

## 第五章 Spring Boot 封装整合Redis 缓存实战

NoSQL(NoSQL = Not Only SQL)，意即“不仅仅是SQL”。NoSQL 数据库种类繁多(Redis、Mongodb、HBase、Elasticsearch 等等)，他们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。无形之间也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，尤其在大量下数据，更能体现它的优势。这得益于它的无关系性，数据库的结构简单。

### 1. 从零开始认识 redis

#### 1.1 redis 概述

Redis本质上是一个Key-Value类型的内存数据库，很像memcached，整个数据库在内存当中进行操作，定期通过异步操作把数据库数据写到硬盘上进行保存。因为是纯内存操作，Redis的性能非常出色，每秒可以处理超过 10万次读写操作，是已知性能最快的Key-Value DB。Redis的出色之处不仅仅是性能，Redis最大的魅力是支持保存多种数据结构，此外单个value的最大限制是1GB，不像memcached只能保存1MB的数据，因此Redis可以用来实现很多有用的功能。默认16个数据库，类似数组下标是从零开始，初始默认使用零号库，统一的密码管理，16个库都是同样密码，要么都连上要么一个也连接不上，redis默认端口是6379。

redis的优点：

- 速度快：因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)
- 持久化：定期通过异步操作把数据库数据写到硬盘上进行保存
- 支持丰富数据类型：支持string，list，set，sorted set，hash
- 支持事务：操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行
- 丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

#### 1.2 redis 安装

- 下载好软件，我这里是用绿色压缩包版

此电脑 > 新加卷 (H:) > Business > 课前软件软件 >

名称	修改日期	类型	大小
apache-activemq-5.9.0-bin	2017/6/22 11:08	360压缩 ZIP 文件	60,428 KB
apache-maven-3.3.9	2019/6/30 19:51	360压缩 ZIP 文件	8,424 KB
apache-tomcat-8.5.15-windows-x64	2017/5/25 16:05	360压缩 ZIP 文件	10,757 KB
ideaU-2017.1.3	2017/5/25 14:41	应用程序	510,526 KB
ideaU-2018.2.8	2019/7/21 17:13	应用程序	533,465 KB
idea注册码	2019/6/30 20:00	文本文档	1 KB
jdk_8.0.1310.11_64	2018/4/6 21:13	应用程序	202,784 KB
mysql-5.5.56-winx64	2017/6/17 11:42	Windows Install...	38,508 KB
Navicat_Premium_11.0.17	2015/1/4 11:01	应用程序	28,698 KB
Postman-win64-4.10.2-Setup	2017/8/13 12:50	应用程序	64,301 KB
RedisDesktopManager	2019/8/11 13:22	360压缩 ZIP 文件	15,619 KB
redis-latest-windows	2017/9/21 18:53	360压缩 ZIP 文件	5,756 KB
settings	2019/7/21 18:34	Executable Jar File	22 KB

- 解压到相应路径(这里我的路径是F:\ide\redis)

此电脑 > 新加卷 (F:) > ide >

名称	修改日期	类型	大小
12306Bypass	2018/1/30 21:33	文件夹	
android-studio	2017/4/21 20:06	文件夹	
apache-activemq-5.9.0	2017/6/22 11:23	文件夹	
data	2017/6/22 15:58	文件夹	
logs	2017/6/22 12:48	文件夹	
mongo_data	2017/7/8 17:08	文件夹	
mongodb	2017/7/8 15:20	文件夹	
postman-4.1.2	2018/5/4 20:46	文件夹	
redis	2019/6/17 22:51	文件夹	
..			

- 启动redis(进入redis找到redis-server.exe双击启动)

```
F:\ide\redis\redis-server.exe
[1308] 11 Aug 13:39:30.900 # Warning: no config file specified, using the default config. In order to specify a config file use F:\ide\redis\redis-server.exe /path/to/redis.conf

Redis 3.2.100 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 1308

http://redis.io

[1308] 11 Aug 13:39:30.906 # Server started, Redis version 3.2.100
[1308] 11 Aug 13:39:30.906 * DB loaded from disk: 0.000 seconds
[1308] 11 Aug 13:39:30.906 * The server is now ready to accept connections on port 6379
```

- 启动redis客户端(进入redis找到redis-cli.exe 双击启动)

```
选择F:\ide\redis\redis-cli.exe
127.0.0.1:6379>
```

- 测试

```
set name zhangsan
OK
get name
"zhangsan"
```

```
F:\ide\redis\redis-cli.exe
127.0.0.1:6379> set name zhangsan
OK
127.0.0.1:6379> get name
"zhangsan"
127.0.0.1:6379>
```

这样redis就安装成功了(windows)

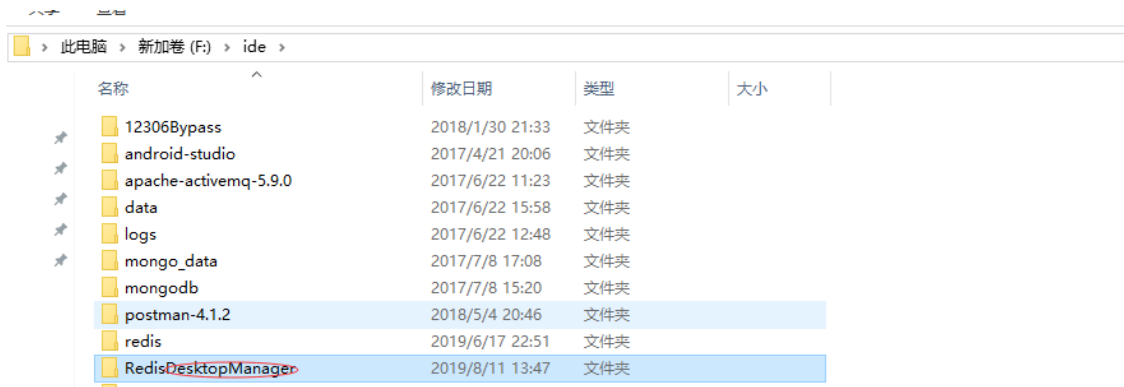
### 1.3 redis 图形管理工具的安装

官方出了RedisDesktopManager-0.9.8版本后要购买了。所以本次我们要使用的就是RedisDesktopManager-0.9.8版本

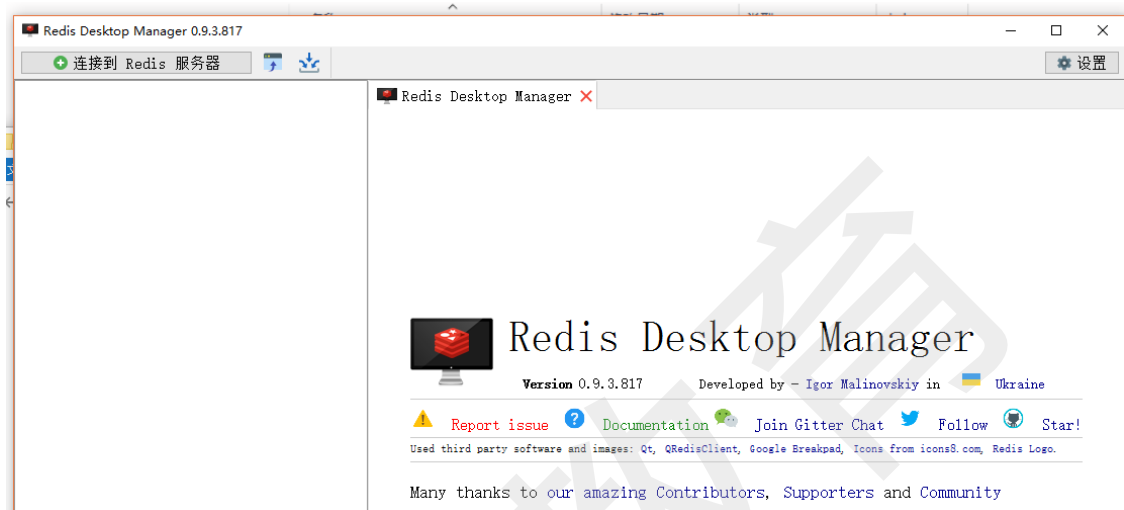
- 下载安装(这里我已经帮大家下载好了)

此电脑 > 新加卷 (H:) > Business > 课前软件软件 >				
名称	修改日期	类型	大小	
apache-activemq-5.9.0-bin	2017/6/22 11:08	360压缩 ZIP 文件	60,428 KB	
apache-maven-3.3.9	2019/6/30 19:51	360压缩 ZIP 文件	8,424 KB	
apache-tomcat-8.5.15-windows-x64	2017/5/25 16:05	360压缩 ZIP 文件	10,757 KB	
idealU-2017.1.3	2017/5/25 14:41	应用程序	510,526 KB	
idealU-2018.2.8	2019/7/21 17:13	应用程序	533,465 KB	
idea注册码	2019/6/30 20:00	文本文档	1 KB	
jdk_8.0.1310.11_64	2018/4/6 21:13	应用程序	202,784 KB	
mysql-5.5.56-winx64	2017/6/17 11:42	Windows Install...	38,508 KB	
Navicat_Premium_11.0.17	2015/1/4 11:01	应用程序	28,698 KB	
Postman-win64-4.10.2-Setup	2017/8/13 12:50	应用程序	64,301 KB	
RedisDesktopManager	2019/8/11 13:22	360压缩 ZIP 文件	15,619 KB	
redis-latest-windows	2017/9/21 18:53	360压缩 ZIP 文件	5,756 KB	
settings	2019/7/21 18:34	Executable Jar File	22 KB	

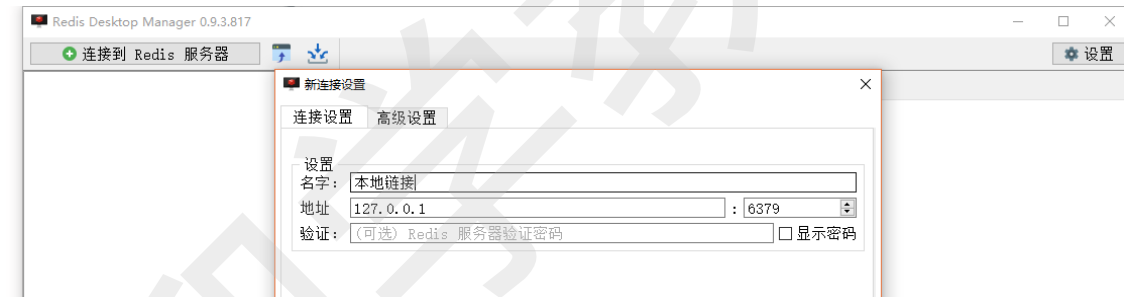
- 解压到相应的目录(这里我的路径是F:\ide\RedisDesktopManager)



- 进入到RedisDesktopManager找到rdm.exe 双击



- 创建链接



- 测试



## 2. redis 常用命令操作

### 2.1 redis的数据类型

redis存储的是：key,value格式的数据，其中key都是字符串，value有5种不同的数据结构

#### 2.1.1 字符串类型 string

- string是redis最基本的类型，你可以理解成与Memcached一模一样的类型，一个key对应一个value；
- string类型是二进制安全的。意思是redis的string可以存储任何数据。如图片或者序列化的对象；

#### 2.1.2 哈希类型 hash

- redis的hash 是一个键值对集合；
- redis hash是一个string类型的field和value的映射表，hash特别适用于存储对象；
- 类似Java里面的Map<String,Object>；

### 2.1.3 列表类型 list

- Redis的list是每个子元素都是String类型的双向链表，可以通过push和pop操作从列表的头部或者尾部添加或者删除元素，这样List即可以作为栈，也可以作为队列。使用List结构，我们可以轻松地实现最新消息排行等功能。

### 2.1.4 集合类型 set

- redis的set是string类型的无序集合。它是通过HashTable实现的。

### 2.1.5 有序集合类型 sortedset

- redis的sortedset和 set 一样也是string类型元素的集合；
- 不同的是每个元素都会关联一个double类型的分数；
- redis正是通过分数来为集合中的成员进行从小到大的排序。sortedset的成员是唯一的,但分数(score)却可以重复；

## 2.2 常用的指令

### 2.2.1 字符串类型 String

- 赋值命令：SET key value

```
F:\ide\redis\redis-cli.exe
127.0.0.1:6379> set name zhangsan
OK
127.0.0.1:6379> _
```

**解读：**如果key不存在创建该key并赋值，返回“OK”，如果该Key已经存在，则覆盖其原有值。返回“OK”。

- 取值命令：GET key

```
F:\ide\redis\redis-cli.exe
127.0.0.1:6379> set name zhangsan
OK
127.0.0.1:6379> get name
"zhangsan"
127.0.0.1:6379> get name1
(nil)
127.0.0.1:6379> _
```

**解读：**获取指定 Key 的 Value。如果 key 存在返回对应的 value，如果该 Key 不存在，返回 nil

- 数字递增：INCR key

```
127.0.0.1:6379> incr num
(integer) 1
127.0.0.1:6379> get num
"1"
127.0.0.1:6379> _
```

**解读：**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

- 数字递减：DECR key

```
127.0.0.1:6379> set name zhangsan
OK
127.0.0.1:6379> get name
"zhangsan"
127.0.0.1:6379> get name1
(nil)
127.0.0.1:6379> incr num
(integer) 1
127.0.0.1:6379> get num
"1"
127.0.0.1:6379> decr num
(integer) 0
127.0.0.1:6379> get num
"0"
127.0.0.1:6379> _
```

**解读：**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。操作返回递减后的 value 值。

- 增加指定的整数：**INCRBY** key increment

```
(integer) 0
127.0.0.1:6379> get num
"0"
127.0.0.1:6379> incrby num 10
(integer) 10
127.0.0.1:6379> get num
"10"
127.0.0.1:6379>
```

**解读：**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。操作成功则返回增加后的value值。

- 减少指定的整数：**DECRBY** key decrement

```
0
127.0.0.1:6379> incrby num 10
(integer) 10
127.0.0.1:6379> get num
"10"
127.0.0.1:6379> decrby num 5
(integer) 5
127.0.0.1:6379> get num
"5"
127.0.0.1:6379>
```

**解读：**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。操作成功则返回减少后的value值。

- 向尾部追加值：**APPEND** key value

```
0
127.0.0.1:6379> append name 123
(integer) 11
127.0.0.1:6379> get name
"zhangsan123"
127.0.0.1:6379> appeng salary 22000
(error) ERR unknown command 'appeng'
127.0.0.1:6379> append salary 22000
(integer) 5
127.0.0.1:6379> get salary
"22000"
127.0.0.1:6379>
```

**解读：**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。返回追加后Value的字符串长度。

## 2.2.2 哈希类型 hash

- 赋值命令：**HSET** key field value

```
22000
127.0.0.1:6379> hset userinfo username zhangsan
(integer) 1
127.0.0.1:6379> hset userinfo username lisi
(integer) 0
127.0.0.1:6379>
```

**解读：**为指定的 Key (对象/集合)添加属性并设置值，如果 Key 不存在，该命令将创建新 Key (对象/集合) 以用于存储参数中的属性对，如果参数中的 Field(属性) 在该 Key 中已经存在，则用新值覆盖其原有值。返回1表示新的 Field(属性) 被设置了新值，0表示Field已经存在，用新值覆盖原有值。

- 取值命令：**HGET** key field

```
127.0.0.1:6379> hset userinfo username zhangsan
(integer) 1
127.0.0.1:6379> hset userinfo username lisi
(integer) 0
127.0.0.1:6379> hget userinfo username
"lisi"
127.0.0.1:6379> hget userinfo age
(nil)
127.0.0.1:6379>
```

**解读：**返回指定 Key (对象/集合)中指定 Field(属性) 的关联值，如果参数中的 Key 或 Field 不存在，返回nil。

- 删除字段：**HDEL** key field [field ...]

```
(nil)
127.0.0.1:6379> hdel userinfo username
(integer) 1
127.0.0.1:6379> hdel userinfo1 username
(integer) 0
127.0.0.1:6379>
```

**解读：**从指定Key (对象/集合)的Hashes Value中删除参数中指定的多个字段，如果不存在字段将被忽略，返回实际删除的Field数量。如果Key不存在，则将其视为空Hashes，并返回0。

- 设置多个字段的值：**HMSET** key field value [field value ...]

```
(integer) 0
127.0.0.1:6379> hmset userinfo username zhangsan age 19 email 2639990486@qq.com
OK
127.0.0.1:6379> hkeys
(error) ERR wrong number of arguments for 'hkeys' command
127.0.0.1:6379> hkeys userinfo
1) "username"
2) "age"
3) "email"
127.0.0.1:6379> hvals userinfo
1) "zhangsan"
2) "19"
3) "2639990486@qq.com"
127.0.0.1:6379>
```

解读：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 (对象/集合)不存在，则创建新Key (对象/集合)，同时设定参数中的Field/Value。

- 获取多个字段的值：**HMGET** key field [field ...]

```
(integer) 0
127.0.0.1:6379> hmset userinfo username zhangsan age 19 email 2639990486@qq.com
OK
127.0.0.1:6379> hkeys
(error) ERR wrong number of arguments for 'hkeys' command
127.0.0.1:6379> hkeys userinfo
1) "username"
2) "age"
3) "email"
127.0.0.1:6379> hvals userinfo
1) "zhangsan"
2) "19"
3) "2639990486@qq.com"
127.0.0.1:6379> hmget userinfo username email
1) "zhangsan"
2) "2639990486@qq.com"
127.0.0.1:6379>
```

解读：获取和参数中指定 Fields 关联的一组 Values，其返回顺序等同于 Fields 的请求顺序。如果请求的 Field 不存在，其值返回null。

### 2.2.3 列表类型 list

- 向列表头部添加元素：**LPUSH** key value [value ...]

```
2639990486@qq.com
127.0.0.1:6379> lpush list 1 2 3 4 5
(integer) 5
127.0.0.1:6379>
```

解读：向列表左边添加元素。如果该 Key 不存在，创建一个与该 Key 关联的空链表，之后再数据从链表的头部插入。操作成功则返回插入后链表中元素的数量。

654321

- 向列表尾部添加元素：**RPUSh** key value [value ...]

```
2639990486@qq.com
127.0.0.1:6379> lpush list 1 2 3 4 5
(integer) 5
127.0.0.1:6379> rpush list 6 7
(integer) 7
127.0.0.1:6379>
```

解读：向列表右边添加元素。如果该 Key 不存在，创建一个与该 Key 关联的空链表，之后再数据从链表的尾部插入。操作成功则返回插入后链表中元素的数量。

65432178

- 获得列表：**LRANGE** key start stop

```
127.0.0.1:6379> lpush list 1 2 3 4 5
(integer) 5
127.0.0.1:6379> rpush list 6 7
(integer) 7
127.0.0.1:6379> lrange list 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
6) "6"
7) "7"
127.0.0.1:6379>
```

解读：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。返回指定范围内元素的列表。

- 列表进行修剪：**LTRIM** KEY\_NAME START STOP

```

127.0.0.1:6379> lrange list 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
6) "6"
7) "7"
127.0.0.1:6379> ltrim list 0 5
OK
127.0.0.1:6379> lrange list 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
6) "6"
127.0.0.1:6379>

```

解读：让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。下标 0 表示列表头部第一个元素，以 1 表示列表头部的第二个元素，以此类推。你也可以使用负数下标，以 -1 表示列表尾部的最后一个元素，-2 表示列表的倒数第二个元素，以此类推。

- 元素从左边出栈：LPOP key

```

127.0.0.1:6379> lrange list 0 -1
1) "5"
2) "4"
3) "3"
4) "2"
5) "1"
6) "6"
127.0.0.1:6379> lpop list
"5"
127.0.0.1:6379> lrange list 0 -1
1) "4"
2) "3"
3) "2"
4) "1"
5) "6"
127.0.0.1:6379>

```

解读：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。

- 元素从右边出栈：RPOP key

```

127.0.0.1:6379> lrange list 0 -1
1) "4"
2) "3"
3) "2"
4) "1"
5) "6"
127.0.0.1:6379> rpop list
"6"
127.0.0.1:6379> lrange list 0 -1
(error) ERR value is not an integer or out of range
127.0.0.1:6379> lrange list 0 -1
1) "4"
2) "3"
3) "2"
4) "1"
127.0.0.1:6379>

```

解读：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。RPOP 命令执行两步操作：第一步是将列表右边的元素从列表中移除，第二步是返回被移除的元素值。

- 获取列表中元素的个数：LLEN key

- (error) ERR value is not an integer or out of range
- ```

127.0.0.1:6379> lrange list 0 -1
1) "4"
2) "3"
3) "2"
4) "1"
127.0.0.1:6379> llen list
(integer) 4
127.0.0.1:6379>

```

**解读：**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。

- 获取指定索引的元素值：**LINDEX** key index

```
127.0.0.1:6379> lrange list 0 -1
1) "4"
2) "3"
3) "2"
4) "1"
127.0.0.1:6379> llen list
(integer) 4
127.0.0.1:6379> lindex list 3
"1"
127.0.0.1:6379> lindex list -1
"1"
127.0.0.1:6379> lindex list -2
"2"
127.0.0.1:6379>
```

**解读：**该命令将返回链表中指定位置(index)的元素，index 是0-based，表示从头部位置开始第 index 的元素，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。如果超出 index 返回这返回 nil。

#### 2.2.4 集合类型 set

- 增加元素：**SADD** key member [member ...]

```
127.0.0.1:6379> sadd myset 1 2 3
(integer) 3
127.0.0.1:6379>
```

**解读：**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。返回实际插入的成员数量。

- 获得集合中元素个数：**SCARD** key

```
127.0.0.1:6379> sadd myset 1 2 3
(integer) 3
127.0.0.1:6379> scard myset
(integer) 3
127.0.0.1:6379>
```

**解读：**返回Set中成员的数量，如果该Key并不存在，返回0。

- 判断元素是否在集合中：**SISMEMBER** key member

```
127.0.0.1:6379> sadd myset 1 2 3
(integer) 3
127.0.0.1:6379> scard myset
(integer) 3
127.0.0.1:6379> sismember myset 2
(integer) 1
127.0.0.1:6379> sismember myset 5
(integer) 0
127.0.0.1:6379>
```

**解读：**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1表示已经存在，0表示不存在，或该Key本身并不存在。

- 获得集合中的所有元素：**SMEMBERS** key

```
(integer) 0
127.0.0.1:6379> smembers key
(empty list or set)
127.0.0.1:6379> smembers myset
1) "1"
2) "2"
3) "3"
127.0.0.1:6379>
```

- **解读：**获取与该Key关联的Set中所有的成员。如果ket不存在返回空集合
- 从集合中弹出一个元素：**SPOP** key

```
127.0.0.1:6379> spop myset
"3"
127.0.0.1:6379> smembers myset
1) "1"
2) "2"
127.0.0.1:6379>
```

**解读：**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。返回移除的成员，如果该Key并不存在，则返回nil。

- 删除元素：**SREM** key member [member ...]



```

127.0.0.1:6379> srem myset 3
(integer) 0
127.0.0.1:6379> srem myset 1
(integer) 1
127.0.0.1:6379> smembers myset
1) "2"
127.0.0.1:6379>

```

**解读：**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。返回从Set中实际移除的成员数量，如果没有则返回0。

- 随机获得集合中的元素：**SRANDMEMBER** key [count]

```

127.0.0.1:6379> smembers myset
1) "1"
2) "2"
3) "3"
4) "4"
5) "5"
6) "6"
7) "7"
127.0.0.1:6379> srandmember myset
"7"
127.0.0.1:6379> srandmember myset
"6"
127.0.0.1:6379> srandmember myset
"1"
127.0.0.1:6379>

```

**解读：**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，如果|count|大于集合中的元素，就会返回全部元素不够的以重复元素补齐，如果key不存在则返回nil。

## 2.2.5 有序集合类型 sorted set

- 增加元素：**ZADD** key score member [score] [member]

```

F:\ide\redis\redis-cli.exe
127.0.0.1:6379> zadd zsort 1 a 2 b 3 c
(integer) 3
127.0.0.1:6379> zadd zsort a 2
(error) ERR value is not a valid float
127.0.0.1:6379> a

```

**解读：**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。添加成功返回实际插入的成员数量。

- 获得集合中元素个数：**ZCARD** key

```

27.0.0.1:6379> zadd zsort 1 a 2 b 3 c
(integer) 3
27.0.0.1:6379> zadd zsort a 2
error) ERR value is not a valid float
27.0.0.1:6379> zadd zsort
error) ERR wrong number of arguments for 'zadd' command
27.0.0.1:6379> zcard zsort
(integer) 3
27.0.0.1:6379> zcard sort
(integer) 0
27.0.0.1:6379>

```

**解读：**返回Sorted Set中的成员数量，如果该Key不存在，返回0。

- 获得指定分数范围内的元素个数：**ZCOUNT** key min max

```

127.0.0.1:6379> zadd zsort 1 a 2 b 3 c
(integer) 3
127.0.0.1:6379> zadd zsort a 2
(error) ERR value is not a valid float
127.0.0.1:6379> zadd zsort
(error) ERR wrong number of arguments for 'zadd' command
127.0.0.1:6379> zcard zsort
(integer) 3
127.0.0.1:6379> zcard sort
(integer) 0
127.0.0.1:6379> zcount zsort 1 3
(integer) 3
127.0.0.1:6379> zcount zsort 1 2
(integer) 2
127.0.0.1:6379> zcount zsort 1 10
(integer) 3
127.0.0.1:6379>

```

**解读：**该命令用于获取分数(score)在min和max之间的成员数量。(  $\text{min} \leq \text{score} \leq \text{max}$  ) 如果加上了“(”着表明是开区间例如 `zcount key (min max` 则表示  $(\text{min} < \text{score} \leq \text{max})$  同理 `zcount key min (max` 则表明  $(\text{min} \leq \text{score} < \text{max})$  返回指定返回数量。

- 获得元素的分数：**ZSCORE** key member

```

127.0.0.1:6379> zscore sort a
(nil)
127.0.0.1:6379> zscore zsort a
"1"
127.0.0.1:6379>

```

**解读：**如果该成员存在，以字符串的形式返回其分数，否则返回nil。

- 增加某个元素的分数：**ZINCRBY** key increment member

```

127.0.0.1:6379> zscore zsort a
"1"
127.0.0.1:6379> zincrby zsort 10 a
"11"
127.0.0.1:6379> zscore zsort a
"11"
127.0.0.1:6379>

```

**解读：**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。如果不报错则以串形式表示的新分数。

- 获得排名在某个范围的元素列表(分数从小到大)：**ZRANGE** key start stop [WITHSCORES]

```

127.0.0.1:6379> zrange zsort 0 -1
1) "b"
2) "c"
3) "a"
127.0.0.1:6379> zrange zsort 0 -2
1) "b"
2) "c"
127.0.0.1:6379> zrange zsort 2 1
(empty list or set)
127.0.0.1:6379> zrange zsort 0 10
1) "b"
2) "c"
3) "a"
127.0.0.1:6379>

```

**解读：**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  $\text{start} > \text{stop}$ ，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...`。

- 获得排名在某个范围的元素列表（元素分数从大到小排序）：**ZREVRANGE** key start stop [WITHSCORES]

```

127.0.0.1:6379> zrange myzset 1 10
1) "c"
2) "d"
3) "a"
4) "e"
127.0.0.1:6379> zrange myzset 1 10 withscores
1) "c"
2) "3"
3) "d"
4) "5"
5) "a"
6) "6"
7) "e"
8) "6"
127.0.0.1:6379> zrevrange myzset 1 10
1) "a"
2) "d"
3) "c"
4) "b"
127.0.0.1:6379> zrevrange myzset 1 10 withscores
1) "a"
2) "6"
3) "d"
4) "5"
5) "c"
6) "3"
7) "b"
8) "2"

```

解读：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。

- 获得指定分数范围的元素(分数从小到大)：**ZRANGEBYSCORE** key min max [WITHSCORES] [LIMIT offset count]

```

127.0.0.1:6379> zadd zsort 2 c 5 d 6 e
(integer) 2
127.0.0.1:6379> zrange zsort 0 -1 withscores
1) "b"
2) "2"
3) "c"
4) "2"
5) "d"
6) "5"
7) "e"
8) "6"
9) "a"
10) "11"
127.0.0.1:6379> zrangebyscore zsort 2 10
1) "b"
2) "c"
3) "d"
4) "e"
127.0.0.1:6379> zrangebyscore zsort 2 11
1) "b"
2) "c"
3) "d"
4) "e"
5) "a"
127.0.0.1:6379> zrangebyscore zsort (2 11
1) "d"
2) "e"
3) "a"
127.0.0.1:6379> zrangebyscore zsort 2 (11
1) "b"
2) "c"
3) "d"
4) "e"
127.0.0.1:6379>

```

解读：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 $\min \leq \text{score} \leq \max$ 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。

- 获得指定分数范围的元素（元素分数从大到小排序）：**ZREVRANGEBYSCORE** key max min [WITHSCORES] [LIMIT offset count]

```

127.0.0.1:6379> zrevrangebyscore zsort 11 2 withscores
1) "a"
2) "11"
3) "e"
4) "6"
5) "d"
6) "5"
7) "c"
8) "2"
9) "b"
10) "2"
127.0.0.1:6379>

```

解读：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。

- 获得元素的排名：**ZRANK** key member

```

27.0.0.1:6379> zrevrangebyscore zsort 11 2 withscores
1) "a"
2) "11"
3) "e"
4) "6"
5) "d"
6) "5"
7) "c"
8) "2"
9) "b"
10) "2"
27.0.0.1:6379> zrank zsort a
(integer) 4
27.0.0.1:6379> zrank zsort 9
nil)
27.0.0.1:6379>

```

解读：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。如果该成员存在，则返回它的位置索引值。否则返回nil。

- 删除一个或多个元素：ZREM key member [member ...]

```

127.0.0.1:6379> zrem zsort b
(integer) 1
127.0.0.1:6379> zrange zsort 0 -1
1) "c"
2) "d"
3) "e"
4) "a"
127.0.0.1:6379> zrange zsort 0 -1 withscores
1) "c"
2) "2"
3) "d"
4) "5"
5) "e"
6) "6"
7) "a"
8) "11"
127.0.0.1:6379>

```

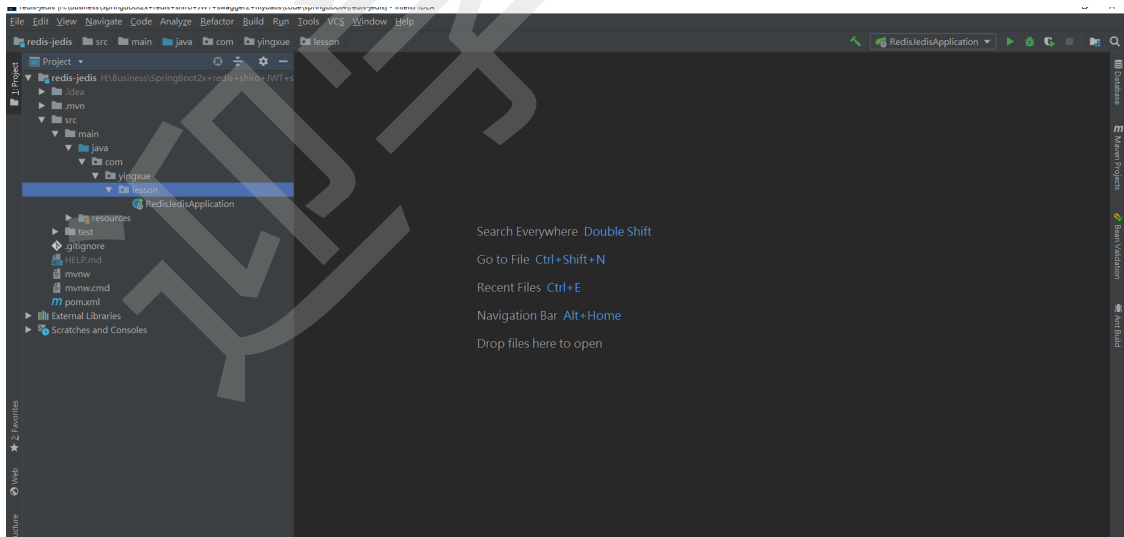
解读：该命令将移除参数中指定的成员，其中不存在的成员将被忽略。如果与该Key关联的Value不是Sorted Set，相应的错误信息将被返回。如果操作成功则返回实际被删除的成员数量。

### 3. SpringBoot+jedis 企业实战开发工具类封装

因为 Jedis 集成了 redis 的命令操作，所以 Jedis 是 Redis 官方推荐的面向 Java 的操作 Redis 的客户端。

#### 3.1 Spring Initializr 创建项目

1. 目录如下



pom.xml内容如下

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.6.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    
```

```

</parent>
<groupId>com.yingxue.lesson</groupId>
<artifactId>redis-jedis</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>redis-jedis</name>
<description>Demo project for Spring Boot</description>

<properties>
  <java.version>1.8</java.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
</project>

```

## 3.2 加入 redis 依赖

### 1. 加入redis pom依赖

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
  <groupId>redis.clients</groupId>
  <artifactId>jedis</artifactId>
</dependency>

```

## 3.3 jedisPool连接池管理 jedis

客户端连接 Redis 使用的是 TCP 协议，直连的方式每次需要建立 TCP 连接，而连接池的方式是可以预先初始化好 Jedis 连接，所以每次只需要从 Jedis 连接池借用即可，而借用和归还操作是在本地进行的，只有少量的并发同步开销，远远小于新建TCP连接的开销。另外直连的方式无法限制Jedis对象的个数，在极端情况下可能会造成连接泄露，而连接池的形式可以有效的保护和控制资源的使用。

### 1. 配置链接池属性

```

#jedisPool配置开始
# 连接池中的最大空闲连接
redis.maxIdle=30
# 连接池中的最小空闲连接
redis.minIdle=1
#连接池最大连接数（使用负值表示没有限制）
redis.maxTotal=100
# 连接池最大阻塞等待时间（使用负值表示没有限制）10秒
redis.maxwait=10000
# Redis服务器地址
redis.host=localhost
# Redis服务器连接端口
redis.port=6379
# Redis链接超时时间 10秒
redis.timeout=10000

```

## 2. 新建RedisConfig 配置链接词

```
package com.yingxue.lesson.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;

/**
 * @ClassName: RedisConfig
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Configuration
public class RedisConfig {
    @Value("${redis.host}")
    private String host;
    @Value("${redis.port}")
    private int port;
    @Value("${redis.maxTotal}")
    private int maxTotal;
    @Value("${redis.maxIdle}")
    private int maxIdle;
    @Value("${redis.minIdle}")
    private int minIdle;
    @Value("${redis.timeout}")
    private int timeout;
    @Value("${redis.maxWait}")
    private long maxWait;

    @Bean
    public JedisPool getJedisPool(){
        JedisPoolConfig jedisPoolConfig=new JedisPoolConfig();
        //连接池阻塞最大等待时间
        jedisPoolConfig.setMaxWaitMillis(maxWait);
        //连接池最大空闲连接数
        jedisPoolConfig.setMaxIdle(maxIdle);
        //连接池最小空闲连接数
        jedisPoolConfig.setMinIdle(minIdle);
        //连接池最大链接数
        jedisPoolConfig.setMaxTotal(maxTotal);
        JedisPool jedisPool=new JedisPool(jedisPoolConfig,host,port,timeout);
        return jedisPool;
    }
}
```

## 3.4 创建 RedisService

### 1. 创建RedisService类

```
package com.yingxue.lesson.service;

import org.springframework.stereotype.Service;

/**
 * @ClassName: RedisService
 * redis 封装工具类
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Service
public class RedisService {
}
```

### 3.5 jedis 企业开发工具类封装

Jedis实例都在JedisPool中，所以：

1. 获取Jedis实例需要从JedisPool中获取；
2. 用完Jedis实例需要还给JedisPool；
3. 如果Jedis在使用过程中出错，则也需要还给JedisPool；

修改RedisService 加入如下代码

```
package com.yingxue.lesson.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;

import java.util.List;
import java.util.Map;
import java.util.Set;

/**
 * @ClassName: RedisService
 * redis 封装工具类
 * @Author: 小霍
 * @CreateDate: 2019/11/13 19:36
 * @UpdateUser: 小霍
 * @UpdateDate: 2019/11/13 19:36
 * @Version: 0.0.1
 */
@Service
public class RedisService {
    @Autowired
    private JedisPool jedisPool;
    /**
     //*****对 key 常用操作*****
    */
    * 判断key是否存在
    * @Author: 小霍
    * @UpdateUser:
    * @Version: 0.0.1
    * @param key
    * @return boolean true 存在 false 不存在
    * @throws
    */
    public boolean exists(String key){
        Jedis jedis = null;
        boolean result;
        try {
            jedis=jedisPool.getResource();
            result=jedis.exists(key);
        } finally {
            if(jedis!=null){
                jedis.close();
            }
        }
        return result;
    }
    /**
    * 删除指定的key,也可以传入一个包含key的数组
    * @Author: 小霍
    * @UpdateUser:
    * @Version: 0.0.1
    * @param keys
    * @return java.lang.Long 返回删除成功的个数
    * @throws
    */
    public Long del(String... keys) {
        Jedis jedis = null;
        Long result;
        try {
            jedis = jedisPool.getResource();
            result= jedis.del(keys);
        } finally {
            if(jedis!=null){
```

```

        jedis.close();
    }
}
return result;
}

/**
 * KEYS pattern 通配符模式匹配
 * 查找所有符合给定模式 pattern 的 key 。
 * KEYS * 匹配数据库中所有 key 。
 * KEYS h?llo 匹配 hello , hallo 和 hxlllo 等。
 * KEYS h*llo 匹配 hlllo 和 heeeello 等。
 * KEYS 的速度非常快，但在一个大的数据库中使用它仍然可能造成性能问题，如果你需要从一个数据集中查找特定的 key ，
 * 你最好还是用 Redis 的集合结构(set)来代替。
 * @Author: 小霍
 * @UpdateUser:
 * @Version: 0.0.1
 * @param pattern
 * @return java.util.Set<java.lang.String>
 * @throws
 */
public Set<String> keys(String pattern) {
    Jedis jedis = null;
    Set<String> result ;
    try {
        jedis = jedisPool.getResource();
        result = jedis.keys(pattern);
    } finally {
        if(jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 设置过期时间
 * @Author: 小霍
 * @UpdateUser:
 * @Version: 0.0.1
 * @param key
 * • * @param seconds
 * @return Long 1: 表示设置成功, 0: 设置失败
 * @throws
 */
public Long expire(String key,int seconds){
    Jedis jedis=null;
    Long result=0L;
    try {
        jedis=jedisPool.getResource();
        if(seconds>0){
            result=jedis.expire(key,seconds);
        }
    } finally {
        if(jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 移除给定 key 的生存时间，将这个 key 从『易失的』（带生存时间 key ）转换成『持久的』（永不过期的 key ）
 * @Author: 小霍
 * @UpdateUser:
 * @Version: 0.0.1
 * @param key
 * @return java.lang.Long 当生存时间移除成功时，返回 1 ,如果 key 不存在或 key 没有设置生存时间，返回 0
 * @throws
 */
public Long persist(String key) {
    Jedis jedis = null;
    Long result;
    try {
        jedis = jedisPool.getResource();
        result=jedis.persist(key);
    }

```



```

    } finally {
        if(jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 以秒为单位，返回给定 key 的剩余生存时间
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * @return      java.lang.Long 当 key 不存在时，返回 -2 。当 key 存在但没有设置剩余生存时间时，返回 -1 。否则，以秒为单位，返回 key
 *              的剩余生存时间
 * @throws
 */
public Long ttl(String key) {
    Jedis jedis = null;
    Long result;
    try {
        jedis = jedisPool.getResource();
        result=jedis.ttl(key);
    } finally {
        if(jedis!=null){
            jedis.close();
        }
    }
    return result;
}

//*****String数据类型*****
/**
 * 获取指定Key的Value。如果与该Key关联的Value不是string类型，Redis将抛出异常，
 * 因为GET命令只能用于获取string value，如果该Key不存在，返回null
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * @return      成功返回value 失败返回null
 * @throws
 */
public String get(String key) {
    Jedis jedis = null;
    String value ;
    try {
        jedis = jedisPool.getResource();
        value = jedis.get(key);
    } finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return value;
}

/**
 * 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。总是返回"OK"。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * • * @param value
 * • * @param expire 过期时间秒
 * @return      void
 * @throws
 */
public String set(String key, String value,int expire) {

    Jedis jedis = null;
    String result;
    try {
        jedis = jedisPool.getResource();

```

```

        result=jedis.set(key,value);
        if(expire>0){
            jedis.expire(key, expire);
        }
    } finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 加锁操作: jedis.set(key,value,"NX","EX",timeOut)【保证加锁的原子操作】
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key key就是redis的key值作为锁的标识,
 * * @param value value在这里作为客户端的标识,
 * * @param nxxx NX: 只有这个key不存才的时候才会进行操作, if not exists;
 * * @param nxxx XX: 只有这个key存才的时候才会进行操作, if it already exist;
 * * @param expx EX: 设置key的过期时间为秒, 具体时间由第5个参数决定
 * * @param expx PX: 设置key的过期时间为毫秒, 具体时间由第5个参数决定
 * * @param time 通过timeOut设置过期时间保证不会出现死锁【避免死锁】
 * @return      java.lang.String 成功OK不成功null
 * @throws
 */

public String set(String key, String value, String nxxx, String expx, long time){
    Jedis jedis=null;
    String result;
    try {
        jedis=jedisPool.getResource();
        result = jedis.set(key, value, nxxx, expx, time);
    } finally {
        if(jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 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Hi, 该操作将执行失败并抛出相应的异常。
 * 注意: 该操作的取值范围是64位有符号整型; 返回递增后的value值。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * @return      java.lang.Long 加值后的结果
 * @throws
 */

public Long incr(String key) {
    Jedis jedis = null;
    Long result;
    try {
        jedis = jedisPool.getResource();
        result = jedis.incr(key);
    } finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 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decr之后其值为-1。
 * 如果value的值不能转换为整型值, 如Hi, 该操作将执行失败并抛出相应的异常。
 * 注意: 该操作的取值范围是64位有符号整型; 返回递减后的value值。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * @return      java.lang.Long 加值后的结果

```

```

    * @throws
    */
    public Long decr(String key) {
        Jedis jedis = null;
        Long result;
        try {
            jedis = jedisPool.getResource();
            result = jedis.decr(key);
        } finally {
            if (jedis!=null){
                jedis.close();
            }
        }
        return result;
    }
}

/**
 * *****hash数据类型*****
 */
/**
 * 通过key 和 field 获取指定的 value
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * * @param field
 * @return      java.lang.String
 * @throws
 */
public String hget(String key, String field) {
    Jedis jedis = null;
    String result ;
    try {
        jedis = jedisPool.getResource();
        result = jedis.hget(key, field);
    } finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 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。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * * @param field
 * * @param value
 * @return      java.lang.Long
 * @throws
 */
public Long hset(String key, String field, String value) {
    Jedis jedis = null;
    Long result;
    try {
        jedis = jedisPool.getResource();
        result = jedis.hset(key, field, value);
    } finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 判断指定Key中的指定Field是否存在，返回true表示存在，false表示参数中的Field或Key不存在。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key

```

```

    • * @param field
    * @return      java.lang.Boolean
    * @throws
    */
    public Boolean hexists(String key, String field) {

        Jedis jedis = null;
        Boolean result;
        try {
            jedis = jedisPool.getResource();
            result = jedis.hexists(key, field);
        } finally {
            if(jedis!=null){
                jedis.close();
            }
        }
        return result;
    }

}

/**
 * 从指定Key的Hashes Value中删除参数中指定的多个字段，如果不存在字段将被忽略，
 * 返回实际删除的Field数量。如果Key不存在，则将其视为空Hashes，并返回0。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
    • * @param fields
    * @return      java.lang.Long
    * @throws
    */
    public Long hdel(String key, String... fields) {
        Jedis jedis = null;
        Long result;
        try {
            jedis = jedisPool.getResource();
            result = jedis.hdel(key, fields);
        } finally {
            if(jedis!=null){
                jedis.close();
            }
        }
        return result;
    }

}

/**
 * 通过key获取所有的field和value
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
    * @return      java.util.Map<java.lang.String,java.lang.String>
    * @throws
    */
    public Map<String, String> hgetAll(String key) {
        Jedis jedis = null;
        Map<String, String> result;
        try {
            jedis = jedisPool.getResource();
            result = jedis.hgetAll(key);
        } finally {
            if (jedis!=null){
                jedis.close();
            }
        }
        return result;
    }

}

/**
 * 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。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1

```

```

    * @param key
    • * @param hash
    * @return      java.lang.String
    * @throws
    */
    public String hmset(String key, Map<String, String> hash) {

        Jedis jedis = null;
        String result;
        try {
            jedis = jedisPool.getResource();
            result = jedis.hmset(key, hash);
        } finally {
            if (jedis!=null){
                jedis.close();
            }
        }
        return result;
    }

    /**
     * 对应key的字段自增相应的值
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * • * @param field
     * • * @param increment
     * @return      java.lang.Long
     * @throws
     */
    public Long hincrBy(String key,String field,long increment){

        Jedis jedis=null;
        Long result;
        try {
            jedis=jedisPool.getResource();
            return jedis.hincrBy(key, field, increment);
        } finally {
            if(jedis!=null){
                jedis.close();
            }
        }
    }

    /**
     * *****List数据类型*****
     *
     * 向列表左边添加元素。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，该命令将会抛出相关异常。操作成功则返回插入后链表中元素的数量。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * • * @param strs 可以使一个string 也可以使string数组
     * @return      java.lang.Long 返回操作的value个数
     * @throws
     */
    public Long lpush(String key, String... strs) {

        Jedis jedis = null;
        Long result;
        try {
            jedis = jedisPool.getResource();
            result = jedis.lpush(key, strs);
        } finally {
            if (jedis!=null){
                jedis.close();
            }
        }
        return result;
    }

    /**
     * 向列表右边添加元素。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，该命令将会抛出相关异常。操作成功则返回插入后链表中元素的数量。
* @Author:      小霍
* @UpdateUser:
* @Version:     0.0.1
* @param key
* * @param strs 可以使一个string 也可以使string数组
* @return      java.lang.Long 返回操作的value个数
* @throws
*/
public Long rpush(String key, String... strs) {

    Jedis jedis = null;
    Long result;
    try {
        jedis = jedisPool.getResource();
        result = jedis.rpush(key, strs);
    }finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}
/**
* 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。
* @Author:      小霍
* @UpdateUser:
* @Version:     0.0.1
* @param key
* @return      java.lang.String
* @throws
*/
public String lpop(String key) {

    Jedis jedis = null;
    String result ;
    try {
        jedis = jedisPool.getResource();
        result = jedis.lpop(key);
    }finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}
/**
* 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。
* RPOP命令执行两步操作：第一步是将列表右边的元素从列表中移除，第二步是返回被移除的元素值。
* @Author:      小霍
* @UpdateUser:
* @Version:     0.0.1
* @param key
* @return      java.lang.String
* @throws
*/
public String rpop(String key) {

    Jedis jedis = null;
    String result ;
    try {
        jedis = jedisPool.getResource();
        result = jedis.rpop(key);
    } finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}
/**

```

```

*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。返回指定范围内元素的列表。
* @Author:      小霍
* @UpdateUser:
* @Version:     0.0.1
* @param key
* * @param start
* * @param end
* @return      java.util.List<java.lang.String>
* @throws
*/
public List<String> lrange(String key, long start, long end) {

    Jedis jedis = null;
    List<String> result;
    try {
        jedis = jedisPool.getResource();
        result = jedis.lrange(key, start, end);
    } finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
* 该命令将返回链表中指定位置(index)的元素，index是0-based，表示从头部位置开始第index的元素，
* 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。 如果超出index返回这返回nil。
* @Author:      小霍
* @UpdateUser:
* @Version:     0.0.1
* @param key
* * @param index
* @return      java.lang.String
* @throws
*/
public String lindex(String key, long index) {

    Jedis jedis = null;
    String result;
    try {
        jedis = jedisPool.getResource();
        result = jedis.lindex(key, index);
    } finally {
        if(jedis!=null){
            jedis.close();
        }
    }
    return result;
}

//*****Set数据类型*****
/**
* 如果在插入的过程中，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。返回实际插入的成员数量。
* @Author:      小霍
* @UpdateUser:
* @Version:     0.0.1
* @param key
* * @param members 可以是一个String 也可以是一个String数组
* @return      java.lang.Long 添加成功的个数
* @throws
*/
public Long sadd(String key, String... members) {

    Jedis jedis = null;
    Long result;

```

```

    try {
        jedis = jedisPool.getResource();
        result = jedis.sadd(key, members);
    } finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 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1表示已经存在，0表示不存在，或该key本身并不存在。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * * @param member
 * @return      java.lang.Boolean
 * @throws
 */
public Boolean sismember(String key, String member) {

    Jedis jedis = null;
    Boolean result;
    try {
        jedis = jedisPool.getResource();
        result = jedis.sismember(key, member);
    } finally {
        if(jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 通过key获取set中所有的value
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * @return      java.util.Set<java.lang.String>
 * @throws
 */
public Set<String> smembers(String key) {

    Jedis jedis = null;
    Set<String> result ;
    try {
        jedis = jedisPool.getResource();
        result = jedis.smembers(key);
    } finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}

//*****Sorted Set 数据类型*****
/**
 *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。添加成功返回实际插入的成员数量。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * * @param score

```



```

    • * @param member
    * @return      java.lang.Long
    * @throws
    */
    public Long zadd(String key, double score, String member) {

        Jedis jedis = null;
        Long result;
        try {
            jedis = jedisPool.getResource();
            result = jedis.zadd(key, score, member);
        } finally {
            if (jedis!=null){
                jedis.close();
            }
        }
        return result;
    }

}

/**
 * 返回Sorted Set中的成员数量，如果该Key不存在，返回0。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * @return      java.lang.Long
 * @throws
 */
public Long zcard(String key) {

    Jedis jedis = null;
    Long result;
    try {
        jedis = jedisPool.getResource();
        result = jedis.zcard(key);
    } finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 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。如果不报错则以串形式表示的新分数。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
    • * @param score
    • * @param member
 * @return      java.lang.Double
 * @throws
 */
public Double zincrby(String key, double score, String member) {

    Jedis jedis = null;
    Double result;
    try {
        jedis = jedisPool.getResource();
        result = jedis.zincrby(key, score, member);
    } finally {
        if(jedis!=null){
            jedis.close();
        }
    }
    return result;
}

}

/**

```

```

    * 如果该成员存在，以字符串的形式返回其分数，否则返回null
    * @Author:      小霍
    * @UpdateUser:
    * @Version:     0.0.1
    * @param key
    • * @param member
    * @return       java.lang.Double
    * @throws
    */
public Double zscore(String key, String member) {
    Jedis jedis = null;
    Double result;
    try {
        jedis = jedisPool.getResource();
        result = jedis.zscore(key, member);
    } finally {
        if(jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 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...。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
    • * @param min
    • * @param max
    * @return       java.util.Set<java.lang.String> 指定区间内的有序集成员的列表。
    * @throws
    */
public Set<String> zrange(String key, long start, long stop) {
    Jedis jedis = null;
    Set<String> result;
    try {
        jedis = jedisPool.getResource();
        result= jedis.zrange(key, start, stop);
    } finally {
        if (jedis!=null){
            jedis.close();
        }
    }
    return result;
}

/**
 * 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。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
    • * @param start
    • * @param end
    * @return       java.util.Set<java.lang.String>
    * @throws
    */
public Set<String> zrevrange(String key, long start, long end) {
    Jedis jedis = null;
    Set<String> result;
    try {
        jedis = jedisPool.getResource();
        result = jedis.zrevrange(key, start, end);
    }finally {
        if(jedis!=null){
            jedis.close();
        }
    }
}

```

```

        return result;
    }

    /**
     * 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。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * * @param max
     * * @param min
     * @return      java.util.Set<java.lang.String>
     * @throws
     */
    public Set<String> zrevrangeByScore(String key, double max, double min) {
        Jedis jedis = null;
        Set<String> result ;
        try {
            jedis = jedisPool.getResource();
            result = jedis.zrevrangeByScore(key, max, min);
        } finally {
            if(jedis!=null){
                jedis.close();
            }
        }
        return result;
    }
}

```

## 4. Spring Boot+RedisTemplate工具类封装自定义序列化方式

### 4.1概述

#### spring-boot-starter-data-redis

Spring Boot 提供了对 Redis 集成的组件包：spring-boot-starter-data-redis，它依赖于 spring-data-redis 和lettuce。Spring Boot 1.0 默认使用的是 Jedis 客户端，2.0 替换成了 Lettuce，但如果你从 Spring Boot 1.5.X切换过来，几乎感受不大差异，这是因为 spring-boot-starter-data-redis 为我们隔离了其中的差异性。

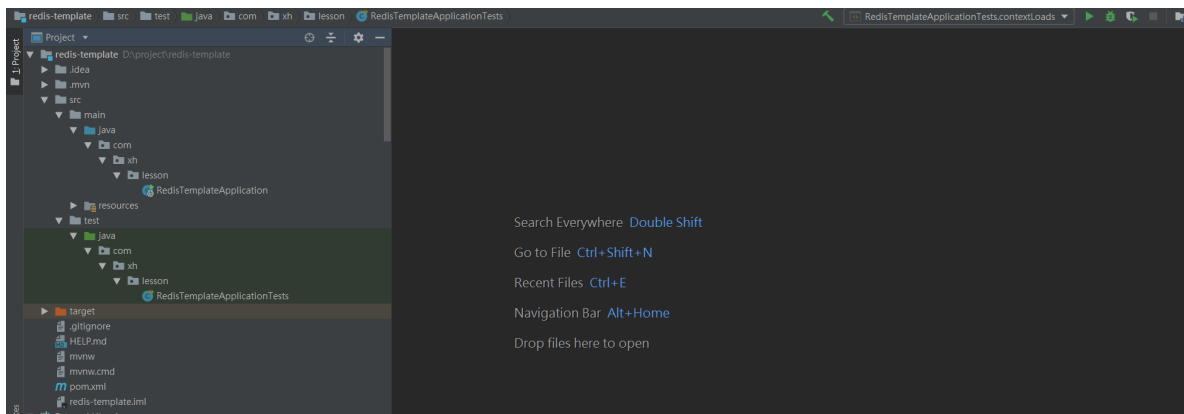
- Lettuce：是一个可伸缩线程安全的 Redis 客户端，多个线程可以共享同一个 RedisConnection，它利用优秀 Netty NIO 框架来高效地管理多个连接。
- Spring Data：是 Spring 框架中的一个主要项目，目的是为了简化构建基于 Spring 框架应用的数据访问，包括非关系数据库、Map-Reduce 框架、云数据服务等，另外也包含对关系数据库的访问支持。
- Spring Data Redis：是 Spring Data 项目中的一个主要模块，提供了一个高度封装的“**RedisTemplate**”类实现了对 Redis 客户端 API 的高度封装，使得对 Redis 的操作更加便捷而且**Spring data redis** 的连接池可以自动管理，针对数据的“序列化/反序列化”，提供了多种可选择策略(RedisSerializer)等等....

**RedisTemplate**中定义了对应redis 5种数据结构操作

- opsForValue();//操作字符串
- opsForHash();//操作hash
- opsForList();//操作list
- opsForSet();//操作set
- opsForZSet();//操作有序set

### 4.2 Spring Initializr 创建项目

目录如下图



### 4.3 引入 redis 依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-pool2</artifactId>
</dependency>
```

引入 commons-pool2是因为Lettuce需要使用 commons-pool2 创建连接池

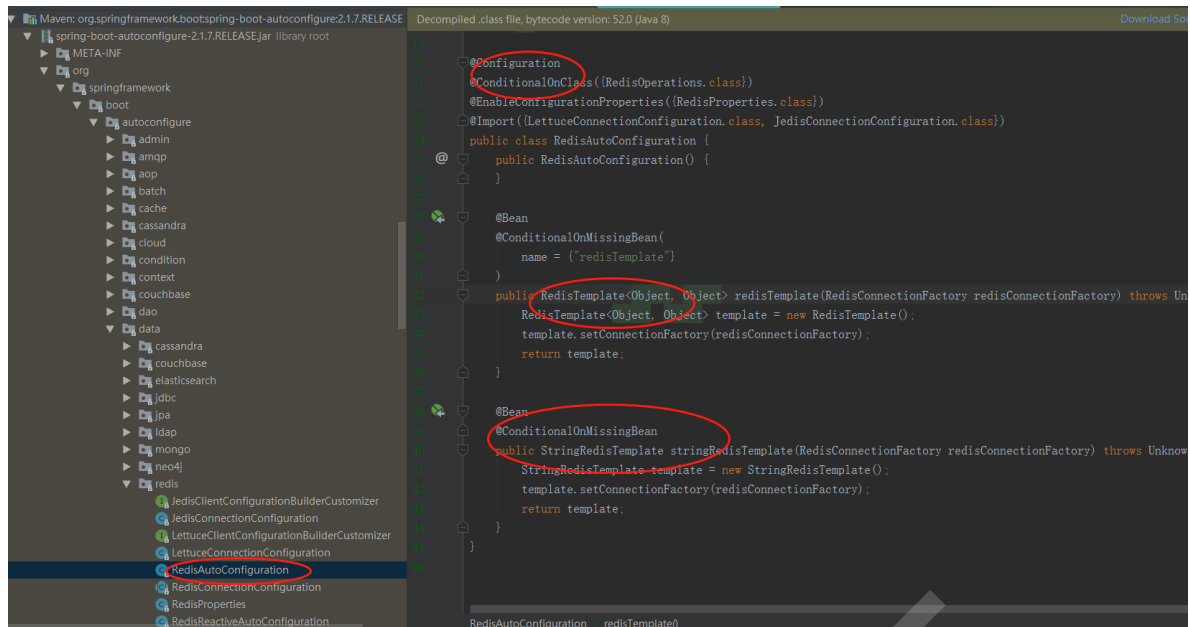
### 4.4 加入配置信息

```
# Redis 服务器地址
spring.redis.host=localhost
# Redis 服务器连接端口
spring.redis.port=6379
# 连接池最大连接数（使用负值表示没有限制） 默认 8
spring.redis.lettuce.pool.max-active=100
# 连接池最大阻塞等待时间（使用负值表示没有限制） 默认 -1
spring.redis.lettuce.pool.max-wait=PT10S
# 连接池中的最大空闲连接 默认 8
spring.redis.lettuce.pool.max-idle=30
# 连接池中的最小空闲连接 默认 0
spring.redis.lettuce.pool.min-idle=1
#链接超时时间
spring.redis.timeout=PT10S
```

在平常的开发中我们可以中通过spring的注入注解获取了RedisTemplate对象的引用。

```
@Autowired
private RedisTemplate<String,Object> redisTemplate;
或者
@Resource
private RedisTemplate<String,Object> redisTemplate;
```

为什么我们都没有像spring注入RedisTemplate对象为何就能直接声明获取引用呢？我们来看一下Spring Boot的源码



```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by Fernflower decompiler)
//

package org.springframework.boot.autoconfigure.data.redis;

import java.net.UnknownHostException;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisOperations;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.StringRedisTemplate;

@Configuration
@ConditionalOnClass({RedisOperations.class})
@EnableConfigurationProperties({RedisProperties.class})
@Import({LettuceConnectionConfiguration.class, JedisConnectionConfiguration.class})
public class RedisAutoConfiguration {
    public RedisAutoConfiguration() {
    }

    @Bean
    @ConditionalOnMissingBean(
        name = {"redisTemplate"}
    )
    public RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory)
    throws UnknownHostException {
        RedisTemplate<Object, Object> template = new RedisTemplate();
        template.setConnectionFactory(redisConnectionFactory);
        return template;
    }

    @Bean
    @ConditionalOnMissingBean
    public StringRedisTemplate stringRedisTemplate(RedisConnectionFactory redisConnectionFactory)
    throws UnknownHostException {
        StringRedisTemplate template = new StringRedisTemplate();
        template.setConnectionFactory(redisConnectionFactory);
        return template;
    }
}
```

通过源码可知在spring boot启动后会像spring 注入两个bean RedisTemplate、StringRedisTemplate 那么他们都有什么不一样呢？

我们来看看StringRedisTemplate和RedisTemplate源码

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by Fernflower decompiler)
//
.....
public class StringRedisTemplate extends RedisTemplate<String, String> {
    public StringRedisTemplate() {
        RedisSerializer<String> stringSerializer = new StringRedisSerializer();
        this.setKeySerializer(stringSerializer);
        this.setValueSerializer(stringSerializer);
        this.setHashKeySerializer(stringSerializer);
        this.setHashValueSerializer(stringSerializer);
    }

    .....
}

//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by Fernflower decompiler)
//

.....
public class RedisTemplate<K, V> extends RedisAccessor implements RedisOperations<K, V>,
BeanClassLoaderAware {
    ....

    public RedisTemplate() {
    }

    public void afterPropertiesSet() {
        super.afterPropertiesSet();
        boolean defaultUsed = false;
        if (this.defaultSerializer == null) {
            this.defaultSerializer = new JdkSerializationRedisSerializer(this.classLoader != null ?
this.classLoader : this.getClass().getClassLoader());
        }

        if (this.enableDefaultSerializer) {
            if (this.keySerializer == null) {
                this.keySerializer = this.defaultSerializer;
                defaultUsed = true;
            }

            if (this.valueSerializer == null) {
                this.valueSerializer = this.defaultSerializer;
                defaultUsed = true;
            }

            if (this.hashKeySerializer == null) {
                this.hashKeySerializer = this.defaultSerializer;
                defaultUsed = true;
            }

            if (this.hashValueSerializer == null) {
                this.hashValueSerializer = this.defaultSerializer;
                defaultUsed = true;
            }
        }

        if (this.enableDefaultSerializer && defaultUsed) {
            Assert.notNull(this.defaultSerializer, "default serializer null and not all serializers
initialized");
        }

        if (this.scriptExecutor == null) {
            this.scriptExecutor = new DefaultScriptExecutor(this);
        }
    }
}
```

```

        this.initialized = true;
    }

    .....

}

    如果没有特殊的设置, key 和 value 都是使用 defaultSerializer = new JdkSerializationRedisSerializer();
    进行序列化的。
    //这个是对key的默认序列化器。默认值是JdkSerializationRedisSerializer。
    redisTemplate.setKeySerializer();
    //这个是对value的默认序列化器, 默认值是JdkSerializationRedisSerializer。
    redisTemplate.setValueSerializer();
    //对hash结构数据的hashkey序列化器, 默认值是JdkSerializationRedisSerializer。
    redisTemplate.setHashKeySerializer();
    //对hash结构数据的hashvalue序列化器, JdkSerializationRedisSerializer。
    redisTemplate.setHashValueSerializer();

```

从上述源码可以看出他们的区别

- 两者的关系是StringRedisTemplate继承RedisTemplate。
- RedisTemplate是一个泛型类, 而StringRedisTemplate则不是。
- StringRedisTemplate只能对key=String, value=String的键值对进行操作, RedisTemplate可以对任何类型的key-value键值对操作。
- 他们各自序列化的方式不同, 但最终都是得到了一个字节数组, 殊途同归, StringRedisTemplate使用的是StringRedisSerializer类; RedisTemplate使用的是JdkSerializationRedisSerializer类。反序列化, 则是一个得到String, 一个得到Object

另外针对数据的“序列化/反序列化”, 提供了多种可选择策略(RedisSerializer)

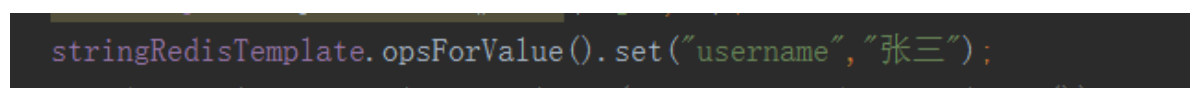
- o JdkSerializationRedisSerializer: 这个序列化方法就是jdk提供的了。首先要求我们要被序列化的类继承自Serializeable接口, 然后通过, 然后通过jdk对象序列化的方法保存。(注: 这个序列化保存的对象, 即使是个String类型的, 在redis控制台, 也是看不出来的, 因为它保存了一些对象的类型什么的额外信息)。是目前最常用的序列化策略。
- o StringRedisSerializer: 就是通过String.getBytes()来实现的。而且在Redis中, 所有存储的值都是字符串类型的。所以这种方法保存后, 通过Redis-cli控制台, 是可以清楚的查看到我们保存了什么key,value是什么。是最轻量级和高效的策略。
- o JacksonJsonRedisSerializer: jackson-json工具提供了java bean与json之间的转换能力, 可以将pojo实例序列化成json格式存储在redis中, 也可以将json格式的数据转换成pojo实例。因为jackson工具在序列化和反序列化时, 需要明确指定Class类型, 因此策略封装起来稍微复杂。

RedisTemplate在操作数据的时候, 存入数据会将数据先序列化成字节数组然后在存入Redis数据库(默认JdkSerializationRedisSerializer:这个序列化方法就是jdk提供的了, 首先要求我们要被序列化的类继承自Serializeable接口, 然后通过jdk对象序列化的方法保存), 这个时候打开Redis查看的时候, 你会看到你的数据不是以可读的形式, 展现的, 而是以字节数组显示, 如下图

(注: 这个序列化保存的对象, 即使是个String类型的, 在redis控制台, 也是看不出来的, 因为它保存了一些对象的类型什么的额外信息)



而StringRedisSerializer在操作数据的时候就是通过String.getBytes()来实现的。而且在Redis中, 所有存储的值都是字符串类型的。所以这种方法保存后, 通过Redis-cli控制台或者Redis-Desktop-Manager图形化工具, 是可以清楚的查看到我们保存了什么key,value是什么。如下图。





这样就会出现这个问题，如当我们从以前的项目升级为RedisTemplate在不指定序列化方式的时候取不到原来的值。

```
@Autowired
private RedisTemplate redisTemplate;

@Test
public void testRedisTemplate() {
    System.out.println(redisTemplate.opsForValue().get("username"));
}

输出: null
```

那是为什么呢？

因为我们在没有升级前使用的最轻量级和高效的策略（StringRedisSerializer），而升级成RedisTemplate他的默认使用序列化却是JdkSerializationRedisSerializer所以当需要获取数据的时候就找不到数据了(因为拿key去匹配的时候两种序列化得到的byte数组是不一样的)。

接下来我们把RedisTemplate序列化方式设置成StringRedisSerializer看看效果

```
@Autowired
@Test
public void testRedisTemplate() {
    stringRedisTemplate.opsForValue().set("username", "张三");
    StringRedisSerializer stringRedisSerializer=new StringRedisSerializer();
    redisTemplate.setKeySerializer(stringRedisSerializer);
    redisTemplate.setValueSerializer(stringRedisSerializer);
    System.out.println(redisTemplate.opsForValue().get("username"));
}

输出: 张三
```

当我们指定使用StringRedisSerializer做value的序列化时，StringRedisSerializer的泛型指定的是String，传其他对象就会报类不能转换为String异常

```
java.lang.ClassCastException: com.xh.lesson.domain.UserInfo cannot be cast to java.lang.String

    at org.springframework.data.redis.serializer.StringRedisSerializer.serialize(StringRedisSerializer.java:36)
    at org.springframework.data.redis.core.AbstractOperations.rawValue(AbstractOperations.java:126)
    at org.springframework.data.redis.core.DefaultValueOperations.set(DefaultValueOperations.java:235)
```

我们来看看StringRedisSerializer源码

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by Fernflower decompiler)
//

package org.springframework.data.redis.serializer;

import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import org.springframework.lang.Nullable;
import org.springframework.util.Assert;

public class StringRedisSerializer implements RedisSerializer<String> {
    private final Charset charset;
    public static final StringRedisSerializer US_ASCII;
    public static final StringRedisSerializer ISO_8859_1;
```



```

public static final StringRedisSerializer UTF_8;

public StringRedisSerializer() {
    this(StandardCharsets.UTF_8);
}

public StringRedisSerializer(Charset charset) {
    Assert.notNull(charset, "Charset must not be null!");
    this.charset = charset;
}

public String deserialize(@Nullable byte[] bytes) {
    return bytes == null ? null : new String(bytes, this.charset);
}

public byte[] serialize(@Nullable String string) {
    return string == null ? null : string.getBytes(this.charset);
}

static {
    US_ASCII = new StringRedisSerializer(StandardCharsets.US_ASCII);
    ISO_8859_1 = new StringRedisSerializer(StandardCharsets.ISO_8859_1);
    UTF_8 = new StringRedisSerializer(StandardCharsets.UTF_8);
}
}

```

所以我们要自定义StringRedisSerializer让他可以接受Object，且不会出现转换异常

#### 4.5 自定义 redis 序列化工具类

```

package com.yingxue.lesson.serializer;

import com.alibaba.fastjson.JSON;
import org.springframework.data.redis.serializer.RedisSerializer;
import org.springframework.util.Assert;

import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;

/**
 * @ClassName: MyStringRedisSerializer
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class MyStringRedisSerializer implements RedisSerializer<Object> {
    private final Charset charset;

    public MyStringRedisSerializer() {
        this(StandardCharsets.UTF_8);
    }

    public MyStringRedisSerializer(Charset charset) {
        Assert.notNull(charset, "Charset must not be null!");
        this.charset = charset;
    }

    @Override
    public String deserialize(byte[] bytes) {
        return (bytes == null ? null : new String(bytes, charset));
    }

    @Override
    public byte[] serialize(Object object) {
        if (object == null) {
            return new byte[0];
        }
        if (object instanceof String) {
            return object.toString().getBytes(charset);
        } else {
            String string = JSON.toJSONString(object);

```

```

        return string.getBytes(charset);
    }

}

}

```

#### 4.6 加入 fastJson 依赖

```

<!--fastJson-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.49</version>
</dependency>

```

#### 4.7 创建 RedisService 类

```

package com.yingxue.lesson.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;

/**
 * @ClassName: RedisService
 * TODO:类文件简单描述
 * @Author: 小霍
 * @CreateDate: 2019/10/19 21:51
 * @UpdateUser: 小霍
 * @UpdateDate: 2019/10/19 21:51
 * @Version: 0.0.1
 */
public class RedisService {
    @Autowired
    private RedisTemplate<String, Object> redisTemplate;
}

```

#### 4.8 创建 RedisConfig 配置类

```

package com.yingxue.lesson.config;

import com.yingxue.lesson.serializer.MyStringRedisSerializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.StringRedisSerializer;

import java.net.UnknownHostException;

/**
 * @ClassName: RedisConfig
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Configuration
public class RedisConfig {

    @Bean

```

```

    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory)
    throws UnknownHostException {
        RedisTemplate<String, Object> template = new RedisTemplate();
        template.setConnectionFactory(redisConnectionFactory);
        StringRedisSerializer stringRedisSerializer=new StringRedisSerializer();
        MyStringRedisSerializer myStringRedisSerializer=new MyStringRedisSerializer();
        template.setKeySerializer(stringRedisSerializer);
        template.setValueSerializer(myStringRedisSerializer);
        template.setHashKeySerializer(stringRedisSerializer);
        template.setHashValueSerializer(myStringRedisSerializer);
        return template;
    }
}

```

#### 4.9 创建自定义运行时异常

```

package com.yingxue.lesson.exception;

/**
 * @ClassName: BusinessException
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class BusinessException extends RuntimeException{
    /**
     * 异常编号
     */
    private final int messageCode;

    /**
     * 对messageCode 异常信息进行补充说明
     */
    private final String detailMessage;

    public BusinessException(int messageCode, String message) {
        super(message);
        this.messageCode = messageCode;
        this.detailMessage = message;
    }

    public int getMessageCode() {
        return messageCode;
    }

    public String getDetailMessage() {
        return detailMessage;
    }
}

```

#### 4.10 RedisTemplate工具类封装自定义序列化方式

修改RedisService 加入如下代码

```

package com.yingxue.lesson.service;

import com.yingxue.lesson.exception.BusinessException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

import java.util.*;
import java.util.concurrent.TimeUnit;

/**
 * @ClassName: RedisService
 * TODO:类文件简单描述
 */

```

```

* @Author: 小霍
* @UpdateUser: 小霍
* @Version: 0.0.1
*/
@Service
public class RedisService {
    @Autowired
    private RedisTemplate<String, Object> redisTemplate;
    /** -----key相关操作----- */

    /**
     * 是否存在key
     * @Author: 小霍
     * @UpdateUser:
     * @Version: 0.0.1
     * @param key
     * @return java.lang.Boolean
     * @throws
     */
    public Boolean hasKey(String key) {
        if (null==key){
            return false;
        }
        return redisTemplate.hasKey(key);
    }

    /**
     * 删除key
     * @Author: 小霍
     * @UpdateUser:
     * @Version: 0.0.1
     * @param key
     * @return Boolean 成功返回true 失败返回false
     * @throws
     */
    public Boolean delete(String key) {
        if (null==key){
            return false;
        }
        return redisTemplate.delete(key);
    }

    /**
     * 批量删除key
     * @Author: 小霍
     * @CreateDate: 2019/8/27 20:27
     * @UpdateUser:
     * @UpdateDate: 2019/8/27 20:27
     * @Version: 0.0.1
     * @param keys
     * @return Long 返回成功删除key的数量
     * @throws
     */
    public Long delete(Collection<String> keys) {
        return redisTemplate.delete(keys);
    }

    /**
     * 设置过期时间
     * @Author: 小霍
     * @UpdateUser:
     * @Version: 0.0.1
     * @param key
     * * @param timeout
     * * @param unit
     * @return java.lang.Boolean
     * @throws
     */
    public Boolean expire(String key, long timeout, TimeUnit unit) {
        if (null==key||null==unit){
            return false;
        }
        return redisTemplate.expire(key, timeout, unit);
    }

```

```

}

/**
 * 查找匹配的key
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param pattern
 * @return        java.util.Set<java.lang.String>
 * @throws
 */
public Set<String> keys(String pattern) {
    if (null==pattern){
        return null;
    }
    return redisTemplate.keys(pattern);
}

/**
 * 移除 key 的过期时间, key 将持久保持
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * @return        java.lang.Boolean
 * @throws
 */
public Boolean persist(String key) {
    if (null==key){
        return false;
    }
    return redisTemplate.persist(key);
}

/**
 * 返回 key 的剩余的过期时间
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * • * @param unit
 * * @return        java.lang.Long 当 key 不存在时, 返回 -2 。当 key 存在但没有设置剩余生存时间时, 返回 -1 。否则, 以秒为单位, 返回 key的剩余生存时间
 * @throws
 */
public Long getExpire(String key, TimeUnit unit) {
    if(null==key||null==unit){
        throw new BusinessException(4001004,"key or TomeUnit 不能为空");
    }
    return redisTemplate.getExpire(key, unit);
}

//*****String相关数据类型*****
/**
 * 设置指定 key 的值
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * • * @param value
 * * @return        void
 * @throws
 */
public void set(String key, Object value) {

    if(null==key||null==value){
        return;
    }
    redisTemplate.opsForValue().set(key, value);
}

/**
 * 设置key 的值 并设置过期时间
 * @Author:      小霍

```

```

    * @UpdateUser:
    * @Version:    0.0.1
    * @param key
    * * @param value
    * * @param time
    * * @param unit
    * @return      void
    * @throws
    */
    public void set(String key, Object value, long time, TimeUnit unit){

        if(null==key||null==value||null==unit){
            return;
        }
        redisTemplate.opsForValue().set(key,value,time,unit);
    }
    /**
    * 设置key 的值 并设置过期时间
    * key存在 不做操作返回false
    * key不存在设置值返回true
    * @Author:      小霍
    * @UpdateUser:
    * @Version:    0.0.1
    * @param key
    * * @param value
    * * @param time
    * * @param unit
    * @return      java.lang.Boolean
    * @throws
    */
    public Boolean setifAbsen(String key, Object value, long time, TimeUnit unit){

        if(null==key||null==value||null==unit){
            throw new BusinessException(4001004, "kkey、value、unit都不能为空");
        }
        return redisTemplate.opsForValue().setIfAbsent(key,value,time,unit);
    }
    /**
    * 获取指定Key的Value。如果与该Key关联的Value不是string类型，Redis将抛出异常，
    * 因为GET命令只能用于获取string value，如果该Key不存在，返回null
    * @Author:      小霍
    * @UpdateUser:
    * @Version:    0.0.1
    * @param key
    * @return      java.lang.Object
    * @throws
    */
    public Object get(String key){

        if(null==key){
            return null;
        }
        return redisTemplate.opsForValue().get(key);
    }
    /**
    * 很明显先get再set就说先获取key值对应的value然后再set 新的value 值。
    * @Author:      小霍
    * @UpdateUser:
    * @Version:    0.0.1
    * @param key
    * * @param value
    * @return      java.lang.Object
    * @throws
    */
    public Object getSet(String key, Object value){

        if(null==key){
            return null;
        }
        return redisTemplate.opsForValue().getAndSet(key,value);
    }
    /**
    * 通过批量的key获取批量的value
    * @Author:      小霍

```

```

* @UpdateUser:
* @Version:    0.0.1
* @param keys
* @return      java.util.List<java.lang.Object>
* @throws
*/
public List<Object> mget(Collection<String> keys){

    if(null==keys){
        return Collections.emptyList();
    }
    return redisTemplate.opsForValue().multiGet(keys);
}

/**
 * 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Hi，该操作将执行失败并抛出相应异常。操作成功则返回增加后的value值。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:    0.0.1
 * @param key
 * * @param increment
 * @return      long
 * @throws
 */
public long incrby(String key,long increment){
    if(null==key){
        throw new BusinessException(4001004,"key不能为空");
    }
    return redisTemplate.opsForValue().increment(key,increment);
}

/**
 *
 * 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Hi，该操作将执行失败并抛出相应的异常。操作成功则返回减少后的value值。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:    0.0.1
 * @param key
 * * @param decrement
 * @return      java.lang.Long
 * @throws
 */
public Long decrby(String key,long decrement){
    if(null==key){
        throw new BusinessException(4001004,"key不能为空");
    }
    return redisTemplate.opsForValue().decrement(key,decrement);
}

/**
 *
 * 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。返回追加后Value的字符串长度。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:    0.0.1
 * @param key
 * * @param value
 * @return      java.lang.Integer
 * @throws
 */
public Integer append(String key,String value){
    if(key==null){
        throw new BusinessException(4001004,"key不能为空");
    }
    return redisTemplate.opsForValue().append(key,value);
}

//*****hash数据类型*****
/**
 * 通过key 和 field 获取指定的 value
 * @Author:      小霍
 * @UpdateUser:
 * @Version:    0.0.1
 * @param key
 * @param field

```

```

    * @return      java.lang.Object
    * @throws
    */
    public Object hget(String key, Object field) {
        if(null==key||null==field){
            return null;
        }
        return redisTemplate.opsForHash().get(key,field);
    }

    /**
     * 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。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @param field
     * @param value
     * @return
     * @throws
     */
    public void hset(String key, Object field, Object value) {
        if(null==key||null==field){
            return;
        }
        redisTemplate.opsForHash().put(key,field,value);
    }

    /**
     * 判断指定Key中的指定Field是否存在, 返回true表示存在, false表示参数中的Field或Key不存在。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @param field
     * @return      java.lang.Boolean
     * @throws
     */
    public Boolean hexists(String key, Object field) {
        if(null==key||null==field){
            return false;
        }
        return redisTemplate.opsForHash().hasKey(key,field);
    }

    /**
     * 从指定Key的Hashes value中删除参数中指定的多个字段, 如果不存在字段将被忽略,
     * 返回实际删除的Field数量。如果Key不存在, 则将其视为空Hashes, 并返回0。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @param fields
     * @return      java.lang.Long
     * @throws
     */
    public Long hdel(String key, Object... fields) {
        if(null==key||null==fields||fields.length==0){
            return 0L;
        }
        return redisTemplate.opsForHash().delete(key,fields);
    }

    /**
     * 通过key获取所有的field和value
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @return      java.util.Map<java.lang.Object,java.lang.Object>
     * @throws

```



```

*/
public Map<Object, Object> hgetAll(String key) {
    if(key==null){
        return null;
    }
    return redisTemplate.opsForHash().entries(key);
}

/**
 * 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。
 * @Author: 小霍
 * @UpdateUser:
 * @Version: 0.0.1
 * @param key
 * @param hash
 * @return
 * @throws
 */
public void hmset(String key, Map<String, Object> hash) {

    if(null==key||null==hash){
        return;
    }
    redisTemplate.opsForHash().putAll(key,hash);
}

/**
 * 获取和参数中指定Fields关联的一组values，其返回顺序等同于Fields的请求顺序。
 * 如果请求的Field不存在，其值对应的value为null。
 * @Author: 小霍
 * @UpdateUser:
 * @Version: 0.0.1
 * @param key
 * @param fields
 * @return java.util.List<java.lang.Object>
 * @throws
 */
public List<Object> hmget(String key, Collection<Object> fields) {

    if(null==key||null==fields){
        return null;
    }

    return redisTemplate.opsForHash().multiGet(key,fields);
}

/**
 * 对应key的字段自增相应的值
 * @Author: 小霍
 * @UpdateUser:
 * @Version: 0.0.1
 * @param key
 * • * @param field
 * • * @param increment
 * @return java.lang.Long
 * @throws
 */
public Long hincrBy(String key,Object field,long increment){
    if (null==key||null==field){
        throw new BusinessException(4001004,"key or field 不能为空");
    }
    return redisTemplate.opsForHash().increment(key,field,increment);
}

}

//*****List数据类型*****
/**
 * 向列表左边添加元素。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，该命令将会抛出相关异常。操作成功则返回插入后链表中元素的数量。
 * @Author: 小霍
 * @UpdateUser:
 * @Version: 0.0.1
 * @param key
 * @param strs 可以使一个string 也可以使string数组

```

```

    * @return      java.lang.Long 返回操作的value个数
    * @throws
    */
    public Long lpush(String key, Object... str) {
        if(null==key){
            return 0L;
        }
        return redisTemplate.opsForList().leftPushAll(key, str);
    }

    /**
     * 向列表右边添加元素。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，该命令将会抛出相关异常。操作成功则返回插入后链表中元素的数量。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @param str 可以使一个string 也可以使string数组
     * @return      java.lang.Long 返回操作的value个数
     * @throws
     */
    public Long rpush(String key, Object... str) {
        if(null==key){
            return 0L;
        }
        return redisTemplate.opsForList().rightPushAll(key, str);
    }

    /**
     * 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。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @return      java.lang.Object
     * @throws
     */
    public Object lpop(String key) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForList().leftPop(key);
    }

    /**
     * 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。
     * RPOP命令执行两步操作：第一步是将列表右边的元素从列表中移除，第二步是返回被移除的元素值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @return      java.lang.Object
     * @throws
     */
    public Object rpop(String key) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForList().rightPop(key);
    }

    /**
     * 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。返回指定范围内元素的列表。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @param start
     * @param end
     * @return      java.util.List<java.lang.Object>

```

```

    * @throws
    */
    public List<Object> lrange(String key, long start, long end) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForList().range(key,start,end);
    }

    /**
     * 让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。
     * 下标 0 表示列表的第一个元素，以 1 表示列表的第二个元素，以此类推。
     * 你也可以使用负数下标，以 -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。
     * @Author: 小霍
     * @UpdateUser:
     * @Version: 0.0.1
     * @param key
     * @param start
     * @param end
     * @return
     * @throws
     */
    public void ltrim(String key, long start, long end) {
        if(null==key){
            return;
        }
        redisTemplate.opsForList().trim(key,start,end);
    }

    /**
     * 该命令将返回链表中指定位置(index)的元素，index是0-based，表示从头部位置开始第index的元素，
     * 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。 如果超出index返回这返回nil。
     * @Author: 小霍
     * @UpdateUser:
     * @Version: 0.0.1
     * @param key
     * @param index
     * @return java.lang.Object
     * @throws
     */
    public Object lindex(String key, long index) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForList().index(key,index);
    }

    /**
     * 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，则抛出相关的异常。
     * @Author: 小霍
     * @UpdateUser:
     * @Version: 0.0.1
     * @param key
     * @return java.lang.Long
     * @throws
     */
    public Long llen(String key) {
        if(null==key){
            return 0L;
        }
        return redisTemplate.opsForList().size(key);
    }

    /**
     * *****Set数据类型*****
     */
    /**
     * 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。返回实际插入的成员数量。
     * @Author: 小霍
     * @UpdateUser:
     * @Version: 0.0.1
     * @param key
     * * @param members 可以是一个String 也可以是一个String数组
     * @return java.lang.Long 添加成功的个数

```

```

    * @throws
    */
    public Long sadd(String key, Object... members) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForSet().add(key, members);
    }

    /**
     * 返回Set中成员的数量，如果该Key并不存在，返回0。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @return        java.lang.Long
     * @throws
     */
    public Long scard(String key) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForSet().size(key);
    }

    /**
     * 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true表示已经存在，false表示不存在，或该key本身并不存在。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * • * @param member
     * @return        java.lang.Boolean
     * @throws
     */
    public Boolean sismember(String key, Object member) {
        if (null==key){
            return false;
        }
        return redisTemplate.opsForSet().isMember(key, member);
    }

    /**
     * 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。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @return        java.lang.String
     * @throws
     */
    public Object srandmember(String key) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForSet().randomMember(key);
    }

    /**
     * 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，如果|count|大与集合中的元素，
     * 就会返回全部元素不够的以重复元素补齐，如果key不存在则返回nil。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * • * @param count

```

```

    * @return      java.util.List<java.lang.String>
    * @throws
    */
    public List<Object> srandmember(String key,int count) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForSet().randomMembers(key, count);
    }

    /**
     * 通过key随机删除一个set中的value并返回该值
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @return      java.lang.String
     * @throws
     */
    public Object spop(String key) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForSet().pop(key);
    }

    /**
     * 通过key获取set中所有的value
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @return      java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> smembers(String key) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForSet().members(key);
    }

    /**
     * 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。返回从Set中实际移除的成员数量，如果没有则返回0。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * • * @param members
     * * @return      java.lang.Long
     * * @throws
     */
    public Long srem(String key, Object... members) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForSet().remove(key, members);
    }

    /**
     * 将元素value从一个集合移到另一个集合
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param srckey
     * • * @param dstkey
     * • * @param member
     * * @return      java.lang.Long
     * * @throws
     */

```

```

public Boolean smove(String srckey, String dstkey, Object member) {
    if (null==srckey||null==dstkey){
        return false;
    }
    return redisTemplate.opsForSet().move(srckey,member,dstkey);
}

/**
 * 获取两个集合的并集
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * * @param otherKeys
 * @return      java.util.Set<java.lang.Object> 返回两个集合合并值
 * @throws
 */
public Set<Object> sunion(String key, String otherKeys) {
    if (null==key||otherKeys==null){
        return null;
    }
    return redisTemplate.opsForSet().union(key, otherKeys);
}
//*****Sorted Set 数据类型*****
/**
 *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。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * * @param score
 * * @param member
 * @return      java.lang.Long
 * @throws
 */
public Boolean zadd(String key, double score, Object member) {
    if (null==key){
        return false;
    }
    return redisTemplate.opsForZSet().add(key,member,score);
}

/**
 * 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。 如果操作成功则返回实际被删除的成员数量。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * * @param members 可以使一个string 也可以是一个string数组
 * @return      java.lang.Long
 * @throws
 */
public Long zrem(String key, Object... members) {
    if (null==key||null==members){
        return 0L;
    }
    return redisTemplate.opsForZSet().remove(key,members);
}

/**
 * 返回Sorted Set中的成员数量，如果该Key不存在，返回0。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:      0.0.1
 * @param key
 * @return      java.lang.Long

```

```

    * @throws
    */
    public Long zcard(String key) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForZSet().size(key);
    }

    /**
     * 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。如果不报错则以串形式表示的新分数。
     * @Author: 小霍
     * @UpdateUser:
     * @Version: 0.0.1
     * @param key
     * * @param score
     * * @param member
     * @return java.lang.Double
     * @throws
     */
    public Double zincrby(String key, double score, Object member) {
        if (null==key){
            throw new BusinessException(4001004,"key 不能为空");
        }
        return redisTemplate.opsForZSet().incrementScore(key,member,score);
    }

    /**
     * 该命令用于获取分数(score)在min和max之间的成员数量。
     * (min=<score<=max) 如果加上了“(”着表明是开区间例如zcount key (min max 则 表示 (min<score<=max)
     * 同理zcount key min (max 则表明(min=<score<max) 返回指定返回数量。
     * @Author: 小霍
     * @UpdateUser:
     * @Version: 0.0.1
     * @param key
     * * @param min
     * * @param max
     * @return java.lang.Long
     * @throws
     */
    public Long zcount(String key, double min, double max) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForZSet().count(key, min, max);
    }

    /**
     * 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。 如果该成员存在，则返回它的位置索引值。否则返回nil。
     * @Author: 小霍
     * @UpdateUser:
     * @Version: 0.0.1
     * @param key
     * * @param member
     * @return java.lang.Long
     * @throws
     */
    public Long zrank(String key, Object member) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().rank(key,member);
    }

    /**
     * 如果该成员存在，以字符串的形式返回其分数，否则返回null
     * @Author: 小霍
     * @UpdateUser:

```

```

    * @Version:      0.0.1
    * @param key
    * * @param member
    * @return        java.lang.Double
    * @throws
    */
    public Double zscore(String key, Object member) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().score(key,member);
    }

    /**
     * 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...。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
    * * @param min
    * * @param max
     * @return        java.util.Set<java.lang.String> 指定区间内的有序集成员的列表。
     * @throws
     */
    public Set<Object> zrange(String key, long min, long max) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().range(key, min, max);
    }

    /**
     * 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。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
    * * @param start
    * * @param end
     * @return        java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> zReverseRange(String key, long start, long end) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().reverseRange(key, start, end);
    }

    /**
     * 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。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
    * * @param max
    * * @param min
     * @return        java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> zrangebyscore(String key, double min, double max) {
        if (null==key){
            return null;
        }
    }

```



```

        return redisTemplate.opsForZSet().rangeByScore(key, min, max);
    }

    /**
     * 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。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * * @param max
     * * @param min
     * @return      java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> zrevrangeByScore(String key, double min, double max) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().reverseRangeByScore(key, min, max);
    }
}

```

## 5. Spring Boot2.x redis 实战-分布式 Session 共享

### 5.1 业务场景分析

以往我们的项目都是部署在单台服务器运行，因为客户的所有请求都是由唯一服务器来处理，sessionId 保存在这台服务器上是没有问题的。但是当项目同时部署在多台服务器上时，就会出现 sessionId 共享问题。

现在有两台服务器同时运行，分别是 Server A 和 Server B，通过 nginx 配置了负载均衡，客户端的请求会被随机分配到两台服务器上进行处理。假设客户现在有第一次请求（登录请求）被分配到 Server A 进行处理，Server A 接受到请求之后会生成 sessionId 并且保存到内存当中，然后返回给客户（浏览器），浏览器会把 sessionId 保存到 cookie 中，第一次请求完成。如果之后每一次请求还是由 Server A 来处理当然一切正常，但是一旦出现意外（比如 Server A 宕机）或者nginx 采用了轮询、weight方式负载均衡，请求被分配到 Server B 进行处理，这时候 Server B 拿到客户请求的 sessionId 是由 Server A 生成的，两边对不上啊！于是客户会发现，本来还用的好好的，怎么会突然跳到登录页需要重新登录了呢！！？？？

那我们该怎么去解决呢？

既然问题的根源出在 sessionId 无法共享上面，那么是不是让 sessionId 可以在多台服务器之间共享就可以了？换个思路，即把 sessionId 保存到数据库即可（最终选择redis,因为会比较快！），验证时不再从当前服务器获取 sessionId 改从 redis 中获取即可！

实现思路：

1. 登录页面提交用户名密码。
2. 登录成功后生成token。Token相当于原来的sessionId，字符串，可以使用uuid。
3. 把用户信息保存到redis。Key就是token，value就是userId。
4. 设置key的过期时间。模拟Session的过期时间。一般一个小时。
5. 拦截器拦截请求校验 sessionId。

### 5.2 代码实现

1. 登录成功 生成 sessionId 存入 redis

```

package com.yingxue.lesson.service.impl;

import com.yingxue.lesson.entity.SysUser;
import com.yingxue.lesson.exception.BusinessException;
import com.yingxue.lesson.mapper.SysuserMapper;
import com.yingxue.lesson.service.RedisService;
import com.yingxue.lesson.service.UserService;
import com.yingxue.lesson.utils.PasswordUtils;
import com.yingxue.lesson.vo.req.LoginReqVO;
import com.yingxue.lesson.vo.resp.LoginRespVO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.UUID;
import java.util.concurrent.TimeUnit;

```

```

/**
 * @ClassName: UserServiceImpl
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private SysUserMapper sysUserMapper;
    @Autowired
    private RedisService redisService;
    @Override
    public LoginRespVO login(LoginReqVO vo) {
        SysUser sysUser=sysUserMapper.getUserInfoByName(vo.getUsername());
        if(sysUser==null){
            throw new BusinessException(4001004,"用户名密码不匹配");
        }
        if(sysUser.getStatus()==2){
            throw new BusinessException(4001004,"该用户已被禁用请联系系统管理员");
        }
        if(!PasswordUtils.matches(sysUser.getSalt(),vo.getPassword(),sysUser.getPassword())){
            throw new BusinessException(4001004,"用户名密码不匹配");
        }
        LoginRespVO respVO=new LoginRespVO();
        respVO.setToken(UUID.randomUUID().toString());
        respVO.setId(sysUser.getId());
        //凭证存入redis 60分钟失效
        redisService.set(respVO.getToken(),respVO.getId(),60, TimeUnit.MINUTES);
        return respVO;
    }
}

```

## 2. SessionInterceptor 拦截器校验 sessionId

```

package com.yingxue.lesson.interceptor;

import com.yingxue.lesson.exception.BusinessException;
import com.yingxue.lesson.service.RedisService;
import io.netty.util.internal.StringUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.servlet.HandlerInterceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * @ClassName: LoginInterceptor
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class LoginInterceptor implements HandlerInterceptor {
    @Autowired
    private RedisService redisService;
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        //无论访问的地址是不是正确的，都进行登录验证，登录成功后的访问再进行分发，404的访问自然会进入到错误控制器中
        String sessionId=request.getHeader("sessionId");
        if (StringUtil.isEmpty(sessionId)){
            throw new BusinessException(4001002,"授权信息为空请重新登录");
        }else {
            if(!redisService.hasKey(sessionId)){
                throw new BusinessException(4001002,"授权信息无效请重新登录");
            }
        }
        return true;
    }
}

```

```
}
```

```
package com.yingxue.lesson.config;

import com.yingxue.lesson.interceptor.SessionInterceptor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

/**
 * @ClassName: WebAppConfig
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Configuration
public class WebAppConfig implements WebMvcConfigurer {
    @Bean
    public SessionInterceptor getSessionInterceptor(){
        return new SessionInterceptor();
    }
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        //所有已api开头的访问都要进入LoginInterceptor拦截器进行登录验证，并排除login接口(全路径)。必须写成链
        //式，分别设置的话会创建多个拦截器。
        //必须写成getSessionInterceptor()，否则SessionInterceptor中的@Autowired会无效

        registry.addInterceptor(getSessionInterceptor()).addPathPatterns("/api/**").excludePathPatterns("/api/user/login");
    }
}
```

## 6. Spring Boot2.x redis 实战-异地登录提醒下线

### 6.1 业务分析

最近接到产品提的一个需求说，一个账号同时只能在一个地方登录，如果在其他地方登录则提示已在别处登录，同时，同一浏览器同时只能登录一个用户。

思路：

1. 登录页面提交用户名密码。
2. 登录成功后生成token。Token相当于原来的 sessionid，字符串，可以使用uuid。
3. 把用户信息保存到redis。Key就是token，value就是userId。
4. 设置key的过期时间。模拟Session的过期时间。一般一个小时。
5. 标记 Token把Token存入redis，key为 userId,value 就是 Token过期时间和 key 为 Token 的过期时间一致
6. 拦截器拦截请求校验 token。
7. 获取 userId 后再去比较 header 携带的token和redis标记的token是否一致，不一致则提示用户已经异地登录。

### 6.2 代码实现

#### 1. 修改登录逻辑

```
@Override
public LoginRespVO login(LoginReqVO vo) {
    SysUser sysUser=sysUserMapper.getUserInfoByName(vo.getUsername());
    if(sysUser==null){
        throw new BusinessException(4001004,"用户名密码不匹配");
    }
    if(sysUser.getStatus()==2){
        throw new BusinessException(4001004,"该用户已被禁用请联系系统管理员");
    }
    if(!PasswordUtils.matches(sysUser.getSalt(),vo.getPassword(),sysUser.getPassword())){
        throw new BusinessException(4001004,"用户名密码不匹配");
    }
    LoginRespVO respVO=new LoginRespVO();
```

```

respVO.setToken(UUID.randomUUID().toString());
respVO.setId(sysUser.getId());
//凭证存入redis 60分钟失效
redisService.set(respVO.getToken(), respVO.getId(), 60, TimeUnit.MINUTES);
/**
 * 异地登录失效逻辑
 */
redisService.set(sysUser.getId(), respVO.getToken(), 60, TimeUnit.MINUTES);
return respVO;
}

```

## 2. 修改token拦截器逻辑

```

@Override
public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
handler) throws Exception {
//无论访问的地址是不是正确的，都进行登录验证，登录成功后的访问再进行分发，404的访问自然会进入到错误控制器中
String sessionId=request.getHeader("sessionId");
if (StringUtil.isNullOrEmpty(sessionId)){
throw new BusinessException(4001002,"授权信息为空请重新登录");
}else {
if(!redisService.hasKey(sessionId)){
throw new BusinessException(4001002,"授权信息无效请重新登录");
}
/**
 * 异地登录后 上一个登录凭证失效提示
 */
String userId= (String) redisService.get(sessionId);
if(redisService.hasKey(userId)&&!sessionId.equals(redisService.get(userId))){
throw new BusinessException(4001002,"您的账号已经在异地登录请重新登录");
}
}
return true;
}
}

```

## 7. Spring Boot2.x redis 实战-注册短信验证码

### 7.1 业务场景分析

短信验证码是所有项目必不可少的基础功能模块之一，假如突然有一天你领导给你布置的一个需求。在用户注册的时候要校验手机号。

要求如下：

1. 注册的时候校验手机号
2. 每个手机号每天最多发送五条注册短信验证码
3. 验证码5分钟内有效。

思路：

1. 发送前验证手机号是否符合要求。
2. 生成短信验证码。
3. 发送验证码到手机。
4. 把验证码存入redis
5. 标记手机号
6. 注册的时候校验手机号和验证码是否正确

### 7.2 代码实现

1. controller

```

@GetMapping("/code/{phone}")
@ApiOperation(value = "获取验证码接口")
public void getCode(@PathVariable("phone")String phone){
    userService.getCode(phone);
}

@PostMapping("/user/register")
@ApiOperation(value = "用户注册接口接口")
public String register(@RequestBody RegisterReqVO vo){
    return userService.register(vo);
}

```

## 2. RegisterReqVO

```

package com.yingxue.lesson.vo.req;

import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

/**
 * @ClassName: RegisterReqVO
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class RegisterReqVO {
    @ApiModelProperty(value = "账号")
    private String username;
    @ApiModelProperty(value = "手机号")
    private String phone;
    @ApiModelProperty(value = "密码")
    private String password;
    @ApiModelProperty(value = "验证码")
    private String code;
}

```

## 3. service

```

/**
 * 判断是否达上线的key
 */
private final String REGISTER_CODE_COUNT_KEY="register-code-count-key-";

/**
 * 验证码有效期key
 */
private final String REGISTER_CODE_COUNT_VALIDITY_KEY="register-code-count-validity-key-";
@Override
public String register(RegisterReqVO vo) {

    //判断验证码是否有效
    if(!redisService.hasKey(REGISTER_CODE_COUNT_VALIDITY_KEY+vo.getPhone())){
        throw new BusinessException(4001004,"验证码已失效重新获取");
    }
    //校验验证码是否正确
    if(!vo.getCode().equals(redisService.get(REGISTER_CODE_COUNT_VALIDITY_KEY+vo.getPhone()))){
        throw new BusinessException(4001004,"验证码不正确请重新输入");
    }
    SysUser userInfoByName = sysUserMapper.getUserInfoByName(vo.getUsername());
    if(userInfoByName!=null){
        throw new BusinessException(4001004,"用户名已被占用");
    }
    SysUser sysUser=new SysUser();
    BeanUtils.copyProperties(vo,sysUser);
    sysUser.setId(UUID.randomUUID().toString());
    sysUser.setCreateTime(new Date());
    int i = sysUserMapper.insertSelective(sysUser);
    if(i!=1){

```

```

        throw new BusinessException(4001004,"操作失败");
    }
    //注册成功失效验证码
    redisService.delete(REGISTER_CODE_COUNT_VALIDITY_KEY+vo.getPhone());
    return "操作成功";
}

/**
 * 验证码有效期key
 */
private final String REGISTER_CODE_COUNT_VALIDITY_KEY="register-code-count-validity-key_";
@Override
public String getCode(String phone) {
    //验证手机号是否合法
    Pattern pattern = Pattern.compile("^1(3|4|5|7|8)\\d{9}$");
    Matcher matcher = pattern.matcher(phone);
    if(!matcher.matches()) {
        throw new BusinessException(4001004,"手机号格式错误");
    }
    //判断手机号是否超限
    long count = redisService.incrby(REGISTER_CODE_COUNT_KEY+phone,1);
    if(count>5){
        throw new BusinessException(4001004,"当日发送已达上限");
    }
    //生成6位随机数
    String code=generateCode();
    //发送短信(具体根据你们公司所用的api文档来)

    //存入 redis 过期时间为 5 分钟
    redisService.set(REGISTER_CODE_COUNT_VALIDITY_KEY+phone, code,5,TimeUnit.MINUTES);
    //发送短信这里用输出模拟
    System.out.println(code);
    return code;
}

/**
 * 生成六位验证码
 * @return
 */
private String generateCode(){
    Random random = new Random();
    int x = random.nextInt(899999);
    String code = String.valueOf(x + 100000);
    return code;
}

```

#### 4. 加上获取验证码、注册接口白名单

```

@Configuration
public class WebAppConfig implements WebMvcConfigurer {
    @Bean
    public LoginInterceptor getSessionInterceptor(){
        return new LoginInterceptor();
    }
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        //所有已api开头的访问都要进入getMyInterceptor拦截器进行登录验证，并排除login接口(全路径)。必须写成链
        //式，分别设置的话会创建多个拦截器。
        //LoginInterceptor(), LoginInterceptor @Autowired会无效

        registry.addInterceptor(getSessionInterceptor()).addPathPatterns("/api/**").excludePathPatterns("/api/user/login","/api/code/**","/api/user/register");
    }
}

```

## 8. Spring Boot2.x redis 实战-计数器(订单号/特殊有规律编码/点赞数)

### 8.1 业务场景分析

在现实开发中，经常遇到数据组或者产品给的需求，比如统计某个功能 pv、uv 数量、文章点赞数、或着有规律的编码等。

需求：

生成订单号为20191020D0000001 一天内可以生成9999999个不重复订单号(这个由我们自己设定)

实现思路：

1. 特定常量+当前日期作为key(确保唯一)
2. 每新生成一个订单号 incr 自增
3. 编码=日期+类型+自增部分

## 8.2代码实现

### 1. 接口

```
package com.yingxue.lesson.controller;

import com.yingxue.lesson.service.CodeService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

/**
 * @ClassName: CodeController
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@RestController
@RequestMapping("/api")
@Api(tags = "编码模块")
public class CodeController {
    @Autowired
    private CodeService codeService;
    @GetMapping("/code")
    @ApiOperation(value = "获取订单号")
    public String getOrderCode(@RequestParam(required = false) String type){
        return codeService.getOrderCode(type);
    }
}
```

### 2. service 业务

```
package com.yingxue.lesson.service.impl;

import com.yingxue.lesson.service.CodeService;
import com.yingxue.lesson.service.RedisService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.text.SimpleDateFormat;

/**
 * @ClassName: CodeServiceImpl
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Service
public class CodeServiceImpl implements CodeService {
    @Autowired
    private RedisService redisService;
    @Override
    public String getOrderCode(String type) {
        SimpleDateFormat s = new SimpleDateFormat("yyyyMMdd");
        String curDate = s.format(System.currentTimeMillis());
        Long number=redisService.incrby(type+curDate,1);
        String pad=padRight(number.toString(),7,"0");
    }
}
```

```

        return curDate+type+pad;
    }

    /**
     * 右补位，左对齐
     * @param oriStr 原字符串
     * @param len 目标字符串长度
     * @param alexin 补位字符
     * @return 目标字符串
     * 以alexin 做为补位
     */
    public String padRight(String oriStr,int len,String alexin){
        String str = "";
        int strlen = oriStr.length();
        if(strlen < len){
            for(int i=0;i<len-strlen;i++){
                str=str+alexin;
            }
        }
        str=str+oriStr;
        return str;
    }
}

```

## 9. Spring Boot2.x redis 实战-购物车

### 9.1业务场景分析





迎学教育  
YING XUE

问题在线解答  
2019新版SpringBoot2.0权限管理  
视频源码

SpringBoot2.x零基础到高级实战全套视频源码

轻松掌握	redis实战	实战框架
快速熟悉	解析备注	Shiro实战
swagger2	JWT的使用	学习交流群

¥ 49.99-69.99

库存13329件

请选择 校区

## 付款方式

全额支付

## 校区

源码+VIP指导+赠送视频

源码+赠送视频

## 购买数量

—

1

+

确定

在做购物车的时候要考虑到对于一个客户来说 不同规格，不同商品，在实际应该中我们该怎么处理呢？

需求：

1. 加入购物车先得登录。
2. 记录用户购物车数据。

思路：

我们在做购物车功能的时候，我们可以用 redis hash类型来做。首先前端提交购物车的时候需要先判断是否登录，需要把商品的 skuld、规格属性specificationIds (多个的id以',' 拼接)、和商品的数量。展示的时候去redis 拿到对应购物车数据查询最新数据

redis hash 格式为

key	filed	value
'cart_'+userId(确保唯一、且能确认身份)	skuld+","+specificationIds(规格属性id拼接生成)	num

## 9.2 代码实现

### 1. 接收购物车数据VO

```
package com.yingxue.lesson.vo.req;

import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

/**
 * @ClassName: AddCartReqVO
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class AddCartReqVO {
    @ApiModelProperty(value = "商品skuId")
    private String sukId;
    @ApiModelProperty(value = "属性规格id拼接集合(以逗号隔开)")
    private String specificationIds;
    @ApiModelProperty(value = "加入购物车数量")
    private Integer num;
}
```

### 2. service 业务逻辑

```
package com.yingxue.lesson.service.impl;

import com.yingxue.lesson.service.CartService;
import com.yingxue.lesson.service.RedisService;
import com.yingxue.lesson.vo.req.AddCartReqVO;
import com.yingxue.lesson.vo.resp.CartRespVO;
import com.yingxue.lesson.vo.resp.GoodsItemRespVO;
import com.yingxue.lesson.vo.resp.ValueItemRespVO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

/**
 * @ClassName: CartServiceImpl
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Service
public class CartServiceImpl implements CartService {
```

```

@Autowired
private RedisService redisService;
@Override
public String addCart(AddCartReqVO vo, String sessionId) {
    //获取用户id
    String userId= (String) redisService.get(sessionId);
    //生成用户购物车key
    String hashKey="cart_"+userId;
    //生成 hash 中的 Filed 值
    String filed=vo.getSukId()+vo.getSpecificationIds();
    //把购物车数据存入 redis
    redisService.hIncrBy(hashKey, filed, vo.getNum());
    return "操作成功";
}
}

```

### 3. 展示用户购物车.

#### 1. 新增相应前端数据VO

```

package com.yingxue.lesson.vo.resp;

import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

import java.math.BigDecimal;
import java.util.List;

/**
 * @ClassName: CartRespVO
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class CartRespVO {
    @ApiModelProperty(value = "商品skuId")
    private String skuId;
    @ApiModelProperty(value = "商品名称")
    private String productName;
    @ApiModelProperty(value = "规格属性")
    private List<ValueItemRespVO> valueItemList;
    @ApiModelProperty(value = "价格")
    private BigDecimal price;
    @ApiModelProperty(value = "数量")
    private Integer num;
    @ApiModelProperty(value = "图标")
    private String icon;
}

```

```

package com.yingxue.lesson.vo.resp;

import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

/**
 * @ClassName: ValueItemRespVO
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class ValueItemRespVO {
    @ApiModelProperty(value = "商品规格属性id")
    private String valueId;
    @ApiModelProperty(value = "商品规格属性名称")
    private String valueName;
}

```

```

    @ApiModelProperty(value = "商品规格属性类型名称")
    private String typeName;
    @ApiModelProperty(value = "商品规格属性类型")
    private String type;
}

```

```

package com.yingxue.lesson.vo.resp;

import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

import java.math.BigDecimal;

/**
 * @ClassName: GoodsItemRespVO
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class GoodsItemRespVO {
    @ApiModelProperty(value = "商品skuId")
    private String skuId;
    @ApiModelProperty(value = "商品名称")
    private String productName;
    @ApiModelProperty(value = "价格")
    private BigDecimal price;

    @ApiModelProperty(value = "图标")
    private String icon;
}

```

## 2. 具体业务实现

修改 CartServiceImpl

```

@Override
public List<CartRespVO> showCart(String sessionId) {
    //获取用户id
    String userId= (String) redisService.get(sessionId);
    //生成用户购物车key
    String hashKey="cart_"+userId;
    //获取用户购物车所有数据
    Map<Object, Object> maps = redisService.hgetAll(hashKey);
    //解析数据
    List<CartRespVO> result=new ArrayList<>();
    if(null==maps){
        return result;
    }
    // 1. entrySet遍历，在键和值都需要时使用（最常用）
    for (Map.Entry<Object, Object> entry : maps.entrySet()) {
        CartRespVO respVO=new CartRespVO();
        //商品数量
        Integer num=Integer.valueOf(entry.getValue().toString());
        String file= (String) entry.getKey();
        //获取商品skuId和商品熟悉id(数组第一个为skuId)
        String ids[]=file.split(",");
        //查询商品相信信息
        String skuId=ids[0];

        //拿到商品详细
        GoodsItemRespVO itemVO=getItem(skuId);
        respVO.setIcon(itemVO.getIcon());
        respVO.setPrice(itemVO.getPrice());
        respVO.setSkuId(itemVO.getSkuId());
        respVO.setProductName(itemVO.getProductName());
    }
}

```

```

        respVO.setNum(num);
        List<ValueItemRespVO> list=new ArrayList<>();
        for (int i=1;i<ids.length;i++){
            //拿属性值
            ValueItemRespVO item=getValueItem(ids[i]);
            list.add(item);
        }
        respVO.setValueItemList(list);
        result.add(respVO);
    }
    return result;
}

private GoodsItemRespVO getItem(String skuId) {
    //这里是 mock 数据
    GoodsItemRespVO respVO=new GoodsItemRespVO();
    respVO.setIcon("http://image.baidu.com/search/detail?
ct=503316480&z=0&ipn=d&word=%E5%9B%BE%E7%89%87&hs=0&pn=0&spn=0&di=7590&pi=0&rn=1&tn=baiduimagedetai
l&is=0%2C0&ie=utf-8&oe=utf-
8&c1=2&lm=-1&cs=1411728850%2C1869975885&os=69787666%2C250391253&simid=3412044283%2C207046138&adpici
d=0&lpn=0&ln=30&fr=ala&fm=&sme=&cg=&bdtype=0&oriquery=&objurl=http%3A%2F%2Fpic18.nipic.com%2F201201
03%2F8993051_170340691334_2.jpg&fromurl=ippr_z2C%24qAzdH3FAzdH3Fooo_z%26e3Bgttrtv_z%26e3Bv54AzdH3Ffi
5oAzdH3F9AzdH3F8a0AzdH3Fccb9nlmhbbud8kj_z%26e3Bip4s&gsm=&islist=&querylist=");
    respVO.setPrice(new BigDecimal(69.99));
    respVO.setSkuId(skuId);
    respVO.setProductName("迎学教育-spring boot 实战系列");
    return respVO;
}

private ValueItemRespVO getValueItem(String id){
    //这里是 mock 数据
    ValueItemRespVO respVO=new ValueItemRespVO();
    respVO.setType("1");
    respVO.setTypeNames("普通属性");
    respVO.setValueName("VIP 教学");
    respVO.setValueId(id);
    return respVO;
}

```

### 3. 接口

```

package com.yingxue.lesson.controller;

import com.yingxue.lesson.service.CartService;
import com.yingxue.lesson.vo.req.AddCartReqVO;
import com.yingxue.lesson.vo.resp.CartRespVO;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.HttpServletRequest;
import java.util.List;

/**
 * @ClassName: CartController
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@RestController
@RequestMapping("/api")
@Api(tags = "购物车模块")
public class CartController {
    @Autowired
    private CartService cartService;
    @PostMapping("/cart")
    @ApiOperation(value = "加入购物车接口")
    public String addCart(@RequestBody AddCartReqVO vo, HttpServletRequest request){
        String sessionId=request.getHeader("sessionId");
        return cartService.addCart(vo,sessionId);
    }
}

```

```
@ApiOperation(value = "展示购物车接口")
@GetMapping("/cart")
private List<CartRespVO> showCart(HttpServletRequest request){
    String sessionId=request.getHeader("sessionId");
    return cartService.showCart(sessionId);
}
}
```