[redis基本使用 3](#_Toc1946269)

[搭建Redis服务器 3](#_Toc1946270)

[客户访问服务 5](#_Toc1946271)

[redis数据类型 10](#_Toc1946272)

[字符串操作 10](#_Toc1946273)

[Hash 表 14](#_Toc1946274)

[List 列表 15](#_Toc1946275)

[Redis中文不乱码 19](#_Toc1946276)

[配置LNMP环境+redis 19](#_Toc1946277)

[redis集群环境 21](#_Toc1946278)

[redis 服务器 ip 地址及端口规划 21](#_Toc1946279)

[创建集群 51-56 22](#_Toc1946280)

[查看集群信息 24](#_Toc1946281)

[测试集群 25](#_Toc1946282)

[redis集群工作原理 26](#_Toc1946283)

[如何选举 master 主机 26](#_Toc1946284)

[管理集群: 27](#_Toc1946285)

[向集群中添加新节点主机 28](#_Toc1946286)

[从集群中删除节点主机 30](#_Toc1946287)

[把redis服务器从集群中恢复成独立的主机 31](#_Toc1946288)

[Redis 主从复制 31](#_Toc1946289)

[主从复制工作原理 31](#_Toc1946290)

[配置从库 32](#_Toc1946291)

[一主多从 33](#_Toc1946292)

[配置主从从 33](#_Toc1946293)

[哨兵模式，主库宕机后,从库自动升级为主库 34](#_Toc1946294)

[配置带验证的主从复制 35](#_Toc1946295)

[redis持久化之 RDB 36](#_Toc1946296)

[持久化之 AOF 38](#_Toc1946297)

[memcached 41](#_Toc1946298)

[搭建memcached 41](#_Toc1946299)

[LNMP+memcached 43](#_Toc1946300)

[session 共享 44](#_Toc1946301)

[MongoDB 46](#_Toc1946302)

[搭建 MongoDB 服务器 47](#_Toc1946303)

[装包 47](#_Toc1946304)

[创建配置文件 48](#_Toc1946305)

[启动服务 48](#_Toc1946306)

[连接服务 48](#_Toc1946307)

[MongoDB 基本使用 49](#_Toc1946308)

[数据类型 51](#_Toc1946309)

[基本数据类型 字符 string/ 布尔 bool/ 空 null 51](#_Toc1946310)

[数值 / 数组 array 52](#_Toc1946311)

[代码 / 日期 / 对象 52](#_Toc1946312)

[内嵌 / 正则表达式 52](#_Toc1946313)

[数据导入导出 53](#_Toc1946314)

[数据备份恢复 55](#_Toc1946315)

[MongoDB副本集,主从，自动备份，高可用 56](#_Toc1946316)

[• Replica Sets 复制集（目前主要使用这个模式） 57](#_Toc1946317)

[验证副本集配置 62](#_Toc1946318)

[MongoDB 文档管理 63](#_Toc1946319)

数据库分为关系型数据库RDBMS和非关系型数据库NOSQL。

**关系型数据库**数据按照预先设置的组织结构,存储在物理存储介质上。 数据之间可以做关联操作。• 主流的 RDBMS 软件： Oracle， DB2， WS-sqlserver， MySQL。

**非关系型数据库NoSQL** 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的类型和结构。• 主流软件 Redis， MongoDB， Memcached， CouchDB， Neo4j， FlockDB

# redis基本使用

– Remode DIctionary Server( 远程字典服务器 )

– 使用 C 语言编写的,遵守 BSD 的开源软件

– 是一款高性能的 (Key/Values) 分布式内存数据库

– 并支持数据持久化的 NoSQL 数据库服务软件 //将内存的数据定期的写入到硬盘中，持久化.

– 中文网站 www.redis.cn

• Redis 特点:

– 支持数据持久化,可以把内存里数据保存到硬盘中

– 不仅仅支持 key/values 类型的数据,同时还支持 list hash set zset 类型

– 支持 master-salve 模式数据备份

memcache不支持持久化,只支持数字,字符

## 搭建Redis服务器

• Redis的安装包可以从官网下载。本次一redis-4.08.tar.gz为例

1、装包

[root@host51 redis-4.0.8]# rpm -q gcc gcc-c++

gcc-4.8.5-16.el7.x86\_64

gcc-c++-4.8.5-16.el7.x86\_64

[root@host51 redis-4.0.8]# make

[root@host51 redis-4.0.8]# make install

2、初始化 (执行以下脚本)

#./utls/install\_server.sh // 初始化，否则就得自己手动初始化。 在源码目录下面

[root@host51 redis-4.0.8]# ./utils/install\_server.sh

Selected config:

Port : 6379

Config file : /etc/redis/6379.conf

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

Data dir : /var/lib/redis/6379 //实现持久性化的数据库文件

Executable : /usr/local/bin/redis-server //不适用这个命令来启动服务

Cli Executable : /usr/local/bin/redis-cli //连接redis服务的命令

**进程名称**：redis-server

安装后默认开启服务

[root@host51 redis-4.0.8]# ss -antpul | grep 6379

tcp LISTEN 0 128 127.0.0.1:6379 \*:\* users:(("redis-server",pid=5364,fd=6))

默认值监听127.0.0.1，也就是只能本机访问，需要修改配置文件才能从别的机器访问

查看数据库目录列表: //实现持久性化的数据库文件

ls /var/lib/redis/6379/dump.db

**服务控制**

/etc/init.d/redis\_<portnumber> start 启动服务

/etc/init.d/redis\_<portnumber> stop 停止服务

/etc/init.d/redis\_<portnumber> status 查看状态 。如果running，则显示 redis is running(pid),如果没有运行，则括号里面没有pid

**客户访问**

redis-cli -p 端口号 -h bindip -a 密码 shutdown

查看服务信息

– ps -C redis

– netstat -utnlp | grep redis

## 客户访问服务

– redis-cli

// 连接本机 redis 服务

[root@host51 redis-4.0.8]# redis-cli

127.0.0.1:6351> ping

PONG

连接正常

常用操作指令

– Set keyname keyvalue // 存储，如果key已经存在，会覆盖原先的值

– get keyname // 获取

– Select 数据库编号 0-15 // 切换库.一共有16个数据库,默认在0,0号库没有提示,其他命令行提示符都有[num].

– Keys \* // 打印所有a开头的key

– Keys a? // 打印a开头，后面还有一个字符的key

– Exists keyname // 测试key是否存在

– ttl keyname // 查看生存时间，默认永不过期。返回值-1表示永不过期，-2表示过期

– type keyname // 查看类型 默认都是string类型.

– move keyname dbname // 移动变量

– expire keyname 10 // 设置有效时间，单位秒

– del keyname // 删除变量.

– flushall // 删除内存里所有变量.慎用.老师说不会清dump.db的内容,实际也清除了.

– save // 保存变量

– shutdown // 关闭服务

127.0.0.1:6379[1]> ttl name

(integer) -1 //永不过期.set的时候不设置,就永不过期

127.0.0.1:6379[1]> ttl name

(integer) -2 //过期了,取值返回nil

127.0.0.1:6379[1]>move keyname num //把key这个键值对移动到num库

**常用配置选项 /etc/redis/6379.conf**

– port 6379 // 端口

– bind 127.0.0.1 //IP 地址,多个ip用空格间隔

– tcp-backlog 511 //tcp 连接总数,正在连接+已经链接的个数

– timeout 0 // 连接超时时间,0是永不超时

– tcp-keepalive 300 // 长连接时间 每隔300秒检查连接的状态

– daemonize yes // 守护进程方式运行,常驻内存.不是守护进程的话,没人链接就睡觉

– databases 16 // 数据库个数

– logfile /var/log/redis\_6379.log //pid 文件

– maxclients 10000 // 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

– volatile-lru 最近最少使用 (针对设置了过期时间的 key )

– allkeys-lru 删除最少使用的 key

– volatile-random 在设置了过期的 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，默认值，生产环境需要改

• 选项默认设置

– maxmemory <bytes> // 最大内存

– maxmemory-policy noeviction // 定义使用的策略

– maxmemory-samples 5 // 选取模板数据的个数(针对 lru 和 ttl 策略).越多越精准。针对内存清除策略的模板

**设置连接密码**

• 设置密码

– grep -n requirepass /etc/redis/6379.conf

501:requirepass 123456

//修改端口配置后停止服务出错,停不掉. //etc/init.d/redis\_6379 stop

//用一下命令停服务:

redis-cli -p 6352 shutdown -a password

设置密码后登录有两个方式

1. redis-cli -p 6352 -a password
2. redis-cli -p 6352

127.0.0.1:6352>auth password

修改端口,密码,绑定的ip配置后,需要修改启动脚本,

[root@host51 ~]# vim /etc/init.d/redis\_6379

EXEC=/usr/local/bin/redis-server

CLIEXEC=/usr/local/bin/redis-cli

PIDFILE=/var/run/redis\_6379.pid

CONF="/etc/redis/6379.conf"

REDISPORT=`awk '/^port/{print $2}' /etc/redis/6379.conf`

PASS=`awk '/^requirepass/{print $2}' /etc/redis/6379.conf`

HOST=`awk '/^bind/{print $2}' /etc/redis/6379.conf`

...

case "$1" in

start)

if [ -f $PIDFILE ]

then

echo "$PIDFILE exists, process is already running or crashed"

else

echo "Starting Redis server..."

$EXEC $CONF

fi

;;

stop)

if [ ! -f $PIDFILE ]

then

echo "$PIDFILE does not exist, process is not running"

else

PID=$(cat $PIDFILE)

echo "Stopping ..."

[ -z $PASS ] && $CLIEXEC -h $HOST -p $REDISPORT shutdown \

|| $CLIEXEC -h $HOST -p $REDISPORT -a $PASS shutdown

while [ -x /proc/${PID} ]

do

echo "Waiting for Redis to shutdown ..."

sleep 1

done

echo "Redis stopped"

fi

;;

status)

if [ ! -e $PIDFILE ]

then

echo 'Redis is not running'

else

PID=$(cat $PIDFILE)

echo "Redis is running ($PID)"

fi

;;

restart)

$0 stop

$0 start

;;

\*)

echo "Please use start, stop, restart or status as first argument"

;;

esac

## redis数据类型

支持的5种类型

string

list

hash

set

zset

### 字符串操作

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

– 设置 key 及值,过期时间可以设置为秒（ex）或毫秒(px)为单位

– nx 只有 key 不存在,才对 key 进行操作

– xx 只有 key 已存在,才对 key 进行操作

• setrange key offset value

– 从偏移量开始复写 key 的特定位的值

>set first "hello world"

>setrange first 6 “Redis”

• strlen key

– 统计字串长度

>strlen first

192.168.4.51:6351> set tel 13345677654

OK

192.168.4.51:6351> strlen tel

(integer) 11

192.168.4.51:6351> setrange tel 3 \*\*\*\*

(integer) 11

192.168.4.51:6351> get tel

"133\*\*\*\*7654"

• append key value

– 字符存在则追加,不存在则创建 key 及 value

– 返回值为 key 的长度

>append myname jacob

• setbit key offset value

– 对 key 所存储字串,设置或清除特定偏移量上的位 (bit)

– Value 值可以为 1 或 0 , offset 为 0~2^32 之间 //0表示没有，1表示有

– key 不存在,则创建新 key

>setbit bit 0 1

>setbit bit 1 0

bit: 第 0 位为 1 ,第一位为 0

可以用来记录每天的登录情况 第一天登录，第二三天没登录，第四天登录

192.168.4.51:6351> setbit loginnum 1 1

(integer) 0

192.168.4.51:6351> setbit loginnum 2 0

(integer) 0

192.168.4.51:6351> setbit loginnum 3 0

(integer) 0

192.168.4.51:6351> setbit loginnum 4 1

(integer) 0

192.168.4.51:6351> bitcount loginnum

(integer) 2 //一共登录2次

• bitcount key

– 统计字串中被设置为 1 的比特位数量

>setbit bits 0 1

//0001

>setbit bits 3 1

//1001

>bitcount bits

// 结果为 2

记录网站用户上线频率,如用户 A 上线了多少天等类似的数据如用户在某天上线,则使用 setbit ,以用户名为 key ,将网站上线日为 offset ,并在该 offset 上设置 1 ,最后计算用户总上线次数时,使用 bitcount 用户名即可

这样,即使网站运行 10 年,每个用户仅占用 10\*365 比特位即 456

字节即可

>setbit peter 100 1 // 网站上线 100 天用户登录了一次

>setbit peter 105 1 // 网站上线 105 天用户登录了一次

>bitcount peter

• get key

– 返回 key 所存储的字符串值

– 如果 key 不存在则返回特殊值 nil

– 如果 key 的值不是字串,则返回错误, get 只能处理字串

• getrange key start end

– 返回字串值中的子字串,截取范围为 start 和 end

– 负数偏移量表述从末尾计数, -1 表示最后一个字符, -2表示倒数第二个字符

>set first “hello,the world”

>getrange first -5 -1 //返回world

>getrange first 0 4 //返回hello

• decr key

– 将 key 中的值减 1 , key 不存在则先初始化为 0 ,再减 1. 返回结果

>set test 10

>decr test

• decrby key decrement

– 将 key 中的值,减去 decrement

>set count 100

>decrby count 20 //返回80

• incr key

– 将 key 的值加 1 ,如果 key 不存在,则初始为 0 后再加 1，返回计算结果

– 主要应用为计数器

>set page 20

>incr page

• incrby key increment

– 将 key 的值增加 increment

• incrbyfloat key increment

– 为 key 中所储存的值加上浮点数增量 increment

>set num 16.1

>incrbyfloat num 1.1

• mget key [key...]

– 一次获取一个或多个 key 的值,空格分隔, < 具有原子性>

• mset key value [key value ...]

– 一次设置多个 key 及值,空格分隔, < 具有原子性 >

### Hash 表

Hash 表简介

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

• 一个 key 可对应多个 field ,一个 field 对应一个 value

• 将一个对象存储为 hash 类型,较于每个字段都存储成string 类型更能节省内存

Hash 表操作

• hset key field value

– 将 hash 表中 field 值设置为 value

>hset site google 'www.g.cn‘

>hset site baidu 'www.baidu.com'

• hget key filed

– 获取 hash 表中 field 的值

>hget site google

• **hmset** key field value [field value...]

– 同时给 hash 表中的多个 field 赋值

>hmset site google www.g.cn baidu www.baidu.com

• **hmget** key field [field...]

– 返回 hash 表中多个 field 的值

>hmget site google baidu

• hkeys key

– 返回 hash 表中所有 field 名称

>hmset site google www.g.cn baidu www.baidu.com

>hkeys site

• hgetall key

– 返回 hash 表中所有 field 的值（包含字段名）

一行field

一行value

• hvals key

– 返回 hash 表中所有 filed 的值

>hvals key

每行都是value

• hdel key field [field...]

– 删除 hash 表中多个 field 的值,不存在则忽略

>hdel site google baidu

### List 列表

List 列表简介

• Redis 的 list 是一个字符队列

• 先进后出

• 一个 key 可以有多个值

**List 列表操作**

• lpush key value [value...]

– 将一个或多个值 value 插入到列表 key 的表头

– Key 不存在,则创建 key

>lpush list a b c //list1 值依次为 c b a

等同于 lpush list a; lpush list b; lpush list c

• lrange key start stop

– 从开始位置读取 key 的值到 stop 结束

>lrange list 0 2 // 从 0 位开始,读到 2 位为止

>lrange list 0 -1 // 从开始读到结束为止

>lrange list 0 -2 // 从开始读到倒数第 2 位值

• lpop key

– 移除并返回列表头元素数据, key 不存在则返回 nil

>lpop list

// 删除表头元素,可以多次执行

• llen key

– 返回列表 key 的长度

• lindex key index

– 返回列表中第 index 个值

如 lindex key 0 ; lindex key 2; lindex key -2

• lset key index value

– 将 key 中 index 位置的值修改为 value

>lset list 3 test // 将 list 中第 3 个值修改为 test

• rpush key value [value...]

– 将 value 插入到 key 的末尾

>rpush list3 a b c //list3 值为 a b c

>rpush list3 d // 末尾插入 d

• rpop key

– 删除并返回 key 末尾的值

– >rpush list3 a b c //list3 值为 a b c

>rpush list3 d // 末尾插入 d

其他操作指令

• del key [key...]

– 删除一个或多个 key

• exists key

– 测试一个 key 是否存在

• expire key seconds

– 设置 key 的生存周期

• persist key

– 设置 key 永不过期

• ttl key

– 查看 key 的生存周期

• keys 匹配

– 找符合匹配条件的 key ,特殊符号用 \ 屏蔽

>keys \* // 显示所有 key

>keys h?llo // 匹配 hello,hallo,hxllo 等

>keys h\*llo // 匹配 hllo 或 heeello 等

>keys h[ae]lo // 匹配 hello 和 hallo

• flushall

– 清空所有数据

• select id

– 选择数据库, id 用数字指定,默认数据库为 0

>select 0

>select 2

• move key db\_id

– 将当前数据库的 key 移动到 db\_id 数据库中

>move key 1 // 将 key 移动到 1 数据库中

• rename key newkey

– 给 key 改名为 newkey , newkey 已存在时,则覆盖其值

• renamenx key newkey

– 仅当 newkey 不存在时,才将 key 改名为 newkey

• sort key

– 对 key 进行排序

>lpush cost 1 8 7 2 5

>sort cost

// 默认对数字排序,升序

>sort cost desc // 降序

>lpush test “about” “site” “rename”

>sort test alpha // 对字符排序

>sort cost alpha limit 0 3 // 排序后提取 0-3 位数据

>sort cost alpha limit 0 3 desc

>sort cost STORE cost2 // 对 cost 排序并保存为 cost2

• type key

– 返回 key 的数据类型

### Redis中文不乱码

如果想要看到的中文不乱码，解决方案有两种：

一、使用echo

$ echo -e `redis-cli get k1`

中国

二、redis-cli 后面加上–raw

$ redis-cli --raw

127.0.0.1:6379> get k1

中国

## 配置LNMP环境+redis

LNMP的部署见《网站服务器.doc》,在53上部署lnmp

**配置 PHP 支持 Redis**

1. 在53网站本机运行redis服务
2. 安装 php 扩展，php链接redis的模块php-redis-2.2.4.tar.gz

#yum -y install autoconf

#yum -y install automake

#rpm -ivh php-devel-5.4.16-42.el7.x86\_64.rpm ///为了获取usr/bin/phpize

# tar -zxf php-redis-2.2.4.tar.gz

# cd phpredis-2.2.4/

# /usr/bin/phpize //初始化环境,会生成/usr/bin/php-config，php的配置文件

Configuring for:

PHP Api Version: 20100412

Zend Module Api No: 20100525

Zend Extension Api No: 220100525

# ./configure --with-php-config=/usr/bin/php-config

[root@bogon phpredis-2.2.4]# make

[root@bogon phpredis-2.2.4]# make install

输出： installing shared extentions: /usr/lib64/php/modules 表示模块的安装路径。这个模块可以提供php连接redis的工具，路径后面有用

1. 修改php程序配置文件php.ini，加载redis模块

[root@host53 ~]# vim /etc/php.ini

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

extension = "redis.so"

[root@host53 ~]# systemctl restart php-fpm.service

[root@host53 modules]# php -m //查看php的所有安装支持的模块

[root@host53 modules]# php -m | grep -i redis

Redis //能够查找到表示已经启用了

1. 测试环境

[root@host53 modules]# vim /usr/local/nginx/html/linkredis.php

<?php

$redis = new redis();

$redis->connect('127.0.0.1',6379);

$redis->set('x','777');

echo $redis->get('x');

?>

[root@host53 modules]# firefox http://192.168.4.53/linkredis.php

可以看到输出777在屏幕上

登录redis数据库

[root@host53 modules]# redis-cli

127.0.0.1:6379> keys \*

1) "x"

2) "name"

127.0.0.1:6379> get x

"777"

有变量x了

# redis集群环境

## redis 服务器 ip 地址及端口规划

– redisA 192.168.4.51 6351

– redisB 192.168.4.52 6352

– redisC 192.168.4.53 6353

– redisD 192.168.4.54 6354

– redisE 192.168.4.55 6355

– redisF 192.168.4.56 6366

创建 Redis 集群

• 在 6 台主机上做如下配置,并运行 redis 服务

#yum -y install gcc gcc-c++

#tar -zxvf redis-4.0.8.tar.gz

#cd redis-4.0.8/

#make

#make install

#./utils/install\_server.sh

**• 修改配置文件**

#vim /etc/redis/redis.conf

bind IP // 只写物理接口 IP 地址

daemonize yes // 守护进程方式运行

port xxxx // 端口号不要使用默认的 6379

cluster-enabled yes // 开启集群

cluster-config-file nodes.conf // 集群的配置文件不要使用默认的名称

cluster-node-timeout 5000 // 请求超时 5 秒 和集群中的其他主机通信，超过5秒就认为断开连接

6台机器都是同样的状态

[root@host52 ~]# ss -anptul | grep redis

tcp LISTEN 0 128 192.168.4.52:6352 \*:\* users:(("redis-server",pid=1930,fd=6))

tcp LISTEN 0 128 192.168.4.52:16352 \*:\* users:(("redis-server",pid=1930,fd=8))

16352是集群之间通信的端口号

集群:使用多台服务器提供相同的服务

## 创建集群 51-56

• 在任意一台 redis 服务器上,执行创建集群的脚本都可以。先得把所有数据库都清空

– 51创建ruby 脚本运行环境

#yum -y install ruby rubygems

#rpm -ivh --nodeps ruby-devel-2.0.0.648-30.el7.x86\_64.rpm //脚本以来的程序

#gem install redis-3.2.1.gem //脚本以来的程序

#cd redis-3.2.0/src/ //在redis源码目录下有部署集群的ruby脚本，redis-trib.rb。如果有能力，可以自己写python的脚本创建集群也行。

– 创建集群

#./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 //前提：redis已经运行 。 --replicas 1 表示 自动为每一个 master 节点分配一个 slave 节点，这样会形成3对主从关系

>>> Creating cluster

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

Using 3 masters:

192.168.4.51:6351

192.168.4.52:6352

192.168.4.53:6353

Adding replica 192.168.4.55:6355 to 192.168.4.51:6351

Adding replica 192.168.4.56:6356 to 192.168.4.52:6352

Adding replica 192.168.4.54:6354 to 192.168.4.53:6353

M: 8ca9c7bd80fff351daaa0470138f9fc54776e7a9 192.168.4.51:6351

slots:0-5460 (5461 slots) master //哈希槽的范围和个数，主才分配，一共16384个槽

M: 2821d40f02e76493ce62a20b20a0da20969af7d9 192.168.4.52:6352

slots:5461-10922 (5462 slots) master

M: 53270d4e64283df9715beebb1ae501c641b37e7e 192.168.4.53:6353

slots:10923-16383 (5461 slots) master

S: bc13e827ed9233eee74b3067c24c8168cd93b9be 192.168.4.54:6354

replicates 53270d4e64283df9715beebb1ae501c641b37e7e

S: d0e4437a2e30c4c4ea6ac8df7edaaef06110be4e 192.168.4.55:6355

replicates 8ca9c7bd80fff351daaa0470138f9fc54776e7a9

S: 4bbd5498031b5e9748a316831b70192d81dd6ebe 192.168.4.56:6356

replicates 2821d40f02e76493ce62a20b20a0da20969af7d9

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

如果创建不成功,把/var/lib/redis/6351~6356/\* 删除,排除错误，重新创建

## 查看集群信息

• 任意一台主机访问本机的 redis 服务,查看即可

– redis-cli -c -h IP 地址 //-c 是连接集群,进入集群模式

> cluster nodes # 查看本机信息

> cluster info # 查看集群信息

[root@host51 src]# redis-cli -c -h 192.168.4.51 -p 6351

192.168.4.51:6351> cluster nodes

2821d40f02e76493ce62a20b20a0da20969af7d9 192.168.4.52:6352@16352 master - 0 1528426994232 2 connected 5461-10922

53270d4e64283df9715beebb1ae501c641b37e7e 192.168.4.53:6353@16353 master - 0 1528426995000 3 connected 10923-16383

bc13e827ed9233eee74b3067c24c8168cd93b9be 192.168.4.54:6354@16354 slave 53270d4e64283df9715beebb1ae501c641b37e7e 0 1528426995000 4 connected

4bbd5498031b5e9748a316831b70192d81dd6ebe 192.168.4.56:6356@16356 slave 2821d40f02e76493ce62a20b20a0da20969af7d9 0 1528426995237 6 connected

8ca9c7bd80fff351daaa0470138f9fc54776e7a9 192.168.4.51:6351@16351 myself,master - 0 1528426994000 1 connected 0-5460

d0e4437a2e30c4c4ea6ac8df7edaaef06110be4e 192.168.4.55:6355@16355 slave 8ca9c7bd80fff351daaa0470138f9fc54776e7a9 0 1528426994000 5 connected

192.168.4.51:6351> cluster info

cluster\_state: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:2589

cluster\_stats\_messages\_pong\_sent:2731

cluster\_stats\_messages\_sent:5320

cluster\_stats\_messages\_ping\_received:2726

cluster\_stats\_messages\_pong\_received:2589

cluster\_stats\_messages\_meet\_received:5

cluster\_stats\_messages\_received:5320

192.168.4.51:6351>

192.168.4.51:6351> exit

## 测试集群

• 在客户端连接redis服务器（主库）

在其中任意一台上存储数据 ,在其他任意一台上都可以获取数据。

存数据 set name jerry

取数据 get name

[root@host51 src]# redis-cli -c -h 192.168.4.51 -p 6351

192.168.4.51:6351> ping

PONG

192.168.4.51:6351> set name natasha

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

OK

192.168.4.52:6352> get name

"natasha"

在其他主机上查看

## redis集群工作原理

redis-cli -h 192.168.4.51 –p 6351 //访问任意主库

把key使用crc16 算法做计算，然后把计算结果和16384做求模运算，看结果在哪个哈希槽内，就存储到哪个主库上去。从库自动复制主库数据

%16384 比如497

0-5460

5461-10922

10923-16383

根据取余结果,匹配对应的槽位数

分配到对应的master里面,对应的从库上也会有,

其他的2组主从里面没有这个数据,查看从库的数据时,在从库登录要用-c 登录

## 如何选举 master 主机

• 把是 master 角色主机上的 Redis 服务 停止

#redis-cli -h ip -p 端口 shutdown

• 对应是 slave 角色主机，能自动升级为 master

注意: 原先是 master 服务启动后身份是当前 master 的 slave。

如果同时有两个主挂了，集群就挂了。

把52的redis服务关掉,56自动变为主库

[root@host56 ~]# redis-cli -c -h 192.168.4.56 -p 6356

192.168.4.56:6356> cluster nodes

bc13e827ed9233eee74b3067c24c8168cd93b9be 192.168.4.54:6354@16354 slave 53270d4e64283df9715beebb1ae501c641b37e7e 0 1528440604741 4 connected

2821d40f02e76493ce62a20b20a0da20969af7d9 192.168.4.52:6352@16352 master,fail - 1528439971770 1528439971570 2 disconnected

d0e4437a2e30c4c4ea6ac8df7edaaef06110be4e 192.168.4.55:6355@16355 slave 8ca9c7bd80fff351daaa0470138f9fc54776e7a9 0 1528440605244 5 connected

8ca9c7bd80fff351daaa0470138f9fc54776e7a9 192.168.4.51:6351@16351 master - 0 1528440603235 1 connected 0-5460

4bbd5498031b5e9748a316831b70192d81dd6ebe 192.168.4.56:6356@16356 myself,master - 0 1528440604000 7 connected 5461-10922

53270d4e64283df9715beebb1ae501c641b37e7e 192.168.4.53:6353@16353 master - 0 1528440605000 3 connected 10923-16383

把52的服务回复正常,52自动变为56的从

192.168.4.56:6356> cluster nodes

bc13e827ed9233eee74b3067c24c8168cd93b9be 192.168.4.54:6354@16354 slave 53270d4e64283df9715beebb1ae501c641b37e7e 0 1528440688000 4 connected

2821d40f02e76493ce62a20b20a0da20969af7d9 192.168.4.52:6352@16352 slave 4bbd5498031b5e9748a316831b70192d81dd6ebe 0 1528440688587 7 connected

d0e4437a2e30c4c4ea6ac8df7edaaef06110be4e 192.168.4.55:6355@16355 slave 8ca9c7bd80fff351daaa0470138f9fc54776e7a9 0 1528440688000 5 connected

8ca9c7bd80fff351daaa0470138f9fc54776e7a9 192.168.4.51:6351@16351 master - 0 1528440688687 1 connected 0-5460

4bbd5498031b5e9748a316831b70192d81dd6ebe 192.168.4.56:6356@16356 myself,master - 0 1528440687000 7 connected 5461-10922

53270d4e64283df9715beebb1ae501c641b37e7e 192.168.4.53:6353@16353 master - 0 1528440687000 3 connected 10923-16383

## 管理集群:

redis-trib.rb 脚本

• 语法格式

– Redis-trib.rb

选项 参数

• 选项

– add-node 添加新节点 //默认是主

– check 对节点主机做检查

– reshard 对节点主机重新分片

– add-node --slave 添加从节点主机 //默认添加到从最少的主，也可以指定

– del-node 删除节点主机

### 向集群中添加新节点主机

默认添加进来的角色是master,但是没有hash slot范围和个数

**添加新节点**

[root@host51 src]# ./redis-trib.rb add-node 192.168.4.57:6357 192.168.4.51:6351

[root@host51 src]# ./redis-trib.rb check 192.168.4.51:6351 从51节点开始查看

默认新主机被选为主

• 手动对集群进行重新分片

[root@host51 src]# ./redis-trib.rb reshard 192.168.4.51:6351

后面可以写51-56的ip进行重新分片

[OK] All 16384 slots covered.

How many slots do you want to move (from 1 to 16384)? 4096(因为有4个主,平均分配16384/4=4096)

What is the receiving node ID? e5f366981aad8d116956ee66a3f938413182fd60//此处输入新主机的id

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 //这里填写all是从所有主机节点获取,也可以从某个主节点主机获取

• 查看分配的槽位

[root@host51 src]# ./redis-trib.rb check 192.168.4.51:6351

添加从节点

• 对添加为从节点的主机做如下配置

– 装包 修改配置文件 启动 redis 服务

• 添加从节点

# ./redis-trib.rb add-node --slave --master-id id 值

从节点 ip: 端口 192.168.4.51:6351

如果不指定主节点的 id 的话,会把新节点 随机添加为 从节点 最少的主的从

[root@host51 src]# ./redis-trib.rb add-node --slave --master-id e5f366981aad8d116956ee66a3f938413182fd60 192.168.4.58:6358 192.168.4.51:6351

[root@host51 src]# ./redis-trib.rb check 192.168.4.51:6351

检查一下 192.168.4.58的主节点是不是57

### 从集群中删除节点主机

**移除从节点**

• 从节点主机没有槽位范围,直接执行移除命令即可

– redis-trib.rb del-node 192.168.4.51:6351 被移除主机的 ID

移除后自动关闭服务

[root@host51 src]# ./redis-trib.rb del-node 192.168.4.51:6351 e18b3551d69a36ceca5b595254be70d3d0f5c293

>>> Removing node e18b3551d69a36ceca5b595254be70d3d0f5c293 from cluster 192.168.4.51:6351

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

>>> SHUTDOWN the node.

[root@host51 src]# ./redis-trib.rb check 192.168.4.51:6351

如果把删除的主机添加回来,需要把要添加的主机库里的cluster信息清空, cluster reset

**删除主节点**

先得释放哈希槽，再删除主库

Redis-trib.rb reshard 192.168.4.51:6351 //这是集群信息，然后根据提示输入信息

多少个

给哪个主

从哪个主机给

Done

Yes

./redis-trib.rb check 192.168.4.51:6351

redis-trib.rb del-node 192.168.4.51:6351 被移除主机的 ID //集群信息可以填写集群里面的任意一个节点ip

### 把redis服务器从集群中恢复成独立的主机

#Cluster-enabled yes

Rm –rf /var/lib/redis/6379/\*

重启服务，检查端口没有163XX的端口即可

# Redis 主从复制

主从复制结构模式

• 结构模式

– 一主一从

– 一主多从

– 主从从

## 主从复制工作原理

• 工作原理

– Slave 向 maste 发送 sync 命令

– Master 启动后台存盘进程,同时收集所有修改数据命令

– Master 执行完后台存盘进程后,传送整个数据文件到slave

– Slave 接收数据文件后,将其存盘并加载到内存中完成首次完全同步

– 后续有新数据产生时, master 继续将新的所有收集到的修改命令依次传给 slave ,完成同步。

主从复制缺点

• 缺点

– 网络繁忙,会产生数据同步延时问题

– 系统繁忙,会产生数据同步延时问题

配置主从复制(启动后自己默认都主库，只需要配置从库就可以)

redis 51 redis 52 redis 53

## 配置从库

• 配置从库 192.168.4.52/24

– redis 服务运行后,默认都是 master 服务器

– 修改服务使用的 IP 地址 bind 192.168.4.X

[root@redis52 ~]# redis-cli -h 192.168.4.52 -p 6352

192.168.4.52:6352> info replication // 查看主从配置信息

# Replication

role:master

connected\_slaves:0

......

192.168.4.52:6352> FLUSHALL

192.168.4.52:6352> save

192.168.4.52:6352> SLAVEOF 192.168.4.51 6351 //命令行指定主库（马上生效）

OK //SLAVEOF 主库IP 地址 端口号

192.168.4.52:6352> info replication

# Replication

role:slave

master\_host:192.168.4.51

master\_port:6351

## 一主多从

将53也配置成51的从

192.168.4.53:6353> SLAVEOF 192.168.4.51 6351

查看51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=335,lag=1

slave1:ip=192.168.4.53,port=6353,state=online,offset=335,lag=0

## 配置主从从

命令行配置，只需要把redis服务重启，就取消从库的角色

这时把53设置成52的从就可以

[root@host53 ~]# /etc/init.d/redis\_6353 stop

[root@host53 ~]# /etc/init.d/redis\_6353 start

192.168.4.53:6353> SLAVEOF 192.168.4.52 6352

查看52的主从信息

192.168.4.52:6352> info replication

# Replication

role:slave

master\_host:192.168.4.51

master\_port:6351

...

connected\_slaves:1

slave0:ip=192.168.4.53,port=6353,state=online,offset=881,lag=1

...

反客为主

• 反客为主

– 主库宕机后,手动将从库设置为主库

[root@redis52 ~]# redis-cli -h 192.168.4.52

192.168.4.52:6379> SLAVEOF no one // 设置为主库

192.168.4.52:6379> info replication

# Replication

role:master

## 哨兵模式，主库宕机后,从库自动升级为主库

可以用keepalive等第三方软件让客户端自动访问主库

51，52,53 主从从模式

– 在 slave 主机编辑 sentinel.conf 文件

– 在 slave 主机运行哨兵程序

[root@redis52 ~]# vim /etc/sentinel.conf //文件名必须是这个

sentinel monitor redis51 192.168.4.51 6379 1

[root@redis52 ~]# redis-sentinel /etc/sentinel.conf

sentinel monitor 主机名 ip 地址 端口 票数

主机名:自定义

IP 地址: master 主机的 IP 地址

端 口: master 主机 redis 服务使用的端口

票 数:主库宕机后, 票数大于 1 的主机被升级为主库

将51down掉，52的角色会变成master

将51恢复，51会自动变为52的从

//哨兵模式会自动修改配置文件，把51变成52的从，具体见下面一节

192.168.4.52:6352> info replication

# Replication

role:master

connected\_slaves:2

slave0:ip=192.168.4.53,port=6353,state=online,offset=5425,lag=1

slave1:ip=192.168.4.51,port=6351,state=online,offset=5425,lag=1

...

## 配置带验证的主从复制

• 配置 master 主机

– 设置连接密码 ,启动服务,连接服务

[root@redis52 ~]# sed -n '70p;501p' /etc/redis/6379.conf

bind 192.168.4.51

requirepass 123456 // 密码

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

Starting Redis server...

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

192.168.4.52:6352>

• 配置 slave 主机

– 指定主库 IP ,设置连接密码,启动服务

[root@redis53 ~]# sed -n '70p;282p;289p' /etc/redis/6379.conf

bind 192.168.4.53

slaveof 192.168.4.52 6352 // 主库 IP 与端口

masterauth 123456 // 主库密码

[root@redis53 ~]#

[root@redis53 ~]# /etc/init.d/redis\_6379 start

Starting Redis server...

[root@redis53 ~]# redis-cli -h 192.168.4.53 -p 6353

192.168.4.53:6353> INFO replication

# Replication

role:slave

master\_host:192.168.4.52

master\_port:6352

## redis持久化之 RDB

RDB 介绍

• 全称 Reids DataBase

– 数据持久化方式之一

– 在指定时间间隔内,将内存中的数据集快照写入硬盘

– 术语叫 Snapshot 快照。

– 恢复时,将快照文件直接读到内存里。

相关配置参数

• 文件名

– dbfilename “dump.rdb” // 文件名

– save “” // 禁用 RDB

• 数据从内存保存到硬盘的频率

– save 900 1 // 900 秒内且有 1 次修改存盘

– save 300 10 //300 秒内且有 10 次修改存盘

– save 60 10000 //60 秒内且有 10000 修改存盘

• 手动立刻存盘

– > save // 阻塞写存盘

– > bgsave // 不阻塞写存盘

使用备份的文件

/var/lib/redis/6379/dump.rdb可以对文件进行恢复

• 压缩

– rdbcompression yes | no

• 在存储快照后,使用 crc16 算法做数据校验

– rdbchecksum yes|no

压缩会消耗cpu，校验也会消耗cpu。工作中要保证数据一致性，才开

• bgsave 出错停止写操作 , 对数据一致性要求不高设置为 no

– stop-writes-on-bgsave-error yes|no

使用 RDB 文件恢复数据

• 备份数据

– 备份 dump.rdb 文件到其他位置

– ~]# cp 数据库目录 /dump.rdb

备份目录

• 恢复数据

– 把备份的 dump.rdb 文件拷贝回数据库目录 , 重启 redis 服务

– cp 备份目录 /dump.rdb 数据库目录 /

– /etc/redid/redis\_ 端口 start

RDB 优点 / 缺点

• RDB 优点

– 持久化时, Redis 服务会创建一个子进程来进行持久化,会先将数据写入到一个临时文件中,待持久化过程都结束了,再用这个临时文件替换上次持久化好的文件;整个过程中主进程不做任何 IO 操作,这就确保了极高的性能。

– 如果要进行大规模数据恢复,且对数据完整行要求不是非常高,使用 RDB 比 AOF 更高效。

• RDB 的缺点

Redis忙的时候，会影响redis的处理速度

– 意外宕机,最后一次持久化的数据会丢失。

## 持久化之 AOF

（在RDB和AOF同时存在时，优先按照AOF的文件启动）

默认AOF不开启

• 只追加操作的文件

– Append Only File

– 记录 redis 服务所有写操作。

– 不断的将新的写操作,追加到文件的末尾。

– 使用 cat 命令可以查看文件内容

相关配置参数

• 文件名

– appendfilename "appendonly.aof" // 文件名

– appendonly yes // 启用 aof ,默认 no

• AOF 文件记录,写操作的三种方式

– appendfsync always // 有新的写操作立即记录,

性能差,完整性好。

– appendfsync everysec // 每秒记录一次,宕机时会

丢失 1 秒的数据

– appendfsync no // 从不记录

• 日志重写 ( 日志文件会不断增大 ) ,何时会触发日志重写?

– redis 会记录上次重写时 AOF 文件的大小,默认配置

是当 aof 文件是上次 rewrite 后大小的 1 倍且文件大于64M 时触发。

– auto-aof-rewrite-percentage 100

– auto-aof-rewrite-min-size 64mb

• 修复 AOF 文件,

– 把文件恢复到最后一次的正确操作

[root@redis53 6379]# redis-check-aof --fix appendonly.aof

0x

83: Expected \r\n, got: 6166

AOF analyzed: size=160, ok\_up\_to=123, diff=37

This will shrink the AOF from 160 bytes, with 37 bytes, to 123

bytes

Continue? [y/N]: y

Successfully truncated AOF

把服务还没启动的pid删除

[root@host51 6351]# rm -rf /var/run/redis\_6351.pid

AOF进行备份

[root@host51 6351]# cp appendonly.aof /root/appendonly.aof.bak

使用 AOF 文件恢复数据(恢复前可以可以先修复AOF文件)

• 备份数据

– 备份 aof 文件到其他位置

– ~]# cp 数据库目录/appendonly.aof 备份目录

• 恢复数据

– 把备份的 aof 文件拷贝回数据库目录 , 重启 redis 服务

– Cp 备份目录 /appendonly.aof 数据库目录/ //在恢复前可以把aof文件里面的flushall命令删除

– /etc/redid/redis\_ 端口 start

AOF 优点 / 缺点

• AOF优点

– 可以灵活的设置同步持久化 appendfsync alwayls 或

异步持久化 appendfsync verysec

– 宕机时,仅可能丢失 1 秒的数据

• AOF缺点

– AOF 文件的体积通常会大于 RDB 文件的体积。执行 fsync 策略时的速度可能会比 RDB 慢。

# memcached

## 搭建memcached

1**）使用yum安装软件包memcached**

yum -y install memcached

rpm -qa memcached

memcached-1.4.15-10.el7\_3.1.x86\_64

**2) memcached配置文件 不需要修改**

vim /usr/lib/systemd/system/memcached.service

ExecStart=/usr/bin/memcached -u $USER -p $PORT -m $CACHESIZE -c $MAXCONN $OPTIONS

[root@proxy ~]# vim /etc/sysconfig/memcached

PORT="11211"

USER="memcached"

MAXCONN="1024"

CACHESIZE="64"

OPTIONS=""

**3）启动服务并查看网络连接状态验证是否开启成功：**

[root@proxy ~]# systemctl start memcached

[root@proxy ~]# systemctl status memcached

[root@proxy ~]# netstat -anptu | grep memcached

tcp    0    0 0.0.0.0:11211        0.0.0.0:\*        LISTEN        2839/memcached

tcp    0    0 :::11211            :::\*                LISTEN        2839/memcached

udp    0    0 0.0.0.0:11211        0.0.0.0:\*                    2839/memcached

udp    0    0 :::11211            :::\*                            2839/memcached

[root@proxy ~]# setenforce 0

[root@proxy ~]# firewall-cmd --set-default-zone=trusted

**客户端使用telnet访问memcached服务器**

1）使用yum安装telnet

yum -y install telnet

2)使用telnet连接服务器测试memcached服务器功能，包括增、删、改、查等操作。

[root@proxy ~]# telnet 192.168.4.5 11211

Trying 192.168.4.5...

……

set name 0 180 3          //定义变量，变量名称为name (不压缩，存储几秒，几个字节)

plj                            //输入变量的值，值为plj

STORED

get name                        //获取变量的值

VALUE name 0 3             //输出结果

plj

END

add myname 0 180 10            //新建，myname不存在则添加，存在则报错

set myname 0 180 10            //添加或替换变量

replace myname 0 180 10        //替换，如果myname不存在则报错

get myname                    //读取变量

append myname 0 180 10        //向变量中追加数据

delete myname                    //删除变量

stats                        //查看状态

flush\_all                        //清空所有

quit                            //退出登录

## LNMP+memcached

1. 搭建LNMP
2. 为PHP添加memcache扩展

yum -y install php-pecl-memcache

1. 启动Nginx服务
2. 启动MySQL服务
3. 启动PHP-FPM服务
4. 关闭SELinux、防火墙

**创建PHP页面，使用PHP语言测试memcached服**务

vim /usr/local/nginx/html/test.php

<?php

$memcache=new Memcache;                //创建memcache对象

$memcache->connect('localhost',11211) or die ('could not connect!!');

$memcache->set('key','test');          //定义变量

$get\_values=$memcache->get('key');     //获取变量值

echo $get\_values;

?>

<?php

$memcache=new Memcache;

$memcache->connect('localhost',11211) or die ('could not connect!!!');

$memcache->set('key','test');

$get\_values=$memcache->get('key');

echo $get\_values;

?>

步骤四：客户端测试

Yum -y install php-pecl-memcache

Systemctl restart php-fpm

客户端使用浏览器访问服务器PHP首页文档，检验对memcached的操作是否成功：

### session 共享

1. 部署后端LNMP服务,启动nginx,mysqld ,php-fpm

nginx需要支持动静分离

1. 部署前端Nginx调度服务器

Nginx配置文件中，通过upstream定义后端服务器地址池，默认调度策略为轮询，使用proxy\_pass调用upstream定义的服务器地址池：

vim /usr/local/nginx/conf/nginx.conf

upstream webs {

server 192.168.2.100:80;

server 192.168.2.200:80;

}

server {

listen 80;

server\_name localhost;

location / {

             proxy\_pass http://webs;

root html;

index index.html index.htm;

}

}

[root@web1 ~]# netstat -nutlp |grep java        //查看java监听的端口

tcp 0 0 :::8080 :::\* LISTEN 2778/java

tcp 0 0 ::ffff:127.0.0.1:8005 :::\* LISTEN 2778/java

1. 部署测试页面,测试. 此时完成nginx轮询调度\
2. 后端部署有登录功能的页面,测试session.

测试页面可以参考lnmp\_soft/php\_scripts/php-memcached-demo.tar.gz。

cd lnmp\_soft/php\_scripts/

tar -xf php-memcached-demo.tar.gz

cd php-memcached-demo

cp -a \* /usr/local/nginx/html/

5. 调度器安装Memcached服务,并启动

yum -y install memcached

yum -y install php-pecl-memcache

1. 后端lnmp修改session存储位置. 修改PHP-FPM配置文件，并重启服务

vim /etc/php-fpm.d/www.conf

修改前效果如下:

php\_value[session.save\_handler] = files

php\_value[session.save\_path] = /var/lib/php/session

//原始文件，默认定义Sessoin会话信息本地计算机（默认在/var/lib/php/session）

修改后效果如下:

php\_value[session.save\_handler] = memcache

php\_value[session.save\_path] = "tcp://192.168.2.5:11211"

//定义Session信息存储在公共的memcached服务器上，主机参数中为memcache（没有d）

//通过path参数定义公共的memcached服务器在哪（服务器的IP和端口）

systemctl restart php-fpm

快速恢复到初始状态

pssh -i -h host.txt /etc/init.d/redis\* stop

pssh -i -h host.txt rm -rf /var/lib/redis/\*/\*

pssh -i -h host.txt sed -i '/^cluster/s/^/#/' /etc/redis/63\*

pssh -i -h host.txt /etc/init.d/redis\* start

pssh -i -h host.txt /etc/init.d/redis\* status

pssh -i -h host.txt ss -anptul | grep redis

# MongoDB

软件介绍

• 介于关系数据库和非关系数据库之间的产品

– 一个基于分布式文件存储的数据库。

– 由 C++ 语言编写。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

– MongoDB 将数据存储为一个文档,数据结构由键值(key=>value) 对组成。

– MongoDB 文档类似于 JSON 对象。字段值可以包含其他文档,数组及文档数组。

软件特点

– 安装简单

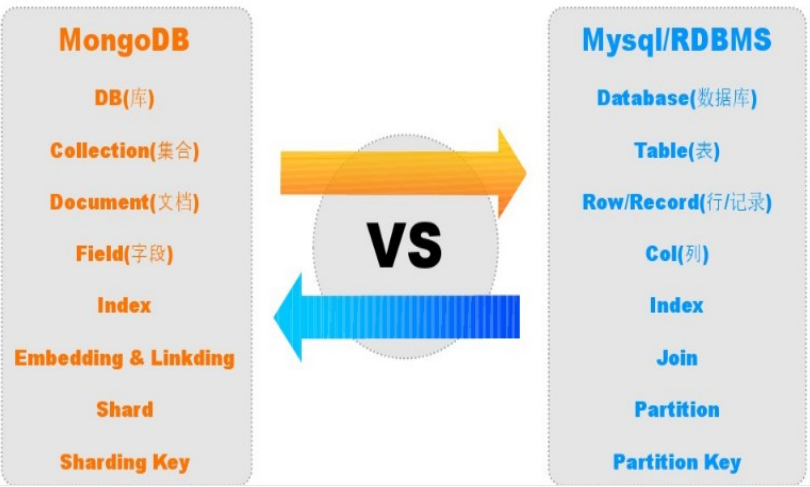
– 面向文档存储,操作比较简单容易

– 支持丰富的查询表达

– 可以设置任何属性的索引

– 支持主流编程语言 RUBY|PYTHON|JAVA|PHP|C++

– 支持副本集,分片



MongoDB相关概念

## 搭建 MongoDB 服务器

### 装包

• 免安装,解压后即可使用

[root@bogon ~]# mkdir /usr/local/mongodb

[root@bogon ~]# tar -zxf mongodb-linux-x86\_64-rhel70-3.6.3.tgz

[root@bogon ~]# cp -r

mongodb-linux-x86\_64-rhel70-3.6.3/bin /usr/local/mongodb/

[root@bogon ~]# cd /usr/local/mongodb/

[root@bogon mongodb]# mkdir etc

[root@bogon mongodb]# mkdir log

[root@bogon mongodb]# mkdir -p data/db //这个目录是固定的名字

### 创建配置文件

可以使用 ./bin/mongod --help 查看需要哪些字段，ip地址，端口等没有指定，ip地址就可以使用很多个

• 手动创建服务主配置文件

[root@bogon mongodb]# vim etc/mongodb.conf

logpath=/usr/local/mongodb/log/mongodb.log

logappend=true # 追加的方式记录日志信息

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

fork=true # 守护进程方式运行

### 启动服务

./bin/mongod -f ./etc/mongodb.conf

进程名：mongod

默认端口：27017

日志文件： mongodb.log，拍错的重要依据

### 连接服务

• 本地连接,默认没有密码

[root@bogon ~]# /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

......

> show dbs // 显示已有的库，有默认的三个库

admin 0.000GB

config 0.000GB

local 0.000GB

> exit # 断开连接

bye

[root@bogon ~]#

use bbsdb 如果没有则创建该库，并进入

## MongoDB 基本使用

数据库管理

• 库管理命令: 查看 创建 切换 删除，命令严格区分大小写

– show dbs 看已有的库

– db 显示当前所在的库，默认在test库

– use 库名 切换库,若库不存在的话， 自动延时创建库

– show collections 或 show tables 查看库下已有的集合

– db.dropDatabase() 删除当前所在的库，

• 数据库名可以是满足以下条件的任意 UTF-8 字符串。

– 不能是空字符串( "") 。

– 不得含有 ' ' (空格 ) 、 . 、 $ 、 / 、 \ 和 \0 ( 空字符 )。

– 应全部小写。

– 最多 64 字节。

集合管理

• 集合管理命令:查看 创建 删除

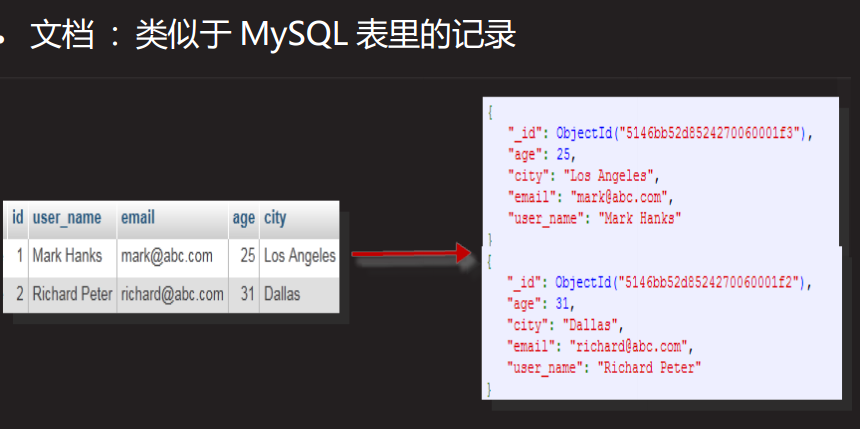
– show collections 或 show tables # 查看集合

– db. 集合名 .drop() # 删除集合

– db. 集合名 .save({'',''}) # 创建集合，集合不存在时,创建并添加文档

> db.user.save({'name':'bob','age':'21'}) //创建了一个集合user，里面有一个文档，文档各个字段之间用逗号隔开。大括号里面是一个文档的内容，可以有多个大括，各个文档之间的字段可以不用

WriteResult({ "nInserted" : 1 }) //这是返回信息



• 合法的集合名

– 集合名不能是空字符串 "" 。

– 集合名不能含有 \0 字符(空字符 ) ,这个字符表示集合名的结尾。

– 集合名不能以 "system." 开头,这是为系统集合保留的前缀。

– 用户创建的集合名字不能含有保留字符。

文档基本管理

• 文档 : 类似于 MySQL 表里的记录

• 文档管理命令: 查看 统计 添加 删除

– db. 集合名 .find()

– db. 集合名 .count()

– db. 集合名 .insert({“name”:”jim”}) //save也可以，两个有区别

– db. 集合名 .find({ 条件 }) //db.user.find({name:”yaya”})查找name是yaya的文档，find()没有条件就返回所有文档,这里可以不用加{}。

– db. 集合名 .findOne() # 返回一条文档

– db. 集合名 .remove({}) # 删除所有文档，必须有{}

– db. 集合名 .remove({ 条件 }) # 删除与条件匹配的所有文档

• 插入记录

>db.col.insert(

{ title: 'MongoDB 教程 ',

description: 'MongoDB 是一个 Nosql 数据库 ',

by: 'MongoDB 中文网 ',

url: 'http://www.mongodb.org.cn',

tags: ['mongodb', 'database', 'NoSQL'],

likes: 100

}

)

>db.col.remove({‘ttle’:‘MongoDB 教程’ }) 删除记录

## 数据类型

### 基本数据类型 字符 string/ 布尔 bool/ 空 null

• 字符串 string

– UTF-8 字符串都可以表示为字符串类型的数据，用引号引上

– {name:” 张三” } 或 { school:“tarena”}

• 布尔 bool

– 布尔类型有两个值 true 和 false , {x:true}

• 空 null

– 用于表示空值或者不存在的字段, {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 {x : 3} 。

– NumberInt ( 4 字节整数) {x:NumberInt(3)} //如果数字是小数，则保留整数

– NumberLong ( 8 字节整数) {x:NumberLong(3)} //如果数字是小数，则报错

• 数组 array

– 数据列表或数据集可以表示为数组

– {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

– db.t2.save{language:’php’, x: function( ){/\* 代码 \*/}}

• 日期

– 日期被存储为自新纪元依赖经过的毫秒数,不存储时区,显示的时候还是年月日等

– {x:new Date( )}

• 对象

– 对象 id 是一个 12 字节的字符串,是文档的唯一标识

– {x: ObjectId() }

### 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

– {tarena: {address:“Beijing”,tel:“888888”,person:”hanshaoyun”}}

• 正则表达式

– 查询时,使用正则表达式作为限定条件

– {x:/ 正则表达式 /}

db.t2.sve({name:’bob’,match:/^a/})

db.t1.save({name:"jerry",code:function(){/\*<?php echo "haha" ?>\*/})

## 数据导入导出

数据导出

• 语法格式 1

– #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

• 语法格式 2

– #mongoexport --host IP 地址 --port 端口

- 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

注意:导出为 csv 格式必须使用 -f 指定字段名列表 !!!

[root@host51 mongodb]# cat /usr/local/mongodb/etc/mongodb.conf

logpath=/usr/local/mongodb/log/mongodb.log

logappend=true

dbpath=/usr/local/mongodb/data/db

fork=true

bind\_ip=192.168.4.51

port=27017

更改数据库的登录ip和端口

• 语法格式 3

#mongoexport [ --host IP 地址 --port 端口 ]

-d 库名 -c 集合名 [ -q ‘{ 条件 }’ –f 字段列表]

--type=json > 目录名 / 文件名 .json

[root@host51 mongodb]# mongoexport --host 192.168.4.51 --port 27017 -d bbsdb -c t1 > /mdir/a1.csv

//可以不写类型，这样把表格内容全都导出来

[root@host51 mongodb]# mongoexport --host 192.168.4.51 --port 27017 \

-d userdb -c user -f name,uid,gid --type=csv > /mdir/user.csv

//csv格式必须要选择字段名，-f

[root@host51 mongodb]# mongoexport --host 192.168.4.51 --port 27017 -d userdb -c user --type=json > /mdir/user.json

[root@host51 mongodb]# cat /mdir/user.json

{"\_id":{"$oid":"5b1f444571fdc25c689a160c"},"name":"root","passwd":"x","uid":0.0,"gid":0.0,"comment":"root","homedir":"/root","shell":"/bin/bash"}

数据导入

• 语法格式 1

– #mongoimport --host IP 地址 --port 端口

-d 库名 – c 集合名 --type=json 目录名/文件名.json

• 语法格式 2

– #mongoimport --host IP 地址 --port 端口

-d 库名 – c 集合名 --type=csv --headerline [--drop] 目录名/文件名 .csv

注意:导入数据时库和集合不存在时,会创建库和集合后导入数据

反之以追加的方式导入数据到集合里,使用— drop 选项可以删除原有数据后

导入新数据 --headerline 忽略标题，把csv文件里面的标题去掉 --headerline和 -f 字段1,字段2，不能同时使用

--type一定要输入，不然会报错

如果json导入的文件和库里的集合里有相同的\_id值，则不能导入，可以编辑json文件，把id值删除，然后导入

[root@host51 mongodb]# sed -i 's/:/,/g' /mdir/passwd

[root@host51 mongodb]# mongoimport --host 192.168.4.51 --port 27017 -d webdb -c userinfo --type=csv -f name,passwd,gid,uid,comment,homedir,shell --drop /mdir/passwd

## 数据备份恢复

数据备份

• 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 ]

• 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 目录

目录无需事先创建 备份时指定即可!!!

• 查看 bson 文件内容

#bsondump ./dump/bbs/t1.bson

数据恢复

• 语法格式

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

[root@host51 bbsdb]# mongorestore --host 192.168.4.51 --port 27017 -d gamedb -c passwd /root/monbak/dump/webdb/userinfo.bson

[root@host51 bbsdb]# mongorestore --host 192.168.4.51 --port 27017 -d gamedb /root/monbak/dump/webdb/

把整个库进行恢复

## MongoDB副本集,主从，自动备份，高可用

• 副本集是什么

– MongoDB 复制是将数据同步在多个服务器的过程。

– 复制提供了数据的冗余备份,并在多个服务器上存储数据副本,提高了数据的可用性, 并可以保证数据的安全性。

– 复制还允许您从硬件故障和服务中断中恢复数据

复制原理

• 副本集工作过程

– mongodb 的复制至少需要两个节点。其中一个是主节点,负责处理客户端请求,其余的都是从节点,负责复制主节点上的数据。

– mongodb 各个节点常见的搭配方式为:一主一从、一主多从。

– 主节点记录在其上的所有操作 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。

副本集实现方式

• Master-Slave 主从复制

– 实现数据同步只需要在某一台服务器启动时加上 "-master" 参数,以指明此服务器的角色是 primary ;另一台服务器加上 "-slave" 和 "-source" 参数,以指明此服务器的角色是 slave 。

• 主从复制的优点如下:

– 从服务器可以执行查询工作,降低主服务器访问压力

– 在从服务器执行备份,避免备份期间锁定主服务器的数据。

– 当主服务器出现故障时,可以快速切换到从服务器,减少当机时间。

### • Replica Sets 复制集（目前主要使用这个模式）

– MongoDB 在 1.6 版本对开发了新功能 replica set,这比之前的 replication 功能要强大一 些,增加了故障自动切换和自动修复成员节点,各个 DB 之间数据完全一致,大大降低了维护成本。使用replica set故障切换完全自动。

– Replica Sets 的结构类似一个集群,完全可以把它当成一个集群,因为它确实与集群实现的作用是一样的:如果其中一个节点出现故障,其他节点马上会将业务接管过来而无须停机操作

配置 Replica Sets 副本集

1. 运行服务，或者修稿配置文件

• 启动服务时,指定主机所在副本集名称

– 副本集成员间使用相同的副本集名称

– --replSet rs1 // 指定副本集名称

[root@server0 ~]#mkdir /data/db

[root@server0 ~]#./mongod --bind\_ip 192.168.4.61 \

--logpath=/var/log/mongod.log --replSet rs1 &

[root@server0 ~]# jobs

Running .........

或者直接在配置文件中添加

replSet=rs1

重新启动服务

1. 配置节点信息

• 在任意一台主机连接 mongod 服务,执行如下操作

[root@server0 ~]# ./mongo --host 192.168.4.51

config = {

\_id:"rs1",

members:[

{\_id:0,host:“IP 地址 : 端口 "},

{\_id:1,host:“IP 地址 : 端口 "},

{\_id:2,host:“IP 地址 : 端口 "}

]

};

[root@host51 mongodb]# mongo --host 192.168.4.51 --port 27051

config = {\_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"}

]

}

输出正确结果

{

"\_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"

}

]

}

1. **初始化 Replica Sets 环境**

> rs.initiate(config)

{

"ok" : 1,

"operationTime" : Timestamp(1528854249, 1),

"$clusterTime" : {

"clusterTime" : Timestamp(1528854249, 1),

"signature" : {

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

"keyId" : NumberLong(0)

}

}

}

rs1:SECONDARY>

1. 查看副本集信息

• 查看状态信息

– > rs.status( )

• 查看是否是master库

– > rs .isMaster( )

> rs.status()

"members" : [

{

"\_id" : 0,

"name" : "192.168.4.51:27051",

"health" : 1,

**"state" : 1, //1 表示是主库**

"stateStr" : "PRIMARY", //primary标示

"uptime" : 963,

"optime" : {

"ts" : Timestamp(1528854372, 1),

"t" : NumberLong(1)

},

"optimeDate" : ISODate("2018-06-13T01:46:12Z"),

"infoMessage" : "could not find member to sync from",

"electionTime" : Timestamp(1528854260, 1),

"electionDate" : ISODate("2018-06-13T01:44:20Z"),

"configVersion" : 1,

"self" : true

},

{

"\_id" : 1,

"name" : "192.168.4.52:27052",

"health" : 1,

"state" : 2,

"stateStr" : "SECONDARY",

"uptime" : 128,

"optime" : {

"ts" : Timestamp(1528854372, 1),

"t" : NumberLong(1)

},

"optimeDurable" : {

"ts" : Timestamp(1528854372, 1),

"t" : NumberLong(1)

},

"optimeDate" : ISODate("2018-06-13T01:46:12Z"),

"optimeDurableDate" : ISODate("2018-06-13T01:46:12Z"),

"lastHeartbeat" : ISODate("2018-06-13T01:46:17.067Z"),

"lastHeartbeatRecv" : ISODate("2018-06-13T01:46:16.501Z"),

"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",

"uptime" : 128,

"optime" : {

"ts" : Timestamp(1528854372, 1),

"t" : NumberLong(1)

},

"optimeDurable" : {

"ts" : Timestamp(1528854372, 1),

"t" : NumberLong(1)

},

"optimeDate" : ISODate("2018-06-13T01:46:12Z"),

"optimeDurableDate" : ISODate("2018-06-13T01:46:12Z"),

"lastHeartbeat" : ISODate("2018-06-13T01:46:17.067Z"),

"lastHeartbeatRecv" : ISODate("2018-06-13T01:46:16.358Z"),

"pingMs" : NumberLong(0),

"syncingTo" : "192.168.4.51:27051",

"configVersion" : 1

### 验证副本集配置

• 同步数据验证

– >db.getMongo().setSlaveOk() 允许从库查看数据，在每个库中都要设置

• 自动切换主库验证

– > rs.isMaster() 查看是否是主库

将51的mongodb服务关闭，会自动从52和53 里面选择一个主，可以通过re.status()查看

将51的mongodb服务开启，会自动变为新主的从

## MongoDB 文档管理

插入文档

save( )

• 格式

– db. 集合名 .save({ key:“ 值”, key:” 值” })

• 注意

– 集合不存在时创建集合,后插入记录

– \_id 字段值 已存在时 修改文档字段值

– \_id 字段值 不已存在时 插入文档

insert()

• 格式

– db. 集合名 .insert({key:“ 值”, key:” 值” })

• 注意

– 集合不存在时创建集合,后插入记录

– \_id 字段值 已存在时放弃插入

– \_id 字段值 不已存在时 插入文档

rs1:PRIMARY> db.c2.save({\_id:1,name:"lili"})

WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "\_id" : 1 })

rs1:PRIMARY> db.c2.find()

{ "\_id" : 1, "name" : "lili" }

rs1:PRIMARY> db.c2.save({\_id:1,name:"lilei"})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

rs1:PRIMARY> db.c2.find()

{ "\_id" : 1, "name" : "lilei" }

如果用insert会报错

rs1:PRIMARY> db.c2.insert({\_id:1,name:"lilei"})

WriteResult({

"nInserted" : 0,

"writeError" : {

"code" : 11000,

"errmsg" : "E11000 duplicate key error collection: userdb.c2 index: \_id\_ dup key: { : 1.0 }"

}

})

• 插入多条记录

db. 集合名 .insertMany(

[

{name:"xiaojiu“,age:19} ,

{name:"laoshi“,email:”yaya@tedu.cn”}

]

)

rs1:PRIMARY> db.c2.insertMany([{\_id:2,name:"hanmeimei"},{\_id:3,name:"dachui"}])

查询文档

• 显示所有行,默认一次只输出 20 行 输入 it 显示后续的

– db. 集合名 .find()

• 显示第 1 行

– > db. 集合名 .findOne()

• 指定查询条件并指定显示的字段

– > db. 集合名 .find ( { 条件 },{ 定义显示的字段 } )

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

– 0 不显示 1 显示

rs1:PRIMARY> db.user.find({},{\_id:0,name:1,passwd:1})

rs1:PRIMARY> db.user.find({name:"root"},{\_id:0,name:1,passwd:1})

行数显示限制

• limit( 数字 ) // 显示前几行

> db. 集合名 .find().limit(3)

• skip( 数字 ) // 跳过前几行

– > db. 集合名 .find().skip(2)

• sort( 字段名 ) // 排序

– > db. 集合名 .find().sort({age:1}) 1 升序 -1 降序

– > db.user.find({shell:"/sbin/nologin"},{\_id:0,name:1,uid:

1,shell:1}).skip(2).limit(2)

命令执行的顺序是

1、排序

2、跳过

3、显示前几行

rs1:PRIMARY> db.user.find({},{\_id:0,name:1,passwd:1}).limit(10)

{ "name" : "root", "passwd" : "x" }

{ "name" : "bin", "passwd" : "x" }

{ "name" : "daemon", "passwd" : "x" }

{ "name" : "adm", "passwd" : "x" }

{ "name" : "lp", "passwd" : "x" }

{ "name" : "sync", "passwd" : "x" }

{ "name" : "shutdown", "passwd" : "x" }

{ "name" : "halt", "passwd" : "x" }

{ "name" : "mail", "passwd" : "x" }

{ "name" : "operator", "passwd" : "x" }

rs1:PRIMARY> db.user.find({},{\_id:0,name:1,passwd:1}).skip(2).limit(10)

{ "name" : "daemon", "passwd" : "x" }

{ "name" : "adm", "passwd" : "x" }

{ "name" : "lp", "passwd" : "x" }

{ "name" : "sync", "passwd" : "x" }

{ "name" : "shutdown", "passwd" : "x" }

{ "name" : "halt", "passwd" : "x" }

{ "name" : "mail", "passwd" : "x" }

{ "name" : "operator", "passwd" : "x" }

{ "name" : "games", "passwd" : "x" }

{ "name" : "ftp", "passwd" : "x" }

rs1:PRIMARY> db.user.find({},{\_id:0,name:1,uid:1}).skip(2).limit(10).sort({uid:1})

{ "name" : "daemon", "uid" : 2 }

{ "name" : "adm", "uid" : 3 }

{ "name" : "lp", "uid" : 4 }

{ "name" : "sync", "uid" : 5 }

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

{ "name" : "halt", "uid" : 7 }

{ "name" : "mail", "uid" : 8 }

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

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

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

rs1:PRIMARY> db.user.find({},{\_id:0,name:1,uid:1}).skip(2).limit(10).sort({uid:-1})

{ "name" : "student", "uid" : 1000 }

{ "name" : "polkitd", "uid" : 999 }

{ "name" : "libstoragemgmt", "uid" : 998 }

{ "name" : "colord", "uid" : 997 }

{ "name" : "saslauth", "uid" : 996 }

{ "name" : "chrony", "uid" : 995 }

{ "name" : "geoclue", "uid" : 994 }

{ "name" : "setroubleshoot", "uid" : 993 }

{ "name" : "gnome-initial-setup", "uid" : 992 }

{ "name" : "systemd-network", "uid" : 192 }

• 简单条件

– db. 集合名 .find({key:” 值” })

– db. 集合名 .find({key:” 值”, keyname:” 值” }) //逻辑与

– db.user.find({shell:"/bin/bash"})

– db.user.find({shell:"/bin/bash",name:"root"})

查询条件(续 1 )

• 范围比较

– $in 在...里

– $nin 不在...里

– $or 或

– > db.user.find({uid:{$in:[1,6,9]}})

– > db.user.find({uid:{$nin:[1,6,9]}})

– > db.user.find({$or: [{name:"root"},{uid:1} ]})

rs1:PRIMARY> db.user.find({name:"root",uid:0},{\_id:0,name:1,uid:1})

rs1:PRIMARY> db.user.find({uid:{$in:[1,6,7]}},{\_id:0,name:1,uid:1})

rs1:PRIMARY> db.user.find({uid:{$nin:[1,6,7]}},{\_id:0,name:1,uid:1})

rs1:PRIMARY> db.user.find({$or:[{uid:1},{uid:3}]},{\_id:0,name:1,uid:1})

• 正则匹配

– > db.user.find({name: /^a/ })

• 数值比较

– $lt $lte $gt $gte $ne

< <= > >= !=

– db.user.find( { uid: { $gte:10,$lte:40} } , {\_id:0,name:1,uid:1})

– db.user.find({uid:{$lte:5,}})

rs1:PRIMARY> db.user.find({$or:[{uid:{$lte:5}},{uid:{$gte:100}}]},{\_id:0,name:1,uid:1})

uid<=5或者uid>=100

• 匹配 null , 也可以匹配没有的字段

– > db.user.save({name:null,uid:null})

– > db.user.find({name:null})

{"\_id":5"), "name" : null, "uid" : null }

更新文档

update()

• 语法格式

– > db.集合名.update({条件},{修改的字段} )

注意:把文件的其他字段都删除了,只留下了password字段,且只修改与条件匹配的第 1 行 !!!

$set / $unset

• $set 条件匹配时,修改指定字段的值

– db.user.update({ 条件 },{$set: { 修改的字段 }})

– db.user3.update({name:"bin"},{$set:{password:"A"}})

• $unset 删除与条件匹配文档的字段

– db. 集合名 .update({ 条件 },{$unset:{key:values}})

– db.user3.update({name:"bin"},{$unset:{password:"A"}}) //这里删除时，"A"这里可以随便写

rs1:PRIMARY> db.user.find({uid:1},{\_id:0})

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

rs1:PRIMARY> db.user.update({uid:1},{$set:{passwd:"FFF"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

rs1:PRIMARY> db.user.find({uid:1},{\_id:0})

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

多文档更新

• 语法格式:默认只更新与条件匹配的第 1 行

– > db.user.update({ 条件 },{$set:{ 修改的字段 }} ,false,true)

– > db.user.update({name:“bin”},{$set:{password: "abc12123"}},false,true)

$inc

• $inc 条件匹配时,字段值自加或自减

– Db. 集合名 .update({ 条件 },{$inc:{ 字段名 : 数字 }}) //正整数自加 负整数自减!!!!

– db.user.update({name:"bin"},{$inc:{uid:2}}) 字段值自加 2

– db.user.update({name:“bin”},{$inc:{uid:-1}}) 字段自减 1

$push / $addToSet

• $push 向数组中添加新元素

– db. 集合名 .update({ 条件 },{$push:{ 数组名 :“ 值” }})

– db.user.insert({name:"bob",likes:

["a","b","c","d","e","f"]})

– db.user.update({name:“bob”},{$push:{likes:“w"}})

• $addToSet 避免重复添加

– db. 集合名 .update({ 条件 },{$addToSet:{数组名:"值"}}) db.user.update({name:"bob"},{$addToSet:{likes:"f"}}) //值区分大小写

rs1:PRIMARY> db.c2.save({\_id:4,name:"xiaochui",likes:["a","b","c"]})

WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "\_id" : 4 })

rs1:PRIMARY> db.c2.find({name:"xiaochui"},{})

{ "\_id" : 4, "name" : "xiaochui", "likes" : [ "a", "b", "c" ] }

rs1:PRIMARY> db.c2.update({name:"xiaochui"},{$push:{likes:"d"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

rs1:PRIMARY> db.c2.find({name:"xiaochui"},{})

{ "\_id" : 4, "name" : "xiaochui", "likes" : [ "a", "b", "c", "d" ] }

rs1:PRIMARY> db.c2.update({name:"xiaochui"},{$addToSet:{likes:"d"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })

rs1:PRIMARY> db.c2.update({name:"xiaochui"},{$addToSet:{likes:"D"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

rs1:PRIMARY> db.c2.find({name:"xiaochui"},{})

{ "\_id" : 4, "name" : "xiaochui", "likes" : [ "a", "b", "c", "d", "D" ] }

$pop /$pull

• $pop 从数组头部删除一个元素

– db. 集合名 .update({ 条件 },{$pop:{ 数组名 : 数字 }})

– db.user.update({name:"bob"},{$pop:{likes:1}})

– db.user.update({name:"bob"},{$pop:{likes:-1}})

1 删除数组尾部元素 -1 删除数组头部元素

• $pull 删除数组指定元素

– db. 集合名 .update({ 条件 },{$pull:{ 数组名 : 值 }})

– db.user.update({name:"bob"},{$pull:{likes:"b"}})

rs1:PRIMARY> db.c2.update({name:"xiaochui"},{$pop:{likes:1}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

rs1:PRIMARY> db.c2.find({name:"xiaochui"},{})

{ "\_id" : 4, "name" : "xiaochui", "likes" : [ "a", "b", "c", "d" ] }

rs1:PRIMARY> db.c2.update({name:"xiaochui"},{$pop:{likes:-1}})

rs1:PRIMARY> db.c2.find({name:"xiaochui"},{})

{ "\_id" : 4, "name" : "xiaochui", "likes" : [ "b", "c", "d" ] }

rs1:PRIMARY> db.c2.find({name:"xiaochui"},{})

{ "\_id" : 4, "name" : "xiaochui", "likes" : [ "b", "c", "d", "c", "c", "c" ] }

rs1:PRIMARY> db.c2.update({name:"xiaochui"},{$pull:{likes:"c"}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

rs1:PRIMARY> db.c2.find({name:"xiaochui"},{})

{ "\_id" : 4, "name" : "xiaochui", "likes" : [ "b", "d" ] }

$drop/$remove

• $drop 删除集合的同时删除索引

– db.集合名.drop()

– db.user.drop()

• remove() 删除文档时不删除索引

– db. 集合名 .remove({}) // 删除所有文档

– db. 集合名 .remove({ 条件 }) // 删除与条件匹配的文档

– db.user.remove({uid:{$lte:10}})

– db.user.remove({})

数据库管理（DBA基础 DBA进阶 NoSQL）

RDBMS->MySQL Redis->Mongodb

面试偏理论 40% 60%

自主投dba简历

DBA面试题 偏理论较多

书籍：pdf

深入理解mysql核心

mysql技术内幕

深入理解mysql

mysql核心内幕

DBA基础

set global validate\_password\_policy=0

set global validate\_password\_length=6

连接mysql服务器

命令行 图形 php python

库

表

连接 mysql -uroot -p密码 -hip

数据类型：数值 字符 日期时间 枚举

字段约束条件 null default 键值key 额外设置autoincrement

修改表结构 alter table 表名 执行动作；

add (first after) drop modify change rename engine

键值：index primary key foreign key

fulltext unqiue 全文索引

存储引擎：myisam innodb

数据导入、导出

管理表记录 select insert update delete 条件

多表查询

图像管理工具 phpmyadmin

mysql-5.7

mysql-5.8

用户授权 grant revoke mysql库

数据备份和恢复

完全备份、恢复 mysqldump sql xxx.sql

增量备份、恢复 binlog 二进制 percona：innobackupex（xtrabackup）生产环境用的多，不锁表

shell脚本（执行mysqldump innobackupex）

od

xxd 来查看2进制文件

模式 mixed

DBA进阶

mysql主从同步：什么是主从同步 IO SQL线程 出错原因

一主一从 主从从 一主多从 主主结构

复制模式 全同步模式 异步模式 半同步模式

数据读写分离 maxscale/mycat + 主从同步

多实例

master和slave在同一台服务器

/dev/sdb1 /dev/sdc1挂载在不同的硬盘

优化mysql：

mysql ha集群 mha+一主多从

（使用多台数据库服务器提供数据库存储服务mysql）

51 52 53 54 55

视图 不同的访问用户看到不同的表字段 （查看 创建 ）

存储过程（查看 创建 删除 调用 变量 参数类型 in out inout 流程控制if while loop repeate 循环控制）

数据分片 mysql

client---->mycat

mysql

分片规则10种 -->算法 rule.xml java jdk

nosql redis mongodb

DBA工作日常

数据备份 脚本 主从

监控 zabbix---> mysql

存储备份的硬盘的使用空间

服务器的运行状况

mysql优化

内存 缓存 查询相关

my.cnf配置文件参数详解

show variables