Redis在web中的应用（简单版）

1.redis在maven中的配置

|  |
| --- |
| <jedis.version>2.7.2</jedis.version>  <!-- Redis客户端 -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  <version>${jedis.version}</version>  </dependency>  <!-- Redis客户端 -->  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  </dependency> |

2.redis.xml的配置

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:context=*"http://www.springframework.org/schema/context"*  xmlns:jee=*"http://www.springframework.org/schema/jee"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:aop=*"http://www.springframework.org/schema/aop"*  xsi:schemaLocation=*"*  *http://www.springframework.org/schema/beans*  *http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop*  *http://www.springframework.org/schema/aop/spring-aop-4.0.xsd"*>  <!-- <context:property-placeholder location="classpath:resouces/db.properties" /> -->  <!-- 连接池配置 -->  <bean id=*"jedisPoolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>  <!-- 最大连接数 -->  <property name=*"maxTotal"* value=*"30"* />  <!-- 最大空闲连接数 -->  <property name=*"maxIdle"* value=*"10"* />  <!-- 每次释放连接的最大数目 -->  <property name=*"numTestsPerEvictionRun"* value=*"1024"* />  <!-- 释放连接的扫描间隔（毫秒） -->  <property name=*"timeBetweenEvictionRunsMillis"* value=*"30000"* />  <!-- 连接最小空闲时间 -->  <property name=*"minEvictableIdleTimeMillis"* value=*"1800000"* />  <!-- 连接空闲多久后释放, 当空闲时间>该值 且 空闲连接>最大空闲连接数 时直接释放 -->  <property name=*"softMinEvictableIdleTimeMillis"* value=*"10000"* />  <!-- 获取连接时的最大等待毫秒数,小于零:阻塞不确定的时间,默认-1 -->  <property name=*"maxWaitMillis"* value=*"1500"* />  <!-- 在获取连接的时候检查有效性, 默认false -->  <property name=*"testOnBorrow"* value=*"true"* />  <!-- 在空闲时检查有效性, 默认false -->  <property name=*"testWhileIdle"* value=*"true"* />  <!-- 连接耗尽时是否阻塞, false报异常,ture阻塞直到超时, 默认true -->  <property name=*"blockWhenExhausted"* value=*"false"* />  </bean>  <!-- jedis客户端单机版 -->  <bean id=*"redisClient"* class=*"redis.clients.jedis.JedisPool"*>  <constructor-arg name=*"host"* value=*"localhost"*></constructor-arg>  <constructor-arg name=*"port"* value=*"6379"*></constructor-arg>  <constructor-arg name=*"poolConfig"* ref=*"jedisPoolConfig"*></constructor-arg>  </bean>    <bean id=*"jedisClient"* class=*"com.sunshine.dao.impl.JedisClientImpl"*/>  <!-- 集群版 -->  <!-- <bean id="redisClient" class="redis.clients.jedis.JedisCluster">  <constructor-arg name="nodes">  <set>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"></constructor-arg>  <constructor-arg name="port" value="7001"></constructor-arg>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"></constructor-arg>  <constructor-arg name="port" value="7002"></constructor-arg>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"></constructor-arg>  <constructor-arg name="port" value="7003"></constructor-arg>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"></constructor-arg>  <constructor-arg name="port" value="7004"></constructor-arg>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"></constructor-arg>  <constructor-arg name="port" value="7005"></constructor-arg>  </bean>  <bean class="redis.clients.jedis.HostAndPort">  <constructor-arg name="host" value="192.168.25.153"></constructor-arg>  <constructor-arg name="port" value="7006"></constructor-arg>  </bean>  </set>  </constructor-arg>  <constructor-arg name="poolConfig" ref="jedisPoolConfig"></constructor-arg>  </bean>    <bean id="jedisCluster" class="com.sunshine.dao.impl.JedisClinetCluster"/>    -->  </beans> |

1. redis接口

|  |
| --- |
| **package** com.sunshine.dao;  **import** java.util.Map;  **import** java.util.Set;  **public** **interface** JedisClient<T> {    String get(String key);  String set(String key, String value);  String hget(String hkey, String key);  **long** hset(String hkey, String key, String value);  **long** incr(String key);  **long** expire(String key,**int** second);  **long** ttl(String key);  **void** setValue(String key, T value);  T getValue(String key);  String hmset(String key, Map<String, String> map);  Set<String> hkeys(String key);  } |

1. 单机版的实现类

|  |
| --- |
| package com.sunshine.dao.impl;  import java.beans.PropertyDescriptor;  import java.lang.reflect.InvocationTargetException;  import java.util.Date;  import java.util.HashMap;  import java.util.Map;  import java.util.Set;  import org.apache.commons.beanutils.BeanUtils;  import org.apache.commons.beanutils.PropertyUtils;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.data.redis.core.RedisCallback;  import org.springframework.data.redis.core.RedisTemplate;  import org.springframework.data.redis.serializer.RedisSerializer;  import com.sunshine.dao.JedisClient;  import redis.clients.jedis.Jedis;  import redis.clients.jedis.JedisPool;  public class JedisClientImpl<T> implements JedisClient<T> {    private Logger logger = LoggerFactory.getLogger(getClass());  private String category = "hap:cache";  private String name;  private Class<?> type;    public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public String getCategory() {  return category;  }  public void setCategory(String category) {  this.category = category;  }  protected RedisSerializer<String> strSerializer;    private RedisTemplate<String, String> redisTemplate;    protected String getFullKey(String key) {  return new StringBuilder(getCategory()).append(":").append(getName()).append(":").append(key).toString();  }    @Autowired  private JedisPool jedisPool;    public String get(String key) {  Jedis jedis = jedisPool.getResource();  String string = jedis.get(key);  jedis.close();  return string;  }  public String set(String key, String value) {  Jedis jedis = jedisPool.getResource();  String string = jedis.set(key, value);  jedis.close();  return string;  }  public String hget(String hkey, String key) {  Jedis jedis = jedisPool.getResource();  String string = jedis.hget(hkey, key);  jedis.close();  return string;  }  public long hset(String hkey, String key, String value) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.hset(hkey, key, value);  jedis.close();  return result;  }  public long incr(String key) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.incr(key);  jedis.close();  return result;  }  public long expire(String key, int second) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.expire(key, second);  jedis.close();  return result;  }  public long ttl(String key) {  Jedis jedis = jedisPool.getResource();  Long result = jedis.ttl(key);  jedis.close();  return result;  }    private Map<String, Object> convertToMap(Object obj)  throws IllegalAccessException, NoSuchMethodException, InvocationTargetException {  if (obj instanceof Map) {  return (Map<String, Object>) obj;  }  Map<String, Object> map = PropertyUtils.describe(obj);  System.out.println(map);  map.remove("class"); // describe会包含 class 属性,此处无用  return map;  }  @Override  public void setValue(String key, T value) {  try {  Map<String, Object> map = convertToMap(value);  setValue(key, map);  } catch (Exception e) {    }  }  private void setValue(String key, Map<String, Object> value) {  byte[] keyBytes = strSerializer.serialize(getFullKey(key));  Map<byte[], byte[]> data = new HashMap<>();  value.forEach((k, v) -> {  //排除特殊字段  if(k.charAt(0) == '\_'){  return;  }  if(v instanceof java.util.Date){  v = ((Date) v).getTime();  }  if(v != null){  data.put(strSerializer.serialize(k), strSerializer.serialize(v.toString()));  }  });    redisTemplate.execute((RedisCallback<Object>) (connection) -> {  connection.hMSet(keyBytes, data);  return null;  });  }  @Override  public T getValue(String key) {  return redisTemplate.execute((RedisCallback<T>) (connection) -> {  byte[] keyBytes = strSerializer.serialize(getFullKey(key));  Map<byte[], byte[]> value = connection.hGetAll(keyBytes);  if (value.size() == 0) {  return (T) null;  }  try {  Object bean = type.newInstance();  for (Map.Entry<byte[], byte[]> entry : value.entrySet()) {  String pName = strSerializer.deserialize(entry.getKey());  String pValue = strSerializer.deserialize(entry.getValue());  if (bean instanceof Map) {  ((Map) bean).put(pName, pValue);  continue;  }  PropertyDescriptor pd = PropertyUtils.getPropertyDescriptor(bean, pName);  if (pd == null) {  continue;  }  Class<?> pType = pd.getPropertyType();  if (pType == java.util.Date.class) {  Long time = pValue.length() == 0 ? null : Long.parseLong(pValue);  BeanUtils.setProperty(bean, pName, time);  } else {  BeanUtils.setProperty(bean, pName, pValue);  }  }  return (T) bean;  } catch (Exception e) {  if (logger.isErrorEnabled()) {  logger.error(e.getMessage(), e);  }  }  return (T) null;  });  }  @Override  public String hmset(String key, Map<String, String> map) {  Jedis jedis = jedisPool.getResource();  String string = jedis.hmset(key, map);  jedis.close();  return string;  }  @Override  public Set<String> hkeys(String key) {  Jedis jedis = jedisPool.getResource();  Set<String> set = jedis.hkeys(key);  jedis.close();  return set;  }  } |

1. 在service中的应用

|  |
| --- |
| @Override  **public** List<Student> getByStudent(Student student) {    //从缓存中去数据  **try** {  Student stu = studentMapper.getByName(student.getName());  Set<String> set = jedisClient.hkeys("redis\_student\_id\_"+stu.getId());  **for** (String string : set) {  String v = jedisClient.hget("redis\_student\_id\_"+stu.getId(), string);  **if**(!StringUtils.*isBlank*(v)) {  List<Student> stuList = JsonUtils.*jsonToList*(v, Student.**class**);  **return** stuList;  }  }  } **catch** (Exception e) {  e.printStackTrace();  }    List<Student> list = studentMapper.getByStudent(student);    //存入缓存  **try** {  Map<String, String> map = **new** HashMap<>();  String value = JsonUtils.*objectToJson*(list);  System.***out***.println("value====="+value);  map.put("name", value);  **for** (Student student2 : list) {  jedisClient.hmset("redis\_student\_id\_"+student2.getId(), map);  }    } **catch** (Exception e) {  e.printStackTrace();  }  **return** list;  } |