

PRedis使用指南

Redis简介

1. Redis: remote dictionary server 远程字典服务器
2. Redis 是完全开源免费的，遵守BSD协议，是一个高性能的key-value数据库。
3. Redis 与其他 key - value 缓存产品有以下三个特点：
 - Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
 - Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。
 - Redis支持数据的备份，即master-slave模式的数据备份。

PRedis简介

提供了一个Redis的网站客户端，不用每次在终端输入命令，可以直接使用PRedis直接操作。

```
redis-server /myredis/redis.conf  
redis-cli -p6379
```

PRedis v1.0.1使用指南

本地测试环境

系统：deepin15.8

软件：redis-5.0.0

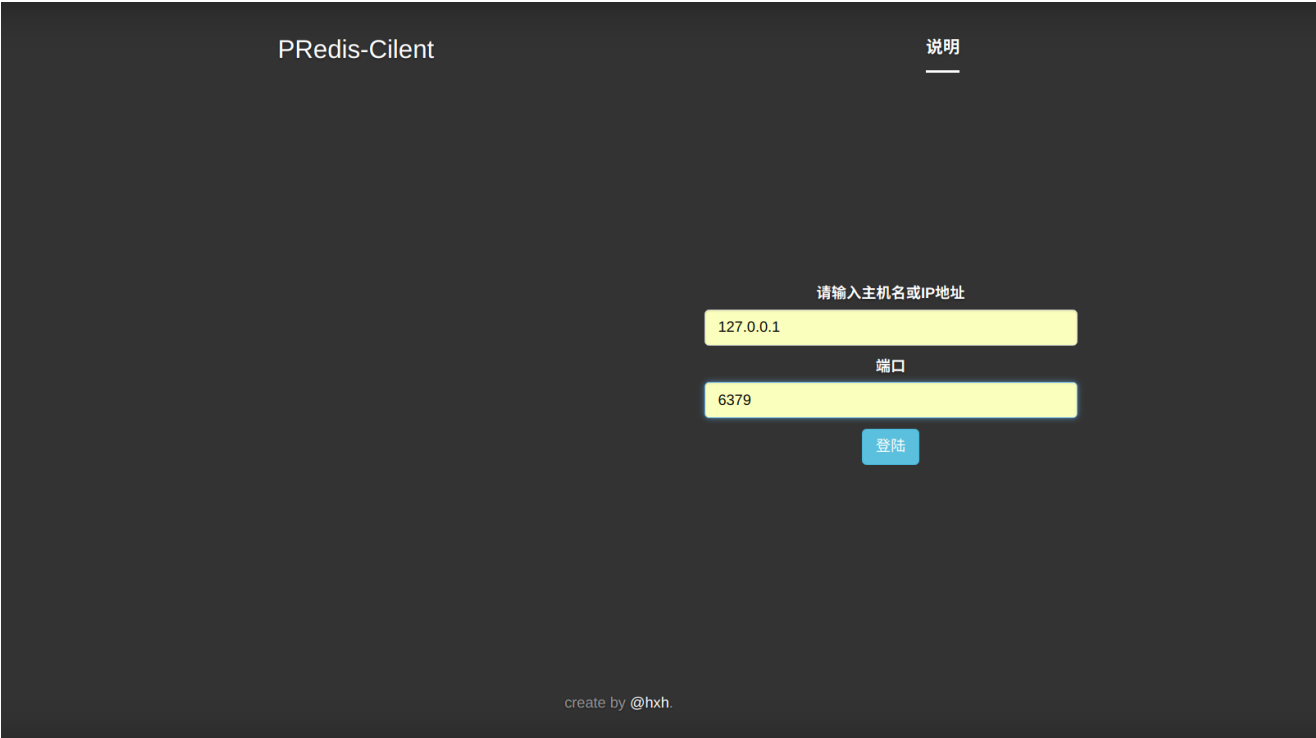
下面演示有右边在deepin上Terminal对比

下面结果返回的都是数组或者bool

1. 登录

- 输入host地址或主机名：127.0.0.1

- 输入Redis配置的端口： 默认是**6379**

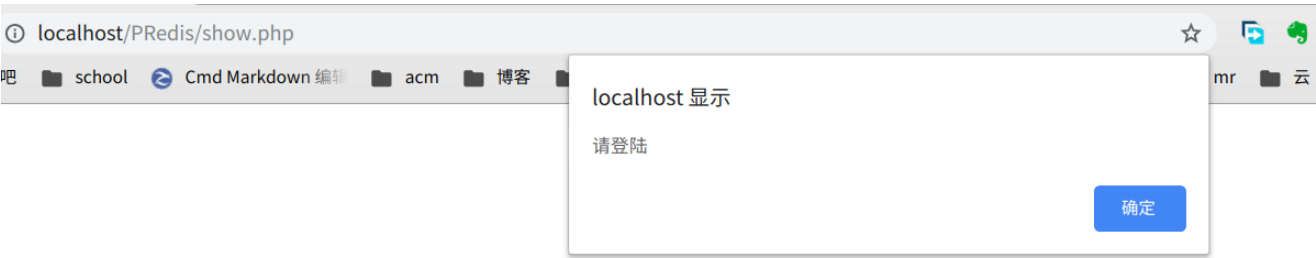


输入不能为空或者输错，链接失败返回信息：

```
Uncaught exception 'PhpRedisException' with message '无法连接redis服务器：拒绝连
```

2. 验证是否登录

如果没有登录，不允许进行数据操作，返回登录界面



3. 验证是否连通：ping

请阅读帮助后输入语句

请输入

1

ping

运行

结果:
string(4) "PONG"

4. 输入格式：option， key， value

注意每个后面跟着“，”和空格“ ”

5. 查看数据库的容量：dbsize

请阅读帮助后输入语句

请输入

1

dbsize

运行

结果:
string(1) "8"

6. key

- 查看所有的key：keys， *

请阅读帮助后输入语句

请输入

1 keys *

运行

结果:
array(8) { [0]=> string(5) "hello" [1]=> string(6) "foobar" [2]=> string(3) "foo" [3]=> string(2) "k1" [4]=> string(7) "animals" [5]=> string(5) "count" [6]=> string(6) "list01" [7]=> string(5) "Count" }

```
bin bin bin +
root@shuxnhs-PC:/usr/local/bin# redis-server /myredis/redis.conf
28336:C 17 Nov 2018 14:14:04.038 # 0000o000o000o Redis is starting o000
o000o000o
28336:C 17 Nov 2018 14:14:04.038 # Redis version=5.0.0, bits=64, commit
=000000000, modified=0, pid=28336, just started
28336:C 17 Nov 2018 14:14:04.038 # Configuration loaded
root@shuxnhs-PC:/usr/local/bin# redis-cli -p 6379
127.0.0.1:6379> keys *
1) "hello"
2) "foobar"
3) "foo"
4) "k1"
5) "animals"
6) "count"
7) "list01"
8) "Count"
127.0.0.1:6379>
```

- 设置key：set, key, value

请阅读帮助后输入语句

请输入

1 set, hello, world

运行

结果:
string(2) "OK"

```
bin bin bin +
0000o000o
28336:C 17 Nov 2018 14:14:04.038 # Redis version=5.0.0, bits=64, commit
=000000000, modified=0, pid=28336, just started
28336:C 17 Nov 2018 14:14:04.038 # Configuration loaded
root@shuxnhs-PC:/usr/local/bin# redis-cli -p 6379
127.0.0.1:6379> keys *
1) "hello"
2) "foobar"
3) "foo"
4) "k1"
5) "animals"
6) "count"
7) "list01"
8) "Count"
127.0.0.1:6379> get hello
"world"
127.0.0.1:6379>
```

- 获取key的值:get, key

请阅读帮助后输入语句

请输入

1 get, hello

运行

结果:
string(5) "world"

```
bin bin bin +
o000o000o
28336:C 17 Nov 2018 14:14:04.038 # Redis version=5.0.0, bits=64, commit
=00000000, modified=0, pid=28336, just started
28336:C 17 Nov 2018 14:14:04.038 # Configuration loaded
root@shuxnhs-PC: /usr/local/bin# redis-cli -p 6379
127.0.0.1:6379> keys *
1) "hello"
2) "foobar"
3) "foo"
4) "k1"
5) "animals"
6) "count"
7) "list01"
8) "Count"
127.0.0.1:6379> get hello
"world"
127.0.0.1:6379>
```

- 删除key的值:del, key

请阅读帮助后输入语句

请输入

1 del, hello

运行

结果:
string(1) "1"

```
bin bin bin +
=00000000, modified=0, pid=28336, just started
28336:C 17 Nov 2018 14:14:04.038 # Configuration loaded
root@shuxnhs-PC: /usr/local/bin# redis-cli -p 6379
127.0.0.1:6379> keys *
1) "hello"
2) "foobar"
3) "foo"
4) "k1"
5) "animals"
6) "count"
7) "list01"
8) "Count"
127.0.0.1:6379> get hello
"world"
127.0.0.1:6379> get hello
(nil)
127.0.0.1:6379>
```

- 查看键过期时间：ttl, key

请阅读帮助后输入语句

请输入

```
1 set, hello, world
```

运行

结果:
string(2) "1"

```
root@shuxnhs-PC: /usr/local/bin# redis-cli -p 6379
127.0.0.1:6379> keys *
1) "hello"
2) "foobar"
3) "foo"
4) "k1"
5) "animals"
6) "count"
7) "list01"
8) "Count"
127.0.0.1:6379> get hello
"world"
127.0.0.1:6379> get hello
(nil)
127.0.0.1:6379> ttl hello
(integer) -1
127.0.0.1:6379>
```

-1表示永不过期，-2表示已过期
如果过期get与keys *得不到

- 设置键过期时间：expire, key, seconds

请阅读帮助后输入语句

请输入

```
1 expire, hello, 10
```

运行

结果:
string(1) "1"

```
3) "foo"
4) "k1"
5) "animals"
6) "count"
7) "list01"
8) "Count"
127.0.0.1:6379> get hello
"world"
127.0.0.1:6379> get hello
(nil)
127.0.0.1:6379> ttl hello
(integer) -1
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> ttl hello
(integer) 7
127.0.0.1:6379>
```

本地设置十秒的过期时间，但是终端显示7秒，原因：
因为运行开始便开始计时， 等我输入终端已经过去了三秒
十秒过去后：ttl显示为-2表示已经过期

请阅读帮助后输入语句

请输入

1 expire, hello, 10

结果:
string(1) "1"

bin bin bin +

5) "animals"
6) "count"
7) "list01"
8) "Count"
127.0.0.1:6379> get hello
"world"
127.0.0.1:6379> get hello
(nil)
127.0.0.1:6379> ttl hello
(integer) -1
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> ttl hello
(integer) 7
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379>

7.string（单值单value）

- 查看字符串长度：strlen， key

请阅读帮助后输入语句

请输入

1 strlen, k1

运行

结果:
string(2) "12"

bin bin bin +

7) "list01"
8) "Count"
127.0.0.1:6379> get hello
"world"
127.0.0.1:6379> get hello
(nil)
127.0.0.1:6379> ttl hello
(integer) -1
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> ttl hello
(integer) 7
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> get k1
"hellostring"
127.0.0.1:6379>

- 在字符串后面添加字符串：append， key， value

PRedis version-1.0.1

请阅读帮助后输入语句

请输入

1 append, k1, hello

运行

结果:
string(2) "17"

bin bin bin +

127.0.0.1:6379> get hello
"world"
127.0.0.1:6379> get hello
(nil)
127.0.0.1:6379> ttl hello
(integer) -1
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> ttl hello
(integer) 7
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> get k1
"helllostring"
127.0.0.1:6379> get k1
"helllostringhello"
127.0.0.1:6379>

>注意看终端，一开始是helllostring，添加后变成helllostringhello

- 数字的自增：incr, key

请阅读帮助后输入语句

请输入

1 incr, count

运行

结果:
string(1) "4"

bin bin bin +

127.0.0.1:6379> ttl hello
(integer) -1
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> ttl hello
(integer) 7
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> get k1
"helllostring"
127.0.0.1:6379> get k1
"helllostringhello"
127.0.0.1:6379> get count
"3"
127.0.0.1:6379> get count
"4"
127.0.0.1:6379>

注意看终端，一开始是3，添加后变成4

- 数字的自增：dncr, key

请阅读帮助后输入语句

请输入

1 decr, count

运行

结果:
string(1) "3"

```
bin bin bin +
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> ttl hello
(integer) 7
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> get k1
"hellostring"
127.0.0.1:6379> get k1
"hellostringhello"
127.0.0.1:6379> get count
"3"
127.0.0.1:6379> get count
"4"
127.0.0.1:6379> get count
"3"
127.0.0.1:6379>
```

注意看终端，一开始是4，添加后变成3

- 设置键过期时间：setex, key, seconds, value

请阅读帮助后输入语句

请输入

1 setex, k1, 20, 1

运行

结果:
string(2) "OK"

```
bin bin bin +
127.0.0.1:6379> ttl hello
(integer) -2
127.0.0.1:6379> get k1
"hellostring"
127.0.0.1:6379> get k1
"hellostringhello"
127.0.0.1:6379> get count
"3"
127.0.0.1:6379> get count
"4"
127.0.0.1:6379> get count
"3"
127.0.0.1:6379> get k1
"1"
127.0.0.1:6379> ttl k1
(integer) 10
127.0.0.1:6379>
```

本地设置二十秒的过期时间，但是终端显示十秒，原因：
因为运行开始便开始计时，等我输入终端已经过去了十秒
二十秒过去后：ttl显示为-2表示已经过期

请阅读帮助后输入语句

请输入

1 setex, k1, 20, 1

运行

结果:

string(2) "OK"

bin bin bin +

127.0.0.1:6379> get k1
"hellostring"
127.0.0.1:6379> get k1
"hellostringhello"
127.0.0.1:6379> get count
"3"
127.0.0.1:6379> get count
"4"
127.0.0.1:6379> get count
"3"
127.0.0.1:6379> get k1
"1"
127.0.0.1:6379> ttl k1
(integer) 10
127.0.0.1:6379> ttl k1
(integer) -2
127.0.0.1:6379>

- 设置键：setnx, key, value

set if not exist, 如果不存在则设置值返回1, 存在返回0

当key不存在返回的情况:1

请阅读帮助后输入语句

请输入

1 setnx, k1, v1

运行

结果:

string(1) "1"

bin bin bin +

127.0.0.1:6379> get k1
"hellostringhello"
127.0.0.1:6379> get count
"3"
127.0.0.1:6379> get count
"4"
127.0.0.1:6379> get count
"3"
127.0.0.1:6379> get k1
"1"
127.0.0.1:6379> ttl k1
(integer) 10
127.0.0.1:6379> ttl k1
(integer) -2
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379>

当key存在返回的情况：0

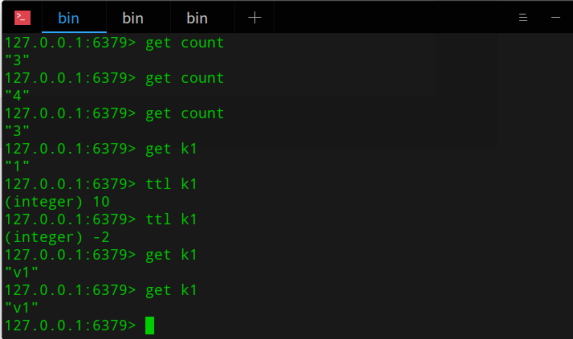
请阅读帮助后输入语句

请输入

1 setnx, k1, v1

运行

结果:
string(1) "0"



- 设置多个值：mset, key1, value1, key2, value2, key3, value3...

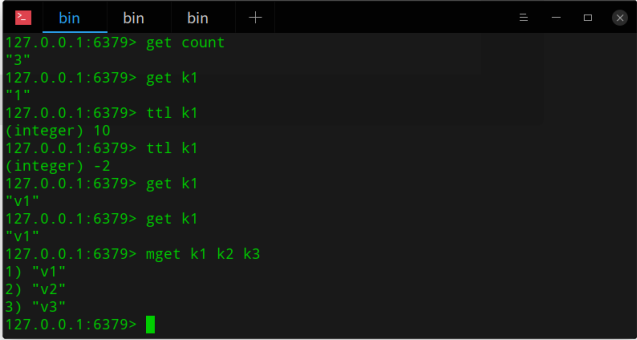
请阅读帮助后输入语句

请输入

1 mset, k1, v1, k2, v2, k3, v3

运行

结果:
string(2) "OK"



- 获得多个值：mget, key1, key2, key3...

请阅读帮助后输入语句

请输入

```
1 mget, k1, k2, k3
```

运行

结果:
array(3) { [0]=> string(2) "v1" [1]=> string(2) "v2" [2]=> string(2) "v3" }

```
bin bin bin +
127.0.0.1:6379> get count
"3"
127.0.0.1:6379> get k1
"1"
127.0.0.1:6379> ttl k1
(integer) 10
127.0.0.1:6379> ttl k1
(integer) -2
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379> get k1
"v1"
127.0.0.1:6379> mget k1 k2 k3
1) "v1"
2) "v2"
3) "v3"
127.0.0.1:6379>
```

8.list（单值多value）

- 插入：lpush/rpush，list，value
- 下面演示全部以lpush为例，rpush与lpush相反，具体请参考redis语法

请阅读帮助后输入语句

请输入

```
1 lpush, list03, 1, 2, 3, 4, 5
```

运行

结果:
string(1) "5"

```
bin bin bin +
2) "v2"
3) "v3"
127.0.0.1:6379> lrange list01 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
6) "2,3,4"
7) "1 2 3 4 5 "
127.0.0.1:6379> lrange list03 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379>
```

- 输出：lrange，list，0， -1

请阅读帮助后输入语句

请输入

运行

```
1 lrange, list03, 0, -1
```

结果:

```
array(5) {[0]=> string(1) "5" [1]=> string(1) "4" [2]=> string(1) "3" [3]=> string(1) "2" [4]=> string(1) "1" }
```



```
2) "v2"
3) "v3"
127.0.0.1:6379> lrange list01 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
6) "2,3,4"
7) "1 2 3 4 5 "
127.0.0.1:6379> lrange list03 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379>
```

- pop弹出一个：lpop/rpop, list
下面演示全部以lpop为例，rpop与lpop相反，具体请参考redis语法

请阅读帮助后输入语句

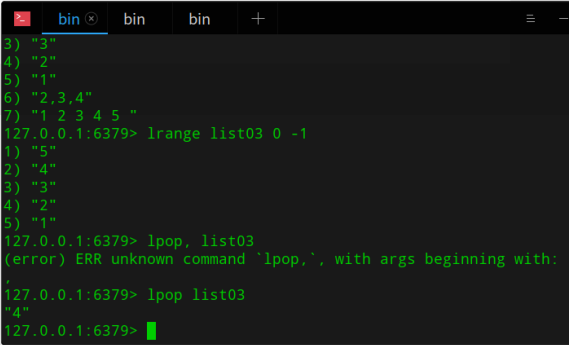
请输入

运行

```
1 lpop, list03
```

结果:

```
string(1) "5"
```



```
3) "3"
4) "2"
5) "1"
6) "2,3,4"
7) "1 2 3 4 5 "
127.0.0.1:6379> lrange list03 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379> lpop, list03
(error) ERR unknown command 'lpop.', with args beginning with: 'list03'
127.0.0.1:6379> lpop list03
"4"
127.0.0.1:6379>
```

为什么弹出与终端不一样？
因为5在网页已经pop，在终端是pop下一个4

- 取第x个：lindex, list, x

请阅读帮助后输入语句

请输入

1 lindex, list03, 2

运行

结果:
string(1) "1"

```
5) "1"
6) "2,3,4"
7) "1 2 3 4 5 "
127.0.0.1:6379> lrange list03 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379> lpop, list03
(error) ERR unknown command 'lpop.', with args beginning with: 'list03'
127.0.0.1:6379> lpop list03
"4"
127.0.0.1:6379> lindex list03 2
"1"
127.0.0.1:6379>
```

- 求list的长度：llen, list

请阅读帮助后输入语句

请输入

1 llen, list03

运行

结果:
string(1) "3"

```
7) "1 2 3 4 5 "
127.0.0.1:6379> lrange list03 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379> lpop, list03
(error) ERR unknown command 'lpop.', with args beginning with: 'list03'
127.0.0.1:6379> lpop list03
"4"
127.0.0.1:6379> lindex list03 2
"1"
127.0.0.1:6379> llen list03
(integer) 3
127.0.0.1:6379>
```

- 将list第x位设置成值：lset, list, x, value

注意列表是从0开始的

请阅读帮助后输入语句

请输入

```
1 lset, list04, 2, 6
```

运行

结果:

string(2) "OK"

bin bin bin +

```
4) "2"
5) "1"
6) "2,3,4"
7) "1 2 3 4 5 "
127.0.0.1:6379> lrange list04 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
127.0.0.1:6379> lrange list04 0 -1
1) "5"
2) "4"
3) "6"
4) "2"
5) "1"
127.0.0.1:6379>
```

- 值前插：linsert, list, before, value1, value2

请阅读帮助后输入语句

请输入

```
1 linsert, list04, before, 6, 7
```

运行

结果:

string(1) "6"

bin bin bin +

```
3) "3"
4) "2"
5) "1"
127.0.0.1:6379> lrange list04 0 -1
1) "5"
2) "4"
3) "6"
4) "2"
5) "1"
127.0.0.1:6379> lrange list04 0 -1
1) "5"
2) "4"
3) "7"
4) "6"
5) "2"
6) "1"
127.0.0.1:6379>
```

- 值后插：linsert, list, after, value1, value2

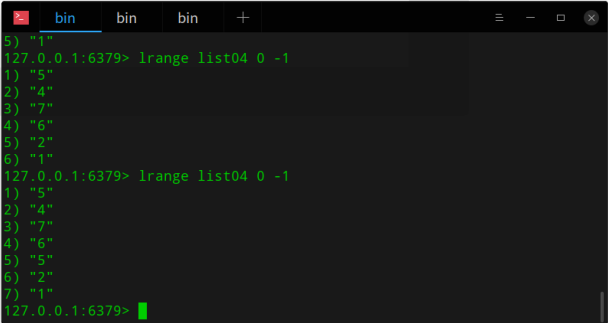
请阅读帮助后输入语句

请输入

1 linsert, list04, after, 6, 5

运行

结果:
string(1) "7"



8.set（单值多集合）

与list的区别是不能插入重复值

- 插入：sadd, set, value1, value2, value3...
当插入没有重复

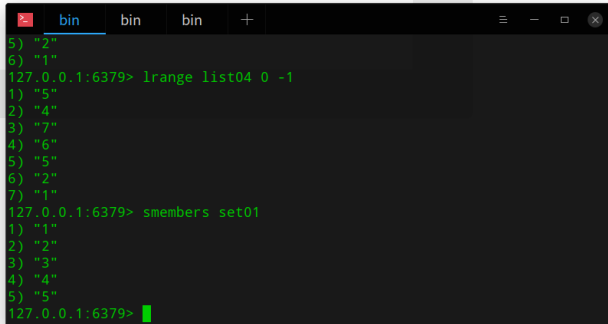
请阅读帮助后输入语句

请输入

1 sadd, set01, 1, 2, 3, 4, 5

运行

结果:
string(1) "5"



当插入重复时自动去掉重复

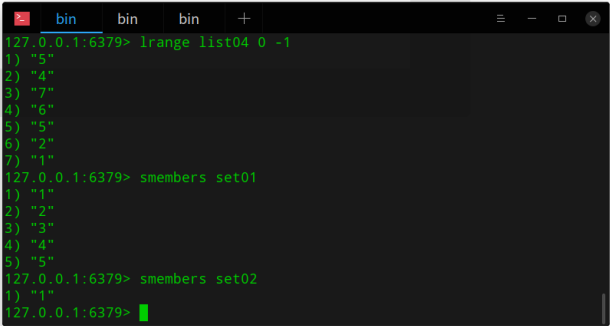
请阅读帮助后输入语句

请输入

```
1 sadd, set02, 1, 1, 1, 1, 1
```

运行

结果:
string(1) "1"



- 查看集合：smembers, set

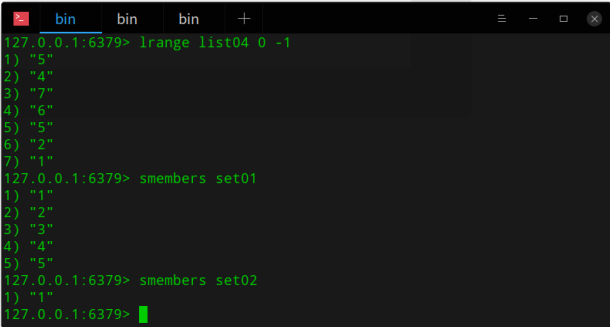
请阅读帮助后输入语句

请输入

```
1 smembers, set01
```

运行

结果:
array(5) { [0]=> string(1) "1" [1]=> string(1) "2" [2]=> string(1) "3" [3]=> string(1) "4" [4]=> string(1) "5" }



- 查看set里面有几个值：scard, set

请阅读帮助后输入语句

请输入

1

scard, set01

运行

结果:
string(1) "5"

- 删除set里面的值：srem, set, value

删除3，下面演示是否存在3

请阅读帮助后输入语句

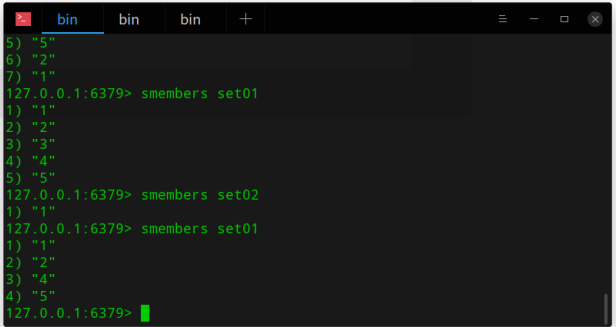
请输入

1

srem, set01, 3

运行

结果:
string(1) "1"



- 判断集合里面是否存在值：sismember, set, value
当值存在返回1

请阅读帮助后输入语句

请输入

1

sismember, set01, 4

运行

结果:
string(1) "1"

当值不存在返回0

请阅读帮助后输入语句

请输入

1

sismember, set01, 3

运行

结果:
string(1) "0"

- 从集合中随机返回x个随机数：srandmember, set, x

请阅读帮助后输入语句

请输入

1

srandmember, set01, 2

运行

结果:
array(2) { [0]=> string(1) "4" [1]=> string(1) "2" }

- 从集合中pop一个：spop， set

请阅读帮助后输入语句

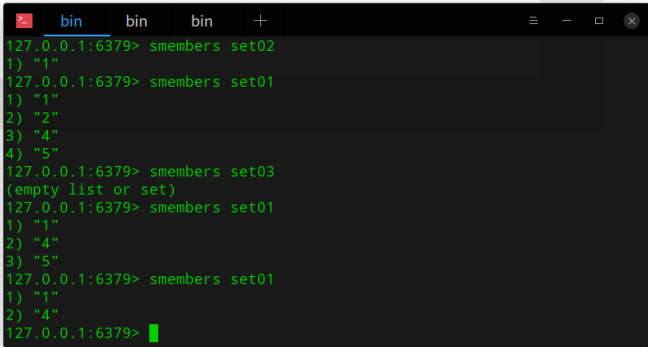
请输入

1

spop, set01

运行

结果:
string(1) "5"



- 从集合1中第x个移到集合2：smove， set01， set02， x

set04一开始只有7,8,9；从set01移入第一个

请阅读帮助后输入语句

请输入

1 smove, set01, set04, 1

运行

结果:
string(1) "1"

bin bin bin +

3) "4"
4) "5"
127.0.0.1:6379> smembers set03
(empty list or set)
127.0.0.1:6379> smembers set01
1) "1"
2) "4"
3) "5"
127.0.0.1:6379> smembers set01
1) "1"
2) "4"
127.0.0.1:6379> smembers set04
1) "1"
2) "7"
3) "8"
4) "9"
127.0.0.1:6379> []

- 求两个几个的交叉并集：
 - 交集：sdiff, set01, set02
 - 差集：sinter, set01, set02
 - 并集：sunion, set01, set02下面只演示并集

请阅读帮助后输入语句

请输入

1 sunion, set01, set04

运行

结果:
array(5) { [0]=> string(1) "1" [1]=> string(1) "4" [2]=> string(1) "7" [3]=> string(1) "8" [4]=> string(1) "9" }

bin bin bin +

127.0.0.1:6379> smembers set03
(empty list or set)
127.0.0.1:6379> smembers set01
1) "1"
2) "4"
3) "5"
127.0.0.1:6379> smembers set01
1) "1"
2) "4"
127.0.0.1:6379> smembers set04
1) "1"
2) "7"
3) "8"
4) "9"
127.0.0.1:6379> smembers set01
1) "4"
127.0.0.1:6379> █

9. hash（KV模式不变，但V是一个键值对）

- 为散列设置值：hmset, name, key1, value1, key2, value2, key3, value3...

请阅读帮助后输入语句

请输入

运行

1

hmset, people, name, aa, age, 1, id, 1|

结果:
string(2) "OK"

- 查询：hmget, name, key1, key2, key3

请阅读帮助后输入语句

请输入

运行

1

hmget, people, id, name, age

结果:
array(3) { [0]=> string(1) "1" [1]=> string(2) "aa" [2]=> string(1) "1" }

- 查看所有key：hkeys, name

请阅读帮助后输入语句

请输入

1

hkeys, people

运行

结果:
array(3) { [0]=> string(4) "name" [1]=> string(3) "age" [2]=> string(2) "id" }

- 查看所有value：hvals, name

请阅读帮助后输入语句

请输入

1

hvals, people|

运行

结果:
array(3) { [0]=> string(2) "aa" [1]=> string(1) "1" [2]=> string(1) "1" }

- 查询所有值：hgetall, name

请阅读帮助后输入语句

请输入

运行

1 hgetall, people

结果:
array(6) { [0]=> string(4) "name" [1]=> string(2) "aa" [2]=> string(3) "age" [3]=> string(1) "1" [4]=> string(2) "id" [5]=> string(1) "1" }

- 删除某个key: hdel, name, key

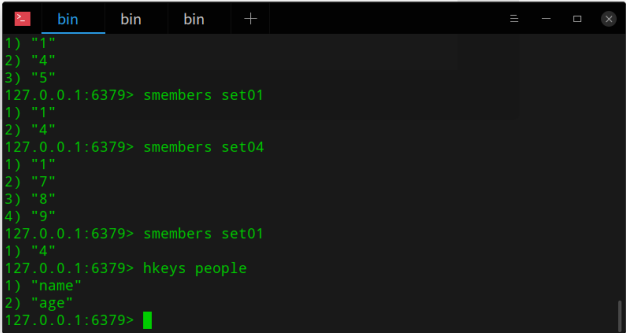
请阅读帮助后输入语句

请输入

运行

1 hdel, people, id

结果:
string(1) "1"



- 判断是否有某个key: hexists, name, key
存在返回1

请阅读帮助后输入语句

请输入

运行

1

hexists, people, name

结果:
string(1) "1"

不存在返回0

请阅读帮助后输入语句

请输入

运行

1

hexists, people, id

结果:
string(1) "0"

10.zset：参考set