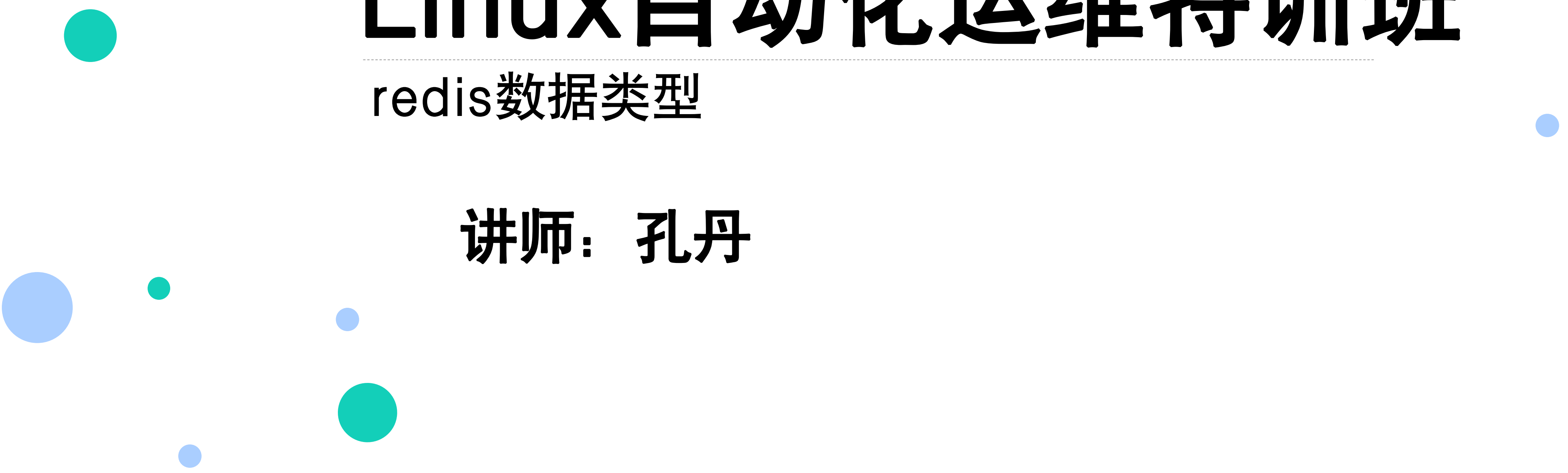




Linux自动化运维特训班

redis数据类型

讲师：孔丹



大纲

- string 类型及操作
- hash 类型及操作
- list 类型及操作
- set 类型及操作
- zset 类型及操作
- key 操作命令

string 类型及操作

- ❑ 字符串类型是Redis中最为基础的数据存储类型，它在Redis中是二进制安全的，这便意味着该类型可以接受任何格式的数据，如JPEG图像数据或Json对象描述信息等。在Redis中字符串类型的Value最多可以容纳的数据长度是512M。

- ❑ 查看帮助

```
127.0.0.1:6379> help set
```

```
SET key value [EX seconds] [PX milliseconds]  
[NX|XX]
```

```
summary: Set the string value of a key
```

```
since: 1.0.0
```

```
group: string
```

string 类型及操作

□ 基本操作

1. set: 设置 key 对应的值为 string 类型
2. setnx: 设置 key 对应的值为 string 类型, 如果 key 已经存在, 返回 0, nx 是 not exist 的意思
3. get: 获取 key 对应的 string 值, 如果 key 不存在返回 nil
4. mset & mget 同时设置和获取多个键值对
5. incrby: 对 key 的值做加加 (指定值) 操作, 并返回新的值
6. del: 删除一个已创建的 key
7. exists: 判断该键是否存在, 存在返回1, 否则返回0
8. append: 该键存在时返回追加后Value的长度,不存在时返回当前Value的长度
9. strlen: 获取指定Key的字符长度, 等效于C库中strlen函数。
10. setex: 设置指定Key的过期时间,单位为秒。

string 类型及操作

□ 基本操作示例

1. SET/GET/APPEND/STRLEN:

/> redis-cli #执行Redis客户端工具。

#判断该键是否存在，存在返回1，否则返回0。

```
redis 127.0.0.1:6379>exists mykey
```

```
(integer) 0
```

#该键并不存在，因此append命令返回当前Value的长度。

```
redis 127.0.0.1:6379> append mykey "hello"
```

```
(integer) 5
```

#该键已经存在，因此返回追加后Value的长度。

```
redis 127.0.0.1:6379>append mykey " world"
```

```
(integer) 11
```

#通过get命令获取该键，以判断append的结果。

```
redis 127.0.0.1:6379>get mykey
```

```
"hello world"
```

#通过set命令为键设置新值，并覆盖原有值。

```
redis 127.0.0.1:6379>set mykey "this is a test"
```

```
OK
```

```
redis 127.0.0.1:6379>get mykey
```

```
"this is a test"
```

#获取指定Key的字符长度，等效于C库中strlen函数。

```
redis 127.0.0.1:6379>strlen mykey
```

```
(integer) 14
```

string 类型及操作

□ 基本操作示例

2. INCR/DECR/INCRBY/DECRBY:

#设置Key的值为20

```
redis 127.0.0.1:6379>set mykey 20
```

OK

#该Key的值递增1

```
redis 127.0.0.1:6379>incr mykey
```

(integer) 21

#该Key的值递减1

```
redis 127.0.0.1:6379>decr mykey
```

(integer) 20

#删除已有键。

```
redis 127.0.0.1:6379>del mykey
```

(integer) 1

#对空值执行递减操作，其原值被设定为0，递减后的值为-1

```
redis 127.0.0.1:6379>decr mykey
```

(integer) -1

```
redis 127.0.0.1:6379>del mykey
```

(integer) 1

#对空值执行递增操作，其原值被设定为0，递增后的值为1

```
redis 127.0.0.1:6379>incr mykey
```

(integer) 1

#将该键的Value设置为不能转换为整型的普通字符串。

```
redis 127.0.0.1:6379>set mykey hello
```

OK

string 类型及操作

□ 基本操作示例

3. MSET/MGET/MSETNX:

#批量设置了key1和key2两个键。

```
redis 127.0.0.1:6379>mset key1 "hello" key2 "world"
```

OK

#批量获取了key1和key2两个键的值。

```
redis 127.0.0.1:6379>mget key1 key2
```

1) "hello"

2) "world"

#批量设置了key3和key4两个键，因为之前他们并不存在，所以该命令执行成功并返回1。

```
redis 127.0.0.1:6379>msetnx key3 "stephen" key4 "liu"
```

(integer) 1

```
redis 127.0.0.1:6379>mget key3 key4
```

1) "stephen"

2) "liu"

#批量设置了key3和key5两个键，但是key3已经存在，所以该命令执行失败并返回0。

```
redis 127.0.0.1:6379>msetnx key3 "hello" key5 "world"
```

(integer) 0

#批量获取key3和key5，由于key5没有设置成功，所以返回nil。

```
redis 127.0.0.1:6379>mget key3 key5
```

1) "stephen"

2) (nil)

hash 类型及操作

- 我们可以将Redis中的Hashes类型看成具有String Key和String Value的map容器。所以该类型非常适合于存储值对象的信息。如Username、Password和Age等。如果Hash中包含很少的字段，那么 该类型的数据也将仅占用很少的磁盘空间。每一个Hash可以存储4294967295个键值对。

hash 类型及操作

□ 基本操作

1. hset: 设置 hash field 为指定值, 如果 key 不存在, 则先创建。
2. hget、hmset、hmget 意义同上近似
3. hdel: 删除指定表中的某一个键值对
4. hgetall: 列出表中的所有键值对

hash 类型及操作

□ 基本操作示例

1. HSET/HGET/HDEL/HEXISTS/HLEN/HSETNX:

#给键值为myhash的键设置字段为field1， 值为stephen。

```
redis 127.0.0.1:6379>hset myhash field1 "stephen"
```

```
(integer) 1
```

#获取键值为myhash， 字段为field1的值。

```
redis 127.0.0.1:6379>hget myhash field1
```

```
"stephen"
```

#myhash键中不存在field2字段， 因此返回nil。

```
redis 127.0.0.1:6379>hget myhash field2
```

```
(nil)
```

#给myhash关联的Hashes值添加一个新的字段field2， 其值为liu。

```
redis 127.0.0.1:6379>hset myhash field2 "liu"
```

```
(integer) 1
```

#获取myhash键的字段数量。

```
redis 127.0.0.1:6379>hlen myhash
```

```
(integer) 2
```

#判断myhash键中是否存在字段名为field1的字段， 由于存在， 返回值为1。

```
redis 127.0.0.1:6379>hexists myhash field1
```

```
(integer) 1
```

#删除myhash键中字段名为field1的字段， 删除成功返回1。

```
redis 127.0.0.1:6379>hdel myhash field1
```

```
(integer) 1
```

#判断myhash键中是否存在field1字段， 由于上一条命令已经将其删除， 因为返回0。

```
redis 127.0.0.1:6379>hexists myhash field1
```

```
(integer) 0
```

hash 类型及操作

□ 基本操作示例

2. HINCRBY:

#删除该键，便于后面示例的测试。

```
redis 127.0.0.1:6379>del myhash
```

```
(integer) 1
```

#准备测试数据，该myhash的field字段设定值1。

```
redis 127.0.0.1:6379>hset myhash field 5
```

```
(integer) 1
```

#给myhash的field字段的值加1，返回加后的结果。

```
redis 127.0.0.1:6379>hincrby myhash field 1
```

```
(integer) 6
```

#给myhash的field字段的值加-1，返回加后的结果。

```
redis 127.0.0.1:6379>hincrby myhash field -1
```

```
(integer) 5
```

#给myhash的field字段的值加-10，返回加后的结果。

```
redis 127.0.0.1:6379>hincrby myhash field -10
```

```
(integer) -5
```

hash 类型及操作

□ 基本操作示例

3. HGETALL/HKEYS/HVALS/HMGET/HMSET:

#删除该键，便于后面示例测试。

```
redis 127.0.0.1:6379>del myhash
```

(integer) 1

#为该键myhash，一次性设置多个字段，分别是field1 = "hello", field2 = "world"。

```
redis 127.0.0.1:6379>hmset myhash field1 "hello" field2 "world"
```

OK

#获取myhash键的多个字段，其中field3并不存在，因为在返回结果中与该字段对应的值为nil。

```
redis 127.0.0.1:6379>hmget myhash field1 field2 field3
```

1) "hello"

2) "world"

3) (nil)

#返回myhash键的所有字段及其值，从结果中可以看出，他们是逐对列出的。

```
redis 127.0.0.1:6379>hgetall myhash
```

1) "field1"

2) "hello"

3) "field2"

4) "world"

#仅获取myhash键中所有字段的名称。

```
redis 127.0.0.1:6379>hkeys myhash
```

#仅获取myhash键中所有字段的值。

```
redis 127.0.0.1:6379>hvals myhash
```

list 类型及操作

- 在Redis中，List类型是字符串链表。和数据结构中的普通链表一样，我们可以在其头部(left)和尾部(right)添加新的元素。在插入时，如果该键并不存在，Redis将为该键创建一个新的链表。与此相反，如果链表中所有的元素均被移除，那么该键也将会被从数据库中删除。List中可以包含的最大元素数量是4294967295。
- 从元素插入和删除的效率视角来看，如果我们是在链表的两头插入或删除元素，这将会是非常高效的操作，即使链表中已经存储了百万条记录，该操作也可以在常量时间内完成。然而需要说明的是，如果元素插入或删除操作是作用于链表中间，那将会是非常低效的。相信对于有良好数据结构基础的开发者而言，这一点并不难理解。

list 类型及操作

□ 基本操作

1. lpush: 在 key 对应 list 的头部添加字符串元素。
2. lrange: 从指定链表中获取指定范围的元素
3. rpush: 在指定Key所关联的List Value的尾部插入参数中给出的所有Values。
4. lpop: 返回并弹出指定Key关联的链表中的第一个元素，即头部元素，。
如果该Key不存，返回nil。
5. rpop: 返回并弹出指定Key关联的链表中的最后一个元素，即尾部元素，
如果该Key不存，返回nil。

list 类型及操作

□ 基本操作示例

1. LPUSH/LPUSHX/LRANGE:

```
redis 127.0.0.1:6379>del mykey
```

```
(integer) 1
```

#mykey键并不存在，该命令会创建该键及与其关联的List，之后在将参数中的values从左到右依次插入。

```
redis 127.0.0.1:6379>lpush mykey a b c d
```

#取从位置0开始到位置2结束的3个元素。

```
redis 127.0.0.1:6379> lrange mykey 0 2
```

```
1) "d"
```

```
2) "c"
```

```
3) "b"
```

#取链表中的全部元素，其中0表示第一个元素，-1表示最后一个元素。

```
redis 127.0.0.1:6379> lrange mykey 0 -1
```

#mykey2键此时并不存在，因此该命令将不会进行任何操作，其返回值为0。

```
redis 127.0.0.1:6379> lpushx mykey2 e
```

#可以看到mykey2没有关联任何List Value。

```
redis 127.0.0.1:6379>lrange mykey2 0 -1
```

```
(empty list or set)
```

#mykey键此时已经存在，所以该命令插入成功，并返回链表中当前元素的数量。

```
redis 127.0.0.1:6379> lpushx mykey e
```

#获取该键的List Value的头部元素。

```
redis 127.0.0.1:6379> lrange mykey 0 0
```

```
1) "e"
```


list 类型及操作

□ 基本操作示例

2. LPOP/LLEN:

```
redis 127.0.0.1:6379>lpush mykey a b c d
```

```
(integer) 4
```

```
redis 127.0.0.1:6379>lpop mykey
```

```
"d"
```

```
redis 127.0.0.1:6379>lpop mykey
```

```
"c"
```

#在执行lpop命令两次后，链表头部的两个元素已经被弹出，此时链表中元素的数量是2

```
redis 127.0.0.1:6379>llen mykey
```

```
(integer) 2
```

3. LINSERT:

#删除该键便于后面的测试。

```
redis 127.0.0.1:6379> del mykey
```

#为后面的示例准备测试数据。

```
redis 127.0.0.1:6379>lpush mykey a b c d e
```

#在a的前面插入新元素a1。

```
redis 127.0.0.1:6379> linsert mykey before a a1
```

#查看是否插入成功，从结果看已经插入。注意index的index值是0-based。

```
redis 127.0.0.1:6379>lindex mykey 0
```

```
"e"
```

#在e的后面插入新元素e2，从返回结果看已经插入成功。

```
redis 127.0.0.1:6379> linsert mykey after e e2
```

list 类型及操作

□ 基本操作示例

4. RPush/RPushX/RPop/RPopLPush:

#删除该键，以便于后面的测试。

```
redis 127.0.0.1:6379> del mykey  
(integer) 1
```

#从链表的尾部插入参数中给出的values，插入顺序是从左到右依次插入。

```
redis 127.0.0.1:6379> rpush mykey a b c d  
(integer) 4
```

#通过lrange的可以获悉rpush在插入多值时的插入顺序。

```
redis 127.0.0.1:6379> lrange mykey 0 -1  
1) "a"
```

set 类型及操作

- 在Redis中，我们可以将Set类型看作为没有排序的字符集合，和List类型一样，我们也可以在该类型的数据值上执行添加、删除或判断某一元素是否存在等操作。需要说明的是，这些操作的时间复杂度为 $O(1)$ ，即常量时间内完成次操作。Set可包含的最大元素数量是4294967295。和List类型不同的是，这一点和C++标准库中的set容器是完全相同的。换句话说，如果多次添加相同元素，Set中将仅保留该元素的一份拷贝。和List类型相比，Set类型在功能上还存在着一个非常重要的特性，即在服务器端完成多个Sets之间的聚合计算操作，如 unions、intersections和differences。由于这些操作均在服务端完成，因此效率极高，而且也节省了大量的网络IO开销。

无序性：集合里面数据是没顺序区分。

确定性：集合里面数据个数是确定的。

唯一性：集合里面数据不能彼此重复。

set 类型及操作

□ 基本操作

1. sadd: 添加一个或多个元素到集合中
2. smembers: 获取集合里面所有的元素
3. srem: 从集合中删除指定的一个或多个元素
4. spop: 随机从集合中删除一个元素, 并返回
5. scard: 获取集合里面的元素个数
6. sdiff: 返回集合 1 与集合 2 的差集。以集合 1 为主
redis127.0.0.1:6379>sdiff mset1 mset2
7. sinter: 获得两个集合的交集
8. sunion: 获得指定集合的并集

set 类型及操作

□ 基本操作示例

1. SADD/SMEMBERS/SCARD/SISMEMBER:

#插入测试数据，由于该键myset之前并不存在，因此参数中的三个成员都被正常插入。

```
redis 127.0.0.1:6379>sadd myset a b c
```

```
(integer) 3
```

#由于参数中的a在myset中已经存在，因此本次操作仅仅插入了d和e两个新成员。

```
redis 127.0.0.1:6379>sadd myset a d e
```

```
(integer) 2
```

#判断a是否已经存在，返回值为1表示存在。

```
redis 127.0.0.1:6379>sismember myset a
```

```
(integer) 1
```

#判断f是否已经存在，返回值为0表示不存在。

```
redis 127.0.0.1:6379>sismember myset f
```

```
(integer) 0
```

#通过smembers命令查看插入的结果，从结果可以，输出的顺序和插入顺序无关。

```
redis 127.0.0.1:6379> smembers myset
```

```
1) "c"
```

```
2) "d"
```

```
3) "a"
```

```
4) "b"
```

```
5) "e"
```

#获取Set集合中元素的数量。

```
redis 127.0.0.1:6379>scard myset
```

```
(integer) 5
```

set 类型及操作

□ 基本操作示例

2. SPOP/SREM/SRANDMEMBER:

#删除该键，便于后面的测试。

```
redis 127.0.0.1:6379>del myset
```

#为后面的示例准备测试数据。

```
redis 127.0.0.1:6379>sadd myset a b c d
```

#从结果可以看出，该命令确实是随机的返回了某一成员。

```
redis 127.0.0.1:6379>randmember myset
```

```
"c"
```

#Set中尾部的成员b被移出并返回，事实上b并不是之前插入的第一个或最后一个成员。

```
redis 127.0.0.1:6379>spop myset
```

```
"b"
```

#查看移出后Set的成员信息。

```
redis 127.0.0.1:6379> smembers myset
```

```
1) "c"
```

```
2) "d"
```

```
3) "a"
```

#从Set中移出a、d和f三个成员，其中f并不存在，因此只有a和d两个成员被移出，返回为2。

```
redis 127.0.0.1:6379>srem myset a d f
```

```
(integer) 2
```


set 类型及操作

□ 基本操作示例

3. SDIFF/SINTER/SUNION:

#为后面的命令准备测试数据。

```
redis 127.0.0.1:6379>sadd myset a b c d
```

```
redis 127.0.0.1:6379> sadd myset2 c
```

```
redis 127.0.0.1:6379>sadd myset3 a c e
```

#myset和myset2相比，a、b和d三个成员是两者之间的差异成员。再用这个结果继续和myset3进行差异比较，b和d是myset3不存在的成员。

```
redis 127.0.0.1:6379>sdiff myset myset2 myset3
```

```
1) "d"
```

```
2) "b"
```

#从之前准备的数据就可以看出，这三个Set的成员交集只有c。

```
redis 127.0.0.1:6379>sinter myset myset2 myset3
```

```
1) "c"
```

#获取3个集合中的成员的并集。

```
redis 127.0.0.1:6379>sunion myset myset2 myset3
```

```
1) "b"
```

```
2) "c"
```

```
3) "d"
```

```
4) "e"
```

```
5) "a"
```


set 类型及操作

□ 实际应用

- 1). 可以使用Redis的Set数据类型跟踪一些唯一性数据，比如访问某一博客的唯一IP地址信息。对于此场景，我们仅需在每次访问该博客时将访问者的IP存入Redis中，Set数据类型会自动保证IP地址的唯一性。
- 2). 充分利用Set类型的服务端聚合操作方便、高效的特性，可以用于维护数据对象之间的关联关系。比如所有购买某一电子设备的客户ID被存储在一个指定的 Set中，而购买另外一种电子产品的客户ID被存储在另外一个Set中，如果此时我们想获取有哪些客户同时购买了这两种商品时，Set的 intersections命令就可以充分发挥它的方便和效率的优势了。

set 类型及操作

□ 实际应用示例

1、设计四个好友

```
[root@localhost ~]# redis-cli
```

```
127.0.0.1:6379> set it_user:id:1:username tom
```

```
127.0.0.1:6379> set it_user:id:1:email tom@qq.com
```

```
127.0.0.1:6379> set it_user:id:2:username john
```

```
127.0.0.1:6379> set it_user:id:2:email john@qq.com
```

```
127.0.0.1:6379> set it_user:id:3:username bob
```

```
127.0.0.1:6379> set it_user:id:3:email bob@qq.com
```

```
127.0.0.1:6379> set it_user:id:4:username smith
```

```
127.0.0.1:6379> set it_user:id:4:email smith@qq.com
```

```
127.0.0.1:6379> keys it_user:id*
```

```
1) "it_user:id:4:username"
```

```
2) "it_user:id:1:username"
```

```
3) "it_user:id:2:username"
```

```
4) "it_user:id:2:email"
```

```
5) "it_user:id:1:email"
```

```
6) "it_user:id:3:email"
```

```
7) "it_user:id:4:email"
```

```
8) "it_user:id:3:username"
```

set 类型及操作

□ 实际应用示例

2、设定好友集合

1号好友为2号和3号

```
127.0.0.1:6379> sadd set:user:id:1:friend 2
```

```
(integer) 1
```

```
127.0.0.1:6379> sadd set:user:id:1:friend 3
```

```
(integer) 1
```

```
127.0.0.1:6379> SMEMBERS set:user:id:1:friend
```

```
1) "2"
```

```
2) "3"
```

4号好友为3号

```
127.0.0.1:6379> sadd set:user:id:4:friend 3
```

```
(integer) 1
```

```
127.0.0.1:6379> SMEMBERS set:user:id:4:friend
```

```
1) "3"
```

set 类型及操作

□ 实际应用示例

3、好友关系

共同好友（交集）

```
127.0.0.1:6379> SINTER set:user:id:1:friend set:user:id:4:friend  
1) "3"
```

全部好友（并集）

```
127.0.0.1:6379> sunion set:user:id:1:friend set:user:id:4:friend  
1) "2"  
2) "3"
```

推荐好友（差集）

```
127.0.0.1:6379> SDIFF set:user:id:1:friend set:user:id:4:friend  
1) "2"
```

zset 类型及操作

- ❑ Sorted-Sets和Sets类型极为相似，它们都是字符串的集合，都不允许重复的成员出现在一个Set中。它们之间的主要差别是Sorted-Sets中的每一个成员都会有一个分数(score)与之关联，Redis正是通过分数来为集合中的成员进行从小到大的排序。然而需要额外指出的是，尽管Sorted-Sets中的成员必须是唯一的，但是分数(score)却是可以重复的。
- ❑ 在Sorted-Set中添加、删除或更新一个成员都是非常快速的操作，其时间复杂度为集合中成员数量的对数。由于Sorted-Sets中的成员在集合中的位置是有序的，因此，即便是访问位于集合中部的成员也仍然是非常高效的。事实上，Redis所具有的这一特征在很多其它类型的数据库中是很难实现的，换句话说，在该点上要想达到和Redis同样的高效，在其它数据库中进行建模是非常困难的。

zset 类型及操作

□ 基本操作

1. zadd: 向一个指定的有序集合中添加元素，每一个元素会对应的有一个分数。你可以指定多个分数/成员组合。如果一个指定的成员已经在对应的有序集合中了，那么其分数就会被更新成最新的，并且该成员会重新调整到正确的位置，以确保集合有序。分数的值必须是一个表示数字的字符串

2. zrange: 返回有序集合中，指定区间内的成员。其中成员按照 score（分数）值从小到大排序。具有相同 score 值的成员按照字典顺序来排列。

```
redis127.0.0.1:6379>zrange zset 0 -1 withscores
```

注: withscores 返回集合中元素的同时，返回其分数（score）

3. zrem: 删除有序集合中指定的值

```
redis127.0.0.1:6379>zrem zset zhangsan
```

4. zcard: 返回有序集合元素的个数

zset 类型及操作

□ 基本操作示例

1. ZADD/ZCARD/ZREM/ZINCRBY/ZSCORE/ZRANGE/ZRANK:

#添加一个分数为1的成员。

```
redis 127.0.0.1:6379> zadd myzset 1 "one"
```

#添加两个分数分别是2和3的两个成员。

```
redis 127.0.0.1:6379>zadd myzset 2 "two" 3 "three"
```

#0表示第一个成员， -1表示最后一个成员。WITHSCORES选项表示返回的结果中包含每个成员及其分数， 否则只返回成员。

```
redis 127.0.0.1:6379>zrange myzset 0 -1 WITHSCORES
```

#获取成员one在Sorted-Set中的位置索引值。0表示第一个位置。

```
redis 127.0.0.1:6379>zrank myzset one
```

#成员four并不存在， 因此返回nil。

```
redis 127.0.0.1:6379>zrank myzset four
```

```
(nil)
```

#获取myzset键中成员的数量。

```
redis 127.0.0.1:6379> zcard myzset
```

#删除成员one和two， 返回实际删除成员的数量。

```
redis 127.0.0.1:6379>zrem myzset one two
```

#获取成员three的分数。返回值是字符串形式。

```
redis 127.0.0.1:6379>zscore myzset three
```

```
"3"
```

#将成员one的分数增加2， 并返回该成员更新后的分数。

```
redis 127.0.0.1:6379>zincrby myzset 2 one
```

```
"3"
```


zset 类型及操作

□ 基本操作示例

应用范围：

1). 可以用于一个大型在线游戏的积分排行榜。每当玩家的分数发生变化时，可以执行ZADD命令更新玩家的分数，此后再通过ZRANGE命令获取积分TOP TEN的用户信息。当然我们也可以利用ZRANK命令通过username来获取玩家的排行信息。最后我们将组合使用ZRANGE和ZRANK命令快速的获取和某个玩家积分相近的其他用户的信息。

2). Sorted-Sets类型还可用于构建索引数据。

key 操作命令

- 在前面主要介绍的是与Redis数据类型相关的命令，如String、List、Set、Hashes和Sorted-Set。这些命令都具有一个共同点，即所有的操作都是针对与Key关联的Value的。下面主要讲述与Key相关的Redis命令。学习这些命令对于学习 Redis是非常重要的基础，也是能够充分挖掘Redis潜力的利器。

key 操作命令

□ 基本操作

1. EXISTS key: 判断指定的key是否存在
2. DEL key [key ...]: 从数据库中删除指定的key
3. KEYS pattern: 获取所有匹配模式的keys
4. MOVE key db: 将当前数据库中指定的键Key移动到参数中指定的数据库中
5. RENAME key newkey: 为指定指定的键重新命名
6. EXPIRE key seconds: 设置key的超时时间,单位秒
7. TTL key: 获取key的超时描述
8. RANDOMKEY: 返回随机的key
9. TYPE key: 获取与参数中指定键关联值的类型, 该命令将以字符串的格式返回

key 操作命令

□ 基本操作示例

1. KEYS/RENAME/DEL/EXISTS/MOVE/RENAMENX:

#清空当前选择的数据库，以便于对后面示例的理解。

```
redis 127.0.0.1:6379>flushdb
```

#添加String类型的模拟数据。

```
redis 127.0.0.1:6379>set mykey 2
```

```
redis 127.0.0.1:6379>set mykey2 "hello"
```

#添加Set类型的模拟数据。

```
redis 127.0.0.1:6379>sadd mysetkey 1 2 3
```

#添加Hash类型的模拟数据。

```
redis 127.0.0.1:6379>hset mmtest username "stephen"
```

#根据参数中的模式，获取当前数据库中符合该模式的所有key，从输出可以看出，该命令在执行时并不区分与Key关联的Value类型。

```
redis 127.0.0.1:6379>keys my*
```

```
1) "mysetkey"
```

```
2) "mykey"
```

```
3) "mykey2"
```

#删除了两个Keys。

```
redis 127.0.0.1:6379>del mykey mykey2
```

#将当前数据库中的mysetkey键移入到ID为1的数据库中，从结果可以看出已经移动成功。

```
redis 127.0.0.1:6379>move mysetkey 1
```

#准备新的测试数据。

```
redis 127.0.0.1:6379>set mykey "hello"
```

#将mykey改名为mykey1

```
redis 127.0.0.1:6379> rename mykey mykey1
```

key 操作命令

□ 基本操作示例

2. PERSIST/EXPIRE/EXPIREAT/TTL:

#为后面的示例准备的测试数据。

```
redis 127.0.0.1:6379>set mykey "hello"
```

#将该键的超时设置为100秒。

```
redis 127.0.0.1:6379>expire mykey 100
```

#通过ttl命令查看一下还剩下多少秒。

```
redis 127.0.0.1:6379>tll mykey
```

(integer) 97

#立刻执行persist命令，该存在超时的键变成持久化的键，即将该Key的超时去掉。

```
redis 127.0.0.1:6379>persist mykey
```

(integer) 1

#ttl的返回值告诉我们，该键已经没有超时了。

```
redis 127.0.0.1:6379>tll mykey
```

(integer) -1

总结

- string 类型及操作
- hash 类型及操作
- list 类型及操作
- set 类型及操作
- zset 类型及操作
- key 操作命令

作业

□ 作业一：string list hash结构中，每个至少完成5个命令，包含插入 修改 删除 查询，list 和hash还需要增加遍历的操作命令

1、 string类型数据的命令操作：

- (1) 设置键值：
- (2) 读取键值：
- (3) 数值类型自增1：
- (4) 数值类型自减1：
- (5) 查看值的长度：

2、 list类型数据的命令操作：

- (1) 对列表city插入元素：Shanghai Suzhou Hangzhou
- (2) 将列表city里的头部的元素移除
- (3) 将name列表的尾部元素移除到number列表的头部
- (4) 对一个已存在的列表插入新元素
- (5) 查看list的值长度

作业

3、 hash类型数据的命令操作:

- (1) 设置一个hash表, order表里包括的键值信息有: id: 1, customer_name: 张三
- (2) 创建一个hash表, 表里的键值批量插入
- (3) 获取order对应的map的所有key
- (4) 获取order对应的map的键值数量
- (5) 获取order表里的id值

4、 Keys相关的命令操作

- (1) 查看key是否存在
- (2) 查找满足pattern的keys
- (3) 查看key的超时时间
- (4) 遍历key

作业二: 举例说明list和hash的应用场景, 每个至少一个场景



谢谢观看

更多好课，请关注[万门大学APP](#)

