Redis 的Key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。通配符：

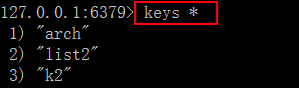
\* ：表示 0-多个字符 ，例如：keys \* 查询所有的 key。

？：表示单个字符，例如：wo?d , 匹配 word , wood

[]：匹配[]里边的一个字符

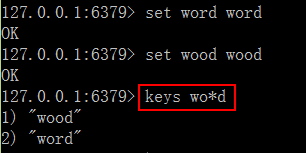
例 1：显示所有的 key

keys \*：查看数据库中所有的key



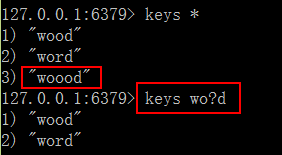
例 2：使用 \* 表示 0 或多个字符  
keys k\*：查看数据库中所有以k开头的key

keys k\*o：查看数据库中以h开头、以o结尾的key



例 3：使用 ？ 表示单个字符

keys h?o：查看数据库中所有以h开头，以o结尾、并且中间只有一个字符的key



例4:keys h[abc]llo：查看数据库中以h开头、以llo结尾，并且h后边只能去[abc]中的一个字符的key

* exists

语法：exists key [key…]

作用：判断 key 是否存在

返回值：

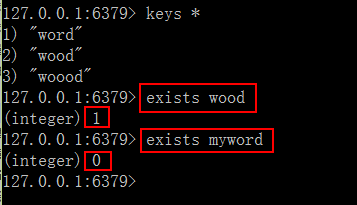
exists key

整数，存在 key 返回 1，其他返回 0.

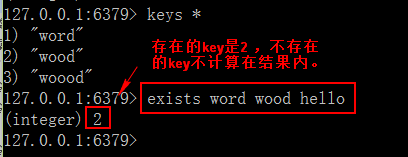
exists [key key ...]

使用多个 key，返回存在的 key 的数量。

例 1：检查指定 key 是否存在



例 2：检查多个 key



移动指定key到指定数据库实例：move key index

move k 1

* 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

例 1： 设置红灯的倒计时是 5 秒



* ttl

语法：ttl key

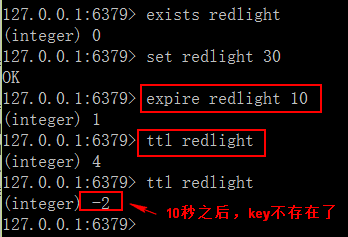
作用：以秒为单位，返回 key 的剩余生存时间（ttl: time to live） 返回值：

● -1 ：没有设置 key 的生存时间， key 永不过期

● -2 ：key 不存在

● 数字：key 的剩余时间，秒为单位

例 1：设置 redlight 的过期时间是 10， 查看剩余时间



* 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

● none (key 不存在)

● string (字符串)

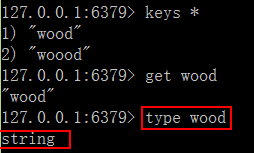
● list (列表)

● set (集合)

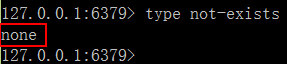
● zset (有序集)

● hash (哈希表)

例 1：查看存储字符串的 key ：wood



例 2：查看不存在的 key



* 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。

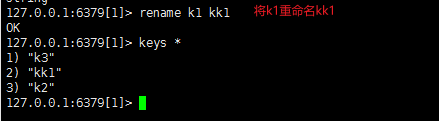
例 1：删除指定的 key



* rename

重命名key

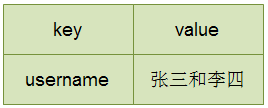
rename key newkey



**Redis中5种数据类型**

【1】字符串类型 string

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。最大 512M。



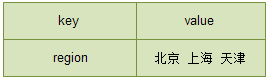
【2】哈希类型 hash

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



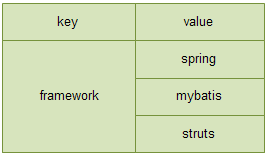
【3】列表类型 list

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素到列表的头部（左边）或者尾部（右边）



【4】集合类型 set

Redis 的 Set 是 string 类型的无序集合，集合成员是唯一的，即集合中不能出现重复的数据.



【5】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。

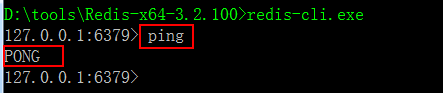


Redis 字符串类型

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。

基本命令

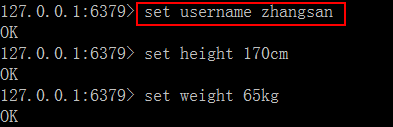
先测试能连接到 redis 服务器



* set

将字符串值 value 设置到 key 中

语法：set key value



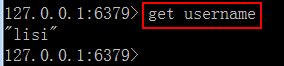
向已经存在的 key 设置新的 value，会覆盖原来的值

IMG_256

* get

获取 key 中设置的字符串值语法： get key

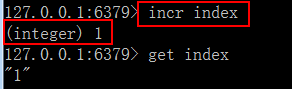
例如：获取 username 这个 key 对应的 value



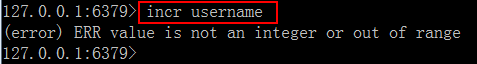
* incr

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（只能对数字类型的数据操作） 语法：incr key

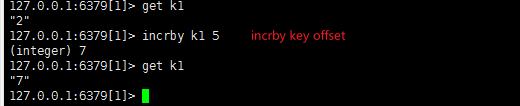
例 1：操作key,值增加 1



例 2：对非数字的值操作是不行的



将字符串数值进行加offset运算：incrby key offset

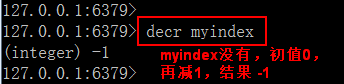


* decr

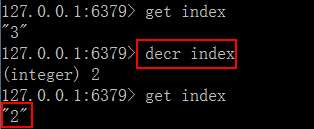
将 key 中储存的数字值减1，如果 key 不存在，则么 key 的值先被初始化为 0 再执行 decr 操作（只能对数字类型的数据操作）

语法：decr key

例1：不存在的key，初值为0，再减 1

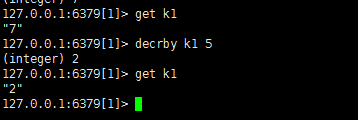


例2：对存在的数字值的 key ，减 1 。先执行 incr index ,增加到3



incr ，decr 在实现关注人数上，文章的点击数上。

将字符串数值进行减offset运算：decrby key offset



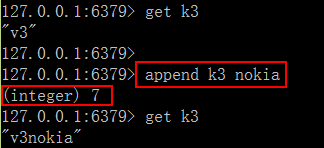
* append

语法：append key value

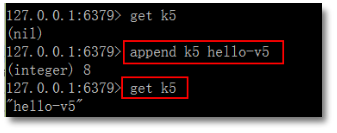
说明：如果 key 存在， 则将 value 追加到 key 原来旧值的末尾如果 key 不存在， 则将 key 设置值为 value

返回值：追加字符串之后的总长度

例 1：追加内容到存在的 key



例 2：追加到不存在的 key，同 set key value



* strlen

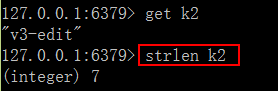
语法：strlen key

说明：返回 key 所储存的字符串值的长度返回值：

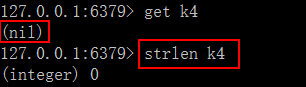
  ①：如果key存在，返回字符串值的长度

  ②：key不存在，返回0

例 1：计算存在 key 的字符串长度



设置中文 set k4 中文长度 ， 按字符个数计算



* getrange

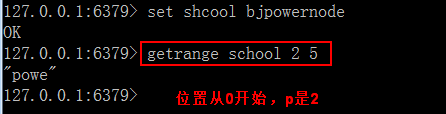
语法：getrange key start end

作用：获取 key 中字符串值从 start 开始 到 end 结束 的子字符串,包括 start 和 end, 负数表示从字符串的末尾开始， -1 表示最后一个字符，负下标表示自右向左。

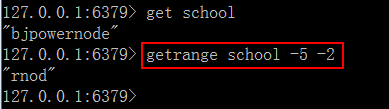
返回值：截取的子字符串

使用的字符串 key: school, value: bjpowernode

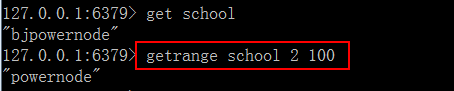
例 1: 截取从 2 到 5 的字符



例 2：从字符串尾部截取，start ,end 是负数，最后一位是 -1



例 3：超出字符串范围的截取 ，获取合理的子串



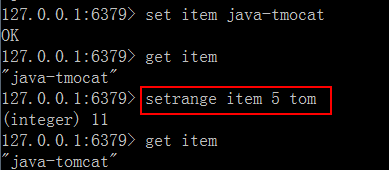
* setrange

语法：setrange key offset value

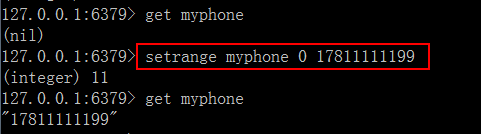
说明：用 value 覆盖（替换）key 的存储的值从 offset 开始,不存在的 key 做空白字符串。

返回值：修改后的字符串的长度

例 1：替换给定的字符串



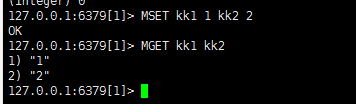
例 2：设置不存在的 key



* mset

语法：mset k1 v1 k2 v2...

批量设置k-v

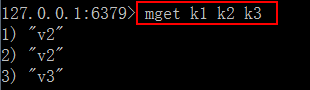


* mget

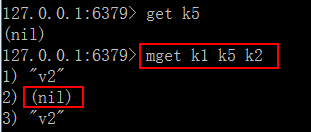
语法：mget key [key …]

作用：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

例 1：返回多个 key 的存储值



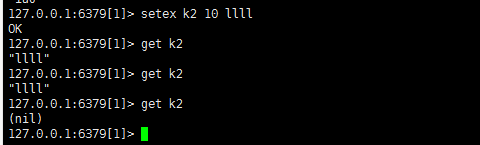
例 2：返回不存在的



* setex

语法：setex key seconds value

设置字符串数据的同时，设置它的最大声明周期。



* setnx

语法：setnx key value

设置字符串类型的数据到redis数据库，当key不存在时创建成功，否则放弃设置。



* msetnx

语法：msetnx k1 v1 k2 v2...

批量设置字符串类型的数据到redis数据库，当key不存在时创建成功，否则只要有1个存在就全部放弃设置。



Redis中list操作命令

Redis 列表是简单的字符串列表，按照插入顺序排序。你可以添加一个元素导列表的头部（左边）或者尾部（右边）

* lpush

语法：lpush key value [value…]

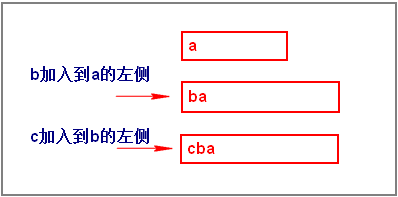
作用：将一个或多个值 value 插入到列表 key 的表头（最左边），从左边开始加入值，从左到右的顺序依次插入到表头

返回值：数字，新列表的长度

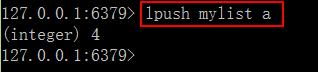
类似数据结构的链表头插法

例 1：将 a,b,c 插入到 mylist 列表类型





例 2：插入重复值到 list 列表类型



* rpush

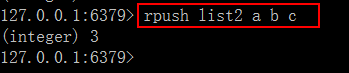
语法：rpush key value [value…]

作用：将一个或多个值 value  插入到列表 key  的表尾（最右边），各个 value 值按从左到右的顺序依次插入到表尾

返回值：数字，新列表的长度

类似数据结构的链表尾插法

例 1：插入多个值到列表



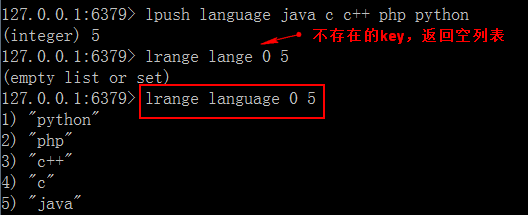
* lrange

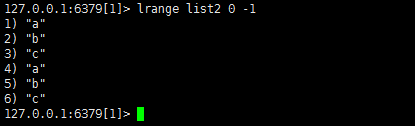
语法：lrange key start stop

作用：获取列表 key 中指定区间内的元素，0 表示列表的第一个元素，以 1 表示列表的第二个元素；start , stop 是列表的下标值，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。 start ，stop 超出列表的范围不会出现错误。

返回值：指定区间的列表

例 1：返回列表的全部内容





例 2：显示列表中第 2 个元素，下标从 0 开始

IMG_256

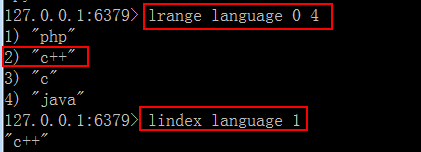
* lindex

语法：lindex key index

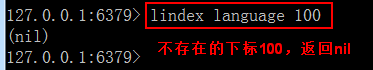
作用：获取列表 key 中下标为指定 index 的元素，列表元素不删除，只是查询。0 表示列表的第一个元素，以 1 表示列表的第二个元素；start , stop 是列表的下标值，也可以负数的下标， -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。

返回值：指定下标的元素；index 不在列表范围，返回 nil

例 1：返回下标是 1 的元素



例 2：不存在的下标



* llen

语法：llen key

作用：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

例 1：显示存在 key 的列表元素的个数

IMG_256

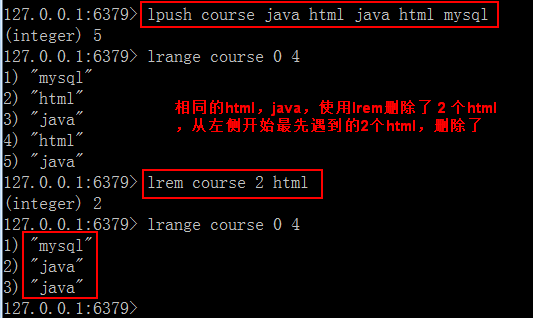
* lrem

语法：lrem key count value

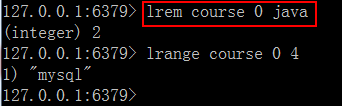
作用：根据参数 count 的值，移除列表中与参数 value 相等的元素， count >0 ，从列表的左侧向右开始移除； count < 0 从列表的尾部开始移除；count = 0 移除表中所有与 value 相等的值。

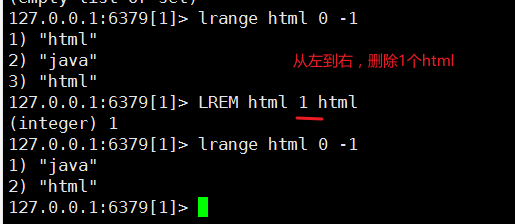
返回值：数值，移除的元素个数

例 1：删除 2 个相同的列表元素



例 2：删除列表中所有的指定元素，删除所有的 java





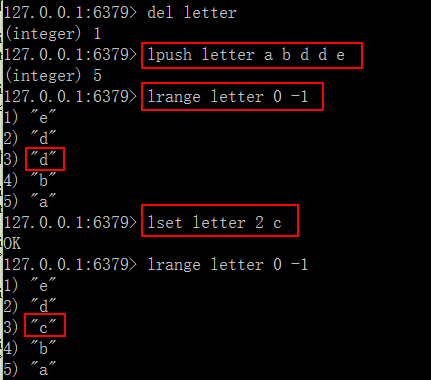
* lset

语法：lset key index value

作用：将列表 key 下标为 index 的元素的值设置为 value。

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

例 1：设置下标 2 的 value 为“c”



* linsert

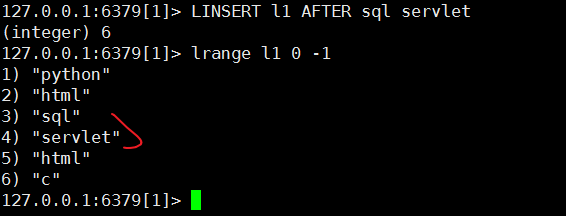
语法：linsert key BEFORE|ALFTER pivot value

作用：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，pivot 不在列表中，不执行任何操作。

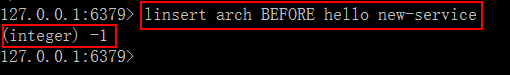
返回值：命令执行成功，返回新列表的长度。没有找到 pivot 返回 -1， key 不存在返回 0

例 1：修改列表 arch，在值 dao 之前加入 service





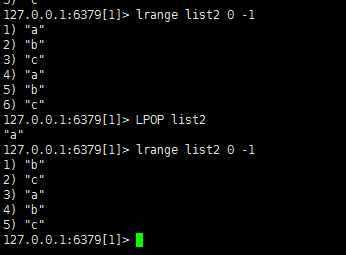
例 2：操作不存在的 pivot



* lpop

语法：lpop key

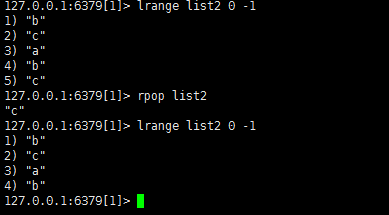
从指定列表移除并返回表头元素



* rpop

语法：rpop key

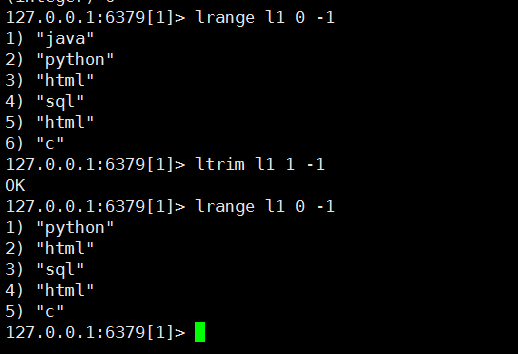
从指定列表移除并且返回表尾元素



* ltrim

语法：ltrim key startIndex endIndex

截取指定列表中指定下表区间的元素组成新的列表，并且赋值给key



Redis中set操作命令

redis 的 Set 是 string 类型的无序集合，集合成员是唯一的，即集合中不能出现重复的数据

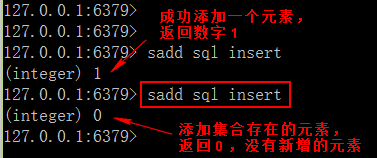
* sadd

语法：sadd key member [member…]

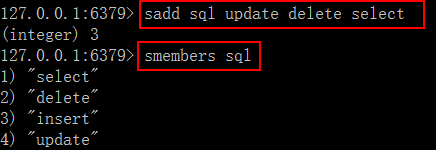
作用：将一个或多个 member 元素加入到集合 key 当中，已经存在于集合的 member 元素将被忽略，不会再加入。

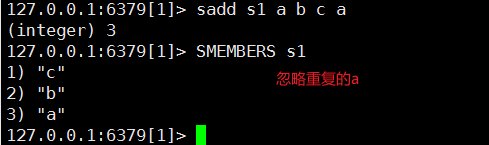
返回值：加入到集合的新元素的个数。不包括被忽略的元素

例 1：添加单个元素



例 2：添加多个元素



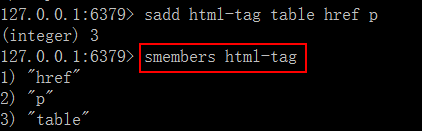


* smembers

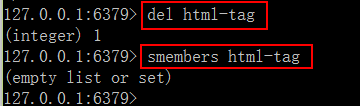
语法：smembers key

作用：获取集合 key 中的所有成员元素，不存在的 key 视为空集合

例 1：查看集合的所有元素



例 2：查看不存在的集合

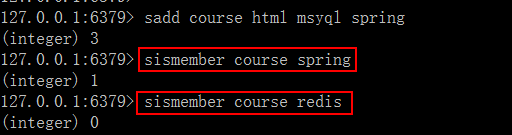


* sismember

语法：sismember key member

作用：判断 member 元素是否是集合 key 的成员返回值：member 是集合成员返回 1，其他返回 0

例 1：检查元素是否存在集合中



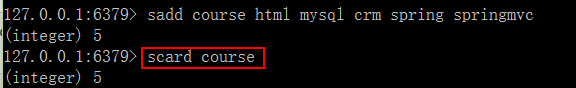
* scard

语法：scard key

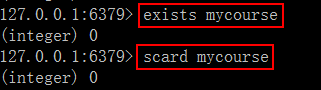
作用：获取集合里面的元素个数

返回值：数字，key 的元素个数。 其他情况返回 0

例 1：统计集合的大小



例 2：统计不存在的 key

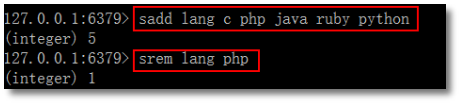


* srem

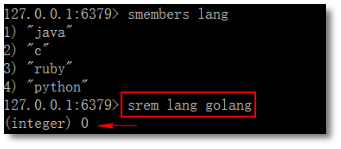
语法：srem key member [member…

作用：删除集合 key 中的一个或多个 member 元素，不存在的元素被忽略。返回值：数字，成功删除的元素个数，不包括被忽略的元素。

例 1：删除存在的一个元素，返回数字 1



例 2：删除不存在的元素



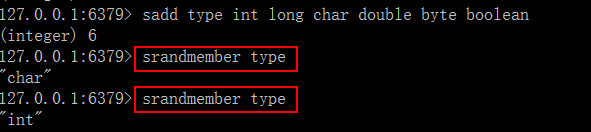
* srandmember

语法：srandmember key [count]

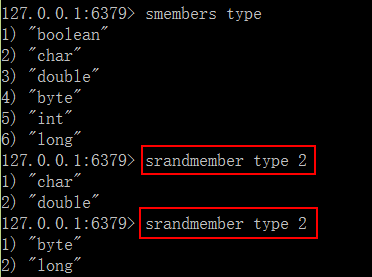
作用：只提供 key，随机返回集合中一个元素，元素不删除，依然在集合中；提供了 count 时，count 正数, 返回包含 count 个数元素的集合， 集合元素各不相同。count 是负数，返回一个 count 绝对值的长度的集合， 集合中元素可能会重复多次。

返回值：一个元素；多个元素的集合

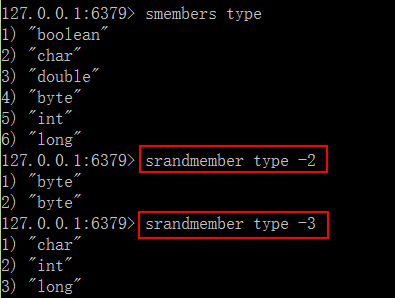
例 1：随机显示集合的一个元素



例 2：使用 count 参数， count 是正数



例 3：使用 count 参数，count 是负数

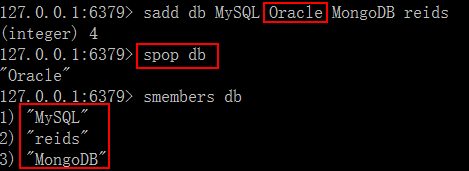


* spop

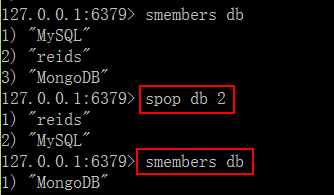
语法：spop key [count]

作用：随机从集合中删除一个元素, count 是删除的元素个数。返回值：被删除的元素，key 不存在或空集合返回 nil

例如 1：随机从集合删除一个元素



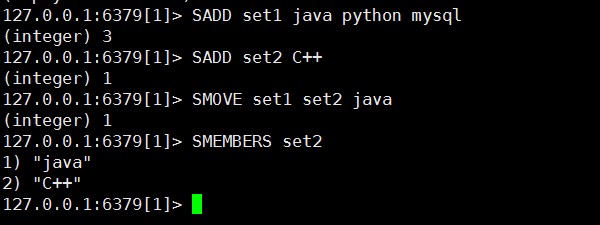
例 2：随机删除指定个数的元素



* smove

语法：smove k1 k2 member

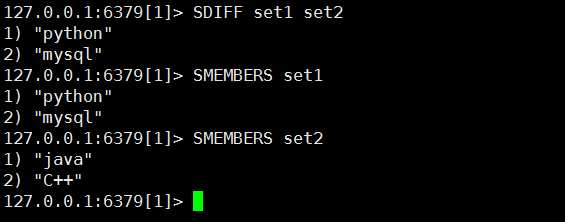
将指定集合中的指定元素移动到另一个元素。



* sdiff

语法：sdiff key key [key key ...]

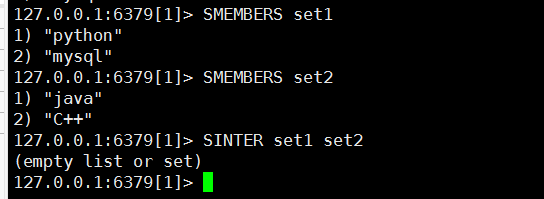
获取一个集合中有，其他集合中都没有的元素。



* sinter

语法：sinter k1 k2 k3...

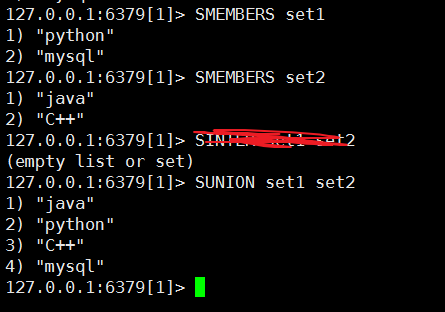
获取所有指定结合中所有元素组成的集合。

\

* sunion

语法：sunion k1 k2 k3...

获取所有集合中所有元素组成的大集合



Redis中hash操作命令

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

单key:field-value

field-value

students:id-1001

name-zhangsan

age-20

* hset

语法：hset hash 表的 key field value

作用：将哈希表 key 中的域 field 的值设为 value ，如果 key 不存在，则新建 hash 表，执行赋值，如果有 field ,则覆盖值。

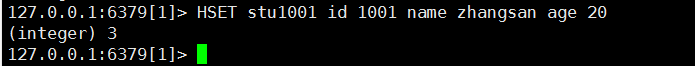
返回值：

①如果 field 是 hash 表中新 field，且设置值成功，返回 1

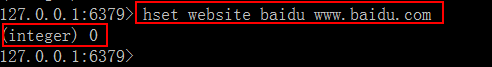
②如果 field 已经存在，旧值覆盖新值，返回 0

例 1：新的 field

IMG_256



例 2：覆盖旧的的



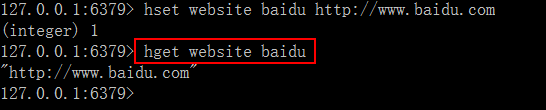
* hget

语法：hget key field

作用：获取哈希表 key 中给定域 field 的值

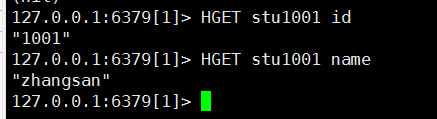
返回值：field 域的值，如果 key 不存在或者 field 不存在返回 nil

例 1：获取存在 key 值的某个域的值



例 2：获取不存在的 field

IMG_256



* hmset(弃用)

语法：hmset key field value [field value…]

说明：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，执行 hmset.

返回值：设置成功返回 ok， 如果失败返回一个错误

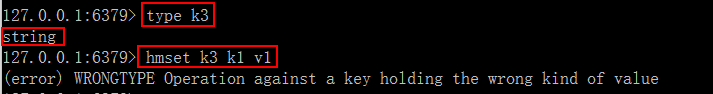
例 1：同时设置多个 field-value



使用 redis-desktop-manager 工具查看 hash 表 website 的数据结构



例 2：key 类型不是 hash,产生错误



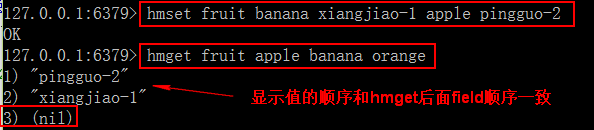
* hmget（弃用）

语法：hmget key field [field…]

作用: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

例 1：获取多个 field 的值



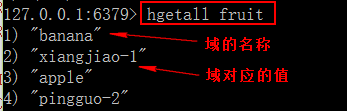
* hgetall

语法：hgetall key

作用：获取哈希表 key  中所有的域和值

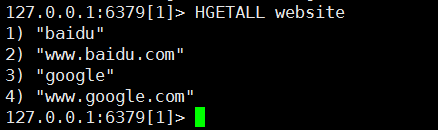
返回值：以列表形式返回 hash 中域和域的值 ，key 不存在，返回空 hash

例 1：返回 key 对应的所有域和值



例 2：不存在的 key，返回空列表

IMG_256

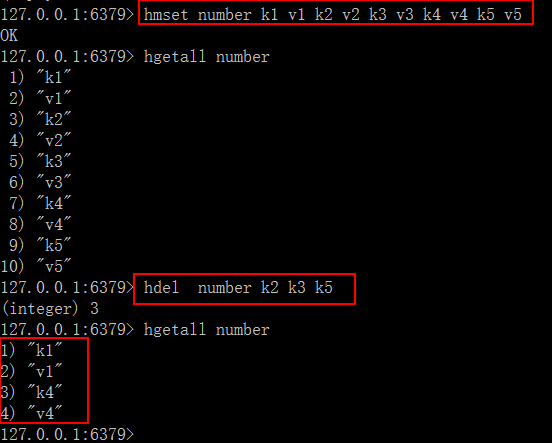


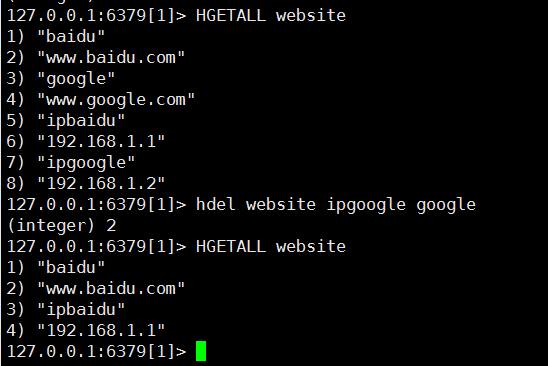
* hdel

语法：hdel key field [field…]

作用：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

例 1：删除指定的 field





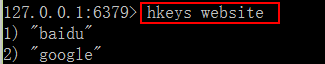
* hkeys

语法：hkeys key

作用：查看哈希表 key 中的所有 field 域

返回值：包含所有 field 的列表，key 不存在返回空列表

例 1：查看 website 所有的域名称



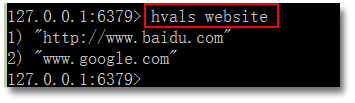
* hvals

语法：hvals key

作用：返回哈希表 中所有域的值

返回值：包含哈希表所有域值的列表，key 不存在返回空列表

例 1：显示 website 哈希表所有域的值

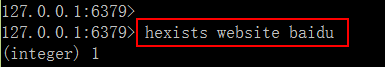


* hexists

语法：hexists key field

作用：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

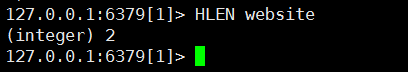
例 1：查看存在 key 中 field 域是否存在



* hlen

语法：hlen key

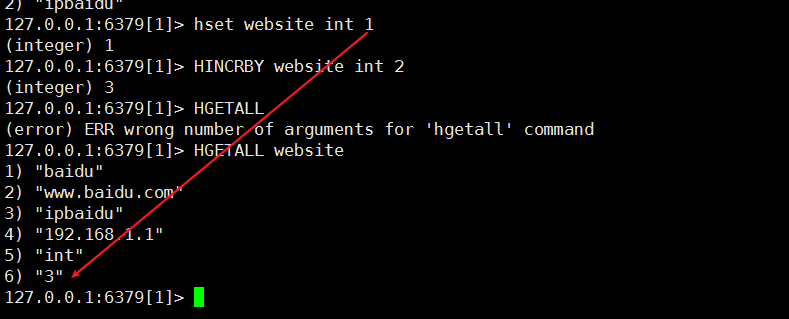
获取指定hash表中的field个数



* hincrby

hincrby key field int

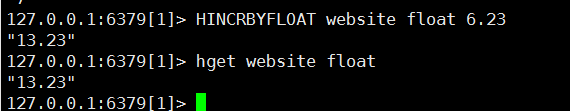
对指定hash表中指定field值进行加法运算。



* hincrbyfloat

语法：hincrbyfloat key field float

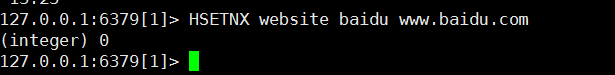
对指定hash表中指定的field值进行浮点数加法运算。



* hsetnx

语法：hsetnx key field value

将一个field-value对设置到hash表中，当key-field已经存在时放弃设置；



Redis中zset操作命令

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。

分数可以重复

* zadd

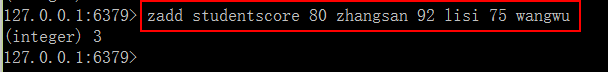
语法：zadd key score member [score member…]

作用：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

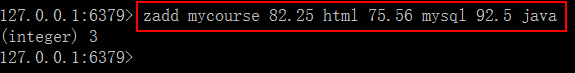
存在集合中，则更新值；score 可以是整数或浮点数

返回值：数字，新添加的元素个数

例 1：创建保存学生成绩的集合



例 2：使用浮点数作为 score



* zrange

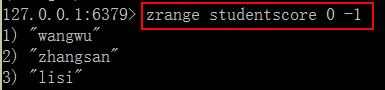
语法：zrange key start stop [WITHSCORES]

作用：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。 start，

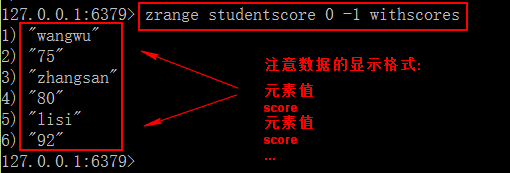
stop 都是从 0 开始。0 是第一个元素，1 是第二个元素，依次类推。以 -1 表示最后一个成员，-2 表示倒数第二个成员。WITHSCORES 选项让 score 和 value 一同返回。

返回值：自定区间的成员集合

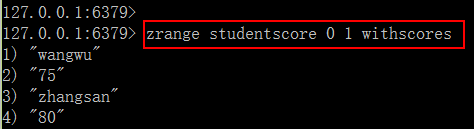
例 1：显示集合的全部元素，不显示 score，不使用 WITHSCORES



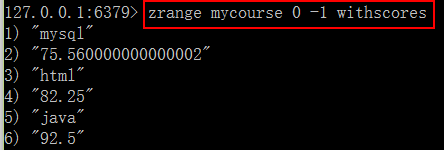
例 2：显示集合全部元素，并使用 WITHSCORES



例 3：显示第 0,1 二个成员



例 4：排序显示浮点数的 score



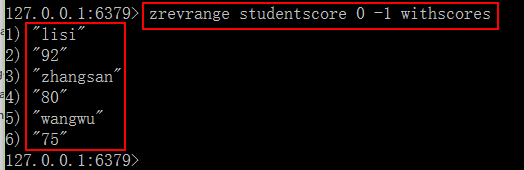
* zrevrange

语法：zrevrange key start stop [WITHSCORES]

作用：返回有序集 key 中，指定区间内的成员。其中成员的位置按 score 值递减(从大到小) 来排列。其它同 zrange 命令。

返回值：自定区间的成员集合

例 1：成绩榜



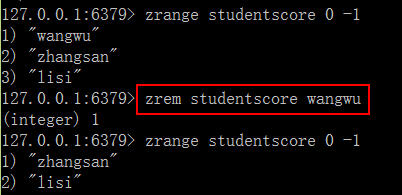
* zrem

语法：zrem key member [member…]

作用：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。

例 1：删除指定一个成员 wangwu



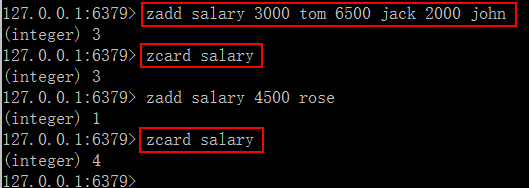
* zcard

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

例 1：查询集合的元素个数



* zrangebyscore

语法：zrangebyscore key min max [WITHSCORES ] [LIMIT offset count]

作用：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。

min ,max 是包括在内 ， 使用符号 ( 表示不包括。 min ， max 可以使用 -inf

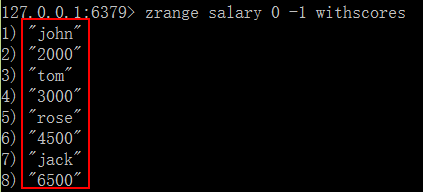
+inf 表示最小和最大

limit 用来限制返回结果的数量和区间。

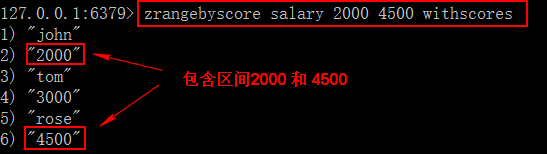
withscores 显 示 score 和 value

返回值：指定区间的集合数据

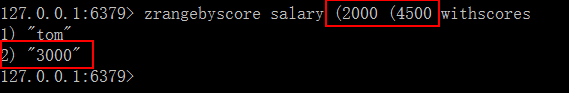
使用的准备数据



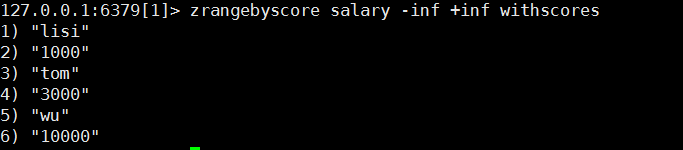
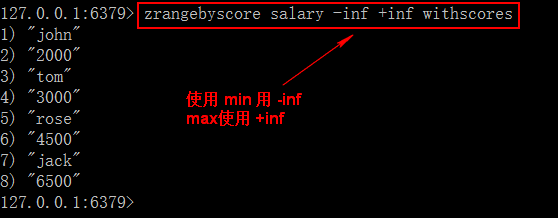
例 1：显示指定具体区间的数据

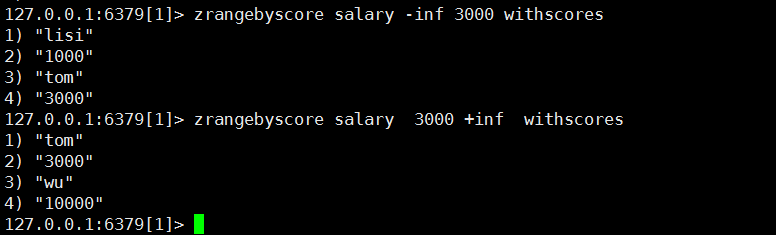


例 2：显示指定具体区间的集合数据，开区间（不包括 min，max）



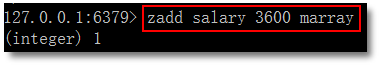
例 3：显示整个集合的所有数据

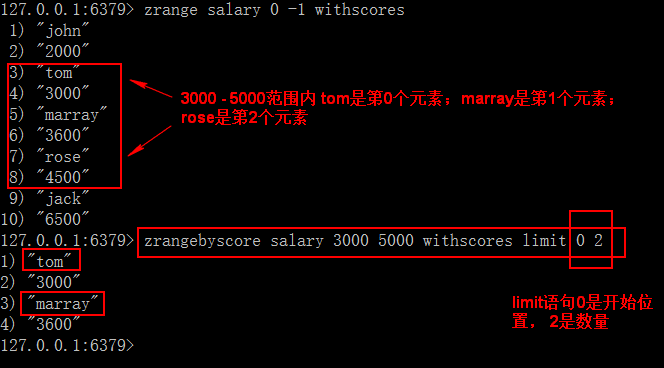




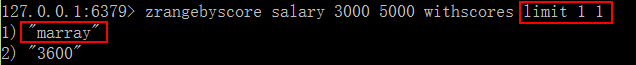
例 4：使用 limit

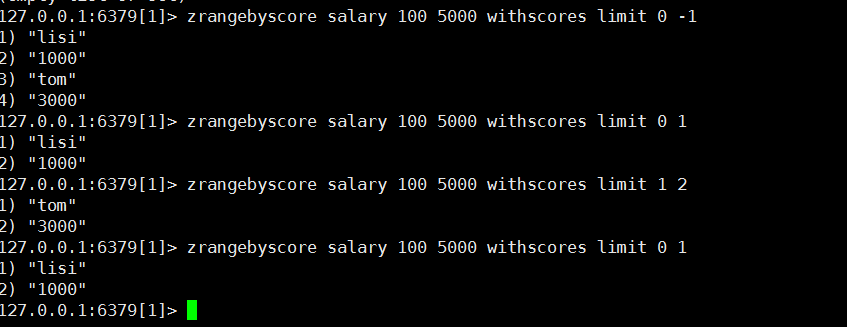
增加新的数据：





显示从第一个位置开始，取一个元素。



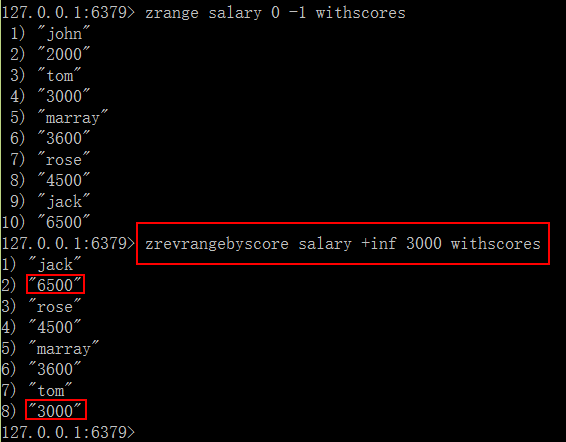


* zrevrangebyscore

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

作用：返回有序集 key 中， score 值介于 max 和 min 之间(默认包括等于 max 或 min )的所有的成员。有序集成员按 score 值递减(从大到小)的次序排列。其他同 zrangebyscore

例 1：查询工资最高到 3000 之间的员工

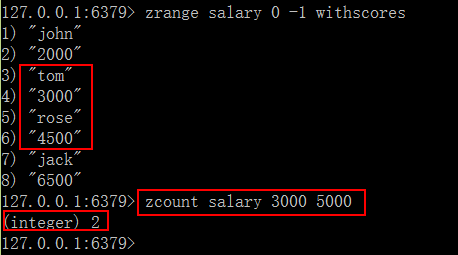


* zcount

语法：zcount key min max

作用：返回有序集 key 中， score 值在 min 和 max 之间(默认包括 score 值等于 min 或 max )的成员的数量

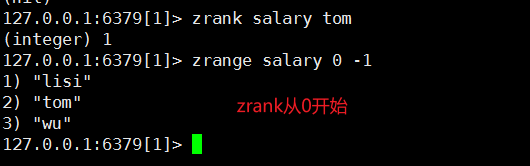
例 1：求工资在 3000-5000 的员工数量



* zrank

zrank key member

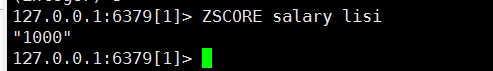
在获取指定有序集合中指定元素的排名（排名从0开始）



* zscore

zscore key member

获取指定有序集合中指定元素的分数



Redis配置文件

* redis.conf配置文件位置

在redis根目录下提供redis.conf配置文件，可以配置redis服务端运行时的参数，

如果不使用配置文件，那么redis会按照默认的参数执行。

如果使用配置文件，在启动redis服务时必须指定所使用的配置文件。

* redis配置文件中关于网络的配置

################################## NETWORK #####################################

# By default, if no "bind" configuration directive is specified, Redis listens

# for connections from all the network interfaces available on the server.

# It is possible to listen to just one or multiple selected interfaces using

# the "bind" configuration directive, followed by one or more IP addresses.

#

# Examples:

#

# bind 192.168.1.100 10.0.0.1

# bind 127.0.0.1 ::1

#

# ~~~ WARNING ~~~ If the computer running Redis is directly exposed to the

# internet, binding to all the interfaces is dangerous and will expose the

# instance to everybody on the internet. So by default we uncomment the

# following bind directive, that will force Redis to listen only into

# the IPv4 loopback interface address (this means Redis will be able to

# accept connections only from clients running into the same computer it

# is running).

#

# IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO ALL THE INTERFACES

# JUST COMMENT THE FOLLOWING LINE.

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

bind 127.0.0.1

# Protected mode is a layer of security protection, in order to avoid that

# Redis instances left open on the internet are accessed and exploited.

#

# When protected mode is on and if:

#

# 1) The server is not binding explicitly to a set of addresses using the

# "bind" directive.

# 2) No password is configured.

#

# The server only accepts connections from clients connecting from the

# IPv4 and IPv6 loopback addresses 127.0.0.1 and ::1, and from Unix domain

# sockets.

#

# By default protected mode is enabled. You should disable it only if

# you are sure you want clients from other hosts to connect to Redis

# even if no authentication is configured, nor a specific set of interfaces

# are explicitly listed using the "bind" directive.

protected-mode yes

# Accept connections on the specified port, default is 6379 (IANA #815344).

# If port 0 is specified Redis will not listen on a TCP socket.

port 6379

# TCP listen() backlog.

#

# In high requests-per-second environments you need an high backlog in order

# to avoid slow clients connections issues. Note that the Linux kernel

# will silently truncate it to the value of /proc/sys/net/core/somaxconn so

# make sure to raise both the value of somaxconn and tcp\_max\_syn\_backlog

# in order to get the desired effect.

tcp-backlog 511

# Unix socket.

#

# Specify the path for the Unix socket that will be used to listen for

# incoming connections. There is no default, so Redis will not listen

# on a unix socket when not specified.

#

# unixsocket /tmp/redis.sock

# unixsocketperm 700

# Close the connection after a client is idle for N seconds (0 to disable)

timeout 0

# TCP keepalive.

#

# If non-zero, use SO\_KEEPALIVE to send TCP ACKs to clients in absence

# of communication. This is useful for two reasons:

#

# 1) Detect dead peers.

# 2) Take the connection alive from the point of view of network

# equipment in the middle.

#

# On Linux, the specified value (in seconds) is the period used to send ACKs.

# Note that to close the connection the double of the time is needed.

# On other kernels the period depends on the kernel configuration.

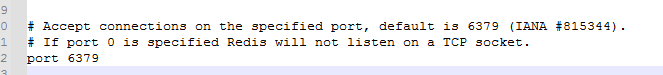
#

# A reasonable value for this option is 300 seconds, which is the new

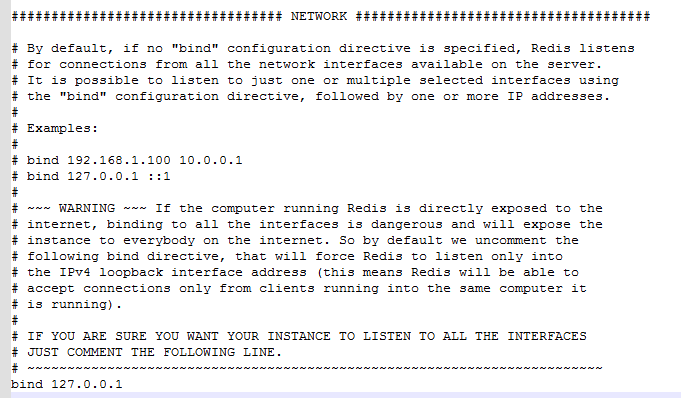
# Redis default starting with Redis 3.2.1.

tcp-keepalive 300

port：指定redis服务所使用的端口，默认使用6379。



bind：配置客户端连接redis服务时，所能使用的ip地址，默认可以使用redis服务所在主机上任何一个ip都可以，一般情况下，都会配置1个ip，



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

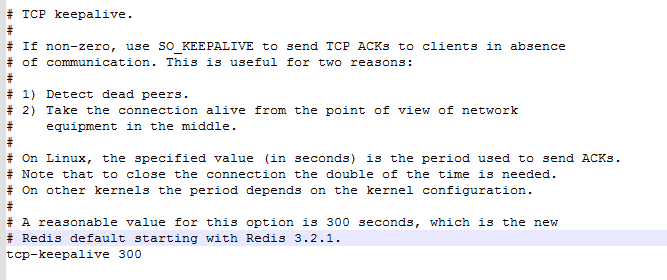
redis-server redis.conf & 后台启动自定义配置文件的redis服务。

redis-cli -h 192.168.11.128 -p 6380 启动客户端指定ip 和端口号。

redis-cli -h 192.168.11.128 -p 6380 shutdown 关闭时也要指定ip和端口号。

ps -ef|grep redis 查看redis服务状态

tcp-keepalive：TCP连接保活策略，通过tcp-keepalive配置设置，单位秒，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，用来检查客户端是否已经挂掉，对于无响应的客户端会关闭连接。如果设置为0，则不会进行保活检测。



* redis常规配置

################################# GENERAL #####################################

# By default Redis does not run as a daemon. Use 'yes' if you need it.

# Note that Redis will write a pid file in /var/run/redis.pid when daemonized.

daemonize no

# If you run Redis from upstart or systemd, Redis can interact with your

# supervision tree. Options:

# supervised no - no supervision interaction

# supervised upstart - signal upstart by putting Redis into SIGSTOP mode

# supervised systemd - signal systemd by writing READY=1 to $NOTIFY\_SOCKET

# supervised auto - detect upstart or systemd method based on

# UPSTART\_JOB or NOTIFY\_SOCKET environment variables

# Note: these supervision methods only signal "process is ready."

# They do not enable continuous liveness pings back to your supervisor.

supervised no

# If a pid file is specified, Redis writes it where specified at startup

# and removes it at exit.

#

# When the server runs non daemonized, no pid file is created if none is

# specified in the configuration. When the server is daemonized, the pid file

# is used even if not specified, defaulting to "/var/run/redis.pid".

#

# Creating a pid file is best effort: if Redis is not able to create it

# nothing bad happens, the server will start and run normally.

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 debug level)

# notice (moderately verbose, what you want in production probably)

# warning (only very important / critical messages are logged)

loglevel notice

# Specify the log file name. Also the empty string can be used to force

# Redis to log on the standard output. Note that if you use standard

# output for logging but daemonize, logs will be sent to /dev/null

logfile ""

# To enable logging to the system logger, just set 'syslog-enabled' to yes,

# and optionally update the other syslog parameters to suit your needs.

# syslog-enabled no

# Specify the syslog identity.

# syslog-ident redis

# Specify the syslog facility. Must be USER or between LOCAL0-LOCAL7.

# syslog-facility local0

# Set the number of databases. The default database is DB 0, you can select

# a different one on a per-connection basis using SELECT <dbid> where

# dbid is a number between 0 and 'databases'-1

databases 16

# By default Redis shows an ASCII art logo only when started to log to the

# standard output and if the standard output is a TTY. Basically this means

# that normally a logo is displayed only in interactive sessions.

#

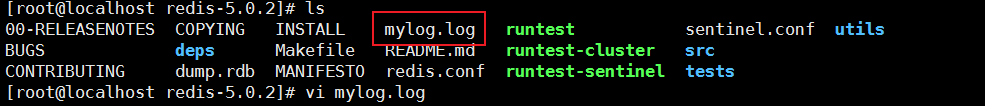
# However it is possible to force the pre-4.0 behavior and always show a

# ASCII art logo in startup logs by setting the following option to yes.

always-show-logo yes

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：指定日志文件名，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。还要在redis启动时指定所使用的配置文件，否则不起作用。



databases：配置redis数据库的个数，默认是16个。

* redis安全配置

################################## SECURITY ###################################

# Require clients to issue AUTH <PASSWORD> before processing any other

# commands. This might be useful in environments in which you do not trust

# others with access to the host running redis-server.

#

# This should stay commented out for backward compatibility and because most

# people do not need auth (e.g. they run their own servers).

#

# Warning: since Redis is pretty fast an outside user can try up to

# 150k passwords per second against a good box. This means that you should

# use a very strong password otherwise it will be very easy to break.

#

# requirepass foobared

# Command renaming.

#

# It is possible to change the name of dangerous commands in a shared

# environment. For instance the CONFIG command may be renamed into something

# hard to guess so that it will still be available for internal-use tools

# but not available for general clients.

#

# Example:

#

# rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52

#

# It is also possible to completely kill a command by renaming it into

# an empty string:

#

# rename-command CONFIG ""

#

# Please note that changing the name of commands that are logged into the

# AOF file or transmitted to replicas may cause problems.

requirepass ：配置redis的访问密码，默认不配置密码，即访问不需要密码验证，此配置项需要在protected-mode=yes时起作用。使用密码登录客户端： redis-cli -h ip -p 6379 -a pwd

* redis的RDB配置（持久化）
* redis的AOF配置（持久化）

Redis持久化

持久化可以理解为存储，就是将数据存储到一个不会丢失的地方，如果把数据放在内存中，电脑关闭或重启数据就会丢失，所以放在内存中的数据不是持久化的，而放在磁盘就算是一种持久化。

Redis 的数据存储在内存中，内存是瞬时的，如果 linux 宕机或重启，又或者 Redis 崩溃或重启，所有的内存数据都会丢失，为解决这个问题，Redis 提供两种机制对数据进行持久化存储，便于发生故障后能迅速恢复数据。

持久化方式

● RDB 方式

################################ SNAPSHOTTING ################################

#

# Save the DB on disk:

#

# save <seconds> <changes>

#

# Will save the DB if both the given number of seconds and the given

# number of write operations against the DB occurred.

#

# In the example below the behaviour will be to save:

# after 900 sec (15 min) if at least 1 key changed

# after 300 sec (5 min) if at least 10 keys changed

# after 60 sec if at least 10000 keys changed

#

# Note: you can disable saving completely by commenting out all "save" lines.

#

# It is also possible to remove all the previously configured save

# points by adding a save directive with a single empty string argument

# like in the following example:

#

# save ""

save 900 1

save 300 10

save 60 10000

# By default Redis will stop accepting writes if RDB snapshots are enabled

# (at least one save point) and the latest background save failed.

# This will make the user aware (in a hard way) that data is not persisting

# on disk properly, otherwise chances are that no one will notice and some

# disaster will happen.

#

# If the background saving process will start working again Redis will

# automatically allow writes again.

#

# However if you have setup your proper monitoring of the Redis server

# and persistence, you may want to disable this feature so that Redis will

# continue to work as usual even if there are problems with disk,

# permissions, and so forth.

stop-writes-on-bgsave-error yes

# Compress string objects using LZF when dump .rdb databases?

# For default that's set to 'yes' as it's almost always a win.

# If you want to save some CPU in the saving child set it to 'no' but

# the dataset will likely be bigger if you have compressible values or keys.

rdbcompression yes

# Since version 5 of RDB a CRC64 checksum is placed at the end of the file.

# This makes the format more resistant to corruption but there is a performance

# hit to pay (around 10%) when saving and loading RDB files, so you can disable it

# for maximum performances.

#

# RDB files created with checksum disabled have a checksum of zero that will

# tell the loading code to skip the check.

rdbchecksum yes

# The filename where to dump the DB

dbfilename dump.rdb

# The working directory.

#

# The DB will be written inside this directory, with the filename specified

# above using the 'dbfilename' configuration directive.

#

# The Append Only File will also be created inside this directory.

#

# Note that you must specify a directory here, not a file name.

dir ./

A、什么是 RDB 方式?

Redis  Database（RDB），就是在指定的时间间隔内将内存中的数据集快照写入磁盘，数据恢复时将快照文件直接再读到内存。

RDB 保存了在某个时间点的数据集（全部数据）。存储在一个二进制文件中，只有一个文件。默认是 dump.rdb。RDB 技术非常适合做备份，可以保存最近一个小时，一天，一个月的全部数据。保存数据是在单独的进程中写文件，不影响 Redis 的正常使用。RDB 恢复数据时比其他 AOF 速度快。

B、 如何实现?

RDB 方式的数据持久化，仅需在 redis.conf 文件中配置即可，默认配置是启用的。

在配置文件 redis.conf 中搜索 SNAPSHOTTING， 查找在注释开始和结束之间的关于 RDB的配置说明。配 SNAPSHOTTING 置地方有三处。

①：配置执行 RDB 生成快照文件的时间策略。

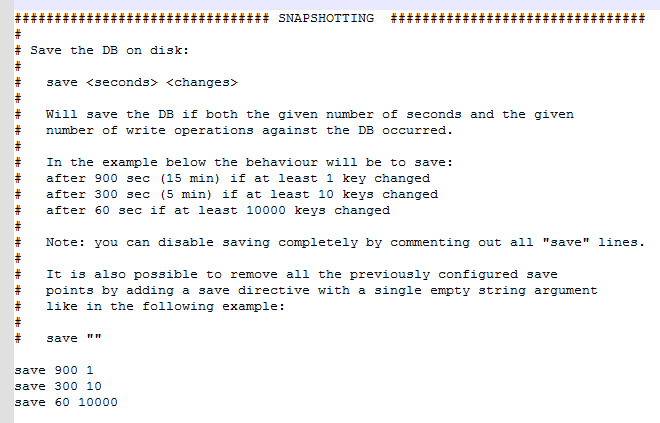
对 Redis 进行设置， 让它在“ N 秒内数据集至少有 M 个 key 改动”这一条件被满足时， 自动保存一次数据集。

配置格式：save <seconds> <changes>

save 900 1

save 300 10

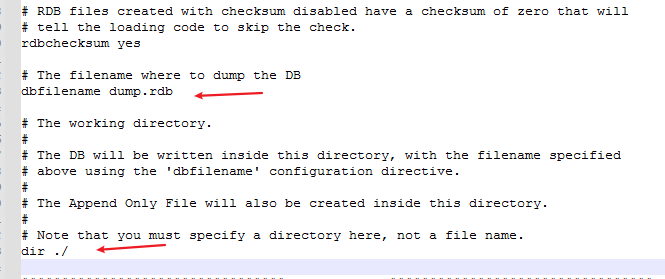
save 60 10000

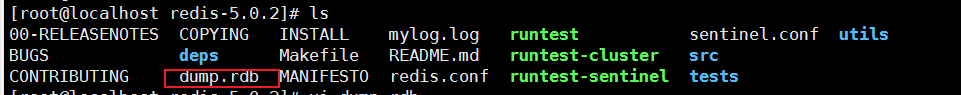


②：dbfilename：设置 RDB 的文件名，默认文件名为 dump.rdb



③：dir：指定 RDB 文件的存储位置，默认是 ./ 当前目录



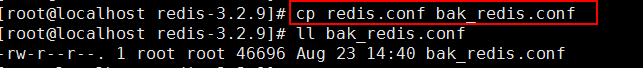


配置步骤：

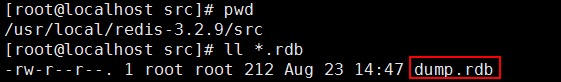
①：查看 ps -ef | grep redis ，如果 redis 服务启动，先停止。

IMG_256

②：修改 redis.conf 文件， 修改前先备份，执行 cp redis.conf bak\_redis.conf



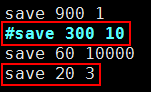
查看默认启用的 RDB 文件



③：编辑 redis.conf 增加 save 配置， 修改文件名等。vim redis.conf



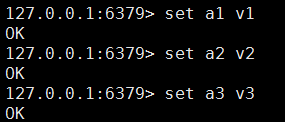
修改的内容：



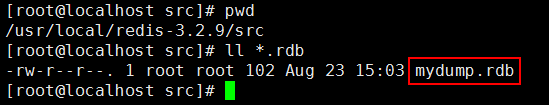
把原来的默认的 dump.rdb 删除，修改 redis.conf 后，重新启动 redis

IMG_256

④：在 20 秒内容，修改三个 key 的值



⑤：查看生成的 rdb 文件



C、 总结

优点：

由于存储的是数据快照文件，恢复数据很方便，也比较快

缺点：

1）会丢失最后一次快照以后更改的数据。如果你的应用能容忍一定数据的丢失，那么使用 rdb 是不错的选择；如果你不能容忍一定数据的丢失，使用 rdb 就不是一个很好的选择。

2）由于需要经常操作磁盘，RDB 会分出一个子进程。如果你的 redis 数据库很大的话， 子进程占用比较多的时间，并且可能会影响 Redis  暂停服务一段时间（millisecond  级别），如果你的数据库超级大并且你的服务器 CPU 比较弱，有可能是会达到一秒。

● AOF 方式

############################## APPEND ONLY MODE ###############################

# By default Redis asynchronously dumps the dataset on disk. This mode is

# good enough in many applications, but an issue with the Redis process or

# a power outage may result into a few minutes of writes lost (depending on

# the configured save points).

#

# The Append Only File is an alternative persistence mode that provides

# much better durability. For instance using the default data fsync policy

# (see later in the config file) Redis can lose just one second of writes in a

# dramatic event like a server power outage, or a single write if something

# wrong with the Redis process itself happens, but the operating system is

# still running correctly.

#

# AOF and RDB persistence can be enabled at the same time without problems.

# If the AOF is enabled on startup Redis will load the AOF, that is the file

# with the better durability guarantees.

#

# Please check http://redis.io/topics/persistence for more information.

appendonly no

# The name of the append only file (default: "appendonly.aof")

appendfilename "appendonly.aof"

# The fsync() call tells the Operating System to actually write data on disk

# instead of waiting for more data in the output buffer. Some OS will really flush

# data on disk, some other OS will just try to do it ASAP.

#

# Redis supports three different modes:

#

# no: don't fsync, just let the OS flush the data when it wants. Faster.

# always: fsync after every write to the append only log. Slow, Safest.

# everysec: fsync only one time every second. Compromise.

#

# The default is "everysec", as that's usually the right compromise between

# speed and data safety. It's up to you to understand if you can relax this to

# "no" that will let the operating system flush the output buffer when

# it wants, for better performances (but if you can live with the idea of

# some data loss consider the default persistence mode that's snapshotting),

# or on the contrary, use "always" that's very slow but a bit safer than

# everysec.

#

# More details please check the following article:

# http://antirez.com/post/redis-persistence-demystified.html

#

# If unsure, use "everysec".

# appendfsync always

appendfsync everysec

# appendfsync no

# When the AOF fsync policy is set to always or everysec, and a background

# saving process (a background save or AOF log background rewriting) is

# performing a lot of I/O against the disk, in some Linux configurations

# Redis may block too long on the fsync() call. Note that there is no fix for

# this currently, as even performing fsync in a different thread will block

# our synchronous write(2) call.

#

# In order to mitigate this problem it's possible to use the following option

# that will prevent fsync() from being called in the main process while a

# BGSAVE or BGREWRITEAOF is in progress.

#

# This means that while another child is saving, the durability of Redis is

# the same as "appendfsync none". In practical terms, this means that it is

# possible to lose up to 30 seconds of log in the worst scenario (with the

# default Linux settings).

#

# If you have latency problems turn this to "yes". Otherwise leave it as

# "no" that is the safest pick from the point of view of durability.

no-appendfsync-on-rewrite no

# Automatic rewrite of the append only file.

# Redis is able to automatically rewrite the log file implicitly calling

# BGREWRITEAOF when the AOF log size grows by the specified percentage.

#

# This is how it works: Redis remembers the size of the AOF file after the

# latest rewrite (if no rewrite has happened since the restart, the size of

# the AOF at startup is used).

#

# This base size is compared to the current size. If the current size is

# bigger than the specified percentage, the rewrite is triggered. Also

# you need to specify a minimal size for the AOF file to be rewritten, this

# is useful to avoid rewriting the AOF file even if the percentage increase

# is reached but it is still pretty small.

#

# Specify a percentage of zero in order to disable the automatic AOF

# rewrite feature.

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

# An AOF file may be found to be truncated at the end during the Redis

# startup process, when the AOF data gets loaded back into memory.

# This may happen when the system where Redis is running

# crashes, especially when an ext4 filesystem is mounted without the

# data=ordered option (however this can't happen when Redis itself

# crashes or aborts but the operating system still works correctly).

#

# Redis can either exit with an error when this happens, or load as much

# data as possible (the default now) and start if the AOF file is found

# to be truncated at the end. The following option controls this behavior.

#

# If aof-load-truncated is set to yes, a truncated AOF file is loaded and

# the Redis server starts emitting a log to inform the user of the event.

# Otherwise if the option is set to no, the server aborts with an error

# and refuses to start. When the option is set to no, the user requires

# to fix the AOF file using the "redis-check-aof" utility before to restart

# the server.

#

# Note that if the AOF file will be found to be corrupted in the middle

# the server will still exit with an error. This option only applies when

# Redis will try to read more data from the AOF file but not enough bytes

# will be found.

aof-load-truncated yes

# When rewriting the AOF file, Redis is able to use an RDB preamble in the

# AOF file for faster rewrites and recoveries. When this option is turned

# on the rewritten AOF file is composed of two different stanzas:

#

# [RDB file][AOF tail]

#

# When loading Redis recognizes that the AOF file starts with the "REDIS"

# string and loads the prefixed RDB file, and continues loading the AOF

# tail.

aof-use-rdb-preamble yes

A、什么是 AOF 方式

Append-only File（AOF），Redis 每次接收到一条改变数据的命令时，它将把该命令写到一个 AOF 文件中（只记录写操作，读操作不记录），当 Redis 重启时，它通过执行 AOF 文件中所有的命令来恢复数据。

B、 如何实现

AOF 方式的数据持久化，仅需在 redis.conf 文件中配置即可配置项：

①：appendonly：默认是 no，改成 yes 即开启了 aof 持久化

②：appendfilename：指定 AOF 文件名，默认文件名为 appendonly.aof

③：dir : 指定 RDB 和 AOF 文件存放的目录，默认是 ./

④：appendfsync：配置向 aof 文件写命令数据的策略：

no：不主动进行同步操作，而是完全交由操作系统来做（即每 30 秒一次），比较快但不是很安全。

always：每次执行写入都会执行同步，慢一些但是比较安全。

everysec：每秒执行一次同步操作，比较平衡，介于速度和安全之间。这是默认项。

⑤：auto-aof-rewrite-min-size：允许重写的最小 AOF 文件大小，默认是 64M 。当 aof 文件大于 64M 时，开始整理 aop 文件， 去掉无用的操作命令。缩小 aop 文件。

例 1：

①：停止运行的 redis ， 备份要修改的 redis.conf

shutdown 关闭redis服务

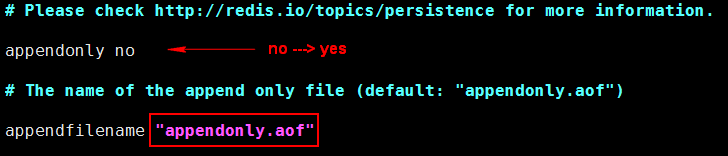
②：查看 redis 安装目录/src 下有无 .aof 文件。 默认是在 redis 的当前目录

IMG_256

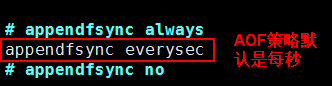
③：编辑 redis.conf

设置 appendonly 为 yes 即可。

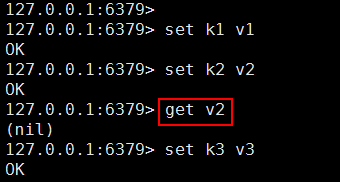
查看 appendfsync 的当前策略。



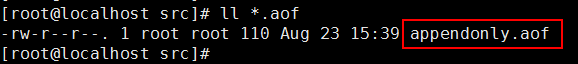
查看 appendfilname 的文件名称

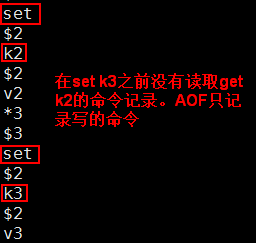


④：在 redis 客户端执行 写入命令



⑤ 查看 aof 文件





● 总结

append-only 文件是另一个可以提供完全数据保障的方案；

2.AOF 文件会在操作过程中变得越来越大。比如，如果你做一百次加法计算，最后你只会在数据库里面得到最终的数值，但是在你的 AOF 里面会存在 100 次记录，其中 99 条记录对最终的结果是无用的；但 Redis 支持在不影响服务的前提下在后台重构 AOF 文件，让文件得以整理变小

3.可以同时使用这两种方式，redis 默认优先加载 aof 文件（aof 数据最完整）；

Redis事务

事务是指一系列操作步骤，这一系列的操作步骤，要么完全地执行，要么完全地不执行。保证操作的原子性。

Redis  中的事务（transaction）是一组命令的集合，至少是两个或两个以上的命令，redis 事务保证这些命令被执行时中间不会被任何其他操作打断。

* 事务操作的命令

● multi

语法： multi

作用：标记一个事务的开始。事务内的多条命令会按照先后顺序被放进一个队列当中。

返回值：总是返回 ok

● exec

语法：exec

作用：执行所有事务块内的命令（执行事务队列的所有命令）

返回值：事务内的所有执行语句内容，事务被打断（影响）返回 nil

redis事务只能保证部分原子性：

【1】如果有一组命令在压入队列时发生错误的命令，则事务本身所有的命令都不执行，能够保证事务的原子性。

multi

set k1 v1

seta k2 v2 ----> 错误命令

set k3 v3

exec

报错！

【2】如果一组命令中，在压入队列过程中正常，但是在执行事务队列时发生了错误，则只会影响发生错误的命令，不会影响其他命令的执行，不能够保证事务的原子性。

multi

set k1 v1

incr k2 --->执行过程中报错

set k3 v3

exec

正常执行。

● discard

语法：discard

作用：取消事务，放弃执行事务块内的所有命令返回值：总是返回 ok

●  watch

语法：watch key [key ...]

作用：监视一个(或多个) key ，如果在事务执行之前这个(或这些) key 被其他命令所改动， 那么事务将被打断。

返回值：总是返回 ok



●  unwatch

语法：unwatch

作用：取消 WATCH 命令对所有 key 的监视。如果在执行 WATCH 命令之后， EXEC 命令或 DISCARD 命令先被执行了的话，那么就不需要再执行 UNWATCH 了

返回值：总是返回 ok

* 事务的实现

● 正常执行事务

事务的执行步骤： 首先开启事务， 其次向事务队列中加入命令，最后执行事务提交

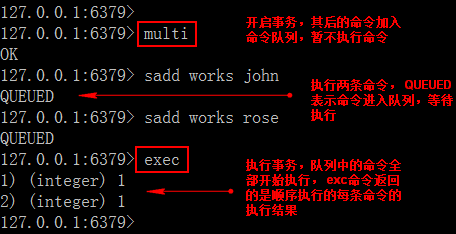
例 1：事务的执行:

1、multi ： 用 multi 命令告诉 Redis，接下来要执行的命令你先不要执行，而是把它们暂时存起来 （开启事务）

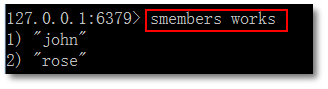
2、sadd works john 第一条命令进入等待队列（命令入队）

3、sadd works rose 第二条命令进入等待队列（命令入队）

4、exce 告知 redis 执行前面发送的两条命令（提交事务）



查看 works 集合



● 事务执行 exec 之前，入队命令错误（语法错误；严重错误导致服务器不能正常工作（例如内存不足）），放弃事务。

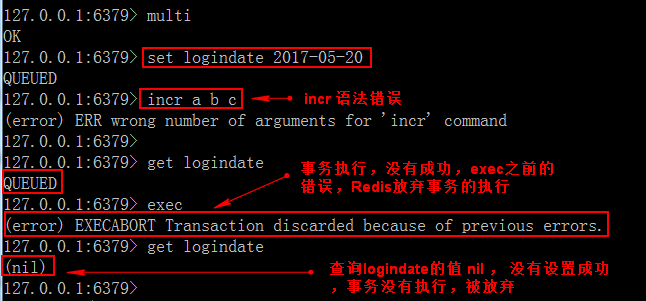
执行事务步骤：

1、MULTI 正常命令

2、SET key value 正常命令

3、INCR 命令语法错误

4、EXEC 无法执行事务，那么第一条正确的命令也不会执行，所以 key 的值不会设置成功



结论：事务执行 exec 之前，入队命令错误，事务终止，取消，不执行。

● 事务执行 exec 命令后，执行队列命令，命令执行错误，事务提交

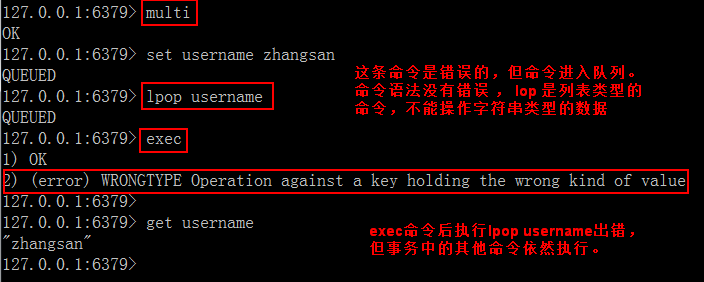
执行步骤：

1、MULTI 正常命令

2、SET username zhangsan 正常命令

3、lpop username 正常命令，语法没有错误，执行命令时才会有错误

4、EXEC 正常执行 ,发现错误可以在事务提交前放弃事务，执行 discard



结论：在 exec 执行后的所产生的错误， 即使事务中有某个/某些命令在执行时产生了错误，事务中的其他命令仍然会继续执行。

Redis 在事务失败时不进行回滚，而是继续执行余下的命令。

Redis 这种设计原则是：Redis 命令只会因为错误的语法而失败（这些问题不能在入队时发现），或是命令用在了错误类型的键上面，失败的命令并不是 Redis 导致，而是由编程错误造成的，这样错误应该在开发的过程中被发现，生产环境中不应出现语法的错误。就是在程序的运行环境中不应该出现语法的错误。而 Redis 能够保证正确的命令一定会被执行。再者不需要对回滚进行支持，所以 Redis 的内部可以保持简单且快速。

● 放弃事务

执行步骤：

1.MULTI 开启事务

2.SET age 25 命令入队

3.SET age 30 命令入队

4.DISCARD 放弃事务，则命令队列不会被执行

例 1：



● Redis 的 watch 机制

1、Redis 的 WATCH 机制

WATCH 机制原理：

WATCH 机制：使用 WATCH 监视一个或多个 key , 跟踪 key 的 value 修改情况， 如果有key 的 value 值在事务 EXEC 执行之前被修改了， 整个事务被取消。EXEC返回提示信息，表示事务已经失败

WATCH 机制使的事务 EXEC 变的有条件，事务只有在被 WATCH 的 key 没有修改的前提下才能执行。不满足条件，事务被取消。使用 WATCH 监视了一个带过期时间的键， 那么即使这个键过期了， 事务仍然可以正常执行

大多数情况下， 不同的客户端会访问不同的键， 相互同时竞争同一 key 的情况一般都很少， 乐观锁能够以很好的性能解决数据冲突的问题

2、何时取消 key 的监视（WATCH）

① WATCH 命令可以被调用多次。 对键的监视从 WATCH 执行之后开始生效， 直到调用 EXEC 为止。不管事务是否成功执行， 对所有键的监视都会被取消。

② 当客户端断开连接时， 该客户端对键的监视也会被取消。

③ UNWATCH 命令可以手动取消对所有键的监视

3、 WATCH 的事例

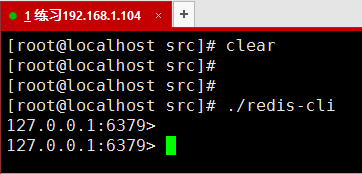
执行步骤：

首先启动 redis-server , 在开启两个客户端连接。 分别叫 A 客户端 和 B 客户端。

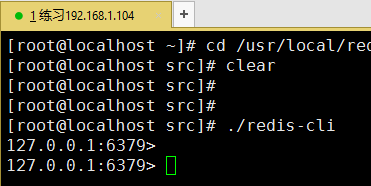
启动 Redis 服务器



A 客户端（红色）：WATCH 某个 key  ，同时执行事务



B  客户端（黄色）：对 A 客户端 WATCH 的 key 修改其 value 值。



（1）在 A 客户端设置 key : str.lp 登录人数为 10

（2）在 A 客户端监视 key : str.lp

（3）在 A 客户端开启事务 multi

（4）在 A 客户端修改 str.lp 的值为 11

（5）在 B 客户端修改 str.lp 的值为 15

（6）在 A 客户端执行事务 exec

（7）在 A 客户端查看 str.lp 值，A 客户端执行的事务没有提交，因为 WATCH 的 str.lp 的值已经被修改了， 所有放弃事务。

例 1：乐观锁



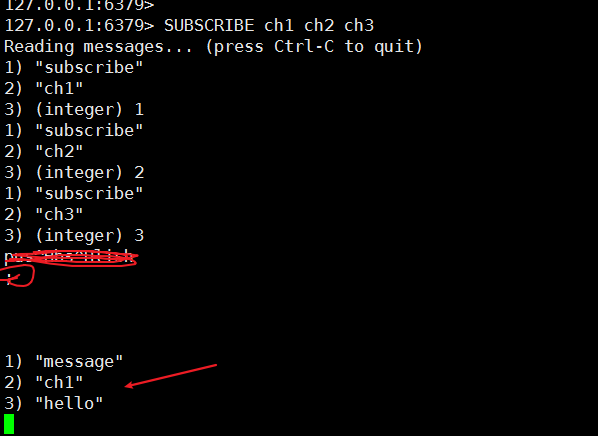
Redis消息的发布和订阅（了解）

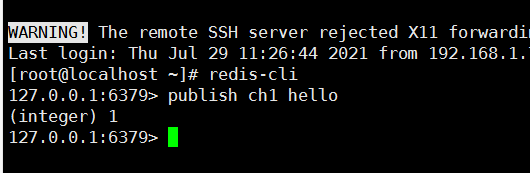
redis客户端订阅频道，消息的发布者往频道上发布消息，所有订阅次频道的客户端都能接收到消息。

【1】subscribe：订阅一个或者多个频道的消息。

subscribe ch1 ch2 ch3

【2】publish ：将消息发布到指定频道





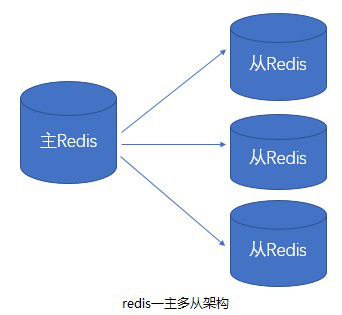
【3】psubscribe：订阅一个或者多个频道

Redis的主从复制

通过持久化功能，Redis 保证了即使在服务器重启的情况下也不会丢失（或少量丢失）数据，但是由于数据是存储在一台服务器上的，如果这台服务器出现故障，比如硬盘坏了， 也会导致数据丢失。

为了避免单点故障，我们需要将数据复制多份部署在多台不同的服务器上，即使有一台服务器出现故障其他服务器依然可以继续提供服务。

这就要求当一台服务器上的数据更新后，自动将更新的数据同步到其他服务器上，那该怎么实现呢？ Redis 的主从复制。



Redis 提供了复制（replication）功能来自动实现多台 redis 服务器的数据同步。

（每天19 点 新闻联播，基本从 cctv1-8,各大卫视都会播放）

我们可以通过部署多台 redis，并在配置文件中指定这几台 redis 之间的主从关系，主负责写入数据， 同时把写入的数据实时同步到从机器， 这种模式叫做主从复制， 即master/slave，并且 redis 默认 master 用于写，slave 用于读，向 slave 写数据会导致错误。

* Redis 主从复制实现（master/salve）

方式 1：修改配置文件，启动时，服务器读取配置文件，并自动成为指定服务器的从服务器，从而构成主从复制的关系

方式 2： ./redis-server --slaveof <master-ip> <master-port>，在启动 redis 时指定当前服务成为某个主 Redis 服务的从 Slave

方式 1 的实现步骤：

模拟多 Reids 服务器， 在一台已经安装 Redis 的机器上，运行多个 Redis 应用模拟多个 Reids 服务器。一个 Master，两个 Slave。

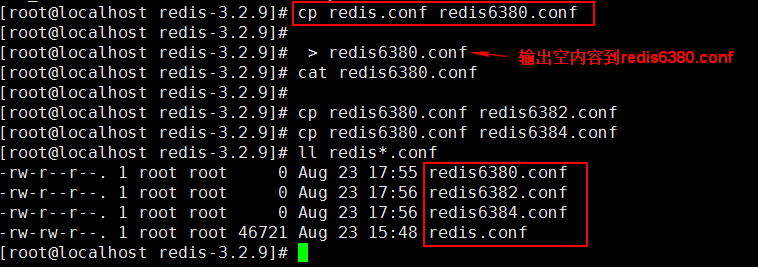
A、新建三个 Redis 的配置文件

如果 Redis 启动，先停止。redis-cli shutdown

作为 Master 的 Redis 端口是 6380

作为 Slaver 的 Redis 端口分别是 6382 , 6384

从原有的 redis.conf 拷贝三份，分别命名为 redis6380.conf, redis6382.conf , redis6384.conf



B、 编辑 Master 配置文件

编辑 Master 的配置文件 redis6380.conf : 在空文件加入如下内容

include /usr/local/redis-3.2.9/redis.conf

daemonize yes

port 6380

pidfile /var/run/redis\_6380.pid

logfile 6380.log

dbfilename dump6380.rdb

配置项说明：

include ： 包含原来的配置文件内容。/usr/local/redis-3.2.9/redis.conf 按照自己的目录设置。

daemonize：yes 后台启动应用，相当于 ./redis-server & , &的作用。

port : 自定义的端口号

pidfile : 自定义的文件，表示当前程序的 pid ,进程 id。

logfile：日志文件名

dbfilename：持久化的 rdb 文件名

C、 编辑 Slave 配置文件

编辑 Slave 的配置文件 redis6382.conf 和 redis6384.conf: 在空文件加入如下内容

①：redis6382.conf：

include /usr/local/redis-3.2.9/redis.conf

daemonize yes

port 6382

pidfile /var/run/redis\_6382.pid

logfile 6382.log

dbfilename dump6382.rdb

slaveof 127.0.0.1 6380

配置项说明：

slaveof ： 表示当前 Redis 是谁的从。当前是 127.0.0.0 端口 6380 这个 Master 的从。

②：redis6384.conf：

include /usr/local/redis-3.2.9/redis.conf daemonize yes

port 6384

pidfile /var/run/redis\_6384.pid logfile 6384.log

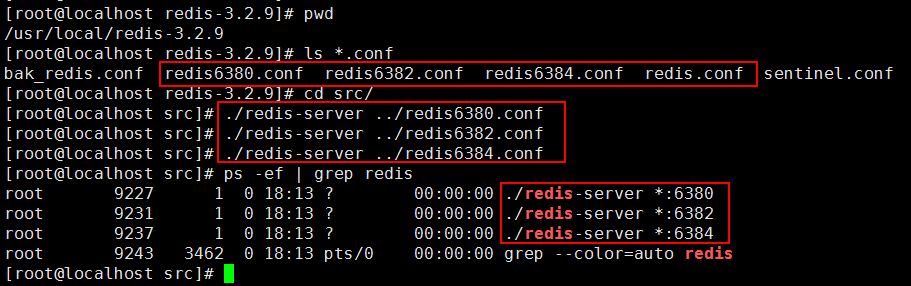
dbfilename dump6384.rdb

slaveof 127.0.0.1 6380

D、启动服务器 Master/Slave 都启动

启动方式 ./redis-server 配置文件

启动 Redis,并查看启动进程



E、 查看配置后的服务信息

命令：

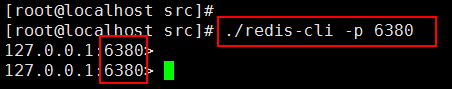
①： Redis 客户端使用指定端口连接 Redis 服务器

./redis-cli -p 端口

②：查看服务器信息

info replication

登录到 Master：6380



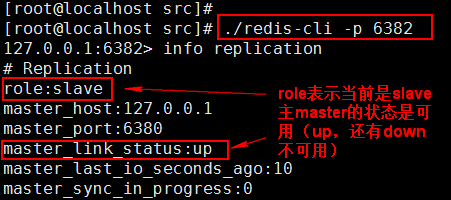
查看当前服务信息

在客户端的 Redis 内执行命令 info replication

Master 服务的查看结果:



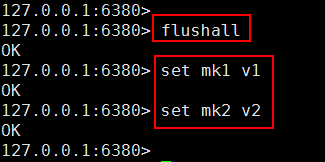
在新的 Xshell 窗口分别登录到 6382 ，6384 查看信息



6384 也登录内容同 6382

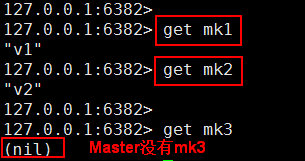
F、 向 Master 写入数据

在 6380 执行 flushall 清除数据，避免干扰的测试数据。 生产环境避免使用。



G、在从 Slave 读数据

6382,6384 都可以读主 Master 的数据，不能写



Slave 写数据失败

IMG_256

全量复制：一旦主从关系确定，会自动把主机上已有的同步复制到从库。

增量复制：主库写数据会自动同步到从库。

容灾处理

【1】主机宕机，从机待定。

master 上（冷处理：机器挂掉了，再处理）当 Master 服务出现故障，需手动将 slave 中的一个提升为 master， 剩下的 slave 挂至新的。

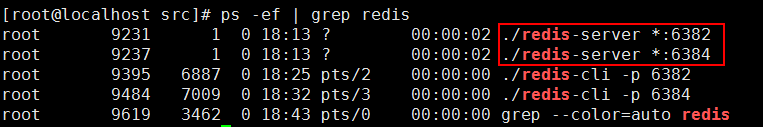
命令：

①：slaveof no one，将一台 slave 服务器提升为 Master （提升某 slave 为 master）

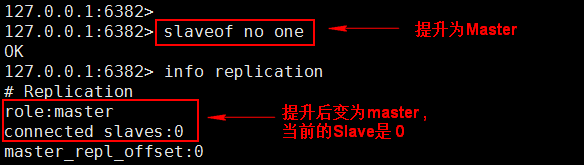
②：slaveof 127.0.0.1 6381 （将 slave 挂至新的 master 上）

执行步骤：

A、将 Master:6380 停止（模拟挂掉）

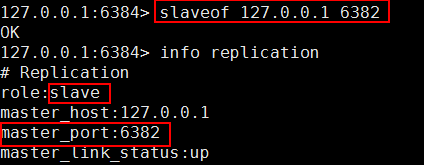


1. 选择一个 Slave 升到 Master，其它的 Slave 挂到新提升的 Master



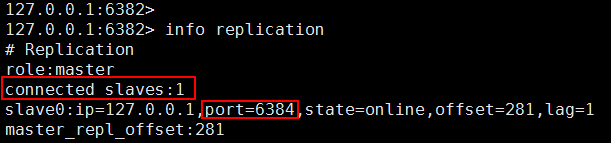
C、 将其他 Slave 挂到新的 Master

在 Slave 6384 上执行



现在的主从（Master/Slave）关系：Master 是 6382  ， Slave 是 6384

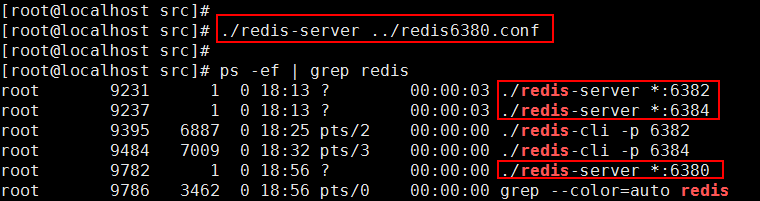
查看 6382：



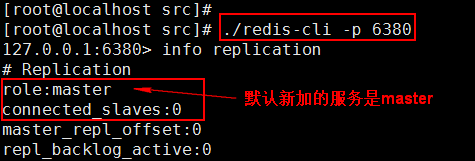
D、原来的服务器重新添加到主从结构中

6380 的服务器修改后，从新工作，需要把它添加到现有的Master/Slave 中

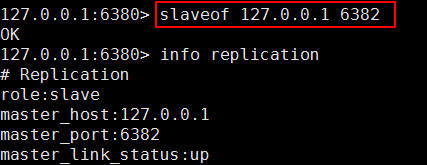
先启动 6380 的 Redis 服务



连接到 6380 端口

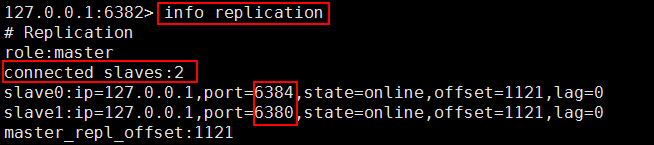


当前服务挂到 Master 上



E、 查看新的 Master 信息

在 6382 执行：



现在的 Master/Slaver 关系是：

Master: 6382

Slave: 6380

6384

操作命令

进入客户端需指定端口：./redis-cli -p 6380

【2】从机宕机

主机少一个从机，其他从机不变。

从机需要重新设置主从关系。slaveof host port

不配置启动默认都是主 master

info replication 查看 redis 服务器所处角色

总结

A、一个 master 可以有多个 slave

B、slave 下线，读请求的处理性能下降

C、master 下线，写请求无法执行

D、当 master 发生故障，需手动将其中一台 slave 使用 slaveof no one 命令提升为 master，其它 slave 执行 slaveof 命令指向这个新的master，从新的master处同步数据。

E、主从复制模式的故障转移需要手动操作，要实现自动化处理，这就需要 Sentinel 哨兵，实现故障自动转移。

Redis的哨兵

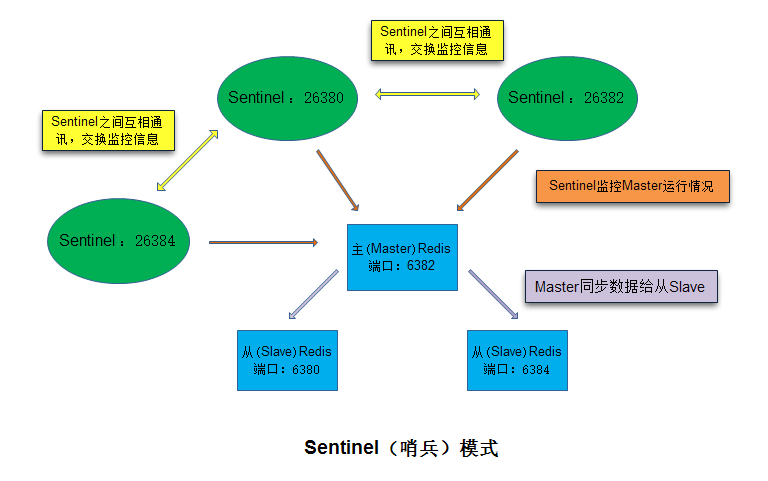
Sentinel 哨兵是 redis 官方提供的高可用方案，可以用它来监控多个 Redis 服务实例的运行情况。Redis Sentinel 是一个运行在特殊模式下的 Redis 服务器。Redis Sentinel 是在多个Sentinel 进程环境下互相协作工作的。

Sentinel 系统有三个主要任务：

● 监控：Sentinel 不断的检查主服务和从服务器是否按照预期正常工作。

● 提醒：被监控的 Redis 出现问题时，Sentinel 会通知管理员或其他应用程序。

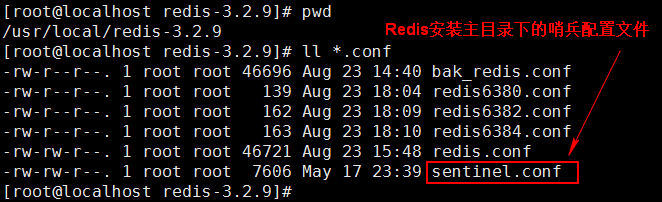
● 自动故障转移：监控的主 Redis 不能正常工作，Sentinel 会开始进行故障迁移操作。将一个从服务器升级新的主服务器。 让其他从服务器挂到新的主服务器。同时向客户端提供新的主服务器地址。



1、Sentinel 配置

Sentinel 配置文件

复制三份sentinel.conf文件



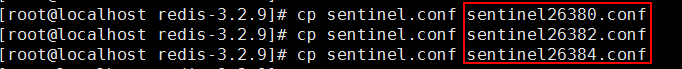
三个文件分别命名：

● sentinel26380.conf

● sentinel26382.conf

● sentinel26384.conf

● 执行复制命令 cp sentinel.conf xxx.conf



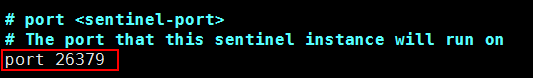
2、三份 sentinel 配置文件修改

● 修改 port 26380、 port 26382、 port 26384

● 修改 sentinel monitor mymaster 127.0.0.1 6380 2

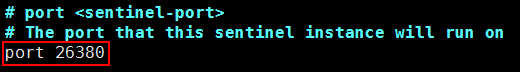
格式：sentinel monitor <name> <masterIP> <masterPort> <Quorum 投票数>

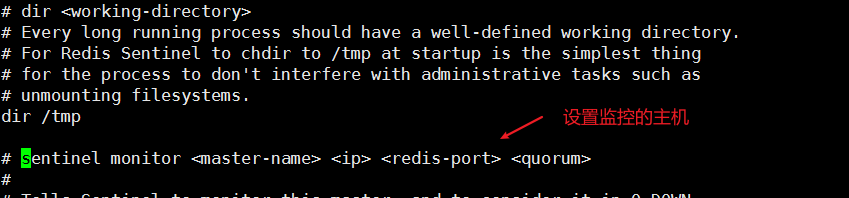
Sentinel监控主(Master)Redis, Sentinel根据Master的配置自动发现Master的Slave,Sentinel默认端口号为26379 。

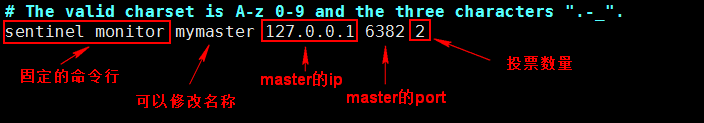


sentinel26380.conf

（1） 修改 port



1. 修改监控的 master 地址
2. 

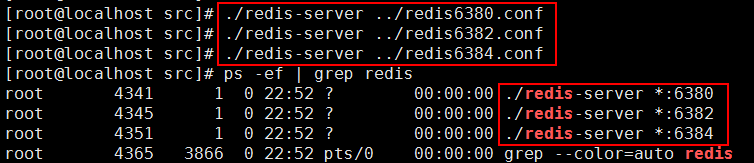


sentinel26382.conf 修改port 26382 , master的port 6382

sentinel26384.conf 修改port 26384 , master的port 6382

3、启动主从（Master/Slave）Redis

启动 Reids

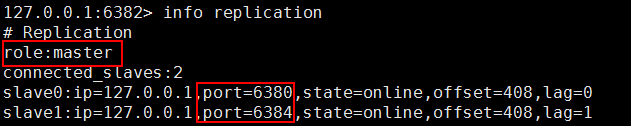


查看 Master 的配置信息

连接到 6382 端口

IMG_256

使用 info 命令查看 Master/Slave



4、启动 Sentinel

redis安装时make编译后就产生了redis-sentinel程序文件，可以在一个redis中运行多个sentinel进程。

启动一个运行在Sentinel模式下的Redis服务实例

./redis-sentinel sentinel 配置文件

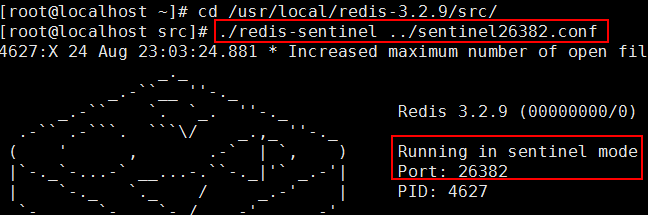
执行以下三条命令，将创建三个监视主服务器的Sentinel实例：

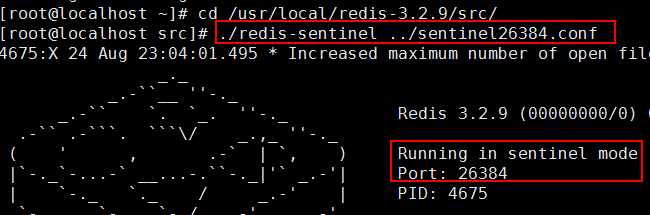
./redis-sentinel ../sentinel26380.conf

./redis-sentinel ../sentinel26382.conf

./redis-sentinel ../sentinel26384.conf

在 XShell 开启三个窗口分别执行：

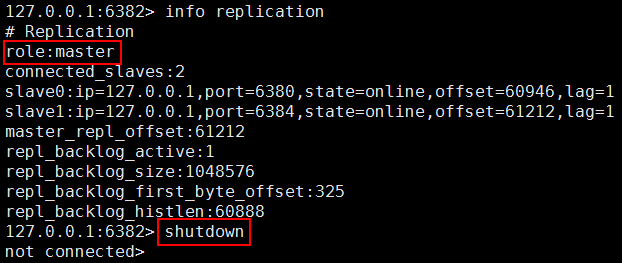




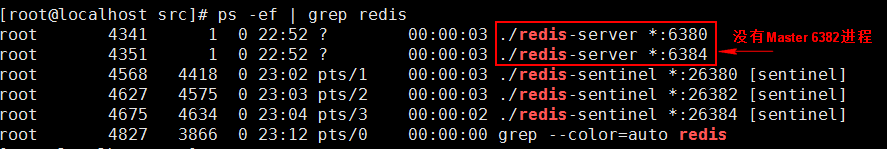
5、主 Redis 不能工作

让 Master 的 Redis 停止服务， 执行 shutdown

先执行 info replication 确认 Master 的 Redis ，再执行 shutdown



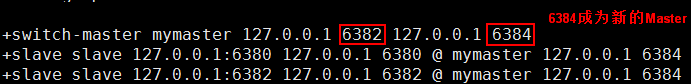
查看当前 Redis 的进程情况



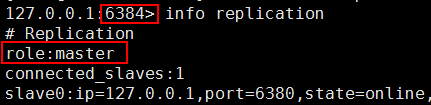
6、Sentinel 的起作用

在 Master 执行 shutdown 后， 稍微等一会 Sentinel 要进行投票计算，从可用的 Slave选举新的 Master。

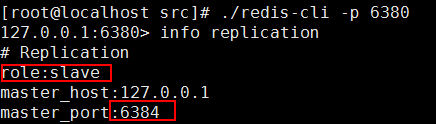
查看 Sentinel 日志，三个 Sentinel 窗口的日志是一样的。



查看新的 Master



查看原 Slave 的变化

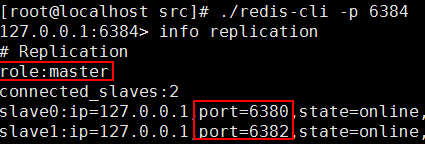


7、新的 Redis 加入 Sentinel 系统，自动加入 Master

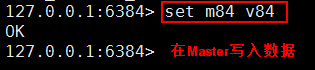
重新启动 6382

IMG_256

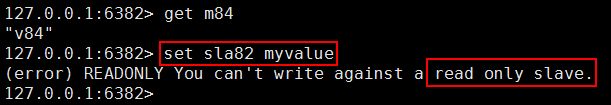
查看 6384 的信息



测试数据：在 Master 写入数据



在 6382 上读取数据，不能写入



● Sentinel 会不断检查 Master 和 Slave 是否正常

● 如果 Sentinel 挂了，就无法监控，所以需要多个哨兵，组成 Sentinel 网络，一个健康的Sentinel 至少有 3 个 Sentinel 应用。 彼此在独立的物理机器或虚拟机。

● 监控同一个 Master 的 Sentinel 会自动连接，组成一个分布式的 Sentinel 网络，互相通信并交换彼此关于被监控服务器的信息

● 当一个 Sentinel 认为被监控的服务器已经下线时，它会向网络中的其它 Sentinel 进行确认，判断该服务器是否真的已经下线

● 如果下线的服务器为主服务器，那么 Sentinel 网络将对下线主服务器进行自动故障转移， 通过将下线主服务器的某个从服务器提升为新的主服务器，并让其从服务器转移到新的主服务器下，以此来让系统重新回到正常状态

● 下线的旧主服务器重新上线，Sentinel 会让它成为从，挂到新的主服务器下

9、总结

主从复制，解决了读请求的分担，从节点下线，会使得读请求能力有所下降，Master 下线，写请求无法执行

Sentinel 会在 Master 下线后自动执行故障转移操作，提升一台 Slave 为 Master，并让其它Slave 成为新 Master 的 Slave。

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。jedis 完全兼容 redis 2.8.x and 3.x.x

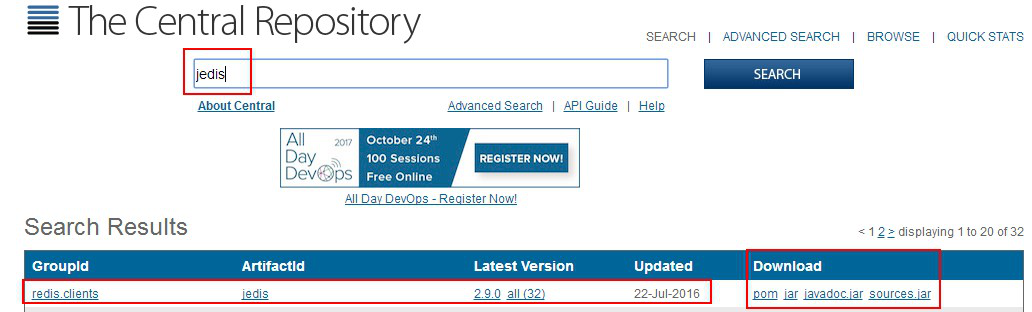
● Jedis 源码：https://github.com/xetorthio/jedis

● api 文档：http://xetorthio.github.io/jedis/

● 下载：http://search.maven.org/ ，搜索 jedis

下载 Jedis

浏览器打开：http://search.maven.org/ ，搜索 jedis。在 Download 处，点击 jar



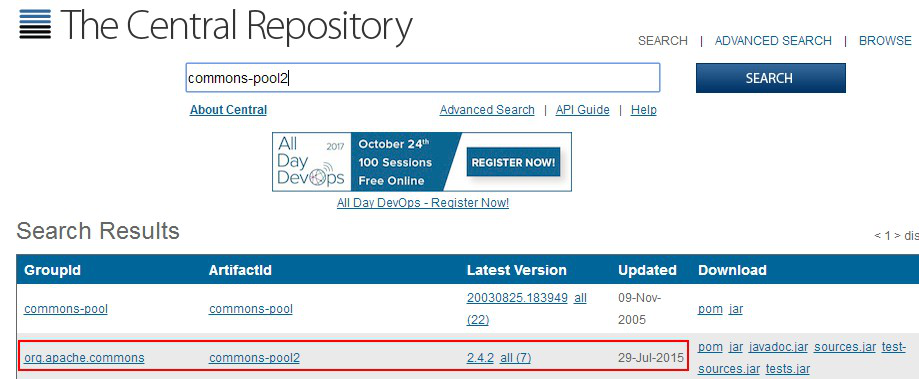
最新的版本 jedis-2.9.0



下载 Commons-Pool

Jedis 对象并不是线程安全的，在多线程下使用同一个 Jedis 对象会出现并发问题。为了避免每次使用 Jedis 对象时都需要重新构建，Jedis 提供了 JedisPool。JedisPool 是基于Commons Pool 2 实现的一个线程安全的连接池

浏览器打开：http://search.maven.org/ ，搜索 commons-pool2。在 Download 处，点击 jar



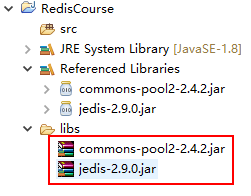


项目中加入 jar：

● jedis-2.9.0.jar

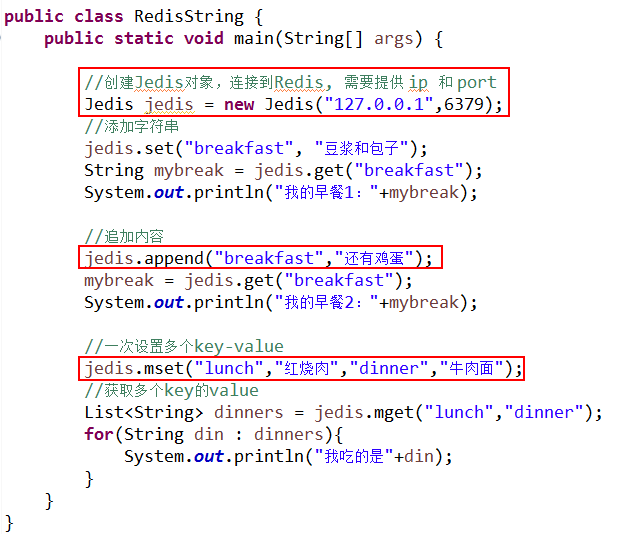
● commons-pool2-2.4.2.jar

加入项目后如图：



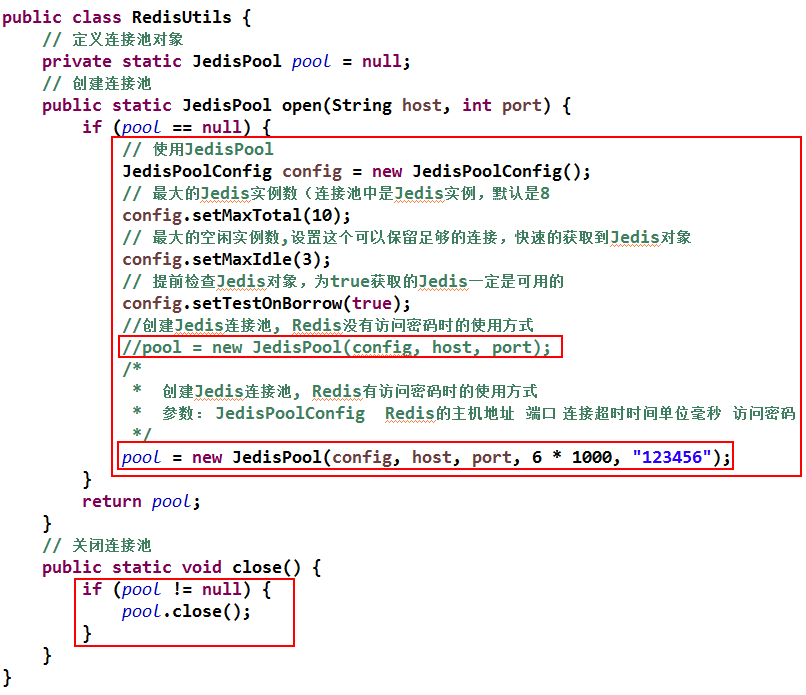
使用之前要到如slf4j api、和slf4j simple 两个jar包！

字符串(string)



哈希（hash）

● 使用 Jedis 连接实例池。



● 使用连接池操作 hash 数据类型

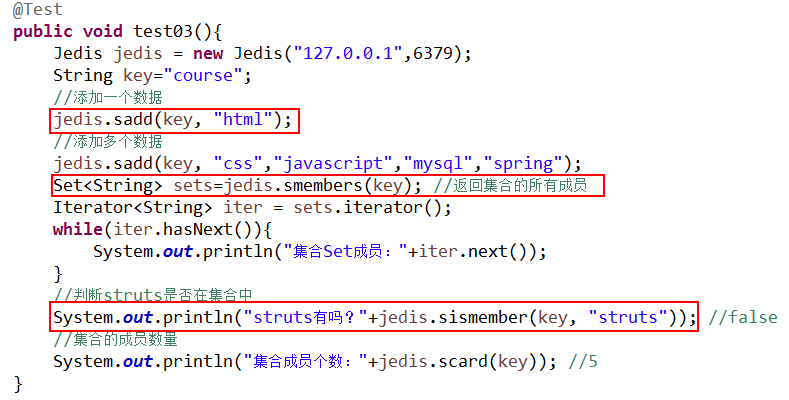




列表 list



集合 Set



有序集合 Sorted Set



事务（Transaction）



支持部分事务操作