Redis 【第一篇】五种数据类型的常用命令以及应用场景模拟

文章目录

一、Redis基础知识介绍

1、什么是NoSql

NoSQL，泛指非关系型的数据库，NoSQL即Not-Only SQL，它可以作为关系型数据库的良好补充，是为了解决高并发、高可扩展、高可用、大数据存储问题而产生的数据库解决方案。

2、Nosql数据库分类

(1) 键值(Key-Value)存储数据

```
相关产品： Tokyo Cabinet/Tyrant、Redis、Voldemort、Berkeley DB
典型应用： 内容缓存，主要用于处理大量数据的高访问负载。
```

```
数据模型： 一系列键值对
优势： 快速查询
劣势： 存储的数据缺少结构化
```

(2) 列存储数据库

```
相关产品：Cassandra, HBase, Riak
典型应用：分布式的文件系统
数据模型：以列簇式存储，将同一列数据存在一起
优势：查找速度快，可扩展性强，更容易进行分布式扩展
劣势：功能相对局限
```

(3) 图形(Graph)数据库

```
相关数据库：Neo4J、InfoGrid、Infinite Graph
典型应用：社交网络
数据模型：图结构
优势：利用图结构相关算法。
劣势：需要对整个图做计算才能得出结果，不容易做分布式的集群方案。
```

3、什么是Redis：

(1) 是用C语言开发的一个开源的高性能键值对（key-value）数据库，它通过提供多种键值数据类型来适应不同场景下的存储需求。
(2)redis的应用场景

```
缓存（数据查询、短连接、新闻内容、商品内容等等）(使用最多)
分布式集群架构中的session分离。
聊天室的在线好友列表。
任务队列。（秒杀、抢购、12306等等）
应用排行榜。
网站访问统计。
数据过期处理（可以精确到毫秒）
```

4、Redis支持的键值数据类型：

(1) String:字符串类型
(2) hash：散列类型
(3) list：列表类型
(4) set：集合类型
(5) zset：有序集合类型

注意:Redis的key永远都是string，数据类型指的是value的类型。

二、Redis的使用(基于windows版)

前提需知:

(1) 我的redis安装目录是：D:\software\redis，启动的时候一定要在redis安装的路径下输入命令:
(2) 默认主机地址是127.0.0.1
(3) 默认端口是6379

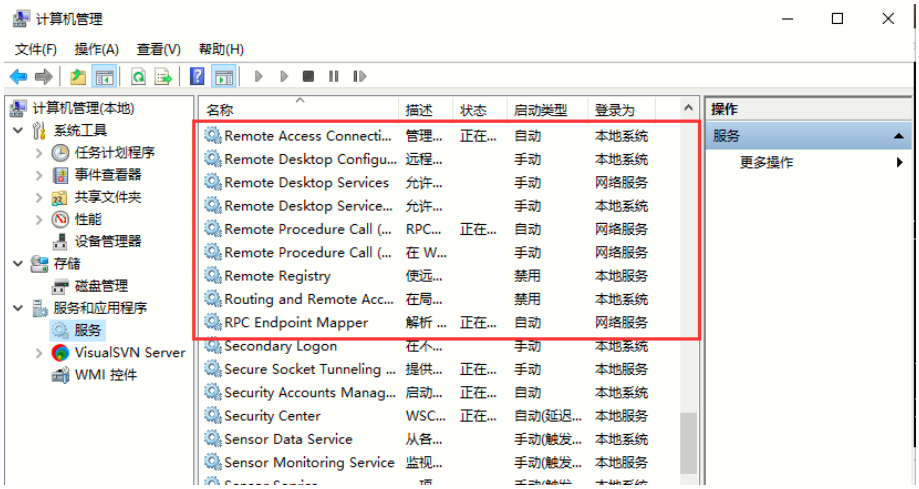
1、启动服务器端，在cmd窗口里面启动

```
Microsoft Windows [版本 10.0.18363.1139]
(c) 2019 Microsoft Corporation。保留所有权利。
C:\Users\sougu>d:                                       <!-- 进入D盘-->

D:\>cd software\redis                                   <!-- 进入redis的安装目录-->

D:\software\redis>redis-server redis.windows.conf       <!--输入启动服务器端的命令,并且加载redis.windows.conf 文件-->
                _.-_
           _.-``__ ''-._
      _.-``    `.  `_.  ''-._           Redis 3.0.504 (00000000/0) 64 bit
  .-`` .-```.  ```\/    _.,_ ''-._
```

注意：出现上面的图案就证明服务器端启动成功, Redis 3.0.504 (00000000/0) 64 :即redis的版本,是64位的;Port: 6379 即此redis服务器端的默认端口号为6379
补充:关闭服务器端快捷键:Ctrl + C

2、设置Redis服务

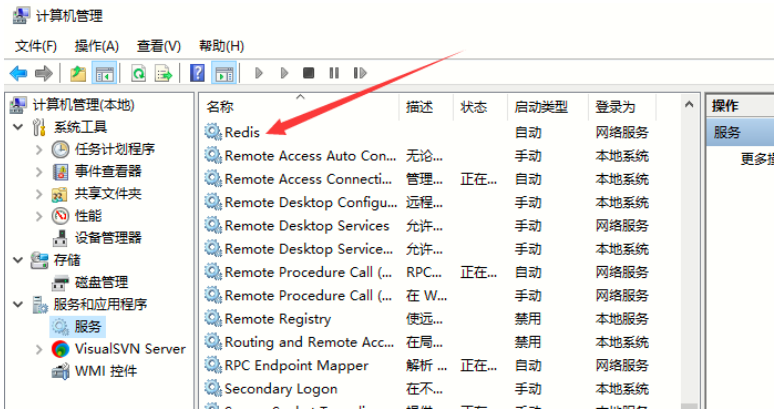
(1) 由于上面虽然启动了redis，但是只要一关闭cmd窗口，redis就会消失,因为它不属于windows下面的服务,打开计算机中的服务即下图，首先发现是没用这个Redis服务的,所以要把redis设置成windows下的服务.



(2) 设置服务命令

```
D:\software\redis>redis-server --service-install redis.windows-service.conf --loglevel verbose
```

注意:一定要做redis的安装目录下,回车后没有报错,即为设置成功,再打开计算机服务即下图,发现redis服务已经存在了



(3)、常用的redis服务命令。

卸载服务：redis-server --service-uninstall

开启服务：redis-server --service-start

停止服务：redis-server --service-stop

3、启动客户端，新打开一个cmd窗口

```
Microsoft Windows [版本 10.0.18363.1139]
(c) 2019 Microsoft Corporation。保留所有权利。
C:\Users\sougu>d:                                       <!--进入D盘-->

D:\>cd software\redis                                   <!-- 进入redis的安装目录-->

D:\software\redis>redis-cli  -h 127.0.0.1 -p 6379       <!--指定主机和端口登录客户端-->

127.0.0.1:6379>                                         <!--出现这句话,表示客户端启动成功-->
```

补充:关闭客户端快捷键:Ctrl + C

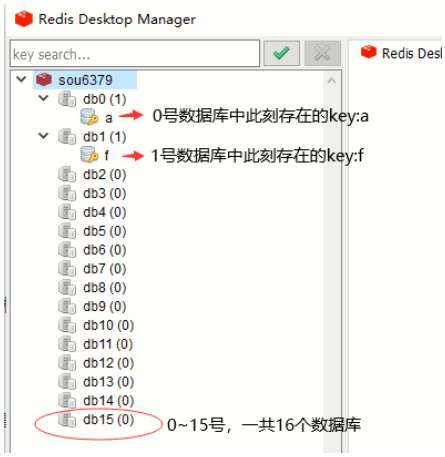
三、Redis通用命令

1、Redis数据库通用指令

(1) 默认一共是16个数据库，每个数据库之间是相互隔离。数据库的数量是在redis.conf中配置的。

```
databases 16
```

(2) 通过可视化工具:RedisDesktopManager 查看默认数据库个数



(3) 数据操作返回值

① 表示运行是否成功

(integer)0 —> false 失败
(integer)1 —> true 成功

② 表示运行结果值

(integer)3 —> 3 结果是数字3
(integer)1 —> 1 结果是数字1
未获取到数据 : (nil) 等同于null

③ 数据的存储与最大值

一个数据最大的存储量是512M
数值的最大范围，Long.MAX_VALUE

(4) 通用指令

```
select index          切换到index号(0~15)数据库

ping                  测试服务器是否连通

echo message          打印message到控制台

move key db           把key移动到指定数据库

flushall              清空当前服务的所有16个数据库

dbsize                查看当前数据库有多少key

flushdb               清空当前数据库

quit                  退出当前数据库
```

演示案例

```
127.0.0.1:6379> select 1            <!--切换到1号数据库-->
OK
127.0.0.1:6379[1]> select 0         <!--切换到0号数据库-->
OK
127.0.0.1:6379> ping                <!--测试服务器是否连通-->
PONG
127.0.0.1:6379> echo hello          <!--打印hello到控制台-->
"hello"
127.0.0.1:6379> get f               <!--能查询到之前set的key:f的值-->
"123"
127.0.0.1:6379> move f 1            <!--将f移到1号数据库-->
```

2、 keys通用指令

2.1- 基本操作

```
set key value     新建key ,值为value
get key           获取key的值
exists key        判断key是否存在
type key          获取key的类型
del key           删除key
```

演示案例

```
127.0.0.1:6379> set a 1             <!-- 新建key:a ,值为1 -->
OK
127.0.0.1:6379> get a               <!-- 获取key:a 的值 -->
"1"
127.0.0.1:6379> exists a            <!--判断key:a是否存在-->
(integer) 1
127.0.0.1:6379> type a              <!--获取key:a 的数据类型-->
string
127.0.0.1:6379> del a               <!--删除key:a-->
(integer) 1
```

2.2- 设置key的生存时间

(1) Redis在实际使用过程中更多的用作缓存，然而缓存的数据一般都是需要设置生存时间的，即：到期后数据销毁。

```
expire key seconds          设置key的生存时间（单位：秒）key在多少秒后会自动删除
ttl key                     查看key剩余的生存时间， 返回-2代表不存在,返回-1代表永久存在。
persist key                 清除生存时间即设置为永久存在
pexpire key milliseconds    生存时间设置单位为：毫秒
```

演示案例

```
127.0.0.1:6379> set a abc                   <!--设置key: a,值为 abc-->
OK
127.0.0.1:6379> expire a 50                 <!--设置key: a 的生存时间为50秒-->
(integer) 1
127.0.0.1:6379> ttl a                       <!--查询key: a 的剩余生存时间为46秒-->
(integer) 46
127.0.0.1:6379> persist a                   <!--清除key: a 的生存时间即设置为永久存在-->
(integer) 1
127.0.0.1:6379> pexpire a 3600              <!--设置key: a 的生存时间为3600毫秒-->
(integer) 1
```

2.3- 查询key:keys pattern 根据条件查询数据库中的所有的key

(1) * 匹配任意数量字符

(2) ? 匹配一个字符

(3) [ ] 匹配一个指定符号，比如[st]，意思是包含s或者t

演示案例

```
127.0.0.1:6379> set a a             <!--设置key: a ,值为a-->
OK
127.0.0.1:6379> set ab ab           <!--设置key: ab ,值为ab-->
OK
127.0.0.1:6379> set abs abs         <!--设置key: abs ,值为abs-->
OK
127.0.0.1:6379> keys a*             <!--查询所有以a开头的key-->
1) "a"
2) "ab"
3) "abs"
127.0.0.1:6379> keys a?             <!--查询所有以a开头的,后面只含一个字符的key-->
1) "ab"
127.0.0.1:6379> keys a[sb]          <!--匹配包含b或者s指定符的key,即as或者ab,此数据库中只有ab,没有as所以返回ab-->
1) "ab"
```

2.4- 重命名key和对key排序,只能是list，set，sorted set、hash元素（元素可以为数值与字符串）的排序。

```
rename key newkey           改名，重名会覆盖。
renamenx key newkey         如果不存在才改名，重名不会更改。
sort key                    对key进行排序，必须是set、list或者sorted_set。
```

注意:默认升序，原数据顺序不变,后面加上desc，逆序排序。

演示案例:

```
127.0.0.1:6379> set a 123                                       <!--新建key: a,值为123-->
OK
127.0.0.1:6379> rename a a                                      <!--将key: a 重命名为 a 会有错误提醒-->
(error) ERR source and destination objects are the same
127.0.0.1:6379> rename a b                                      <!--将key: a 重命名为 b-->
OK
127.0.0.1:6379> get a                                           <!--重命名后查询a为空-->
(nil)
127.0.0.1:6379> get b                                           <!--重命名后查询 b 的值为原来 a 的值-->
"123"
127.0.0.1:6379> renamenx b b                                    <!--将key: b 重命名为 b 会有错误提醒-->
```

四、五种数据类型的通用指令

一、String类型

1、命令

(1) 基本命令：

```
set key value               将字符串值 value 关联到 key ，如果 key 已经持有其他值， SET 就覆写旧值，无视类型。
get key                     取key的值value
del key                     删除key:删除操作成功 返回(integer)1;删除操作失败 返回(integer)0
```

演示案例

```
127.0.0.1:6379> set a 123       <!--新建key: a 并且赋值为123-->
OK
127.0.0.1:6379> get a           <!--获取key: a 的值-->
"123"
127.0.0.1:6379> set a 1234      <!--修改key: a 赋值,会覆盖之前的值123,此时 a 的新值为1234-->
OK
127.0.0.1:6379> get a           <!--获取修改后的key: a 的值-->
"1234"
127.0.0.1:6379> del a           <!--删除key: a-->
(integer) 1
```

(2) 高级命令

```
mset k1 v1 k2 v2 k3 v3 ...      一次性添加或修改多个键值对
mget k1 k2 k3...                一次性获取k1 k2 k3...的value
strlen k                        获取k对应的v的字符串长度
append k v                      往k对应的v尾部追加数据，如果不存在就新建,这时候相当于set k v
```

演示案例

```
127.0.0.1:6379> mset a 1 b 2 c 3    <!-- 一次性添加a、b、c三个key,键值分别为1、2、3-->
OK
127.0.0.1:6379> mget a b c          <!-- 一次性获取a、b、c三个key的值-->
1) "1"
2) "2"
3) "3"
127.0.0.1:6379> strlen a            <!--获取a对应的v的字符串长度-->
(integer) 1
127.0.0.1:6379> append a 23         <!--往a对应的v的字符串后面追加23字符串-->
(integer) 3
127.0.0.1:6379> get a               <!--查询追加23字符串后的a的值-->
"123"
```

(3) 其它命令

① 自增自减操作控制数据库主键

```
incr key                    对应的value加1

decr key                    对应的value减1

incrby key increment        对应的value+increment

decrby key increment        对应的value-increment

incrbyfloat key increment   对应的value+一个浮点数
```

演示案例

```
127.0.0.1:6379> set a 6                             <!--新建k: a 的值为 6-->
OK
127.0.0.1:6379> incr a                              <!--自增1-->
(integer) 7
127.0.0.1:6379> incr a                              <!--自增1-->
(integer) 8
127.0.0.1:6379> decr a                              <!--自减1-->
(integer) 7
127.0.0.1:6379> decr a                              <!--自减1-->
(integer) 6
127.0.0.1:6379> incrby a 2                          <!--对应的value + 2-->
```

注意：按数值进行的操作，如果原始数据不能转换为数字，或者超过了redis的数值上限，操作会报错，其中数值上限是java中long的最大值。

```
<!--对应的b的值为a,加减运算不能自动转为数字，会报错-->

127.0.0.1:6379> set b a
OK
```

```
127.0.0.1:6379> incr b
(error) ERR value is not an integer or out of range
127.0.0.1:6379>
```

② 设置数据的生命周期

```
setex key seconds value              将值 value 关联到 key ，并将 key 的生存时间设为 seconds （以秒为单位）。如果 key 已经存在， SETEX 命令将覆写旧值。

psetex key milliseconds value        将值 value 关联到 key ，并将 key 的生存时间设为 seconds （以毫秒为单位）。如果 key 已经存在， SETEX 命令将覆写旧值。
```

演示案例

```
127.0.0.1:6379> setex a 60 123    <!--新建key: a ,生存时间为60秒，value为123-->
OK
127.0.0.1:6379> psetex b 60 1234    <!--新建key: a ,生存时间为60毫秒，value为1234-->
OK
```

注意： 如果setex一个k后，再set这个k，那么定时的k会被清除，变成不定时的。

2、应用场景

(1) 在Redis中为CSDN用户设定用户信息，以用户主键和属性值作为key，后台设置定时刷新策略
user:id qq_41918166:fans -> 123456
user:id qq_41918166:blogs -> 2333
(2) 存json数据
set user: id:12345 {id:12345,fans:123456,blogs:2333,focus:666}
(3) 数据库的热点数据key命名规范:

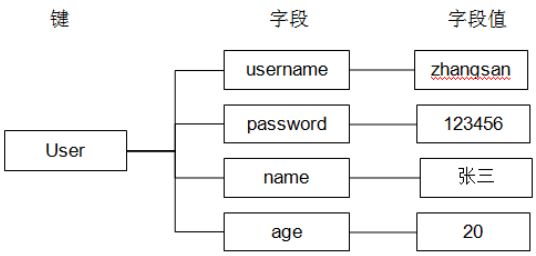
```
表名:主键名:主键值:字段名
```

二、Hash 类型

提出原由：假设有User对象以JSON序列化的形式存储到Redis中，User对象有id，username、password、age、name等属性，存储的过程如下： 保存、更新 。 User “{“username”:”gyf”,”age”:”80”}”
思考：如果在业务上只是更新age属性，其他的属性并不做更新我应该怎么做呢？ 如果仍然采用上边的方法在传输、处理时会造成资源浪费，接下来的hash可以很好的解决这个问题。

1、hash介绍

(1) 基本概念：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。



(2) 格式：一个存储空间(key)存储多个键值对，底层通过哈希表进行存储。

```
key -- {filed1 - v1, filed2 - v2,...}
```

注意：如果filed数量较多时，会被优化为类数组的结构，如果filed数量多，就是HashMap。

2、命令

(1) 基本命令：

```
hset key field value    新增/修改某个field的v，新增时返回1，修改时返回0
hsetnx key field value  如果key中没有field字段则设置field值为value，否则不做任何操作
hget key field          获取某个field的v

hgetall key             获取这个key的所有f-v

hdel key field          删除某个field，可以删除一个或多个，返回值是被删除的字段个数

del key                 删除整个key
```

演示案例

```
127.0.0.1:6379> hset user username zhangsan <!--新增 user username属性，value为 zhangsan-->
(integer) 1
127.0.0.1:6379> hset user age 20            <!--新增 user age属性，value为 20-->
(integer) 1
127.0.0.1:6379> hset user sex man           <!--新增 user sex属性，value为 man-->
(integer) 1
```

```
127.0.0.1:6379> hsetnx user age 30          <!--user中有age字段则不做任何修改，age 的值还是20-->
(integer) 0
127.0.0.1:6379> hget a age                  <!--验证上一步hsetnx没有做任何修改，age 的值还是20-->
"20"
```

(2) 一次性多个数据的操作

```
hmset key f1 v1 f2 v2 f3 v3 ...     新增/修改某个field的f1、f2、f3，值分别为v1、v2、v3

hmget key f1 f2 f3...               获取某个field的f1、f2、f3 的值

hlen key                            获取key的字段数量，就是field的数量

hexists key field                   判断key中是否存在field这个字段
```

演示案例

```
127.0.0.1:6379> hmset user name lisi age 20     <!--新增 user name属性value为 lisi,age 为 20-->
OK
127.0.0.1:6379> hmget user name age             <!--获取 user name和age属性的value-->
1) "lisi"
2) "20"
127.0.0.1:6379> hlen user                       <!--获取user的字段数量-->
(integer) 2
127.0.0.1:6379> hexists user name               <!--判断user中是否存在name字段-->
(integer) 1
127.0.0.1:6379> hexists user sex                <!--判断user中是否存在sex字段-->
(integer) 0
```

(3) 其它命令

```
hkeys key                           获取key对应的所有的field

hvals key                           获取key对应的所有的value

hincrby key field increment         key中field对应的数值+increment

hincrbyfloat key field increment    key中field对应的数值+浮点型increment
```

演示案例

```
127.0.0.1:6379> hkeys user                      <!--获取user对应的所有的field-->
1) "name"
2) "age"
127.0.0.1:6379> hvals user                      <!--获取user对应的所有的value-->
1) "lisi"
2) "20"
127.0.0.1:6379> hincrby user age 2              <!--user中age对应的数值+ 2-->
(integer) 22
127.0.0.1:6379> hincrbyfloat user age 1.5       <!--user中age对应的数值+ 1.5-->
"23.5"
```

3、注意事项

(1) hash类型的value只能存储string，不允许嵌套存储。如果获取不到对应的数据，返回的是(nil)。

(2) 每个hash最多存储2^32 -1 个键值对。

(3) hash最初设计不是为了存对象，不要把hash当成对象列表使用。

(4) hgetall 可以获取全部属性，如果field过多，遍历一次会很慢，影响程序效率。

4、应用场景

(1) 电商购物车：添加购物车、浏览购物车商品、更改购物车商品数量、删除商品、清空商品均可实现。

key : userID

field : 商品ID

value : 商品购买数量

演示案例

```
127.0.0.1:6379> hmset userid:1001 id 10011 name phone number 10
OK
127.0.0.1:6379> hmset userid:1002 id 10012 name xiaomi number 15
OK
127.0.0.1:6379> hgetall userid:1001
1) "id"
2) "10011"
3) "name"
```

```
4) "phone"
5) "number"
6) "10"
```

优化：

把商品详情提取出来做一个独立的hash。

购物车只存id和数量。

(2) 商家限量抢购

每卖一个就 hincrby key field -1。

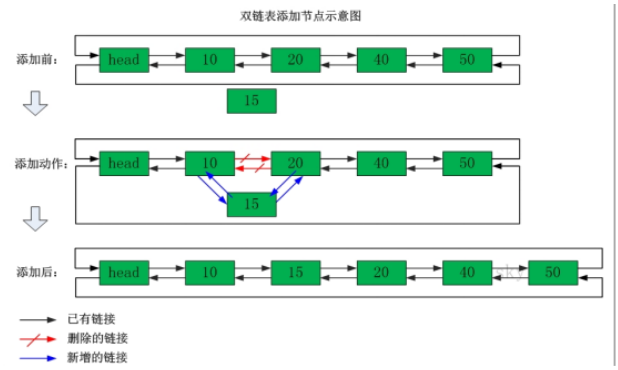
```
127.0.0.1:6379> hincrby user number -1      <!--user中number对应的数值 -1 -->
```

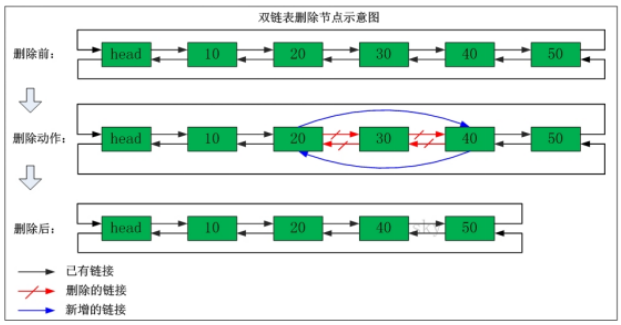
三、List 类型

1、ArrayList与LinkedList的区别

(1) ArrayList使用数组方式存储数据，所以根据索引查询数据速度快，而新增或者删除元素时需要设计到位移操作，所以比较慢

(2) LinkedList使用双向链表方式存储数据，每个元素都记录前后元素的指针，所以插入、删除数据时只是更改前后元素的指针指向即可，速度非常快。然后通过下标查询元素时需要从头开始索引，所以比较慢，但是如果查询前几个元素或后几个元素速度比较快





2、Redis list介绍

(1) list类型：可以存储一个有序的字符串列表，常用的操作是向列表两端添加元素(可以看出双端队列)，或者获得列表的某一个片段

(2) list类型：内部是使用双向链表（double linked list）实现的，所以向列表两端添加元素的时间复杂度为0(1)，获取越接近两端的元素速度就越快。这意味着即使是一个有几千万个元素的列表，获取头部或尾部的10条记录也是极快的

注意： list中的元素有序可重复

3、命令

(1) 基本命令：

```
lpush key value1 value2 ...              左侧插入
rpush key value1 value2 ...              右侧插入
lrange key start stop                    从start开始到stop结束的下标的数据，引从0开始，如果是负数结束，比如stop=-1，那就是截止到倒数第一个
lindex key index                         找到index位置的数据
lpop key                                 移除并返回第一个元素
rpop key                                 移除并返回最后一个元素
llen                                     获取列表中元素个数
lrem key count value                     删除list列表中number个value(因为list元素可以重复,所以要指定count)
                                             1）当count>0时， lrem会从列表左边开始删除
                                             2）当count<0时， lrem会从列表后边开始删除
                                             3）当count=0时， lrem删除所有值为value的元素
ltrim key start stop                     只保留列表中start开始到stop结束之间指定片段的数据
linsert key before|after pivot value     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
 rpoplpush source destination            将元素(左侧第一个元素)从一个列表转移到另一个列表中(左侧插入)，并且返回被移入的元素，
                                         新列表destination不存在的话，会自动新建列表destination
```

演示案例

```
127.0.0.1:6379> lpush list:1 1 2 3          <!--从左侧插入1、2、3-->
(integer) 3
127.0.0.1:6379> rpush list:1 4 5 6          <!--从右侧插入4、5、6-->
(integer) 6
127.0.0.1:6379> lrange list:1 0 2           <!--从右侧取出下标为0~2之间的元素-->
1) "3"
2) "2"
3) "1"
127.0.0.1:6379> lrange list:1 0 -1          <!--取出所有元素-->
1) "3"
2) "2"
3) "1"
4) "4"
5) "5"
6) "6"
127.0.0.1:6379> lindex list:1 3             <!--获取下标为3的元素-->
"4"
127.0.0.1:6379> lpop list:1                 <!--移除并返回第一个元素3-->
"3"
127.0.0.1:6379> rpop list:1                 <!--移除并返回最后一个元素6-->
"6"
127.0.0.1:6379> llen list:1                 <!--获取list中的元素个数-->
(integer) 4
127.0.0.1:6379> lrange list:1 0 -1          <!--获取list中的元素，验证上步的执行-->
1) "2"
2) "1"
3) "4"
4) "5"
127.0.0.1:6379> lrem list:1 1 2             <!--从左侧删除1个2-->
(integer) 1
127.0.0.1:6379> lrem list:1 -1 5            <!--从右侧删除1个5-->
(integer) 1
127.0.0.1:6379> lrange list:1 0 -1          <!--获取list中的元素，验证上步的执行-->
1) "1"
2) "4"
127.0.0.1:6379> lrem list:1 0 1             <!--删除list中所有的1-->
(integer) 1
127.0.0.1:6379> lrange list:1 0 -1          <!--获取list中的元素，验证上步的执行-->
1) "4"
127.0.0.1:6379> linsert list:1 after 4 1    <!--在4后面添加元素1-->
(integer) 2
127.0.0.1:6379> lrange list:1 0 -1          <!--获取list中的元素，验证上步的执行-->
1) "4"
2) "1"
127.0.0.1:6379>  rpoplpush list:1 newlist   <!--将list:1列表左侧第一个元素 1 移入列表 newlist中，并且返回被移入的元素 1-->
"1"
127.0.0.1:6379> lrange newlist 0 -1         <!--获取newlist中的元素，验证上步的执行-->
1) "1"
127.0.0.1:6379> lrange list:1 0 -1          <!--获取list中的元素，验证上上一步的执行-->
1) "4"
127.0.0.1:6379>  rpoplpush list:1 newlist   <!--将list:1列表左侧第一个元素 4 移入列表 newlist中，并且返回被移入的元素 4 -->
"4"
127.0.0.1:6379> lrange newlist 0 -1         <!--获取newlist中的元素，验证上一步的执行,是从newlist列表左侧插入的-->
1) "4"
2) "1"

```

(2) 扩展操作：

① 规定时间内获取并移除元素。阻塞式数据获取。

② 如果本来没元素了，但是timeout时间内等到了元素(有其他人插入)，就返回

```
blpop key timeout
brpop key timeout
```

4、应用场景:可以应用于任务队列

(1) 朋友圈点赞,按顺序显示点赞的朋友

(2) 商品评论列表

在Redis中创建商品评论列表

用户发布商品评论，将评论信息转成json存储到list中

用户在页面查询评论列表，从redis中取出json数据展示到页面

演示案例

定义商品评论列表key：商品编号为1001的商品评论key【items: comment:1001】

```
127.0.0.1:6379> lpush items:comment:1001 '{"id":1,"name":"it is wery good","date":2020/11/02}'
(integer) 1
127.0.0.1:6379> lrange items:comment:1001 0 -1
1) "{\"id\":1,\"name\":\"it is wery good\",\"date\":2020/11/02}"
```

5、注意事项

(1) list保存的数据都是string
(2) 一个list存的数量最多是2^32 - 1个

(3) 虽然有索引的概念，但是最好按照队列和栈的模式来操作

(4) Redis可以对数据进行分页：
① 一般第一页数据，比如淘宝商品列表第一页，放在Redis，加速访问
② 第二页开始通过数据库形式访问

四、set 类型

1、Redis set介绍

(1) 存储大量数据、查询速度快
(2) 集合中的数据是不重复且没有顺序
(3) 集合类型的Redis内部是使用值为空的散列表实现，所有这些操作的时间复杂度都为0(1)
(4) 集合类型的常用操作是向集合中加入或删除元素、判断某个元素是否存在等，除此之外Redis还提供了多个集合之间的交集、并集、差集的运算。

2、命令

(1) 基本命令：

```
sadd key value1 value2 ...      添加数据

smembers key                    获取全部数据

scard key                       获取数据总量

sismember key value             判断value是否是key集合内的数据

srem key value                  删除数据
```

演示案例

```
127.0.0.1:6379> sadd set1 1 2 3         <!--新建一个集合set1,集合中放入数据1、2、3-->
(integer) 3
127.0.0.1:6379> smembers set1           <!--获取集合set1中的所有数据-->
1) "1"
2) "2"
3) "3"
127.0.0.1:6379> scard set1              <!--获取集合set1中的数据数量-->
(integer) 3
127.0.0.1:6379> sismember set1 2        <!--判断数据 2 是否是集合set1中的数据-->
(integer) 1
127.0.0.1:6379> sismember set1 4        <!--判断数据 4 是否是集合set1中的数据-->
(integer) 0
127.0.0.1:6379> srem set1 3             <!--删除集合set1中的数据 3 -->
(integer) 1
127.0.0.1:6379> smembers set1           <!--获取集合set1中的所有数据验证上步操作-->
1) "1"
2) "2"

```

(2) set 运算命令

```
sinter set1 set2 ...                求多个集合的交集：属于set1且属于set2的元素构成的集合

sunion set1 set2 ...                求多个集合的并集：属于set1或者属于set2的元素构成的集合

sdiff set1 set2 ...                 求多个集合的差集：属于set1并且不属于set2 的元素构成的集合

sinterstore des set1 set2 ...       求多个集合的交集,并且把交集存入集合des中

sunionstore des set1 set2 ...       求多个集合的并集,并且把交集存入集合des中

sdiffstore des set1 set2 ...        求多个集合的差集,并且把交集存入集合des中

smove src des value                 将指定数据value从源集合src移动到目标集合des
```

演示案例

```
127.0.0.1:6379> sadd setA 1 2 3                 <!--新建set集合 setA,存入数据 1、2、3-->
(integer) 3
127.0.0.1:6379> sadd setB 2 3 4                 <!--新建set集合 setB,存入数据 2、3、4-->
(integer) 3
127.0.0.1:6379> sinter setA setB                <!--求集合setA与setB的交集-->
1) "2"
2) "3"
127.0.0.1:6379> sunion setA setB                <!--求集合setA与setB的并集-->
1) "1"
2) "2"
3) "3"
4) "4"
```

```
127.0.0.1:6379> sdiff setA setB                <!--求集合setA与setB的差集-->
1) "1"
127.0.0.1:6379> sdiff setB setA                <!--求集合setB与setA的差集-->
1) "4"
127.0.0.1:6379> sinterstore setC setA setB     <!--求集合setA与setB的交集,并且把交集存入集合setC中-->
(integer) 2
127.0.0.1:6379> smembers setC                  <!--获取集合setC中的数据，验证上一步操作-->
1) "2"
2) "3"
127.0.0.1:6379> sunionstore setD setA setB     <!--求集合setA与setB的并集,并且把并集存入集合setD中-->
(integer) 4
127.0.0.1:6379> smembers setD                  <!--获取集合setD中的数据，验证上一步操作-->
1) "1"
2) "2"
3) "3"
4) "4"
127.0.0.1:6379> sdiffstore setE setA setB      <!--求集合setA与setB的差集,并且把差集存入集合setE中-->
(integer) 1
127.0.0.1:6379> smembers setE                  <!--获取集合setE中的数据，验证上一步操作-->
1) "1"
127.0.0.1:6379> smove setE setF 1              <!--将指定数据 1 从集合setE 中 移动到 集合setF 中-->
(integer) 1
127.0.0.1:6379> smembers setF                  <!--获取集合setF中的数据，验证上一步操作-->
1) "1"
```

3、注意事项

(1) set不允许重复。如果重复添加，会添加失败
(2) set虽然结构上是hash，但是存储值的空间无法启用
① hash: key-{field:value}
② set: key-{value:nil}
(3) Redis最好只提供基础数据，别提供校验结果
(4) set的特征进行同类型数据快速去重

五、SortedSet类型zset(有序集合)

1、Redis zset介绍

(1) 在集合类型的基础上，有序集合类型为集合中的每个元素都关联一个分数(即排序字段叫score)，这使得我们不仅可以完成插入、删除和判断元素是否存在在集合中，还能够获得分数最高或最低的前N个元素、获取指定分数范围内的元素等与分数有关的操作

2、list与zset的区别

(1) 相同点
① 二者都是有序的
② 二者都可以获得某一范围的元素
(2) 二者区别
① 列表类型(list)是通过链表实现的，获取靠近两端的数据速度极快，而当元素增多后，访问中间数据的速度会变慢,list允许重复，zset不允许重复。
② 有序集合类型(zset)使用散列表实现，所有即使读取位于中间部分的数据也很快
③ 列表(list)中不能简单的调整某个元素的位置，但是有序集合可以（通过更改分数实现）
④ 有序集合类型(zset)要比列表类型更耗内存

3、命令

(1) 基本命令：

```
zadd key score1 value1 score2 value2..--添加数据，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

zrange key start stop [WITHSCORES]-------获取数据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），
                                       如果WITHSCORES在末尾，则会把score也输出出来
zrevrange key srart stop [WITHSCORES]----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），
                                       如果WITHSCORES在末尾，则会把score也输出出来。
zcard key                   -------------获取数据总量

zcount key min max          -------------获取[min, max]范围内的数据数量

zrem key value ------------------------- 删除数据
```

演示案例

```
127.0.0.1:6379> zadd scores 80 lisi 90 zhangsan 85 xiaowang  <!--添加数据-->
(integer) 3
127.0.0.1:6379> zrange scores 0 -1                          <!--获取数据-升序-->
1) "lisi"
2) "xiaowang"
3) "zhangsan"
127.0.0.1:6379> zrange scores 0 -1 withscores                <!--获取数据-升序-显示分数-->
1) "lisi"
2) "80"
3) "xiaowang"
4) "85"
```

(2) 按条件操作数据命令

```
zrangebyscore key min max [withscores] [limit]  升序范围查询(前提 key中数据必须已经是升序的,不然会提示:empty list or set)
                                                limit中的参数: offset、count
                                                    offset表示开始位置
                                                    count表示数据总量
zrevrangebyscore key min max [withscores]       降序范围查询(前提 key中数据必须已经是降序的, 不然会提示:empty list or set)

zremrangebyrank key start stop                  按索引删除

zremrangebyscore key min max                    按范围删除
```

演示案例

```
127.0.0.1:6379>  zadd scores 1 a 2 b 3 c 4 d
(integer) 4
127.0.0.1:6379> zrangebyscore scores 1 4               <!--min=1,  max=4指定范围升序查询-->
1) "a"
2) "b"
3) "c"
4) "d"
127.0.0.1:6379> zrangebyscore scores 1 4 limit 1 2     <!-- min=1,  max=4,  所以结果是a b c d. 又因为有limit, 偏移量是1,
                                                           所以从b开始, count是2, 输出两个, 所以最终结果是: -->
1) "b"
2) "c"
```

(3) 交并操作

```
******计算多个有序集合的交集, 并把结果有序集合存在到一个新的key****
ZINTERSTORE destination numkeys key [key …] [WEIGHTS weight [weight …]] [AGGREGATE SUM|MIN|MAX]

<*******详解*******>
ZINTERSTORE ------------------命令关键字
destination ------------------指定结果集保存的集合key
numkeys key [key …] -------- numkeys 指定有多少个集合参与交集运算,  key 指定参与交集运算的集合的key
[WEIGHTS weight [weight …]] -指定参与交集运算各集合score的权重参数
[AGGREGATE SUM|MIN|MAX] -----指定交集中元素的score的取值方式, 例如: sum 等于各集合中该元素的score乘以权重求和

********注意:
```

(3) 获取排名(索引)

```
zrank key value                 获取value在key中的升序排名, score小的排在前面

zrevrank key value              降序排名

zscore key value                拿到value的score

zincrby key increment value     给value加上对应的increment
```

演示案例

```
127.0.0.1:6379> zadd scores 30 a 50 b 10 c 35 d
(integer) 4
127.0.0.1:6379> zrank scores c                    <!--获取c 在key中的升序排名, score小的排在前面-->
(integer) 0
127.0.0.1:6379> zrevrank scores c                 <!--获取c 在key中的降序排名, score大的排在前面-->
(integer) 3
127.0.0.1:6379> zscore scores b                   <!--获取b 在key中的值 50-->
"50"
127.0.0.1:6379> zincrby scores 5 c                <!--给c 加上 5-->
"15"
```

4、注意事项

(1) score是64位整数
(2) 如果score是小数, 那就是双精度浮点。基于双精度浮点的特征, 可能会丢失精度
(3) 如果添加重复的数据, score会被最后一次的覆盖

5、应用场景:商品销售排行榜

(1) 需求：根据商品销售量对商品进行排行显示

(2) 思路：定义商品销售排行榜（sorted set集合），Key为items:sellsort，分数为商品销售量

演示案例

```
127.0.0.1:6379> zadd items:sellsort 9 1001 10 1002          <!--商品编号1001的销量是9，商品编号1002的销量是10-->
(integer) 2
127.0.0.1:6379> zincrby items:sellsort 1 1001               <!--商品编号1001的销量加1-->
"10"
127.0.0.1:6379> zrange items:sellsort 0 9 withscores        <!--获取商品销量前10名-->
1) "1001"
2) "10"
3) "1002"
4) "10"
```

总结

命令比较多，我特意列出了比较详细的案例以及注释，希望可以帮助需要的朋友。有不懂的地方可以在此博客下面评论，看到后会及时回复。

Redis【第二篇】Redis持久化设置，主从复制，Redis集群与分片式集群