Redis

**1. 关于关系型数据库和nosql数据库（键值对存储）**

关系型数据库是基于关系表的数据库，最终会将数据持久化到磁盘上，而nosql数据库是基于特殊的结构，并将数据存储到内存的数据库。从性能上而言，nosql数据库要优于关系型数据库，从安全性上而言关系型数据库要优于nosql数据库，所以在实际开发中一个项目中nosql和关系型数据库会一起使用，达到性能和安全性的双保证。

**2. Redis简介**

redis是一个key-value存储系统。和Memcached类似，它支持存储的value类型相对更多，包括string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。为了保证效率，数据都是缓存在内存中，因为是纯内存操作，Redis的性能非常出色，每秒可以处理超过 10万次读写操作，是已知综合性能最快的Key-Value DB。redis是单进程单线程的，采用多路 I/O 复用技术。

**3. Linux环境下Redis安装**

官网：https://redis.io

下载：http://download.redis.io/releases

中文网：https://www.redis.net.cn/

官网提供安装方式如下：（安装redis编译的c环境，此步骤没有编译）

|  |
| --- |
| **Installation**  Download, extract and compile Redis with:  $ wget [http://download.redis.io/releases/redis-5.0.5.tar.gz](http://download.redis.io/releases/redis-5.0.5.tar.gz )  $ tar xzf redis-5.0.5.tar.gz $ cd redis-5.0.5  $ make  The binaries that are now compiled are available in the src directory. Run Redis with:  $ src/redis-server  You can interact with Redis using the built-in client:  $ src/redis-cli  redis> set foo bar  OK  redis> get foo  "bar" |

**安装步骤：**

1.把下载好的redis-5.0.4.tar.gz安装包拷贝到当前虚拟机root目录下，解压到/usr/local下

[root@localhost ~]# tar -zxvf redis-5.0.4.tar.gz -C /usr/local

2.编译的c环境

[root@localhost ~]# yum install gcc-c++

3.进入redis-5.0.4目录 使用make命令编译redis（若报错，先make distclean，再make）

[root@localhost redis-5.0.4]# make

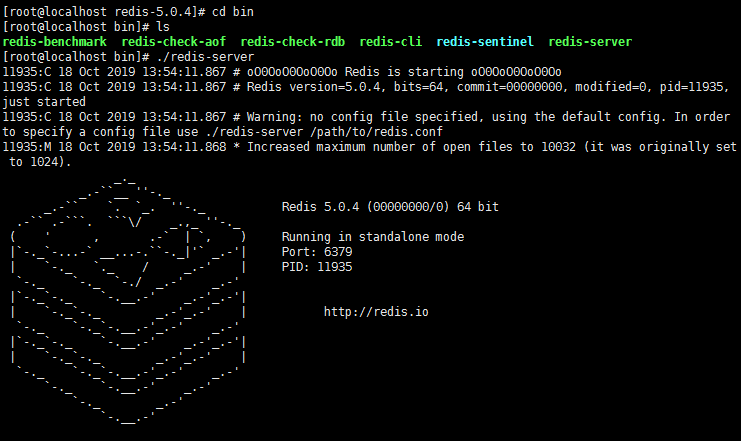
4.使用make PREFIX=/usr/local/redis-5.0.4 install命令安装（安装后会出现bin目录）

[root@localhost redis-5.0.4]# make PREFIX=/usr/local/redis-5.0.4 install

5.启动redis服务端（前台启动）

[root@localhost redis-5.0.4]# cd bin

[root@localhost bin]# ./redis-server



启动后看到如上欢迎页面，但此窗口不能关闭，窗口关闭就认为redis也关闭了，所以我们需要在后台启动，然后再启动客户端进行连接，所以首先Ctrl+C退出。

解决：可以通过修改配置文件配置redis的后台启动（即服务器启动了但不会创建控制台窗口）

步骤如下：

1.切换到redis-5.0.4目录下，把当前目录下的redis.conf文件拷贝到bin目录下

[root@localhost bin]# cd ../

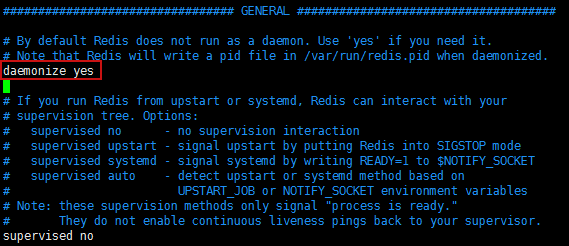
[root@localhost redis-5.0.4]# cp redis.conf bin/redis.conf

2.切换到bin目录下，修改redis.conf文件

[root@localhost redis-5.0.4]# cd bin

[root@localhost bin]# vim redis.conf

3.将redis.conf文件中的daemonize的值从no修改成yes表示后台启动



4.启动redis服务端（后台启动）

[root@localhost bin]# ./redis-server redis.conf

5.查看是否启动成功

[root@localhost bin]# ps -ef | grep redis

6.启动客户端

[root@localhost bin]# ./redis-cli

7.存取数据进行测试

127.0.0.1:6379> set name jack

OK

127.0.0.1:6379> get name

"jack"

**4.Redis的常用命令**

redis是一种高级的key-value的存储系统，其中的key是字符串类型，尽可能满足如下几点：

1.key不要太长，最好不要操作1024个字节，这不仅会消耗内存还会降低查找效率

2.key不要太短，如果太短会降低key的可读性

3.在项目中，key最好有一个统一的命名规范（根据企业的需求）

value最常用的五种数据类型：

1.字符串(String)

2.列表(List)

3.集合(Set)

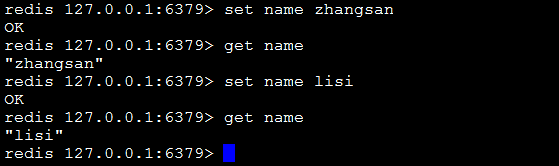
4.有序集合(sorted set)

5.哈希(Hash)

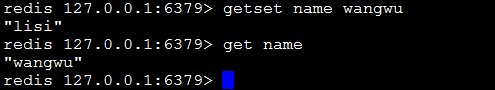
**1.字符串(String)**

set key value：设定key持有指定的字符串value，如果该key存在则进行覆盖操作，总是返回”OK”。

get key：获取key的value。如果与该key关联的value不是String类型，redis将返回错误信息，因为get命令只能用于获取String value，如果该key不存在，返回null。

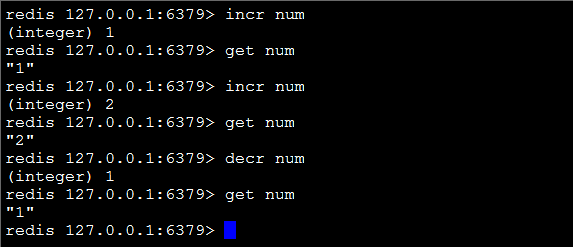


getset key value：先获取该key的值，然后在设置该key的值。



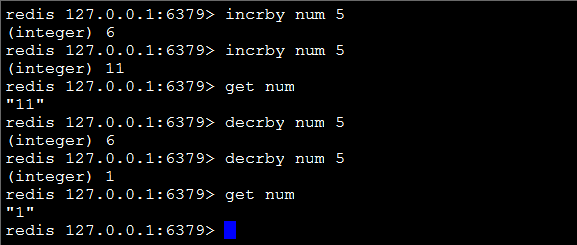
incr key：将指定的key的value原子性的递增1.如果该key不存在，其初始值为0，在incr之后其值为1。如果value的值不能转成整型，如hello，该操作将执行失败并返回相应的错误信息。

decr key：将指定的key的value原子性的递减1.如果该key不存在，其初始值为0，在incr之后其值为-1。如果value的值不能转成整型，如hello，该操作将执行失败并返回相应的错误信息。

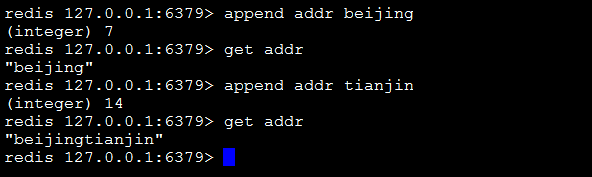


incrby key increment：将指定的key的value原子性增加increment，如果该key不存在，器初始值为0，在incrby之后，该值为increment。如果该值不能转成整型，如hello则失败并返回错误信息。

decrby key decrement：将指定的key的value原子性减少decrement，如果该key不存在，器初始值为0，在decrby之后，该值为decrement。如果该值不能转成整型，如hello则失败并返回错误信息。



append key value：如果该key存在，则在原有的value后追加该值；如果该key不存在，则重新创建一个key/value。

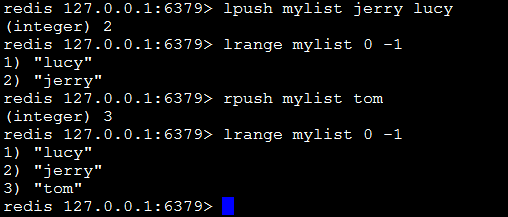


**2.列表(List)**

lpush key value1 value2...：在指定的key所关联的list的头部插入所有的values，如果该key不存在，该命令在插入的之前创建一个与该key关联的空链 表，之后再向该链表的头部插入数据。插入成功，返回元素的个数。

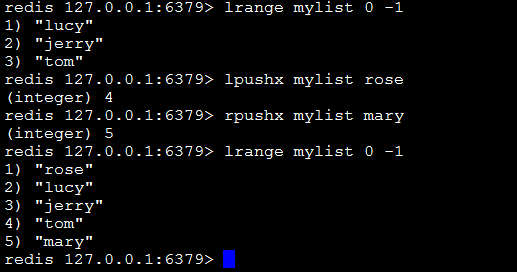
rpush key value1 value2…：在该list的尾部添加元素。

lrange key start end：获取链表中从start到end的元素的值，start、end可为负数，若为-1则表示链表尾部的元素，-2则表示倒数第二个，依次类推….



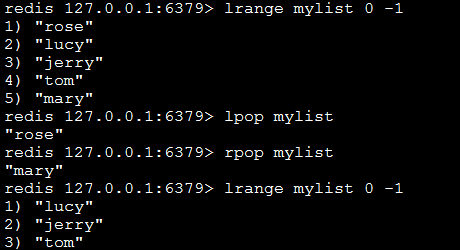
lpushx key value：仅当参数中指定的key存在时（如果与key管理的list中没有值时，则该key是不存在的）在指定的key所关联的list的头部插入value。

rpushx key value：在该list的尾部添加元素。

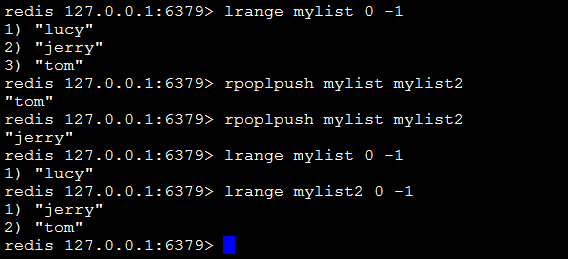


lpop key：返回并弹出指定的key关联的链表中的第一个元素，即头部元素。

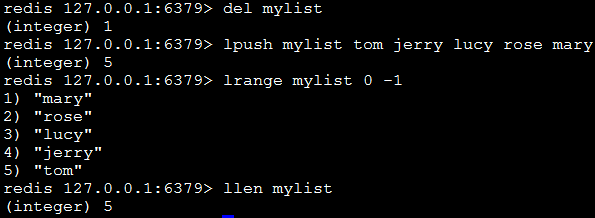
rpop key：从尾部弹出元素。



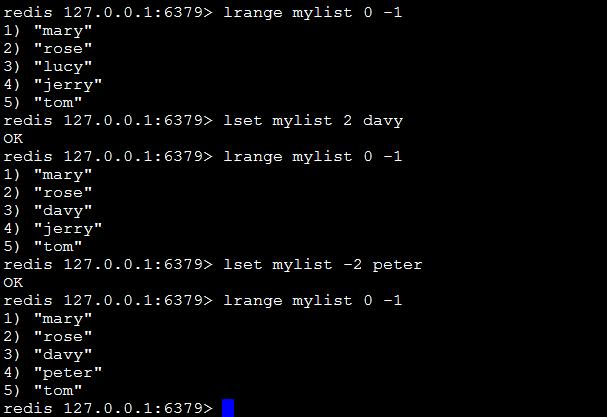
rpoplpush resource destination：将链表中的尾部元素弹出并添加到头部。



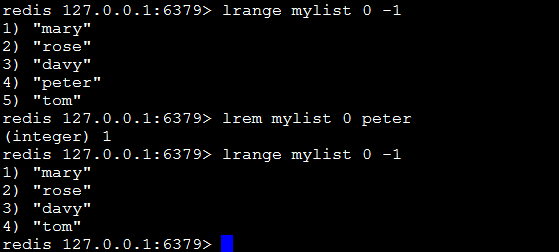
llen key：返回指定的key关联的链表中的元素的数量。



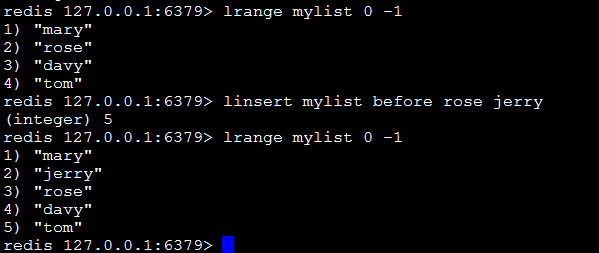
lset key index value：设置链表中的index的脚标的元素值，0代表链表的头元素，-1代表链表的尾元素。



lrem key count value：删除count个值为value的元素，如果count大于0，从头向尾遍历并删除count个值为value的元素，如果count小于0，则从尾向头遍历并删除。如果count等于0，则删除链表中所有等于value的元素。



linsert key before|after pivot value：在pivot元素前或者后插入value这个元素。

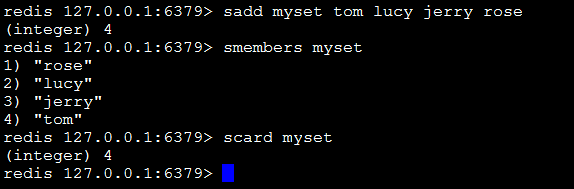


**3.集合(Set，不允许出现重复的元素)**

sadd key value1 value2…：向set中添加数据，如果该key的值已有则不会重复添加。

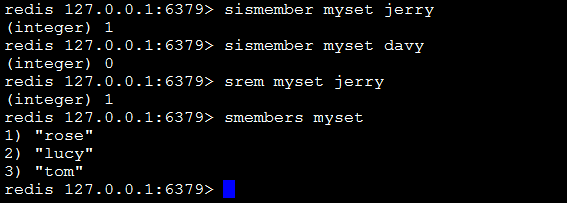
smembers key：获取set中所有的成员。

scard key：获取set中成员的数量。

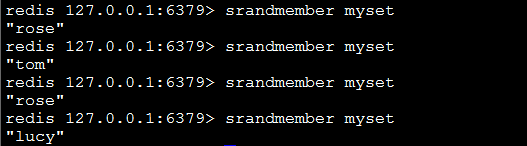


sismember key member：判断参数中指定的成员是否在该set中，1表示存 在，0表示不存在或者该key本身就不存在。

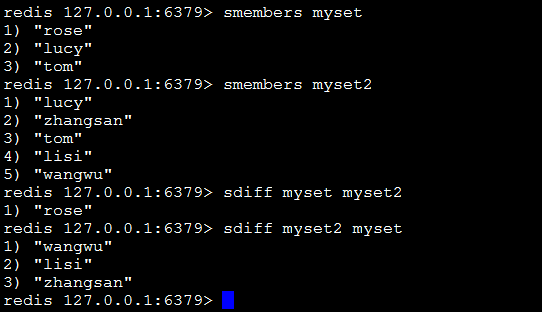
srem key member1 member2…：删除set中指定的成员。



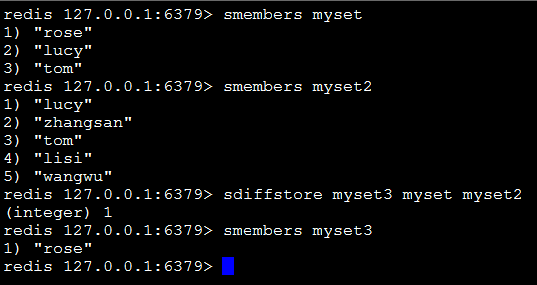
srandmember key：随机返回set中的一个成员。



sdiff key1 key2：返回key1与key2中相差的成员，而且与key的顺序有关，即返回差集。

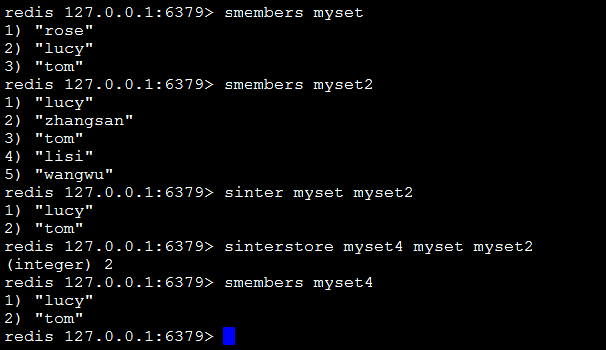


sdiffstore destination key1 key2：将key1、key2相差的成员存储在destination上。

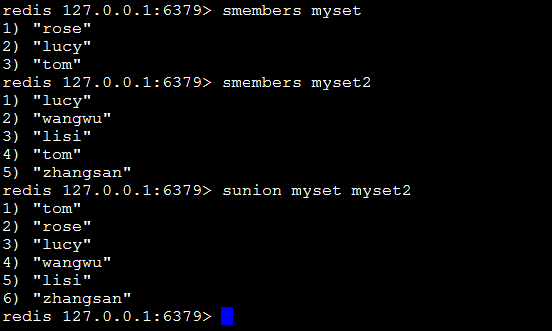


sinter key[key1,key2…]：返回交集。

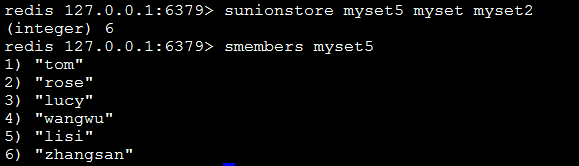
sinterstore destination key1 key2：将返回的交集存储在destination上。



sunion key1 key2：返回并集。



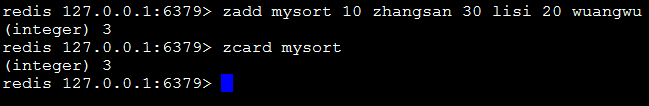
sunionstore destination key1 key2：将返回的并集存储在destination上



**4.有序集合(sorted set)**

zadd key score member score2 member2 … ：将所有成员以及该成员的分数存放到sorted-set中。

zcard key：获取集合中的成员数量。



zcount key min max：获取分数在[min,max]之间的成员。

zincrby key increment member：设置指定成员的增加的分数。

zrange key start end [withscores]：获取集合中脚标为start-end的成员，[withscores]参数表明返回的成员包含其分数。

zrangebyscore key min max [withscores] [limit offset count]：返回分数在[min,max]的成员并按照分数从低到高排序。[withscores]：显示分数；[limit offset count]：offset，表明从脚标为offset的元素开始并返回count个成员。

zrank key member：返回成员在集合中的位置。

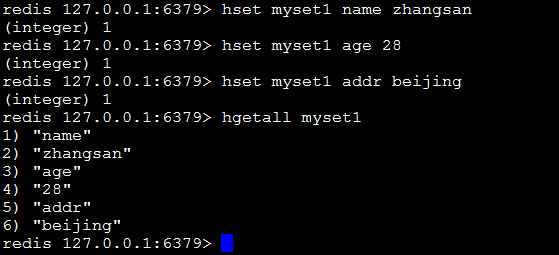
zrem key member[member…]：移除集合中指定的成员，可以指定多个成员。

zscore key member：返回指定成员的分数。

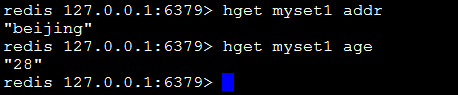
**5.哈希(Hash)**

hset key field value：为指定的key设定field/value对（键值对）。

hgetall key：获取key中的所有filed-vaule。



hget key field：返回指定的key中的field的值。



hmset key fields：设置key中的多个filed/value。

hmget key fileds：获取key中的多个filed的值。

hexists key field：判断指定的key中的filed是否存在。

hlen key：获取key所包含的field的数量。

hincrby key field increment：设置key中filed的值增加increment，如：age增加20。

**5.Keys的通用操作**

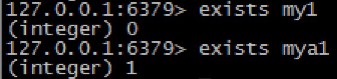
keys patten：获取所有与patten匹配的key，\*表示任意字符，?表示一个字符。



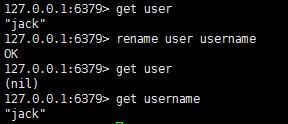
del key1 key2....：删除指定的key。



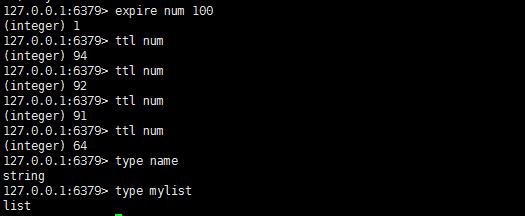
exists key：判断该key是否存在，1表示存在，0表示不存在。



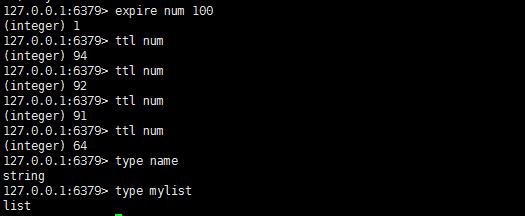
renamel key newkey：为当前key重命名。



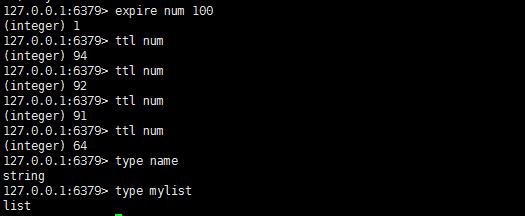
expire key second：为当前key设置过期时间（单位：秒）。



ttl key：查看当前key剩余过期时间。

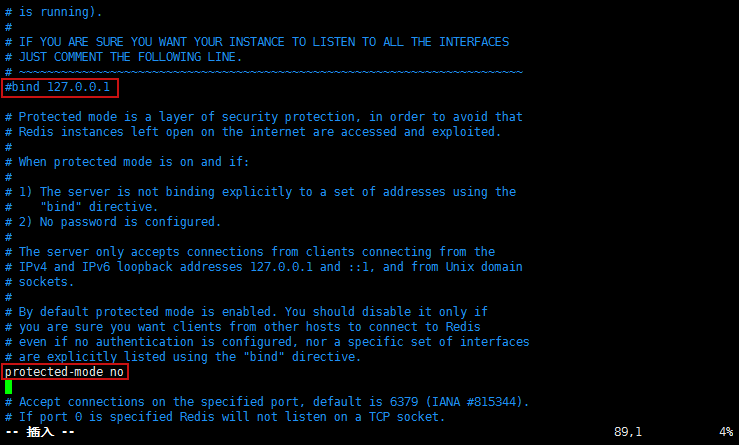


type key：查看当前key类型。



**6.Jedis的使用**

1.修改/usr/local/redis-5.0.4/bin目录下的redis.conf配置文件，然后启动redis服务端



将绑定127.0.0.1注释掉，然后把保护模式关掉即可.

2.创建Maven工程，导入依赖

|  |
| --- |
| <**dependency**>  <**groupId**>redis.clients</**groupId**>  <**artifactId**>jedis</**artifactId**>  <**version**>3.1.0</**version**> </**dependency**> |

3.编写测试类

|  |
| --- |
| **package** com.qf.jedis;  **import** org.junit.Test; **import** redis.clients.jedis.Jedis; **import** redis.clients.jedis.JedisPool; **import** redis.clients.jedis.JedisPoolConfig;  **public class** JedisDemo {   *// 通过java程序访问redis数据库   // 获得单一的jedis对象操作数据库* @Test  **public void** test1() {   *//获得连接对象* Jedis jedis = **new** Jedis(**"192.168.153.132"**, 6379);   *//获得之前redis中存储的数据* String name = jedis.get(**"name"**);  System.***out***.println(name);   *//存储数据* jedis.set(**"password"**, **"123"**);  System.***out***.println(jedis.get(**"password"**));   *//关闭* jedis.close();  }   *//通过jedis的pool获得jedis连接对象* @Test  **public void** test2() {  *// 0、创建池子的配置对象* JedisPoolConfig poolConfig = **new** JedisPoolConfig();  poolConfig.setMaxIdle(30);*// 最大闲置个数* poolConfig.setMinIdle(10);*// 最小闲置个数* poolConfig.setMaxTotal(50);*// 最大连接数   // 1、创建一个redis的连接池* JedisPool pool = **new** JedisPool(poolConfig, **"192.168.153.132"**, 6379);   *// 2、从池子中获取redis的连接资源* Jedis jedis = pool.getResource();   *// 3、操作数据库* jedis.set(**"age"**, **"18"**);  System.***out***.println(jedis.get(**"age"**));   *// 4、关闭资源* jedis.close();  pool.close();   } } |

**7.手机验证码练习**

要求：

1.输入手机号，随机发送6位验证码，2分钟内有效。

2.验证码每天只能发送三次

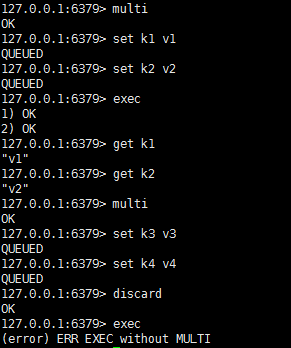
**8.Redis中的事务**

Redis中的事务和MySQL中的事务有本质的区别，Redis中的事务是一个单独的隔离操作，事务中所有的命令都会序列化，按照顺序执行，事务在执行的过程中，不会被其他客户端发来的命令所打断，因为Redis服务端是个单线程的架构，不同的Client虽然看似可以同时保持连接，但发出去的命令是序列化执行的，这在通常的数据库理论下是最高级别的隔离。

Redis中的事务的作用就是串联多个命令，防止别的命令插队。

常用命令：multi、exec、discard、watch、unwacth

当输入multi命令时，之后输入的命令都会被放在队列中，但不会执行，直到输入exec后，Redis会将队列中的命令依次执行，discard用来撤销Exec之前被暂存的命令，并不是回滚。

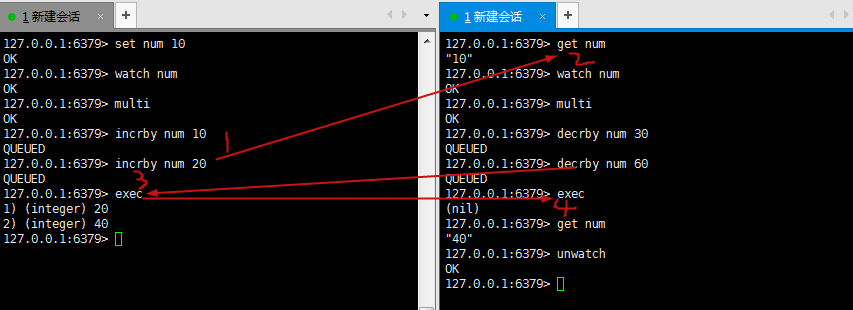


watch/unwacth

在执行multi之前，先执行watch key1 [key2...] ，watch提供的乐观锁功能（初始时一个版本号，exec之后会更新当前版本号），在你exec的那一刻，如果被watch的键发生过改动，则multi到exec之间的指令全部不执行。

watch表示监控，相当于加锁，但在执行完exec时就会解锁。

unwacth取消所有锁。



**Redis中的事务的特性总结**

**1.单独的隔离操作**

事务中的所有命令都会序列化，然后按顺序执行，在执行过程中，不会被其他客户端发送的命令打断。

**2.没有隔离级别的概念**

队列中的命令没有被提交之前都不会执行。

**3.不能保证原子性**

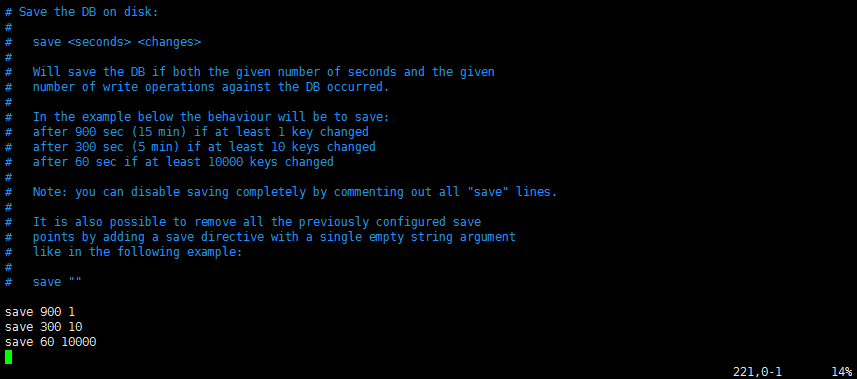
Redis同一个事务中如果有一条命令执行失败，其后的命令仍然会被执行，不会回滚。

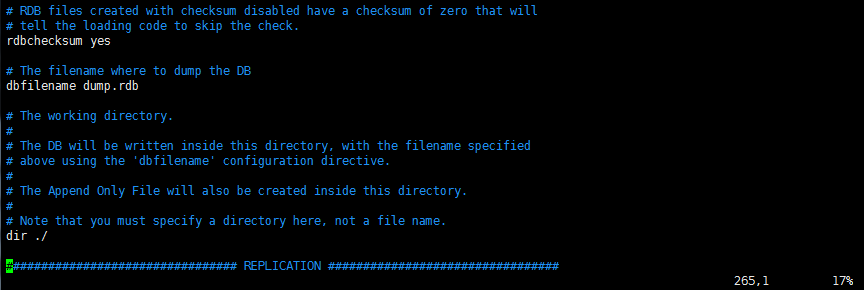
**9.Redis中的持久化**

Redis有两种持久化方式：RDB和AOF。

**1.RDB（Redis DataBase）**

将内存中的数据以快照的方式写入磁盘中，在redis.conf文件中，我们可以找到如下配置：





save 900 1

save 300 10

save 60 10000

配置含义：

900秒内，如果超过1个key被修改，则发起快照保存

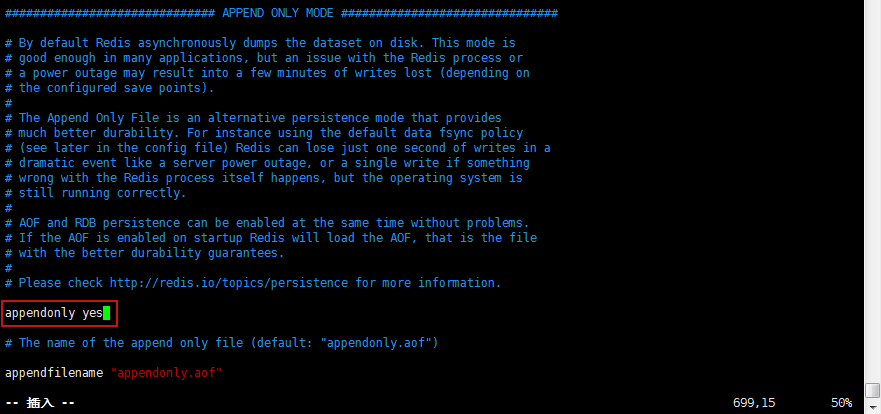
300秒内，如果超过10个key被修改，则发起快照保存

60秒内，如果1万个key被修改，则发起快照保存

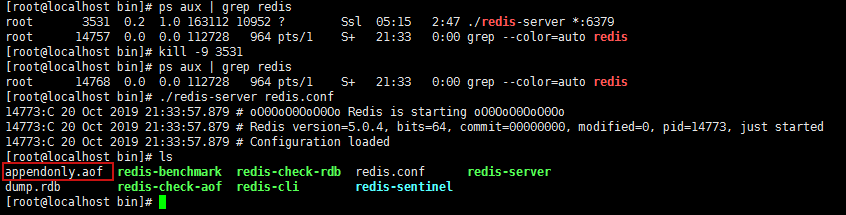
RDB方式存储的数据会在dump.rdb文件中（在哪个目录启动redis服务端，该文件就会在对应目录下生成），该文件不能查看，如需备份，对Redis操作完成之后，只需拷贝该文件即可（Redis服务端启动时会自动加载该文件）

**2.AOF（Append Of File）**

AOF默认是不开启的，需要手动开启，同样是在redis.conf文件中开启，如下：



配置文件中的appendonly修改为yes，开启AOF持久化。开启后，启动redis服务端，发现多了一个appendonly.aof文件。



之后任何的操作都会保存在appendonly.aof文件中，可以进行查看，Redis启动时会将appendonly.aof文件中的内容执行一遍。

如果AOF和RDB同时开启，系统会默认读取AOF的数据。

**RDB优点与缺点**

**优点**

如果要进行大规模数据的恢复，RDB方式要比AOF方式恢复速度要快。

RDB是一个非常紧凑(compact)的文件,它保存了某个时间点的数据集，非常适合用作备份，同时也非常适合用作灾难性恢复，它只有一个文件，内容紧凑，通过备份原文件到本机外的其他主机上，一旦本机发生宕机，就能将备份文件复制到redis安装目录下，通过启用服务就能完成数据的恢复。

**缺点**

RDB这种持久化方式不太适应对数据完整性要求严格的情况，因为，尽管我们可以用过修改快照实现持久化的频率，但是要持久化的数据是一段时间内的整个数据集的状态，如果在还没有触发快照时，本机就宕机了，那么对数据库所做的写操作就随之而消失了并没有持久化本地dump.rdb文件中。

**AOF优点与缺点**

**优点**

AOF有着多种持久化策略：

appendfsync always:每修改同步，每一次发生数据变更都会持久化到磁盘上，性能较差，但数据完整性较好。

appendfsync everysec: 每秒同步，每秒内记录操作，异步操作，如果一秒内宕机，有数据丢失。

appendfsync no:不同步。

AOF文件是一个只进行追加操作的日志文件，对文件写入不需要进行seek，即使在追加的过程中，写入了不完整的命令（例如：磁盘已满），可以使用redis-check-aof工具可以修复这种问题

Redis可以在AOF文件变得过大时，会自动地在后台对AOF进行重写：重写后的新的AOF文件包含了恢复当前数据集所需的最小命令集合。整个重写操作是绝对安全的，因为Redis在创建AOF文件的过程中，会继续将命令追加到现有的AOF文件中，即使在重写的过程中发生宕机，现有的AOF文件也不会丢失。一旦新AOF文件创建完毕，Redis就会从旧的AOF文件切换到新的AOF文件，并对新的AOF文件进行追加操作。

**缺点**

对于相同的数据集来说，AOF文件要比RDB文件大。

根据所使用的持久化策略来说，AOF的速度要慢于RDB。一般情况下，每秒同步策略效果较好。不使用同步策略的情况下，AOF与RDB速度一样快。

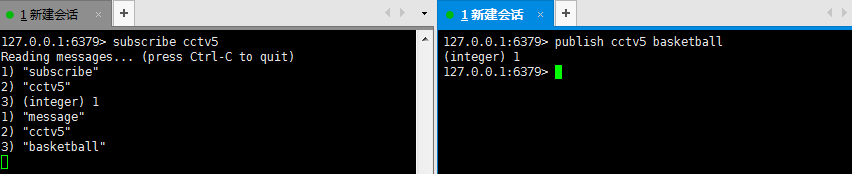
**10.Redis中的消息的订阅与发布**

subscribe channel 订阅频道 例如：subscribe cctv5

subscribe channel\* 批量订阅频道 例如：subscribe cctv\* 表示订阅以cctv开头的频道

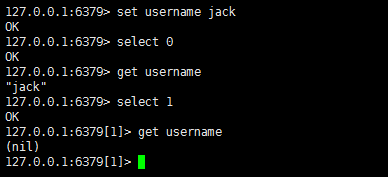
publish channel content 在指定频道中发布内容 例如：publish cctv5 basketball

同时打开两个客户端，一个订阅频道，一个在频道中发布内容，订阅频道的客户端会接收到消息。

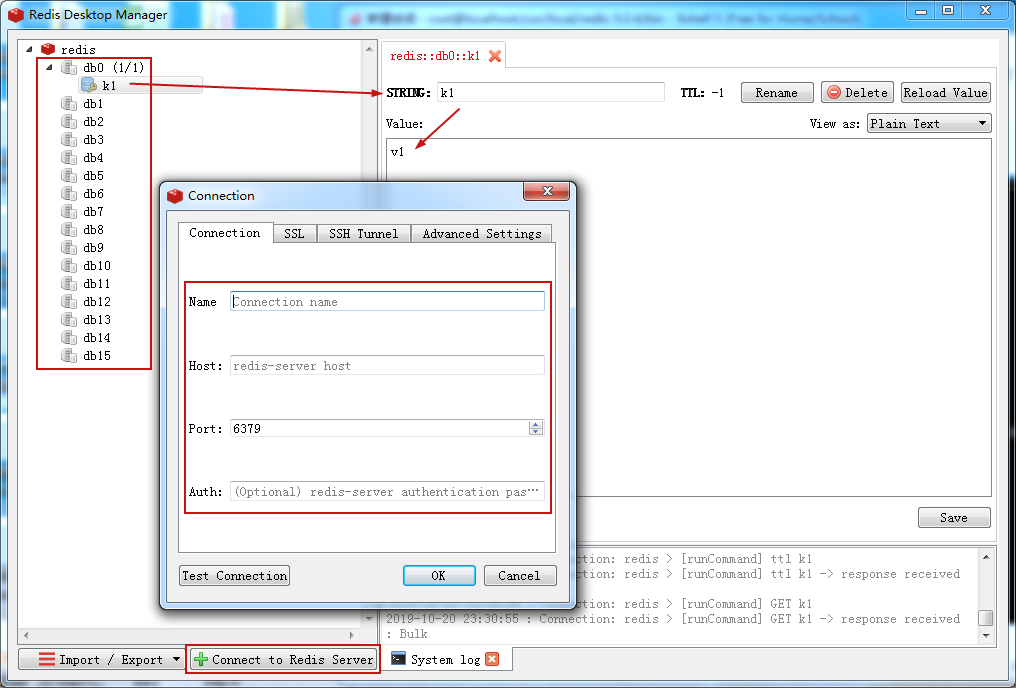


**11.使用Redis图形化程序界面**

Redis中默认有16个库，可以在不同的库中存储数据，默认使用0号库存储数据，使用select 0-15可以选择不同的库。



安装之后，输入IP地址登录即可看到如下界面：



**Redis的主从复制和集群（扩展内容）**