**添加POM依赖**

<!--集成redis-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-redis</artifactId>

<version>2.1.7.RELEASE</version>

</dependency>

<!--序列化-->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.54</version>

<scope>compile</scope>

</dependency>

<!--lombok,自动生成set、get等方法-->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>1.18.10</version>

<scope>compile</scope>

</dependency>

**配置连接的application.properties文件：**

spring.redis.host=127.0.0.1

spring.redis.port=6379

**测试连接：我们写个测试方法来进行最基本的连接测试！**

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.data.redis.core.RedisTemplate;

@SpringBootTest

public class testRedis {

@Autowired(required = false)

private RedisTemplate redisTemplate;

@Test

void getName(){

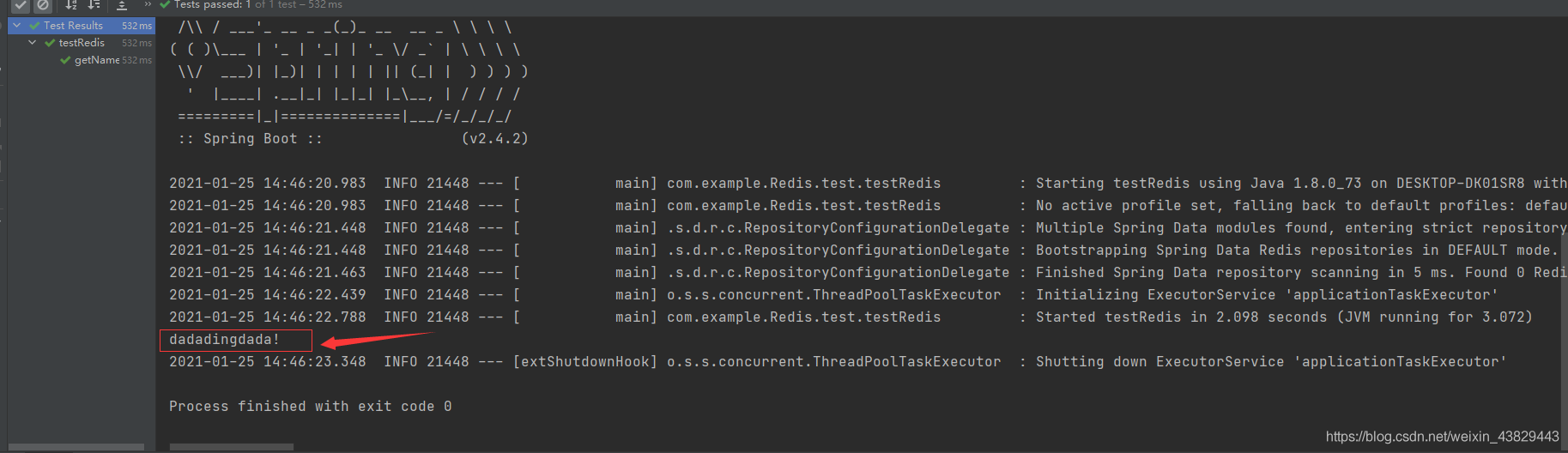
redisTemplate.opsForValue().set("name","dadadingdada!");

System.out.println(redisTemplate.opsForValue().get("name"));

}

}

运行效果如下！证明Redis连接成功！并且加数据获取数据也成功了！



**其他方法如下：**

// redisTemplate #操作不同的数据类型，api和我们的指令是一样的

// opsForValue #操作字符串 类似String

// opsForList #操作List 类似List

// opsForSet #操作set

// opsForHash #操作hash

// opsForZSet #操作zset

// opsForGeo #操作geo

// opsForHyperLogLog #操作HyperLogLog

// 除了进本的操作，我们常用的方法都可以直接通过redisTemplate操作，比如事务，和基本的 CRUD

// 获取redis的连接对象

// RedisConnection connection = redisTemplate.getConnectionFactory().getConnection();

// connection.flushDb();

// connection.flushAll();

**对象的保存和读取**

新增一个User类：

import lombok.Data;

@Data

public class User {

private String name;

private Integer age;

private Integer high;

}

测试代码如下：

@Test

void setObject(){

User user = new User();

user.setName("dingdada");

user.setAge(23);

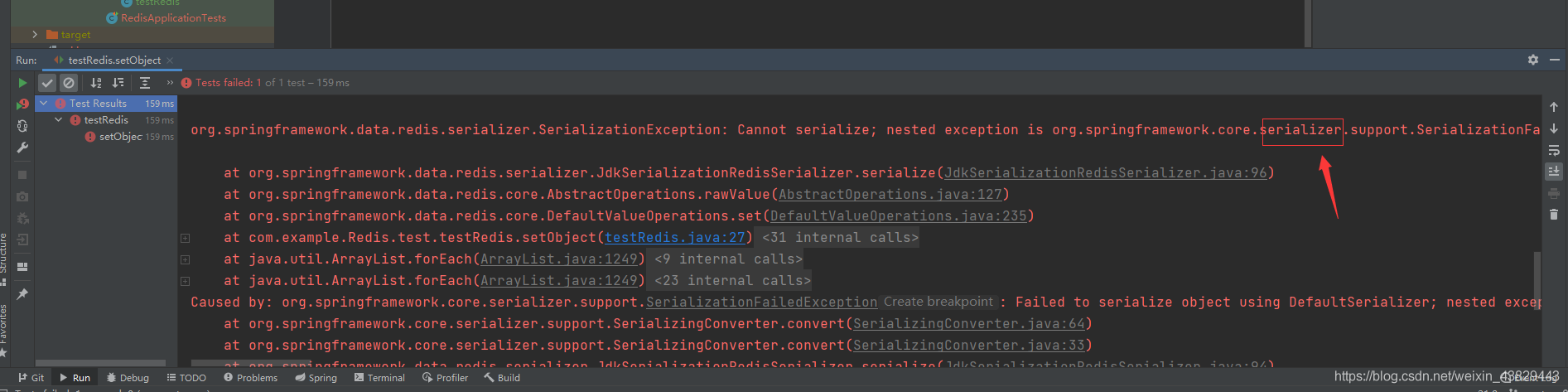
user.setHigh(172);

redisTemplate.opsForValue().set("user",user);

System.out.println(redisTemplate.opsForValue().get("user"));

}

报错如下：



结论：所以在操作Redis中，关于对象的保存我们得序列化才可以正常操作！

**自定义封装RedisTemplate类**

解决上方报错

import com.fasterxml.jackson.annotation.JsonAutoDetect;

import com.fasterxml.jackson.annotation.PropertyAccessor;

import com.fasterxml.jackson.databind.ObjectMapper;

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.Jackson2JsonRedisSerializer;

import org.springframework.data.redis.serializer.StringRedisSerializer;

@Configuration

public class RedisConfig {

// 这是我给大家写好的一个固定模板，大家在企业中，拿去就可以直接使用！

// 自己定义了一个RedisTemplate

@Bean

@SuppressWarnings("all")

public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {

// 我们为了自己开发方便，一般直接使用 <String, Object>

RedisTemplate<String, Object> template = new RedisTemplate<String, Object>();

template.setConnectionFactory(factory);

// Json序列化配置

Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new Jackson2JsonRedisSerializer(Object.class);

ObjectMapper om = new ObjectMapper();

om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);

om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON\_FINAL);

jackson2JsonRedisSerializer.setObjectMapper(om);

// String 的序列化

StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();

// key采用String的序列化方式

template.setKeySerializer(stringRedisSerializer);

// hash的key也采用String的序列化方式

template.setHashKeySerializer(stringRedisSerializer);

// value序列化方式采用jackson

template.setValueSerializer(jackson2JsonRedisSerializer);

// hash的value序列化方式采用jackson

template.setHashValueSerializer(jackson2JsonRedisSerializer);

template.afterPropertiesSet();

return template;

}

}

**封装RedisUtils类：**

不可能用**RedisTemplate** 来操作Redis的，因为实在太繁琐，所以我们一般自定义一个RedisUtils工具类来操作Redis！

import org.springframework.data.redis.core.RedisTemplate;

import org.springframework.stereotype.Component;

import org.springframework.util.CollectionUtils;

import javax.annotation.Resource;

import java.util.Collection;

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.concurrent.TimeUnit;

@Component

public final class RedisUtil {

@Resource

private RedisTemplate<String, Object> redisTemplate;

public Set<String> keys(String keys){

try {

return redisTemplate.keys(keys);

}catch (Exception e){

e.printStackTrace();

return null;

}

}

/\*\*

\* 指定缓存失效时间

\* @param key 键

\* @param time 时间(秒)

\* @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 获取过期时间

\* @param key 键 不能为null

\* @return 时间(秒) 返回0代表为永久有效

\*/

public long getExpire(String key) {

return redisTemplate.getExpire(key, TimeUnit.SECONDS);

}

/\*\*

\* 判断key是否存在

\* @param key 键

\* @return true 存在 false不存在

\*/

public boolean hasKey(String key) {

try {

return redisTemplate.hasKey(key);

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 删除缓存

\* @param key 可以传一个值 或多个

\*/

@SuppressWarnings("unchecked")

public void del(String... key) {

if (key != null && key.length > 0) {

if (key.length == 1) {

redisTemplate.delete(key[0]);

} else {

redisTemplate.delete((Collection<String>) CollectionUtils.arrayToList(key));

}

}

}

/\*\*

\* 普通缓存获取

\* @param key 键

\* @return 值

\*/

public Object get(String key) {

return key == null ? null : redisTemplate.opsForValue().get(key);

}

/\*\*

\* 普通缓存放入

\* @param key 键

\* @param value 值

\* @return true成功 false失败

\*/

public boolean set(String key, Object value) {

try {

redisTemplate.opsForValue().set(key, value);

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 普通缓存放入, 不存在放入，存在返回

\* @param key 键

\* @param value 值

\* @return true成功 false失败

\*/

public boolean setnx(String key, Object value) {

try {

redisTemplate.opsForValue().setIfAbsent(key,value);

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 普通缓存放入并设置时间

\* @param key 键

\* @param value 值

\* @param time 时间(秒) time要大于0 如果time小于等于0 将设置无限期

\* @return true成功 false 失败

\*/

public boolean set(String key, Object value, long time) {

try {

if (time > 0) {

redisTemplate.opsForValue().set(key, value, time, TimeUnit.SECONDS);

} else {

set(key, value);

}

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 普通缓存放入并设置时间,不存在放入，存在返回

\* @param key 键

\* @param value 值

\* @param time 时间(秒) time要大于0 如果time小于等于0 将设置无限期

\* @return true成功 false 失败

\*/

public boolean setnx(String key, Object value, long time) {

try {

if (time > 0) {

redisTemplate.opsForValue().setIfAbsent(key, value, time, TimeUnit.SECONDS);

} else {

set(key, value);

}

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 递增

\* @param key 键

\* @param delta 要增加几(大于0)

\* @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 键

\* @param delta 要减少几(小于0)

\* @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);

}

/\*\*

\* HashGet

\* @param key 键 不能为null

\* @param item 项 不能为null

\* @return 值

\*/

public Object hget(String key, String item) {

return redisTemplate.opsForHash().get(key, item);

}

/\*\*

\* 获取hashKey对应的所有键值

\* @param key 键

\* @return 对应的多个键值

\*/

public Map<Object, Object> hmget(String key) {

return redisTemplate.opsForHash().entries(key);

}

/\*\*

\* HashSet

\* @param key 键

\* @param map 对应多个键值

\* @return true 成功 false 失败

\*/

public boolean hmset(String key, Map<String, Object> map) {

try {

redisTemplate.opsForHash().putAll(key, map);

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* HashSet 并设置时间

\* @param key 键

\* @param map 对应多个键值

\* @param time 时间(秒)

\* @return true成功 false失败

\*/

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,如果不存在将创建

\* @param key 键

\* @param item 项

\* @param value 值

\* @return true 成功 false失败

\*/

public boolean hset(String key, String item, Object value) {

try {

redisTemplate.opsForHash().put(key, item, value);

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 向一张hash表中放入数据,如果不存在将创建

\* @param key 键

\* @param item 项

\* @param value 值

\* @param time 时间(秒) 注意:如果已存在的hash表有时间,这里将会替换原有的时间

\* @return true 成功 false失败

\*/

public boolean hset(String key, String item, Object value, long time) {

try {

redisTemplate.opsForHash().put(key, item, value);

if (time > 0) {

expire(key, time);

}

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 删除hash表中的值

\* @param key 键 不能为null

\* @param item 项 可以使多个 不能为null

\*/

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 键 不能为null

\* @param item 项 不能为null

\* @return true 存在 false不存在

\*/

public boolean hHasKey(String key, String item) {

return redisTemplate.opsForHash().hasKey(key, item);

}

/\*\*

\* hash递增 如果不存在,就会创建一个 并把新增后的值返回

\* @param key 键

\* @param item 项

\* @param by 要增加几(大于0)

\* @return

\*/

public double hincr(String key, String item, double by) {

return redisTemplate.opsForHash().increment(key, item, by);

}

/\*\*

\* hash递减

\* @param key 键

\* @param item 项

\* @param by 要减少记(小于0)

\* @return

\*/

public double hdecr(String key, String item, double by) {

return redisTemplate.opsForHash().increment(key, item, -by);

}

/\*\*

\* 根据key获取Set中的所有值

\* @param key 键

\* @return

\*/

public Set<Object> sGet(String key) {

try {

return redisTemplate.opsForSet().members(key);

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

/\*\*

\* 根据value从一个set中查询,是否存在

\* @param key 键

\* @param value 值

\* @return true 存在 false不存在

\*/

public boolean sHasKey(String key, Object value) {

try {

return redisTemplate.opsForSet().isMember(key, value);

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 将数据放入set缓存

\* @param key 键

\* @param values 值 可以是多个

\* @return 成功个数

\*/

public long sSet(String key, Object... values) {

try {

return redisTemplate.opsForSet().add(key, values);

} catch (Exception e) {

e.printStackTrace();

return 0;

}

}

/\*\*

\* 将set数据放入缓存

\* @param key 键

\* @param time 时间(秒)

\* @param values 值 可以是多个

\* @return 成功个数

\*/

public long sSetAndTime(String key, long time, Object... values) {

try {

Long count = redisTemplate.opsForSet().add(key, values);

if (time > 0){

expire(key, time);

}

return count;

} catch (Exception e) {

e.printStackTrace();

return 0;

}

}

/\*\*

\* 获取set缓存的长度

\* @param key 键

\* @return

\*/

public long sGetSetSize(String key) {

try {

return redisTemplate.opsForSet().size(key);

} catch (Exception e) {

e.printStackTrace();

return 0;

}

}

/\*\*

\* 移除值为value的

\* @param key 键

\* @param values 值 可以是多个

\* @return 移除的个数

\*/

public long setRemove(String key, Object... values) {

try {

Long count = redisTemplate.opsForSet().remove(key, values);

return count;

} catch (Exception e) {

e.printStackTrace();

return 0;

}

}

// ===============================list=================================

/\*\*

\* 获取list缓存的内容

\* @param key 键

\* @param start 开始

\* @param end 结束 0 到 -1代表所有值

\* @return

\*/

public List<Object> lGet(String key, long start, long end) {

try {

return redisTemplate.opsForList().range(key, start, end);

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

/\*\*

\* 获取list缓存的长度

\* @param key 键

\* @return

\*/

public long lGetListSize(String key) {

try {

return redisTemplate.opsForList().size(key);

} catch (Exception e) {

e.printStackTrace();

return 0;

}

}

/\*\*

\* 通过索引 获取list中的值

\* @param key 键

\* @param index 索引 index>=0时， 0 表头，1 第二个元素，依次类推；index<0时，-1，表尾，-2倒数第二个元素，依次类推

\* @return

\*/

public Object lGetIndex(String key, long index) {

try {

return redisTemplate.opsForList().index(key, index);

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

/\*\*

\* 将list放入缓存

\* @param key 键

\* @param value 值

\* @return

\*/

public boolean lSet(String key, Object value) {

try {

redisTemplate.opsForList().rightPush(key, value);

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 将list放入缓存

\* @param key 键

\* @param value 值

\* @param time 时间(秒)

\* @return

\*/

public boolean lSet(String key, Object value, long time) {

try {

redisTemplate.opsForList().rightPush(key, value);

if (time > 0){

expire(key, time);

}

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 将list放入缓存

\* @param key 键

\* @param value 值

\* @return

\*/

public boolean lSet(String key, List<Object> value) {

try {

redisTemplate.opsForList().rightPushAll(key, value);

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 将list放入缓存

\*

\* @param key 键

\* @param value 值

\* @param time 时间(秒)

\* @return

\*/

public boolean lSet(String key, List<Object> value, long time) {

try {

redisTemplate.opsForList().rightPushAll(key, value);

if (time > 0){

expire(key, time);

}

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 根据索引修改list中的某条数据

\* @param key 键

\* @param index 索引

\* @param value 值

\* @return

\*/

public boolean lUpdateIndex(String key, long index, Object value) {

try {

redisTemplate.opsForList().set(key, index, value);

return true;

} catch (Exception e) {

e.printStackTrace();

return false;

}

}

/\*\*

\* 移除N个值为value

\* @param key 键

\* @param count 移除多少个

\* @param value 值

\* @return 移除的个数

\*/

public long lRemove(String key, long count, Object value) {

try {

Long remove = redisTemplate.opsForList().remove(key, count, value);

return remove;

} catch (Exception e) {

e.printStackTrace();

return 0;

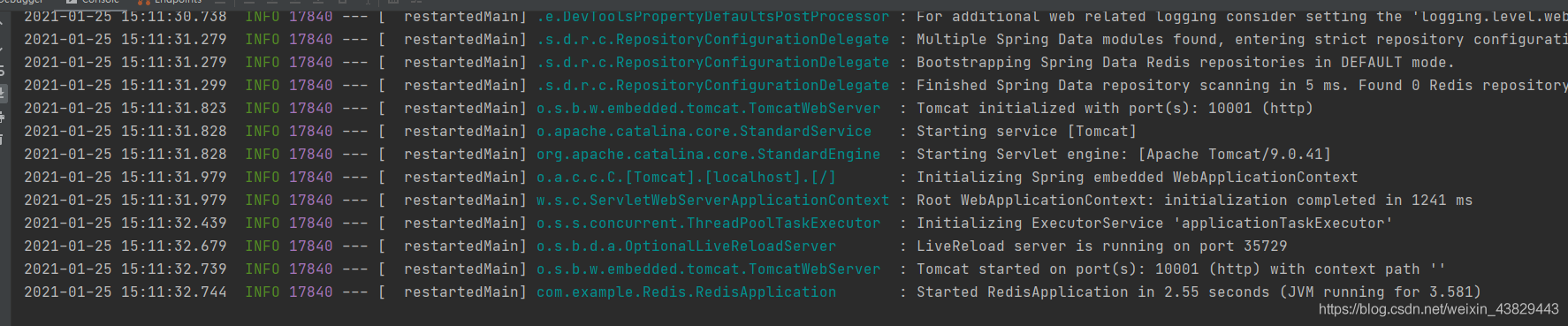
}

}

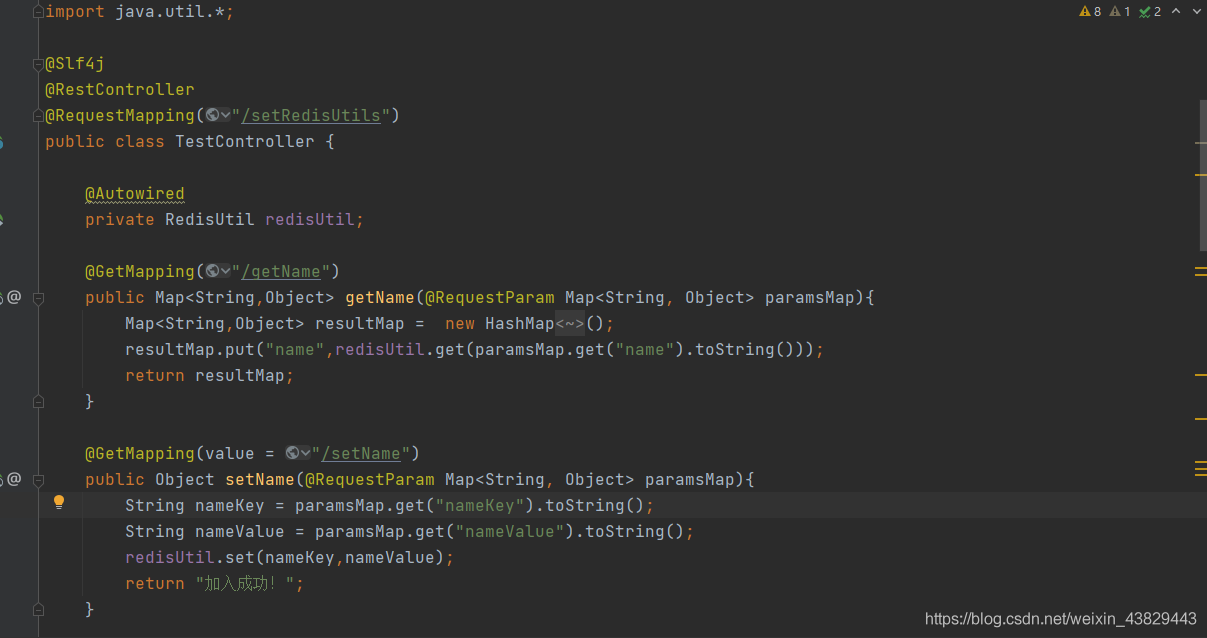
}

**使用RedisUtils**

①首先将springboot项目启动起来：



②添加Contoller在web上测试添加和获取！



③测试工具类的基本使用：

在谷歌浏览器输入：返回成功

测试获取：获取成功！

