数据结构之哈希

说明

```
redis的无序字典是一个string类型的field和value的映射表，内部结构类似HashMap。 每个 hash 可以存储 2^32-1 个键值对（40多亿）。
    对应关系(HashMap的key，对应我们Redis的keyName)：
      HashMap<String, String> ==> string;
      HashMap<String, List> ==> list;
      HashMap<String, Map<String, String> ==> hash;
      HashMap<String, Set<String>> ==> set;
      HashMap<String, Set<Double, String>> ==> zset;
```

命令

```
1. hset hashKeyName fieldName value : 设置键名为hashKeyName,值为<fieldName, value>,可多次设置，那么效果就是： hashKeyName -> [{key1, value1}, {key2,value2}, {key3, value3}, {key4, value4}, ...]
  如 : hset hash1 key1 value1 , 成功返回（Integer）1，错误返回Integer（0）
  注 : hash1 -> <key1, value1> , redis从实际来说，都是key-value，只是这个value的类型不一样而已。

2. hget hashKeyName fieldName : 取出键名为hashKeyName,属性为fieldName 的 value
  如 : hget hash1 key1

3. hkeys hashKeyName : 取出当前hashKeyName所包含的所有key
   hvals hashKeyName : 取出当前hashKeyName所包含的所有value

4. 批量set ： hmset hashKeyName field1 value1 [field2 value ...] ， 成功返回 OK
  如 : hmset hash1 key3 values key4 value4 , 向hash1内设置多个key-value
   批量get : hmget hashKeyName fieldName1 fieldName2 fieldName3 ... ， 从hashKeyName中取出多个对应的值
    如 : 127.0.0.1:6379> hmget hash1 key1 key3 key4
         1) "value1"
         2) "value3"
         3) "value4"
         127.0.0.1:6379>
5. hgetall hashKeyName : 获取hashKeyName中所有的key和value

6. hdel hashKeyName fieldName field1 [field2 field3 ...] ：删除hashKeyName中对应属性和值
7. hlen hashKeyName : 查看数据长度
8. hexists hashKeyName fieldName ： 查看hashKeyName中是否存在fieldName，存在返回1，不存在0

9. 整合命令：
```

```
   hsetnx hashKeyName fieldName value : 如果hashKeyName内不存在fieldName，设置，设置成功返回1，失败0
```

```
10. 当map内的value值，是数值，还是可以进行增加操作的 ： hincrby key field increment
  127.0.0.1:6379> hset hashNum num1 1
    (integer) 1
    127.0.0.1:6379> hincrby hashNum num1 3
    (integer) 4
    127.0.0.1:6379> hget hashNum num1
    "4"
    127.0.0.1:6379>

---
命令汇总：
* CRUD ： hset / hget / hmset / hmget / hdel / hsetnx
* 遍历操作：hgetall / hlen / hexists / hkeys / hvals
* 数值操作：hincrby / hincrbyfloat
应用：存储对象。
```

原理

```
底层的实现结构，与HashMap也一样，是“数组+链表”的二维结构，
第一维hash的数据位置碰撞时，将碰撞元素用链表串接起来，
不同的是，redis字典的值只能是字符串，而且两者的rehash方式不同。
java的hashmap是一次全部 rehash，
耗时较高，redis为了优化性能，采用渐进式rehash策略。
具体方式为，同时保留新旧两个hash结构，然后逐步搬迁，搬迁完成后再取而代之。
```

数据结构之集合

说明

```
redis的集合是string类型的无序不重复的元素集。同时提供求交集、并集、差集等操作。
集合中最大的成员数为 2^32-1（40多亿）。
Java集合中有list/set/map
  list可以存储有序的重复数据，而set可以存储无序不重复数据
  hashset的底层就是hashmap
  redis中的set，本质也是通过hash来实现的
```

命令

```
1. sadd setKeyName member [member2 member3 ...] : 向setKeyName内插入成员，返回集合中的元素个数
2. scard setKeyName : 查看setKeyName集合的元素个数
3. smembers setKeyName : 查看setKeyName内的所有元素的值
    比如：
        127.0.0.1:6379> sadd set1 1 2 3 4 3 4 2
        (integer) 4
        127.0.0.1:6379> scard set1
        (integer) 4
        127.0.0.1:6379> smembers set1
        1) "1"
        2) "2"
        3) "3"
        4) "4"
        127.0.0.1:6379>
5. srem setKeyName member [member2 member3 ...] : 移除setKeyName集合中的1至n个member，成功返回1，失败0
6. sismember setKeyName member : 判断member是否在setKeyName所对应的set中，成功返回1，失败0
7. srandmember setKeyName count : 随机从setKeyName对应的集合中随机出count个member
8. spop setKeyName count :  随机从setKeyName对应的集合中随机弹出count个member
9. smove setKeyName1 setKeyName2 member : 将member从setKeyName1移动至setKeyName2
  比如：
      127.0.0.1:6379> smembers set1
        1) "1"
        2) "3"
        3) "4"
        127.0.0.1:6379> sadd set2 11 13 144
        127.0.0.1:6379> smove set1 set2 3
        (integer) 1
        127.0.0.1:6379> smembers set2
        1) "3"
        2) "11"
        3) "13"
        4) "144"
        127.0.0.1:6379>
        127.0.0.1:6379> smembers set1
        1) "1"
        2) "4"
        127.0.0.1:6379>
10. 并集、交集(A集合，B集合都有的)、差集(A集合，B集合都没有的)
  交集：sinter setKeyName [setKeyName2 setKeyName3 ...] ：  求均在setKeyName、setKeyName2...的元素
  差集：sdiff setKeyName [setKeyName2 setKeyName3 ...] ：  求在setKeyName但是不在其他集合中的元素
  并集：sunion setKeyName [setKeyName2 setKeyName3 ...] ：  求集合的所有元素
    比如：
```

```
        127.0.0.1:6379> smembers set1
             1) "1"
             2) "2"
             3) "3"
             4) "4"
             127.0.0.1:6379> smembers set2
             1) "3"
             2) "11"
             3) "13"
             4) "144"
             127.0.0.1:6379> sinter set1 set2
             1) "3"
             127.0.0.1:6379> sdiff set1 set2
             1) "1"
             2) "2"
             3) "4"
             127.0.0.1:6379> sunion set1 set2
             1) "1"
             2) "2"
             3) "3"
             4) "4"
             5) "11"
             6) "13"
             7) "144"
  拓展问题：在Java的set中，如何求交集、差集、并集?
      分别调用：retainAll()、removeAll()、addAll()
---
命令汇总：
* CRUD：sadd / srem / spop
* 遍历操作：scard / smembers / sismember / srandmember
* 交叉操作：smove / sdiff / sinter / sunion
```

原理

```
类似HashSet，也是通过哈希表实现的，相当于所有的value都是空。
通过计算hash的方式来快速排重，也是set能提供判断一个成员是否在集合内的原因。
```

数据类型之有序集合

说明

```
redis的有序集合和set一样也是string类型元素的集合，且不允许重复的成员。不同的是,每个元素都会关联一个double类型的分数。redis正是通过分数来为集合中的成员进行从小到大排序。zset的成员是唯一的，但分数（score）却可以重复。
```

命令

```
1. zadd zsetKeyName score member [score2 member3 score3 member3 ...] : 增加zsetKeyName,并设置分数和值(每个元素值，都必须包含<score, value>)
2. zrange zsetKeyName start stop [withscores]: 查看[start, stop]的元素
  比如（看结果就是有序的）：
        127.0.0.1:6379> zadd zset1 60 xiaotian 70 zhongtian 80 datian 100 zhoudbw
        (integer) 4
        127.0.0.1:6379> zrange zset1 0 -1
        1) "xiaotian"
        2) "zhongtian"
        3) "datian"
        4) "zhoudbw"
        127.0.0.1:6379> zrange zset1 0 -1 withscores
        1) "xiaotian"
        2) "60"
        3) "zhongtian"
        4) "70"
        5) "datian"
        6) "80"
        7) "zhoudbw"
        8) "100"
3. zrangebyscore zsetKeyName minValue maxValue [withscore] [limit offset count] : 通过score，范围查找
  比如:
        127.0.0.1:6379> zrangebyscore zset1 60 80
        1) "xiaotian"
        2) "zhongtian"
        3) "datian"
        * 上述命令score的范围是[60, 80]
        127.0.0.1:6379> zrangebyscore zset1 60 80 limit 0 2
        1) "xiaotian"
        2) "zhongtian"
        * 上述命令增加了limit，表示，返回的是[60, 80]内，从0开始的，两个元素。
        127.0.0.1:6379> zrangebyscore zset1 (60 (80
        1) "zhongtian"
        * 上述命令，通过增加左括号，就变成了(60, 80)，变为了开区间
4. zrem zsetKeyName member : 删除zsetKeyName的member，同时score也被删除。

5. zcard zsetKeyName : 统计zset中元素的个数
6. zcount zsetKeyName min max : 统计分数在[min, max]范围内的元素个数
7. zscore zsetKeyName member : 查询zsetKeyName中member的分数
8. zrank zsetKeyName member : 查询member所在的位置索引（zset是有序的，所以位置就代表了排名）
9. zrevrank zsetKeyName member : 查询member所在位置索引，是逆序的排名
10. zrevrange zsetKeyName start stop : 查询逆序的zset元素
  比如:
        127.0.0.1:6379> zcard zset1
        (integer) 4
```

```
        127.0.0.1:6379> zcount zset1 60 80
        (integer) 3
        127.0.0.1:6379> zscore zset1 xiaotian
        "60"
        127.0.0.1:6379> zrank zset1 datian
        (integer) 2
        127.0.0.1:6379> zrevrank zset1 datian
        (integer) 1
        127.0.0.1:6379> zrevrange zset1 0 -1
        1) "zhoudbw"
        2) "datian"
        3) "zhongtian"
        4) "xiaotian"
---
命令汇总:
* CRUD : zadd / zrem / zrange + withscores
* 范围操作: zrangebyscore + 用"("代表不包含 + limit l 统计操作: zcard / zcount / zrank
* 逆序操作: zrevrank / zrevrange / zrevrangebyscore
```

原理

```
类似于SortedSet和HashMap的结合，内部实现是一种叫"跳跃列表"的数据结构。
通过层级制，将元素分层并串联，
每隔几个元素选出一个代表，
再将代表使用另外一级指针串起来，
以此类推，形成金字塔结构。
同一个 元素在不同层级之间身兼数职，是为"跳跃"。新元素插入时，逐层下潜，直到找到合适的位置。
```