在线oj项目-redis引入

- **安装redis**

```
docker pull redis
```

- **启动redis容器**

```
docker run --name oj-redis -d -p 6379:6379 redis --requirepass "123456"
```

- **创建oj-common-redis工程**
- **引入redis依赖**

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

        <!-- Alibaba Fastjson -->
        <dependency>
            <groupId>com.alibaba.fastjson2</groupId>
            <artifactId>fastjson2</artifactId>
            <version>2.0.43</version>
        </dependency>
```

- **提供redis配置**

```
package com.bite.common.redis.config;

import com.alibaba.fastjson2.JSON;
import org.springframework.data.redis.serializer.RedisSerializer;
import org.springframework.data.redis.serializer.SerializationException;

import java.nio.charset.Charset;
```

```
8
9
10 public class JsonRedisSerializer<T> implements RedisSerializer<T> {
11     public static final Charset DEFAULT_CHARSET = Charset.forName("UTF-8");
12
13     private Class<T> clazz;
14
15
16     public JsonRedisSerializer(Class<T> clazz) {
17         super();
18         this.clazz = clazz;
19     }
20
21     @Override
22     public byte[] serialize(T t) throws SerializationException {
23         if (t == null) {
24             return new byte[0];
25         }
26         return JSON.toJSONString(t).getBytes(DEFAULT_CHARSET);
27     }
28     @Override
29     public T deserialize(byte[] bytes) throws SerializationException {
30         if (bytes == null || bytes.length <= 0) {
31             return null;
32         }
33         String str = new String(bytes, DEFAULT_CHARSET);
34
35         return JSON.parseObject(str, clazz);
36     }
37 }
```

```
1 package com.bite.common.redis.config;
2
3 import org.springframework.boot.autoconfigure.AutoConfigureBefore;
4 import
  org.springframework.boot.autoconfigure.data.redis.RedisAutoConfiguration;
5 import org.springframework.cache.annotation.CachingConfigurerSupport;
6 import org.springframework.cache.annotation.EnableCaching;
7 import org.springframework.context.annotation.Bean;
8 import org.springframework.context.annotation.Configuration;
9 import org.springframework.data.redis.connection.RedisConnectionFactory;
10 import org.springframework.data.redis.core.RedisTemplate;
11 import org.springframework.data.redis.serializer.StringRedisSerializer;
12
13 /**
```

```
 * redis配置
 */
@Configuration
public class RedisConfig extends CachingConfigurerSupport {
    @Bean
    public RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory connectionFactory) {
        RedisTemplate<Object, Object> template = new RedisTemplate<>();
        template.setConnectionFactory(connectionFactory);

        JsonRedisSerializer serializer = new JsonRedisSerializer(Object.class);

        // 使用StringRedisSerializer来序列化和反序列化redis的key值
        template.setKeySerializer(new StringRedisSerializer());
        template.setValueSerializer(serializer);

        // Hash的key也采用StringRedisSerializer的序列化方式
        template.setHashKeySerializer(new StringRedisSerializer());
        template.setHashValueSerializer(serializer);

        template.afterPropertiesSet();
        return template;
    }
}
```

- **封装service**
  - 为什么封装service：

    抽象与解耦：封装第三方组件可以提供一个更高级的抽象层，使得你的代码与具体的第三方实现解耦。这样，如果将来需要更换第三方组件或调整其配置，你只需要修改封装的service层，而不需要修改整个应用中的大量代码。

    统一接口：即使多个第三方工具提供相似的功能，它们的API和用法也可能各不相同。通过封装，我们可以提供一个统一的接口，使得其他开发者无需关心底层工具的具体差异。

    扩展性：通过封装，我们可以更容易地为第三方工具添加额外的功能或逻辑。以满足项目的特定的需求。

    错误处理与异常管理：第三方工具可能会抛出特定的异常或错误。通过封装，我们可以统一处理这些错误，并将它们转换为更通用或更有意义的异常，这样其他开发者就可以更容易地理解和处理这些错误。

    代码可读性与维护性：使用封装的service可以使代码更加清晰和易于理解，因为你可以为service层提供有意义的名称和文档，以便其他开发者知道如何使用它以及它的功能。同时，如果将来有新人加入项目，他们也可以更容易地理解和使用封装的service。

```java
package com.bite.common.redis.service;

import com.alibaba.fastjson2.JSON;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.ValueOperations;
import org.springframework.stereotype.Component;
import org.springframework.util.CollectionUtils;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;

@Component
public class RedisService {

    @Autowired
    public RedisTemplate redisTemplate;

    //************************ 操作key ***************************

    /**
     * 判断 key是否存在
     *
     * @param key 键
     * @return true 存在 false不存在
     */
    public Boolean hasKey(String key) {
        return redisTemplate.hasKey(key);
    }


    /**
     * 设置有效时间
     *
     * @param key     Redis键
     * @param timeout 超时时间
     * @return true=设置成功；false=设置失败
     */
    public boolean expire(final String key, final long timeout) {
        return expire(key, timeout, TimeUnit.SECONDS);
    }

```

```
48      /**
49       * 设置有效时间
50       *
51       * @param key     Redis键
52       * @param timeout 超时时间
53       * @param unit    时间单位
54       * @return true=设置成功；false=设置失败
55       */
56      public boolean expire(final String key, final long timeout, final TimeUnit
    unit) {
57          return redisTemplate.expire(key, timeout, unit);
58      }
59
60      /**
61       * 删除单个对象
62       *
63       * @param key
64       */
65      public boolean deleteObject(final String key) {
66          return redisTemplate.delete(key);
67      }
68
69      //************************ 操作String类型 ***************************
70
71      /**
72       * 缓存基本的对象，Integer、String、实体类等
73       *
74       * @param key   缓存的键值
75       * @param value 缓存的值
76       */
77      public <T> void setCacheObject(final String key, final T value) {
78          redisTemplate.opsForValue().set(key, value);
79      }
80
81      /**
82       * 缓存基本的对象，Integer、String、实体类等
83       *
84       * @param key      缓存的键值
85       * @param value    缓存的值
86       * @param timeout  时间
87       * @param timeUnit 时间颗粒度
88       */
89      public <T> void setCacheObject(final String key, final T value, final Long
    timeout, final TimeUnit timeUnit) {
90          redisTemplate.opsForValue().set(key, value, timeout, timeUnit);
91      }
92
```

```
93      /**
94       * 获得缓存的基本对象。
95       *
96       * @param key 缓存键值
97       * @return 缓存键值对应的数据
98       */
99      public <T> T getCacheObject(final String key, Class<T> clazz) {
100         ValueOperations<String, T> operation = redisTemplate.opsForValue();
101         T t = operation.get(key);
102         if (t instanceof String) {
103             return t;
104         }
105         return JSON.parseObject(String.valueOf(t), clazz);
106     }
107
108     //*************** 操作list结构 ***************
109
110     /**
111      * 获取list中存储数据数量
112      *
113      * @param key
114      * @return
115      */
116     public Long getListSize(final String key) {
117         return redisTemplate.opsForList().size(key);
118     }
119
120     /**
121      * 获取list中指定范围数据
122      *
123      * @param key
124      * @param start
125      * @param end
126      * @param clazz
127      * @param <T>
128      * @return
129      */
130     public <T> List<T> getCacheListByRange(final String key, long start, long
    end, Class<T> clazz) {
131         List range = redisTemplate.opsForList().range(key, start, end);
132         if (CollectionUtils.isEmpty(range)) {
133             return null;
134         }
135         return JSON.parseArray(JSON.toJSONString(range), clazz);
136     }
137
138     /**
```

```
139      * 底层使用list结构存储数据(尾插 批量插入)
140      */
141     public <T> Long rightPushAll(final String key, Collection<T> list) {
142         return redisTemplate.opsForList().rightPushAll(key, list);
143     }
144 
145     /**
146      * 底层使用list结构存储数据(头插)
147      */
148     public <T> Long leftPushForList(final String key, T value) {
149         return redisTemplate.opsForList().leftPush(key, value);
150     }
151 
152     /**
153      * 底层使用list结构,删除指定数据
154      */
155     public <T> Long removeForList(final String key, T value) {
156         return redisTemplate.opsForList().remove(key, 1L, value);
157     }
158 
159 
160     //************************ 操作Hash类型 ***************************
161     public <T> T getCacheMapValue(final String key, final String hKey,
    Class<T> clazz) {
162         Object cacheMapValue = redisTemplate.opsForHash().get(key, hKey);
163         if (cacheMapValue != null) {
164             return JSON.parseObject(String.valueOf(cacheMapValue), clazz);
165         }
166         return null;
167     }
168 
169 
170     /**
171      * 获取多个Hash中的数据
172      *
173      * @param key   Redis键
174      * @param hKeys Hash键集合
175      * @param clazz 待转换对象类型
176      * @param <T>   泛型
177      * @return Hash对象集合
178      */
179     public <T> List<T> getMultiCacheMapValue(final String key, final
    Collection<String> hKeys, Class<T> clazz) {
180         List list = redisTemplate.opsForHash().multiGet(key, hKeys);
181         List<T> result = new ArrayList<>();
182         for (Object item : list) {
183             result.add(JSON.parseObject(JSON.toJSONString(item), clazz));
```

```
184         }
185
186         return result;
187     }
188
189     /**
190      * 往Hash中存入数据
191      *
192      * @param key   Redis键
193      * @param hKey  Hash键
194      * @param value 值
195      */
196     public <T> void setCacheMapValue(final String key, final String hKey, final
    T value) {
197         redisTemplate.opsForHash().put(key, hKey, value);
198     }
199
200     /**
201      * 缓存Map
202      *
203      * @param key
204      * @param dataMap
205      */
206     public <K, T> void setCacheMap(final String key, final Map<K, T> dataMap) {
207         if (dataMap != null) {
208             redisTemplate.opsForHash().putAll(key, dataMap);
209         }
210     }
211
212     public Long deleteCacheMapValue(final String key, final String hKey) {
213         return redisTemplate.opsForHash().delete(key, hKey);
214     }
215 }
```

- **创建org.springframework.boot.autoconfigure.AutoConfiguration.imports文件**

```
1 com.bite.common.redis.service.RedisService
2 com.bite.common.redis.config.RedisConfig
```