# Redis

## NoSQL的概述

### 什么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。

### 为什么有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

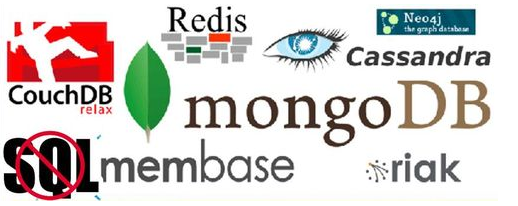
类似Facebook，twitter，Friendfeed这样的SNS网站，每天用户产生海量的用户动态，以Friendfeed为例，一个月就达到了2.5亿条用户动态，对于关系数据库来说，在一张2.5亿条记录的表里面进行SQL查询，效率是极其低下乃至不可忍受的。再例如大型web网站的用户登录系统，例如腾讯，盛大，动辄数以亿计的帐号，关系数据库也很难应付。

* 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。

### 常见的NoSQL



* 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。

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

* + 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

* + 文档型数据库

相关产品：CouchDB、MongoDB

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

相关数据库：Neo4J、InfoGrid、Infinite Graph

典型应用：社交网络

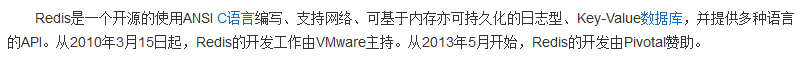
数据模型：图结构

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。

## Redis的概述

### 什么是Redis



目前为止Redis支持的键值数据类型如下：

1. 字符串类型
2. 散列类型
3. 列表类型
4. 集合类型
5. 有序集合类型。

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。

### 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）。（最多使用）**
* 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。

## 安装Redis

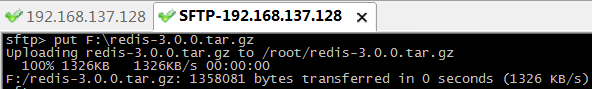
### 安装Redis

* 步骤1：将Windows下下载的压缩文件上传到Linux下。通过secureCRT进行上传，步骤如下：

 alt + p

 put F:/redis-3.0.0.tar.gz





* 步骤2： 解压文件

 tar –zxvf redis-3.0.0.tar.gz



* 步骤3： 编译redis (编译，将.c文件编译为.o文件)

 进入解压文件夹，cd redis-3.0.0

 执行make



1）如果没有安装gcc，编译将出现错误提示。（如果安装失败，必须删除文件夹，重写解压）



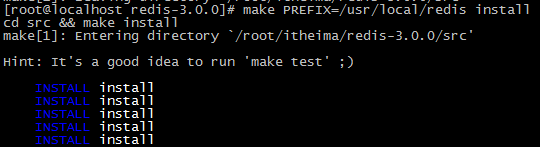
2）安装成功



* 步骤4： 安装

 make PREFIX=/usr/local/src/redis install





安装完后，在/usr/local/src/redis/bin下有几个可执行文件

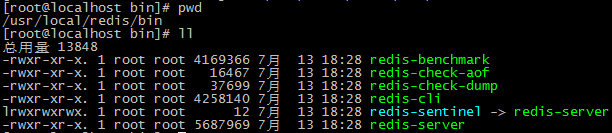
redis-benchmark ----性能测试工具

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

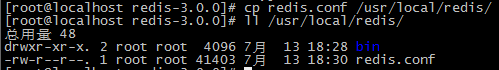
redis-server ----redis服务器启动命令



* 步骤5： copy文件

redis启动需要一个配置文件，可以修改端口号等信息。

cp redis.conf /usr/local/redis



### Reids的启动

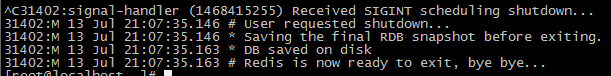
#### 前端模式启动

直接运行bin/redis-server将以前端模式启动，前端模式启动的缺点是启动完成后，不能再进行其他操作，如果要操作必须使用ctrl+c，同时redis-server程序结束，不推荐使用此方法。

/usr/local/redis/bin/redis-server



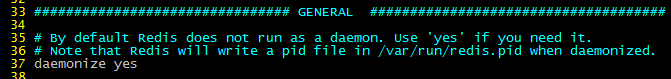
使用CTRL+ C 停止前端模式



#### 后端模式

* 修改redis.conf配置文件， daemonize yes 以后端模式启动。

vim /usr/local/redis/redis.conf



* 启动时，指定配置文件

cd /usr/local/redis/

./bin/redis-server ./redis.conf



* Redis默认端口6379，通过当前服务进行查看

ps -ef | grep -i redis

## 使用Redis

### Redis五种数据类型

1. 字符串（String）
2. 哈希（hash）
3. 字符串列表（list）
4. 字符串集合（set）
5. 有序字符串集合（sorted set）

### 字符串类型

#### 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”



#### 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)。

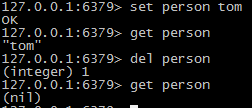


* getset key value：先获取该key的值，然后在设置该key的值。



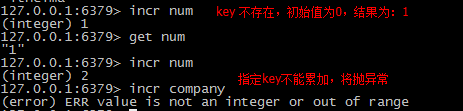
#### 删除

* del key ：删除指定key



#### 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。

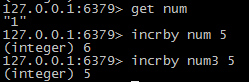


* decr key：将指定的key的value原子性的递减1.如果该key不存在，其初始值为0，在incr之后其值为-1。如果value的值不能转成整型，如hello，该操作将执行失败并返回相应的错误信息。



#### 扩展命令（了解）

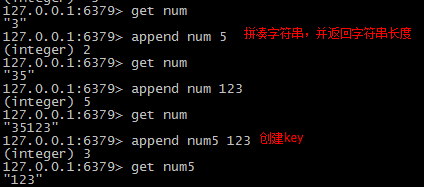
* incrby key increment：将指定的key的value原子性增加increment，如果该key不存在，器初始值为0，在incrby之后，该值为increment。如果该值不能转成整型，如hello则失败并返回错误信息



* 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



### hash数据类型

#### 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。

#### 存值

* hset key field value



* hmset key field value field value



#### 取值：

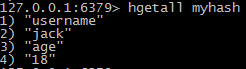
* hget key field：返回指定的key中的field的值



* hmget key fileds：获取key中的多个filed的值

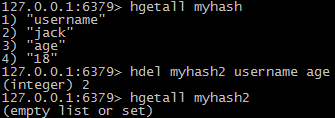


* hgetall key：获取key中的所有filed-vaule

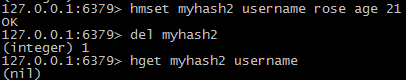


#### 删除：

* 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 ：删除整个内容



#### 增加数字

* hincrby key field value：为某个key的某个属性增加值

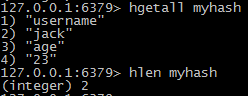


#### 自学命令（了解）

* 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



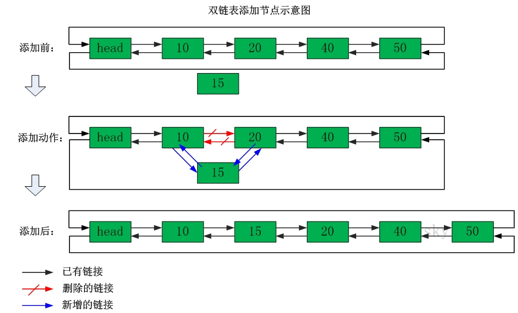
## list类型

### 概述

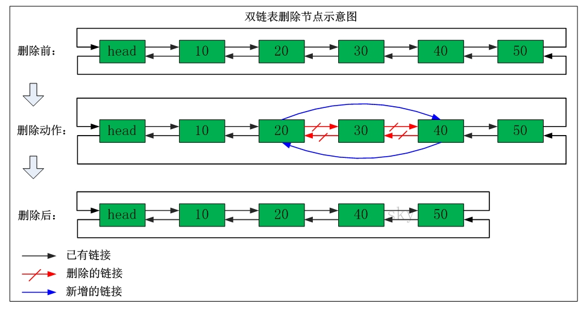
在Redis中，List类型是按照插入顺序排序的字符串链表。和数据结构中的普通链表一样，我们可以在其头部(left)和尾部(right)添加新的元素。在插入时，如果该键并不存在，Redis将为该键创建一个新的链表。与此相反，如果链表中所有的元素均被移除，那么该键也将会被从数据库中删除。List中可以包含的最大元素数量是4294967295。

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设计到位移操作，所以比较慢。
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。
3. 双向链表中添加数据



1. 双向链表中删除数据



### 常用命令

#### 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。

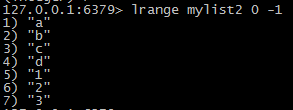
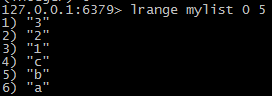


* rpush key values[value1、value2…]：在该list的尾部添加元素。



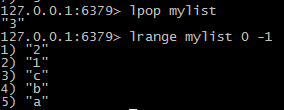
#### 查看列表

* 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…

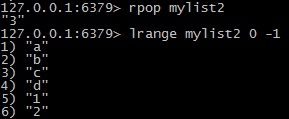


#### 两端弹出

* 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。

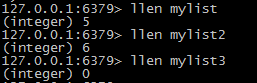


* rpop key：从尾部弹出元素。



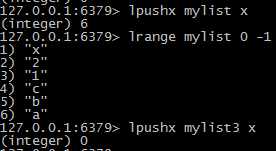
#### 获取列表中元素的个数

* llen key：返回指定的key关联的链表中的元素的数量。

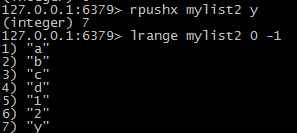


#### 扩展命令（了解）

* 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。

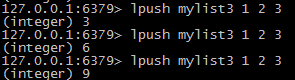


* rpushx key value：在该list的尾部添加元素



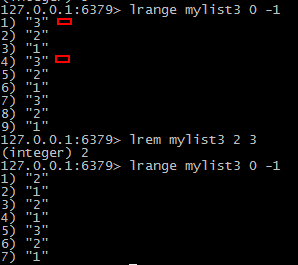
* lrem key count value：删除count个值为value的元素，如果count大于0，从头向尾遍历**并删除count个值为value的元素**，如果count小于0，则从尾向头遍历并删除。如果count等于0，则删除链表中所有等于value的元素。

0)初始化数据



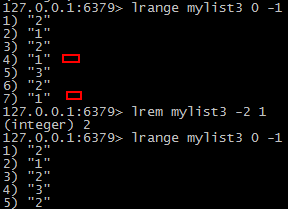
1) 从头删除，2个数字“3”

lrem mylist3 2 3



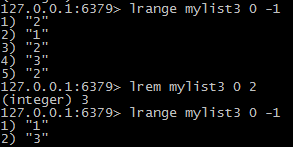
2) 从尾删除，2个数字“1”

lrem mylist3 -2 1

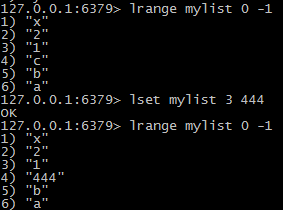


3) 删除所有数字“2”

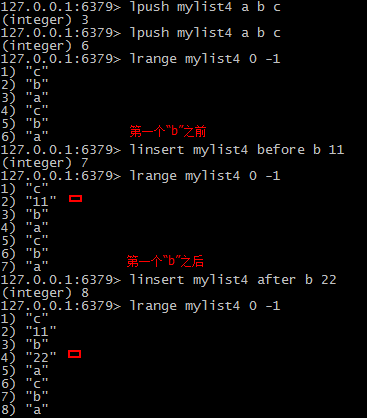
lrem mylist3 0 2



* lset key index value：设置链表中的index的脚标的元素值，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。



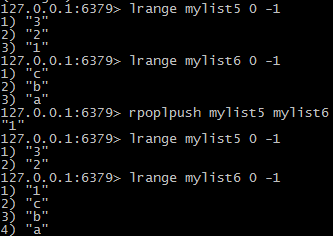
* linsert key before|after pivot value：在pivot元素前或者后插入value这个元素。



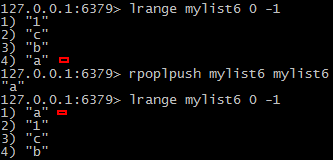
* rpoplpush resource destination：将链表中的尾部元素弹出并添加到头部。[循环操作]



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

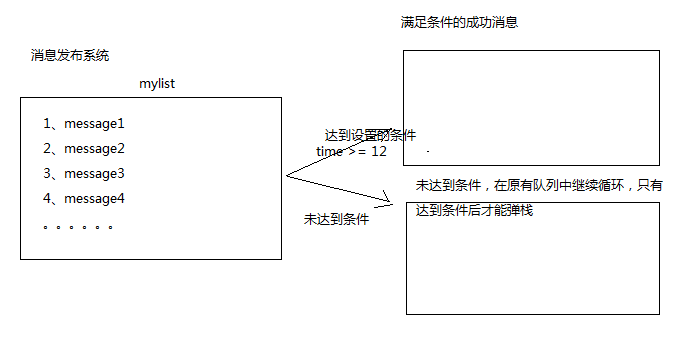


### 使用场景

**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。

## 存储set（了解）



### 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、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。

### 常用命令

#### 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

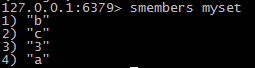


* srem key members[member1、member2…]：删除set中指定的成员



#### 获得集合中的元素

* smembers key：获取set中所有的成员



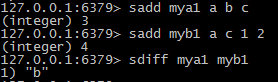
* sismember key member：判断参数中指定的成员是否在该set中，1表示存在，0表示不存在或者该key本身就不存在。（无论集合中有多少元素都可以极速的返回结果）



#### 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**。

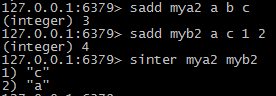
（属于A并且不属于B的元素构成的集合）



#### 集合的交集运算 A ∩ B

* sinter key1 key2 key3…：**返回交集**。

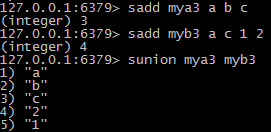
（属于A且属于B的元素构成的集合）



#### 集合的并集运算 A ∪ B

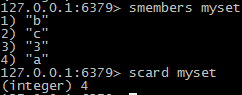
* sunion key1 key2 key3…：**返回并集**。

（属于A或者属于B的元素构成的集合）



#### 扩展命令（了解）

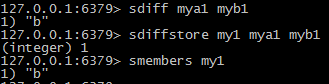
* scard key：获取set中成员的数量



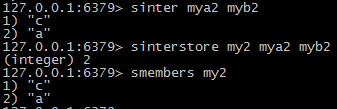
* srandmember key：随机返回set中的一个成员



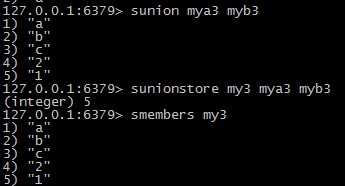
* sdiffstore destination key1 key2…：将key1、key2相差的成员存储在destination上



* 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



### 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。

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。

## 存储sortedset（了解）

### 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，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。

### 常用命令

#### 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。



#### 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



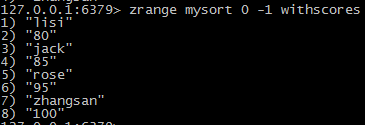
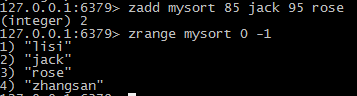
#### 删除元素

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

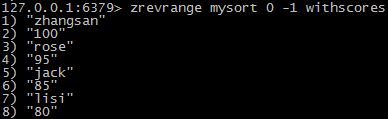


#### 范围查询

* 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。



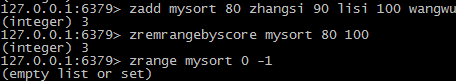
* zrevrange key start stop [withscores]：照元素分数从大到小的顺序返回索引从start到stop之间的所有元素（包含两端的元素）



* zremrangebyrank key start stop: 按照排名范围删除元素

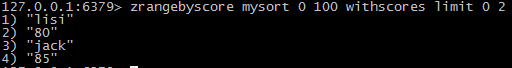
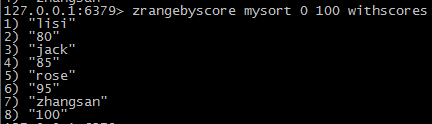


* zremrangebyscore key min max：按照分数范围删除元素



#### 扩展命令（了解）

* 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）



### 使用场景

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。

2、Sorted-Set类型还可用于构建索引数据。

## keys的通用操作

* 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



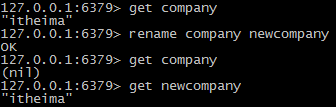
* del key1 key2…：删除指定的key



* 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



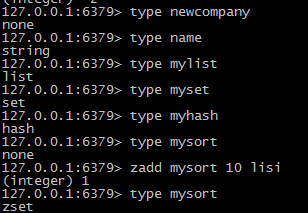
* expire key ：设置过期时间，单位：秒



* 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。



* 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。



# Jedis

## Jedis的概述

### Jedis的概述

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

## Jedis的入门

### 第一步：引入jedis的jar包



Maven依赖

<dependency>

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.8.0</version>

</dependency>

### 第二步：Jedis的入门



## Jedis的连接池的方式



# 列表页面的班级信息的显示

## 准备工作

### 创建表

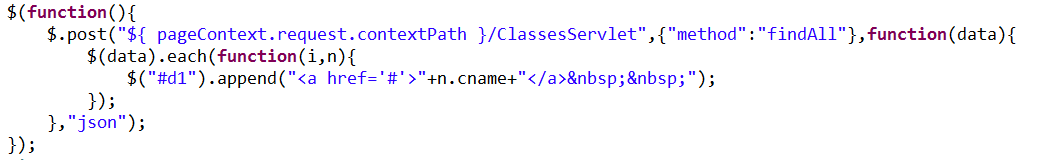
### 修改页面

### 创建相关的类

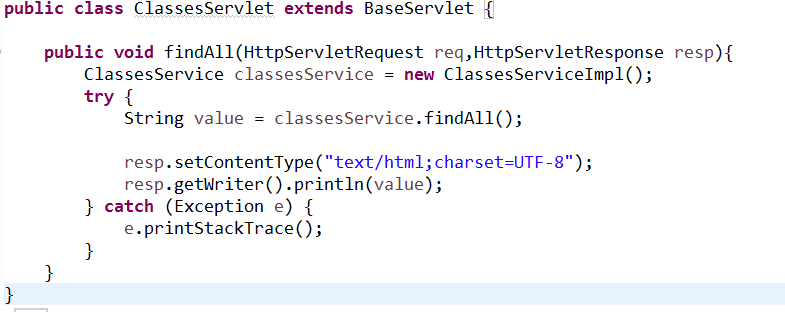
* 创建DAO、Service

## 代码实现

### 编写页面代码



### 编写Servlet



### 编写Service



### 编写DAO

