链滴

Spring-RedisTemplate 写入数据乱码问题的复现与解决

作者：liumapp

原文链接：https://ld246.com/article/1565075590346

来源网站：链滴

org.springframework.data.redis是Spring框架对Redis的默认集成，我们在实际项目中，也经常使它的RedisTemplate去操作Redis，一般来说没什么问题，但是细心一点的同学会发现，经过这种方写入redis的数据会出现乱码问题

问题复现

项目依赖

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

Redis配置

- yaml文件配置

```
spring:
  application:
    name: booklet-redis
  redis:
    host: 127.0.0.1
    port: 6379
    password: adminadmin
    timeout: 5000ms
```

- Redis配置类

```
package com.liumapp.booklet.redis.config;

import org.springframework.beans.factory.annotation.Autowired;
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.*;
import org.springframework.data.redis.serializer.JdkSerializationRedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;

@Configuration
public class RedisConfig {

    /**
```

```
     * 注入 RedisConnectionFactory
     */
    @Autowired
    RedisConnectionFactory redisConnectionFactory;

    @Bean
    public RedisTemplate<String,Object> functionDomainRedisTemplate() {
        RedisTemplate<String,Object> redisTemplate = new RedisTemplate<>();
        initDomainRedisTemplate(redisTemplate, redisConnectionFactory);
        return redisTemplate;
    }

    /**
     * 设置数据存入 redis 的序列化方式
     *
     * @param redisTemplate
     * @param factory
     */
    private void initDomainRedisTemplate(RedisTemplate<String,Object> redisTemplate, Redi
ConnectionFactory factory) {
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setHashKeySerializer(new StringRedisSerializer());
        redisTemplate.setHashValueSerializer(new JdkSerializationRedisSerializer());
        redisTemplate.setValueSerializer(new JdkSerializationRedisSerializer());
        redisTemplate.setConnectionFactory(factory);
    }
    /**
     * 实例化 HashOperations 对象,可以使用 Hash 类型操作
     *
     * @param redisTemplate
     * @return
     */
    @Bean
    public HashOperations<String, String, Object> hashOperations(RedisTemplate<String, Obj
ct> redisTemplate) {
        return redisTemplate.opsForHash();
    }

    /**
     * 实例化 ValueOperations 对象,可以使用 String 操作
     *
     * @param redisTemplate
     * @return
     */
    @Bean
    public ValueOperations<String, Object> valueOperations(RedisTemplate<String, Object> r
disTemplate) {
        return redisTemplate.opsForValue();
    }

    /**
     * 实例化 ListOperations 对象,可以使用 List 操作
     *
     * @param redisTemplate
```

```
     * @return
     */
    @Bean
    public ListOperations<String, Object> listOperations(RedisTemplate<String, Object> redis
emplate) {
        return redisTemplate.opsForList();

    }

    /**
     * 实例化 SetOperations 对象,可以使用 Set 操作
     *
     * @param redisTemplate
     * @return
     */
    @Bean
    public SetOperations<String, Object> setOperations(RedisTemplate<String, Object> redis
emplate) {
        return redisTemplate.opsForSet();
    }

    /**
     * 实例化 ZSetOperations 对象,可以使用 ZSet 操作
     *
     * @param redisTemplate
     * @return
     */
    @Bean
    public ZSetOperations<String, Object> zSetOperations(RedisTemplate<String, Object> red
sTemplate) {
        return redisTemplate.opsForZSet();
    }
}
```

测试代码

```
package com.liumapp.booklet.redis;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.redis.core.ListOperations;
import org.springframework.test.context.junit4.SpringRunner;

import javax.annotation.Resource;
import java.util.ArrayList;
import java.util.List;

@SpringBootTest(classes = BookletRedisMain.class)
@RunWith(SpringRunner.class)
public class BookletRedisMainTest {

    @Resource
```

```
    private ListOperations<String, Object> listOperations;

    @Test
    public void leftPushTest () {
        List<String> list = new ArrayList<>();
        list.add("hello world");
        listOperations.leftPush("listKey", list);
    }

}
```

运行上述测试代码后，我们会在redis中插入一组list类型的数据，其key为listKey，value为只有一个素的list对象

接下来我们通过redis-cli去获取listKey这个值，可以看到乱码的出现：

```
127.0.0.1:6379> LRANGE listKey 0 10
1) "\xac\xed\x00\x05sr\x00\x13java.util.ArrayListx\x81\xd2\x1d\x99\xc7a\x9d\x03\x00\x01I\x
0\x04sizexp\x00\x00\x00\x01w\x04\x00\x00\x00\x01t\x00\x0bhello worldx"
```

当然，这对于我们项目的实际使用没有什么影响，在程序中再次获取listKey也不会出现乱码，只有通redis-cli等工具直接取值的时候，才会出现乱码

问题出现原因

问题原因在于我们对Redis进行配置的这一段代码（事实上这也是redisTemplate的默认配置代码）：

```
private void initDomainRedisTemplate(RedisTemplate<String,Object> redisTemplate, RedisC
nnectionFactory factory) {
    redisTemplate.setKeySerializer(new StringRedisSerializer());
    redisTemplate.setHashKeySerializer(new StringRedisSerializer());
    redisTemplate.setHashValueSerializer(new JdkSerializationRedisSerializer());
    redisTemplate.setValueSerializer(new JdkSerializationRedisSerializer());
    redisTemplate.setConnectionFactory(factory);
}
```

在这里，redisTemplate对HashValue和Value的序列化类采用的是JDK默认的序列化策略，而不是Strng类型的序列化策略，所以我们在redis-cli中看到的value会因为序列化策略的问题，出现乱码

解决办法

将JDK默认的序列化策略更换为String类型的序列化策略

```
redisTemplate.setHashValueSerializer(new StringRedisSerializer());
redisTemplate.setValueSerializer(new StringRedisSerializer());
```

但是这样做的话，我们在进行存储的时候，也只能存储String类型的数据，所以测试代码要进行如下改

```
@Test
public void leftPushTest () {
    List<String> list = new ArrayList<>();
    list.add("hello world2");
```

```
    listOperations.leftPush("listKey", list.toString());
}
```

再一次去redis-cli中取值，得到如下结果：

```
127.0.0.1:6379> LRANGE listKey 0 10
1) "[hello world2]"
2) "\xac\xed\x00\x05sr\x00\x13java.util.ArrayListx\x81\xd2\x1d\x99\xc7a\x9d\x03\x00\x01I\x
0\x04sizexp\x00\x00\x00\x01w\x04\x00\x00\x00\x01t\x00\x0bhello worldx"
```

可以发现乱码问题已经解决

总结

不建议更换redisTemplate默认的序列化策略，有乱码就让它乱着吧，反正知道正确的解码策略就不影响程序的正常运行（不过通过php等其他语言去获取redis的值貌似不太好解决）

如果一定要更换策略，那么前往要注意，存储数据的类型要根据所选择的序列化策略去进行切换

项目案例源代码：github/booklet-redis