**redis介绍（远程字典服务器）**

使用c语言编写

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模式备份数据

中文网站www.redis.cn

**常用指令操作**

set keyname keyvalues //存储变量

get keyname //获取变量

select 数据库编号0-15 //切换库

keys \* //打印所有变量

keys a? //打印指定变量

exists keyname //测试变量是否存在

ttl keyname //查看变量生存时间

type keyname //查看变量类型

moved keyname dbname //移动变量

expire keyname 10 //设置变量有效时间

del keyname //删除变量

flushall //删除内存所有变量

save //保存所有变量

shutdown //关闭redis服务

**配置文件解析**

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

timeout 0 //连接超时时间

tcp-keepalive 300 //长时间连接

daemonize yes //守护进程方式运行

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TTL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) //移除最近国企的key

noeviction //不删除，写满时报错

选项默认值

maxmemory <bytes> //最大内存

maxmemory-policy noeviction //定义使用策略

maxmemory-samples 5 //选取模版数据的个数（针对lru和ttl策略）

**安装**

从源码包编译安装

tar -xf redis-4.0.8.tar.gz

cd redis-4.0.8

make && make install

初始化配置（初始化完成默认开启服务）

端口：6379

主配置文件：/etc/redis/6379.conf

日志：/var/log/redis\_6379.log

数据库目录（内存往硬盘存储的目录）：/var/lib/redis/6379

pid文件：/var/run/redis\_6379.pid

启动程序：/usr/local/bin/redis-server

utils/install\_server.sh //初始化

启动/停止服务

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

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

redis-cli -h ip地址 -p 端口 shutdown

redis-cli -h ip地址 -p 端口

>shutdown

**连接数据库服务**

ps -C redis

netstat -utnlp | grep redis

redis-cli //连接本机的redis数据库服务

redis-cli -h ip地址 -p 端口 //连接远程的redis数据库服务器

**设置连接密码**

vim /etc/redis/6379.conf

requirepass 密码

/etc/init.d/redis\_<portnumber> restart

redis-cli -h ip地址 -p 端口

>auth 密码

redis-cli -h ip地址 -p 端口 -a 密码

**部署LNMP+redis**

部署LNMP

安装nginx和PHP

yum -y install gcc gcc-c+ pcre-devel zlib-devel

tar -xf nginx-1.12.2.tar.gz

cd nginx-1.12.2

./configure --prefix=/usr/local/nginx

make && make install

yum -y install php-common

yum -y install php-fpm-5.4.16-42.el7.x86\_64.rpm

修改nginx配置文件，使其支持PHP

启动服务

/usr/local/nginx/sbin/nginx

netstat -utnlp | grep nginx

systemctl start php-fpm

netstat -utnlp | grep 9000

配置PHP支持redis

运行redis服务

装包并启动服务

tar -xf redis-4.0.8.tar.gz

cd redis-4.0.8

make

make install

./utils/install\_server.sh //初始化

service redis\_6379 start

配置PHP支持redis

安装php扩展

yum -y install autoconf automake

yum -y install php-devel-5.4.16-46.el7.x86\_64.rpm

tar -xf phpredis-2.2.4.tar.gz

cd phpredis-2.2.4

phpize

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

make && make install

vim /etc/php.ini

extension\_dir=”扩展路径”

extension=”redis.so”

systemctl start php-fpm

测试配置

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

<?php

$redis=new redis();

$redis->connect(‘127.0.0.1’,6379);

$redis->set(‘redistest’,’666666’);

echo $redis->get(redistest);

?>

**redis高可用集群**

准备环境

六台redis

IP及端口规划

A：192.168.1.221 6351

B：192.168.1.238 6352

C：192.168.1.213 6353

D：192.168.1.83 6354

E：192.168.1.252 6355

F：192.168.1.72 6356

工作原理

用户存数据到数据库根据hash slot，放在不同的数据库服务器内，取得时候，在根据相应的公式进行计算，再去相应的数据库读出来，从库是同步主库的数据，所以不需要hash slot，当主机宕机后，从库自动提升为主库，原来主库重新启动后，将配置为当前主库的从库。当主库宕机后，没有从库，集群将不能正常运转。存储：key/CRC16 值%16384查看余数在哪个hash槽的范围内，就存到哪个节点，hash槽范围0-16383。

创建redis集群

装包

为6台主机安装并运行redis服务

yum -y install gcc gcc-g++

tar -xf redis-4.0.8.tar.gz

cd redis-4.0.8

make && make install

调整配置文件

vim /etc/redis/redis.conf

bind IP地址 //只写物理接口IP地址

daemonize yes//守护进程运行方式

port xxxx//端口号不要使用默认的6379

cluster-enabled yes//启用集群

cluster-config-file nodes-xxxx.conf//指定集群信息

cluster-node-timeout 5000//请求超时5秒

查看服务信息

/etc/init.d/redis\_6379 status

netstat -utnlp | grep redis-server

查看集群信息

redis-cli -c -h ip地址 -p 端口

>cluster info

>cluster nodes

创建集群

在选中的一台redis服务器上，执行创建集群脚本

部署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

gem install redis-3.2.1.gem

cd redis-3.2.0/src

./redis-trib.rb create --replicas 1 host:port host:port ......

--replicas 1，自动为每一个master节点分配1个slave节点

创建集群

./redis-trib.rb create --replicas 1 192.168.1.221:6351 192.168.1.238:6352 192.168.1.213:6353 192.168.1.83:6354 192.168.1.252:6355 192.168.1.72:6356

查看集群信息

任意一台主机访问本机的redis服务即可查看

cluster info//查看集群信息

cluster nodes//查看集群节点信息

测试集群

在客户端访问任意一台masster主机存数据

redis-cli -c -h ip地址 -p 端口

set key values//存数据

在客户端访问任意一台master主机取数据（从库不允许查看键值）

redis-cli -c -h ip地址 -p 端口

get key//取数据

管理集群

管理命令

redis-cli命令

查看命令帮助

redis-cli -h

常用选项

-h ip地址

-p 端口

-c 集群模式

redis-trib-rb脚本

语法格式

redis-trib.rb 选项 参数

选项

create 创建集群

add-node 添加master主机

check 检测集群

reshard 重新分片

add-noide --slave 添加slave主机

del-node 删除主机

master选举测试

选举master主机

停止master主机的redis服务

master宕机后对应的slave自动被选举为master

原master启动后 会自动配置为当前master的slave

查看集群主机的信息

redis-cli -h master\_ip -p master\_port

redis-trib.rb check IP地址:端口

添加新节点

添加master主机

部署一台新的redis服务器

装包

初始化

启用集群配置重启服务

添加master主机步骤

添加master主机

添加时指定主机角色，默认新主机为master

./redis-trib.rb add-node 新主机IP:端口 192.168.1.221:63511

检查主机

./redis-trib.rb check 192.168.1.221:6351

新主机角色为master

无槽位数量

重新分片

./redis-trib.rb reshard 192.168.1.221:6351

从当前的集群中拿出多少个hash slots

新主库的ID值

all从当前的集群中所有主机里获取hash slots

yes确认信息

添加slave主机（哪个主机少往哪个添加，如果一个，随机分配，可以指定，如果不指定主节点的id的话，会吧新节点随机添加为节点最少的主的从）

部署一台新redis服务器

装包

初始化

启用集群配置

运行服务

添加slave主机

./redis-trib.rb add-node --slave [--master-id id值] 新主机ip地址:端口 192.168.1.221:6351

移除节点

移除master主机

配置步骤

重新分片释放占用的hash槽、

指定移出hash槽个数

指定接收hash槽主机ID

指定移出hash槽主机ID

rdis-trib.rb reshard 192.168.1.221:6351

move

rece ID

source ID（如果不需要的话写done）

移除master主机

redis-trib.rb del-node 192.168.1.221:6351 要移出master主机ID

移除slave主机

从节点主机没有槽位范围，直接移出即可

rediis-trib.rb delnode 192.168.1.221.6351 要移除主机ID值

**redis主从复制**

主从复制模式：一主一从、一主多从、主从从

工作原理：从库发送同步请求到主库，主库创建一个线程将修改过的数据存放在一个文件里，然后将文件发送给从库，从库保存到内存并执行完成首次同步

主从复制缺点：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

配置从库

配置

redis服务启动后，默认都是master服务器（用info replication可以查看相关信息）

127.0.0.1:6379> slaveof 主库IP地址 端口号（手动设为从库，临时生效，重启服务后失效）

修改配置文件（永久生效）

vim /etc/redis/6379.conf

bind 物理接口地址

slaveof 主库IP 主库端口

masterauth 主库密码（根据情况可有可无）

/etc/init.d/redis\_6379 start

删除

重启服务或slaveof no one（针对临时配置操作）

配置文件取消相应选项（永久取消）

反客为主

主机宕机后，手动将从库设置为主库（针对临时配置操作，slaveof no one）

哨兵模式

主机宕机后，从库自动升级为主库

在slave主机编辑sentinel.conf文件

vim /etc/sentinel.conf

sentinel monitor mymaster 192.168.1.11 6379 1

（sentinel monitor 主机名 Ip地址 端口 票数）

sentinel auth-pass mymaster 123456（针对主库有密码需要加该设置）

bind 0.0.0.0

protected-mode no

daemonize yes

port 26379

dir /tmp

sentinel down-after-milliseconds mymaster 3000

sentinel parallel-syncs mymaster 1

sentinel failover-timeout mymaster 5000

sentinel client-reconfig-script mymaster /usr/local/redis/conf/reconfig.sh

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：主机宕机后，票数大于1的主机被升级为主库

在slave主机运行哨兵模式

redis-sentinel /etc/sentinel.conf

**redis持久化RBD/AOF**

**如果同时存在AOF文件和RDB文件，会先读AOF文件，不会读RDB文件**

持久化之RDB（默认模式）（快照、覆盖）

RDB介绍

数据持久化方式之一

按照时间间隔，将内存中的数据集写入硬盘（默认300秒）

术语叫snapshot快照

恢复时，将快照直接读入内存

工作过程

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，持久化结束后，再用这个临时文件替换上次持久化（dump.rdb），过程中主进程不做任何IO操作

相关配置参数

文件名（配置文件中）

dbfilename “dump.rdb” //文件名（服务关闭的时候或者到达指定的时间就会创建该文件）

save ”” //禁用RDB

数据从内存保存到硬盘的频率（可手动修改）

save 900 1//900秒内且有1次修改  
 save 300 10//300秒内且有10次修改

save 60 10000//60秒内且有10000次修改

手动立刻存盘（命令行下输入立即保存）

save//阻塞写存盘（存的时候不能写）

bgsave//不阻塞写存盘（存的时候可以写）

压缩

rdbcom pression yes|no

在存储快照后，使用crc16算法做数据校验

rdbchecksum yes|no

bgsave出错时停止写操作

stop-writes-on-bgsave-error yes|no

使用RDB为文件恢复数据

停止服务时，会自动生成dump.rdb文件

备份数据：

拷贝dump.rdb文件到其他位置

cp dump.rdb /etc/root/dump.rdb

恢复数据

停止服务，删除文件，拷贝备份文件到数据库目录，启动服务

service redis\_6379 stop

rm -rf /var/lib/redis/6379/dump.rdb

cp /etc/root/dump.rdb /var/lib/redis/6379/dump.rdb

service redis\_6379 start

RDB优缺点

优点：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

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（类似于mysql里的binlog（记录除读之外的命令）日志）（增量）

AOF介绍

只做追加操作，记录redis服务所有写操作

不断将新的写操作，追加到文件末尾

使用cat命令可以查看文件内容

工作过程

先将数据写入到aof\_buf缓冲区，然后由这个缓冲写入磁盘，异步的性质，定期会按照规则重写aof文件。

相关配置参数（配置文件中）

文件名

appendiflename “appendonly.aof” //指定文件名

appendonly yes//启用aof，默认no

aof文件记录写操作的方式

appendfsync always//有新写操作立即记录

appendfsync everysec//每秒记录一次（默认）

appendfsync no//从不记录

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

auto-aof-rewrite-percentage 100

默认配置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-min-size 64mb

修复aof文件（把文件恢复到最后一次的正确操作）

redis-check-aof --fix appendonly.aof

使用aof恢复数据

备份数据

备份appendonly.aof文件到其他位置

cp appendonly.aof /root/appendonly.aof

恢复数据

停止服务，拷贝备份文件到数据库目录，启动redis服务

service redis\_6379 stop

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

service redis\_6379 start

AOF优缺点

优点：可以灵活设置持久化方式，同步持久化appendfsync always或异步持久化appendfsync verysec

出现意外宕机时，可能丢失1秒数据

缺点：持久化文件的体积通常大于RDB方式

执行fsync策略时的速度可能会比RDB方式慢

**数据类型**

string字符串

字符串操作

set key value [ex seconds] [px millisenconds] [nx|xx]

ex,px设置key及值，过期时间可以使用秒或毫秒为单位

nx不存在，当变量不存在时，就赋值

xx存在，当变量存在时，就赋值（默认是xx）

setrange key offset（特定位的值） value（修改变量的值，修改一部分）

从偏移量开始复写key的特定位的值（从0开始）

srtlen key统计字串长度

append key value

存在则追加，不存在则创建key和value，返回key长度

setbit key offset value

对key所存储字串，设置或删除特定偏移量上的位（bit）

value值可以为1或0（0表示不存在，1表示存在），offset为0~2^32之间

key不存在，则创建新key

bitcount key

统计字串中被设置为1的比特位数量

decr key

将key中的值减1，key不存在则先设置为0，再减1

decrby key decrement

将key中的值，减去decrement

get key

返回key参与并抽根烟的字符串值，若key不存在则返回nil

若key的值不是字串，则返回错误，get只能处理字串

getrange key start end

返回字串值中的子字串，截取范围为start和end

负数的偏移量表示从末尾开始计数，-1表示最后一个字符，-2表示倒数第2个字符

incr key

将key的值加1，如果key不存在，则初始化为0后再加1

主要应用为计数器

incrby key increment

将key的值增加increment

incrbyfloat key increment

为key中所存储的值加上浮点数增量increment

mget key [key...]

获取一个或多个key的值，空格分隔

mset key value [key value...]

设置多个key及值，空格分隔

list列表（类似于shell里的数组）

简介

redis的list是一个字符队列

先进先出

一个key可以有多个值

list列表操作

lpush key value [value...]

将一个或多个value插入到列表key的表头

key不存在，则创建key

lrange key start stop

从开始位置读取key的值到stop结束

lpop key

移除并返回列表元素数据，key不存在则返回nil

llen key

返回列表key长度

lindex key index

返回列表中第Index个值

lset key index value（修改已有下标的值）

将key中index位置的值改为value

rpush key value [value...]

将value插入到key的末尾

rpop key

删除并返回key末尾的值

hash表（让定以的变量可以存储多个key和values）

简介

redis hash是一个strring类型的field（列）和values的映射表

一个key可以对应多个列，一个列对应一个值

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 fied value

将hash表中的field值设置为value

hget key field

获取hash表中field的值

hmset key field value [field value...]

同时给hash表中的多个field赋值

hmget key field [field...]

返回hash表中多个field的值

hkeys key

返回hash表中所有field名称

hgetall key

返回hash表中所有field的值

havals key

返回hash表中所有field的值

hdel key field [field...]

删除hash表中多个field的值，不存在则忽略

总结

数据类型

字符类型、hash表类型、list列表类型

管理命令

del key [key...]

删除一个或多个key

exists key

检测一个key是否存在

expire key seconds

设置key的生命周期

persist key

设置key永不过期

ttl key

查看key的生存周期