03 Redis的数据结构（五大基本类型+四大扩展类型）

数据结构
String（字符串），类似ArrayLi st
Li st（列表），有序可重复，类似LinkedLi st，插入和删除快，复杂度O(1)，索引慢，复杂度O(n)
Hash（字典），类似HashMap
Set（集合），无序无重复，类似HashSet
ZSet（有序集合），有序无重负，类似SortedSet和HashMap的结合，可以扩展使用**GEO**。
Bitmap（位图）、**HyperLogLog**（基数统计）、**Stream**（流）

```
Redis内部使用一个redisObject对象来表示所有的key和value，
redisObject中，type代表一个value对象具体是何种数据类型，encoding代表是不同数据类型在redis内部的存储方式。
支持命令查看：object encoding key 其中list、hash、set、zset为容器型结构，共享两个规则：不存在就创建，没有元素就删除。
```

数据类型之字符串

string类型是Redis最基本的数据类型，一个键最大能存储512MB。string类型是二进制安全的。意思是 redis的string可以包含任何数据。

命令

```
1. strlen keyName : 查看keyName对应的value的长度
2. append keyName appendStr : 拼接字符串在原有的keyName所对应的value后，返回拼接结束的value的长度
3. 当value的值是整数时，可以通过命令直接进行数学运算，
      比如：incr keyName，自增，返回最新的值（Integer）num
      比如：decr keyName，自减，返回最新的值（Integer）num
      比如：incrby keyName increment，增加，区别是增加了一个计算的数
      比如：decrby keyName decrement，同上。
4. getrange keyName start end : 截取keyName的value的[start end]，返回截取后的值，从0开始计数
   setrange keyName index value : 将index位置的值设置为value，返回修改后的值的长度，且改变的位数和给定的value值位数相同
        比如：
        127.0.0.1:6379> set new1 "hello111222"
        OK
        127.0.0.1:6379> get new1
        "hello111222"
        127.0.0.1:6379> setrange new1 5 333
        (integer) 11
        127.0.0.1:6379> get new1
        "hello333222"
5. 整合命令
    setex = set + expire，格式：setex keyName time value，设置keyName的有效期为time，且设置value
```

```
    setnx = set + exists = set if not exists，格式：setex keyName value，keyName不存在设置keyName = value
        如果key不存在，则设置数据返回1；如果存在，不设置，返回0。
    getset keyName value : 拿到原先keyName的值，设置为给定的value值。
6. 批量操作
    mget key1 key2 key3... : 批量操作，返回多个value
    mset key1 value1 key2 value2 key3 value3... : 批量设置多个key=value，参数本身就是map，成功返回OK。
```

原理

```
redis的字符串是动态字符串，内部结构类似ArrayList。
采用预分配冗余空间的方式减少内存的频繁分配。
内部为字符串分配的实际空间一般高于字符串长度，
  * 当字符串长度<1MB时，扩容方式是直接加倍，
  * 如果 >1MB，一次扩容只扩1MB，直到扩大到512MB。
```

数据类型之列表

redis的列表是一个字符链表，内部结构类似LinkedList。
left，right都可以插入添加。
如果键不存在，创建新的链表。如果键已经存在，新增内容。
如果值全移除，对应的键也就消失了。
列表最多可存储 2^32 - 1 元素 (4294967295, 每个列表可存储40多亿)。

命令

```
有先后之分，对应数据结构就是左右之分，使用时必须明确从左开始存，还是从右开始存，不同的方向，对应的数据结构就不一样了）
* 理解什么是栈 —— 先进后出（像生活中的口袋）
    -* push压栈 pop弹出
    比如说（从左边开始存）：
      127.0.0.1:6379> lpush list1 1 2 3 4 5
      (integer) 5
      127.0.0.1:6379> lrange list1 0 -1
      1) "5"
      2) "4"
      3) "3"
      4) "2"
      5) "1"
      127.0.0.1:6379>
      这样就是栈的结构了，存 12345 取 54321

* 理解什么是队列——先进先出 （像生活中的排队）
  -* 比如说（从右侧将其存进去）
      127.0.0.1:6379> rpush list2 1 2 3 4 5
      (integer) 5
      127.0.0.1:6379> lrange list2 0 -1
```

```
      1) "1"
      2) "2"
      3) "3"
      4) "4"
      5) "5"
      127.0.0.1:6379>
      这样就是队列的结构了，存 12345 取 12345

上述，对应的就是存取操作，其实存取操作就是这三个关键词：push、pop、range
  lpush:从左存入数据 —— 栈结构，弹出时执行lpop，弹出最左侧的元素
  rpush:从右存入数据 —— 队列结构，弹出时执行lpop，弹出最左侧的元素
  lrange:范围内查看数据，使用方式：lrange start end,读到[start, end]的值
  ---

  到这里我们开始真正的取值，对于栈而言，每次都拿栈顶的元素。使用 lpop listName，弹出后就不存在这个数了。
  当然也是可以从右侧弹出的，不过这个时候，这个数据结构就变化了。
  无论用那种push方式，看我们弹出的方式，弹出方式不同，随之而来也就改变了数据结构。
```

```
1. llen listName : 返回List的元素个数
2. lindex listName index : 获取index位置的值，从0开始计数，返回改值
3. lrem listName count value : 删除列表中count个值为value的元素，返回删除的个数
4. ltrim listName start end : 截取[start, end]，成功返回OK，此时listName的值为截取后的值
5. linsert listName before|after oldValue newValue，插入newValue在oldValue，的之前或者之后，返回元素的个数
      比如：linsert list3 before 3 2 : 插入list3，在3之前插入 2。
      如果要插入数据到头部，还可以使用lpushx，使用方式：lpushx listName value
      如果要插入数据到尾部，还可以使用rpushx，使用方式：rpushx listName value
6. lset listName index value : 将index位置的值变为value
```

原理

```
底层是一个“快速链表”(quicklist)的结构，
在列表元素较少时，使用连续的内存存储成压缩列表ziplist。
当数据量较多时，改成quicklist，也就是将多个ziplist使用双向指针串起来使用，以减少内存的碎片化。
```