# Redis互动教程（中文）

Author:Jason-wong

Url:<https://github.com/jason-wong/documents_translate>

### Intro

欢迎使用Redis，本文将带你演示Redis数据库的基本操作。

输入**TUTORIAL**开始一个简短的教程，或者输入**HELP**查看所有命令列表，还可以尝试输入任意有效的Redis命令了解和学习数据库。

* TUTORIAL

### Part 1

Redis就是所谓的key-value存储系统，通常被称为NOSQL数据库，key-value存储系统的本质就是能够存储一些数据，通过key设置值。当我们定义了key，那么即可在任何时候通过key来得到相应的值。我们可以使用**SET**命令设置一个key为“server:name”，value为“fido”的数据：

SET server:name "fido"

Redis会永久的存储我们的数据，所以我们之后可以在任何时候问Redis：“server:name中存储了什么值？”，然后Redis会回答——”fido”：

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

输入**NEXT**命令继续教程

### Part 2

还有一些其他的用来操作key-value的命令，比如 **DEL** 用来删除一个指定key的值，SET-if-not-exists（Redis中叫做**SETNX**）用来为一个不存在的key设定值，**INCR** 用来为数字实现自增：

SET connections 10

INCR connections *=> 11*

INCR connections *=> 12*

DEL connections

INCR connections *=> 1*

输入**NEXT**命令继续教程

### Part 3

关于**INCR** 的特性。我们其实可以通过很少的代码轻松实现数字的自增，为什么还要提供这样一个命令？比如这样：

x = GET count

x = x + 1

SET count x

问题在于这样的自增方法在只有一个客户端的情况下能够很好的工作，当有两个客户端在同一时间内做相同的操作会发生什么：

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.

我们希望值变为12，但实际上值为11。这是因为这样的自增方式不是原子运算，在Redis中使用**INCR** 命令可以防止这种情况发生，因为**INCR**实现了这种原子运算，Redis为不同的数据类型提供了许多这样的原子运算机制。

### Part 4

Redis能够为一个key数据定义一个有效期，可以通过**EXPIRE**和**TTL**命令来实现。

SET resource:lock "Redis Demo"

EXPIRE resource:lock 120

这将导致resource:lock 这个数据在120秒后被删除。你可以使用**TTL**命令来查看一个key的剩余有效期，**TTL**每次将会返回指定key生存期内的剩余秒数，直到它逾期后被删除。

TTL resource:lock => 113

TTL count => -1

上面演示中返回的-1代表这个key是永不过期的。需要注意的是，当你重新设置了key，则该key的**TTL**命令返回值将会被重置：

SET resource:lock "Redis Demo 1"

EXPIRE resource:lock 120

TTL resource:lock => 119

SET resource:lock "Redis Demo 2"

TTL resource:lock => -1

### Part 5

Redis还支持一些更复杂的数据结构。首先我们先来认识列表，列表包含一系列值，这里有一些重要的命令用来执行列表的交互操作：**RPUSH**, **LPUSH**, **LLEN**, **LRANGE**, **LPOP**, **RPOP**。你可以立即开始使用一个key设置列表，只要它们不是不同的类型。

**RPUSH** 将一个新值压入列表的末尾（入栈）

RPUSH friends "Tom"

RPUSH friends "Bob"

**LPUSH** 将一个新值插入到列表开头

LPUSH friends "Sam"

**LRANGE** 取出列表中的子集。第一个参数设置为子集在列表中起始位置的索引，第二个参数设置为子集在列表中结束位置的索引，系统会返回列表中从开始索引到结束索引中间的所有元素（包含起止索引位置上的这两个元素），当第二个参数设置为-1时，会取出列表中从第一个参数索引位置开始直到列表最后一个值（包含）中间的所有元素。

LRANGE friends 0 -1 => ["Sam","Tom","Bob"]

LRANGE friends 0 1 => ["Sam","Tom"]

LRANGE friends 1 2 => ["Tom","Bob"]

### Part 6

**LLEN** 返回列表的长度

LLEN friends => 3

**LPOP** 移除列表中的第一个元素并将其返回

LPOP friends => "Sam"

**RPOP** 移除列表中的最后一个元素并将其返回

RPOP friends => "Bob"

列表现在只剩下了一个元素

LLEN friends => 1

LRANGE friends 0 -1 => ["Tom"]

### Part 7

下面认识的数据结构是集合，集合除了没有特定的顺序并且每个元素只出现一次，基本和列表类似。集合的一些重要命令：**SADD**, **SREM**, **SISMEMBER**, **SMEMBERS**, **SUNION**

**SADD** 为集合增加一个值

SADD superpowers "flight"

SADD superpowers "x-ray vision"

SADD superpowers "reflexes"

**SREM** 从集合中移除一个值

SREM superpowers "reflexes"

**SISMEMBER** 检测值是否存在于集合中

SISMEMBER superpowers "flight" => true

SISMEMBER superpowers "reflexes" => false

**SMEMBERS** 将集合中的所有成员作为列表返回

SMEMBERS superpowers => ["flight","x-ray vision"]

**SUNION** 合并两个或更多的集合并将所有元素作为列表返回（重复的值会被合并为一个）

SADD birdpowers "pecking"

SADD birdpowers "flight"

SUNION superpowers birdpowers => ["flight","x-ray vision","pecking"]

### Part 8

Redis最后一个数据结构是有序集合。和普通集合类似，但它的每一个值都拥有一个关联标记，这个标记用来排序集合中的元素。

ZADD hackers 1940 "Alan Kay"

ZADD hackers 1953 "Richard Stallman"

ZADD hackers 1965 "Yukihiro Matsumoto"

ZADD hackers 1916 "Claude Shannon"

ZADD hackers 1969 "Linus Torvalds"

ZADD hackers 1912 "Alan Turing"

上面列子中定义了一个包含一些著名人物的有序集合，集合中关联标记是每个人的出生年份，而值为每个人的名字。

ZRANGE hackers 2 4 => ["Alan Kay","Richard Stallman","Yukihiro Matsumoto"]

以上就是Redis的互动教程，你可以在控制台中随意尝试一些你的想法。

或者可以通过以下链接继续学习Redis。

Redis Documentation http://redis.io/documentation

Command Reference http://redis.io/commands

Implement a Twitter Clone in Redis http://redis.io/topics/twitter-clone

Introduction to Redis Data Types http://redis.io/topics/data-types-intro