# Day37\_Redis

## 学习目标

1. 能够理解nosql的概念

2. 能够说出redis的常用数据类型

3. 能够使用redis的string操作命令

4. 能够使用redis的hash操作命令

5. 能够使用redis的list操作命令

6. 能够使用redis的set操作命令

7. 能够说出redis的两种持久化机制

8. 能够使用jedis对redis进行操作

## 一、NoSQL概述

### 1. 什么是NoSQL

NoSQL(NoSQL = Not Only SQL)，意即“不仅仅是SQL”，是一项全新的数据库理念，泛指非关系型的数据库。

(关系型数据库<MySQL、Oracle、DB2、SQLServer>：有表，而且表与表直接存在关系<主外键>)

NoSQL与关系型数据库SQL是互相弥补的……

关系型数据库优点：对数据划分得比较清晰(表结构和多表)

关系型数据库缺点：进行查询，涉及多表操作！！！(查询效率低！！！)

非关系型数据缺点：关系描述没有！！！！！

非关系型数据优点：查询效率很高！！！！

### 2. 为什么要使用NoSQL

随着互联网web2.0网站的兴起，非关系型的数据库现在成了一个极其热门的新领域，非关系数据库产品的发展非常迅速。而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，例如：

* 1、High performance - 对数据库高并发读写的需求

web2.0网站要根据用户个性化信息来实时生成动态页面和提供动态信息，所以基本上无法使用动态页面静态化技术，因此数据库并发负载非常高，往往要达到每秒上万次读写请求。关系数据库应付上万次SQL查询还勉强顶得住，但是应付上万次SQL写数据请求，硬盘IO就已经无法承受了。其实对于普通的BBS网站，往往也存在对高并发写请求的需求，例如网站的实时统计在线用户状态，记录热门帖子的点击次数，投票计数等，因此这是一个相当普遍的需求。

* 2、Huge Storage - 对海量数据的高效率存储和访问的需求

类似Facebook，twitter，Friendfeed这样的SNS网站，每天用户产生海量的用户动态，以Friendfeed为例，一个月就达到了2.5亿条用户动态，对于关系数据库来说，在一张2.5亿条记录的表里面进行SQL查询，效率是极其低下乃至不可忍受的。再例如大型web网站的用户登录系统，例如腾讯，盛大，动辄数以亿计的帐号，关系数据库也很难应付。

* 3、High Scalability && High Availability- 对数据库的高可扩展性和高可用性的需求

在基于web的架构当中，数据库是最难进行横向扩展的，当一个应用系统的用户量和访问量与日俱增的时候，你的数据库却没有办法像web server和app server那样简单的通过添加更多的硬件和服务节点来扩展性能和负载能力。对于很多需要提供24小时不间断服务的网站来说，对数据库系统进行升级和扩展是非常痛苦的事情，往往需要停机维护和数据迁移，为什么数据库不能通过不断的添加服务器节点来实现扩展呢？

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。

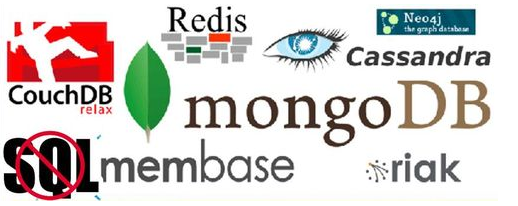
### 3. NoSQL特点

在大数据存取上具备关系型数据库无法比拟的**性能优势(查询)**，例如：

1. 易扩展
   * 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。也无形之间，在架构的层面上带来了可扩展的能力。
2. 大数据量，高性能
   * 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。
3. 灵活的数据模型
   * 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。这点在大数据量的Web2.0时代尤其明显。
4. 高可用
   * NoSQL在不太影响性能的情况，就可以方便的实现高可用的架构。比如Cassandra，HBase模型，通过复制模型也能实现高可用。

综上所述，NoSQL的非关系特性使其成为了后Web2.0时代的宠儿，助力大型Web2.0网站的再次起飞，是一项全新的数据库革命性运动。

### 4. 主流NoSQL产品



* NoSQL数据库的四大分类如下：
  + 键值(Key-Value)存储数据库

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。 **缓存、分布式集群**

数据模型： 一系列键值对

优势： 快速查询(由它的存储结构决定的)

劣势： 存储的数据缺少结构化

* + 列存储数据库

相关产品：Cassandra, HBase, Riak (大数据)

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

* + 文档型数据库

相关产品：CouchDB、MongoDB 类似html

典型应用：Web应用（与Key-Value类似，Value是结构化的）

数据模型： 一系列键值对

优势：数据结构要求不严格

劣势： 查询性能不高，而且缺乏统一的查询语法

* + 图形(Graph)数据库(腾讯) A **B** C D E

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。

## 二、Redis概述

### 1. Redis的由来

2008年，意大利的一家创业公司Merzia推出了一款基于MySQL的网站实时统计系统LLOOGG，然而没过多久该公司的创始人 Salvatore Sanfilippo便 对MySQL的性能感到失望，于是他决定亲自为LLOOGG量身定做一个数据库，并于2009年开发完成，这个数据库就是Redis。 不过Salvatore Sanfilippo并不满足只将Redis用于LLOOGG这一款产品，而是希望更多的人使用它，于是在同一年Salvatore Sanfilippo将Redis开源发布，并开始和Redis的另一名主要的代码贡献者Pieter Noordhuis一起继续着Redis的开发，直到今天。

Salvatore Sanfilippo自己也没有想到，短短的几年时间，Redis就拥有了庞大的用户群体。Hacker News在2012年发布了一份数据库的使用情况调查，结果显示有近12%的公司在使用Redis。国内如新浪微博、街旁网、知乎网，国外如GitHub、Stack Overflow、Flickr等都是Redis的用户。

### 2. 什么是Redis

Redis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，目前为止Redis支持的键值数据类型如下：[下面的这些类型都是value的数据类型]

1. 字符串类型string【最为重要！】
2. 散列类型hash
3. 列表类型list
4. 集合类型set
5. 有序集合类型sortedset

key都是string类型的

官方提供测试数据：50个并发执行100000个请求,读的速度是110000次/s,写的速度是81000次/s 。数据仅供参考，根据服务器配置会有不同结果。

### 3. Redis常用应用场景

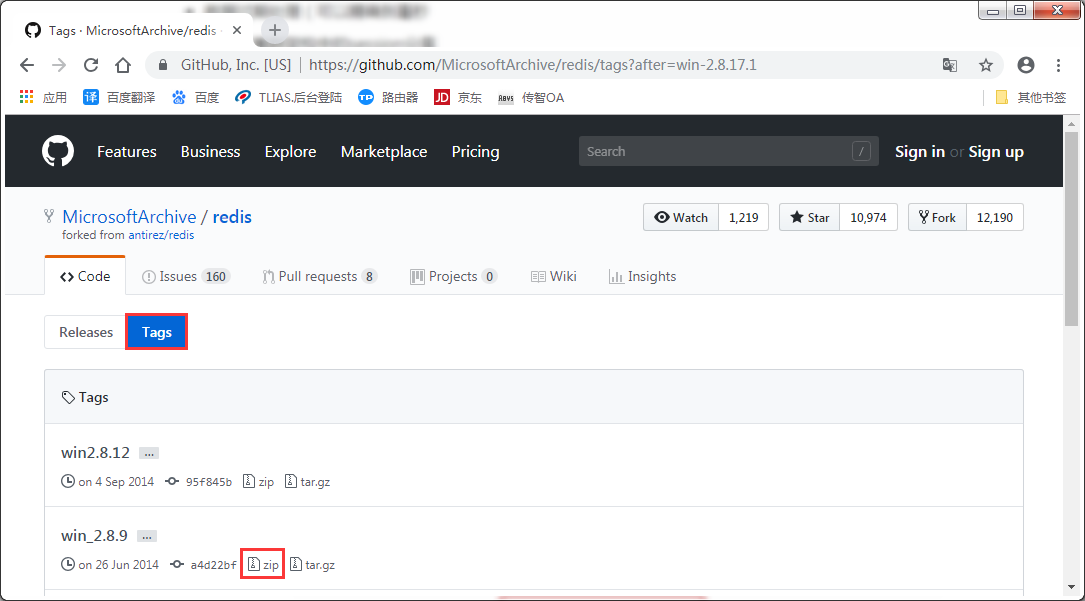
* Redis充当缓存（数据查询、短连接、新闻内容、商品内容等等）。（最多使用，变化不大）
* 聊天室的在线好友列表。
* 任务队列。（秒杀、抢购、12306等等）
* 应用排行榜。
* 网站访问统计。
* 数据过期处理（可以精确到毫秒）
* 分布式集群架构中的session分离。【第一个大型项目中使用】

## 三、Windows版本的Redis安装和使用

### 1. windows版Redis的下载

官方提倡使用Linux版的Redis，所以官网值提供了Linux版的Redis下载，我们可以从GitHub上下载window版的Redis，具体链接地址如下：

* 官网下载地址：http://redis.io/download
* github下载地址：https://github.com/MSOpenTech/redis/tags



在今天的课程资料中提供的下载完毕的window版本的Redis：



### 2. window版Redis的目录结构

解压Redis压缩包后，见到如下目录机构：

|  |  |
| --- | --- |
| **目录或文件** | **作用** |
| redis-benchmark | 性能测试工具 |
| redis-check-aof | AOF文件修复工具 |
| redis-check-dump | RDB文件检查工具（快照持久化文件） |
| redis-cli | 命令行客户端 |
| redis-server | redis服务器启动命令 |
| redis.windows.conf | redis核心配置文件 |

### 3. window版Redis的安装与启动

#### 3.1 window版Redis的安装

window版的安装及其简单，解压Redis压缩包完成即安装完毕

#### 3.2 window版Redis的启动与关闭

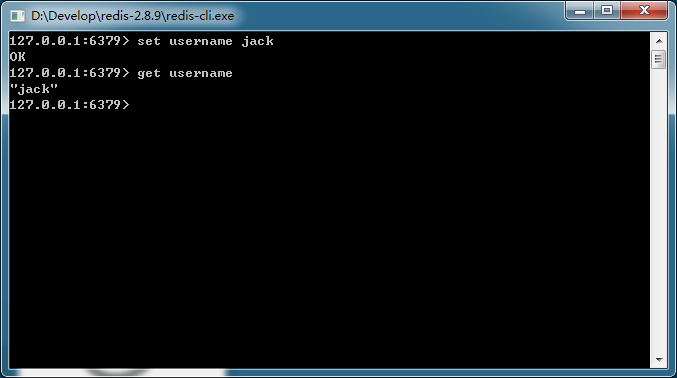
* 双击Redis目录中redis-server.exe可以启动redis服务，Redis服务占用的端口是6379



* 关闭Redis的控制台窗口就可以关闭Redis服务

#### 3.3 window版Redis的使用

* 双击Redis目录中redis-cli.exe可以启动redis客户端



## 四、Redis数据类型

### 1. Redis数据结构介绍

Redis是一种高级的key-value的存储系统，其中value支持五种数据类型

**字符串(string) 使用最多，和json连用**

哈希(hash) 一般用于存储JavaBean数据，但是被string+json形式完全替代

字符串列表(list) 链表，实时排名

字符串集合(set) 唯一性，使用环境较少

有序字符串集合(sorted set) 用于网站排名，竞价排名，变化不能太快！

Key全都是字符串的！value有上述5种类型，最重要的是string！！！

而关于key的定义，需要大家注意以下几点：

* Key不要太长，最好不要超过1024个字节，这不仅会消耗内存还会降低查找的效率
* Key不要太短，如果太短会降低key的可读性

在项目中，定义key最好有一个统一的命名规范，越短效率越高，但是必须有意义！

### 2. 存储string

#### 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**存入和获取的数据相同【不存在乱码问题！】**。在Redis中字符串类型的Value最多可以容纳的数据长度是512M。

#### 常用命令

##### 2.2.1 赋值

* set key value：设定key持有指定的字符串value，如果该key存在则进行覆盖操作。总是返回”OK”

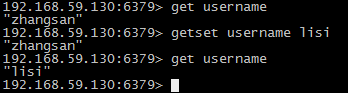


##### 2.2.2 取值

* get key：获取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；如果该key不存在，返回(nil)。

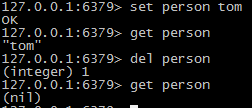


* getset key value：先获取该key的值，然后在设置该key的值。



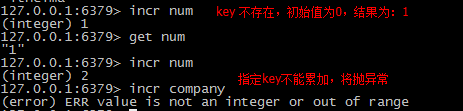
##### 2.2.3 删除

* del key ：删除指定key 【其它类型的数据也可以删除！！！！】



##### 2.2.4 数值增减

* incr key：将指定的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的值不能转成整型，如hello，该操作将执行失败并返回相应的错误信息。

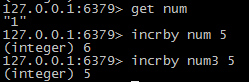


* decr key：将指定的key的value原子性的递减1.如果该key不存在，其初始值为0，在incr之后其值为-1。如果value的值不能转成整型，如hello，该操作将执行失败并返回相应的错误信息。



#### 2.3 扩展命令（了解）

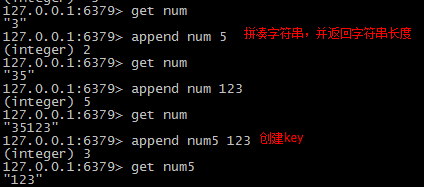
* incrby key increment：将指定的key的value原子性增加increment，如果该key不存在，器初始值为0，在incrby之后，该值为increment。如果该值不能转成整型，如hello则失败并返回错误信息



* decrby key decrement：将指定的key的value原子性减少decrement，如果该key不存在，器初始值为0，在decrby之后，该值为decrement。如果该值不能转成整型，如hello则失败并返回错误信息



* append key value：拼凑字符串。如果该key存在，则在原有的value后追加该值；如果该key不存在，则重新创建一个key/value



### 3. 存储hash(了解,实际开发会使用string+json替换它)

#### 3.1 概述

Redis中的Hash类型可以看成具有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。

#### 常用命令

##### 3.2.1 赋值

* hset key field value：为指定的key设定field/value对（键值对）。



* hmset key field value [field2 value2 …]：设置key中的多个filed/value



##### 3.2.2 取值

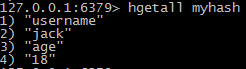
* hget key field：返回指定的key中的field的值



* hmget key fileds：获取key中的多个filed的值

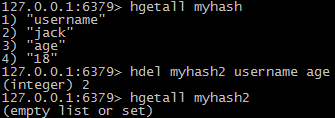


* hgetall key：获取key中的所有filed-vaule



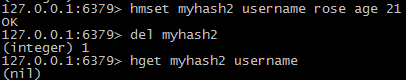
##### 3.2.3 删除

* hdel key field [field … ] ：可以删除一个或多个字段，返回值是被删除的字段个数



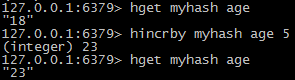


* del key ：删除整个list



##### 3.2.4 增加数字

* hincrby key field increment：设置key中filed的值增加increment，如：age增加20

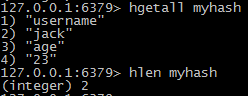


#### 3.3 自学命令

* hexists key field：判断指定的key中的filed是否存在



* hlen key：获取key所包含的field的数量



* hkeys key ：获得所有的key

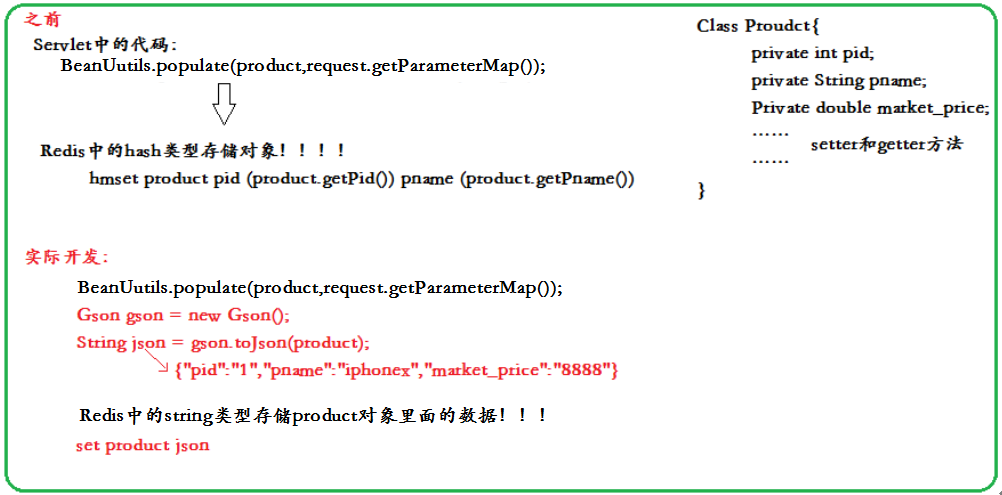


* hvals key：获得所有的value



#### 3.4 为什么hash类型不怎么使用了？

实际开发中会使用string类型+json数据格式的数据来取代hash类型！



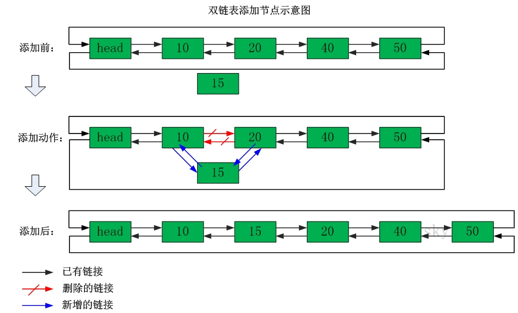
### 4. 存储list

#### 4.1. 概述

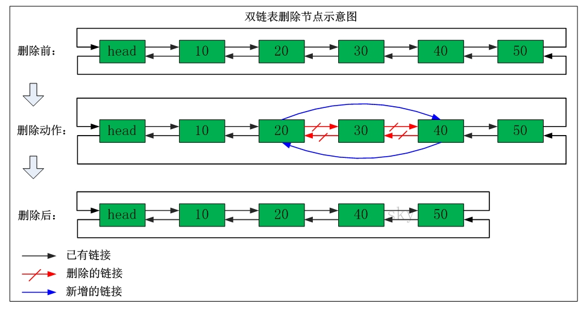
在Redis中，List类型是按照插入顺序排序的字符串**链表**。和数据结构中的普通链表一样，我们可以在其头部(left)和尾部(right)添加新的元素。在插入时，如果该键并不存在，Redis将为该键创建一个新的链表。与此相反，如果链表中所有的元素均被移除，那么该键也将会被从数据库中删除。List中可以包含的最大元素数量是4294967295。

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。

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涉及到位移操作，所以比较慢。(存放的内容越多，速度越慢)
2. LinkedList使用双向链接方式存储数据，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。(插入修改快，查询慢)
3. 双向链表中添加数据



1. 双向链表中删除数据



#### 4.2. 常用命令

##### 4.2.1 两端添加

* lpush key values[value1 value2…]：在指定的key所关联的list的头部插入所有的values，如果该key不存在，该命令在插入的之前创建一个与该key关联的空链表，之后再向该链表的头部插入数据。插入成功，返回元素的个数。（l：left ；push:推）

lpush mylist a b c



进行查看(使用lrange)，由于它是从左侧依次存放数据(类似于压栈：先进后出)

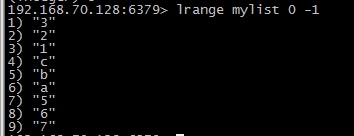


再次运行 lpush mylist 1 2 3

* rpush key values[value1、value2…]：在该list的尾部添加元素。（r：right ；push:推）

rpush mylist 5 6 7

再进行查看：使用lrange 命令不需要指定长度，直接给起始索引位置。

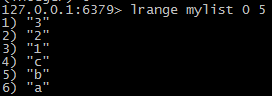


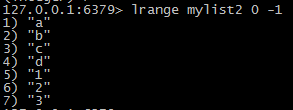
##### 4.2.2 查看列表

* lrange key start end：获取链表中从start到end的元素的值，start、end从0开始计数；也可为负数，若为-1则表示链表尾部的元素，-2则表示倒数第二个，依次类推…

(lrange: l:从左侧，range:范围)

直接参看上述图例：rpush的命令演示

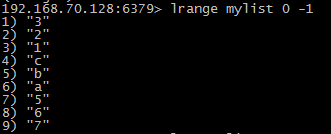




##### 4.2.3 两端弹出(本质是删除)

* lpop key：返回并弹出指定的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；若key存在，则返回链表的头部元素。

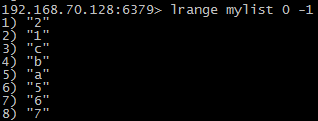
初始数据：



执行lpop命令，将链表的第一个元素弹出返回(之前里面的数据会被删除)



查看mylist中的数据

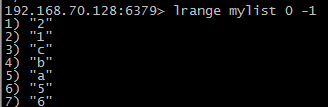


* rpop key：从尾部弹出元素。

从上面的数据可以得知最终被弹出的数据是7，在mylist保存着2 1 c b a 5 6



查看mylist中的数据



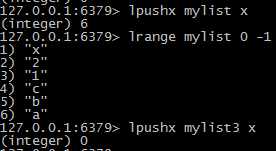
##### 4.2.4 获取列表中元素的个数

* llen key：返回指定的key关联的链表中的元素的数量。

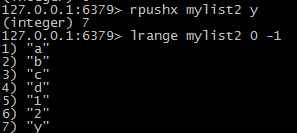


#### 4.3. 扩展命令（了解）

* lpushx key value：仅当参数中指定的key存在时，向关联的list的头部插入value。如果不存在，将不进行插入。(只能插入一个数据)

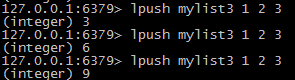


* rpushx key value：在该list的尾部添加元素



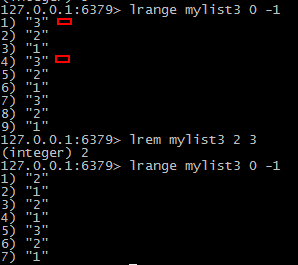
* lrem key count value：删除count个值为value的元素，如果count大于0，从头向尾遍历**并删除count个值为value的元素**，如果count小于0，则从尾向头遍历并删除。如果count等于0，则删除链表中所有等于value的元素。

0)初始化数据



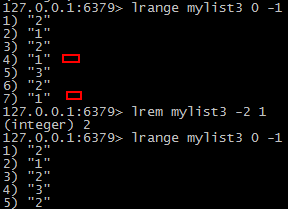
1) 从头删除，2个数字“3”

lrem mylist3 2 3



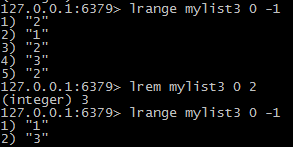
2) 从尾删除，2个数字“1” **-2不是代表倒数第二个元素，而是从右边开始，删除2个元素**

lrem mylist3 -2 1



3) 删除所有数字“2”  **参数0 不代表个数，而是从0起始索引位置删除所有**

lrem mylist3 0 2



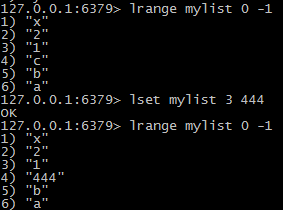
* lset key index value：设置链表中的index的脚标的元素值value(执行的是替换该位置的数据)，0代表链表的头元素，-1代表链表的尾元素。操作链表的脚标不存在则抛异常。

先查看mylist的长度

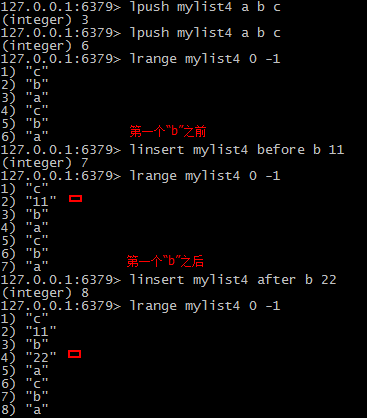


设置指定角标值(最大角标是9)





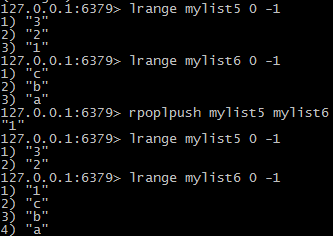
* linsert key before|after pivot value：在pivot元素前或者后插入value这个元素。



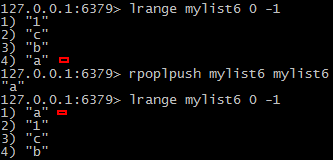
* rpoplpush resource destination：将链表中的尾部元素弹出并添加到头部。[循环操作] resource和destination指的是2个list集合



1) 将mylist5右端弹出，压入到mylist6左边。



2) 将mylist6右端数据弹出，压入到左端



#### 4.4. 使用场景

**rpoplpush的使用场景**：

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操作向链表中左侧添加新的元素，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操作从链表中右侧取出元素，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。

|  |
| --- |
|  |

### 5. 存储set

#### 5.1. 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。需要说明的是，这些操作的时间复杂度为O(1)，即常量时间内完成次操作。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(并集)、intersections(交集)和differences(差集)。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。

#### 5.2. 常用命令

##### 5.2.1 添加/删除元素

* sadd key values[value1、value2…]：向set中添加数据，如果该key的值已有则不会重复添加



可以查看里面的数据：smembers myset



* srem key members[member1、member2…]：删除set中指定的成员(成员可以是多个)



再次查看myset中的数据



##### 5.2.2 获得集合中的元素

* smembers key：获取set中所有的成员 s：set member:成员



* sismember key member：判断参数中指定的成员是否在该set中，1表示存在，0表示不存在或者该key本身就不存在。（无论集合中有多少元素都可以极速的返回结果）

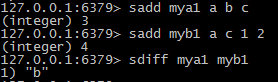


##### 5.2.3 集合的差集运算 A-B

* sdiff key1 key2…：返回key1与key2中相差的成员，**而且与key的顺序有关**。即**返回差集**。



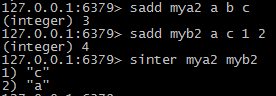
（属于A并且不属于B的元素构成的集合）



##### 5.2.4 集合的交集运算 A ∩ B

* sinter key1 key2 key3…：**返回交集**。

（属于A且属于B的元素构成的集合）

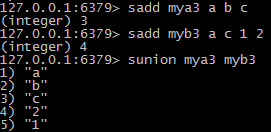


##### 5.2.5 集合的并集运算 A ∪ B

* sunion key1 key2 key3…：**返回并集**。

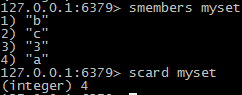


（属于A或者属于B的元素构成的集合）



#### 5.3. 扩展命令（了解）

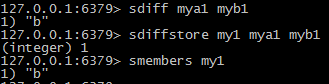
* scard key：获取set中成员的数量



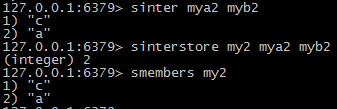
* srandmember key：随机返回set中的一个成员



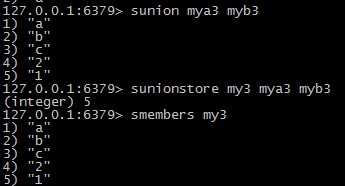
* sdiffstore destination key1 key2…：将key1、key2相差的成员存储在destination上



* sinterstore destination key[key…]：将返回的交集存储在destination上



* sunionstore destination key[key…]：将返回的并集存储在destination上



#### 5.4. 使用场景

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。

Stirng ip = request.getRemoteAddr()

sadd ip ip

smembers ip

scar dip //统计出ip的总数！！

jedis.sadd(“ip”,ip);

jedis.smembers(“ip”);//返回一个集合对象，然后就可以遍历获得所有的数据了！

2.充分利用Set类型的服务端聚合(并集、差集、交集)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(交集：有相同元素的那一部分数据)就可以充分发挥它的方便和效率的优势了。

myset1 c1 c3 c4 c5(iphone)

myset2 c2 c5 c6 c3(ipad)

sinter myset1 myset2 c3 c5

### 6. 存储sortedset

#### 6.1. 概述

Sorted-Set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低到高的排序(默认)。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。(数据不能重复，分数可以，数据可以排序)

在Sorted-Set中添加、删除或更新一个成员都是非常快速的操作，其时间复杂度为集合中成员数量的对数。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。

例如：游戏排名、微博热点话题等使用场景。

#### 6.2. 常用命令

##### 6.2.1 添加元素

* zadd key score member score2 member2 … ：将所有成员以及该成员的分数存放到sorted-set中。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。(score可以重复，member不可以重复)



再次添加zhangsan，数据(member)不会被添加成功，分数被修改



进行查看，发现分数被修改了



zrange key start end [withscores] 查询所有信息

##### 6.2.2 获得元素

* zscore key member：返回指定成员的分数



* zcard key：获取集合中的成员数量 (set中 scard key)

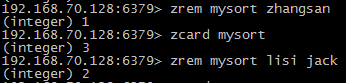
之前存放了3个，然后再次添加zhangsan不成功(只是替换了分数)，最后添加了Jack，所以总共只有4个成员



##### 6.2.3 删除元素

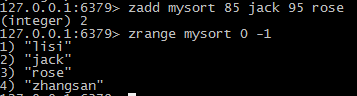
* zrem key member[member…]：移除集合中指定的成员，可以指定多个成员。

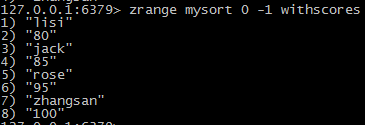
先删除1个，剩余3个，再删除2个，剩余1个(wangwu)



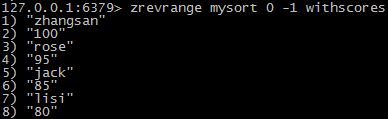
##### 6.2.4 范围查询

* zrange key start end [withscores]：获取集合中脚标为start-end的成员，[withscores]参数表明返回的成员包含其分数。(withscores可选参数)





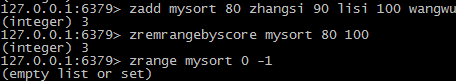
* zrevrange key start stop [withscores]：照元素分数从大到小的顺序返回索引从start到stop之间的所有元素（包含两端的元素）



* zremrangebyrank key start stop: 按照排名范围删除元素(从小到大删除)

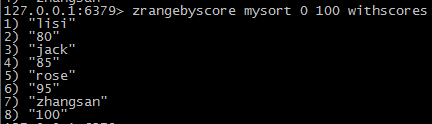


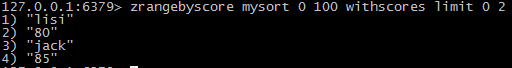
* zremrangebyscore key min max：按照分数范围删除元素



#### 6.3. 扩展命令（了解，课下自行研究）

* zrangebyscore key min max [withscores] [limit offset count]：返回分数在[min,max]的成员并按照分数从低到高排序。[withscores]：显示分数；[limit offset count]：offset，表明从脚标为offset的元素开始并返回count个成员。





* zincrby key increment member：设置指定成员的增加的分数。返回值是更改后的分数。



* zcount key min max：获取分数在[min,max]之间的成员



* zrank key member：返回成员在集合中的排名。（从小到大）



* zrevrank key member：返回成员在集合中的排名。（从大到小）



#### 6.4. 使用场景

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(数据被覆盖，分数被更新)，此后再通过ZRANGE命令(查询)获取积分TOPTEN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。zrangebyscore key min max withscores limit offset count

## 五、Redis通用命令

* keys pattern：获取所有与pattern匹配的key，返回所有与该key匹配的keys。\*表示任意一个或多个字符，?表示任意一个字符



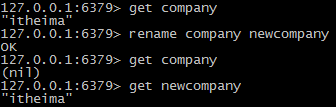
* del key1 key2…：删除指定的key



* exists key：判断该key是否存在，1代表存在，0代表不存在



* rename key newkey：为当前的key重命名



* expire key ：设置过期时间，单位：秒



* ttl key：获取该key所剩的超时时间，如果没有设置超时，返回-1。如果返回-2表示超时不存在(key不存在)。





* type key：获取指定key的类型。该命令将以字符串的格式返回。 返回的字符串为string、list、set、hash和zset，如果key不存在返回none。

## 六、Redis持久化

### 1. Redis持久化概述

Redis的高性能是由于其将所有数据都存储在了内存中，为了使Redis在重启之后仍能保证数据不丢失，需要将数据从内存中同步到硬盘中，这一过程就是持久化。Redis支持两种方式的持久化，一种是RDB方式，一种是AOF方式。可以单独使用其中一种或将二者结合使用。

* RDB持久化（默认支持，无需配置）【快照机制】

该机制是指在指定的时间间隔内将内存中的数据集快照写入磁盘。

* AOF持久化【日志机制】

该机制将以日志的形式记录服务器所处理的每一个写操作，在Redis服务器启动之初会读取该文件来重新构建数据库，以保证启动后数据库中的数据是完整的。

* 无持久化

我们可以通过配置的方式禁用Redis服务器的持久化功能，这样我们就可以将Redis视为一个功能加强版的memcached了。

* redis可以同时使用RDB和AOF

### 2. RDB持久化机制

#### 2.1 RDB持久化机制优点

* 一旦采用该方式，那么你的整个Redis数据库将只包含一个文件，这对于文件备份而言是非常完美的。比如，你可能打算每个小时归档一次最近24小时的数据，同时还要每天归档一次最近30天的数据。通过这样的备份策略，一旦系统出现灾难性故障，我们可以非常容易的进行恢复。
* 对于灾难恢复而言，RDB是非常不错的选择。因为我们可以非常轻松的将一个单独的文件压缩后再转移到其它存储介质上
* 性能最大化。对于Redis的服务进程而言，在开始持久化时，它唯一需要做的只是fork（分叉）出子进程，之后再由子进程完成这些持久化的工作，这样就可以极大的避免服务进程执行IO操作了。

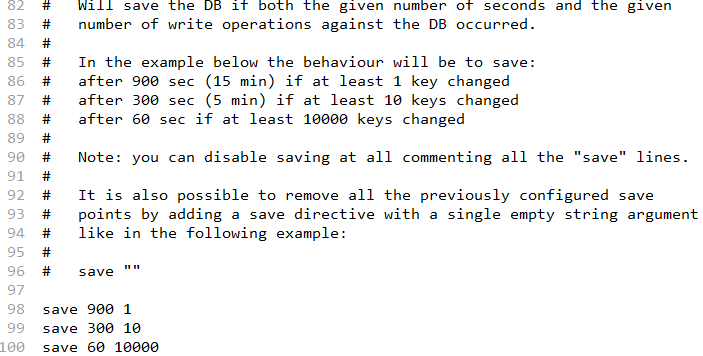
相比于AOF机制，如果数据集很大，**RDB的启动效率会更高**

#### 2.2 RDB持久化机制缺点

* 如果你想保证数据的高可用性，即最大限度的避免**数据丢失**，那么RDB将不是一个很好的选择。因为系统一旦在定时持久化之前出现宕机现象，此前没有来得及写入磁盘的数据都将丢失。
* 由于RDB是通过fork子进程来协助完成数据持久化工作的，因此，如果当数据集较大时，可能会导致整个服务器停止服务几百毫秒，甚至是1秒钟

#### 2.3 RDB持久化机制的配置

在redis.windows.conf配置文件中有如下配置：(默认不用配置，数据持久化需要在客户端执行save命令！)



其中，上面配置的是RDB方式数据持久化时机：

|  |  |  |  |
| --- | --- | --- | --- |
| **关键字** | **时间(秒)** | **key修改数量** | **解释** |
| save | 900 | 1 | 每900秒(15分钟)至少有1个key发生变化，则dump内存快照 |
| save | 300 | 10 | 每300秒(5分钟)至少有10个key发生变化，则dump内存快照 |
| save | 60 | 10000 | 每60秒(1分钟)至少有10000个key发生变化，则dump内存快照 |

### 3. AOF持久化机制(日志)

#### 3.1 AOF持久化机制优点

* 该机制可以带来**更高的数据安全性**，即数据持久性。Redis中提供了3中同步策略，即每秒同步、每修改同步和不同步。事实上，每秒同步也是异步完成的，其效率也是非常高的，所差的是一旦系统出现宕机现象，那么这一秒钟之内修改的数据将会丢失。而每修改同步，我们可以将其视为同步持久化，即每次发生的数据变化都会被立即记录到磁盘中。可以预见，这种方式在效率上是最低的。至于无同步，无需多言，我想大家都能正确的理解它。
* 由于该机制对日志文件的写入操作采用的是append模式，因此在写入过程中即使出现宕机现象，也不会破坏日志文件中已经存在的内容。然而如果我们本次操作只是写入了一半数据就出现了系统崩溃问题，不用担心，在Redis下一次启动之前，我们可以通过redis-check-aof工具来帮助我们解决数据一致性的问题。
* 如果日志过大，Redis可以自动启用rewrite机制。即Redis以append模式不断的将修改数据写入到老的磁盘文件中，同时Redis还会创建一个新的文件用于记录此期间有哪些修改命令被执行。因此在进行rewrite切换时可以更好的保证数据安全性。
* AOF包含一个格式清晰、易于理解的日志文件用于记录所有的修改操作。事实上，我们也可以通过该文件完成数据的重建

#### 3.2 AOF持久化机制缺点

* 对于相同数量的数据集而言，AOF文件通常要大于RDB文件
* 根据同步策略的不同，AOF在运行效率上往往会慢于RDB。总之，每秒同步策略的效率是比较高的，同步禁用策略的效率和RDB一样高效。

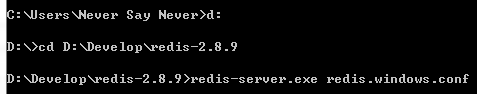
#### AOF持久化机制配置

##### 3.3.1 开启AOF持久化

修改文件：redis.windows.conf



将appendonly修改为yes，开启aof持久化机制，默认会在目录下产生一个appendonly.aof文件(没有的话,执行一下redis.windows.conf文件)



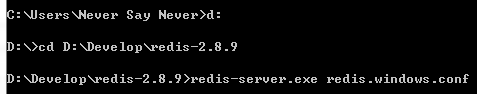
##### 3.3.1 AOF持久化时机



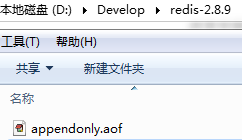
上述配置为aof持久化的时机，解释如下：

|  |  |  |
| --- | --- | --- |
| **关键字** | **持久化时机** | **解释** |
| appendfsync | always | 每执行一次更新命令，持久化一次 |
| appendfsync | everysec | 每秒钟持久化一次 |
| appendfsync | no | 不持久化 |

后面要想这种机制能够顺利完成(获得之前的数据，以及保存新操作的数据)，需要在启动redis服务器端的时候加载redis.windows.conf文件！



此时多了一个文件appendonly.aof



### 4. 两种机制的选择问题

一般来说， 如果想达到足以媲美 PostgreSQL 的数据安全性， 你应该同时使用两种持久化功能。

如果你非常关心你的数据， 但仍然可以承受数分钟以内的数据丢失， 那么你可以只使用 RDB 持久化。

其余情况我个人喜好选择AOF

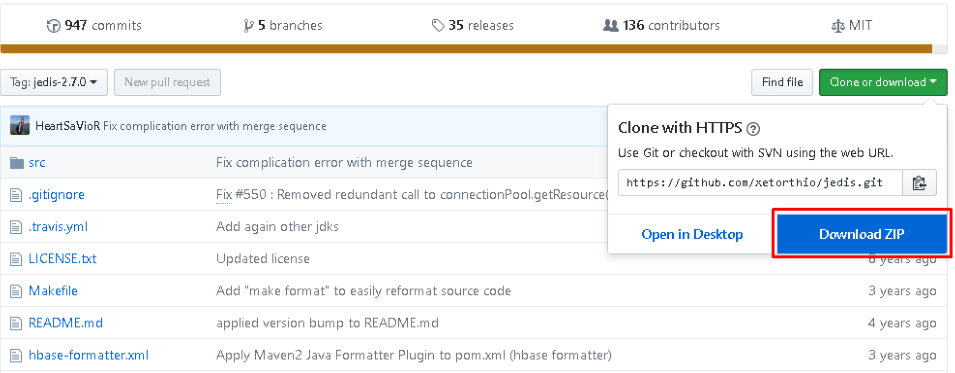
## 七、Jedis基本使用

### 1. Jedis介绍

Redis不仅是使用命令来操作，现在基本上主流的语言都有客户端支持，比如java、C、C#、C++、php、Node.js、Go等。

在官方网站里列一些Java的客户端，有Jedis、Redisson、Jredis、JDBC-Redis、等其中官方推荐使用Jedis和Redisson。 在企业中用的最多的就是Jedis，下面我们就重点学习下Jedis。

Jedis同样也是托管在github上，地址：https://github.com/xetorthio/jedis



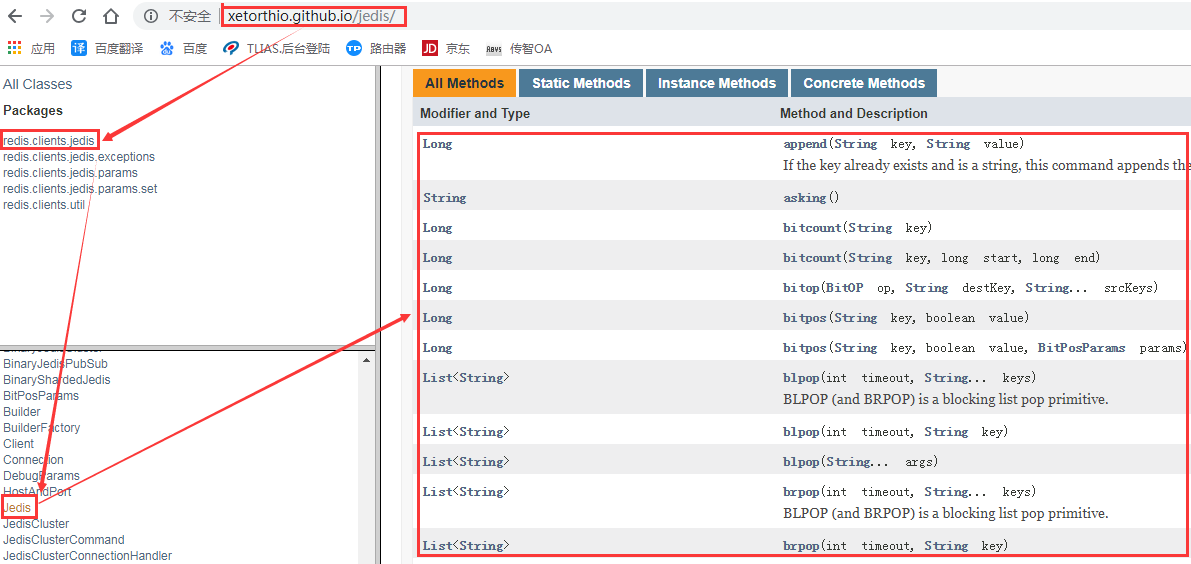
使用Jedis操作redis需要导入jar包如下：



### 2. Jedis基本操作

#### 2.1 Jedis官方文档查询方式

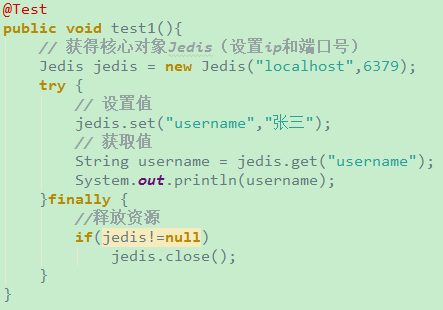
* 官方文档地址：http://xetorthio.github.io/jedis/
* 官方文档查询方式：



#### 2.2 Jedis常用API

|  |  |
| --- | --- |
| **方法** | **解释** |
| new Jedis(host, port) | 创建jedis对象，参数host是redis服务器地址，参数port是redis服务端口 |
| set(key,value) | 设置字符串类型的数据 |
| get(key) | 获得字符串类型的数据 |
| hset(key,field,value) | 设置哈希类型的数据 |
| hget(key,field) | 获得哈希类型的数据 |
| lpush(key,values) | 设置列表类型的数据 |
| lpop(key) | 列表左面弹栈 |
| rpop(key) | 列表右面弹栈 |
| del(key) | 删除指定的key |

#### 2.3 Jedis基本操作代码演示



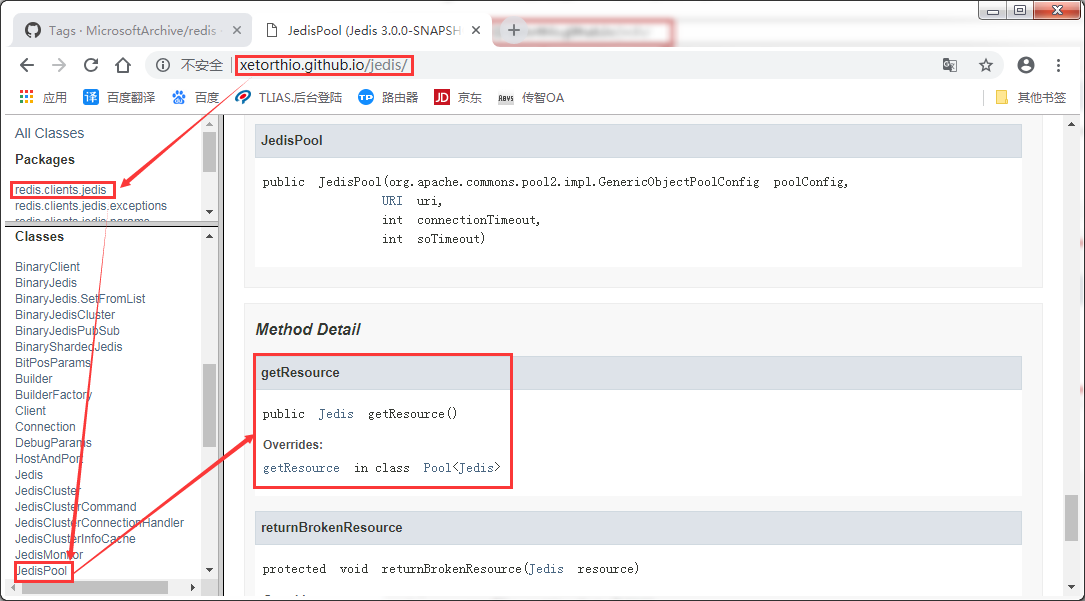
要想让数据持久化到磁盘，可以在设置值之后调用jedis.save()方法！

### 3. Jedis连接池的使用

#### 2.1 Jedis连接池的介绍

jedis连接资源的创建与销毁是很消耗程序性能，所以jedis为我们提供了jedis的池化技术，jedisPool在创建时初始化一些连接资源存储到连接池中，使用jedis连接资源时不需要创建，而是从连接池中获取一个资源进行redis的操作，使用完毕后，不需要销毁该jedis连接资源，而是将该资源归还给连接池，供其他请求使用。

#### 2.2 Jedis官方文档查询方式



#### 2.3 Jedis连接池代码演示(JedisPool)



#### 2.4 Jedis连接池工具类(JedisUtils)

##### 2.4.1 配置文件(jedis.properties)

**jedis.host**=localhost  
**jedis.port**=6379  
**jedis.maxTotal**=30  
**jedis.maxIdle**=10

##### 2.4.2 工具类



##### 2.4.3 工具类测试类

