Error: listDatabases failed:{

"ok" : 0,

"errmsg" : "not master and slaveOk=false",

"code" : 13435,

"codeName" : "NotMasterNoSlaveOk",

"$clusterTime" : {

"clusterTime" : Timestamp(0, 0),

"signature" : {

"hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),

"keyId" : NumberLong(0)

}

}

} :

\_getErrorWithCode@src/mongo/shell/utils.js:25:13

Mongo.prototype.getDBs@src/mongo/shell/mongo.js:65:1

shellHelper.show@src/mongo/shell/utils.js:816:19

shellHelper@src/mongo/shell/utils.js:706:15

@(shellhelp2):1:1

# 三 去掉集群步骤

## 停止数据库服务

## 修改配置文件去掉replSet=rs1

## 启动数据库,进入删库

进入local库 db.dropDatabase()

进入config库 db.dropDatabase()

> use local

switched to db local

> db.dropDatabase()

{ "dropped" : "local", "ok" : 1 }

> use config

switched to db config

> db.dropDatabase()

{ "dropped" : "config", "ok" : 1 }

## 退出停止数据库重启数据库

[root@sql51 ~]# mdbstop

[root@sql51 ~]# mdbstart

# 四 文档管理

## 添加插入文档

### 插入单条记录

save与insert区别

save

集合不存在时创建集合,然后再插入记录

id字段已存在时,修改文档字段值

id字段不存在时,插入文档

insert

集合不存在时创建集合,然后再插入记录

id字段值已存在时,放弃插入

id字段值不存在时,插入文档

### 插入多条数据

insertMany

能插入多条记录

rs1:PRIMARY> db.t1.insertMany([{name:"zdd2",age:19},{name:"zdd3",email:"zdd@tedu.cn"} ])

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("5c7cf16e2c5c3ca306aa8c93"),

ObjectId("5c7cf16e2c5c3ca306aa8c94")

]

}

rs1:PRIMARY> db.t1.find()

{ "\_id" : ObjectId("5c7cf09e2c5c3ca306aa8c8c"), "name" : "zdd" }

{ "\_id" : ObjectId("5c7cf0ee2c5c3ca306aa8c8d"), "name" : "zdd2", "age" : 19 }

{ "\_id" : ObjectId("5c7cf0ee2c5c3ca306aa8c8e"), "name" : "zdd3", "email" : "zdd@tedu.cn" }

## 查询文档

### db.t1.findOne()查看第一行

> db.t1.findOne() #返回一行,查看第一行

### db.t1.find()查看所有

rs1:PRIMARY> db.t1.find()

{ "\_id" : ObjectId("5c7cf09e2c5c3ca306aa8c8c"), "name" : "zdd" }

{ "\_id" : ObjectId("5c7cf0ee2c5c3ca306aa8c8d"), "name" : "zdd2", "age" : 19 }

{ "\_id" : ObjectId("5c7cf0ee2c5c3ca306aa8c8e"), "name" : "zdd3", "email" : "zdd@tedu.cn" }

{ "\_id" : ObjectId("5c7cf1322c5c3ca306aa8c8f"), "name" : "zdd2", "age" : 19 }

{ "\_id" : ObjectId("5c7cf1322c5c3ca306aa8c90"), "name" : "zdd3", "email" : "zdd@tedu.cn" }

{ "\_id" : ObjectId("5c7cf1602c5c3ca306aa8c91"), "name" : "zdd2", "age" : 19 }

{ "\_id" : ObjectId("5c7cf1602c5c3ca306aa8c92"), "name" : "zdd3", "email" : "zdd@tedu.cn" }

{ "\_id" : ObjectId("5c7cf16e2c5c3ca306aa8c93"), "name" : "zdd2", "age" : 19 }

{ "\_id" : ObjectId("5c7cf16e2c5c3ca306aa8c94"), "name" : "zdd3", "email" : "zdd@tedu.cn" }

### db.t1.find().limit() 查看指定行数

rs1:PRIMARY> db.t1.find().limit(3)

{ "\_id" : ObjectId("5c7cf09e2c5c3ca306aa8c8c"), "name" : "zdd" }

{ "\_id" : ObjectId("5c7cf0ee2c5c3ca306aa8c8d"), "name" : "zdd2", "age" : 19 }

{ "\_id" : ObjectId("5c7cf0ee2c5c3ca306aa8c8e"), "name" : "zdd3", "email" : "zdd@tedu.cn" }

### db.t1.find({},{条件}).limit()指定不查看的内容

> db.user.find().limit(2)

{ "\_id" : ObjectId("5c7c80bd3345a27478ee24a9"), "name" : "root", "password" : "x", "uid" : 0, "gid" : 0, "comment" : "root", "homedir" : "/root", "shell" : "/bin/bash" }

{ "\_id" : ObjectId("5c7c80bd3345a27478ee24aa"), "name" : "bin", "password" : "x", "uid" : 1, "gid" : 1, "comment" : "bin", "homedir" : "/bin", "shell" : "/sbin/nologin" }

> db.user.find({},{\_id:0}).limit(2)

{ "name" : "root", "password" : "x", "uid" : 0, "gid" : 0, "comment" : "root", "homedir" : "/root", "shell" : "/bin/bash" }

{ "name" : "bin", "password" : "x", "uid" : 1, "gid" : 1, "comment" : "bin", "homedir" : "/bin", "shell" : "/sbin/nologin" }

> db.user.find({},{\_id:0,name:1}).limit(1)

{ "name" : "root" }

> db.user.find({},{\_id:0,password:0}).limit(2)

### db.t1.find({条件})按条件查看文档

> db.t1.find({条件})

> db.user.find({shell:"/sbin/nologin"}).limit(1)

{ "\_id" : ObjectId("5c7c80bd3345a27478ee24aa"), "name" : "bin", "password" : "x", "uid" : 1, "gid" : 1, "comment" : "bin", "homedir" : "/bin", "shell" : "/sbin/nologin" }

> db.user.find({shell:"/sbin/nologin"}{\_id:0}).limit(1)

2019-03-04T17:55:05.377+0800 E QUERY [thread1] SyntaxError: missing ) after argument list @(shell):1:36

### db.t1.find().skip(2) 跳过前2行开始显示

### db.t1.find().sort(age:1|-1)升序降序

db.t1.find().sort(age:1)升序

db.t1.find().sort(age:-1)降序

# 查询匹配条件

### 数值比较

大于 $gt

大于等于 $gte

不相等 $ne

小于等于 $lte

小于 $lt

> db.user.find({uid:{$gt:10}},{\_id:0,uid:1,name:1})

{ "name" : "operator", "uid" : 11 }

{ "name" : "games", "uid" : 12 }

......

Type "it" for more #

> db.user.find({uid:{$gt:10,$lte:30}},{\_id:0,uid:1,name:1})

{ "name" : "operator", "uid" : 11 }

{ "name" : "games", "uid" : 12 }

{ "name" : "ftp", "uid" : 14 }

{ "name" : "rpcuser", "uid" : 29 }

{ "name" : "mysql", "uid" : 27 }

> db.user.find({uid:{$gt:10,$lte:30}},{\_id:0,uid:1,name:1}).count()

5

### 字符比较

> db.user.find({name:{$nin:["adm"]}},{\_id:0,name:1,shell:1})

{ "name" : "root", "shell" : "/bin/bash" }

{ "name" : "bin", "shell" : "/sbin/nologin" }

### 范围内匹配

$in 在...里

$nin 不在..里

$or 或

> db.user.find({uid:{$in:[1,6,9]}},{\_id:0,uid:1,name:1})

{ "name" : "bin", "uid" : 1 }

{ "name" : "shutdown", "uid" : 6 }

> db.user.find({name:{$in:["mysql","mysql"]}},{\_id:0,name:1,uid:1})

{ "name" : "mysql", "uid" : 27 }

> db.user.find({name:{$in:["mysql","apache"]}},{\_id:0,name:1,uid:1})

{ "name" : "mysql", "uid" : 27 }

{ "name" : "apache", "uid" : 48 }

> db.user.find({name:{$in:["mysql","apache","bin"]}},{\_id:0,name:1,uid:1})

{ "name" : "bin", "uid" : 1 }

{ "name" : "mysql", "uid" : 27 }

{ "name" : "apache", "uid" : 48 }

> db.user.find({shell:{$nin:["/bin/bash","/sbin/nologin"]}},{\_id:0,name:1,shell:1})

{ "name" : "sync", "shell" : "/bin/sync" }

{ "name" : "shutdown", "shell" : "/sbin/shutdown" }

{ "name" : "halt", "shell" : "/sbin/halt" }

{ "name" : "mysql", "shell" : "/bin/false" }

> db.user.find({or:{$nin:["/bin/bash","/sbin/nologin"]}},{\_id:0,name:1,shell:1})

{ "name" : "root", "shell" : "/bin/bash" }

{ "name" : "bin", "shell" : "/sbin/nologin" }

{ "name" : "daemon", "shell" : "/sbin/nologin" }

{ "name" : "adm", "shell" : "/sbin/nologin" }

### 正则匹配

> db.user.find({name:/^...$/},{\_id:0,name:1,shell:1})

{ "name" : "bin", "shell" : "/sbin/nologin" }

{ "name" : "adm", "shell" : "/sbin/nologin" }

{ "name" : "ftp", "shell" : "/sbin/nologin" }

### 相等比较 字段名:值

> db.user.find({shell:"/sbin/nologin"},{\_id:0,name:1,shell:1})

{ "name" : "bin", "shell" : "/sbin/nologin" }

{ "name" : "daemon", "shell" : "/sbin/nologin" }

{ "name" : "adm", "shell" : "/sbin/nologin" }

> db.user.find({uid:1},{\_id:0,name:1,shell:1})

{ "name" : "bin", "shell" : "/sbin/nologin" }

> db.user.find({uid:1},{\_id:0,name:1,shell:1,uid:1})

{ "name" : "bin", "uid" : 1, "shell" : "/sbin/nologin" }

### 逻辑比较

> db.user.find({name:"adm",id:null},{\_id:0,name:1,shell:1})

{ "name" : "adm", "shell" : "/sbin/nologin" }

### 匹配空null

> db.user.find({name:null},{\_id:0,name:1,shell:1})

{ "name" : null }

-------------

## 更新文档

### update()

db.集合名.update({条件},{修改的字段:值,修改的字段:值db})

语法格式:默认只更新与条件匹配的第1行

$set #只更新指定的字段值

$unset #删除与条件匹配文档的字段

**修改之前:uid:0的行的数据**

> db.user.find({uid:0})

{ "\_id" : ObjectId("5c7c80bd3345a27478ee24a9"), "name" : "root", "password" : "x", "uid" : 0, "gid" : 0, "comment" : "root", "homedir" : "/root", "shell" : "/bin/bash" }

**未加$set修改后再查看**

> db.user.update({uid:0},{name:"root",password:"A",uid:0})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({uid:0})

{ "\_id" : ObjectId("5c7c80bd3345a27478ee24a9"), "name" : "root", "password" : "A", "uid" : 0 }

**加了$set修改后再查看**

> db.user.update({uid:0},{$set:{name:"rootB",password:"B"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({uid:0})

{ "\_id" : ObjectId("5c7c80bd3345a27478ee24a9"), "name" : "rootB", "password" : "B", "uid" : 0 }

**$unset删除字段操作**

{ "\_id" : ObjectId("5c7c80bd3345a27478ee24a9"), "name" : "rootB", "password" : "A", "uid" : 0 }

> db.user.update({uid:0},{$unset:{password:"A"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({uid:0})

{ "\_id" : ObjectId("5c7c80bd3345a27478ee24a9"), "name" : "rootB", "uid" : 0 }

**set添加字段**

> db.user.update({uid:0},{$set:{password:"A"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({uid:0})

{ "\_id" : ObjectId("5c7c80bd3345a27478ee24a9"), "name" : "rootB", "uid" : 0, "password" : "A" }

> db.user.update({uid:0},{$set:{xz1:"B",xz2:"C"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({uid:0})

{ "\_id" : ObjectId("5c7c80bd3345a27478ee24a9"), "name" : "rootB", "uid" : 0, "password" : "A", "xz1" : "B", "xz2" : "C" }

## 多文档更新

语法格式:默认只更新与条件匹配的第1行,要修改所有[匹配的行需要加”false,true”

db.user.update({条件},{$set:{修改的字段}},false,true)

> db.user.find({uid:{$lte:5}},{\_id:0,name:1,password:1})

{ "name" : "rootB", "password" : "B" }

{ "name" : "bin", "password" : "x" }

{ "name" : "daemon", "password" : "x" }

{ "name" : "adm", "password" : "x" }

{ "name" : "lp", "password" : "x" }

{ "name" : "sync", "password" : "x" }

未加”false,true”修改

> db.user.update({uid:{$lte:5}},{$set:{password:"A"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({uid:{$lte:5}},{\_id:0,name:1,password:1})

{ "name" : "rootB", "password" : "A" }

{ "name" : "bin", "password" : "x" }

{ "name" : "daemon", "password" : "x" }

{ "name" : "adm", "password" : "x" }

{ "name" : "lp", "password" : "x" }

{ "name" : "sync", "password" : "x" }

加”false,true”修改

> db.user.update({uid:{$lte:5}},{$set:{password:"A"}},false,true)

WriteResult({ "nMatched" : 6, "nUpserted" : 0, "nModified" : 5 })

> db.user.find({uid:{$lte:5}},{\_id:0,name:1,password:1})

{ "name" : "rootB", "password" : "A" }

{ "name" : "bin", "password" : "A" }

{ "name" : "daemon", "password" : "A" }

{ "name" : "adm", "password" : "A" }

{ "name" : "lp", "password" : "A" }

{ "name" : "sync", "password" : "A" }

### 字段值自加自减

> db.user.find({uid:{$lte:5}},{\_id:0,name:1,uid:1})

{ "name" : "rootB", "uid" : 0 }

{ "name" : "bin", "uid" : 1 }

{ "name" : "daemon", "uid" : 2 }

{ "name" : "adm", "uid" : 3 }

{ "name" : "lp", "uid" : 4 }

{ "name" : "sync", "uid" : 5 }

**自加2--- $inc**

> db.user.update({uid:{$lte:5}},{$inc:{uid:2}},false,true)

WriteResult({ "nMatched" : 6, "nUpserted" : 0, "nModified" : 6 })

> db.user.find({uid:{$lte:7}},{\_id:0,name:1,uid:1})

{ "name" : "rootB", "uid" : 2 }

{ "name" : "bin", "uid" : 3 }

{ "name" : "daemon", "uid" : 4 }

{ "name" : "adm", "uid" : 5 }

{ "name" : "lp", "uid" : 6 }

{ "name" : "sync", "uid" : 7 }

**自减1--$inc**

> db.user.update({uid:{$lte:5}},{$inc:{uid:-1}},false,true)

WriteResult({ "nMatched" : 4, "nUpserted" : 0, "nModified" : 4 })

> db.user.find({uid:{$lte:7}},{\_id:0,name:1,uid:1})

{ "name" : "rootB", "uid" : 1 }

{ "name" : "bin", "uid" : 2 }

{ "name" : "daemon", "uid" : 3 }

{ "name" : "adm", "uid" : 4 }

{ "name" : "lp", "uid" : 6 }

{ "name" : "sync", "uid" : 7 }

## 数组

### 向数组中添加新元素

$push (重复添加)

$addToSet (避免重复添加)

> db.user.save({name:"bob",like:["a","b","c","d"]})

WriteResult({ "nInserted" : 1 })

> db.user.find({name:"bob"})

{ "\_id" : ObjectId("5c7de0ad2b5679323f790651"), "name" : "bob", "like" : [ "a", "b", "c", "d" ] }

### **避免重复添加$addToSet**

> db.user.update({name:"bob"},{$addToSet:{like:"e"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({name:"bob"})

{ "\_id" : ObjectId("5c7de0ad2b5679323f790651"), "name" : "bob", "like" : [ "a", "b", "c", "d", "e" ] }

### **重复添加$push**

> db.user.update({name:"bob"},{$push:{like:"e"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({name:"bob"})

{ "\_id" : ObjectId("5c7de0ad2b5679323f790651"), "name" : "bob", "like" : [ "a", "b", "c", "d", "e", "e" ] }

### 删除数组中元素

$pop #从数组头部删除一个元素,只删除一个

$pull #删除数组指定元素

$pop

> db.user.update({name:"bob"},{$pop:{like:1}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({name:"bob"})

{ "\_id" : ObjectId("5c7de0ad2b5679323f790651"), "name" : "bob", "like" : [ "a", "b", "c", "d", "e" ] }

$pop

> db.user.update({name:"bob"},{$pop:{like:-1}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({name:"bob"})

{ "\_id" : ObjectId("5c7de0ad2b5679323f790651"), "name" : "bob", "like" : [ "b", "c", "d", "e" ] }

$pull

> db.user.find({name:"bob"},{\_id:0})

{ "name" : "bob", "like" : [ "b", "c", "d", "e" ] }

> db.user.update({name:"bob"},{$pull:{like:"e"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({name:"bob"},{\_id:0})

{ "name" : "bob", "like" : [ "b", "c", "d" ] }

> db.user.update({name:"bob"},{$pull:{like:"c"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.user.find({name:"bob"},{\_id:0})

{ "name" : "bob", "like" : [ "b", "d" ] }

# 删除文档

$drop #删除集合的同时删除索引与文档,数据与集合一并删除

$remove #只删除文档时不删除索引,只删除数据