

一、简介

特点

二、Redis

linux安装redis

redis客户端

三、redis基本知识命令

测试redis服务性能

查看redis服务是否正常运行

查看redis服务器统计信息

redis的数据库实例

存数据 set key value

取数据 get key

切换数据库命令 select db

查看当前数据库实例中key的数量 dbsize

查看当前数据库实例中所有的key keys *

清空数据库实例 flushdb

清空所有数据库实例 flushall

查查看redis中配置信息 config get *

看redis中指定配置信息 config get parameter

四、redis的五种数据结构

字符串类型 String

列表类型 list

集合类型 set

哈希类型 hash

有序集合类型 zset (sorted set)

五、Redis中的操作命令

redis的key的操作命令

keys

exists

move

ttl

expire

type

rename

del

redis的有关String类型的操作命令 单key单value

set

get

append

strlen

incr

decr

incrby

decrby

getrange

setrange

setex

setnx

mset

mget

msetnx

redis的有关List类型的操作命令 单key 多有序value

lrange

lpush(插头)

rpush(插尾)

- lpop
- rpop
- llen
- lrem
- ltrim
- lset
- linsert

redis的有关set类型的操作命令 单key 多无序value

- sadd
- smembers
- sismember
- scard
- srem
- randmember
- spop
- smove
- sdiff
- sinter
- sunion

redis的有关hash类型的操作命令 单key 对象

- hset
- hget
- hmset
- hmget
- hgetall
- hdel
- hlen
- hexists
- hkeys
- hvals
- hincrby
- hincrbyfloat
- hsetnx

redis的有关zset类型的操作命令 有序集合

- zadd
- zrange
- zrangby
- zrem
- zcard
- zcount
- zrank
- zscore
- zrevrank
- zrevrange
- zrevrangebyscore

六、Redis的配置文件

- Redis.conf存放位置
- Redis的网络相关配置
- Redis的常规配置
- Redis的安全配置

七、redis的持久化

- RDB策略
- AOF策略

八、Redis的事务

- multi命令
- exec命令
- discard命令
- watch命令

unwatch命令

九、redis消息的发布与订阅

Redis发布订阅

subscribe

publish

psubscribe

十、redis的主从复制

主从复制

一主二从

原理

搭建

- 1.修改三份redis.conf:
- 2.分别使用三个redis配置文件，启动三个redis服务
- 3.通过redis客户端分别连接三台redis服务
- 4.查看三台redis服务在集群中的主从角色
- 5.现在6379进行写操作
- 6.设置主从关系：设从不设主
- 7.全量复制：一旦主从关系确定，会自动把主机上已有的数据同步复制到从库
- 8.增量复制:主库写数据会自动同步到从库。
- 9.主写从读，读写分离
- 10.主机宕机:从机原地待命
11. 从机宕机，主机少一个从机，其他从机不变
- 12.主机宕机，从机上位
- 13.主机恢复、天堂变地狱

十一、redis哨兵模式：主机宕机、从机上位的自动版

- 1.搭建一主二从集群架构（上节）
- 2.提供哨兵配置文件
- 3.启动哨兵服务
- 4.关闭主机6379，哨兵程序自动选择从机上位

十二、jedis操作Redis

一、简介

Redis是一种数据库，能够存储数据、管理数据的一种软件。

泛指非关系型的数据库。随着互联网web2.0网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。NoSQL数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题，包括超大规模数据的存储。

（例如谷歌或Facebook每天为他们的用户收集万亿比特的数据）。**这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。**

特点

易扩展

NoSQL数据库种类繁多，但是一个共同的特点都是去掉关系数据库的关系型特性。

数据之间无关系，这样就非常容易扩展。也无形之间，在架构的层面上带来了可扩展的能力。

大数据量高性能

NoSQL数据库都具有非常高的读写性能，尤其在大数据量下，同样表现优秀。这得益于它的无关系性，数据库的结构简单。

一般MySQL使用Query Cache，每次表的更新Cache就失效，是一种大粒度的Cache,在针对web2.0的交互频繁的应用，Cache性能不高。而NoSQL的Cache是记录级的，是一种细粒度的Cache，所以NOSQL在这个层面上来说就要性能高很多了

多样灵活的数据模型

NoSQL无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。而在关系数据库里，增删字段是一件非常麻烦的事情。如果是非常大数据量的表，增加字段简直就是一个噩梦

传统RDBMS和NOSQL

RDBMS

1. 高度组织化结构化数据-结构化查询语言（SQL）
2. 数据和关系都存储在单独的表中。
3. 数据操纵语言，数据定义语言-严格的一致性
4. 基础事务

NoSQL

1. 代表着不仅仅是SQL
2. 没有声明性查询语言
3. 没有预定义的模式
4. 键-值对存储，列存储，文档存储，图形数据库
5. 最终一致性，而非ACID属性
6. 非结构化和不可预知的数据-
7. CAP定理 高性能，高可用性和可伸缩性

Redis中的数据大部分时间都是存储内存中的，适合存储频繁访问、数据量比较小的数据

二、Redis

linux安装redis

第一步：下载redis

<https://redis.io/>



第二步：使用Xftp工具上传redis-5.0.2.tar.gz到linux 系统。

第三步：解压redis-5.0.2.tar.gz到/opt目录

```
[root@bogon softwares]# ls
apache-tomcat-9.0.0.M26.tar.gz  mysql-5.7.18-linux-glibc2.5-x86_64.tar.gz
jdk-8u121-linux-x64.tar.gz      redis-5.0.2.tar.gz
[root@bogon softwares]# tar -zxvf redis-5.0.2.tar.gz -C /opt
```

第四步：编译redis，进入解压目录，并且执行make命令：

```

[root@192 local]# pwd
/usr/local
[root@192 local]# ls
make-4.0.1  bin  etc  games  include  jdk1.8.0_121  lib  lib64  libexec  mysql-5.7.18  redis-5.0.2  sbin  share  src
[root@192 local]# cd redis-5.0.2/
[root@192 redis-5.0.2]# ls
00-RELEASENOTES  BUGS  CONTRIBUTING  COPYING  deps  INSTALL  Makefile  MANIFESTO  README.md  redis.conf  runtest  runtest-cluster  runtest-sentinel  sentinel.conf  src  tests  utils
[root@192 redis-5.0.2]# make
cd src && make all
make[1]: Entering directory '/usr/local/redis-5.0.2/src'
cc Makefile.dep
make[1]: Leaving directory '/usr/local/redis-5.0.2/src'
make[1]: Entering directory '/usr/local/redis-5.0.2/src'
rm -rf redis-server redis-sentinel redis-cli redis-benchmark redis-check-rdb redis-check-aof *.a *.gdb *.gcn *.gcov redis.info lcov-html Makefile.dep dict-benchmark
(cd ../deps && make distclean)

```

报错: gcc命令未找到

```

(cd jemalloc && [ -f Makefile ] && make distclean) > /dev/null || true
(rm -f .make-*)
(echo "" > .make-cflags)
(echo "" > .make-ldflags)
MAKE hiredis
cd hiredis && make static
make[3]: Entering directory '/usr/local/redis-5.0.2/deps/hiredis'
gcc -std=c99 -pedantic -c -O3 -fPIC -Wall -W -Wstrict-prototypes -Wwrite-strings -g -ggdb net.c
make[3]: gcc: Command not found
make[3]: *** [net.o] Error 127
make[3]: Leaving directory '/usr/local/redis-5.0.2/deps/hiredis'
make[2]: *** [hiredis] Error 2
make[2]: Leaving directory '/usr/local/redis-5.0.2/deps'
make[1]: [persist-settings] Error 2 (ignored)
cc adlist.o
/bin/sh: cc: command not found
make[1]: *** [adlist.o] Error 127
make[1]: Leaving directory '/usr/local/redis-5.0.2/src'
make: *** [all] Error 2

```

第五步: 安装gcc。

什么是 gcc ?

gcc是GNU compiler collection的缩写, 它是Linux下一个编译器集合(相当于javac), 是c或c++程序的编译器。

怎么安装gcc?

在有外网的情况下, 使用yum进行安装。执行命令: `yum -y install gcc`。

执行`gcc -v`查看Linux内核版本

```

[root@192 Packages]# gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/lto-wrapper
Target: x86_64-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-bootstrap --enable-shared --enable-threads=posix --enable-checking=release --with-system-zlib --enable-_cx_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-linker-hash-style=gnu --enable-languages=c,c++,objc,obj-c++,java,fortran,ada,go,lto --enable-plugin --enable-initfini-array --disable-libgck --with-isl=/build/buildd/r/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/isl-install --with-cloog=/build/buildd/r/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/cloog-install --enable-gnu-inirect-function --with-tune=generic --with-arch_32=x86_64 --build=x86_64-redhat-linux
Thread model: posix
gcc version 4.8.5 20150623 (Red Hat 4.8.5-11) (GCC)
[root@192 Packages]#

```

第六步: 再次回到redis解压目录执行make命令进行编译

```

[root@192 redis-5.0.2]# pwd
/usr/local/redis-5.0.2
[root@192 redis-5.0.2]# make
cd src && make all
make[1]: Entering directory '/usr/local/redis-5.0.2/src'
cc adlist.o
In file included from adlist.c:34:0:
zmalloc.h:50:31: fatal error: jemalloc/jemalloc.h: No such file or directory
#include <jemalloc/jemalloc.h>
compilation terminated.
make[1]: *** [adlist.o] Error 1
make[1]: Leaving directory '/usr/local/redis-5.0.2/src'
make: *** [all] Error 2
[root@192 redis-5.0.2]#

```

第七步: 进行清理工作

启动方式

① 前台启动 redis-server

```
[root@192 redis-5.0.2]# redis-server
12832:C 26 Nov 2018 20:51:41.880 # o00000000000 Redis is starting o00000000000
12832:C 26 Nov 2018 20:51:41.880 # Redis version=5.0.2, bits=64, commit=00000000, modified=0, pid=12832, just started
12832:C 26 Nov 2018 20:51:41.880 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
12832:M 26 Nov 2018 20:51:41.950 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 5.0.2 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 12832

http://redis.io

12832:M 26 Nov 2018 20:51:41.975 * WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
12832:M 26 Nov 2018 20:51:41.975 * Server initialized
12832:M 26 Nov 2018 20:51:41.975 * WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory=1' for this to take effect.
12832:M 26 Nov 2018 20:51:41.975 * WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis THP is disabled.
```

②后台启动 redis-server &

```
[root@192 redis-5.0.2]# redis-server &
111 12845
[root@192 redis-5.0.2]# 12845:C 26 Nov 2018 20:52:12.105 # o00000000000 Redis is starting o00000000000
12845:C 26 Nov 2018 20:52:12.105 # Redis version=5.0.2, bits=64, commit=00000000, modified=0, pid=12845, just started
12845:C 26 Nov 2018 20:52:12.105 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
12845:M 26 Nov 2018 20:52:12.105 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 5.0.2 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 12845

http://redis.io
```

③根据配置文件启动 启动命令 配置文件 &

```
[root@192 redis-5.0.2]# redis-server redis.conf &
111 3963
[root@192 redis-5.0.2]# 3963:C 27 Nov 2018 13:55:27.797 # o00000000000 Redis is starting o00000000000
3963:C 27 Nov 2018 13:55:27.797 # Redis version=5.0.2, bits=64, commit=00000000, modified=0, pid=3963, just started
3963:C 27 Nov 2018 13:55:27.797 # Configuration loaded
3963:M 27 Nov 2018 13:55:27.799 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 5.0.2 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 3963

http://redis.io
```

注意：如果修改了redis的配置文件redis.conf，必须在启动时指定配置文件，否则修改无效

第十一步：关闭Redis服务

关闭方式：

①使用redis客户端关闭，向服务器发出关闭命令

任意目录下执行 指令redis-cli shutdown

推荐使用这种方式，redis先完成数据操作，然后再关闭。

例如：

```
[root@192 src]# ps -ef|grep redis
root      12897   7918   0 20:55 pts/1    00:00:00 redis-server *:6379
root      13029   7918   0 21:02 pts/1    00:00:00 grep --color=auto redis
[root@192 src]# redis-cli shutdown
[1]+  Done                  nohup redis-server (wd: /usr/local/redis-5.0.2/src)
(wd now: /usr/local/redis-5.0.2/src)
[root@192 src]# ps -ef|grep redis
root      13040   7918   0 21:02 pts/1    00:00:00 grep --color=auto redis
[root@192 src]#
```

②kill pid 或者 kill -9 pid

这种不会考虑当前应用是否有数据正在执行操作，直接就关闭应用。

先使用 ps -ef | grep redis 查出进程号，在使用 kill pid

```
[root@192 src]# ps -ef|grep redis
root      13069    7918  2 21:04 pts/1    00:00:00 redis-server *:6379
root      13074    7918  0 21:04 pts/1    00:00:00 grep --color=auto redis
[root@192 src]# kill -9 13069
[root@192 src]# ps -ef|grep redis
root      13082    7918  0 21:05 pts/1    00:00:00 grep --color=auto redis
[1]+  Killed                  redis-server
[root@192 src]#
```

redis服务已经开启

关闭redis服务进程

redis服务已经关闭

redis客户端

用来连接redis服务，向redis服务端发送命令，并且显示redis服务处理结果。

redis-cli:是redis自带客户端，使用redis-cli就可以启动redis的客户端程序，无需用户名密码

```
[root@localhost bin]# redis-cli
127.0.0.1:6379>
```

redis-cli 默认连接本机127.0.0.1本机的**6379端口**上的redis服务

redis-cli -p 端口号 默认连接本机127.0.0.1本机的**指定端口**上的redis服务

redis-cli -h ip地址 -p 端口号 连接指定ip主机上的**指定端口**上的redis服务

exit 退出客户端

三、redis基本知识命令

测试redis服务性能

redis-benchmark

查看redis服务是否正常运行

ping 正常返回pong

```
127.0.0.1:6379> ping
PONG
```

查看redis服务器统计信息

info 查看所有的统计信息

info [信息段] 查看指定的统计信息，如info Replication


```
127.0.0.1:6379> info
# Server
redis_version:5.0.2
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:240d409f9ec813d
redis_mode:standalone
os:Linux 3.10.0-1160.el7.x86_64 x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:4.8.5
process_id:10681
run_id:bb07813caff5f3c25681d6580ae691aee9c410f7
tcp_port:6379
uptime_in_seconds:5505
uptime_in_days:0
hz:10
configured_hz:10
lru_clock:6063331
executable:/opt/redis-5.0.2/redis-server
config_file:/opt/redis-5.0.2/redis.conf

# Clients
connected_clients:2
client_recent_max_input_buffer:2
```

redis的数据库实例

作用类似于mysql的数据库实例，redis中的数据库实例只能由redis服务来创建和维护，开发人员不能修改和自行创建数据库实例；

默认情况下，redis会**自动创建16个数据库实例**，并且给这些数据库实例进行编号，从0开始一直到15。使用时通过编号来使用数据库；

可以通过配置文件，指定redis自动创建的数据库个数；

redis每一个数据库实例本身占用的存储空间很少，所以也不造成空间的浪费

默认情况下，redis客户端连接的编号是0的数据库实例；可以使用select index切换数据库实例

```
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> get username
(nil)
127.0.0.1:6379[1]> select 0
OK
127.0.0.1:6379> get username
"liuqiang"
```

存数据 set key value

取数据 set key

切换数据库命令 `select db`

查看当前数据库实例中key的数量 `dbsize`

```
127.0.0.1:6379> dbsize
(integer) 5
```

查看当前数据库实例中所有的key `keys *`

```
127.0.0.1:6379> keys *
1) "username"
2) "mylist"
3) "key: __rand_int__"
4) "myset: __rand_int__"
5) "counter: __rand_int__"
```

清空数据库实例 `flushdb`

清空所有数据库实例 `flushall`

查查看redis中配置信息 `config get *`

看redis中指定配置信息 `config get parameter`

四、redis的五种数据结构

程序是用来处理数据的，redis是用来存储数据的；程序处理完的数据要存储到redis中，不同特点的数据要存储在redis中不同类型的数据结构。

字符串类型 String

字符串类型是Redis中最基本的数据结构，它能存储任何类型的数据，包括二进制数据，序列化后的数据，JSON化的对象甚至是一张图片。最大512M。

key	value
username	张三和李四

列表类型 list

Redis列表是简单的字符串列表，按照插入顺序排序，元素可以重复。你可以添加一个元素到列表的头部（左边）或者尾部（右边），底层是个链表结构。

key	value
region	北京 上海 天津

集合类型 set

Redis的Set是string类型的无序无重复集合。

key	value
framework	spring
	mybatis
	struts

哈希类型 hash

单key：对象（属性：值）

Redis hash 是一个string类型的field和value的映射表，hash特别适用于存储对象。

key	loginuser
field	value
uname	张三
times	5
region	北京

有序集合类型 zset (sorted set)

Redis **有序**集合zset和集合set一样也是string类型元素的集合，且**不允许重复**的成员。

不同的是zset的每个元素都会关联一个分数（分数可以重复），redis通过分数来为集合中的成员进行从小到大的排序。

五、Redis中的操作命令

redis的key的操作命令

keys

语法：keys pattern

作用：查找所有符合模式pattern的key. pattern可以使用通配符。

查看所有符合通配符的key、pattern可以使用通配符

通配符：

1. *：表示**0或多个**字符，例如：**keys *** 查询所有的key。**keys k*** 查询所有的以k打头的key。
2. ?：表示**单个**字符，例如：wo?d，匹配 word，wood
3. []：表示选择[]内的**一个**字符，例如wo[or]d，匹配word，wood，不匹配wold、woord，只能是一个字符，括号内两个字符完全符合都不可以

exists

语法：exists key [key...]或exists key1 key2

作用：判断key是否存在

返回值：整数，存在key返回1，其他返回0。使用多个key，返回存在的key的数量。

move

语法: move key db

作用: 移动key到指定的数据库, 移动的key在原库被删除。

返回值: 移动成功返回1, 失败返回0.

ttl

语法: ttl key

作用: 查看key的剩余生存时间 (ttl: time to live) , 以秒为单位。

返回值:

-1 : 没有设置key的生存时间, key永不过期。

-2: key不存在

expire

语法: expire key seconds

作用: 设置key的生存时间, 超过时间, key自动删除。单位是秒。

返回值: 设置成功返回数字 1, 其他情况是 0 。

type

语法: type key

作用: 查看key所存储值的数据类型

返回值: 字符串表示的数据类型

1. none (key不存在)
2. string (字符串)
3. list (列表)
4. set (集合)
5. zset (有序集)
6. hash (哈希表)

rename

语法: rename key newkey

作用: 将key改为名newkey。当 key 和 newkey 相同, 或者 key 不存在时, 返回一个错误。

当 newkey 已经存在时, RENAME 命令将覆盖旧值。

del

语法: del key [key...]

作用: 删除存在的key, 不存在的key忽略。

返回值: 数字, 删除的key的数量。

redis的有关String类型的操作命令 单key单value

字符串类型是Redis中最基本的数据类型，它能存储任何形式的字符串，包括二进制数据，序列化后的数据，JSON化的对象甚至是一张图片。

字符串类型的数据操作总的思想是通过key操作value，key是数据标识，value是我们感兴趣的业务数据。

set

语法：set key value

功能：将字符串值 value 设置到 key 中，如果key已存在，后放的值会把前放的值覆盖掉。

返回值：OK表示成功

get

语法：get key

功能：获取 key 中设置的字符串值

返回值：key存在，返回key对应的value；key不存在，返回nil

append

语法：append key value

功能：如果 key 存在，则将 value 追加到 key 原来旧值的末尾 如果 key 不存在，则将key 设置值为 value

返回值：追加字符串之后的总长度(字符个数)

strlen

语法：strlen key

功能：返回 key 所储存的字符串值的长度

返回值：如果key存在，返回字符串值的长度；

key不存在，返回0

incr

语法：incr key

功能：将 key 中储存的数字值加1，如果 key 不存在，则 key 的值先被初始化为 0 再执行incr操作。所要求的value必须是数值

返回值：返回加1后的key值

decr

语法：decr key

功能：将 key 中储存的数字值减1，如果 key 不存在，则 key 的值先被初始化为 0 再执行 decr 操作。

返回值：返回减1后的key值

incrby

语法: incrby key offset

功能: 将 key 所储存的值加上增量值, 如果 key 不存在, 则 key 的值先被初始化为 0 再执行 INCRBY 命令。

返回值: 返回增量之后的key值。

decrby

语法: decrby key offset

功能: 将 key 所储存的值减去减量值, 如果 key 不存在, 则 key 的值先被初始化为 0 再执行 DECRBY 命令。

返回值: 返回减量之后的key值。

getrange

语法: getrange key startIndex endIndex

功能: 获取 key 中字符串值从 startIndex 开始到 endIndex 结束的子字符串,包括startIndex和endIndex, **负数表示从字符串的末尾开始**, -1 表示最后一个字符。

下标从0开始

setrange

语法: setrange key offsetIndex value

功能: 用value覆盖key的存储的值从offset开始。

返回值: 修改后的字符串的长度。

setex

语法: setex key seconds value

功能: 设置key的值, 并将 key 的生存时间设为 seconds (以秒为单位), 如果key已经存在, 将覆盖旧值。

返回值: 设置成功, 返回OK。

setnx

语法: setnx key value

功能: setnx 是 set if not exists 的简写, 如果key不存在, 则 set 值, 存在则不设置值。

返回值: 设置成功, 返回1

设置失败, 返回0

mset

语法: mset key value [key value...]

功能: 同时设置一个或多个 key-value 对

返回值: 设置成功, 返回OK。

mget

语法: mget key [key ...]

功能: 获取所有(一个或多个)给定 key 的值

返回值: 包含所有key的列表, 如果key不存在, 则返回nil。

msetnx

语法: msetnx key value[key value...]

功能: 同时设置一个或多个 key-value 对, **如果有一个key是存在的, 则设置不成功。**

返回值: 设置成功, 返回1

设置失败, 返回0

redis的有关List类型的操作命令 单key 多有序value

Redis列表是简单的字符串列表, 按照插入顺序排序, 左边(头部)、右边(尾部)或者中间都可以添加元素。链表的操作无论是头或者尾效率都极高, 但是如果对中间元素进行操作, 那效率会大大降低了。

列表类型的数据操作总的思想是通过key和下标操作value, key是数据标识, 下标是数据在列表中的位置, value是我们感兴趣的业务数据。

每一个元素的下标有可是用复数表示, **负下标表示从表尾表示, 从-1开始。正下标表示从表头开始, 从0开始。**

lrange

语法: lrange key startIndex endIndex

功能: 获取列表 key 中指定下标区间内的元素, 下标从0开始, 到列表长度-1; 下标也可以是负数, 表示列表从后往前取, -1表示倒数第一个元素, -2表示倒数第二个元素, 以此类推; startIndex和endIndex超出范围不会报错。

返回值: 获取到的元素列表。

```
127.0.0.1:6379> lpush list01 1 2 3 4 5
(integer) 5
127.0.0.1:6379> lrange list01 2 5
1) "3"
2) "2"
3) "1"
127.0.0.1:6379> lrange list01 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379>
```

← 从左往右获取下标为2到5的元素。

← 从左往右获取下标为第一个到最后一个元素, -1表示最后一个元素。

lpush(插头)

语法: lpush key value [value...]

功能: 将一个或多个值 value 插入到列表 key 的最左边(**表头**), 各个value值依次插入到表头位置。

返回值: 插入之后的列表的长度。

```
127.0.0.1:6379> lpush list01 1 2 3
(integer) 3
127.0.0.1:6379> lpush list01 4 5
(integer) 5
127.0.0.1:6379> lrange list01 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379>
```

从左边插入列表值，返回插入之后列表的长度

rpush(插尾)

语法: rpush key value [value...]

功能: 将一个或多个值 value 插入到列表 key 的最右边（**表尾**），各个 value 值按依次插入到表尾。

返回值: 插入之后的列表的长度。

```
127.0.0.1:6379> rpush list02 1 2 3
(integer) 3
127.0.0.1:6379> rpush list02 4 5
(integer) 5
127.0.0.1:6379> lrange list02 0 -1
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
127.0.0.1:6379>
```

在右边加入列表值

lpop

语法: lpop key

功能: 移除并返回列表key头部第一个元素，即列表左侧的第一个元素。

返回值: 列表左侧第一个元素的值；列表key不存在，返回nil。

```
127.0.0.1:6379> lpush list01 v1 v2 v3 v4 v5
(integer) 5
127.0.0.1:6379> lrange list01 0 -1
1) "v5"
2) "v4"
3) "v3"
4) "v2"
5) "v1"
127.0.0.1:6379> lpop list01
"v5"
127.0.0.1:6379> lpop list01
"v4"
127.0.0.1:6379> lrange list01 0 -1
1) "v3"
2) "v2"
3) "v1"
127.0.0.1:6379> lpop listaa
(nil)
127.0.0.1:6379>
```

移除并返回左侧第一个元素

key不存在，返回nil

rpop

语法: rpop key

功能: 移除并返回列表key尾部第一个元素，即列表右侧的第一个元素。

返回值: 列表右侧第一个元素的值；列表key不存在，返回nil。

```

127.0.0.1:6379> lrange list01 0 -1
1) "v3"
2) "v2"
3) "v1"
127.0.0.1:6379> rpop list01
"v1"
127.0.0.1:6379> rpop list01
"v2"
127.0.0.1:6379> lrange list01 0 -1
1) "v3"
127.0.0.1:6379>

```

移除并返回右侧第一个元素。

语法: `lindex key index`

功能: 获取列表 `key` 中下标为指定 `index` 的元素, 列表元素不删除, 只是查询。0 表示列表的第一个元素, 1 表示列表的第二个元素; `index` 也可以负数的下标, -1 表示列表的最后一个元素, -2 表示列表的倒数第二个元素, 以此类推。

返回值: `key` 存在时, 返回指定元素的值; `key` 不存在时, 返回 `nil`。

llen

语法: `llen key`

功能: 获取列表 `key` 的长度

返回值: 数值, 列表的长度; `key` 不存在返回 0

```

127.0.0.1:6379> lrange list01 0 -1
1) "v5"
2) "v4"
3) "v3"
4) "v2"
5) "v1"
127.0.0.1:6379> llen list01
(integer) 5
127.0.0.1:6379> llen listaa
(integer) 0
127.0.0.1:6379>

```

获取列表的长度

列表不存在, 返回 0

lrem

语法: `lrem key count value`

功能: 根据参数 `count` 的值, 移除列表中与参数 `value` 相等的元素,

`count > 0`, 从列表的左侧向右开始移除;

`count < 0` 从列表的尾部开始移除;

`count = 0` 移除表中所有与 `value` 相等的值。

返回值: 数值, 移除的元素个数

```

127.0.0.1:6379> lpush list01 a a b c a d e a b b
(integer) 10
127.0.0.1:6379> lrem list01 2 a
(integer) 2
127.0.0.1:6379> lrange list01 0 -1
1) "b"
2) "b"
3) "e"
4) "d"
5) "c"
6) "b"
7) "a"
8) "a"
127.0.0.1:6379>

```

移除最左侧 2 个 a

ltrim

语法: ltrim key startIndex endIndex

功能: 截取key的指定下标区间的元素, 并且赋值给key。下标从0开始, 一直到列表长度-1; 下标也可以是负数, 表示列表从后往前取, -1表示倒数第一个元素, -2表示倒数第二个元素, 以此类推; startIndex和endIndex超出范围不会报错。

返回值: 执行成功返回ok

```
127.0.0.1:6379> lpush list02 1 2 3 4 5
(integer) 5
127.0.0.1:6379> ltrim list02 1 3
OK
127.0.0.1:6379> lrange list02 0 -1
1) "4"
2) "3"
3) "2"
```

截取列表list02下标1到3的元素

lset

语法: lset key index value

功能: 将列表 key 下标为 index 的元素的值设置为 value。

功能: 设置成功返回ok; key不存在或者index超出范围返回错误信息。

```
127.0.0.1:6379> lrange list01 0 -1
1) "b"
2) "b"
3) "e"
4) "d"
5) "c"
6) "b"
7) "a"
8) "a"
127.0.0.1:6379> lset list01 3 x
OK
127.0.0.1:6379> lrange list01 0 -1
1) "b"
2) "b"
3) "e"
4) "x"
5) "c"
6) "b"
7) "a"
8) "a"
127.0.0.1:6379>
```

把下标为3的元素设置为x

linsert

语法: linsert key before/after pivot value

功能: 将值 value 插入到列表 key 当中位于值 pivot 之前或之后的位置。key不存在或者pivot不在列表中, 不执行任何操作。

返回值: 命令执行成功, 返回新列表的长度。没有找到pivot返回 -1, key不存在返回0。

```
127.0.0.1:6379> lrange list01 0 -1
1) "b"
2) "b"
3) "e"
4) "x"
5) "c"
6) "b"
7) "a"
8) "a"
127.0.0.1:6379> linsert list01 before x beijing
(integer) 9
127.0.0.1:6379> linsert list01 after x shanghai
(integer) 10
127.0.0.1:6379> lrange list01 0 -1
1) "b"
2) "b"
3) "e"
4) "beijing"
5) "x"
6) "shanghai"
7) "c"
8) "b"
9) "a"
10) "a"
```

在list01列表x元素之前插入beijing元素
在list01列表x元素之后插入shanghai元素

redis的有关set类型的操作命令 单key 多无序value

Redis的Set是string类型的**无序不重复集合**。

集合类型的数据操作总的思想是通过key确定集合，key是集合标识，元素没有下标，只有直接操作业务数据和数据的个数。

sadd

语法：sadd key member [member...]

功能：将一个或多个 member 元素加入到集合 key 当中，已经存在于集合的 member 元素将被忽略，不会再加入。

返回值：加入到集合的新元素的个数(不包括被忽略的元素)。

```
127.0.0.1:6379> sadd set01 a a b b c c
(integer) 3
127.0.0.1:6379> smembers set01
1) "c"
2) "a"
3) "b"
127.0.0.1:6379> sadd set01 b
(integer) 0
127.0.0.1:6379> smembers set01
1) "c"
2) "a"
3) "b"
127.0.0.1:6379>
```

往集合set01里添加数据，重复的只加一次，返回添加的新元素个数。
添加元素时，如果集合中已经有了，则添加不上。

smembers

语法：smembers key

功能：获取集合 key 中的所有成员元素，不存在的key视为空集合。

返回值：返回指定集合的所有元素集合，不存在的key，返回空集合。

```
127.0.0.1:6379> sismember set01 b
(integer) 1
127.0.0.1:6379> sismember set01 x
(integer) 0
127.0.0.1:6379>
```

判断b是否为集合set01的元素，返回1表示是
判断x是否为集合set01的元素，返回0表示不是

sismember

语法：sismember key member

功能：判断 member 元素是否是集合 key 的元素

返回值：member是集合成员返回1，其他返回 0。

```
127.0.0.1:6379> sismember set01 b
(integer) 1
127.0.0.1:6379> sismember set01 x
(integer) 0
127.0.0.1:6379>
```

判断b是否为集合set01的元素，返回1表示是

判断x是否为集合set01的元素，返回0表示不是

scard

语法: scard key

功能: 获取集合里面的元素个数

返回值: 数字, key的元素个数。其他情况返回 0。

```
127.0.0.1:6379> scard set01
(integer) 3
127.0.0.1:6379> scard setaa
(integer) 0
127.0.0.1:6379>
```

获取集合set01的元素个数

key不存在, 返回0

srem

语法: srem key member [member...]

功能: 移除集合中一个或多个元素, 不存在的元素被忽略。

返回值: 数字, 成功移除的元素个数, 不包括被忽略的元素。

```
127.0.0.1:6379> smembers set01
1) "c"
2) "a"
3) "b"
127.0.0.1:6379> srem set01 a d e
(integer) 1
127.0.0.1:6379> smembers set01
1) "c"
2) "b"
127.0.0.1:6379>
```

移除集合中的多个元素, 不存在的忽略。

srandsmember

语法: srandsmember key[count]

功能: 只提供key, **随机返回**集合中一个元素, 元素不删除, 依然在集合中;

提供了count时, count 正数, 返回包含count个数元素的集合, 集合元素各不相同。count是负数, 返回一个count绝对值的长度的集合, 集合中元素可能会重复多次。

返回值: 一个元素或者多个元素的集合

spop

语法: spop key[count]

功能: **随机**从集合中删除一个或count个元素。

返回值: 被删除的元素, key不存在或空集合返回nil。


```

127.0.0.1:6379> smembers set01
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
7) "7"
8) "8"
127.0.0.1:6379> spop set01
"6"
127.0.0.1:6379> spop set01
"5"
127.0.0.1:6379> smembers set01
1) "1"
2) "2"
3) "3"
4) "4"
5) "7"
6) "8"
127.0.0.1:6379> spop set01 2
1) "4"
2) "1"
127.0.0.1:6379> smembers set01
1) "2"
2) "3"
3) "7"
4) "8"

```

← 随机移除1个元素

← 随机移除多个元素

smove

语法: smove src dest member

功能: 将 member 元素从src集合移动到dest集合, member不存在, smove不执行操作, 返回0, 如果dest存在member, 则仅从src中删除member。

返回值: 成功返回 1, 其他返回 0。

```

127.0.0.1:6379> sadd set01 a b c
(integer) 3
127.0.0.1:6379> sadd set02 x y z
(integer) 3
127.0.0.1:6379> smove set01 set02 b
(integer) 1
127.0.0.1:6379> smembers set01
1) "c"
2) "a"
127.0.0.1:6379> smembers set02
1) "y"
2) "x"
3) "b"
4) "z"
127.0.0.1:6379>

```

← 将集合set01中的b元素移到集合set02里

sdiff

语法: sdiff key key [key...]

功能: 返回指定集合的差集, 以第一个集合为准进行比较, 即第一个集合中有但在其它任何集合中都没有的元素组成的集合。

返回值: 返回第一个集合中有而后边集合中都没有的元素组成的集合, 如果第一个集合中的元素在后边集合中都有则返回空集合。

```
127.0.0.1:6379> sadd set01 a b
(integer) 2
127.0.0.1:6379> sadd set02 x y
(integer) 2
127.0.0.1:6379> sadd set03 m n
(integer) 2
127.0.0.1:6379> sdiff set01 set02
1) "a"
2) "b"
127.0.0.1:6379> sdiff set01 set02 set03
1) "a"
2) "b"
127.0.0.1:6379> sadd set03 a b
(integer) 2
127.0.0.1:6379> sdiff set01 set02 set03
(empty list or set)
```

返回集合set01和set02的差集

返回集合set01和set02、set03的差集

差集为空的情况

sinter

语法: `sinter key key [key...]`

功能: 返回指定集合的交集, 即指定的所有集合中都有的元素组成的集合。

返回值: 交集元素组成的集合, 如果没有则返回空集合。

```
127.0.0.1:6379> smembers set01
1) "a"
2) "b"
127.0.0.1:6379> smembers set02
1) "y"
2) "x"
127.0.0.1:6379> smembers set03
1) "n"
2) "a"
3) "m"
4) "b"
127.0.0.1:6379> sinter set01 set02
(empty list or set)
127.0.0.1:6379> sinter set01 set03
1) "a"
2) "b"
127.0.0.1:6379> sinter set01 set02 set03
(empty list or set)
```

返回集合set01和set02的交集

返回集合set01和set03的交集

返回集合set01、set02和set03的交集

sunion

语法: `sunion key key [key...]`

功能: 返回指定集合的并集, 即指定的所有集合元素组成的大集合, 如果元素有重复, 则保留一个。

返回值: 返回所有集合元素组成的大集合, 如果所有key都不存在, 返回空集合。

```
127.0.0.1:6379> smembers set01
1) "a"
2) "b"
127.0.0.1:6379> smembers set02
1) "y"
2) "x"
127.0.0.1:6379> smembers set03
1) "n"
2) "a"
3) "m"
4) "b"
127.0.0.1:6379> sunion set01 set02 set03
1) "m"
2) "n"
3) "a"
4) "y"
5) "x"
6) "b"
127.0.0.1:6379> sunion aa bb
(empty list or set)
127.0.0.1:6379>
```

返回集合set01、set02和set03的并集

所有的key都不存在, 返回空集合

redis的有关hash类型的操作命令 单key 对象

Redis的hash 是一个string类型的key和value的映射表，这里的value是一系列的键值对，hash特别适合用于存储对象。

哈希类型的数据操作总的思想是通过key和field操作value，key是数据标识，field是域，value是我们感兴趣的业务数据。

hset

语法: hset key field value [field value ...]

功能: 将键值对field-value设置到哈希列表key中，如果key不存在，则新建哈希列表，然后执行赋值，如果key下的field已经存在，则value值覆盖。

返回值: 返回设置成功的键值对个数。

```
127.0.0.1:6379> hset user id 1001
(integer) 1
127.0.0.1:6379> hset user name zhangsan age 20
(integer) 2
127.0.0.1:6379> hget user id
"1001"
127.0.0.1:6379> hget user name
"zhangsan"
127.0.0.1:6379> hget user age
"20"
127.0.0.1:6379>
```

设置哈希表的值，如果没有则新建哈希表

设置多个哈希表的值

hget

语法: hget key field

功能: 获取哈希表 key 中给定域 field 的值。

返回值: field域的值，如果key不存在或者field不存在返回nil。

hmset

语法: hmset key field value [field value...]

功能: 同时将多个 field-value (域-值)设置到哈希表 key 中，此命令会覆盖已经存在的field，hash表key不存在，创建空的hash表，再执行hmset。

返回值: 设置成功返回ok，如果失败返回一个错误。

```
127.0.0.1:6379> hmset user id 1001 name zhangsan age 20
OK
127.0.0.1:6379> hget user id
"1001"
127.0.0.1:6379> hget user name
"zhangsan"
127.0.0.1:6379> hget user age
"20"
```

批量往哈希表中设置键值对

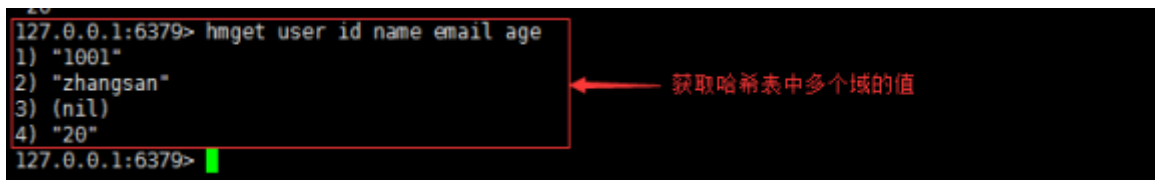
hmget

语法: hmget key field [field...]

功能: 获取哈希表 key 中一个或多个给定域的值

返回值: 返回和field顺序对应的值，如果field不存在，返回nil。

```
127.0.0.1:6379> hmget user id name email age
1) "1001"
2) "zhangsan"
3) (nil)
4) "20"
127.0.0.1:6379>
```

A terminal window showing the command 'hmget user id name email age' being executed. The output is a list of four items: '1) "1001"', '2) "zhangsan"', '3) (nil)', and '4) "20"'. A red box highlights the output, and a red arrow points to it with the text '获取哈希表中多个域的值'.

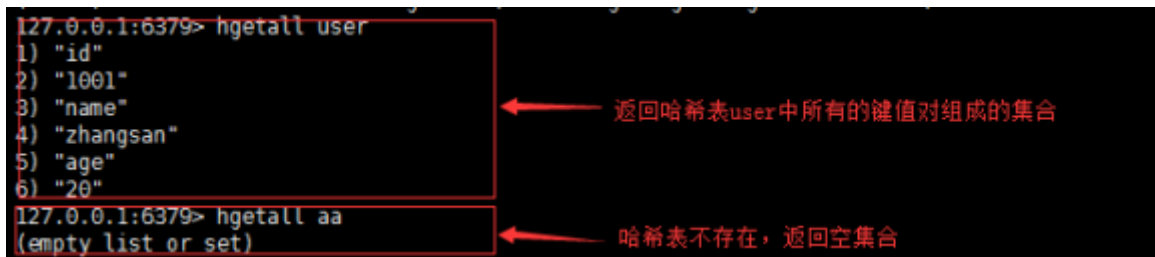
hgetall

语法: hgetall key

功能: 获取哈希表 key 中所有的域和值

返回值: 以列表形式返回hash中域和域的值, key不存在, 返回空hash.

```
127.0.0.1:6379> hgetall user
1) "id"
2) "1001"
3) "name"
4) "zhangsan"
5) "age"
6) "20"
127.0.0.1:6379> hgetall aa
(empty list or set)
127.0.0.1:6379>
```

A terminal window showing two commands. The first is 'hgetall user', which returns a list of six items: '1) "id"', '2) "1001"', '3) "name"', '4) "zhangsan"', '5) "age"', and '6) "20"'. A red box highlights the output, and a red arrow points to it with the text '返回哈希表user中所有的键值对组成的集合'. The second command is 'hgetall aa', which returns '(empty list or set)'. A red box highlights the output, and a red arrow points to it with the text '哈希表不存在, 返回空集合'.

hdel

语法: hdel key field [field...]

功能: 删除哈希表 key 中的一个或多个指定域field, 不存在field直接忽略。

返回值: 成功删除的field的数量。

```
127.0.0.1:6379> hdel user name age email
(integer) 2
127.0.0.1:6379> hget user id
"1001"
127.0.0.1:6379> hget user name
(nil)
127.0.0.1:6379>
```

A terminal window showing the command 'hdel user name age email' being executed. The output is '(integer) 2'. A red box highlights the output, and a red arrow points to it with the text '删除哈希表user中的name、age和email域'. The following commands 'hget user id' and 'hget user name' return '"1001"' and '(nil)' respectively.

hlen

语法: hlen key

功能: 获取哈希表 key 中域field的个数

返回值: 数值, field的个数。key不存在返回0.

```
127.0.0.1:6379> hlen user
(integer) 1
127.0.0.1:6379> hmset user name zhangsan age 20
OK
127.0.0.1:6379> hlen user
(integer) 3
127.0.0.1:6379>
```

A terminal window showing three commands. The first is 'hlen user', which returns '(integer) 1'. The second is 'hmset user name zhangsan age 20', which returns 'OK'. The third is 'hlen user', which returns '(integer) 3'. A red box highlights the output of the third command, and a red arrow points to it with the text '返回哈希表user中field的个数'.

hexists

语法: hexists key field

功能: 查看哈希表 key 中, 给定域 field 是否存在

返回值: 如果field存在, 返回1, 其他返回0.

```
(integer) 3
127.0.0.1:6379> hexists user id
(integer) 1
127.0.0.1:6379> hexists user email
(integer) 0
```

判断哈希表user中是否存在id和email的field

hkeys

语法: hkeys key

功能: 查看哈希表 key 中的所有field域列表

返回值: 包含所有field的列表, key不存在返回空列表

```
(error) wrong number of arguments for 'hkeys' command
127.0.0.1:6379> hkeys user
1) "id"
2) "name"
3) "age"
127.0.0.1:6379> hkeys aa
(empty list or set)
127.0.0.1:6379>
```

返回哈希表user所有的field列表

key不存在, 返回空列表

hvals

语法: hvals key

功能: 返回哈希表 中所有域的值列表

返回值: 包含哈希表所有域值的列表, key不存在返回空列表。

```
127.0.0.1:6379> hvals user
1) "1001"
2) "zhangsan"
3) "20"
127.0.0.1:6379> hvals aa
(empty list or set)
127.0.0.1:6379>
```

返回哈希表user的所有field值组成的列表

key不存在, 返回空列表

hincrby

语法: hincrby key field int

功能: 给哈希表key中的field域增加int

返回值: 返回增加之后的field域的值

```
(empty list or set)
127.0.0.1:6379> hget user age
"20"
127.0.0.1:6379> hincrby user age 2
(integer) 22
127.0.0.1:6379>
```

给哈希表user中age的值增加2

hincrbyfloat

语法: hincrbyfloat key field float

功能: 给哈希表key中的field域增加float

返回值: 返回增加之后的field域的值

```
127.0.0.1:6379> hset user score 90.5
(integer) 1
127.0.0.1:6379> hincrbyfloat user score 0.5
"91"
127.0.0.1:6379>
```

给哈希表user的age域值增加0.5

hsetnx

语法: hsetnx key field value

功能: 将哈希表 key 中的域 field 的值设置为 value , 当且仅当域 field 不存在的时候才设置, 否则不设置。

返回值: 设值成功返回1, 其他返回0.

```
127.0.0.1:6379> hgetall user
1) "id"
2) "1001"
3) "name"
4) "zhangsan"
5) "age"
6) "22"
7) "score"
8) "91"
127.0.0.1:6379> hsetnx user name lisi
(integer) 0
127.0.0.1:6379> hsetnx user email test@163.com
(integer) 1
127.0.0.1:6379> hgetall user
1) "id"
2) "1001"
3) "name"
4) "zhangsan"
5) "age"
6) "22"
7) "score"
8) "91"
9) "email"
10) "test@163.com"
127.0.0.1:6379>
```

往哈希表user中设置键值对, field已经存在, 设置不成功

往哈希表user中设置键值对, field不存在, 设置成功

redis的有关zset类型的操作命令 有序集合

Redis 有序集合zset和集合set一样也是string类型元素的集合, 且**不允许重复**的成员。

不同的是zset的**每个元素都会关联一个分数** (分数可以重复), redis通过分数来为集合中的成员进行从小到大的排序。

zadd

语法: zadd key score member [score member...]

功能: 将一个或多个 member 元素及其 score 值加入到有序集合 key 中, 如果member存在集合中, 则覆盖原来的值; **score可以是整数或浮点数**.

如果元素已经存在则把分数覆盖

返回值: 数字, 新添加的元素个数.

```
127.0.0.1:6379> zadd zset01 20 z1 30 z2 50 z3 40 z4
(integer) 4
127.0.0.1:6379> zrange zset01 0 -1
1) "z1"
2) "z2"
3) "z4"
4) "z3"
```

往有序集合zset01中添加多个元素及其分数

zrange

语法: `zrange key startIndex endIndex [WITHSCORES]`

功能: 查询有序集合, 指定区间的内的元素。集合成员按score值从小到大来排序; `startIndex`和`endIndex`都是从0开始表示第一个元素, 1表示第二个元素, 以此类推; `startIndex`和`endIndex`都可以取负数, 表示从后往前取, -1表示倒数第一个元素; **WITHSCORES选项让score和value一同返回。**

返回值: 指定区间的成员组成的集合。

```
127.0.0.1:6379> zadd zset01 20 z1 30 z2 50 z3 40 z4
(integer) 4
127.0.0.1:6379> zrange zset01 0 -1
1) "z1"
2) "z2"
3) "z4"
4) "z3"
127.0.0.1:6379> zrange zset01 0 -1 withscores
1) "z1"
2) "20"
3) "z2"
4) "30"
5) "z4"
6) "40"
7) "z3"
8) "50"
127.0.0.1:6379>
```

← 获取有序集合zset01所有的元素

← 获取有序集合zset01所有的元素连同分数

zrangbyscore

语法: `zrangbyscore key min max [WITHSCORES] [LIMIT offset count]`

功能: 获取有序集 `key` 中, 所有 `score` 值介于 `min` 和 `max` 之间 (包括`min`和`max`) 的成员, 有序成员是按递增 (从小到大) 排序;

使用符号"`(`" 表示包括`min`但不包括`max`;

`withscores` 显示`score`和 `value`;

`limit`用来限制返回结果的数量和区间, 在结果集中从第`offset`个开始, 取`count`个。

返回值: 指定区间的集合数据

```
127.0.0.1:6379> zrange zset01 0 -1 withscores
1) "z1"
2) "20"
3) "z2"
4) "30"
5) "z4"
6) "40"
7) "z3"
8) "50"
127.0.0.1:6379> zrangbyscore zset01 20 40
1) "z1"
2) "z2"
3) "z4"
127.0.0.1:6379> zrangbyscore zset01 20 40 withscores
1) "z1"
2) "20"
3) "z2"
4) "30"
5) "z4"
6) "40"
127.0.0.1:6379> zrangbyscore zset01 20 (40 withscores
1) "z1"
2) "20"
3) "z2"
4) "30"
127.0.0.1:6379>
```

← 获取分数在20-40之间的元素, 包括20和40

← 获取分数在20-40之间的元素, 包括20和40, 并显示分数

← 获取分数在20-40之间的元素, 包括20不包括40

```
127.0.0.1:6379> zrangebyscore zset01 20 40
1) "z1"
2) "z2"
3) "z4"
127.0.0.1:6379> zrangebyscore zset01 20 40 limit 2 2
1) "z4"
127.0.0.1:6379>
```

获取分数在20-40之间的元素，包括20和40，并且取其中的第2个开始的2个

zrem

语法: `zrem key member [member...]`

功能: 删除有序集合 `key` 中的一个或多个成员，不存在的成员被忽略。

返回值: 被成功删除的成员数量，不包括被忽略的成员。

```
127.0.0.1:6379> zrange zset01 0 -1 withscores
1) "z1"
2) "20"
3) "z2"
4) "30"
5) "z4"
6) "40"
7) "z3"
8) "50"
127.0.0.1:6379> zrem zset01 z2 z4
(integer) 2
127.0.0.1:6379> zrange zset01 0 -1 withscores
1) "z1"
2) "20"
3) "z3"
4) "50"
127.0.0.1:6379>
```

删除有序集合 `zset01` 中的元素 `z2` 和 `z4`

zcard

语法: `zcard key`

作用: 获取有序集 `key` 的元素成员的个数。

返回值: `key` 存在，返回集合元素的个数；`key` 不存在，返回0。

```
127.0.0.1:6379> zrange zset01 0 -1 withscores
1) "z1"
2) "20"
3) "z3"
4) "50"
127.0.0.1:6379> zcard zset01
(integer) 2
127.0.0.1:6379> zcard aa
(integer) 0
127.0.0.1:6379>
```

返回有序集合 `zset01` 元素个数

`key` 不存在，返回0

zcount

语法: `zcount key min max`

功能: 返回有序集 `key` 中，`score` 值在 `min` 和 `max` 之间(包括 `score` 值等于 `min` 或 `max`) 的成員的数量。

返回值: 指定有序集合中分数在指定区间内的元素数量。

```
127.0.0.1:6379> zadd zset01 10 z1 20 z2 50 z3 40 z4 60 z5
(integer) 5
127.0.0.1:6379> zcount zset01 20 50
(integer) 3
127.0.0.1:6379>
```

返回有序集合 `zset01` 中分数在20-50元素的个数，包括20和50

zrank

语法: zrank key member

功能: 获取有序集 key 中成员 member 的排名, 有序集成员按 score 值从小到大顺序排列, 从0开始排名, score最小的是0。

返回值: 指定元素在有序集合中的排名; 如果指定元素不存在, 返回nil。

```
127.0.0.1:6379> zrange zset01 0 -1 withscores
1) "z1"
2) "10"
3) "z2"
4) "20"
5) "z4"
6) "40"
7) "z3"
8) "50"
9) "z5"
10) "60"
127.0.0.1:6379> zrank zset01 z4
(integer) 2
```

← 获取有序集合zset01中元素z4的排名

zscore

语法: zscore key member

功能: 获取有序集合key中元素member的分数。

返回值: 返回指定有序集合元素的分数。

```
127.0.0.1:6379> zrange zset01 0 -1 withscores
1) "z1"
2) "10"
3) "z2"
4) "20"
5) "z4"
6) "40"
7) "z3"
8) "50"
9) "z5"
10) "60"
127.0.0.1:6379> zscore zset01 z4
"40"
127.0.0.1:6379> zscore zset01 z6
(nil)
```

← 获取指定有序集合元素的分数

← 元素不存在, 返回nil

zrevrank

语法: zrevrank key member

功能: 获取有序集 key 中成员 member 的排名, 有序集成员按 score 值从大到小顺序排列, 从0开始排名, score最大的是0。

返回值: 指定元素在有序集合中的排名; 如果指定元素不存在, 返回nil。

```
127.0.0.1:6379> zrange zset01 0 -1 withscores
1) "z1"
2) "10"
3) "z2"
4) "20"
5) "z4"
6) "40"
7) "z3"
8) "50"
9) "z5"
10) "60"
127.0.0.1:6379> zrevrank zset01 z5
(integer) 0
```

← 返回指定有序集合元素在分数倒序排的排名

zrevrange

语法: `zrevrange key startIndex endIndex [WITHSCORES]`

功能: 查询有序集合, 指定区间的内的元素。集合成员按score值从大到小来排序; startIndex和endIndex都是从0开始表示第一个元素, 1表示第二个元素, 以此类推; startIndex和endIndex都可以取负数, 表示从后往前取, -1表示倒数第一个元素; WITHSCORES选项让score和value一同返回。

返回值: 指定区间的成员组成的集合。

```
127.0.0.1:6379> zrange zset01 0 -1 withscores
1) "z1"
2) "10"
3) "z2"
4) "20"
5) "z4"
6) "40"
7) "z3"
8) "50"
9) "z5"
10) "60"
127.0.0.1:6379> zrevrange zset01 0 -1 withscores
1) "z5"
2) "60"
3) "z3"
4) "50"
5) "z4"
6) "40"
7) "z2"
8) "20"
9) "z1"
10) "10"
```

← 按照分数从大到小的顺序取有序集合的指定区间元素

zrevrangebyscore

语法: `zrevrangebyscore key max min [WITHSCORES] [LIMIT offset count]`

功能: 获取有序集 key 中, 所有 score 值介于 max 和 min 之间 (包括max和min) 的成员, 有序成员是按递减 (从大到小) 排序;

使用符号 "(" 表示不包括min和max;

withscores 显示score和 value;

limit用来限制返回结果的数量和区间, 在结果集中从第offset个开始, 取count个。

返回值: 指定区间的集合数据

```
127.0.0.1:6379> zrange zset01 0 -1 withscores
1) "z1"
2) "10"
3) "z2"
4) "20"
5) "z4"
6) "40"
7) "z3"
8) "50"
9) "z5"
10) "60"
127.0.0.1:6379> zrevrangebyscore zset01 50 20
1) "z3"
2) "z4"
3) "z2"
```

← 获取有序集合zset01中指定分数区间(倒序)的元素

六、Redis的配置文件

Redis.conf存放位置

Redis的安装根目录下(/opt/redis-5.0.2)，Redis在启动时会加载这个配置文件，在运行时按照配置进行工作。这个文件有时候我们会拿出来，单独存放在某一个位置，启动的时候必须明确指定使用哪个配置文件，此文件才会生效。

Redis的网络相关配置

bind：绑定ip地址，其他机器可以通过此ip访问redis,默认绑定127.0.0.1，也可以修改为本机的ip地址

port：配置redis占用的端口，默认是6379

tcp-keepalive：TCP连接保活策略，可以通过tcp-keepalive配置像来进行设置，单位为秒。假如设置为60秒，则server端会每60秒向连接空闲的客户端发起一次ACK请求，以检查客户端是否已经挂掉，对于无响应的客户端则会关闭其连接。如果设置为0，则不会进行保活检测。

如果配置了port和bind，则客户端连接redis服务时，必须指定端口和ip:

```
redis-cli -h 192.168.11.128 -p 6380
```

```
redis-cli -h 192.168.11.128 -p 6380 shutdown
```

Redis的常规配置

loglevel:日志级别，开发阶段可以设置成debug,生产阶段通常设置为notice或者warning.

logfile:指定日志文件。如果不指定，redis只进行标准输出。要保证日志文件所在的目录必须存在，文件可以不存在。还要再redis启动时指定所使用的配置文件，否则配置不起做作用

database:配置redis数据库的个数，默认是16个

Redis的安全配置

requirepass：设置访问redis服务时所使用的密码：默认不使用。此参数必须在protected-mode=yes时才起作用。一旦设置了密码验证，客户端连接redis服务时，必须使用密码连接：redis-cli -h ip -p port -a pwd

七、redis的持久化

redis是内存数据库，它把数据存储在内存在中，这样在加快读取速度的同时也对数据完全性产生了新的问题，即当redis所在服务器发生宕机后，redis数据库里所有数据都将会全部丢失。对此redis提供了可持久化功能-RDB和AOF。

redis提供持久化策略，在适当的时机采用适当手段把内存中的数据持久化到磁盘中，每次redis服务启动时都可以把磁盘上的数据再次加载内存中使用。

RDB策略

1. 在指定时间间隔内，redis服务执行指定次数的写操作，会自动触发一次持久化操作。
2. **save** ：配置持久化策略
3. **dbfilename**：配置redis RDB持久化文件所在目录
4. **dir**:配置redis RDB持久化数据保存的目录，默认是./

1.save ：配置复合的快照触发条件，即Redis 在seconds秒内key改变changes次，Redis把快照内的数据保存到磁盘中一次。默认的策略是：

1分钟内改变了1万次

或者5分钟内改变了10次

或者15分钟内改变了1次

如果要禁用Redis的持久化功能，则把所有的save配置都注释掉。

2、stop-writes-on-bgsave-error：当bgsave快照操作出错时停止写数据到磁盘，这样能保证内存数据和磁盘数据的一致性，但如果不在乎这种一致性，要在bgsave快照操作出错时继续写操作，这里需要配置为no。

3、rdbcompression：设置对于存储到磁盘中的快照是否进行压缩，设置为yes时，Redis会采用LZF算法进行压缩；如果不想消耗CPU进行压缩的话，可以设置为no，关闭此功能。

4、rdbchecksum：在存储快照以后，还可以让Redis使用CRC64算法来进行数据校验，但这样会消耗一定的性能，如果系统比较在意性能的提升，可以设置为no，关闭此功能。

5、dbfilename：Redis持久化数据生成的文件名，默认是dump.rdb，也可以自己配置。

6、dir：Redis持久化数据生成文件保存的目录，默认是./即redis的启动目录，也可以自己配置

AOF策略

采用操作日志来记录进行每次写操作，每次redis服务启动时，都会重新执行一遍操作日志中的指令。效率低下，redis默认不开启。

appendonly:配置是否开启AOF策略

appendfilename:配置操作日志文件

appendfsync：AOF异步持久化策略

always：同步持久化，每次发生数据变化会立刻写入到磁盘中。性能较差但数据完整性比较好（慢，安全）

everysec：出厂默认推荐，每秒异步记录一次（默认值）

no：不即时同步，由操作系统决定何时同步。

no-appendfsync-on-rewrite：重写时是否可以运用appendfsync，默认no，可以保证数据的安全性。

auto-aof-rewrite-percentage：设置重写的基准百分比

auto-aof-rewrite-min-size：设置重写的基准值

八、Redis的事务

事务：把一组数据库操作放在一起执行，来保证操作的原子性，要么同时成功，要么同时失败。

redis的事务：允许把一组redis命令放在一起执行，把命令进行序列化，保证部分原子性。

multi命令

标记一个事务的开始

语法：multi

功能：用于标记事务块的开始。**Redis会将后续的命令逐个放入队列中**，然后才能使用EXEC命令原子化地执行这个命令序列。

返回值：开启成功返回OK

exec命令

语法: exec

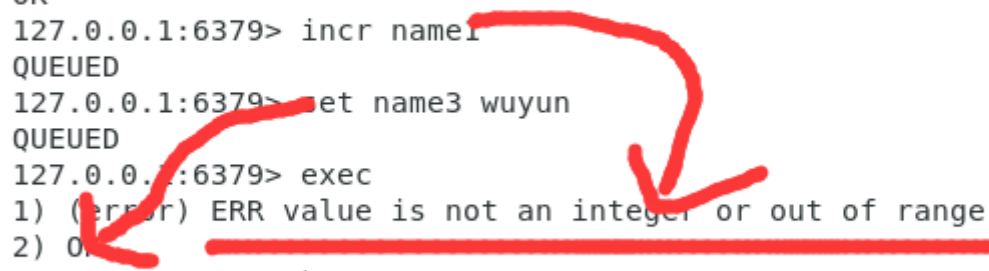
功能: 在一个事务中执行所有先前放入队列的命令, 然后恢复正常的连接状态。

部分原子性: 如果在把命令压入队列的过程中报错, 则整个队列中的命令**都不会执行**, 执行结果报错;

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> seta name3 yuwnzhu
(error) ERR unknown command 'seta', with args beginning with: 'name3', 'yuwnzhu',
127.0.0.1:6379> set name3 yuwnzhu
QUEUED
127.0.0.1:6379> seta name3 yuwnzhu
(error) ERR unknown command 'seta', with args beginning with: 'name3', 'yuwnzhu',
127.0.0.1:6379> set name4 yuwnzhu
QUEUED
127.0.0.1:6379> exec
(error) EXECABORT Transaction discarded because of previous errors.
127.0.0.1:6379>
```

如果在压队列的过程中正常, 在**执行队列中某一个命令报错**, 则只会影响本条命令的执行结果, 其它命令正常运行;

```
127.0.0.1:6379> exec
1) 1) "name1"
   2) "name2"
127.0.0.1:6379> multi
OK
127.0.0.1:6379> incr name1
QUEUED
127.0.0.1:6379> set name3 wuyun
QUEUED
127.0.0.1:6379> exec
1) (error) ERR value is not an integer or out of range
2) 0
127.0.0.1:6379> keys *
```



```
1) "name1"
2) "name3"
3) "name2"
```

当使用WATCH命令时, 只有当受监控的键没有被修改时, EXEC命令才会执行事务中的命令;而一旦执行了exec命令, 之前加的所有watch监控全部取消。

返回值: 这个命令的返回值是一个数组, 其中的每个元素分别是原子化事务中的每个命令的返回值。当使用WATCH命令时, 如果事务执行中止, 那么EXEC命令就会返回一个Null值。

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set name1 liuqiang
QUEUED
127.0.0.1:6379> set name2 yuwnzhu
QUEUED
127.0.0.1:6379> exec
1) OK
2) OK
127.0.0.1:6379> keys *
```



```
1) "name1"
2) "name2"
```

discard命令

语法: discard

功能: **清除所有先前**在一个事务中放入队列的命令, 并且**结束事务**。

如果使用了WATCH命令, 那么DISCARD命令就会将当前连接监控的所有键取消监控。

返回值: 清除成功, 返回OK。

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set name4 wuyun
QUEUED
127.0.0.1:6379> discard
OK
127.0.0.1:6379> exec
(error) ERR EXEC without MULTI
127.0.0.1:6379>
```

watch命令

语法: watch key [key ...]

功能: 当某个事务需要按条件执行时, 就要使用这个命令将给定的键设置为受监控的。如果被监控的key值在本事务外有修改时, 则本事务所有指令都不会被执行。Watch命令相当于关系型数据库中的乐观锁。

返回值: 监控成功, 返回OK。



The screenshot shows a Redis terminal session with the following commands and annotations:

- `127.0.0.1:6379> set balance 100` → 初始化数据
- `127.0.0.1:6379> set debt 0` → 初始化数据
- `127.0.0.1:6379> watch balance` → 监控balance
- `127.0.0.1:6379> multi` → 开启事务
- `127.0.0.1:6379> decrby balance 20` → 事务中的操作指令
- `127.0.0.1:6379> incrby debt 20` → 事务中的操作指令
- `127.0.0.1:6379> 3589:M 03 Dec 2018 17:15:38.820 * 5 changes in 120 seconds. Saving...` → 事务中的操作指令
- `B saved on disk` → 事务中的操作指令
- `5035:C 03 Dec 2018 17:15:38.832 * RDB: 4 MB of memory used by copy-on-write` → 事务中的操作指令
- `127.0.0.1:6379>` → 事务中的操作指令
- `127.0.0.1:6379> exec` → 执行事务。只要在事务中的操作命令过程中, 其它事务对balance进行了修改, 则此处的事务放弃执行。
- `127.0.0.1:6379> get balance` → 结果是其它事务对balance操作之后的结果
- `127.0.0.1:6379>` → 结果是其它事务对balance操作之后的结果

unwatch命令

语法: unwatch

功能: 清除所有先前为一个事务监控的键。

如果在watch命令之后你调用了EXEC或DISCARD命令, 那么就不需要手动调用UNWATCH命令。

返回值: 清除成功, 返回OK。

```
127.0.0.1:6379> mget balance debt
1) "100"
2) "0"
127.0.0.1:6379> watch balance debt
OK
127.0.0.1:6379> unwatch
OK
127.0.0.1:6379> watch debt
OK
127.0.0.1:6379> multi
(error) ERR unknown command 'multi', with args beginning with:
127.0.0.1:6379> multi
OK
127.0.0.1:6379> decrby balance 20
QUEUED
127.0.0.1:6379> incrby debt 20
QUEUED
127.0.0.1:6379> 6290:M 03 Dec 2018 18:28:42.038 * 5 changes in 300 seconds. Saving...
6290:M 03 Dec 2018 18:
B saved on disk
6850:C 03 Dec 2018 18:28:42.050 * RDB: 4 MB of memory used by copy-on-write
6290:M 03 Dec 201
127.0.0.1:6379>
127.0.0.1:6379> exec
1) (integer) 100
2) (integer) 20
127.0.0.1:6379>
```

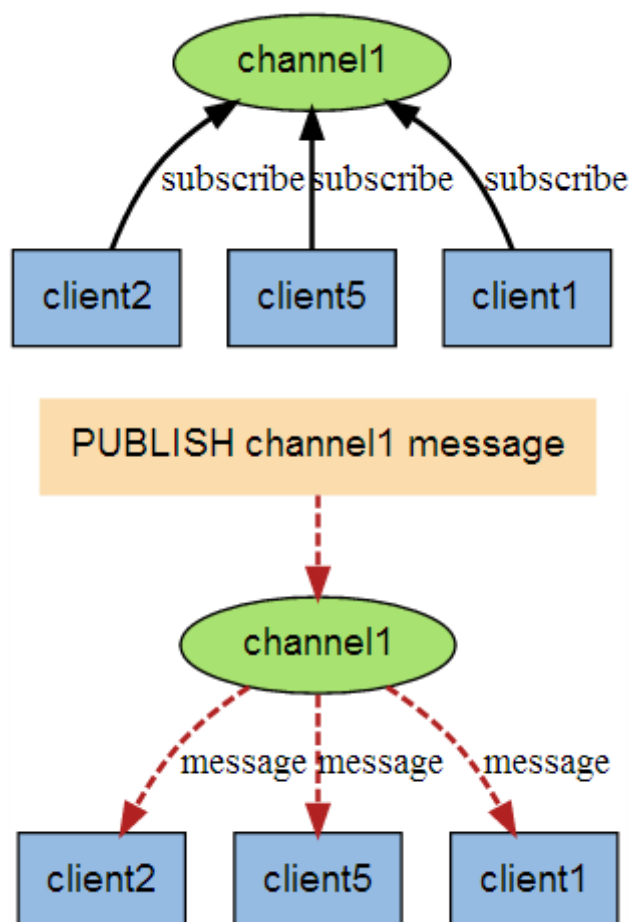
准备数据
监控balance和debt
取消监控所有key
只监控debt
开启事务
事务内指令
即使其它客户端命令修改了balance，本事务照常执行。因为本事务取消了对balance的监控

九、redis消息的发布与订阅

redis客户端订阅频道，消息的发布者往频道上发布消息，所有订阅此频道的客户端都能过接受到。

Redis发布订阅

Redis 发布订阅(pub/sub)是一种消息通信模式：发送者(pub)发送消息，订阅者(sub)接收消息。Redis 客户端可以订阅任意数量的频道。



subscribe

语法: subscribe channel [channel...]

功能: 订阅一个或多个频道的信息

返回值: 订阅的消息

```
127.0.0.1:6379> subscribe ch1 ch2 ch3
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "ch1"
3) (integer) 1
1) "subscribe"
2) "ch2"
3) (integer) 2
1) "subscribe"
2) "ch3"
3) (integer) 3
```

publish

语法: publish channel message

功能: 将信息发送到指定的频道。

返回值: 数字。接收到消息订阅者的数量。

```
127.0.0.1:6379> publish ch1 hello
(integer) 1
127.0.0.1:6379> subscribe ch1 ch2 ch3
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "ch1"
3) (integer) 1
1) "subscribe"
2) "ch2"
3) (integer) 2
1) "subscribe"
2) "ch3"
3) (integer) 3
1) "message"
2) "ch1"
3) "hello"
```

psubscribe

语法: psubscribe pattern [pattern]

功能: 订阅一个或多个符合给定模式的频道。模式以 * 作为通配符, 例如: news.* 匹配所有以 news. 开头的频道。

返回值: 订阅的信息。

十、redis的主从复制

主从复制

主机数据更新后根据配置和策略，自动同步到从机的master/slave机制，master以写为主，Slave以读为主。

主少从多、主学从读、读写分离、主写同步复制到从

一主二从

原理

1. 配从库不配主库
2. 配从库：slaveof 主库ip 主库端口
3. 主写从读、读写分离
4. 从连前后同
5. 主断从待命、从断重新连

搭建

使用一个redis模拟三台redis服务

1.修改三份redis.conf:

rendis6379.conf:

1. 端口号

```
# Accept connections on the specified port, default is 6379
# If port 0 is specified Redis will not listen on a TCP port
port 6379
```

2. pid文件和日志文件

```
# Creating a pid file is best effort: if Redis is not able
# nothing bad happens, the server will start and run normal
pidfile /var/run/redis_6379.pid

# Specify the server verbosity level.
# This can be one of:
# debug (a lot of information, useful for development/testing)
# verbose (many rarely useful info, but not a mess like the
# notice (moderately verbose, what you want in production)
# warning (only very important / critical messages are logged)
loglevel notice

# Specify the log file name. Also the empty string can be used
# Redis to log on the standard output. Note that if you use
# output for logging but daemonize, logs will be sent to /dev/null
logfile "6379.log"
```

3. 持久化文件

```
# The filename where to dump the DB
dbfilename dump6379.rdb
```

4. ip地址

主机恢复，一切正常

```
redis-cli -h 127.0.0.1 -p 6379//客户端连接
```

但是数据已经复制过来了

```
redis-server redis6379.conf
```


2. 原来的主机变成从机

```
slaveof 127.0.0.1 6381
```

3. 6381既是主机也是从机。

小结：一台主机配置多台从机，一台从机又可以配置多台从机，从而形成一个庞大的集群结构。减轻了一台主机的压力，但是也增加了服务间的延迟时间。

十一、redis哨兵模式：主机宕机、从机上位的自动版

从机上位的自动版。Redis提供了哨兵的命令，哨兵命令是一个独立的进程，哨兵通过发送命令，来监控主从服务器的运行状态，如果检测到master故障了根据投票数自动将某一个slave转换master，然后通过消息订阅模式通知其它slave，让它们切换主机。然而，一个哨兵进程对Redis服务器进行监控，可能会出现问题，为此，我们可以使用多哨兵进行监控。

1.搭建一主二从集群架构（上节）

2.提供哨兵配置文件

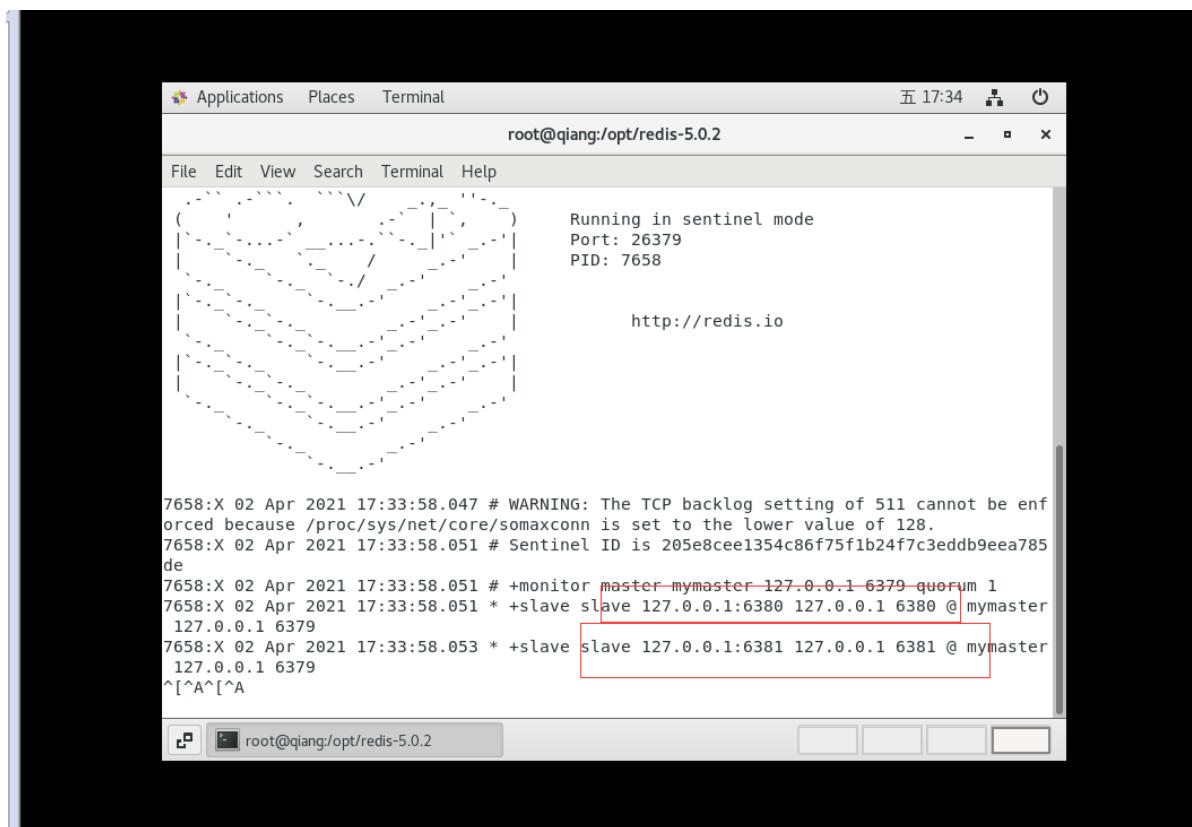
在redis安装目录下创建配置文件：redis_sentinel.conf,并编辑里边的内容

```
# The valid charset is A-Z 0-9 and the three characters . - _ .  
sentinel monitor mymaster 127.0.0.1 6379 1  
# sentinel auth-pass <master-name> <password>
```

1表示：指定监控主机的ip地址，port端口，得到哨兵的投票数(当哨兵投票数大于或者等于此数时切换主从关系)。

3.启动哨兵服务

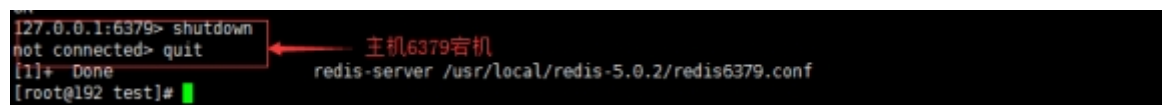
```
redis_sentinel redis_sentinel.conf
```



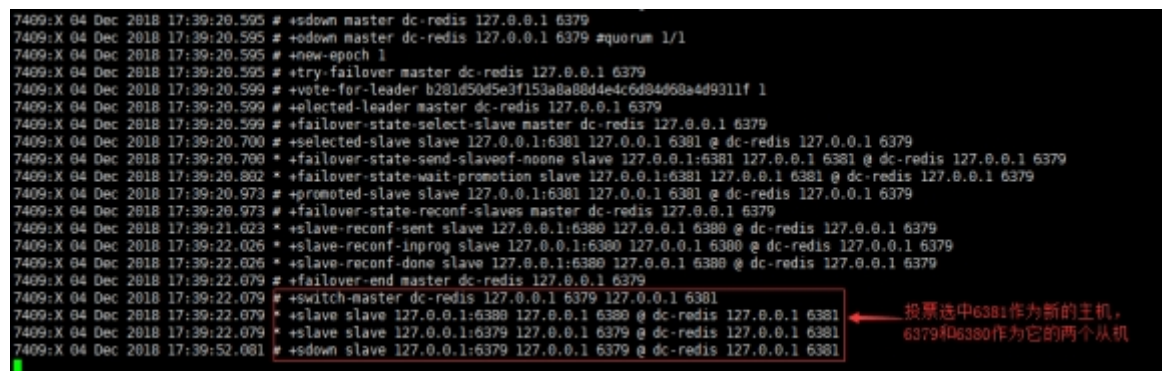
可以看到两个从机

4.关闭主机6379，哨兵程序自动选择从机上位

主机宕机：



等待从机投票，在sentinel窗口中查看打印信息。



查看6380和6381的redis信息：



```
127.0.0.1:6381> info replication
# Replication
role:master
connected_slaves:1
```

6381已经变为主机，自动从机上位

原主机恢复，启动6379：

```
127.0.0.1:6379> info replication
# Replication
role:slave
master_host:127.0.0.1
master_port:6381
```

原来的主机6379变为6381的从机，注意这个过程可能稍有延迟

```
127.0.0.1:6381> info replication
# Replication
role:master
connected_slaves:2
```

6381已经有两个从机

十二、jedis操作redis

使用Redis官方推荐的Jedis，在java应用中操作Redis。Jedis几乎涵盖了Redis的所有命令。操作Redis的命令在Jedis中以方法的形式出现。

1. 假如依赖

```
<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
  <version>3.0.1</version>
</dependency>
```

2. 连接redis

```
public class JedisKeyTest {
    public static void main(String[] args) {
        //连接redis
        Jedis jedis = new Jedis("222.204.55.121",6379);
        //使用jedis对象操作redis服务
        String ret = jedis.ping();
        System.out.println(ret);
        System.out.println(jedis.get("username"));
        System.out.println(jedis.exists("username"));
        System.out.println(jedis.move("k1",1));
        //开启事务
        Transaction transaction = jedis.multi();
        transaction.set("k1","v2");
        transaction.set("k2","v2");
        transaction.exec();
    }
}
```

