**Pom.xml**

*<?***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 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>com.lshs</**groupId**>  
 <**artifactId**>sbtemf</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**packaging**>jar</**packaging**>  
  
 <**name**>sbtemf</**name**>  
 <**description**>Demo project for Spring Boot</**description**>  
  
 <**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.0.4.RELEASE</**version**>  
 *<!--spring boot 2.x版本 ： 如果你用了 spring- druid java.lang.ClassNotFoundException org.springframework.boot.bind.RelaxedDataBinder 改成1.5.X-->* <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**>  
  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**project.reporting.outputEncoding**>UTF-8</**project.reporting.outputEncoding**>  
 <**java.version**>1.8</**java.version**>  
 *<!--使用高版本的模板-->* <**thymeleaf.version**>3.0.9.RELEASE</**thymeleaf.version**>  
 *<!--布局支持 2 和thymeleaf 3 版本结合使用-->* <**thymeleaf-layout-dialect.version**>2.1.1</**thymeleaf-layout-dialect.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-thymeleaf</**artifactId**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>5.1.45</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.github.pagehelper</**groupId**>  
 <**artifactId**>pagehelper-spring-boot-starter</**artifactId**>  
 <**version**>1.2.5</**version**>  
 <**exclusions**>  
 <**exclusion**>  
 <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  
 <**groupId**>org.mybatis.spring.boot</**groupId**>  
 </**exclusion**>  
 </**exclusions**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.mybatis.spring.boot</**groupId**>  
 <**artifactId**>mybatis-spring-boot-starter</**artifactId**>  
 <**version**>1.3.1</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>druid</**artifactId**>  
 <**version**>1.1.0</**version**>  
 </**dependency**>  
  
 *<!--配置redis使用的包-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-redis</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.apache.commons</**groupId**>  
 <**artifactId**>commons-pool2</**artifactId**>  
 <**version**>2.0</**version**>  
 </**dependency**>  
  
 </**dependencies**>  
  
 <**build**>  
 <**plugins**>  
 <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 </**plugin**>  
 </**plugins**>  
 <**resources**>  
 <**resource**>  
 <**directory**>src/main/java</**directory**>  
 <**includes**>  
 <**include**>\*\*/\*.xml</**include**>  
 </**includes**>  
 *<!--<filtering>false</filtering>-->* </**resource**></**resources**>  
 </**build**>  
  
  
</**project**>

**SbtemfApplication.java**

@MapperScan(**"com.lshs.sbtemf.mapper"**)  
@EnableCaching  
**public class** SbtemfApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(SbtemfApplication.**class**, args);  
 }  
}

**application.properties**

*#redis数据库索引是0***spring.redis.database**=**0***#redis服务器地址***spring.redis.host**=**localhost***#redis服务器端口***spring.redis.port**=**6379***#redis链接密码(默认为空)***spring.redis.password**=  
  
*#Redis连接池最大连接数（负值表示没有限制）***spring.redis.jedis.pool.max-active**=**8***#连接池最大等待时间（负值表示没有限制）***spring.redis.jedis.pool.max-wait**=**-1***#连接池最大空闲链接***spring.redis.jedis.pool.max-idle**=**8***#最小链接***spring.redis.lettuce.pool.min-idle**=**0***#连接超时***spring.redis.timeout**=**30000**

*#redis集群  
#spring.redis.cluster.nodes=127.0.0.1:6379,127.0.0.1:6380,127.0.0.1:6381*

**RedisUtil.java**

**package** com.lshs.sbtemf.utils;  
  
**import** org.springframework.data.redis.core.RedisTemplate;  
**import** org.springframework.util.CollectionUtils;  
  
**import** java.util.Collections;  
**import** java.util.List;  
**import** java.util.Map;  
**import** java.util.Set;  
**import** java.util.concurrent.TimeUnit;  
  
*/\*\*  
 \** ***@Description:*** *redis工具类  
 \** ***@author:*** *LuShao  
 \** ***@create:*** *2018-08-31 10:14  
 \*\*/***public class** RedisUtil {  
  
 **private** RedisTemplate<String,Object> **redisTemplate**;  
  
 **public void** setRedisTemplate(RedisTemplate<String, Object> redisTemplate) {  
 **this**.**redisTemplate** = redisTemplate;  
 }  
  
 *//-------------------------Common-----------------------  
  
 /\*\*  
 \* 指定缓存失效时间  
 \** ***@param key*** *\** ***@param time*** *(s)  
 \** ***@return*** *\*/* **public boolean** expire(String key,**long** time){  
 **try** {  
 **if** (time>0){  
 **redisTemplate**.expire(key,time, TimeUnit.***SECONDS***);  
 }  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 根据key得到过期时间  
 \** ***@param key*** *\** ***@return*** *\*/* **public long** getExpire(String key){  
 **return redisTemplate**.getExpire(key);  
 }  
  
 */\*\*  
 \* 判断key是否存在  
 \** ***@param key*** *\** ***@return*** *\*/* **public boolean** hasKey(String key){  
 **try** {  
 **return redisTemplate**.hasKey(key);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 删除key 可以是多个  
 \** ***@param key*** *\*/* **public void** delKey(String ...key){  
 **if**(key!=**null**&&key.**length**>0){  
 **if** (key.**length**==1){  
 **redisTemplate**.delete(key[0]);  
 }**else** {  
 **redisTemplate**.delete(CollectionUtils.*arrayToList*(key));  
 }  
 }  
 }  
  
 *//-------------------------String-----------------------  
  
 /\*\*  
 \* 普通缓存获取  
 \** ***@param key*** *\** ***@return*** *\*/* **public** Object get(String key){  
 **return** key==**null**?**null**:**redisTemplate**.opsForValue().get(key);  
 }  
  
 */\*\*  
 \* 普通缓存放入  
 \** ***@param key*** *\** ***@param o*** *\** ***@return*** *\*/* **public boolean** set(String key,Object o){  
 **try** {  
 **redisTemplate**.opsForValue().set(key,o);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 放入普通缓存并设置时间  
 \** ***@param key*** *\** ***@param o*** *\** ***@param time*** *\** ***@return*** *\*/* **public boolean** set(String key,Object o,**long** time){  
 **try** {  
 **redisTemplate**.opsForValue().set(key,o,time,TimeUnit.***SECONDS***);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 递增  
 \** ***@param key*** *\** ***@param delta*** *\** ***@return*** *\*/* **public long** incr(String key,**long** delta){  
 **if** (delta<0){  
 **throw new** RuntimeException(**"递增因子必须大于0~~~"**);  
 }  
 **return redisTemplate**.opsForValue().increment(key,delta);  
 }  
  
 */\*\*  
 \* 递减  
 \** ***@param key*** *\** ***@param delta*** *\** ***@return*** *\*/* **public long** decr(String key,**long** delta){  
 **if** (delta<0){  
 **throw new** RuntimeException(**"递减因子必须大于0"**);  
 }  
 **return redisTemplate**.opsForValue().increment(key,-delta);  
 }  
  
 *//-------------------------Map-----------------------  
  
 /\*\*  
 \* 得到hashMap对象  
 \** ***@param key*** *\** ***@param item*** *\** ***@return*** *\*/* **public** Object hget(String key,String item){  
 **return redisTemplate**.opsForHash().get(key,item);  
 }  
  
 */\*\*  
 \* 获取hashKey的所有键值  
 \** ***@param key*** *\** ***@return*** *\*/* **public** Map<Object,Object> hmget(String key){  
 **return redisTemplate**.opsForHash().entries(key);  
 }  
  
 */\*\*  
 \* 设置hashMap  
 \** ***@param key*** *\** ***@param map*** *\** ***@return*** *\*/* **public boolean** hmset(String key,Map<String,Object> map){  
 **try** {  
 **redisTemplate**.opsForHash().putAll(key,map);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 设置 hashMap并设置时间  
 \** ***@param key*** *\** ***@param map*** *\** ***@param time*** *\** ***@return*** *\*/* **public boolean** hmset(String key,Map<String,Object> map,**long** time){  
 **try** {  
 **redisTemplate**.opsForHash().putAll(key,map);  
 **if** (time>0) expire(key,time);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 向一张hash表中放入数据，如果不存在则创建  
 \** ***@param key*** *\** ***@param item*** *\** ***@param o*** *\** ***@return*** *\*/* **public boolean** hset(String key,String item,Object o){  
 **try** {  
 **redisTemplate**.opsForHash().put(key,item,o);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 向一张hash表中放入数据，不存在则创建，如果设置时间则会覆盖原有的时间  
 \** ***@param key*** *\** ***@param item*** *\** ***@param o*** *\** ***@param time*** *\** ***@return*** *\*/* **public boolean** hset(String key,String item,Object o,**long** time){  
 **try** {  
 **redisTemplate**.opsForHash().put(key,item,o);  
 **if** (time>0) expire(key,time);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 删除hash表中的值  
 \** ***@param key*** *\** ***@param item*** *\*/* **public void** hdel(String key,Object ...item){  
 **redisTemplate**.opsForHash().delete(key,item);  
 }  
  
 */\*\*  
 \* 判断hash表中是否有该项的值  
 \** ***@param key*** *\** ***@param item*** *\** ***@return*** *\*/* **public boolean** hHasKey(String key,String item){  
 **return redisTemplate**.opsForHash().hasKey(key,item);  
 }  
  
 */\*\*  
 \* hash递增，如果不存在就会新建一个并把新增后的值返回  
 \** ***@param key*** *\** ***@param item*** *\** ***@param dy*** *\** ***@return*** *\*/* **public double** hincr(String key,String item,**double** dy){  
 **return redisTemplate**.opsForHash().increment(key,item,dy);  
 }  
  
 */\*\*  
 \* hash递减  
 \** ***@param key*** *\** ***@param item*** *\** ***@param dy*** *\** ***@return*** *\*/* **public double** hdecr(String key,String item,**double** dy){  
 **return redisTemplate**.opsForHash().increment(key,item,-dy);  
 }  
  
*// --------------------------------set----------------------------  
  
 /\*\*  
 \* 根据key获得set中所有的值  
 \** ***@param key*** *\** ***@return*** *\*/* **public** Set<Object> sGet(String key){  
 **return redisTemplate**.opsForSet().members(key);  
 }  
  
 */\*\*  
 \* 根据o从set中查询是否存在  
 \** ***@param key*** *\** ***@param o*** *\** ***@return*** *\*/* **public boolean** sHasKey(String key,Object o){  
 **return redisTemplate**.opsForSet().isMember(key,o);  
 }  
  
 */\*\*  
 \* 将数据放入set 返回插入多少个  
 \** ***@param key*** *\** ***@param o*** *\*/* **public long** sSet(String key,Object ...o){  
 **try** {  
 **return redisTemplate**.opsForSet().add(key,o);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return** 0;  
 }  
 }  
  
 */\*\*  
 \* 将数据放入set中，返回插入多少个，并设置时间  
 \** ***@param key*** *\** ***@param o*** *\** ***@param time*** *\** ***@return*** *\*/* **public long** sSet(String key,Object o,**long** time){  
 **try** {  
 Long add = **redisTemplate**.opsForSet().add(key, o);  
 **if** (time>0) expire(key,time);  
 **return** add;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return** 0;  
 }  
 }  
  
 */\*\*  
 \* 获得set缓存的长度  
 \** ***@param key*** *\** ***@return*** *\*/* **public long** sGetSize(String key){  
 **return redisTemplate**.opsForSet().size(key);  
 }  
  
 */\*\*  
 \* 移除值为o的key  
 \** ***@param key*** *\** ***@param o*** *\** ***@return*** *\*/* **public long** setRemove(String key,Object ...o){  
 **return redisTemplate**.opsForSet().remove(key,o);  
 }  
  
 *//===============================list================================  
  
 /\*\*  
 \* 根据key得到list  
 \** ***@param key*** *\** ***@param start*** *\** ***@param end*** *\** ***@return*** *\*/* **public** List lGet(String key,**long** start,**long** end){  
 **return redisTemplate**.opsForList().range(key,start,end);  
 }  
  
 */\*\*  
 \* 得到key的list长度  
 \** ***@param key*** *\** ***@return*** *\*/* **public long** lSize(String key){  
 **return redisTemplate**.opsForList().size(key);  
 }  
  
 */\*\*  
 \* 通过索引获得list中的值，index>=0 下标从0开始，index<0 从后往前取值，下标从-1开始  
 \** ***@param key*** *\** ***@param index*** *\** ***@return*** *\*/* **public** Object lGetByIndex(String key,**long** index){  
 **return redisTemplate**.opsForList().index(key,index);  
 }  
  
 */\*\*  
 \* list放入缓存  
 \** ***@param key*** *\** ***@param o*** *\** ***@return*** *\*/* **public boolean** lset(String key,Object o){  
 **try** {  
 **redisTemplate**.opsForList().rightPush(key,o);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* list放入缓存，并设置时间  
 \** ***@param key*** *\** ***@param o*** *\** ***@param time*** *\** ***@return*** *\*/* **public boolean** lset(String key,Object o,**long** time){  
 **try** {  
 **redisTemplate**.opsForList().rightPush(key,o);  
 **if** (time>0) expire(key,time);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* list放入缓存  
 \** ***@param key*** *\** ***@param lo*** *\** ***@return*** *\*/* **public boolean** lset(String key,List<Object> lo){  
 **try** {  
 **redisTemplate**.opsForList().rightPushAll(key,lo);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* list放入缓存并设置时间  
 \** ***@param key*** *\** ***@param lo*** *\** ***@param time*** *\** ***@return*** *\*/* **public boolean** lset(String key,List<Object> lo,**long** time){  
 **try** {  
 **redisTemplate**.opsForList().rightPushAll(key,lo);  
 **if** (time>0) expire(key,time);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 根据index修改list  
 \** ***@param key*** *\** ***@param index*** *\** ***@param o*** *\** ***@return*** *\*/* **public boolean** lupdateByindex(String key,**long** index,Object o){  
 **try** {  
 **redisTemplate**.opsForList().set(key,index,o);  
 **return true**;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return false**;  
 }  
 }  
  
 */\*\*  
 \* 移除n个o  
 \** ***@param key*** *\** ***@param count*** *\** ***@param o*** *\** ***@return*** *\*/* **public long** lRemove(String key,**long** count,Object o){  
 **return redisTemplate**.opsForList().remove(key,count,o);  
 }  
}

**RedisConfig.java**

**package** com.lshs.sbtemf.conf;  
  
**import** com.fasterxml.jackson.annotation.JsonAutoDetect;  
**import** com.fasterxml.jackson.annotation.PropertyAccessor;  
**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** com.lshs.sbtemf.utils.RedisUtil;  
**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.GenericJackson2JsonRedisSerializer;  
**import** org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;  
**import** org.springframework.data.redis.serializer.StringRedisSerializer;  
  
*/\*\*  
 \** ***@Description:*** *redis设置配置  
 \** ***@author:*** *LuShao  
 \** ***@create:*** *2018-08-31 13:30  
 \*\*/*@Configuration  
**public class** RedisConfig {  
  
 */\*\*  
 \* 实例化 RedisTemplate 对象  
 \** ***@param connectionFactory*** *\** ***@return*** *\*/* @Bean  
 **public** RedisTemplate<String,Object> functionDomainRedisTemplate(RedisConnectionFactory connectionFactory){  
 RedisTemplate<String,Object> redisTemplate=**new** RedisTemplate<>();  
 initDomainRedisTemplate(redisTemplate,connectionFactory);  
 **return** redisTemplate;  
 }  
  
 */\*\*  
 \* 设置数据存入 redis 的序列化方式,并开启事务  
 \** ***@param redisTemplate*** *\** ***@param connectionFactory*** *\*/  
 //@Bean* **public void** initDomainRedisTemplate(RedisTemplate<String,Object> redisTemplate,RedisConnectionFactory connectionFactory){  
 *//如果不配置Serializer，那么存储的时候缺省使用String，如果用User类型存储，那么会提示错误User can't cast to String！  
 //GenericJackson2JsonRedisSerializer jackson2JsonRedisSerializer=new GenericJackson2JsonRedisSerializer();  
 // 使用Jackson2JsonRedisSerialize 替换默认序列化* Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = **new** Jackson2JsonRedisSerializer(Object.**class**);  
  
 ObjectMapper objectMapper = **new** ObjectMapper();  
 objectMapper.setVisibility(PropertyAccessor.***ALL***, JsonAutoDetect.Visibility.***ANY***);  
 objectMapper.enableDefaultTyping(ObjectMapper.DefaultTyping.***NON\_FINAL***);  
  
 jackson2JsonRedisSerializer.setObjectMapper(objectMapper);  
 redisTemplate.setKeySerializer(jackson2JsonRedisSerializer);  
 redisTemplate.setHashKeySerializer(**new** StringRedisSerializer());  
 redisTemplate.setHashValueSerializer(**new** GenericJackson2JsonRedisSerializer());  
 redisTemplate.setValueSerializer(**new** GenericJackson2JsonRedisSerializer());  
 *//redisTemplate.afterPropertiesSet();  
 // 开启事务* redisTemplate.setEnableTransactionSupport(**true**);  
 redisTemplate.setConnectionFactory(connectionFactory);  
 }  
  
 */\*\*  
 \* 注入封装RedisTemplate  
 \** ***@param redisTemplate*** *\** ***@return*** *\*/* @Bean(name=**"redisUtil"**)  
 **public** RedisUtil redisUtil(RedisTemplate<String,Object> redisTemplate){  
 RedisUtil redisUtil=**new** RedisUtil();  
 redisUtil.setRedisTemplate(redisTemplate);  
 **return** redisUtil;  
 }  
}

**service.java**

**package** com.lshs.sbtemf.service;  
  
  
**import** com.lshs.sbtemf.conf.DS;  
**import** com.lshs.sbtemf.entry.Category;  
**import** com.lshs.sbtemf.entry.Contains;  
**import** com.lshs.sbtemf.mapper.CategoryMapper;  
**import** com.lshs.sbtemf.utils.RedisUtil;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Service;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
*/\*\*  
 \** ***@Description:*** *\** ***@author:*** *LuShao  
 \** ***@create:*** *2018-08-24 10:23  
 \*\*/*@Service  
**public class** CategoryServiceI **implements** CategoryService {  
  
 @Autowired  
 **private** CategoryMapper **categoryMapper**;  
  
 @Autowired  
 **private** RedisUtil **redisUtil**;  
  
 @Override  
 **public** List<Category> findAll() {  
 */\*外面会多一层[]  
 \* [[Category{categoryid=2, categoryname='风景'}, Category{categoryid=3, categoryname='美女'}, Category{categoryid=4, categoryname='日历'}, Category{categoryid=5, categoryname='游戏'}, Category{categoryid=6, categoryname='动漫'}, Category{categoryid=7, categoryname='动态'}, Category{categoryid=8, categoryname='唯美'}, Category{categoryid=9, categoryname='设计'}, Category{categoryid=10, categoryname='可爱'}, Category{categoryid=11, categoryname='汽车'}, Category{categoryid=12, categoryname='花卉'}, Category{categoryid=13, categoryname='动物'}, Category{categoryid=14, categoryname='节日'}, Category{categoryid=15, categoryname='人物'}, Category{categoryid=16, categoryname='美食'}, Category{categoryid=17, categoryname='水果'}, Category{categoryid=18, categoryname='建筑'}, Category{categoryid=19, categoryname='影视'}, Category{categoryid=20, categoryname='体育'}, Category{categoryid=21, categoryname='军事'}, Category{categoryid=22, categoryname='非主流'}, Category{categoryid=23, categoryname='其他'}, Category{categoryid=24, categoryname='王者荣耀'}, Category{categoryid=25, categoryname='护眼'}, Category{categoryid=26, categoryname='鬼刀'}]]  
 \* \*/* List<Category> list = **new** ArrayList<>();  
 **if** (**redisUtil**.hasKey(Contains.***CATEGORYLIST***)){  
 System.***err***.println(**"从redis中查询。。。。"**);

**public static final** String ***CATEGORYLIST***=**"CategoryList"**;

list= **redisUtil**.lGet(Contains.***CATEGORYLIST***, 0, -1);  
 System.***err***.println(**"Category~~~~"**+list);  
 }**else** {  
 list = **categoryMapper**.findAll();  
 **redisUtil**.lset(Contains.***CATEGORYLIST***,list,600);  
 }  
 *//list = categoryMapper.findAll();* **return** list;  
 }  
}

**下面就说一下集群**

**把之前的host和端口号配置注释，加上如下配置**

*#redis集群***spring.redis.cluster.nodes**=**127.0.0.1:6379,127.0.0.1:6380,127.0.0.1:6381  
spring.redis.cluster.commandTimeout**=**5000**

**JedisClusterConfig.java**

**package** com.lshs.sbtemf.conf.redis;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.autoconfigure.condition.ConditionalOnClass;  
**import** org.springframework.boot.context.properties.ConfigurationProperties;  
**import** org.springframework.boot.context.properties.EnableConfigurationProperties;  
**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.stereotype.Component;  
**import** redis.clients.jedis.HostAndPort;  
**import** redis.clients.jedis.JedisCluster;  
  
**import** java.util.HashSet;  
**import** java.util.Set;  
  
*/\*\*  
 \** ***@Description:*** *jedis集群配置  
 \** ***@author:*** *LuShao  
 \** ***@create:*** *2018-09-04 11:38  
 \*\*/*@Configuration  
@ConditionalOnClass({JedisCluster.**class**})  
@EnableConfigurationProperties(JedisClusterConfig.RedisProperties.**class**)  
**public class** JedisClusterConfig {  
  
 @Autowired  
 **private** RedisProperties **redisProperties**;

@Bean

**public** JedisCluster getJedisCluster(){  
 String[] split = **redisProperties**.getNodes().split(**","**);  
 Set<HostAndPort> set = **new** HashSet<>();  
 **for** (String s : split) {  
 String[] split1 = s.split(**":"**);  
 set.add(**new** HostAndPort(split1[0].trim(),Integer.*valueOf*(split1[1].trim())));  
 }  
 **return new** JedisCluster(set,**redisProperties**.getCommandTimeout());  
 }  
  
 */\*\*  
 \** ***@Description:*** *\** ***@author:*** *LuShao  
 \** ***@create:*** *2018-09-04 14:21  
 \*\*/* @Component  
 @ConfigurationProperties(prefix = **"spring.redis.cluster"**)  
 **static class** RedisProperties {  
  
 **private** String **nodes**;  
 **private** Integer **commandTimeout**;  
  
 **public** String getNodes() {  
 **return nodes**;  
 }  
  
 **public void** setNodes(String nodes) {  
 **this**.**nodes** = nodes;  
 }  
  
 **public** Integer getCommandTimeout() {  
 **return commandTimeout**;  
 }  
 **public void** setCommandTimeout(Integer commandTimeout) {  
 **this**.**commandTimeout** = commandTimeout;  
 }  
  
  
 }  
}

**service.java**

@Autowired  
**private** BiAnMapper **biAnMapper**;

@Autowired  
**private** JedisCluster **jedisCluster**;

**public** BiAn firstData(String spx) {  
 BiAn biAn = **new** BiAn();  
 **if** (**jedisCluster**.exists(**"bian"**)){  
 String bian = **jedisCluster**.get(**"bian"**);  
 JSONObject parse = (JSONObject) JSON.*parse*(bian);  
 JSONObject data = parse.getJSONArray(**"data"**).getJSONObject(0);  
 biAn = JSON.*toJavaObject*(data, BiAn.**class**);}  
 **return** biAn;  
}

<**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 <**version**>1.2.15</**version**>  
</**dependency**>