**redis学习笔记**

官网：http://redis.io

redis定义：是开源、BSD许可、高级的key-value存储系统。可以用来存储字符串、哈希结构、链表、集合，因此常用来提供数据结构服务。

redis和memcached比较：

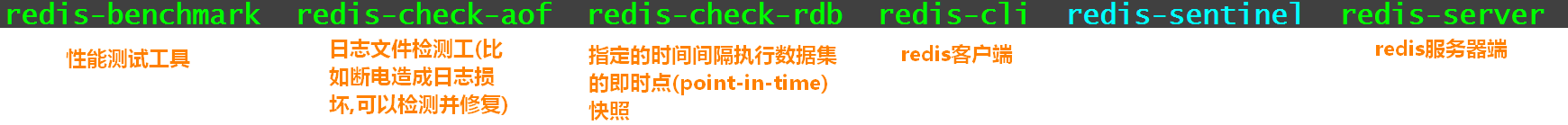
(1) redis可以用来做存储(storge)，而memcached用来做缓存(cache)

(2) redis存储的数据结构有字符串、哈希结构、链表、集合，而memcached只有一种字符串

# 1 redis安装

|  |  |
| --- | --- |
| 下载redis源代码 | https://github.com/antirez/redis/archive/5.0.2.tar.gz |
| 解压 | tar -zxvf 5.0.2.tar |
| 进入目录 | cd redis-5.0.2 |
| 编译 | make  注：官方已经配置好，只需要编译即可，需要注意系统时间必须在redis发布时间之后，否则编译失败 |
| 安装指定目录 | make install PREFIX=/usr/local/redis install |
| 复制配置文件 | cp redis.conf /usr/local/redis/ |

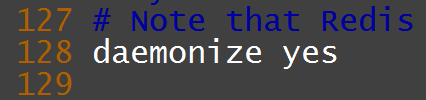
安装后在bin目录下的个文件说明如下图：



(1) 修改配置文件，使redis启动后在后台运行。

**vim /usr/local/redis/redis.conf**

把daemonize属性改为yes，如下图所示



如果需要密码验证，在配置文件中查找requirepass字段，例如requirepass 123456

(2) 设置开机启动，把redis添加到service服务中

在自启动/etc/init.d/文件夹下新建启动脚本

**vim /etc/init.d/redis**

脚本内容如下:

|  |
| --- |
| #!/bin/sh  # chkconfig: 2345 90 10  # description: Redis is a persistent key-value database    PATH=/usr/local/bin:/sbin:/usr/bin:/bin    REDISPORT=6379  EXEC=/usr/local/redis/bin/redis-server  REDIS\_CLI=/usr/local/redis/bin/redis-cli    PIDFILE=/var/run/redis\_6379.pid  CONF="/usr/local/redis/redis.conf"    case "$1" in  start)  if [ -f $PIDFILE ]  then  echo "$PIDFILE exists, process is already running or crashed"  else  echo "Starting Redis server..."  $EXEC $CONF  fi  if [ "$?"="0" ]  then  echo "Redis is running..."  fi  ;;  stop)  if [ ! -f $PIDFILE ]  then  echo "$PIDFILE does not exist, process is not running"  else  PID=$(cat $PIDFILE)  echo "Stopping ..."  $REDIS\_CLI -p $REDISPORT SHUTDOWN  while [ -x ${PIDFILE} ]  do  echo "Waiting for Redis to shutdown ..."  sleep 1  done  echo "Redis stopped"  fi  ;;  restart|force-reload)  ${0} stop  ${0} start  ;;  \*)  echo "Usage: /etc/init.d/redis {start|stop|restart|force-reload}" >&2  exit 1  esac  exit 0 |

设置为可执行权限

**chmod +x /etc/init.d/redis**

添加服务到service

**chkconfig --add /etc/init.d/redis**

以后就可以使用开启和停止服务

启动服务：service redis start

停止服务：service redis stop

(3) 将redis的bin命令添加到PATH

可以在/etc/profile或~/.bashrc(只在当前用户有效)下添加，一般在全局添加

**vim /etc/profile**

在文件末尾添加内容如下：

|  |
| --- |
| export PATH=$PATH:/usr/local/redis/bin |

使profile生效

**source /etc/profile**

启动服务：service redis start

停止无密码redis服务: redis-cli shutdown

停止有密码redis服务: redis-cli -a 123456 shutdown

启动客户端：redis-cli

# 2 redis操作命令

## 2.1 key操作命令

### set 设置kv

|  |  |
| --- | --- |
| **set key value**  设置key-value | 127.0.0.1:6379> set name zhangsan  OK  127.0.0.1:6379> set age 25  OK |

### keys 查询

|  |  |
| --- | --- |
| **keys pattern**  查询相应的key  pattern可以精确也可以模糊匹配，有3个通配符 \*, ? ,[]  \*: 通配任意多个字符  ?: 通配单个字符  []: 通配括号内的某1个字符 | 127.0.0.1:6379> keys \*  1) "age"  2) "name"  3) "gender"  127.0.0.1:6379> keys \*e  1) "age"  2) "name"  127.0.0.1:6379> keys a?e  1) "age"  127.0.0.1:6379> keys nam[aeu]  1) "name" |

### randomkey 随机获取key名

|  |  |
| --- | --- |
| **randomkey**  随机获取key名 | 127.0.0.1:6379> randomkey  "age"  127.0.0.1:6379> randomkey  "age"  127.0.0.1:6379> randomkey  "gender"  127.0.0.1:6379> randomkey  "name"  127.0.0.1:6379> randomkey |

### type 获取key类型

|  |  |
| --- | --- |
| **type key**  获取key存储的值的类型 | 127.0.0.1:6379> type name  string  127.0.0.1:6379> type age  string |

### exists 判断key是否存在

|  |  |
| --- | --- |
| **exists key**  判断key是否存在，存在返回1，不存在返回0 | 127.0.0.1:6379> exists name  (integer) 1  127.0.0.1:6379> exists title  (integer) 0 |

### del 删除key

|  |  |
| --- | --- |
| **del key**  删除key | 127.0.0.1:6379> exists gender  (integer) 1  127.0.0.1:6379> del gender  (integer) 1  127.0.0.1:6379> exists gender |

### rename 重命名key

|  |  |
| --- | --- |
| **rename key newkey**  重命名key，如果newkey已经存在，修改后则替换新key的值 | 127.0.0.1:6379> set title "redis test"  OK  127.0.0.1:6379> exists title  (integer) 1  127.0.0.1:6379> rename title biaoti  OK  127.0.0.1:6379> get biaoti  "redis test" |

### renamenx 重命名不存在的key

|  |  |
| --- | --- |
| **renamenx key newkey**  重命名key，如果newkey已经存在则不修改。  nx表示not exists | 127.0.0.1:6379> keys \*  1) "biaoti"  2) "age"  3) "name"  127.0.0.1:6379> renamenx biaoti name  (integer) 0 |

### move 把key移动到其他库

|  |  |
| --- | --- |
| **move key db**  把key移动到另一个数据库，db为整数 | 127.0.0.1:6379> keys \*  1) "biaoti"  2) "age"  3) "name"  127.0.0.1:6379> move biaoti 1  (integer) 1  127.0.0.1:6379> select 1  OK  127.0.0.1:6379[1]> keys \*  1) "biaoti" |

### expire 设置key有效期

|  |  |
| --- | --- |
| **expire key**  设置key的生命周期  pexpire key表示以毫秒为单位设置声明周期 | 127.0.0.1:6379[1]> set login on  OK  127.0.0.1:6379[1]> expire login 60  (integer) 1  127.0.0.1:6379[1]> ttl login  (integer) 47 |

### ttl 查询key有效期

|  |  |
| --- | --- |
| **ttl key**  查询key的生命周期  大于0 ：生命周期单位为秒，  等于-1：永久有效  等于-2：该key不存在  pttl key表示毫秒为单位 | 127.0.0.1:6379> ttl name  (integer) -1  127.0.0.1:6379> ttl title  (integer) -2 |

### persist 设置key永久有效

|  |  |
| --- | --- |
| **persist key**  设置key永久有效 | 127.0.0.1:6379> set login on  OK  127.0.0.1:6379> expire login 60  (integer) 1  127.0.0.1:6379> ttl login  (integer) 55  127.0.0.1:6379> persist login  (integer) 1  127.0.0.1:6379> ttl login  (integer) -1 |

## 2.2 库操作命令

### dbsize 查看当前有多少个key

|  |  |
| --- | --- |
| **dbsize**  查看当前有多少个key | 127.0.0.1:6379> dbsize  12 |

### select 选择库

|  |  |
| --- | --- |
| **select db**  选择使用哪个数据库，db为整数  默认有16个数据库0~15，如果想修改数据库数量，修改redis.conf配置文件的databases值 | 127.0.0.1:6379> select 1  OK  127.0.0.1:6379[1]> select 2  OK  127.0.0.1:6379[2]> select 15  OK |

### flushdb 删除库中所有key

|  |  |
| --- | --- |
| **flushdb**  删除当前选择数据库中的所有key | 127.0.0.1:6379[1]> keys \*  1) "biaoti"  127.0.0.1:6379[1]> flushdb  OK  127.0.0.1:6379[1]> keys \*  (empty list or set) |

### flushall 删除所有库的key

|  |  |
| --- | --- |
| **flushall**  删除所有数据库中的key | 127.0.0.1:6379[1]> flushall  OK  127.0.0.1:6379[1]> select 0  OK  127.0.0.1:6379> keys \*  (empty list or set) |

# 3 字符串类型操作

## set 设置kv、效期、判断key是否存在

|  |  |
| --- | --- |
| **set key value [ex 秒数]|[px 毫秒数] [nx]|[xx]**  设置kv时也可以设置有效期和判断key是否存在  ex和px不要同时写，否则以后面有效期为准  nx表示key不存在时执行操作  xx表示key存在时执行操作 | 127.0.0.1:6379> set name zhangsan  OK  127.0.0.1:6379> set name zhangsan ex 100  OK  127.0.0.1:6379> ttl name  (integer) 78  127.0.0.1:6379> set name lisi nx  (nil)  127.0.0.1:6379> get name  "zhangsan" |

## mset 一次性输入多个kv

|  |  |
| --- | --- |
| **mset key1 value1 key2 value2......**  一次性输入多个key-value | 127.0.0.1:6379> mset x 1 y 2 z 3  OK  127.0.0.1:6379> keys \*  1) "y"  2) "z"  3) "x" |

## get 获取key的值

|  |  |
| --- | --- |
| **get key**  获取key的值 | 127.0.0.1:6379> get x  "1"  127.0.0.1:6379> get y  "2" |

## mget 一次性获取多个key的值

|  |  |
| --- | --- |
| **mget key1 key2 ......**  一次性获取多个key的值 | 127.0.0.1:6379> mget x y z  1) "1"  2) "2"  3) "3" |

## setrange 修改偏移字节值为value

|  |  |
| --- | --- |
| **setrange key offset value**  把字符串的偏移字节改为value  如果偏移量大于字符长度，中间字符自动补0x00 | 127.0.0.1:6379> set name zhangsan  OK  127.0.0.1:6379> setrange name 5 \*\*\*  (integer) 8  127.0.0.1:6379> get name  "zhang\*\*\*" |

## append 在key的值后面追加字符串

|  |  |
| --- | --- |
| **append key value**  在key的值后面追加value字符串 | 127.0.0.1:6379> set name zhangsan  OK  127.0.0.1:6379> append name "@126.com"  (integer) 16  127.0.0.1:6379> get name  "zhangsan@126.com" |

## getrange 获取key值的部分内容

|  |  |
| --- | --- |
| **getrange key start stop**  获取key值的一部分内容  start表示起始位置  stop表示结束位置，可以为为负数，表示从最后数起  start>length 空字符串  stop>length 截取到结尾 | 127.0.0.1:6379> set title "hello world"  OK  127.0.0.1:6379> getrange title 6 11  "world"  127.0.0.1:6379> getrange title 0 -7  "hello" |

## getset 设置新值返回旧值

|  |  |
| --- | --- |
| **getset key newvalue**  设置新值，并返回旧值 | 127.0.0.1:6379> set login on  OK  127.0.0.1:6379> get login  "on"  127.0.0.1:6379> getset login off  "on"  127.0.0.1:6379> get login  "off" |

## incr/decr 指定key的值加/减1

|  |  |
| --- | --- |
| **incr/decr key**  指定key的值加/减1，返回结果  key不存在时，自动创建并加减1  key的值为字符串时无效 | 127.0.0.1:6379> set num 100  OK  127.0.0.1:6379> incr num  (integer) 101  127.0.0.1:6379> decr num  (integer) 100 |

## incrby/decrby 指定key的值加/减number

|  |  |
| --- | --- |
| **incrby/decrby key number**  指定key的值加减number大小 | 127.0.0.1:6379> set num 100  OK  127.0.0.1:6379> incrby num 50  (integer) 150  127.0.0.1:6379> decrby num 100  (integer) 50 |

## incrbyfloat 指定key的值加浮点数

|  |  |
| --- | --- |
| **incrbyfloat key floatnumber**  指定key的值加浮点数 | 127.0.0.1:6379> set num 10  OK  127.0.0.1:6379> incrbyfloat num 0.5  "10.5"  127.0.0.1:6379> incrbyfloat num -1.5  "9" |

## setbit 设置二进制位上的值

|  |  |
| --- | --- |
| **setbit key offset value**  设置offset对应二进制位上的值  返回改位的旧值  如果offset过大则会在中间填充0  offset最大为，即512M | 127.0.0.1:6379> set letter A  OK  127.0.0.1:6379> setbit letter 2 1  (integer) 0  127.0.0.1:6379> get letter  "a"  把0100 0001(65)改为0110 0001(97)即把大写A改为了小写a |

## getbit 获取二进制位上的值

|  |  |
| --- | --- |
| **getbit key offset**  获取二进制offset对应位的值 | 127.0.0.1:6379> set letter A  OK  127.0.0.1:6379> getbit letter 0  (integer) 0  127.0.0.1:6379> getbit letter 1  (integer) 1  127.0.0.1:6379> getbit letter 7  (integer) 1 |

## bitop 对多个key做指定操作

|  |  |
| --- | --- |
| **bitop operation destkey key1 [key2 ......]**  对key1 key2 keyN做operation，并把结果保存到destkey 上  operation有AND、OR、NOT、XOR | 127.0.0.1:6379> setbit lower 2 1  (integer) 0  127.0.0.1:6379> setbit upper 2 0  (integer) 0  127.0.0.1:6379> set letter A  OK  127.0.0.1:6379> bitop or letter letter lower  (integer) 1  127.0.0.1:6379> get letter  "a" |

# 4 实际应用：统计活跃用户

**场景**

1亿个用户，每个用户登陆做任意操作，标记记为今天活跃，否则记为不活跃。

每周评出，有奖活跃用户，连续7天活动，每月评，等等...

**思路**

每个用户在数据库都有一个id，用第id个位的0和1来表示是否该用户登录，例如

login0721: '011001...............0'

......

login0726: '011001...............0'

login0727: '0110000.............1'

**实现过程**

(1)记录用户登陆，每天按日期生成一个位图，用户登陆后，把user\_id位上的bit值置为1

// 首先把所有用户的位置位0

redis 127.0.0.1:6379> setbit login0721 100000000 0

(integer) 0

redis 127.0.0.1:6379> setbit login0721 3 1

(integer) 0

redis 127.0.0.1:6379> setbit login0721 5 1

(integer) 0

redis 127.0.0.1:6379> setbit login0721 7 1

(integer) 0

......

redis 127.0.0.1:6379> setbit login0722 100000000 0

(integer) 0

redis 127.0.0.1:6379> setbit login0722 3 1

(integer) 0

redis 127.0.0.1:6379> setbit login0722 5 1

(integer) 0

redis 127.0.0.1:6379> setbit login0722 8 1

(integer) 0

......

redis 127.0.0.1:6379> setbit login0723 100000000 0

(integer) 0

redis 127.0.0.1:6379> setbit login0723 3 1

(integer) 0

redis 127.0.0.1:6379> setbit login0723 4 1

(integer) 0

redis 127.0.0.1:6379> setbit login0723 6 1

(integer) 0

......

(2)把1周/月的位图用and计算, 位为1的是连续登陆的用户。

redis 127.0.0.1:6379> bitop and login0721 login0722 login0723......

(integer) 12500001

**优点**

(1) 节约空间，用1亿bit表示1亿人每天的登陆情况，1亿bit约为10M。

(2) 计算方便，计算速度非常快

# 5 链表类型操作

## lpush/rpush 在链表头/尾增加一个成员

|  |  |
| --- | --- |
| **lpush/rpush key value**  在链表头/尾增加一个成员，返回链表成员的个数 | 127.0.0.1:6379> lpush letters A  (integer) 1  127.0.0.1:6379> rpush letters B  (integer) 2  127.0.0.1:6379> rpush letters C  (integer) 3  127.0.0.1:6379> rpush letters D  (integer) 4 |

## lrange 获取链表成员

|  |  |
| --- | --- |
| **lrange key start stop**  返回链表中[start,stop]范围的成员  规律: 左数从0开始,右数从-1开始 | 127.0.0.1:6379> lrange letters 0 -1  1) "A"  2) "B"  3) "C"  4) "D"  127.0.0.1:6379> lrange letters 1 2  1) "B"  2) "C" |

## lpop/rpop 弹出链表中头/尾的成员

|  |  |
| --- | --- |
| **lpop/rpop key**  弹出链表中头/尾的成员 | 127.0.0.1:6379> lrange letters 0 -1  1) "A"  2) "B"  3) "C"  4) "D"  127.0.0.1:6379> lpop letters  "A"  127.0.0.1:6379> rpop letters  "D"  127.0.0.1:6379> lrange letters 0 -1  1) "B"  2) "C" |

## lrem 删除链表成员

|  |  |
| --- | --- |
| **lrem key count value**  从key链表中删除 value值  删除count的绝对值个value后结束  count>0 从表头删除  count<0 从表尾删除 | 127.0.0.1:6379> rpush letters A B C D A B C D A B C D  (integer) 12  127.0.0.1:6379> lrem letters 2 A  (integer) 2  127.0.0.1:6379> lrange letters 0 -1  1) "B"  2) "C"  3) "D"  4) "B"  5) "C"  6) "D"  7) "A"  8) "B"  9) "C"  10) "D"  127.0.0.1:6379> lrem letters -3 D  (integer) 3  127.0.0.1:6379> lrange letters 0 -1  1) "B"  2) "C"  3) "B"  4) "C"  5) "A"  6) "B"  7) "C" |

## ltrim 截取链表片段

|  |  |
| --- | --- |
| **ltrim key start stop**  截取key在 [start,stop]范围一段链表，并把该段重新赋给key  stop >0 从表头开始计算  stop <0 从表尾开始计算 | 127.0.0.1:6379> rpush letters A B C D E F  (integer) 6  127.0.0.1:6379> ltrim letters 1 4  OK  127.0.0.1:6379> lrange letters 0 -1  1) "B"  2) "C"  3) "D"  4) "E"  127.0.0.1:6379> ltrim letters 1 -2  OK  127.0.0.1:6379> lrange letters 0 -1  1) "C"  2) "D" |

## lindex 获取链表索引对应的值

|  |  |
| --- | --- |
| **lindex key index**  获取链表索引index对应的值 | 127.0.0.1:6379> rpush letters A B C D  (integer) 4  127.0.0.1:6379> lindex letters 1  "B"  127.0.0.1:6379> lindex letters 2  "C" |

## llen key 获取链表成员个数

|  |  |
| --- | --- |
| **llen key**  获取链表成员个数 | 127.0.0.1:6379> rpush letters A B C D  (integer) 4  127.0.0.1:6379> llen letters  (integer) 4 |

## linsert 在链表中指定位置插入成员

|  |  |
| --- | --- |
| **linsert key after|before search value**  在key链表中寻找"search"，并在search值之前|之后插入value  如果没有找到，不插入值  如果找到一个search后，命令就结束了，因此不会插入多个value | 127.0.0.1:6379> rpush id 1 3 5 7  (integer) 4  127.0.0.1:6379> linsert id before 3 2  (integer) 5  127.0.0.1:6379> lrange id 0 -1  1) "1"  2) "2"  3) "3"  4) "5"  5) "7"  127.0.0.1:6379> linsert id after 5 6  (integer) 6  127.0.0.1:6379> lrange id 0 -1  1) "1"  2) "2"  3) "3"  4) "5"  5) "6"  6) "7" |

## rpoplpush 把一个链表成员移动到另一个链表中

|  |  |
| --- | --- |
| **rpoplpush source dest**  把source的尾部拿出放在dest的头部，并返回该单元值  应用：用双链表完成安全任务队列  业务逻辑:  1)rpoplpush task bak  2)接收返回值，并做业务处理  3)如果成功，rpop bak清除任务. 如不成功，下次从bak表里取任务 | 127.0.0.1:6379> rpush task a b c d  (integer) 4  127.0.0.1:6379> rpoplpush task bak  "d"  127.0.0.1:6379> rpop bak  "d"  127.0.0.1:6379> rpoplpush task bak  "c"  127.0.0.1:6379> rpop bak  "c" |

## blpop/brpop 一直等待弹出头/尾成员

|  |  |
| --- | --- |
| **blpop/brpop key timeout**  等待弹出key的头/尾成员  Timeout为等待超时时间  如果timeout为0,则一直等待  场景: 长轮询Ajax,在线聊天时,能够用到 | 第一个终端操作：  127.0.0.1:6379> brpop chat 0  1) "chat"  2) "hello"  (40.97s)  第二个终端操作：  127.0.0.1:6379> rpush chat "hello"  (integer) 1 |

# 6 无序集合操作

集合特性：无序性、唯一性、确定性

## sadd 往集合添加成员

|  |  |
| --- | --- |
| **sadd key value1 value2 ...**  往集合key中增加成员  增加相同成员时只会添加一个(唯一性) | 127.0.0.1:6379> sadd names zhangsan lisi  (integer) 2  127.0.0.1:6379> sadd names wangwu wangwu  (integer) 1 |

## srem 删除集合成员

|  |  |
| --- | --- |
| **srem key value1 value2 ...**  删除集合中为value1 value2...成员  返回真正删除掉的成员个数(不包括不存在的成员) | 127.0.0.1:6379> sadd names zhangsan lisi wangwu  (integer) 3  127.0.0.1:6379> srem names zhangsan lisi  (integer) 2  127.0.0.1:6379> smembers names  1) "wangwu" |

## spop 随机删除集合一个成员

|  |  |
| --- | --- |
| **spop key**  随机删除集合key中的一个成员  应用场景：抽奖，抽中的人已经排除，不可能会被再次抽中了 | 127.0.0.1:6379> sadd letters A B C D E F  (integer) 6  127.0.0.1:6379> spop letters  "A"  127.0.0.1:6379> spop letters  "F"  127.0.0.1:6379> spop letters  "B"  127.0.0.1:6379> spop letters  "D" |

## srandmember 随机获取集合成员

|  |  |
| --- | --- |
| **srandmember key [count]**  随机获取集合key的count个成员，默认count是1 | 127.0.0.1:6379> srandmember letters  "C"  127.0.0.1:6379> srandmember letters 2  1) "E"  2) "B"  127.0.0.1:6379> srandmember letters 3  1) "D"  2) "C"  3) "E" |

## smembers 获取集合所有的成员

|  |  |
| --- | --- |
| **smembers key**  返回集合所有的成员  返回值的顺序不一定是添加成员的顺序(无序性) | 127.0.0.1:6379> sadd names zhangsan lisi wangwu  (integer) 3  127.0.0.1:6379> smembers names  1) "lisi"  2) "wangwu"  3) "zhangsan" |

## sismember 判断成员是否存在集合中

|  |  |
| --- | --- |
| **sismember key value**  判断value是否存在集合key中  存在返回1，不存在返回0 | 127.0.0.1:6379> sadd names zhangsan lisi wangwu  (integer) 3  127.0.0.1:6379> sismember names lisi  (integer) 1  127.0.0.1:6379> sismember names zhaoliu  (integer) 0 |

## scard 获取集合成员的个数

|  |  |
| --- | --- |
| **scard key**  获取集合成员的个数 | 127.0.0.1:6379> sadd letters A B C D  (integer) 4  127.0.0.1:6379> sadd letters E F  (integer) 2  127.0.0.1:6379> scard letters  (integer) 6 |

## smove 把一个集合中成员移动到另一个集合

|  |  |
| --- | --- |
| **smove source dest value**  把集合source中的value删除，并添加到集合dest中 | 127.0.0.1:6379> sadd letters A B C  (integer) 3  127.0.0.1:6379> sadd num 1 2 3  (integer) 3  127.0.0.1:6379> smove letters num A  (integer) 1  127.0.0.1:6379> smembers letters  1) "C"  2) "B"  127.0.0.1:6379> smembers num  1) "3"  2) "1"  3) "A"  4) "2" |

## sinter 获取多个集合的交集

|  |  |
| --- | --- |
| **sinter key1 key2 ...**  获取多个集合的交集 | 127.0.0.1:6379> sadd zhangsan A E G  (integer) 3  127.0.0.1:6379> sadd lisi B E F  (integer) 3  127.0.0.1:6379> sadd wangwu C D E  (integer) 3  127.0.0.1:6379> sinter zhangsan lisi wangwu  1) "E" |

## sunion 获取多个集合的并集

|  |  |
| --- | --- |
| **sunion key1 key2 ...**  获取多个集合的并集 | 127.0.0.1:6379> sadd zhangsan A E G  (integer) 3  127.0.0.1:6379> sadd lisi B E F  (integer) 3  127.0.0.1:6379> sadd wangwu C D E  (integer) 3  127.0.0.1:6379> sunion zhangsan lisi wangwu  1) "B"  2) "G"  3) "D"  4) "C"  5) "E"  6) "F"  7) "A" |

## sdiff 获取多个集合的差集

|  |  |
| --- | --- |
| **sdiff key1 key2 ...**  获取key1与key2...的差集  即key1-key2...(key1有其他集合没有的成员) | 127.0.0.1:6379> sadd zhangsan A B C  (integer) 3  127.0.0.1:6379> sadd lisi B D E  (integer) 3  127.0.0.1:6379> sadd wangwu C E F  (integer) 3  127.0.0.1:6379> sdiff zhangsan lisi wangwu  1) "A" |

## sinterstore 获取多个集合的交集并储存

|  |  |
| --- | --- |
| **sinterstore dest key1 key2 ...**  求出key1 key2 ...集合中的交集，并赋给dest | 127.0.0.1:6379> sadd zhangsan A C D  (integer) 3  127.0.0.1:6379> sadd lisi B D E  (integer) 3  127.0.0.1:6379> sadd wangwu D E G  (integer) 3  127.0.0.1:6379> sinterstore class zhangsan lisi wangwu  (integer) 1  127.0.0.1:6379> smembers class  1) "D" |

# 7 有序集合操作

## zadd 往有序集合添加成员

|  |  |
| --- | --- |
| **zadd key score1 key2 score2 key2 ...**  往有序集合key添加成员 | 127.0.0.1:6379> zadd ages 28 zhangsan 24 lisi 26 wangwu  (integer) 0  127.0.0.1:6379> zrange ages 0 -1  1) "lisi"  2) "wangwu"  3) "zhangsan" |

## zrange 按名次取成员

|  |  |
| --- | --- |
| **zrange key start stop [WITHSCORES]**  把集合排序后，返回名次[start,stop]的成员  按名次取成员  默认是升续排列  withscores 是把score也打印出来 | 127.0.0.1:6379> zrange ages 0 -1 withscores  1) "lisi"  2) "24"  3) "wangwu"  4) "26"  5) "zhangsan"  6) "28" |

## zrangebyscore 按分数取成员

|  |  |
| --- | --- |
| **zrangebyscore key min max [withscores] limit offset N**  集合(升续)排序后，取score在[min,max]内的成员，  并跳过offset个，取出N个  按分数取成员 | 127.0.0.1:6379> zadd ages 28 zhangsan 24 lisi 26 wangwu  (integer) 3  127.0.0.1:6379> zrangebyscore ages 25 30  1) "wangwu"  2) "zhangsan"  127.0.0.1:6379> zrangebyscore ages 25 30 limit 1 1  1) "zhangsan" |

## zscore 获取指定成员的分数

|  |  |
| --- | --- |
| **ZSCORE key member**  获取指定成员的分数 | 127.0.0.1:6379> zadd height 175 zhangsan 167 lisi 185 wangwu  (integer) 3  127.0.0.1:6379> zscore height lisi  "167" |

## zcard 获取成员个数

|  |  |
| --- | --- |
| **zcard key**  获取成员个数 | 127.0.0.1:6379> zadd height 175 zhangsan 167 lisi 185 wangwu  (integer) 3  127.0.0.1:6379> zcard height  (integer) 3 |

## zcount 计算分数区间成员个数

|  |  |
| --- | --- |
| **zcount key min max**  计算[min,max]区间内成员的数量 | 127.0.0.1:6379> zadd height 175 zhangsan 167 lisi 185 wangwu  (integer) 3  127.0.0.1:6379> zcount height 170 180  (integer) 1 |

## zrank/zrevrank 获取成员升序/降序的排名

|  |  |
| --- | --- |
| **zrank/zrevrank key member**  查询member的升序/降序排名  名次从0开始 | 127.0.0.1:6379> zadd ages 28 zhangsan 24 lisi 26 wangwu  (integer) 0  127.0.0.1:6379> zrange ages 0 -1  1) "lisi"  2) "wangwu"  3) "zhangsan"  127.0.0.1:6379> zrank ages zhangsan  (integer) 2  127.0.0.1:6379> zrevrank ages zhangsan  (integer) 0 |

## zrem 删除有序集合成员

|  |  |
| --- | --- |
| **zrem key value1 value2 ..**  删除集合中的成员 | 127.0.0.1:6379> zrem ages wangwu  (integer) 1  127.0.0.1:6379> zrange ages 0 -1  1) "lisi"  2) "zhangsan"  127.0.0.1:6379> |

## zremrangebyrank 按排名删除成员

|  |  |
| --- | --- |
| **zremrangebyrank key start end**  按排名删除成员，删除名次在[start,end]之间的 | 127.0.0.1:6379> zadd height 175 zhangsan 167 lisi 185 wangwu 178 zhaoliu  (integer) 1  127.0.0.1:6379> zremrangebyrank height 0 1  (integer) 2  127.0.0.1:6379> zrange height 0 -1  1) "zhaoliu"  2) "wangwu" |

## zremrangebyscore 按分数删除成员

|  |  |
| --- | --- |
| **zremrangebyscore key min max**  作用: 按照socre来删除成员，删除score在[min,max]之间的 | 127.0.0.1:6379> zadd height 175 zhangsan 167 lisi 185 wangwu 178 zhaoliu  (integer) 2  127.0.0.1:6379> zremrangebyscore height 170 180  (integer) 2  127.0.0.1:6379> zrange height 0 -1  1) "lisi"  2) "wangwu" |

## zinterstore 求交集再计算

|  |  |
| --- | --- |
| **zinterstore destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX]**  求key1、key2...的交集，key1、key2...的权重分别是 weight1、weight2...  聚合方法用: sum|min|max  聚合的结果保存在destination集合内 | 127.0.0.1:6379> zadd zhangsan 5 iphone6s 7 galaxyS7 6 huaweiP9  (integer) 3  127.0.0.1:6379> zadd lisi 3 iphone6s 9 galaxyS7 4 huaweiP9 2 HTC10  (integer) 4  127.0.0.1:6379> zinterstore result 2 zhangsan lisi  (integer) 3  127.0.0.1:6379> zrange result 0 -1 withscores  1) "iphone6s"  2) "8"  3) "huaweiP9"  4) "10"  5) "galaxyS7"  6) "16"  127.0.0.1:6379> zinterstore result 2 zhangsan lisi aggregate max  (integer) 3  127.0.0.1:6379> zrange result 0 -1 withscores  1) "iphone6s"  2) "5"  3) "huaweiP9"  4) "6"  5) "galaxyS7"  6) "9" |

## zunionstore 求并集再计算

|  |  |
| --- | --- |
| **zunionstore destination numkeys key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX]**  求key1、key2...的并集，key1、key2...的权重分别是 weight1、weight2...  聚合方法用: sum|min|max  聚合的结果保存在destination集合内 | 127.0.0.1:6379> zadd zhangsan 4 iphone6s 6 huaweiP9 8 xiaomi5  (integer) 3  127.0.0.1:6379> zadd lisi 2 iphone6s 8 galaxS7 5 meizu6  (integer) 3  127.0.0.1:6379> zunionstore result 2 zhangsan lisi  (integer) 5  127.0.0.1:6379> zrange result 0 -1 withscores  1) "meizu6"  2) "5"  3) "huaweiP9"  4) "6"  5) "iphone6s"  6) "6"  7) "galaxS7"  8) "8"  9) "xiaomi5"  10) "8"  127.0.0.1:6379> zunionstore result 2 zhangsan lisi aggregate max  (integer) 5  127.0.0.1:6379> zrange result 0 -1 withscores  1) "iphone6s"  2) "4"  3) "meizu6"  4) "5"  5) "huaweiP9"  6) "6"  7) "galaxS7"  8) "8"  9) "xiaomi5"  10) "8" |

# 8 哈希数据类操作

## hset 设置哈希field域的值

|  |  |
| --- | --- |
| **hset key field value**  把key中 filed域的值设为value  注:如果没有field域，直接添加，如果有，则覆盖原field域的值 | 127.0.0.1:6379> hset user name zhangsan  (integer) 1  127.0.0.1:6379> hset user age 25  (integer) 1  127.0.0.1:6379> hset user gender male  (integer) 1 |

## hmset 设置哈希多个field域的值

|  |  |
| --- | --- |
| **hmset key field1 value1 [field2 value2 field3 value3 ......fieldn valuen]**  一次设置多个field和对应的value | 127.0.0.1:6379> hmset user name lisi age 26 gender male  OK |

## hget 获取field域的值

|  |  |
| --- | --- |
| **hget key field**  获取field域的值 | 127.0.0.1:6379> hmset user name lisi age 26 gender male  OK  127.0.0.1:6379> hget user age  "26 |

## hmget 获取多个field域的值

|  |  |
| --- | --- |
| **hget key field**  获取多个field域的值 | 127.0.0.1:6379> hmset user name lisi age 26 gender male  OK  127.0.0.1:6379> hmget user name age  1) "lisi"  2) "26" |

## hgetall 获取所有field域和值

|  |  |
| --- | --- |
| **hget key**  获取哈希key的所有field域和值 | 127.0.0.1:6379> hmset user name lisi age 26 gender male  127.0.0.1:6379> hgetall user  1) "name"  2) "lisi"  3) "age"  4) "26"  5) "gender"  6) "male" |

## hlen 获取field的数量

|  |  |
| --- | --- |
| **hlen key**  获取field的数量 | 127.0.0.1:6379> hmset user name lisi age 26 gender male  127.0.0.1:6379> hlen user  (integer) 3 |

## hdel 删除field域

|  |  |
| --- | --- |
| **hdel key field**  删除key中 field域 | 127.0.0.1:6379> hmset user name lisi age 26 gender male  127.0.0.1:6379> hdel user age  (integer) 1  127.0.0.1:6379> hgetall user  1) "name"  2) "lisi"  3) "gender"  4) "male" |

## hexists 判断field域是否存在

|  |  |
| --- | --- |
| **hexists key field**  判断key中有没有field域 | 127.0.0.1:6379> hmset user name lisi age 26 gender male  OK  127.0.0.1:6379> hexists user age  (integer) 1  127.0.0.1:6379> hexists user height  (integer) 0 |

## hincrby 使field域的值加上整数

|  |  |
| --- | --- |
| **hincrby key field value**  使key中的field域的值加上整型值value | 127.0.0.1:6379> hmset user name zhangsan height 158  OK  127.0.0.1:6379> hincrby user height 2  (integer) 160 |

## hincrbyfloat 使field域的值加上浮点数

|  |  |
| --- | --- |
| **hincrbyfloat key field value**  使key中的field域的值加上浮点值value | 127.0.0.1:6379> hmset user name zhangsan height 158  OK  127.0.0.1:6379> hincrbyfloat user height 5.5  "165.5" |

## hkeys 获取所有所有field域的名字

|  |  |
| --- | --- |
| **hkeys key**  获取key中所有的field | 127.0.0.1:6379> hmset user name zhangsan age 25 gender male  OK  127.0.0.1:6379> hkeys user  1) "name"  2) "age"  3) "gender" |

## kvals 获取所有所有field域的值

|  |  |
| --- | --- |
| **kvals key**  作用: 返回key中所有的value | 127.0.0.1:6379> hmset user name zhangsan age 25 gender male  OK  127.0.0.1:6379> hvals user  1) "zhangsan"  2) "25"  3) "male" |

# 9 redis事务

redis支持简单的事务。

(1) redis语法

|  |
| --- |
| 开启事务命令：muitl  操作数据： 普通命令  在提交前取消：discard  提交： exec |

(2) Redis与 mysql事务的对比

|  |  |  |
| --- | --- | --- |
|  | Mysql | Redis |
| 开启 | start transaction | muitl |
| 语句 | 普通sql | 普通命令 |
| 失败 | rollback 回滚 | discard 取消 |
| 提交 | commit | exec |

注: rollback与discard 的区别

如果已经成功执行了2条语句, 第3条语句出错。

Rollback后，前2条的语句影响消失。

Discard只是结束本次事务，前2条语句已经执行，造成的影响仍然还在。

语句出错有两种情况：

①语法就有问题，exec时报错，所有语句取消执行，没有对数据造成影响。

②语法本身没错，但适用对象有问题(比如 zadd 操作list对象)，exec之后，会执行正确的语句，并跳过有不适当的语句，对数据会造成影响，这点由程序员负责。

(3) redis乐观锁

redis的事务中启用的是乐观锁，只负责监测key没有被改动，如果在事务中发现key被改动，则取消事务。

使用watch命令来监控一个或多个key，使用unwatch命令来取消监控所有key。

使用示例：

|  |
| --- |
| watch key  muitl  操作数据...  exec  unwatch |

(4) 模拟抢票

场景：客户买一张票，扣掉100元

|  |  |
| --- | --- |
| 在zhangsan买票过程中，在提交事务前一瞬间，有人成功买到票，ticket已经改变(即使ticket还有票)，导致zhangsan抢票失败。 | lisi在买票整个过程都没有人抢票，所以lisi一次抢票成功。 |
| 127.0.0.1:6379> watch ticket  OK  127.0.0.1:6379> multi  OK  127.0.0.1:6379> decr ticket  QUEUED  127.0.0.1:6379> decrby zhangsan 100  QUEUED  127.0.0.1:6379> exec  (nil)  127.0.0.1:6379> get zhangsan  "1000"  127.0.0.1:6379> get ticket  "2"  127.0.0.1:6379> unwatch  OK | 127.0.0.1:6379> watch ticket  OK  127.0.0.1:6379> multi  OK  127.0.0.1:6379> decrby lisi 100  QUEUED  127.0.0.1:6379> decr ticket  QUEUED  127.0.0.1:6379> exec  1) (integer) 700  2) (integer) 1  127.0.0.1:6379> unwatch |

# 10 频道发布与订阅

语法：

|  |  |
| --- | --- |
| 发布端 | 订阅端 |
| publish 频道名称 发布内容 | subscribe 频道名称 |
| psubscribe 匹配频道名称 |

示例：

|  |  |  |
| --- | --- | --- |
| **发布端** | **订阅music频道** | **订阅music\*频道** |
| 127.0.0.1:6379> publish music\_2 "It's Not Goodbye"  (integer) 1  127.0.0.1:6379> publish music "just one last dance"  (integer) 2  127.0.0.1:6379> publish music "stay"  (integer) 2 | 127.0.0.1:6379> subscribe music  Reading messages... (press Ctrl-C to quit)  1) "subscribe"  2) "music"  3) (integer) 1  1) "message"  2) "music"  3) "just one last dance"  1) "message"  2) "music"  3) "stay" "music"  3) (integer) 1  1) "message"  2) "music"  3) "just one last dance"  1) "message"  2) "music"  3) "stay" | 127.0.0.1:6379> psubscribe music\*  Reading messages... (press Ctrl-C to quit)  1) "psubscribe"  2) "music\*"  3) (integer) 1  1) "pmessage"  2) "music\*"  3) "music"  4) "just one last dance"  1) "pmessage"  2) "music\*"  3) "music"  4) "stay" |
| 127.0.0.1:6379> publish music\_2 "It's Not Goodbye"  (integer) 1 |  | 1) "pmessage"  2) "music\*"  3) "music\_2"  4) "It's Not Goodbye" |

# 11 redis持久化

## 11.1 rdb快照持久化

rdb的工作原理：每隔N分钟或N次写操作后，从内存dump数据形成rdb文件，压缩放在备份目录。

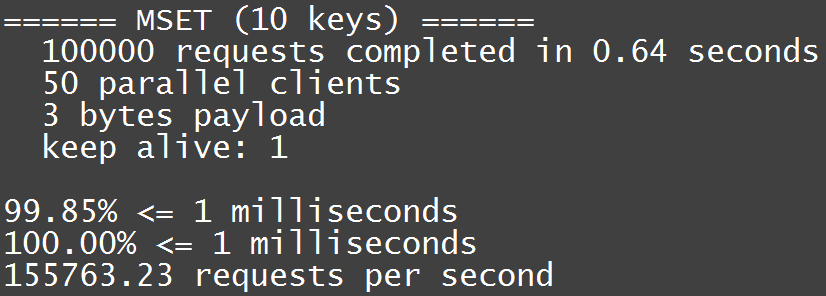
设置配置文件参数

**vim /usr/local/redis/ redis.conf**

|  |
| --- |
| save 900 1 # 900秒内，有1条写入，则产生快照  save 300 1000 # 如果300秒内有1000次写入，则产生快照  save 60 10000 # 如果60秒内有10000次写入，则产生快照  (这3个选项都屏蔽,则rdb禁用)  stop-writes-on-bgsave-error yes # 后台备份进程出错时，主进程停不停止写入  rdbcompression yes # 导出的rdb文件是否压缩  Rdbchecksum yes # 导入rbd恢复时数据时，要不要检验rdb的完整性  dbfilename dump.rdb # 导出来的rdb文件名  dir /usr/local/redis/data # rdb的放置路径 |

用压力测试来检测是否启用了rdb快照

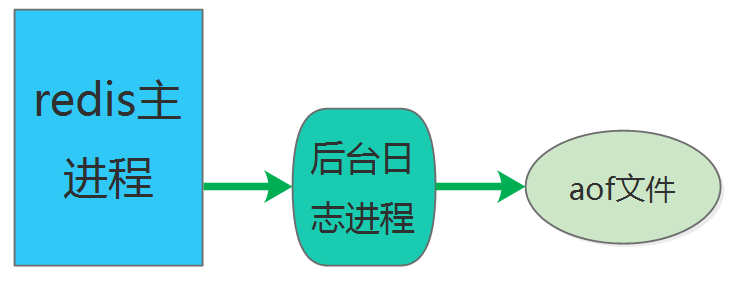
**/usr/local/redis/bin/redis-benchmark**



rdb的缺陷：在2个保存点之间断电，将会丢失1-N分钟的数据

## 11.2 aof日志持久化

工作原理：



设置配置文件参数

**vim /usr/local/redis/ redis.conf**

|  |
| --- |
| appendonly no # 是否打开 aof日志功能  appendfilename "appendonly.aof" # aof文件名，和rdb的dir公用一个路径  #appendfsync always # 每1个写命令都立即同步到aof文件，安全但速度慢  appendfsync everysec # 折衷方案，每秒写1次  上面方案选择一种，一般选择everysec  appendfsync no # 写入工作交给操作系统，由操作系统判断缓冲区大小，统一写入到aof文件，同步频率低，但速度快  no-appendfsync-on-rewrite yes # 正在导出rdb快照的过程中，要不要停止同步aof  auto-aof-rewrite-percentage 100 # aof文件大小比起上次重写时的大小，增长率100%时重写  auto-aof-rewrite-min-size 64mb # aof文件至少超过64M时才重写 |

注意：一般推荐rdb和aof同时开启，同时开启后redis进程启动优先选择aof恢复数据。rdb恢复速度快。

在dump rdb过程中，aof如果停止同步，会不会丢失?

答: 不会，所有的操作缓存在内存的队列里，dump完成后统一操作.

aof重写是指什么?

答: aof重写是指把内存中的数据，逆化成命令，写入到.aof日志里，

以解决 aof日志过大的问题。手动重写aof命令：bgrewriteaof

# 12 设置远程客户端登录

配置redis.conf文件，找到bind关键词

**vim /usr/local/redis/redis.conf**

|  |
| --- |
| bind 192.168.8.102 # 默认绑定127.0.0.1，修改为绑定本机ip启动  bind 192.168.8.101 # 允许远程连接ip，多个ip用空格分开 |

重启redis

**pkill -9 redis**

**/usr/local/redis/bin/redis-server redis.conf**

# 13 redis的开发语言使用

## 13.1 redis的php开发

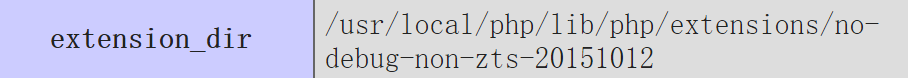
**(1) 安装redis的php扩展**

|  |  |
| --- | --- |
| 下载源码 | wget http://pecl.php.net/get/redis-2.2.8.tgz |
| 解压 | tar zxvf redis-2.2.8.tgz |
| 进入目录 | cd redis-2.2.8 |
| 使用phpize命令生成配置文件 | /usr/local/php/bin/phpize |
| 检查编译环境 | ./configure --with-php-config=/usr/local/php/bin/php-config |
| 编译和安装 | make && make install |

最后配置php.ini，添加redis扩展

通过phpinfo()函数输出，在页面上看到的配置文件路径和扩展包的路径





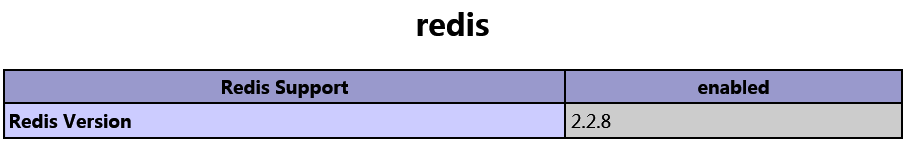
打开配置文件：**vim /usr/local/php/lib/php.ini**

(如果配置文件不存在则从php安装包中复制复制一份配置文件)

查找.so，在附近添加下面内容：

|  |
| --- |
| extension=/usr/local/php/lib/php/extensions/no-debug-non-zts-20121212/redis.so |

通过phpinfo()函数输出结果，如果出现redis扩展，如下图所示说明安装php扩展成功



**(2) redis的php连接使用**

string简单使用如下，其他类型使用请查看官方文档<https://github.com/phpredis/phpredis>

|  |
| --- |
| **<?php** *// 新建redis对象* $redis = **new** Redis();  *// 连接redis数据库* $redis->connect(**'127.0.0.1'**, 6379);  *// 设置键值* $redis->set(**'site'**,**'www.baidu.com'**);  *// 读取键值* **echo** $redis->get(**'site'**),**'<br>'**;  *// 修改键值* $redis->set(**'site'**,**'www.taobao.com'**);  **echo** $redis->get(**'site'**);  *// 关闭连接*  $redis->close(); |

## 13.2 redis的go语言开发

打开终端安装redis第三方包：

go get github.com/garyburd/redigo/redis

string简单使用如下，其他类型使用请查看官方文档<http://godoc.org/github.com/garyburd/redigo/redis>

|  |
| --- |
| **package** main  **import** (  **"github.com/garyburd/redigo/redis"  "fmt"** )  **var** Rconn redis.Conn  **func** main() {  *// 连接redis服务器* **var** err error  Rconn, err = redis.Dial(**"tcp"**, **"192.168.8.101:6379"**)  **if** err != nil {  fmt.Println(err.Error())  **return** }   *// 字符串类型* Rconn.Do(**"set"**,**"site"**,**"www.baidu.com"**)  result,\_:=redis.String(Rconn.Do(**"get"**,**"site"**))  fmt.Println(result) } |

# 14 redis导出导入数据库

**(1) 安装redis-dump工具**

yum install ruby rubygems ruby-devel

gem sources --remove <http://ruby.taobao.org/>

gem sources -a <https://ruby.taobao.org/>

gem install redis-dump -V

**(2) 导出redis数据**

redis-dump -u 127.0.0.1:6379 > test.json

**(3) 导入redis数据**

< test.json redis-load