redis string list set (sorted set) hash

1、String

常用命令：

除了get、set、incr、decr mget等操作外，Redis还提供了下面一些操作：

获取字符串长度

往字符串append内容

设置和获取字符串的某一段内容

设置及获取字符串的某一位（bit）

批量设置一系列字符串的内容

应用场景：

String是最常用的一种数据类型，普通的key/value存储都可以归为此类，value其实不仅是String，

也可以是数字：比如想知道什么时候封锁一个IP地址(访问超过几次)。INCRBY命令让这些变得很容易，通过原子递增保持计数。

实现方式：

m,decr等操作时会转成数值型进行计算，此时redisObject的encoding字段为int。

常用命令：

hget,hset,hgetall 等。

应用场景：

我们简单举个实例来描述下Hash的应用场景，比如我们要存储一个用户信息对象数据，包含以下信息：

           用户ID，为查找的key，

           存储的value用户对象包含姓名name，年龄age，生日birthday 等信息，

   如果用普通的key/value结构来存储，主要有以下2种存储方式：

       第一种方式将用户ID作为查找key,把其他信息封装成一个对象以序列化的方式存储，

           如：set u001 "李三,18,20010101"

           这种方式的缺点是，增加了序列化/反序列化的开销，并且在需要修改其中一项信息时，需要把整个对象取回，并且修改操作需要对并发进行保护，引入CAS等复杂问题。

       第二种方法是这个用户信息对象有多少成员就存成多少个key-value对儿，用用户ID+对应属性的名称作为唯一标识来取得对应属性的值，

           如：mset user:001:name "李三 "user:001:age18 user:001:birthday "20010101"

           虽然省去了序列化开销和并发问题，但是用户ID为重复存储，如果存在大量这样的数据，内存浪费还是非常可观的。

    那么Redis提供的Hash很好的解决了这个问题，Redis的Hash实际是内部存储的Value为一个HashMap，

    并提供了直接存取这个Map成员的接口，

        如：hmset user:001 name "李三" age 18 birthday "20010101"

            也就是说，Key仍然是用户ID,value是一个Map，这个Map的key是成员的属性名，value是属性值，

            这样对数据的修改和存取都可以直接通过其内部Map的Key(Redis里称内部Map的key为field), 也就是通过

            key(用户ID) + field(属性标签) 操作对应属性数据了，既不需要重复存储数据，也不会带来序列化和并发修改控制的问题。很好的解决了问题。

          这里同时需要注意，Redis提供了接口(hgetall)可以直接取到全部的属性数据,但是如果内部Map的成员很多，那么涉及到遍历整个内部Map的操作，由于Redis单线程模型的缘故，这个遍历操作可能会比较耗时，而另其它客户端的请求完全不响应，这点需要格外注意。

  实现方式：

上面已经说到Redis Hash对应Value内部实际就是一个HashMap，实际这里会有2种不同实现，这个Hash的成员比较少时Redis为了节省内存会采用类似一维数组的方式来紧凑存储，而不会采用真正的HashMap结构，对应的value redisObject的encoding为zipmap,当成员数量增大时会自动转成真正的HashMap,此时encoding为ht。

List

常用命令：

    lpush,rpush,lpop,rpop,lrange,BLPOP(阻塞版)等。

应用场景：

    Redis list的应用场景非常多，也是Redis最重要的数据结构之一。

    我们可以轻松地实现最新消息排行等功能。

    Lists的另一个应用就是消息队列，可以利用Lists的PUSH操作，将任务存在Lists中，然后工作线程再用POP操作将任务取出进行执行。

实现方式：

    Redis list的实现为一个双向链表，即可以支持反向查找和遍历，更方便操作，不过带来了部分额外的内存开销，Redis内部的很多实现，包括发送缓冲队列等也都是用的这个数据结构。

RPOPLPUSH source destination

    命令 RPOPLPUSH 在一个原子时间内，执行以下两个动作：

    将列表 source 中的最后一个元素(尾元素)弹出，并返回给客户端。

    将 source 弹出的元素插入到列表 destination ，作为 destination 列表的的头元素。

    如果 source 和 destination 相同，则列表中的表尾元素被移动到表头，并返回该元素，可以把这种特殊情况视作列表的旋转(rotation)操作。

    一个典型的例子就是服务器的监控程序：它们需要在尽可能短的时间内，并行地检查一组网站，确保它们的可访问性。

    redis.lpush "downstream\_ips", "192.168.0.10"

    redis.lpush "downstream\_ips", "192.168.0.11"

    redis.lpush "downstream\_ips", "192.168.0.12"

    redis.lpush "downstream\_ips", "192.168.0.13"

    Then:

    next\_ip = redis.rpoplpush "downstream\_ips", "downstream\_ips"

BLPOP

  假设现在有 job 、 command 和 request 三个列表，其中 job 不存在， command 和 request 都持有非空列表。考虑以下命令：

  BLPOP job command request 30  #阻塞30秒，0的话就是无限期阻塞,job列表为空,被跳过,紧接着command 列表的第一个元素被弹出。

  1) "command"                             # 弹出元素所属的列表

  2) "update system..."                    # 弹出元素所属的值

  为什么要阻塞版本的pop呢，主要是为了避免轮询。举个简单的例子如果我们用list来实现一个工作队列。执行任务的thread可以调用阻塞版本的pop去获取任务这样就可以避免轮询去检查是否有任务存在。当任务来时候工作线程可以立即返回，也可以避免轮询带来的延迟。

set

4、Set

常用命令：

    sadd,srem,spop,sdiff ,smembers,sunion 等。

应用场景：

    Redis set对外提供的功能与list类似是一个列表的功能，特殊之处在于set是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，set是一个很好的选择，并且set提供了判断某个成员是否在一个set集合内的重要接口，这个也是list所不能提供的。

    比如在微博应用中，每个人的好友存在一个集合（set）中，这样求两个人的共同好友的操作，可能就只需要用求交集命令即可。

    Redis还为集合提供了求交集、并集、差集等操作，可以非常方便的实

实现方式：

    set 的内部实现是一个 value永远为null的HashMap，实际就是通过计算hash的方式来快速排重的，这也是set能提供判断一个成员是否在集合内的原因。

Sort Set

5、Sorted set

  常用命令：

    zadd,zrange,zrem,zcard等

  使用场景：

    以某个条件为权重，比如按顶的次数排序.

    ZREVRANGE命令可以用来按照得分来获取前100名的用户，ZRANK可以用来获取用户排名，非常直接而且操作容易。

    Redis sorted set的使用场景与set类似，区别是set不是自动有序的，而sorted set可以通过用户额外提供一个优先级(score)的参数来为成员排序，并且是插入有序的，即自动排序。

    比如:twitter 的public timeline可以以发表时间作为score来存储，这样获取时就是自动按时间排好序的。

    比如:全班同学成绩的SortedSets，value可以是同学的学号，而score就可以是其考试得分，这样数据插入集合的，就已经进行了天然的排序。

    另外还可以用Sorted Sets来做带权重的队列，比如普通消息的score为1，重要消息的score为2，然后工作线程可以选择按score的倒序来获取工作任务。让重要的任务优先执行。

    需要精准设定过期时间的应用

    比如你可以把上面说到的sorted set的score值设置成过期时间的时间戳，那么就可以简单地通过过期时间排序，定时清除过期数据了，不仅是清除Redis中的过期数据，你完全可以把Redis里这个过期时间当成是对数据库中数据的索引，用Redis来找出哪些数据需要过期删除，然后再精准地从数据库中删除相应的记录。

  实现方式：

    Redis sorted set的内部使用HashMap和跳跃表(SkipList)来保证数据的存储和有序，HashMap里放的是成员到score的映射，而跳跃表里存放的是所有的成员，排序依据是HashMap里存的score,使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。

   6、 Pub/Sub

    Pub/Sub 从字面上理解就是发布（Publish）与订阅（Subscribe），在Redis中，你可以设定对某一个key值进行消息发布及消息订阅，

    当一个key值上进行了消息发布后，所有订阅它的客户端都会收到相应的消息。这一功能最明显的用法就是用作实时消息系统，比如普通的即时聊天，群聊等功能。

客户端1：subscribe  rain

客户端2：PUBLISH  rain "my love!!!"

    (integer) 2 代表有几个客户端订阅了这个消息

 7、Transactions

    谁说NoSQL都不支持事务，虽然Redis的Transactions提供的并不是严格的ACID的事务（比如一串用EXEC提交执行的命令，在执行中服务器宕机，那么会有一部分命令执行了，剩下的没执行），但是这个Transactions还是提供了基本的命令打包执行的功能（在服务器不出问题的情况下，可以保证一连串的命令是顺序在一起执行的，中间有会有其它客户端命令插进来执行）。

    Redis还提供了一个Watch功能，你可以对一个key进行Watch，然后再执行Transactions，在这过程中，如果这个Watched的值进行了修改，那么这个Transactions会发现并拒绝执行。

Session 1

    (1)第1步

    redis 127.0.0.1:6379> get age

    "10"

    redis 127.0.0.1:6379> watch age

    OK

    redis 127.0.0.1:6379> multi

    OK

    redis 127.0.0.1:6379>

Session 2

    (2)第2步

    redis 127.0.0.1:6379> set age 30

    OK

    redis 127.0.0.1:6379> get age

    "30"

    redis 127.0.0.1:6379>

Session 1

    (3)第3步

    redis 127.0.0.1:6379> set age 20

    QUEUED

    redis 127.0.0.1:6379> exec

    (nil)

    redis 127.0.0.1:6379> get age

    "30"

    redis 127.0.0.1:6379>

    第一步，Session 1 还没有来得及对age的值进行修改

　　第二步，Session 2 已经将age的值设为30

　　第三步，Session 1 希望将age的值设为20，但结果一执行返回是nil，说明执行失败，之后我们再取一下age的值是30，这是由于Session   1中对age加了乐观锁导致的。