**1、为什么用NoSQL**

今天我们可以通过第三方平台(如：Google,Facebook等)可以很容易的访问和抓取数据。用户的个人信息，社交网络，地理位置，用户生成的数据和用户操作日志已经成倍的增加。我们如果要对这些用户数据进行挖掘，那SQL数据库已经不适合这些应用了, NoSQL数据库的发展却能很好的处理这些大的数据。

**2、NoSQL是什么**

NoSQL(NoSQL = Not Only SQL )，意即“不仅仅是SQL”，泛指非关系型的数据库。随着互联网web2.0网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的SNS(社交性网络服务)类型的web2.0纯动态网站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。NoSQL数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题，包括超大规模数据的存储。(例如谷歌或Facebook每天为他们的用户收集万亿比特的数据)。这些类型的数据存储不需要固定的模式，无需多余操作就可以横向扩展。

**3、NoSQL能干什么**

NoSQL数据库**种类繁多**，但是一个共同的特点都是**去掉关系数据库的关系型特性**。

数据之间无关系，这样就非常容易扩展。于无形之间，在架构的层面上带来了可扩展的能力。

NoSQL数据库都具有非常高的读写性能，尤其在大数据量下，同样表现优秀。这得益于它的无关系性，数据库的结构简单。

　　一般MySQL使用Query Cache，每次表的更新Cache就失效，是一种大粒度的Cache，在针对web2.0的交互频繁的应用，Cache性能不高。而NoSQL的Cache是记录级的，是一种细粒度的Cache，所以NoSQL在这个层面上来说就要性能高很多了

**redis每秒钟写8万，读11万次**

NoSQL无需事先为要存储的数据建立字段，随时可以存储自定义的数据格式。而在关系数据库里，增删字段是一件非常麻烦的事情。如果是非常大数据量的表，增加字段简直就是一个噩梦。

Expand:

NoSQL的分类：

基于列簇的：HBase

基于图形的：Neo4J

基于文件的：MongoDB

基于k-v的：Redis

**4、Redis简介**

redis是一个高性能的**key-value非关系数据库**，它可以存键(key)与5种不同类型的值(value)之间的映射(mapping)，支持存储的value类型包括:**String(字符串)、list(链表)、set(集合)、zset(有序集合)和hash(散列表)**。这些收据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是**原子性**的。在此基础上，redis支持各种不同方式的排序。与memcached一样，为了保证效率，数据都是**缓存在内存**中。区别的是**redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件**，并且在此基础上实现了master-slave(主从)同步。

##### 4.3redis数据类型之String

4.3.1概述

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。

4.3.2相关命令

1、赋值命令

命令原型：SET key value

命令描述：设定该Key持有指定的字符串Value，如果该Key已经存在，则覆盖其原有值。

返回值：总是返回"OK"。

2、取值命令

命令原型：GET key

命令描述：获取指定Key的Value。如果与该Key关联的Value不是string类型，Redis将返回错误信息，因为GET命令只能用于获取string Value。

返回值：与该Key相关的Value，如果该Key不存在，返回nil。

3、GETSET命令

命令原型：GETSET key value

命令描述：原子性的设置该Key为指定的Value，同时返回该Key的原有值。和GET命令一样，该命令也只能处理string Value，否则Redis将给出相关的错误信息。

返回值：返回该Key的原有值，如果该Key之前并不存在，则返回nil。

4、递增数字

命令原型：INCR key

命令描述：将指定Key的Value原子性的递增1。如果该Key不存在，其初始值为0，在incr之后其值为1。如果Value的值不能转换为整型值，如Hello，该操作将执行失败并返回相应的错误信息。注意：该操作的取值范围是64位有符号整型。

返回值：递增后的Value值。

5、递减数字

命令原型：DECR key

命令描述：将指定Key的Value原子性的递减1。如果该Key不存在，其初始值为0，在decr之后其值为-1。如果Value的值不能转换为整型值，如Hello，该操作将执行失败并返回相应的错误信息。注意：该操作的取值范围是64位有符号整型。

返回值：递增后的Value值。

6、增加指定的整数

命令原型：INCRBY key increment

命令描述：将指定Key的Value原子性的增加increment。如果该Key不存在，其初始值为0，在incrby之后其值为increment。如果Value的值不能转换为整型值，如Hello，该操作将执行失败并返回相应的错误信息。注意：该操作的取值范围是64位有符号整型。

返回值：增加后的value值。

7、减少指定的整数

命令原型：DECRBY key decrement

命令描述：将指定Key的Value原子性的减少decrement。如果该Key不存在，其初始值为0，在decrby之后其值为-decrement。如果Value的值不能转换为整型值，如Hello，该操作将执行失败并返回相应的错误信息。注意：该操作的取值范围是64位有符号整型。

返回值：减少后的value值。

8、SETEX命令

命令原型：SETEX key seconds value

命令描述：原子性完成两个操作，一是设置该Key的值为指定字符串，同时设置该Key在Redis服务器中的存活时间(秒数)。该命令主要应用于Redis被当做Cache服务器使用时。

返回值：

9、SETNX命令

命令原型：SETNX key value

命令描述：如果指定的Key不存在，则设定该Key持有指定字符串Value，此时其效果等价于SET命令。相反，如果该Key已经存在，该命令将不做任何操作并返回。

返回值：1表示设置成功，否则0。

10、向尾部追加值

命令原型：APPEND key value

命令描述：如果该Key已经存在，APPEND命令将参数Value的数据追加到已存在Value的末尾。如果该Key不存在，APPEND命令将会创建一个新的Key/Value。

返回值：追加后Value的长度。

11、获取字符串长度

命令原型：STRLEN key

命令描述：返回指定Key的字符值长度，如果Value不是string类型，Redis将执行失败并给出相关的错误信息。

返回值：返回指定Key的Value字符长度，如果该Key不存在，返回0。

12、设置多个键值

命令原型：MSET key value [key value ...]

命令描述：N表示指定Key的数量。该命令原子性的完成参数中所有key/value的设置操作，其具体行为可以看成是多次迭代执行SET命令。

返回值：该命令不会失败，始终返回OK。

13、获取多个键值

命令原型：MGET key [key ...]

命令描述：N表示获取Key的数量。返回所有指定Keys的Values，如果其中某个Key不存在，或者其值不为string类型，该Key的Value将返回nil。

返回值：返回一组指定Keys的Values的列表。

14、MSETNX命令

命令原型：MSETNX key value [key value ...]

命令描述：N表示指定Key的数量。该命令原子性的完成参数中所有key/value的设置操作，其具体行为可以看成是多次迭代执行SETNX命令。然而这里需要明确说明的是，如果在这一批Keys中有任意一个Key已经存在了，那么该操作将全部回滚，即所有的修改都不会生效。

返回值：1表示所有Keys都设置成功，0则表示没有任何Key被修改。

4.3.3命令示例

1. SET/GET/APPEND/STRLEN:  
redis 127.0.0.1:6379> exists mykey #判断该键是否存在，存在返回1，否则返回0。  
    (integer) 0  
redis 127.0.0.1:6379> append mykey "hello"      #该键并不存在，因此append命令返回当前Value的长度。  
    (integer) 5  
redis 127.0.0.1:6379> append mykey " world"    #该键已经存在，因此返回追加后Value的长度。  
    (integer) 11  
redis 127.0.0.1:6379> get mykey  #通过get命令获取该键，以判断append的结果。  
    "hello world"  
redis 127.0.0.1:6379> set mykey "this is a test" #通过set命令为键设置新值，并覆盖原有值。  
    OK  
    redis 127.0.0.1:6379> get mykey  
    "this is a test"  
redis 127.0.0.1:6379> strlen mykey    #获取指定Key的字符长度，等效于C库中strlen函数。  
    (integer) 14

2. INCR/DECR/INCRBY/DECRBY:  
    redis 127.0.0.1:6379> set mykey 20     #设置Key的值为20  
    OK  
    redis 127.0.0.1:6379> incr mykey         #该Key的值递增1  
    (integer) 21  
    redis 127.0.0.1:6379> decr mykey        #该Key的值递减1  
    (integer) 20  
    redis 127.0.0.1:6379> del mykey          #删除已有键。  
    (integer) 1  
    redis 127.0.0.1:6379> decr mykey        #对空值执行递减操作，其原值被设定为0，递减后的值为-1  
    (integer) -1  
    redis 127.0.0.1:6379> del mykey     
    (integer) 1  
    redis 127.0.0.1:6379> incr mykey #对空值执行递增操作，其原值被设定为0，递增后的值为1  
    (integer) 1  
    redis 127.0.0.1:6379> set mykey hello #将该键的Value设置为不能转换为整型的普通字符串。  
    OK  
    redis 127.0.0.1:6379>   
    (error) ERR value is not an integer or out of range  
    redis 127.0.0.1:6379> set mykey 10  
    OK  
    redis 127.0.0.1:6379> decrby mykey 5   
    (integer) 5  
    redis 127.0.0.1:6379> incrby mykey 10  
    (integer) 15

 3. GETSET：  
    redis 127.0.0.1:6379> incr mycounter  #将计数器的值原子性的递增1  
    (integer) 1  
    #在获取计数器原有值的同时，并将其设置为新值，这两个操作原子性的同时完成。  
    redis 127.0.0.1:6379> getset mycounter 0    
    "1"  
    redis 127.0.0.1:6379> get mycounter       #查看设置后的结果。  
    "0"

 4. SETEX:  
    redis 127.0.0.1:6379> setex mykey 10 "hello"   #设置指定Key的过期时间为10秒。  
    OK      
    #通过ttl命令查看一下指定Key的剩余存活时间(秒数)，0表示已经过期，-1表示永不过期。  
    redis 127.0.0.1:6379> ttl mykey                         
    (integer) 4  
    redis 127.0.0.1:6379> get mykey   #在该键的存活期内我们仍然可以获取到它的Value。  
    "hello"  
    redis 127.0.0.1:6379> ttl mykey   #该ttl命令的返回值显示，该Key已经过期。  
    (integer) 0  
    redis 127.0.0.1:6379> get mykey  #获取已过期的Key将返回nil。  
    (nil)

 5. SETNX:  
    redis 127.0.0.1:6379> del mykey #删除该键，以便于下面的测试验证。  
    (integer) 1  
    redis 127.0.0.1:6379> setnx mykey "hello" #该键并不存在，因此该命令执行成功。  
    (integer) 1  
    redis 127.0.0.1:6379> setnx mykey "world"#该键已经存在，因此本次设置没有产生任何效果。  
    (integer) 0  
    redis 127.0.0.1:6379> get mykey #从结果可以看出，返回的值仍为第一次设置的值。  
    "hello"

1. MSET/MGET/MSETNX:  
       redis 127.0.0.1:6379> mset key1 "hello" key2 "world"   #批量设置了key1和key2两个键。  
       OK  
   redis 127.0.0.1:6379> mget key1 key2  #批量获取了key1和key2两个键的值。  
       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)

##### 4.4redis数据类型之List

一、概述

在Redis中，List类型是按照插入顺序排序的字符串链表。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。

二、相关命令

1、向列表左边添加元素

命令原型：LPUSH key value [value ...]

命令描述：在指定Key所关联的List Value的头部插入参数中给出的所有Values。如果该Key不存在，该命令将在插入之前创建一个与该Key关联的空链表，之后再将数据从链表的头部插入。如果该键的Value不是链表类型，该命令将返回相关的错误信息。

返回值：插入后链表中元素的数量。

2、向列表右边添加元素

命令原型：RPUSH key value [value ...]

命令描述：在指定Key所关联的List Value的尾部插入参数中给出的所有Values。如果该Key不存在，该命令将在插入之前创建一个与该Key关联的空链表，之后再将数据从链表的尾部插入。如果该键的Value不是链表类型，该命令将返回相关的错误信息。

返回值：插入后链表中元素的数量。

3、LPUSHX命令

命令原型：LPUSHX key value

命令描述：仅有当参数中指定的Key存在时，该命令才会在其所关联的List Value的头部插入参数中给出的Value，否则将不会有任何操作发生。

返回值：插入后链表中元素的数量。

4、RPUSHX命令

命令原型：RPUSHX key value

命令描述：仅有当参数中指定的Key存在时，该命令才会在其所关联的List Value的尾部插入参数中给出的Value，否则将不会有任何操作发生。

返回值：插入后链表中元素的数量。

5、从列表左边弹出元素

命令原型：LPOP key

命令描述：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不存在，返回nil。LPOP命令执行两步操作：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。

返回值：链表头部的元素。

6、从列表右边弹出元素

命令原型：RPOP key

命令描述：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不存在，返回nil。

返回值：链表尾部的元素。

7、获取列表中元素的个数

命令原型：LLEN key

命令描述：返回指定Key关联的链表中元素的数量，如果该Key不存在，则返回0。如果与该Key关联的Value的类型不是链表，则返回相关的错误信息。

返回值：链表中元素的数量。

8、获得列表片段

命令原型：LRANGE key start stop

命令描述：时间复杂度中的S为start参数表示的偏移量，N表示元素的数量。该命令的参数start和end都是0-based。即0表示链表头部(leftmost)的第一个元素。其中start的值也可以为负值，-1将表示链表中的最后一个元素，即尾部元素，-2表示倒数第二个并以此类推。该命令在获取元素时，start和end位置上的元素也会被取出。如果start的值大于链表中元素的数量，空链表将会被返回。如果end的值大于元素的数量，该命令则获取从start(包括start)开始，链表中剩余的所有元素。注：Redis的列表起始索引为0。显然，LRANGE numbers 0 -1 可以获取列表中的所有元素。

返回值：返回指定范围内元素的列表。

9、删除列表中指定的值

命令原型：LREM key count value

在指定Key关联的链表中，删除前count个值等于value的元素。如果count大于0，从头向尾遍历并删除，如果count小于0，则从尾向头遍历并删除。如果count等于0，则删除链表中所有等于value的元素。如果指定的Key不存在，则直接返回0。

返回值：返回被删除的元素数量。

10、设置指定索引的元素值

命令原型：LSET key index value

命令描述：时间复杂度中N表示链表中元素的数量。但是设定头部或尾部的元素时，其时间复杂度为O(1)。设定链表中指定位置的值为新值，其中0表示第一个元素，即头部元素，-1表示尾部元素。如果索引值Index超出了链表中元素的数量范围，该命令将返回相关的错误信息。

返回值：

11、获取指定索引的元素值

命令原型：LINDEX key index

命令描述：时间复杂度中N表示在找到该元素时需要遍历的元素数量。对于头部或尾部元素，其时间复杂度为O(1)。该命令将返回链表中指定位置(index)的元素，index是0-based，表示头部元素，如果index为-1，表示尾部元素。如果与该Key关联的不是链表，该命令将返回相关的错误信息。

返回值：返回请求的元素，如果index超出范围，则返回nil。

12、只保留列表指定片段

命令原型：LTRIM key start stop

命令描述：N表示被删除的元素数量。该命令将仅保留指定范围内的元素，从而保证链接中的元素数量相对恒定。start和stop参数都是0-based，0表示头部元素。和其他命令一样，start和stop也可以为负值，-1表示尾部元素。如果start大于链表的尾部，或start大于stop，该命令不报错，而是返回一个空的链表，与此同时该Key也将被删除。如果stop大于元素的数量，则保留从start开始剩余的所有元素。

返回值：

13、向列表中插入元素

命令原型：LINSERT key BEFORE|AFTER pivot value

命令描述：时间复杂度中N表示在找到该元素pivot之前需要遍历的元素数量。这样意味着如果pivot位于链表的头部或尾部时，该命令的时间复杂度为O(1)。该命令的功能是在pivot元素的前面或后面插入参数中的元素value。如果Key不存在，该命令将不执行任何操作。如果与Key关联的Value类型不是链表，相关的错误信息将被返回。

返回值：成功插入后链表中元素的数量，如果没有找到pivot，返回-1，如果key不存在，返回0。

14、将元素从一个列表转到另一个列表

命令原型：RPOPLPUSH source destination

命令描述：原子性的从与source键关联的链表尾部弹出一个元素，同时再将弹出的元素插入到与destination键关联的链表的头部。如果source键不存在，该命令将返回nil，同时不再做任何其它的操作了。如果source和destination是同一个键，则相当于原子性的将其关联链表中的尾部元素移到该链表的头部。

RPOPLPUSH是个很有意思的命令，从名字就可以看出它的功能：先执行RPOP命令再执行LPUSH命令。RPOPLPUSH命令会先从source 列表类型键的右边弹出一个元素，然后将其加入到destination 列表类型键的左边，并返回这个元素的值，整个过程是原子的。

返回值：返回弹出和插入的元素。

三、命令示例

1. LPUSH/LPUSHX/LRANGE:

/> redis-cli    #在Shell提示符下启动redis客户端工具。  
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  
    (integer) 4  
    #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  
    1) "d"  
    2) "c"  
    3) "b"  
    4) "a"  
    #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  
    (integer) 0  
    #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  
    (integer) 5  
    #获取该键的List Value的头部元素。  
    redis 127.0.0.1:6379> lrange mykey 0 0  
    1) "e"

2. LPOP/LLEN:

redis 127.0.0.1:6379> del mykey  
    (integer) 1  
    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. LREM/LSET/LINDEX/LTRIM:

　 redis 127.0.0.1:6379> del mykey  
    (integer) 1  
 #为后面的示例准备测试数据。  
    redis 127.0.0.1:6379> lpush mykey a b c d a c  
    (integer) 6  
 #从头部(left)向尾部(right)变量链表，删除2个值等于a的元素，返回值为实际删除的数量。  
    redis 127.0.0.1:6379> lrem mykey 2 a  
    (integer) 2  
    #看出删除后链表中的全部元素。  
    redis 127.0.0.1:6379> lrange mykey 0 -1  
    1) "c"  
    2) "d"  
    3) "c"  
    4) "b"  
    #获取索引值为1(头部的第二个元素)的元素值。  
    redis 127.0.0.1:6379> lindex mykey 1  
    "d"  
    #将索引值为1(头部的第二个元素)的元素值设置为新值e。  
    redis 127.0.0.1:6379> lset mykey 1 e  
    OK  
    #查看是否设置成功。  
    redis 127.0.0.1:6379> lindex mykey 1  
    "e"  
    #索引值6超过了链表中元素的数量，该命令返回nil。  
    redis 127.0.0.1:6379> lindex mykey 6  
    (nil)  
    #设置的索引值6超过了链表中元素的数量，设置失败，该命令返回错误信息。  
    redis 127.0.0.1:6379> lset mykey 6 hh  
    (error) ERR index out of range  
    #仅保留索引值0到2之间的3个元素，注意第0个和第2个元素均被保留。  
    redis 127.0.0.1:6379> ltrim mykey 0 2  
    OK  
    #查看trim后的结果。  
    redis 127.0.0.1:6379> lrange mykey 0 -1  
    1) "c"  
    2) "e"  
    3) "c"

4. LINSERT:

#删除该键便于后面的测试。  
    redis 127.0.0.1:6379> del mykey  
    (integer) 1  
    #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  
    (integer) 5  
    #在a的前面插入新元素a1。  
    redis 127.0.0.1:6379> linsert mykey before a a1  
    (integer) 6  
  #查看是否插入成功，从结果看已经插入。注意lindex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  
    (integer) 7  
    #再次查看是否插入成功。  
    redis 127.0.0.1:6379> lindex mykey 1  
    "e2"  
    #在不存在的元素之前或之后插入新元素，该命令操作失败，并返回-1。  
    redis 127.0.0.1:6379> linsert mykey after k a  
    (integer) -1  
    #为不存在的Key插入新元素，该命令操作失败，返回0。  
    redis 127.0.0.1:6379> linsert mykey1 after a a2  
    (integer) 0

5. 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"  
    2) "b"  
    3) "c"  
    4) "d"  
    #该键已经存在并且包含4个元素，rpushx命令将执行成功，并将元素e插入到链表的尾部。  
    redis 127.0.0.1:6379> rpushx mykey e  
    (integer) 5  
    #通过lindex命令可以看出之前的rpushx命令确实执行成功，因为索引值为4的元素已经是新元素了。  
    redis 127.0.0.1:6379> lindex mykey 4  
    "e"  
    #由于mykey2键并不存在，因此该命令不会插入数据，其返回值为0。  
    redis 127.0.0.1:6379> rpushx mykey2 e  
    (integer) 0  
    #在执行rpoplpush命令前，先看一下mykey中链表的元素有哪些，注意他们的位置关系。  
    redis 127.0.0.1:6379> lrange mykey 0 -1  
    1) "a"  
    2) "b"  
    3) "c"  
    4) "d"  
    5) "e"  
    #将mykey的尾部元素e弹出，同时再插入到mykey2的头部(原子性的完成这两步操作)。  
    redis 127.0.0.1:6379> rpoplpush mykey mykey2  
    "e"  
    #通过lrange命令查看mykey在弹出尾部元素后的结果。  
    redis 127.0.0.1:6379> lrange mykey 0 -1  
    1) "a"  
    2) "b"  
    3) "c"  
    4) "d"  
    #通过lrange命令查看mykey2在插入元素后的结果。  
    redis 127.0.0.1:6379> lrange mykey2 0 -1  
    1) "e"  
    #将source和destination设为同一键，将mykey中的尾部元素移到其头部。  
    redis 127.0.0.1:6379> rpoplpush mykey mykey  
    "d"  
    #查看移动结果。  
    redis 127.0.0.1:6379> lrange mykey 0 -1  
    1) "d"  
    2) "a"  
    3) "b"  
    4) "c"

四、链表结构的小技巧

针对链表结构的Value，Redis在其官方文档中给出了一些实用技巧，如RPOPLPUSH命令，下面给出具体的解释。  
Redis链表经常会被用于消息队列的服务，以完成多程序之间的消息交换。假设一个应用程序正在执行LPUSH操作向链表中添加新的元素，我们通常将这样的程序称之为"生产者(Producer)"，而另外一个应用程序正在执行RPOP操作从链表中取出元素，我们称这样的程序为"消费者(Consumer)"。如果此时，消费者程序在取出消息元素后立刻崩溃，由于该消息已经被取出且没有被正常处理，那么我们就可以认为该消息已经丢失，由此可能会导致业务数据丢失，或业务状态的不一致等现象的发生。然而通过使用RPOPLPUSH命令，消费者程序在从主消息队列中取出消息之后再将其插入到备份队列中，直到消费者程序完成正常的处理逻辑后再将该消息从备份队列中删除。同时我们还可以提供一个守护进程，当发现备份队列中的消息过期时，可以重新将其再放回到主消息队列中，以便其它的消费者程序继续处理。

##### 4.5redis数据类型之Hash

一、概述

我们可以将Redis中的Hash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。

二、相关命令

1、赋值命令

命令原型：HSET key field value

命令描述：为指定的Key设定Field/Value对，如果Key不存在，该命令将创建新Key以用于存储参数中的Field/Value对，如果参数中的Field在该Key中已经存在，则用新值覆盖其原有值。

返回值：1表示新的Field被设置了新值，0表示Field已经存在，用新值覆盖原有值。

2、取值命令

命令原型：HGET key field

命令描述：返回指定Key中指定Field的关联值。

返回值：返回参数中Field的关联值，如果参数中的Key或Field不存在，返回nil。

3、判断字段是否存在

命令原型：HEXISTS key field

命令描述：判断指定Key中的指定Field是否存在。

返回值：1表示存在，0表示参数中的Field或Key不存在。

4、获得字段数量

命令原型：HLEN key

命令描述：获取该Key所包含的Field的数量。

返回值：返回Key包含的Field数量，如果Key不存在，返回0。

5、删除字段

命令原型：HDEL key field [field ...]

命令描述：时间复杂度中的N表示参数中待删除的字段数量。从指定Key的Hashes Value中删除参数中指定的多个字段，如果不存在的字段将被忽略。如果Key不存在，则将其视为空Hashes，并返回0。

返回值：实际删除的Field数量。

6、当字段不存在时赋值

命令原型：HSETNX key field value

命令描述：只有当参数中的Key或Field不存在的情况下，为指定的Key设定Field/Value对，否则该命令不会进行任何操作。

返回值：1表示新的Field被设置了新值，0表示Key或Field已经存在，该命令没有进行任何操作。

7、增加数字

命令原型：HINCRBY key field increment

命令描述：增加指定Key中指定Field关联的Value的值。如果Key或Field不存在，该命令将会创建一个新Key或新Field，并将其关联的Value初始化为0，之后再指定数字增加的操作。该命令支持的数字是64位有符号整型，即increment可以负数。

返回值：返回运算后的值。

8、获取指定键中所有的字段/值

命令原型：HGETALL key

命令描述：时间复杂度中的N表示Key包含的Field数量。获取该键包含的所有Field/Value。其返回格式为一个Field、一个Value，并以此类推。

返回值：Field/Value的列表。

9、只获取字段名

命令原型：HKEYS key

命令描述：时间复杂度中的N表示Key包含的Field数量。返回指定Key的所有Fields名。

返回值：Field的列表。

10、只获取字段值

命令原型：HVALS key

命令描述：时间复杂度中的N表示Key包含的Field数量。返回指定Key的所有Values名。

返回值：value的列表。

11、设置多个字段的值

命令原型：HMSET key field value [field value ...]

命令描述：时间复杂度中的N表示被设置的Field数量。逐对依次设置参数中给出的Field/Value对。如果其中某个Field已经存在，则用新值覆盖原有值。如果Key不存在，则创建新Key，同时设定参数中的Field/Value。

返回值：

12、获取多个字段的值

命令原型：HMGET key field [field ...]

命令描述：时间复杂度中的N表示请求的Field数量。获取和参数中指定Fields关联的一组Values。如果请求的Field不存在，其值返回nil。如果Key不存在，该命令将其视为空Hash，因此返回一组nil。

返回值：返回和请求Fields关联的一组Values，其返回顺序等同于Fields的请求顺序。

三、命令示例：

1. HSET/HGET/HDEL/HEXISTS/HLEN/HSETNX:  
 #在Shell命令行启动Redis客户端程序  
 /> redis-cli  
 #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  
    #再次删除myhash键中字段名为field1的字段，由于上一条命令已经将其删除，因为没有删除，返回0。  
    redis 127.0.0.1:6379> hdel myhash field1  
    (integer) 0  
    #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  
    #通过hsetnx命令给myhash添加新字段field1，其值为stephen，因为该字段已经被删除，所以该命令添加成功并返回1。  
    redis 127.0.0.1:6379> hsetnx myhash field1 stephen  
    (integer) 1  
    #由于myhash的field1字段已经通过上一条命令添加成功，因为本条命令不做任何操作后返回0。  
    redis 127.0.0.1:6379> hsetnx myhash field1 stephen  
    (integer) 0

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

 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键中所有字段的名字。  
    redis 127.0.0.1:6379> hkeys myhash  
    1) "field1"  
    2) "field2"  
    #仅获取myhash键中所有字段的值。  
    redis 127.0.0.1:6379> hvals myhash  
    1) "hello"  
    2) "world"

##### 4.6redis数据类型之Set

一、概述

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。Set可包含的最大元素数量是4294967295。  
和List类型不同的是，Set集合中不允许出现重复的元素，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。

二、相关命令

1、增加元素

命令原型：SADD key member [member ...]

命令描述：时间复杂度中的N表示操作的成员数量。如果在插入的过程用，参数中有的成员在Set中已经存在，该成员将被忽略，而其它成员仍将会被正常插入。如果执行该命令之前，该Key并不存在，该命令将会创建一个新的Set，此后再将参数中的成员陆续插入。如果该Key的Value不是Set类型，该命令将返回相关的错误信息。

返回值：本次操作实际插入的成员数量。

2、获得集合中元素个数

命令原型：SCARD key

命令描述：获取Set中成员的数量。

返回值：返回Set中成员的数量，如果该Key并不存在，返回0。

3、判断元素是否在集合中

命令原型：SISMEMBER key member

命令描述：判断参数中指定成员是否已经存在于与Key相关联的Set集合中。

返回值：1表示已经存在，0表示不存在，或该Key本身并不存在。

4、获得集合中的所有元素

命令原型：SMEMBERS key

命令描述：时间复杂度中的N表示Set中已经存在的成员数量。获取与该Key关联的Set中所有的成员。

返回值：返回Set中所有的成员。

5、从集合中弹出一个元素

命令原型：SPOP key

命令描述：随机的移除并返回Set中的某一成员。 由于Set中元素的布局不受外部控制，因此无法像List那样确定哪个元素位于Set的头部或者尾部。

返回值：返回移除的成员，如果该Key并不存在，则返回nil。

6、删除元素

命令原型：SREM key member [member ...]

命令描述：时间复杂度中的N表示被删除的成员数量。从与Key关联的Set中删除参数中指定的成员，不存在的参数成员将被忽略，如果该Key并不存在，将视为空Set处理。

返回值：从Set中实际移除的成员数量，如果没有则返回0。

7、随机获得集合中的元素

命令原型：SRANDMEMBER key [count]

命令描述：和SPOP一样，随机的返回Set中的一个成员，不同的是该命令并不会删除返回的成员。还可以传递count参数来一次随机获得多个元素，根据count的正负不同，具体表现也不同。当count 为正数时，SRANDMEMBER 会随机从集合里获得count个不重复的元素。如果count的值大于集合中的元素个数，则SRANDMEMBER 会返回集合中的全部元素。当count为负数时，SRANDMEMBER 会随机从集合里获得|count|个的元素，这些元素有可能相同。

返回值：返回随机位置的成员，如果Key不存在则返回nil。

8、将元素从一个集合转到另一个集合

命令原型：SMOVE source destination member

命令描述：原子性的将参数中的成员从source键移入到destination键所关联的Set中。因此在某一时刻，该成员或者出现在source中，或者出现在destination中。如果该成员在source中并不存在，该命令将不会再执行任何操作并返回0，否则，该成员将从source移入到destination。如果此时该成员已经在destination中存在，那么该命令仅是将该成员从source中移出。如果和Key关联的Value不是Set，将返回相关的错误信息。

返回值：1表示正常移动，0表示source中并不包含参数成员。

9、集合间差集运算

命令原型：SDIFF key [key ...]

命令描述：时间复杂度中的N表示所有Sets中成员的总数量。返回参数中第一个Key所关联的Set和其后所有Keys所关联的Sets中成员的差异。如果Key不存在，则视为空Set。

返回值：差异结果成员的集合。

10、集合间差集运算并将结果存储

命令原型：SDIFFSTORE destination key [key ...]

命令描述：该命令和SDIFF命令在功能上完全相同，两者之间唯一的差别是SDIFF返回差异的结果成员，而该命令将差异成员存储在destination关联的Set中。如果destination键已经存在，该操作将覆盖它的成员。

返回值：返回差异成员的数量。

11、集合间交集运算

命令原型：SINTER key [key ...]

命令描述：时间复杂度中的N表示最小Set中元素的数量，M则表示参数中Sets的数量。该命令将返回参数中所有Keys关联的Sets中成员的交集。因此如果参数中任何一个Key关联的Set为空，或某一Key不存在，那么该命令的结果将为空集。

返回值：交集结果成员的集合。

12、集合间交集运算并将结果存储

命令原型：SINTERSTORE destination key [key ...]

命令描述：命令和SINTER命令在功能上完全相同，两者之间唯一的差别是SINTER返回交集的结果成员，而该命令将交集成员存储在destination关联的Set中。如果destination键已经存在，该操作将覆盖它的成员。

返回值：返回交集成员的数量。

13、集合间并集运算

命令原型：SUNION key [key ...]

命令描述：时间复杂度中的N表示所有Sets中成员的总数量。该命令将返回参数中所有Keys关联的Sets中成员的并集。

返回值：并集结果成员的集合。

14、集合间并集运算并将结果存储

命令原型：SUNIONSTORE destination key [key ...]

命令描述：该命令和SUNION命令在功能上完全相同，两者之间唯一的差别是SUNION返回并集的结果成员，而该命令将并集成员存储在destination关联的Set中。如果destination键已经存在，该操作将覆盖它的成员。

返回值：返回并集成员的数量。

三、命令示例

1. SADD/SMEMBERS/SCARD/SISMEMBER:  
    #在Shell命令行下启动Redis的客户端程序。  
    /> redis-cli

    #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

 2. SPOP/SREM/SRANDMEMBER/SMOVE:  
    #删除该键，便于后面的测试。  
    redis 127.0.0.1:6379> del myset  
    (integer) 1  
    #为后面的示例准备测试数据。  
    redis 127.0.0.1:6379> sadd myset a b c d  
    (integer) 4  
    #查看Set中成员的位置。  
    redis 127.0.0.1:6379> smembers myset  
    1) "c"  
    2) "d"  
    3) "a"  
    4) "b"  
    #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> srandmember 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  
    #查看移出后的输出结果。  
    redis 127.0.0.1:6379> smembers myset  
    1) "c"  
    #为后面的smove命令准备数据。  
    redis 127.0.0.1:6379> sadd myset a b  
    (integer) 2  
    redis 127.0.0.1:6379> sadd myset2 c d  
    (integer) 2  
    #将a从myset移到myset2，从结果可以看出移动成功。  
    redis 127.0.0.1:6379> smove myset myset2 a  
    (integer) 1  
    #再次将a从myset移到myset2，由于此时a已经不是myset的成员了，因此移动失败并返回0。  
    redis 127.0.0.1:6379> smove myset myset2 a  
    (integer) 0  
    #分别查看myset和myset2的成员，确认移动是否真的成功。  
    redis 127.0.0.1:6379> smembers myset  
    1) "b"  
    redis 127.0.0.1:6379> smembers myset2  
    1) "c"  
    2) "d"  
    3) "a"

 3. SDIFF/SDIFFSTORE/SINTER/SINTERSTORE:  
    #为后面的命令准备测试数据。  
    redis 127.0.0.1:6379> sadd myset a b c d  
    (integer) 4  
    redis 127.0.0.1:6379> sadd myset2 c  
    (integer) 1  
    redis 127.0.0.1:6379> sadd myset3 a c e  
    (integer) 3  
    #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"  
    #将3个集合的差异成员存在在diffkey关联的Set中，并返回插入的成员数量。  
    redis 127.0.0.1:6379> sdiffstore diffkey myset myset2 myset3  
    (integer) 2  
    #查看一下sdiffstore的操作结果。  
    redis 127.0.0.1:6379> smembers diffkey  
    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"  
    #将3个集合中的交集成员存储到与interkey关联的Set中，并返回交集成员的数量。  
    redis 127.0.0.1:6379> sinterstore interkey myset myset2 myset3  
    (integer) 1  
    #查看一下sinterstore的操作结果。  
    redis 127.0.0.1:6379> smembers interkey  
    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"  
    #将3个集合中成员的并集存储到unionkey关联的set中，并返回并集成员的数量。  
    redis 127.0.0.1:6379> sunionstore unionkey myset myset2 myset3  
    (integer) 5  
    #查看一下suiionstore的操作结果。  
    redis 127.0.0.1:6379> smembers unionkey  
    1) "b"  
    2) "c"  
    3) "d"  
    4) "e"  
    5) "a"

四、应用范围

     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。

##### 4.7redis数据类型之Sorted Set

一、概述：

Sorted Set（有序集合）和Set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 Set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 Set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。由于Sorted Set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。

二、相关命令

1、增加元素

命令原型：ZADD key score member [score] [member]

命令描述：时间复杂度中的N表示Sorted Set中成员的数量。添加参数中指定的所有成员及其分数到指定key的Sorted Set中，在该命令中我们可以指定多组score/member作为参数。如果在添加时参数中的某一成员已经存在，该命令将更新此成员的分数为新值，同时再将该成员基于新值重新排序。如果键不存在，该命令将为该键创建一个新的Sorted Set Value，并将score/member对插入其中。如果该键已经存在，但是与其关联的Value不是Sorted Set类型，相关的错误信息将被返回。

返回值：本次操作实际插入的成员数量。

2、获得集合中元素个数

命令原型：ZCARD key

命令描述：获取与该Key相关联的Sorted Set中包含的成员数量。

返回值：返回Sorted Set中的成员数量，如果该Key不存在，返回0。

3、获得指定分数范围内的元素个数

命令原型：ZCOUNT key min max

命令描述：时间复杂度中的N表示Sorted-Sets中成员的数量，M则表示min和max之间元素的数量。该命令用于获取分数(score)在min和max之间的成员数量。针对min和max参数需要额外说明的是，-inf和+inf分别表示Sorted-Sets中分数的最高值和最低值。缺省情况下，min和max表示的范围是闭区间范围，即min <= score <= max内的成员将被返回。然而我们可以通过在min和max的前面添加"("字符来表示开区间，如(min max表示min < score <= max，而(min (max表示min < score < max。

返回值：分数指定范围内成员的数量。

4、获得元素的分数

命令原型：ZSCORE key member

命令描述：获取指定Key的指定成员的分数。

返回值：如果该成员存在，以字符串的形式返回其分数，否则返回nil。

5、增加某个元素的分数

命令原型：ZINCRBY key increment member

命令描述：时间复杂度中的N表示Sorted Set中成员的数量。该命令将为指定Key中的指定成员增加指定的分数。如果成员不存在，该命令将添加该成员并假设其初始分数为0，此后再将其分数加上increment。如果Key不存在，该命令将创建该Key及其关联的Sorted Set，并包含参数指定的成员，其分数为increment参数。如果与该Key关联的不是Sorted Set类型，相关的错误信息将被返回。

返回值：以字符串形式表示的新分数。

6、获得排名在某个范围的元素列表

命令原型：ZRANGE key start stop [WITHSCORES]

命令描述：时间复杂度中的N表示Sorted Set中成员的数量，M则表示返回的成员数量。该命令返回顺序在参数start和stop指定范围内的成员，这里start和stop参数都是0-based，即0表示第一个成员，-1表示最后一个成员。如果start大于该Sorted Set中的最大索引值，或start > stop，此时一个空集合将被返回。如果stop大于最大索引值，该命令将返回从start到集合的最后一个成员。如果命令中带有可选参数WITHSCORES选项，该命令在返回的结果中将包含每个成员的分数值，如value1,score1,value2,score2...。

返回值：返回索引在start和stop之间的成员列表。

7、获得排名在某个范围的元素列表（元素分数从大到小排序）

命令原型：ZREVRANGE key start stop [WITHSCORES]

命令描述：时间复杂度中的N表示Sorted Set中成员的数量，M则表示返回的成员数量。该命令的功能和ZRANGE基本相同，唯一的差别在于该命令是通过反向排序获取指定位置的成员，即从高到低的顺序。如果成员具有相同的分数，则按降序字典顺序排序。

返回值：返回指定的成员列表。

8、获得指定分数范围的元素

命令原型：ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]

命令描述：时间复杂度中的N表示Sorted Set中成员的数量，M则表示返回的成员数量。该命令将返回分数在min和max之间的所有成员，即满足表达式min <= score <= max的成员，其中返回的成员是按照其分数从低到高的顺序返回，如果成员具有相同的分数，则按成员的字典顺序返回。可选参数LIMIT用于限制返回成员的数量范围。可选参数offset表示从符合条件的第offset个成员开始返回，同时返回count个成员。可选参数WITHSCORES的含义参照ZRANGE中该选项的说明。最后需要说明的是参数中min和max的规则可参照命令ZCOUNT。

返回值：返回分数在指定范围内的成员列表。

9、获得指定分数范围的元素（元素分数从大到小排序）

命令原型：ZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]

命令描述：时间复杂度中的N表示Sorted Set中成员的数量，M则表示返回的成员数量。该命令除了排序方式是基于从高到低的分数排序之外，其它功能和参数含义均与ZRANGEBYSCORE相同。需要注意的是该命令中的min和max参数的顺序和ZRANGEBYSCORE命令是相反的。

返回值：返回分数在指定范围内的成员列表。

10、获得元素的排名

命令原型：ZRANK key member

命令描述：时间复杂度中的N表示Sorted Set中成员的数量。Sorted Set中的成员都是按照分数从低到高的顺序存储，该命令将返回参数中指定成员的位置值，其中0表示第一个成员，它是Sorted Set中分数最低的成员。

返回值：如果该成员存在，则返回它的位置索引值。否则返回nil。

11、获得元素的排名（元素分数从大到小排序）

命令原型：ZREVRANK key member

命令描述：时间复杂度中的N表示Sorted Set中成员的数量。该命令的功能和ZRANK基本相同，唯一的差别在于该命令获取的索引是从高到低排序后的位置，同样0表示第一个元素，即分数最高的成员。

返回值：如果该成员存在，则返回它的位置索引值。否则返回nil。

12、删除一个或多个元素

命令原型：ZREM key member [member ...]

命令描述：时间复杂度中N表示Sorted Set中成员的数量，M则表示被删除的成员数量。该命令将移除参数中指定的成员，其中不存在的成员将被忽略。如果与该Key关联的Value不是Sorted Set，相应的错误信息将被返回。

返回值：实际被删除的成员数量。

13、按照排名范围删除元素

命令原型：ZREMRANGEBYRANK key start stop

命令描述：时间复杂度中的N表示Sorted Set中成员的数量，M则表示被删除的成员数量。删除索引位置位于start和stop之间的成员，start和stop都是0-based，即0表示分数最低的成员，-1表示最后一个成员，即分数最高的成员。

返回值：被删除的成员数量

14、按照分数范围删除元素

命令原型：ZREMRANGEBYSCORE key min max

命令描述：时间复杂度中的N表示Sorted Set中成员的数量，M则表示被删除的成员数量。删除分数在min和max之间的所有成员，即满足表达式min <= score <= max的所有成员。对于min和max参数，可以采用开区间的方式表示，具体规则参照ZCOUNT。

返回值：被删除的成员数量。

三、命令示例

1. ZADD/ZCARD/ZCOUNT/ZREM/ZINCRBY/ZSCORE/ZRANGE/ZRANK:  
    #在Shell的命令行下启动Redis客户端工具。  
    /> redis-cli  
    #添加一个分数为1的成员。  
    redis 127.0.0.1:6379> zadd myzset 1 "one"  
    (integer) 1  
    #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"  
    (integer) 2  
    #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  
    1) "one"  
    2) "1"  
    3) "two"  
    4) "2"  
    5) "three"  
    6) "3"  
    #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  
    (integer) 0  
    #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  
    (integer) 3  
    #返回与myzset关联的Sorted-Set中，分数满足表达式1 <= score <= 2的成员的数量。  
    redis 127.0.0.1:6379> zcount myzset 1 2  
    (integer) 2  
    #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  
    (integer) 2  
    #查看是否删除成功。  
    redis 127.0.0.1:6379> zcard myzset  
    (integer) 1  
    #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"  
    #由于成员two已经被删除，所以该命令返回nil。  
    redis 127.0.0.1:6379> zscore myzset two  
    (nil)

  #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  
    "2"  
    #将成员one的分数增加-1，并返回该成员更新后的分数。  
    redis 127.0.0.1:6379> zincrby myzset -1 one  
    "1"  
    #查看在更新了成员的分数后是否正确。  
    redis 127.0.0.1:6379> zrange myzset 0 -1 WITHSCORES  
    1) "one"  
    2) "1"  
    3) "three"  
    4) "3"

 2. ZRANGEBYSCORE/ZREMRANGEBYRANK/ZREMRANGEBYSCORE  
    redis 127.0.0.1:6379> del myzset  
    (integer) 1  
    redis 127.0.0.1:6379> zadd myzset 1 one 2 two 3 three 4 four

    (integer) 4  
    #获取分数满足表达式1 <= score <= 2的成员。  
    redis 127.0.0.1:6379> zrangebyscore myzset 1 2  
    1) "one"  
    2) "two"  
    #获取分数满足表达式1 < score <= 2的成员。  
    redis 127.0.0.1:6379> zrangebyscore myzset (1 2  
    1) "two"  
    #-inf表示第一个成员，+inf表示最后一个成员，limit后面的参数用于限制返回成员的自己，  
    #2表示从位置索引(0-based)等于2的成员开始，取后面3个成员。  
    redis 127.0.0.1:6379> zrangebyscore myzset -inf +inf limit 2 3  
    1) "three"  
    2) "four"  
    #删除分数满足表达式1 <= score <= 2的成员，并返回实际删除的数量。  
    redis 127.0.0.1:6379> zremrangebyscore myzset 1 2  
    (integer) 2  
    #看出一下上面的删除是否成功。  
    redis 127.0.0.1:6379> zrange myzset 0 -1  
    1) "three"  
    2) "four"  
    #删除位置索引满足表达式0 <= rank <= 1的成员。  
    redis 127.0.0.1:6379> zremrangebyrank myzset 0 1  
    (integer) 2  
    #查看上一条命令是否删除成功。  
    redis 127.0.0.1:6379> zcard myzset  
    (integer) 0

 3. ZREVRANGE/ZREVRANGEBYSCORE/ZREVRANK:  
    #为后面的示例准备测试数据。  
    redis 127.0.0.1:6379> del myzset  
    (integer) 0  
    redis 127.0.0.1:6379> zadd myzset 1 one 2 two 3 three 4 four  
    (integer) 4  
    #以位置索引从高到低的方式获取并返回此区间内的成员。  
    redis 127.0.0.1:6379> zrevrange myzset 0 -1 WITHSCORES  
    1) "four"  
    2) "4"  
    3) "three"  
    4) "3"  
    5) "two"  
    6) "2"  
    7) "one"  
    8) "1"  
    #由于是从高到低的排序，所以位置等于0的是four，1是three，并以此类推。  
    redis 127.0.0.1:6379> zrevrange myzset 1 3  
    1) "three"  
    2) "two"  
    3) "one"  
    #由于是从高到低的排序，所以one的位置是3。  
    redis 127.0.0.1:6379> zrevrank myzset one  
    (integer) 3  
    #由于是从高到低的排序，所以four的位置是0。  
    redis 127.0.0.1:6379> zrevrank myzset four  
    (integer) 0  
    #获取分数满足表达式3 >= score >= 0的成员，并以相反的顺序输出，即从高到底的顺序。  
    redis 127.0.0.1:6379> zrevrangebyscore myzset 3 0  
    1) "three"  
    2) "two"  
    3) "one"  
    #该命令支持limit选项，其含义等同于zrangebyscore中的该选项，只是在计算位置时按照相反的顺序计算和获取。  
    redis 127.0.0.1:6379> zrevrangebyscore myzset 4 0 limit 1 2

    1) "three"  
    2) "two"

四、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。