* TRY REDIS *

Welcome to **Try Redis**, a demonstration of the Redis database!

Please type TUTORIAL to begin a brief tutorial, HELP to see a list of supported commands, or any valid Redis command to play with the database.

```
> HELP
```

Please type HELP for one of these commands: DECR, DECRBY, DEL, EXISTS, EXPIRE, GET, GETSET, HDEL, HEXISTS, HGET, HGETALL, HINCRBY, HKEYS, HLEN, HMGET, HMSET, HSET, HVALS, INCR, INCRBY, KEYS, LINDEX, LLEN, LPOP, LPUSH, LRANGE, LREM, LSET, LTRIM, MGET, MSET, MSETNX, MULTI, PEXPIRE, RENAME, RENAMENX, RPOP, RPOPLPUSH, RPUSH, SADD, SCARD, SDIFF, SDIFFSTORE, SET, SETEX, SETNX, SINTER, SINTERSTORE, SISMEMBER, SMEMBERS, SMOVE, SORT, SPOP, SRANDMEMBER, SREM, SUNION, SUNIONSTORE, TTL, TYPE, ZADD, ZCARD, ZCOUNT, ZINCRBY, ZRANGE, ZRANGEBYSCORE, ZRANK, ZREM, ZREMRANGEBYSCORE, ZREVRANGE, ZSCORE

```
> TUTORIAL
```

Redis is what is called a key-value store, often referred to as a NoSQL database. The essence of a key-value store is the ability to store some data, called a value, inside a key. This data can later be retrieved only if we know the exact key used to store it. We can use the command SET to store the value "fido" at key "server:name":

```
SET server:name "fido"
```

字符串值的结构：<key, value>

永久地储存数据

Redis will store our data permanently, so we can later ask "What is the value stored at key server:name?" and Redis will reply with "fido":

```
GET server:name => "fido"
```

Tip: You can click the commands above to automatically execute them. The text after the arrow (=>) shows the expected output.

Type NEXT to continue the tutorial.

```
> SET server:name "fido"
OK
> GET server:name
```

批量获取：MGET

```
"fido"
```

```
> NEXT
```

Other common operations provided by key-value stores are DEL to delete a given key and associated value, SET-if-not-exists (called SETNX on Redis) that sets a key only if it does not already exist, and INCR to atomically increment a number stored at a given key:

原子地增加数字

```
SET connections 10
INCR connections => 11
INCR connections => 12
DEL connections
INCR connections => 1
```

Type NEXT to continue the tutorial.

```
> SET connections 10
OK
> INCR connections
(integer) 11
```

INCRBY、INCRBYFLOAT：INCR、DECRBY、DECR
INCRBY使用案例：计分器/计数器/自增序列

```
> INCR connections
(integer) 12
> DEL connections
(integer) 1
> INCR connections
(integer) 1
> SETNX connections 10
(integer) 0
> GET connections
"1"
> DEL connections
(integer) 1
> SETNX connections 10
```

SETNX+EXPIRE/SET使用案例：分布式锁

```
(integer) 1
> GET connections
```

```
"10"

> NEXT
```

There is something special about INCR. Why do we provide such an operation if we can do it ourself with a bit of code? After all it is as simple as:

```
x = GET count
x = x + 1
SET count x
```

问题杀手：组合命令

The problem is that doing the increment in this way will only work as long as there is a single client using the key. See what happens if two clients are accessing this key at the same time:

并发更新问题

1. Client A reads *count* as 10.
2. Client B reads *count* as 10.
3. Client A increments 10 and sets *count* to 11.
4. Client B increments 10 and sets *count* to 11.

We wanted the value to be 12, but instead it is 11! This is because incrementing the value in this way is not an atomic operation. Calling the INCR command in Redis will prevent this from happening, because it *is* an atomic operation. Redis provides many of these atomic operations on different types of data. 所有单个操作的命令都是原子操作。

字符串值操作焦点：值/value、键/key
使用案例：存储原始数据、计分器、分布式锁

Type NEXT to continue the tutorial.

```
> NEXT
```

Key的生存时间/过期时间

Redis can be told that a key should only exist for a certain length of time. This is accomplished with the EXPIRE and TTL commands.

time to live

```
SET resource:lock "Redis Demo"
EXPIRE resource:lock 120
```

时间单位：秒/毫秒

This causes the key *resource:lock* to be deleted in 120 seconds. You can test how long a key will exist with the TTL command. It returns the number of seconds until it will be deleted.

```
TTL resource:lock => 113
// after 113s
TTL resource:lock => -2
```

The *-2* for the TTL of the key means that the key does not exist (anymore). A *-1* for the TTL of the key means that it will never expire. Note that if you SET a key, its TTL will be reset.

永不过期

Key：持久的(persistent) vs 易失的(volatile)

```
SET resource:lock "Redis Demo 1"
EXPIRE resource:lock 120
TTL resource:lock => 119
```

建议：所有键都要设置生存时间！

```
    SET resource:lock "Redis Demo 2"
    TTL resource:lock => -1
```

键操作焦点：key
使用案例：数据过期、持久化

Type NEXT to continue the tutorial.

```
> SET resource:lock "Redis Demo"

OK

> EXPIRE resource:lock 20

(integer) 1

> TTL resource:lock

(integer) 4

> TTL resource:lock

(integer) -2

> SET resource:lock "Redis Demo 1"

OK

> EXPIRE resource:lock 120

(integer) 1

> TTL resource:lock

(integer) 117

> SET resource:lock "Redis Demo 2"
```
MSET：丢失生存时间(禁用)
```
OK

> TTL resource:lock

(integer) -1

> NEXT
```

有序列表：一系列有序的值。（结构：<key, [value]>）

Redis also supports several more complex data structures. The first one we'll look at is a list. A list is a series of ordered values. Some of the important commands for interacting with lists are RPUSH, LPUSH, LLEN, LRANGE, LPOP, and RPOP. You can immediately begin working with a key as a list, as long as it doesn't already exist as a different type.

DEL friends

RPUSH puts the new value at the end of the list.

列表的尾部/表尾（最右边）

```
    RPUSH friends "Alice"
    RPUSH friends "Bob"
```

LPUSH puts the new value at the start of the list.
列表的头部/表头（最左边）

```
LPUSH friends "Sam"
```

子列表（范围查询：分页的最爱）

LRANGE gives a subset of the list. It takes the index of the first element you want to retrieve as its first parameter and the index of the last element you want to retrieve as its second parameter. A value of -1 for the second parameter means to retrieve elements until the end of the list.

```
LRANGE friends 0 -1 => 1) "Sam", 2) "Alice", 3) "Bob"
LRANGE friends 0 1 => 1) "Sam", 2) "Alice"
LRANGE friends 1 2 => 1) "Alice", 2) "Bob"
```

Type NEXT to continue the tutorial.

```
> DEL friends

(integer) 1

> RPUSH friends "Alice"
```

新的空列表被自动创建

```
(integer) 1

> RPUSH friends "Bob"

(integer) 2

> LPUSH friends "Sam"

(integer) 3

> LRANGE friends 0 -1

1) "Sam"
2) "Alice"
3) "Bob"

> LRANGE friends 0 1
```

闭区间：[start, stop]

```
1) "Sam"
2) "Alice"

> LRANGE friends 1 2

1) "Alice"
2) "Bob"

> NEXT
```

LLEN returns the current length of the list.

```
        LLEN friends => 3
```

LPOP removes the first element from the list and returns it.
头元素

```
        LPOP friends => "Sam"
```

RPOP removes the last element from the list and returns it.
尾元素

```
        RPOP friends => "Bob"
```

Note that the list now only has one element:

```
        LLEN friends => 1
        LRANGE friends 0 -1 => 1) "Alice"
```

有序列表值操作焦点：在头部/尾部进行操作(双端链表的优势)
使用案例：分页查询/建二级索引、可靠队列

Type NEXT to continue the tutorial.

```
> LLEN friends
(integer) 3
> LPOP friends
"Sam"
```

阻塞版：BRPOP/BLPOP key timeout

```
> RPOP friends
"Bob"
```

RPOPLPUSH/BRPOPLPUSH：可靠队列、循环列表

```
> LLEN friends
(integer) 1
> LRANGE friends 0 -1
1) "Alice"
> NEXT
```

集合：类似于列表，是一系列无序、唯一的值。（结构：<key, {member}>）

The next data structure that we'll look at is a set. A set is similar to a list, except it does not have a specific order and each element may only appear once. Some of the important commands in working with sets are SADD, SREM, SISMEMBER, SMEMBERS and SUNION.(并集)、SINTER(交集)、SDIFF(差集)

SADD adds the given value to the set.

```
        SADD superpowers "flight"
        SADD superpowers "x-ray vision"
        SADD superpowers "reflexes"
```

SREM removes the given value from the set.

```
        SREM superpowers "reflexes"
```

Type NEXT to continue the tutorial.

```
> SADD superpowers "flight"
```

新的空集合被自动创建

```
(integer) 1

> SADD superpowers "x-ray vision"

(integer) 1

> SADD superpowers "reflexes"

(integer) 1

> SREM superpowers "reflexes"

1

> NEXT
```

断言命令：为什么使用整数作为返回值，而不使用true/false?

SISMEMBER tests if the given value is in the set. It returns *1* if the value is there and *0* if it is not.

```
        SISMEMBER superpowers "flight" => 1
        SISMEMBER superpowers "reflexes" => 0
```

SMEMBERS returns a list of all the members of this set.

```
        SMEMBERS superpowers => 1) "flight", 2) "x-ray vision"
```

SUNION combines two or more sets and returns the list of all elements.

```
        SADD birdpowers "pecking"
        SADD birdpowers "flight"
        SUNION superpowers birdpowers => 1) "pecking", 2) "x-ray vision", 3) "flight"
```

集合值操作焦点：唯一性的成员/member
使用案例：数据去重、共同子集

Type NEXT to continue the tutorial.

```
> SISMEMBER superpowers "flight"

(integer) 1

> SISMEMBER superpowers "reflexes"

(integer) 0

> SMEMBERS superpowers

1) "flight"
2) "x-ray vision"

> SADD birdpowers "pecking"
```

```
(integer) 1

> SADD birdpowers "flight"

(integer) 1

> SUNION superpowers birdpowers

1) "flight"
2) "pecking"
3) "x-ray vision"

> NEXT
```

引入有序集合的缘由：因为集合是无序的，它不适用于若干问题。

Sets are a very handy data type, but as they are unsorted they don't work well for a number of problems. This is why Redis 1.2 introduced Sorted Sets.

有序集合：类似于常规的集合，每个值都有一个关联的分数，其中分数用于对元素进行排序。

A sorted set is similar to a regular set, but now each value has an associated score. This score is used to sort the elements in the set.

（结构：<key, {<member, score>}>）

```
ZADD hackers 1940 "Alan Kay"
ZADD hackers 1906 "Grace Hopper"
ZADD hackers 1953 "Richard Stallman"
ZADD hackers 1965 "Yukihiro Matsumoto"
ZADD hackers 1916 "Claude Shannon"
ZADD hackers 1969 "Linus Torvalds"
ZADD hackers 1957 "Sophie Wilson"
ZADD hackers 1912 "Alan Turing"
```

In these examples, the scores are years of birth and the values are the names of famous hackers.

```
ZRANGE hackers 2 4 => 1) "Claude Shannon", 2) "Alan Kay", 3) "Richard Stallman"
```

有序集合值操作焦点：唯一性元素的分数/score
使用案例：排序、Top N

Type NEXT to continue the tutorial.

```
> ZADD hackers 1940 "Alan Kay"
```

新的空有序集合被自动创建

```
(integer) 1

> ZADD hackers 1906 "Grace Hopper"

(integer) 1

> ZADD hackers 1953 "Richard Stallman"

(integer) 1

> ZADD hackers 1965 "Yukihiro Matsumoto"

(integer) 1

> ZADD hackers 1916 "Claude Shannon"
> ZADD hackers 1916 "Claude Shannon"

(integer) 1

> ZADD hackers 1969 "Linus Torvalds"

(integer) 1

> ZADD hackers 1957 "Sophie Wilson"

(integer) 1

> ZADD hackers 1912 "Alan Turing"

(integer) 1

> ZRANGE hackers 2 4
```

ZRANGE key start stop
ZRANGEBYSCORE key min max

```
1) "Claude Shannon"
2) "Alan Kay"
3) "Richard Stallman"

> NEXT
```

Simple strings, sets and sorted sets already get a lot done but there is one more data type Redis can handle: Hashes.

哈希表是字符串字段和字符串值之间的映射关系，是表示对象的最完美的数据类型。（结构：<key, {<field, value>}>）

Hashes are maps between string fields and string values, so they are the perfect data type to represent objects (eg: A User with a number of fields like name, surname, age, and so forth):

```
HSET user:1000 name "John Smith"
HSET user:1000 email "john.smith@example.com"
HSET user:1000 password "s3cret"
```

To get back the saved data use `HGETALL`:

```
HGETALL user:1000
```

You can also set multiple fields at once: 批量操作：HMSET/HMGET

```
HMSET user:1001 name "Mary Jones" password "hidden" email "mjones@example.com"
```

If you only need a single field value that is possible as well:

```
HGET user:1001 name => "Mary Jones"
```

Type NEXT to continue the tutorial.

```
> HSET user:1000 name "John Smith"
```

新的空哈希表自动被创建

```
> HSET user:1000 name "John Smith"

(integer) 0

> HSET user:1000 email "john.smith@example.com"
```

```
(integer) 1

> HSET user:1000 password "s3cret"

(integer) 1

> HGETALL user:1000

1) "name"
2) "John Smith"
3) "email"
4) "john.smith@example.com"
5) "password"
6) "s3cret"

> HMSET user:1001 name "Mary Jones" password "hidden" email
"mjones@example.com"

OK

> HGET user:1001 name

"Mary Jones"

> NEXT
```

Numerical values in hash fields are handled exactly the same as in simple strings and there are operations to increment this value in an atomic way.

```
HSET user:1000 visits 10
HINCRBY user:1000 visits 1 => 11
HINCRBY user:1000 visits 10 => 21
HDEL user:1000 visits
HINCRBY user:1000 visits 1 => 1
```

Check the full list of Hash commands for more information.

哈希表值操作焦点：唯一性的字段/field
使用案例：表示多维属性对象

Type NEXT to continue the tutorial.

```
> HSET user:1000 visits 10

(integer) 1

> HINCRBY user:1000 visits 1

(integer) 11

> HINCRBY user:1000 visits 10

(integer) 21
```

```
> HDEL user:1000 visits

(integer) 1

> HINCRBY user:1000 visits 1

(integer) 1

> NEXT
```

That wraps up the *Try Redis* tutorial. Please feel free to goof around with this console as much as you'd like.

Check out the following links to continue learning about Redis.

- Redis Documentation
- Command Reference
- Implement a Twitter Clone in Redis
- Introduction to Redis Data Types

>

This site was originally written by Alex McHale (github twitter) and inspired by Try Mongo. It's now maintained and hosted by Jan-Erik Rediger (github twitter)

The source code to Try Redis is available on GitHub.