Redis是一款内存高速缓存数据库，全称Remote Dictionary Server（远程数据服务），使用C语言编写，是一个key-value存储系统，支持丰富的数据类型，如：string、list、set、zset(sorted set)、hash。

**string类型**

- 赋值：`set key value`

```
127.0.0.1:6379> set test 123
OK
```

- 取值：`get key`

```
127.0.0.1:6379> get test
"123"
```

- 取值并赋值：`getset key value`

```
127.0.0.1:6379> getset test 321
"123"
127.0.0.1:6379> get test
"321"
```

- 设置获取多个键值：`mset key value [key value...]`

```
127.0.0.1:6379> mset k1 v1 k2 v2 k3 v3
OK
```

- 获取多个值：`mget key [key...]`

```
127.0.0.1:6379> mget k1 k2
1) "v1"
2) "v2"
```

- 删除：`del key`。

```
127.0.0.1:6379> del test
(integer) 1
```

- 递增：当存储的字符串是整数时，可通过命令INCR让当前键值递增，并返回递增后的值。语法是`incr key`。

```
127.0.0.1:6379> set num 1
OK
127.0.0.1:6379> incr num
(integer) 2
```

- 增加指定的整数：`incrby key increment`。

```
127.0.0.1:6379> incrby num 2
(integer) 4
```

- 递减数值：`decr key`。

```
127.0.0.1:6379> decr num
(integer) 3
```

- 减少指定的数值：`decrby key decrement`。

```
127.0.0.1:6379> decrby num 2
(integer) 1
```

- 向尾部追加值 APPEND的作用是向键值的末尾追加value。如果键不存在则将该键的值设置为value，即相当于 SET key value。返回值是追加后字符串的总长度。 语法：append key value。

```
127.0.0.1:6379> set str hello
OK
127.0.0.1:6379> append str "world"
(integer) 10
127.0.0.1:6379> get str
"helloworld"
```

- 获取字符串长度 STRLEN命令返回键值的长度，如果键不存在则返回0。 语法：strlen key。

```
127.0.0.1:6379> strlen str
(integer) 10
```

**Hash散列类型**

hash叫散列类型，它提供了字段和字段值的映射。字段值只能是字符串类型，不支持散列类型、集合类型等其它类型。

- 赋值`HSET`命令不区分插入和更新操作，当执行插入操作时HSET命令返回1，当执行更新操作时返回0。

  一次只设置一个字段值 语法：`hset key field value`。

```
127.0.0.1:6379> hset user username zhangsan
(integer) 1
```

一次设置多个字段值 语法：`hmset key field value [field value...]`。

```
127.0.0.1:6379> hmset user age 20 username lisi
OK
```

当字段不存在时赋值，类似hset,区别在于如果字段存在，该命令不执行任何操作。 语法：`hsetnx key field value`。

```
127.0.0.1:6379> hsetnx user age 30
(integer) 0
```

- 取值

  一次获取一个字段值 语法：`hget key field`。

```
127.0.0.1:6379> hget user username
"lisi"
```

  一次可以获取多个字段值 语法：`hmget key field [field...]`。

```
127.0.0.1:6379> hmget user age username
1) "20"
2) "lisi"
```

  获取所有字段值 语法：`hgetall key`。

```
127.0.0.1:6379> hgetall user
1) "username"
2) "lisi"
3) "age"
4) "20"
```

- 删除字段 可以删除一个或多个字段，返回值是被删除的字段的个数。 语法：`hdel key field [field...]`。

```
127.0.0.1:6379> hdel user age
(integer) 1
127.0.0.1:6379> hdel user age username
(integer) 1
```

- 增加数字 语法：`hincrby key field increment`。

```
127.0.0.1:6379> hincrby user age 2
(integer) 2
```

- 判断字段是否存在 语法：`hexists key field`。

```
127.0.0.1:6379> hexists user age
(integer) 1
```

- 只获取字段名或字段值 语法： `hkeys key hvals key`。

```
127.0.0.1:6379> hkeys user
1) "age"
```

- 只获取字段名或字段值 语法： `hkeys key`、`hvals key`。

```
127.0.0.1:6379> hkeys user
1) "age"
```

- 获取字段数量 语法：`hlen key`。

```
127.0.0.1:6379> hlen user
(integer) 1
```

**List类型**

Redis的list是采用来链表来存储的，所以对于redis的list数据类型的操作，是操作list的两端数据来操作的。

- 向列表两端增加元素

  向列表左边增加元素 语法：`lpush key value [value...]`。

```
127.0.0.1:6379> lpush list:1 1 2 3
(integer) 3
```

  向列表右边增加元素 语法：`rpush key value [value...]`。

```
127.0.0.1:6379> rpush list:1 4 5 6
(integer) 6
```

- 查看列表 LRANGE命令是列表类型最常用的命令之一，获取列表中的某一片段，将返回start、stop之间的所有元素（包含两端的元素），索引从0开始。索引可以是负数，如：“-1”代表最后边的一个元素。语法：`lrange key start stop`。

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

- 从列表两端弹出元素 LPOP命令从列表左边弹出一个元素，会分两步完成：第一步是将列表左边的元素从列表中移除；第二步是返回被移除的元素值。 语法为：`lpop key`、`rpop key`。

```
127.0.0.1:6379> lpop list:1
"3"
127.0.0.1:6379> rpop list:1
"6"
```

- 获取列表中元素的个数 语法：`llen key`。

```
127.0.0.1:6379> llen list:1
(integer) 4
```

- 删除列表中指定的值 LREM命令会删除列表中前count个值为value的元素，返回实际删除的元素个数。根据count值的不同，该命令的执行方式会有所不同：当count>0时， LREM会从列表左边开始删除；当count<0时， LREM会从列表后边开始删除；当count=0时， LREM删除所有值为value的元素。语法：`lrem key count value`。
- 获得/设置指定索引的元素值

  获得指定索引的元素值 语法：`lindex key index`。

```
127.0.0.1:6379> lindex list:1 2
"4"
```

  设置指定索引的元素值 语法：`lset key index value`。

```
127.0.0.1:6379> lset list:1 2 2
OK
```

  只保留列表指定片段 指定范围和 lrange 一致 语法：`ltrim key start stop`。

```
127.0.0.1:6379> lrange list:1 0 -1
1) "2"
2) "1"
3) "2"
4) "5"
127.0.0.1:6379> ltrim list:1 0 2
OK
127.0.0.1:6379> lrange list:1 0 -1
1) "2"
2) "1"
3) "2"
```

- 向列表中插入元素：该命令首先会在列表中从左到右查找值为pivot的元素，然后根据第二个参数是BEFORE还是AFTER来决定将value插入到该元素的前面还是后面。 语法：`linsert key before | after pivot value`。

```
127.0.0.1:6379> lrange list:1 0 -1
1) "2"
2) "1"
3) "2"
127.0.0.1:6379> linsert list:1 after 1 9
(integer) 4
127.0.0.1:6379> lrange list:1 0 -1
1) "2"
2) "1"
3) "9"
4) "2"
```

- 将元素从一个列表转移到另一个列表 语法：`rpoplpush source destination`。

```
127.0.0.1:6379> lrange list:1 0 -1
1) "2"
2) "1"
3) "9"
4) "2"
127.0.0.1:6379> rpoplpush list:1 newlist
"2"
127.0.0.1:6379> lrange newlist 0 -1
1) "2"
127.0.0.1:6379> lrange list:1 0 -1
1) "2"
2) "1"
3) "9"
```

**set类型**

集合类型：无序、不可重复 列表类型：有序、可重复。

- 增加元素 语法：`sadd key member [member...]`。

```
127.0.0.1:6379> sadd set a b c
(integer) 3
127.0.0.1:6379> sadd set a
(integer) 0
```

- 删除元素，语法：`srem key member [member...]`。

```
127.0.0.1:6379> srem set c
(integer) 1
```

- 获得集合中的所有元素 语法：`smembers key`。

```
127.0.0.1:6379> smembers set
1) "b"
2) "a"
```

- 判断元素是否在集合中 语法：`sismember key member`。

```
127.0.0.1:6379> sismember set a
(integer) 1
127.0.0.1:6379> sismember set h
(integer) 0
```

- 集合的差集运算 A-B 属于 A 并且 不属于 B 的元素构成的集合，语法：`sdiff key [key...]`。

```
127.0.0.1:6379> sadd setA 1 2 3
(integer) 3
127.0.0.1:6379> sadd setB 2 3 4
(integer) 3
127.0.0.1:6379> sdiff setA setB
1) "1"
127.0.0.1:6379> sdiff setB setA
1) "4"
```

- 集合的交集运算 属于A且属于B的元素构成的集合。语法：`sinter key [key...]`。

```
127.0.0.1:6379> sinter setA setB
1) "2"
2) "3"
```

- 集合的并集运算 属于 A 或者 属于 B 的元素构成的集合，语法：`sunion key [key...]`。

```
127.0.0.1:6379> sunion setA setB
1) "1"
2) "2"
3) "3"
4) "4"
```

- 获得集合中元素的个数，语法：`scard key`。

```
127.0.0.1:6379> smembers setA
1) "1"
2) "2"
3) "3"
127.0.0.1:6379> scard setA
(integer) 3
```

- 从集合中弹出一个元素 注意:由于集合是无序的，所有spop命令会从集合中随机选择一个元素弹出， 语法：`spop key`。

```
127.0.0.1:6379> spop setA
"2"
```

**Sortedset 类型**

Sortedset 又叫 zset Sortedset 是有序集合，可排序的，但是唯一。 Sortedset 和 set 的不同之处，会给 set 中元素添加一个分数，然后通过这个分数进行排序。

- 增加元素：向有序集合中加入一个元素和该元素的分数，如果该元素已经存在则会用新的分数替换原有的分数。返回值是新加入到集合中的元素个数，不包含之前已经存在的元素， 语法：`zadd key score member [score member...]`。

```
127.0.0.1:6379> zadd scoreboard 80 zhangsan 89 lisi 94 wangwu
(integer) 3
127.0.0.1:6379> zadd scoreboard 97 lisi
(integer) 0
```

- 获取元素分数，语法：`zscore key member`。

```
127.0.0.1:6379> zscore scoreboard lisi
"97"
```

- 删除元素，移除有序集key中的一个或多个成员，不存在的成员将被忽略。 当key存在但不是有序集类型时，返回一个错误。语法：`zrem key member [member...]`。

```
127.0.0.1:6379> zrem scoreboard lisi
(integer) 1
```

- 获得排名在某个范围的元素列表。

  按照元素分数从小到大的顺序返回索引从start到stop之间的所有元素（包含两端的元素） 语法：`zrange key start stop [withscores]`。

```
127.0.0.1:6379> zrange scoreboard 0 2
1) "zhangsan"
2) "wangwu"
```

  按照元素分数从大到小的顺序返回索引从start到stop之间的所有元素（包含两端的元素） 语法：`zrevrange key start stop [withscores]`。

```
127.0.0.1:6379> zrevrange scoreboard 0 2
1) "wangwu"
2) "zhangsan"
```

  如果需要获得元素的分数可以在命令末尾加上`withscores`参数。

- 获取元素排名。

  从小到大 语法：`zrank key member`。

```
127.0.0.1:6379> zrank scoreboard zhangsan
(integer) 0
```

  从大到小 语法：`zrevrank key member`。

```
127.0.0.1:6379> zrevrank scoreboard zhangsan
(integer) 1
```

- 获得指定分数范围的元素，语法：`zrangebyscore key min max [withscores] [limit offset count]`。

```
127.0.0.1:6379> ZRANGEBYSCORE scoreboard 90 97 WITHSCORES
1) "wangwu"
2) "94"
3) "lisi"
4) "97"
127.0.0.1:6379> ZRANGEBYSCORE scoreboard 70 100 limit 1 2
1) "wangwu"
2) "lisi"
```

- 增加某个元素分数，返回值是更改后的分数 语法：`zincrby key increment member`。

```
127.0.0.1:6379> ZINCRBY scoreboard 4 lisi
"101“
```

- 获得集合中元素数量，语法：`zcard key`。

```
127.0.0.1:6379> zcard scoreboard
(integer) 3
```

- 获得指定分数范围内的元素个数，语法：`zcount key min max`。

```
127.0.0.1:6379> zcount scoreboard 80 90
(integer) 1
```

- 按照排名范围删除元素，语法：`zremrangebyrank key start stop`。

```
127.0.0.1:6379> zremrangebyrank scoreboard 0 1
(integer) 2
127.0.0.1:6379> zrange scoreboard 0 -1
1) "wangwu"
```

- 按照分数范围删除元素，语法：`zremrangebyscore key min max`。

```
127.0.0.1:6379> zadd scoreboard 84 zhangsan
(integer) 1
127.0.0.1:6379> ZREMRANGEBYSCORE scoreboard 80 100
(integer) 1
```