



Spring Authorization Server优化篇：添加Redis缓存支持和统一响应类

叹雪飞花 2023-07-08 1,732 阅读15分钟

关注

前言

今天为大家展示一下如何使用Spring data redis来缓存项目中数据，在项目使用人数少的情况下使用HttpSession问题不大，但是当并发多了就顶不住了，基本都会选择一些NoSQL来做缓存，本人就选择了比较常用的redis来做缓存；关于统一响应类这个东西就是为了规范项目的响应值，方便前端对接接口，其它人对接接口时更轻松。

添加统一响应类

在model包下添加Result.java类

▼ java 复制代码

```
1 package com.example.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6 import org.springframework.http.HttpStatus;
7
8 import java.io.Serializable;
9
10 /**
11  * 公共响应类
12  *
13  * @author vains
14  * @date 2021/3/10 14:10
15  */
16 @Data
```



```
20
21  /**
22   * 响应状态码
23   */
24   private Integer code;
25
26  /**
27   * 响应信息
28   */
29   private String message;
30
31  /**
32   * 接口是否处理成功
33   */
34   private Boolean success;
35
36  /**
37   * 接口响应时携带的数据
38   */
39   private T data;
40
41  /**
42   * 操作成功携带数据
43   * @param data 数据
44   * @param <T> 类型
45   * @return 返回统一响应
46   */
47   public static <T> Result<T> success(T data) {
48       return new Result<>(HttpStatus.OK.value(), ("操作成功."), Boolean.TRUE, data);
49   }
50
51  /**
52   * 操作成功不带数据
53   * @return 返回统一响应
54   */
55   public static Result<String> success() {
56       return new Result<>(HttpStatus.OK.value(), ("操作成功."), Boolean.TRUE, (null));
57   }
58
59  /**
60   * 操作成功携带数据
61   * @param message 成功提示消息
62   * @param data 成功携带数据
63   * @param <T> 类型
64   * @return 返回统一响应
65   */
```



```
69
70 /**
71  * 操作失败返回
72  * @param message 成功提示消息
73  * @param <T> 类型
74  * @return 返回统一响应
75  */
76 public static <T> Result<T> error(String message) {
77     return new Result<>(HttpStatus.INTERNAL_SERVER_ERROR.value(), message, Boolean.FALSE, (n
78 }
79
80 /**
81  * 操作失败返回
82  * @param code 错误码
83  * @param message 成功提示消息
84  * @param <T> 类型
85  * @return 返回统一响应
86  */
87 public static <T> Result<T> error(Integer code, String message) {
88     return new Result<>(code, message, Boolean.FALSE, (null));
89 }
90
91 /**
92  * oauth2 问题
93  * @param message 失败提示消息
94  * @param data 具体的错误信息
95  * @param <T> 类型
96  * @return 返回统一响应
97  */
98 public static <T> Result<T> oauth2Error(Integer code, String message, T data) {
99     return new Result<>(code, message, Boolean.FALSE, data);
100 }
101
102 /**
103  * oauth2 问题
104  * @param message 失败提示消息
105  * @param data 具体的错误信息
106  * @param <T> 类型
107  * @return 返回统一响应
108  */
109 public static <T> Result<T> oauth2Error(String message, T data) {
110     return new Result<>(HttpStatus.UNAUTHORIZED.value(), message, Boolean.FALSE, data);
111 }
112
113 }
```

在controller包下添加 **LoginController** 类

该类中的接口是原 **AuthorizationController** 接口中的，现在挪到该类中，并使用Redis来替换HttpSession存储验证码信息，编写一个 **CaptchaResult** 来将redis中的key返回给前端，前端登录时携带这个key来获取缓存数据,CaptchaResult类在后边

java 复制代码

```
1 package com.example.controller;
2
3 import cn.hutool.captcha.CaptchaUtil;
4 import cn.hutool.captcha.ShearCaptcha;
5 import com.baomidou.mybatisplus.core.toolkit.IdWorker;
6 import com.example.model.Result;
7 import com.example.model.response.CaptchaResult;
8 import com.example.support.RedisOperator;
9 import lombok.RequiredArgsConstructor;
10 import org.springframework.web.bind.annotation.GetMapping;
11 import org.springframework.web.bind.annotation.RestController;
12
13 import static com.example.constant.RedisConstants.*;
14
15 /**
16  * 登录接口，登录使用的接口
17  *
18  * @author vains
19  */
20 @RestController
21 @RequiredArgsConstructor
22 public class LoginController {
23
24     private final RedisOperator<String> redisOperator;
25
26     @GetMapping("/getSmsCaptcha")
27     public Result<String> getSmsCaptcha(String phone) {
28         // 示例项目，固定1234
29         String smsCaptcha = "1234";
30         // 存入缓存中，5分钟后过期
31         redisOperator.set((SMS_CAPTCHA_PREFIX_KEY + phone), smsCaptcha, CAPTCHA_TIMEOUT_SECONDS);
32         return Result.success("获取短信验证码成功.", smsCaptcha);
33     }
34
35     @GetMapping("/getCaptcha")
```

```
39     ShearCaptcha captcha = CaptchaUtil.createShearCaptcha(150, 40, 4, 2);
40     // 生成一个唯一id
41     long id = IdWorker.getId();
42     // 存入缓存中, 5分钟后过期
43     redisOperator.set((IMAGE_CAPTCHA_PREFIX_KEY + id), captcha.getCode(), CAPTCHA_TIMEOUT_SECONDS);
44     return Result.success("获取验证码成功.", new CaptchaResult(String.valueOf(id), captcha.getCode()));
45 }
46
47 }
```

CaptchaResult

生成的key是long，类使用String是因为前端对于long类型的大数据会有精度丢失问题，所以使用String



java 复制代码

```
1 package com.example.model.response;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5
6 /**
7  * 获取验证码返回
8  *
9  * @author vains
10  */
11 @Data
12 @AllArgsConstructor
13 public class CaptchaResult {
14
15     /**
16      * 验证码id
17      */
18     private String captchaId;
19
20     /**
21      * 验证码的值
22      */
23     private String code;
24
25     /**
```



```
29
30 }
```

优化登录页面

页面修改不大，这里只放关键代码，如果有需要的可以去gitee查看完整代码：[代码地址](#)
页面登录表单添加隐藏域，存储redis存储验证码的key

html 复制代码

```
1 <input type="hidden" id="captchaId" name="captchaId" value=""/>
```

获取验证码并设置值的地方修改

javascript 复制代码

```
1 function getVerifyCode() {
2     let requestOptions = {
3         method: 'GET',
4         redirect: 'follow'
5     };
6
7     fetch(`${window.location.origin}/getCaptcha`, requestOptions)
8         .then(response => response.text())
9         .then(r => {
10             if (r) {
11                 let result = JSON.parse(r);
12                 document.getElementById('captchaId').value = result.data.captchaId
13                 document.getElementById('code-image').src = result.data.imageData
14             }
15         })
16         .catch(error => console.log('error', error));
17 }
```

修改 `CaptchaAuthenticationProvider` 从redis中获取验证码

java 复制代码



```
3 import com.example.constant.SecurityConstants;
4 import com.example.exception.InvalidCaptchaException;
5 import com.example.support.RedisOperator;
6 import jakarta.servlet.http.HttpServletRequest;
7 import lombok.extern.slf4j.Slf4j;
8 import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
9 import org.springframework.security.core.Authentication;
10 import org.springframework.security.core.AuthenticationException;
11 import org.springframework.security.core.userdetails.UserDetailsService;
12 import org.springframework.security.crypto.password.PasswordEncoder;
13 import org.springframework.stereotype.Component;
14 import org.springframework.util.ObjectUtils;
15 import org.springframework.web.context.request.RequestAttributes;
16 import org.springframework.web.context.request.RequestContextHolder;
17 import org.springframework.web.context.request.ServletRequestAttributes;
18
19 import java.util.Objects;
20
21 import static com.example.constant.RedisConstants.IMAGE_CAPTCHA_PREFIX_KEY;
22
23 /**
24  * 验证码校验
25  * 注入ioc中替换原先的DaoAuthenticationProvider
26  * 在authenticate方法中添加校验验证码的逻辑
27  * 最后调用父类的authenticate方法并返回
28  *
29  * @author vains
30  */
31 @Slf4j
32 public class CaptchaAuthenticationProvider extends DaoAuthenticationProvider {
33
34     private final RedisOperator<String> redisOperator;
35
36     /**
37      * 利用构造方法在通过{@link Component}注解初始化时
38      * 注入UserDetailsService和passwordEncoder，然后
39      * 设置调用父类关于这两个属性的set方法设置进去
40      *
41      * @param userDetailsService 用户服务，给框架提供用户信息
42      * @param passwordEncoder 密码解析器，用于加密和校验密码
43      */
44     public CaptchaAuthenticationProvider(UserDetailsService userDetailsService, PasswordEncoder
45         this.redisOperator = redisOperator;
46         super.setPasswordEncoder(passwordEncoder);
47         super.setUserDetailsService(userDetailsService);
48     }
49
```



```
53
54     // 获取当前request
55     RequestAttributes requestAttributes = RequestContextHolder.getRequestAttributes();
56     if (requestAttributes == null) {
57         throw new InvalidCaptchaException("Failed to get the current request.");
58     }
59     HttpServletRequest request = ((ServletRequestAttributes) requestAttributes).getRequest()
60
61     // 获取当前登录方式
62     String loginType = request.getParameter(SecurityConstants.LOGIN_TYPE_NAME);
63     if (!Objects.equals(loginType, SecurityConstants.PASSWORD_LOGIN_TYPE)) {
64         // 只要不是密码登录都不需要校验图形验证码
65         log.info("It isn't necessary captcha authenticate.");
66         return super.authenticate(authentication);
67     }
68
69     // 获取参数中的验证码
70     String code = request.getParameter(SecurityConstants.CAPTCHA_CODE_NAME);
71     if (ObjectUtils.isEmpty(code)) {
72         throw new InvalidCaptchaException("The captcha cannot be empty.");
73     }
74
75     String captchaId = request.getParameter(SecurityConstants.CAPTCHA_ID_NAME);
76     // 获取缓存中存储的验证码
77     String captchaCode = redisOperator.getAndDelete((IMAGE_CAPTCHA_PREFIX_KEY + captchaId));
78     if (!ObjectUtils.isEmpty(captchaCode)) {
79         if (!captchaCode.equalsIgnoreCase(code)) {
80             throw new InvalidCaptchaException("The captcha is incorrect.");
81         }
82     } else {
83         throw new InvalidCaptchaException("The captcha is abnormal. Obtain it again.");
84     }
85
86     log.info("Captcha authenticated.");
87     return super.authenticate(authentication);
88 }
89 }
```

修改 SmsCaptchaLoginAuthenticationProvider 从redis中获取短信验证码

[java](#) 复制代码



```
3 import com.example.authorization.captcha.CaptchaAuthenticationProvider;
4 import com.example.constant.SecurityConstants;
5 import com.example.exception.InvalidCaptchaException;
6 import com.example.support.RedisOperator;
7 import jakarta.servlet.http.HttpServletRequest;
8 import lombok.extern.slf4j.Slf4j;
9 import org.springframework.security.authentication.BadCredentialsException;
10 import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
11 import org.springframework.security.core.AuthenticationException;
12 import org.springframework.security.core.userdetails.UserDetails;
13 import org.springframework.security.core.userdetails.UserDetailsService;
14 import org.springframework.security.crypto.password.PasswordEncoder;
15 import org.springframework.security.oauth2.core.endpoint.OAuth2ParameterNames;
16 import org.springframework.stereotype.Component;
17 import org.springframework.web.context.request.RequestAttributes;
18 import org.springframework.web.context.request.RequestContextHolder;
19 import org.springframework.web.context.request.ServletRequestAttributes;
20
21 import java.util.Objects;
22
23 import static com.example.constant.RedisConstants.SMS_CAPTCHA_PREFIX_KEY;
24
25 /**
26  * 短信验证码校验实现
27  *
28  * @author vains
29  */
30 @Slf4j
31 @Component
32 public class SmsCaptchaLoginAuthenticationProvider extends CaptchaAuthenticationProvider {
33
34     private final RedisOperator<String> redisOperator;
35
36     /**
37      * 利用构造方法在通过{@link Component}注解初始化时
38      * 注入UserDetailsService和passwordEncoder，然后
39      * 设置调用父类关于这两个属性的set方法设置进去
40      *
41      * @param userDetailsService 用户服务，给框架提供用户信息
42      * @param passwordEncoder 密码解析器，用于加密和校验密码
43      */
44     public SmsCaptchaLoginAuthenticationProvider(UserDetailsService userDetailsService, Password
45         super(userDetailsService, passwordEncoder, redisOperator);
46         this.redisOperator = redisOperator;
47     }
48
49     @Override
```

```
53     if (authentication.getCredentials() == null) {
54         this.logger.debug("Failed to authenticate since no credentials provided");
55         throw new BadCredentialsException("The sms captcha cannot be empty.");
56     }
57
58     // 获取当前request
59     RequestAttributes requestAttributes = RequestContextHolder.getRequestAttributes();
60     if (requestAttributes == null) {
61         throw new InvalidCaptchaException("Failed to get the current request.");
62     }
63     HttpServletRequest request = ((ServletRequestAttributes) requestAttributes).getRequest()
64
65     // 获取当前登录方式
66     String loginType = request.getParameter(SecurityConstants.LOGIN_TYPE_NAME);
67     // 获取grant_type
68     String grantType = request.getParameter(OAuth2ParameterNames.GRANT_TYPE);
69     // 短信登录和自定义短信认证grant_type会走下方认证
70     // 如果是自定义密码模式则下方的认证判断只要判断下LoginType即可
71     // if (Objects.equals(loginType, SecurityConstants.SMS_LOGIN_TYPE)) {}
72     if (Objects.equals(loginType, SecurityConstants.SMS_LOGIN_TYPE)
73         || Objects.equals(grantType, SecurityConstants.GRANT_TYPE_SMS_CODE)) {
74         // 获取存入缓存中的验证码(UsernamePasswordAuthenticationToken的principal中现在存入的是手
75         String smsCaptcha = redisOperator.getAndDelete((SMS_CAPTCHA_PREFIX_KEY + authenticat
76         // 校验输入的验证码是否正确(UsernamePasswordAuthenticationToken的credentials中现在存入的
77         if (!Objects.equals(smsCaptcha, authentication.getCredentials())) {
78             throw new BadCredentialsException("The sms captcha is incorrect.");
79         }
80         // 在这里也可以拓展其它登录方式，比如邮箱登录什么的
81     } else {
82         log.info("Not sms captcha loginType, exit.");
83         // 其它调用父类默认实现的密码方式登录
84         super.additionalAuthenticationChecks(userDetails, authentication);
85     }
86
87     log.info("Authenticated sms captcha.");
88 }
89 }
```

修改 OAuth2BasicUser 的 authorities 属性，以防序列化失败

上一篇文章中最开始没有添加 `@JsonSerialize` 与 `@JsonIgnoreProperties(ignoreUnknown = true)` 注解，导致授权确认时框架获取用户信息反序列化时失败，会导致JsonMixin异常，

```
1 package com.example.entity;
2
3 import com.baomidou.mybatisplus.annotation.*;
4
5 import java.io.Serial;
6 import java.io.Serializable;
7 import java.time.LocalDateTime;
8 import java.util.Collection;
9
10 import com.example.model.security.CustomGrantedAuthority;
11 import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
12 import com.fasterxml.jackson.databind.annotation.JsonSerialize;
13 import lombok.Data;
14 import org.springframework.security.core.userdetails.UserDetails;
15
16 /**
17  * <p>
18  * 基础用户信息表
19  * </p>
20  *
21  * @author vains
22  */
23 @Data
24 @JsonSerialize
25 @TableName("oauth2_basic_user")
26 @JsonIgnoreProperties(ignoreUnknown = true)
27 public class Oauth2BasicUser implements UserDetails, Serializable {
28
29     @Serial
30     private static final long serialVersionUID = 1L;
31
32     /**
33      * 自增id
34      */
35     @TableId(value = "id", type = IdType.AUTO)
36     private Integer id;
37
38     /**
39      * 用户名、昵称
40      */
41     private String name;
42
43     /**
44      * 账号
45      */
46 }
```



```
49     * 密码
50     */
51     private String password;
52
53     /**
54     * 手机号
55     */
56     private String mobile;
57
58     /**
59     * 邮箱
60     */
61     private String email;
62
63     /**
64     * 头像地址
65     */
66     private String avatarUrl;
67
68     /**
69     * 是否已删除
70     */
71     private Boolean deleted;
72
73     /**
74     * 用户来源
75     */
76     private String sourceFrom;
77
78     /**
79     * 创建时间
80     */
81     @TableField(fill = FieldFill.INSERT)
82     private LocalDateTime createTime;
83
84     /**
85     * 修改时间
86     */
87     @TableField(fill = FieldFill.INSERT_UPDATE)
88     private LocalDateTime updateTime;
89
90     /**
91     * 权限信息
92     * 非数据库字段
93     */
94     @TableField(exist = false)
```



```
98     public Collection<CustomGrantedAuthority> getAuthorities() {  
99         return this.authorities;  
100     }  
101  
102     @Override  
103     public String getUsername() {  
104         return this.account;  
105     }  
106  
107     @Override  
108     public boolean isAccountNonExpired() {  
109         return true;  
110     }  
111  
112     @Override  
113     public boolean isAccountNonLocked() {  
114         return true;  
115     }  
116  
117     @Override  
118     public boolean isCredentialsNonExpired() {  
119         return true;  
120     }  
121  
122     @Override  
123     public boolean isEnabled() {  
124         return !this.deleted;  
125     }  
126 }
```

CustomGrantedAuthority

该类中添加@JsonSerialize也是为了解决JsonMixin问题

[java](#) 复制代码

```
1 package com.example.model.security;  
2  
3 import com.fasterxml.jackson.databind.annotation.JsonSerialize;  
4 import lombok.AllArgsConstructor;  
5 import lombok.Data;  
6 import lombok.NoArgsConstructor;  
7 import org.springframework.security.core.GrantedAuthority;  
8
```



```
12  * @author vains
13  */
14  @Data
15  @JsonSerialize
16  @NoArgsConstructor
17  @AllArgsConstructor
18  public class CustomGrantedAuthority implements GrantedAuthority {
19
20      private String authority;
21
22      @Override
23      public String getAuthority() {
24          return this.authority;
25      }
26  }
```

Redis常量类 RedisConstants



java 复制代码

```
1  package com.example.constant;
2
3  /**
4   * Redis相关常量
5   *
6   * @author vains
7   */
8  public class RedisConstants {
9
10     /**
11      * 短信验证码前缀
12      */
13     public static final String SMS_CAPTCHA_PREFIX_KEY = "mobile_phone:";
14
15     /**
16      * 图形验证码前缀
17      */
18     public static final String IMAGE_CAPTCHA_PREFIX_KEY = "image_captcha:";
19
20     /**
21      * 验证码过期时间，默认五分钟
22      */
23     public static final long CAPTCHA_TIMEOUT_SECONDS = 60L * 5;
```

SecurityConstants 添加常量

▼

java 复制代码

```
1  /**
2   * 登录方式入参名
3   */
4  public static final String LOGIN_TYPE_NAME = "loginType";
5
6  /**
7   * 验证码id入参名
8   */
9  public static final String CAPTCHA_ID_NAME = "captchaId";
10
11 /**
12 * 验证码值入参名
13 */
14 public static final String CAPTCHA_CODE_NAME = "code";
```

整合Spring data redis的步骤

- 1. 引入starter
- 2. 编写redis配置类(可选)
- 3. 编写redis操作类(可选)

引入starer

Spring boot项目中直接引入starter即可

▼

xml 复制代码

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

引入jackson-datatype-jsr310提供对Java8的特性与Java8时间相关序列化支持

```
2     <groupId>com.fasterxml.jackson.datatype</groupId>
3     <artifactId>jackson-datatype-jsr310</artifactId>
4 </dependency>
```

编写Redis配置文件

这个是可选的，但是如果不加这些序列化器会使用jdk的序列化器，导致使用redis客户端查看时与元数据有差异

2023-07-10更新：去除值序列化器，移除内容如下

▼

java 复制代码

```
1 // 设置值序列化器(该内容已移除)
2 redisTemplate.setValueSerializer(valueSerializer);
```

2023-09-23更新：添加 **Security** 提供的 **Json Mixin**，支持Jackson的值序列化器

▼

java 复制代码

```
1 package com.example.config;
2
3 import com.fasterxml.jackson.annotation.JsonAutoDetect;
4 import com.fasterxml.jackson.annotation.JsonTypeInfo;
5 import com.fasterxml.jackson.annotation.PropertyAccessor;
6 import com.fasterxml.jackson.databind.ObjectMapper;
7 import lombok.RequiredArgsConstructor;
8 import org.springframework.context.annotation.Bean;
9 import org.springframework.context.annotation.Configuration;
10 import org.springframework.data.redis.connection.RedisConnectionFactory;
11 import org.springframework.data.redis.core.RedisTemplate;
12 import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
13 import org.springframework.data.redis.serializer.StringRedisSerializer;
14 import org.springframework.http.converter.json.Jackson2ObjectMapperBuilder;
15 import org.springframework.security.jackson2.CoreJackson2Module;
16
17 /**
18  * Redis的key序列化配置类
19  *
20  * @author vains
21  */
22 @Configuration
```



```
26     private final Jackson2ObjectMapperBuilder builder;
27
28     /**
29      * 默认情况下使用
30      *
31      * @param connectionFactory redis链接工厂
32      * @return RedisTemplate
33      */
34     @Bean
35     public RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory connectionFactory)
36     {
37         // 字符串序列化器
38         StringRedisSerializer stringRedisSerializer = new StringRedisSerializer();
39
40         // 创建ObjectMapper并添加默认配置
41         ObjectMapper objectMapper = builder.createXmlMapper(false).build();
42
43         // 序列化所有字段
44         objectMapper.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
45
46         // 此项必须配置，否则如果序列化的对象里边还有对象，会报如下错误：
47         // java.lang.ClassCastException: java.util.LinkedHashMap cannot be cast to XXX
48         objectMapper.activateDefaultTyping(
49             objectMapper.getPolymorphicTypeValidator(),
50             ObjectMapper.DefaultTyping.NON_FINAL,
51             JsonTypeInfo.As.PROPERTY);
52
53         // 添加Security提供的Jackson Mixin
54         objectMapper.registerModule(new CoreJackson2Module());
55
56         // 存入redis时序列化值的序列化器
57         Jackson2JsonRedisSerializer<Object> valueSerializer =
58             new Jackson2JsonRedisSerializer<>(objectMapper, Object.class);
59
60         RedisTemplate<Object, Object> redisTemplate = new RedisTemplate<>();
61
62         // 设置值序列化
63         redisTemplate.setValueSerializer(valueSerializer);
64         // 设置hash格式数据值的序列化器
65         redisTemplate.setHashValueSerializer(valueSerializer);
66         // 默认的Key序列化器为: JdkSerializationRedisSerializer
67         redisTemplate.setKeySerializer(stringRedisSerializer);
68         // 设置字符串序列化器
69         redisTemplate.setStringSerializer(stringRedisSerializer);
70         // 设置hash结构的key的序列化器
71         redisTemplate.setHashKeySerializer(stringRedisSerializer);
```

```
75         return redisTemplate;
76     }
77
78     /**
79      * 操作hash的情况下使用
80      *
81      * @param connectionFactory redis链接工厂
82      * @return RedisTemplate
83      */
84     @Bean
85     public RedisTemplate<Object, Object> redisHashTemplate(RedisConnectionFactory connectionFact
86
87         return redisTemplate(connectionFactory);
88     }
89
90 }
```

编写redis操作类

可选，框架提供了RedisTemplate来操作redis



java 复制代码

```
1 package com.example.support;
2
3 import com.example.util.JsonUtils;
4 import jakarta.annotation.Resource;
5 import org.springframework.data.redis.core.HashOperations;
6 import org.springframework.data.redis.core.ListOperations;
7 import org.springframework.data.redis.core.RedisTemplate;
8 import org.springframework.data.redis.core.ValueOperations;
9 import org.springframework.stereotype.Component;
10 import org.springframework.util.ObjectUtils;
11
12 import java.util.Arrays;
13 import java.util.Collection;
14 import java.util.Map;
15 import java.util.concurrent.TimeUnit;
16
17 /**
18  * Redis操作类
19  *
20  * @param <V> value的类型
```



```
24 public class RedisOperator<V> {
25
26     /**
27      * 这里使用 @Resource 注解是因为在配置文件中注入ioc的泛型是<Object, Object>, 所以类型匹配不上,
28      * resource是会先根据名字去匹配的, 所以使用Resource注解可以成功注入
29      */
30     @Resource
31     private RedisTemplate<String, V> redisTemplate;
32
33     @Resource
34     private RedisTemplate<String, Object> redisHashTemplate;
35
36     /**
37      * 设置key的过期时间
38      *
39      * @param key      缓存key
40      * @param timeout 存活时间
41      * @param unit     时间单位
42      */
43     public void setExpire(String key, long timeout, TimeUnit unit) {
44         redisHashTemplate.expire(key, timeout, unit);
45     }
46
47     /**
48      * 根据key删除缓存
49      *
50      * @param keys 要删除的key, 可变参数列表
51      * @return 删除的缓存数量
52      */
53     public Long delete(String... keys) {
54         if (ObjectUtils.isEmpty(keys)) {
55             return 0L;
56         }
57         return redisTemplate.delete(Arrays.asList(keys));
58     }
59
60     /**
61      * 存入值
62      *
63      * @param key 缓存中的key
64      * @param value 存入的value
65      */
66     public void set(String key, V value) {
67         valueOperations().set(key, value);
68     }
69 }
```



```
73     * @param key 缓存中的key
74     * @return 返回键值对应缓存
75     */
76     public V get(String key) {
77         return valueOperations().get(key);
78     }
79
80     /**
81     * 设置键值并设置过期时间
82     *
83     * @param key      键
84     * @param value    值
85     * @param timeout 过期时间
86     * @param unit     过期时间的单位
87     */
88     public void set(String key, V value, long timeout, TimeUnit unit) {
89         valueOperations().set(key, value, timeout, unit);
90     }
91
92     /**
93     * 设置键值并设置过期时间（单位秒）
94     *
95     * @param key      键
96     * @param value    值
97     * @param timeout 过期时间,单位: 秒
98     */
99     public void set(String key, V value, long timeout) {
100         this.set(key, value, timeout, TimeUnit.SECONDS);
101     }
102
103     /**
104     * 根据key获取缓存并删除缓存
105     *
106     * @param key 要获取缓存的key
107     * @return key对应的缓存
108     */
109     public V getAndDelete(String key) {
110         if (ObjectUtils.isEmpty(key)) {
111             return null;
112         }
113         V value = valueOperations().get(key);
114         this.delete(key);
115         return value;
116     }
117
118     /**
```

```
122     * @param field hash结构的key
123     * @param value 存入的value
124     */
125     public void setHash(String key, String field, V value) {
126         hashOperations().put(key, field, value);
127     }
128
129     /**
130     * 根据key取值
131     *
132     * @param key 缓存中的key
133     * @return 缓存key对应的hash数据中field属性的值
134     */
135     public Object getHash(String key, String field) {
136         return hashOperations().hasKey(key, field) ? hashOperations().get(key, field) : null;
137     }
138
139     /**
140     * 以hash格式存入redis
141     *
142     * @param key 缓存中的key
143     * @param value 存入的对象
144     */
145     public void setHashAll(String key, Object value) {
146         Map<String, Object> map = JsonUtils.objectCovertToObject(value, Map.class, String.class,
147             hashOperations().putAll(key, map);
148     }
149
150     /**
151     * 设置键值并设置过期时间
152     *
153     * @param key 键
154     * @param value 值
155     * @param timeout 过期时间
156     * @param unit 过期时间的单位
157     */
158     public void setHashAll(String key, Object value, long timeout, TimeUnit unit) {
159         this.setHashAll(key, value);
160         this.setExpire(key, timeout, unit);
161     }
162
163     /**
164     * 设置键值并设置过期时间（单位秒）
165     *
166     * @param key 键
167     * @param value 值
```



```
171         this.setHashAll(key, value, timeout, TimeUnit.SECONDS);
172     }
173
174     /**
175      * 从redis中获取hash类型数据
176      *
177      * @param key 缓存中的key
178      * @return redis 中hash数据
179      */
180     public Map<String, Object> getMapHashAll(String key) {
181         return hashOperations().entries(key);
182     }
183
184     /**
185      * 根据指定clazz类型从redis中获取对应的实例
186      *
187      * @param key 缓存key
188      * @param clazz hash对应java类的class
189      * @param <T> redis中hash对应的java类型
190      * @return clazz实例
191      */
192     public <T> T getHashAll(String key, Class<T> clazz) {
193         Map<String, Object> entries = hashOperations().entries(key);
194         if (ObjectUtils.isEmpty(entries)) {
195             return null;
196         }
197         return JsonUtils.objectCovertToObject(entries, clazz);
198     }
199
200     /**
201      * 根据key删除缓存
202      *
203      * @param key 要删除的key
204      * @param fields key对应的hash数据的键值(HashKey)，可变参数列表
205      * @return hash删除的属性数量
206      */
207     public Long deleteHashField(String key, String... fields) {
208         if (ObjectUtils.isEmpty(key) || ObjectUtils.isEmpty(fields)) {
209             return 0L;
210         }
211         return hashOperations().delete(key, (Object[]) fields);
212     }
213
214     /**
215      * 将value添加至key对应的列表中
216      *
```



```
220     public void listPush(String key, V value) {
221         listOperations().rightPush(key, value);
222     }
223
224     /**
225      * 将value添加至key对应的列表中，并添加过期时间
226      *
227      * @param key      缓存key
228      * @param value    值
229      * @param timeout  key的存活时间
230      * @param unit     时间单位
231      */
232     public void listPush(String key, V value, long timeout, TimeUnit unit) {
233         listOperations().rightPush(key, value);
234         this.setExpire(key, timeout, unit);
235     }
236
237     /**
238      * 将value添加至key对应的列表中，并添加过期时间
239      * 默认单位是秒(s)
240      *
241      * @param key      缓存key
242      * @param value    值
243      * @param timeout  key的存活时间
244      */
245     public void listPush(String key, V value, long timeout) {
246         this.listPush(key, value, timeout, TimeUnit.SECONDS);
247     }
248
249     /**
250      * 将传入的参数列表添加至key的列表中
251      *
252      * @param key      缓存key
253      * @param values   值列表
254      * @return         存入数据的长度
255      */
256     public Long listPushAll(String key, Collection<V> values) {
257         return listOperations().rightPushAll(key, values);
258     }
259
260     /**
261      * 将传入的参数列表添加至key的列表中，并设置key的存活时间
262      *
263      * @param key      缓存key
264      * @param values   值列表
265      * @param timeout  key的存活时间
```



```
269     public Long listPushAll(String key, Collection<V> values, long timeout, TimeUnit unit) {
270         Long count = listOperations().rightPushAll(key, values);
271         this.setExpire(key, timeout, unit);
272         return count;
273     }
274
275     /**
276      * 将传入的参数列表添加至key的列表中，并设置key的存活时间
277      * 默认单位是秒(s)
278      *
279      * @param key      缓存key
280      * @param values   值列表
281      * @param timeout  key的存活时间
282      * @return 存入数据的长度
283      */
284     public Long listPushAll(String key, Collection<V> values, long timeout) {
285         return this.listPushAll(key, values, timeout, TimeUnit.SECONDS);
286     }
287
288     /**
289      * 根据key获取list列表
290      *
291      * @param key 缓存key
292      * @return key对应的list列表
293      */
294     public Collection<V> getList(String key) {
295         Long size = listOperations().size(key);
296         if (size == null || size == 0) {
297             return null;
298         }
299         return listOperations().range(key, 0, (size - 1));
300     }
301
302     /**
303      * value操作集
304      *
305      * @return ValueOperations
306      */
307     private ValueOperations<String, V> valueOperations() {
308         return redisTemplate.opsForValue();
309     }
310
311     /**
312      * hash操作集
313      *
314      * @return ValueOperations
```



```
318     }
319
320     /**
321      * hash操作集
322      *
323      * @return ValueOperations
324      */
325     private ListOperations<String, V> listOperations() {
326         return redisTemplate.opsForList();
327     }
328
329 }
```

暂时只提供了string、list、hash的操作，其它的后续在加吧

编写一个测试类测试这个工具类

test目录下



java 复制代码

```
1 package com.example;
2
3 import com.example.entity.Oauth2BasicUser;
4 import com.example.support.RedisOperator;
5 import lombok.SneakyThrows;
6 import lombok.extern.slf4j.Slf4j;
7 import org.junit.jupiter.api.Test;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.security.core.userdetails.UserDetailsService;
11
12 import java.util.Collection;
13 import java.util.List;
14 import java.util.Map;
15 import java.util.concurrent.TimeUnit;
16
17 /**
18  * redis工具类测试
19  *
20  * @author vains
21  */
```



```
25
26     @Autowired
27     private RedisOperator<String> redisOperator;
28
29     @Autowired
30     private UserDetailsService userDetailsService;
31
32     @Autowired
33     private RedisOperator<OAuth2BasicUser> userRedisOperator;
34
35     @Test
36     @SneakyThrows
37     void contextLoads() {
38         // 默认key
39         String defaultKey = "testKey";
40         // 默认缓存值
41         String defaultValue = "123456";
42         // key的存活时间
43         long timeout = 3;
44         // 操作hash的属性声明
45         String name = "name";
46
47         // 清除key
48         redisOperator.delete(defaultKey);
49
50         // 获取用户信息
51         OAuth2BasicUser userDetails = (OAuth2BasicUser) userDetailsService.loadUserByUsername("a
52
53         redisOperator.set(defaultKey, defaultValue);
54         log.info("根据key: {}存入值{}", defaultKey, defaultValue);
55
56         String valueByKey = redisOperator.get(defaultKey);
57         log.info("根据key: {}获取到值: {}", defaultKey, valueByKey);
58
59         String valueByKeyAndDelete = redisOperator.getAndDelete(defaultKey);
60         log.info("根据key: {}获取到值: {},删除key.", defaultKey, valueByKeyAndDelete);
61
62         Long delete = redisOperator.delete(defaultKey);
63         log.info("删除key: {}, 删除数量: {}. ", defaultKey, delete);
64
65         valueByKey = redisOperator.get(defaultKey);
66         log.info("根据key: {}获取到值: {}", defaultKey, valueByKey);
67
68         redisOperator.set(defaultKey, defaultValue, timeout);
69         log.info("根据key: {}存入值{}, 存活时长为: {}", defaultKey, defaultValue, timeout);
70         valueByKey = redisOperator.get(defaultKey);
```

```
74     TimeUnit.SECONDS.sleep((timeout + 1));
75
76     // 重复获取
77     valueByKey = redisOperator.get(defaultKey);
78     log.info("线程睡眠后根据失效的key: {}获取到值: {}", defaultKey, valueByKey);
79
80     redisOperator.setHashAll(defaultKey, userDetails, timeout);
81     log.info("根据key: {}存入hash类型值{},存活时间: {}", defaultKey, userDetails, timeout);
82
83     OAuth2BasicUser basicUser = redisOperator.getHashAll(defaultKey, OAuth2BasicUser.class);
84     log.info("根据key: {}获取到hash类型值: {}", defaultKey, basicUser);
85
86     // 睡眠, 让key失效
87     TimeUnit.SECONDS.sleep((timeout + 1));
88     // 重复获取
89     basicUser = redisOperator.getHashAll(defaultKey, OAuth2BasicUser.class);
90     log.info("线程睡眠后根据失效的key: {}获取到hash类型值: {}", defaultKey, basicUser);
91
92     redisOperator.setHashAll(defaultKey, userDetails, timeout);
93     log.info("根据key: {}存入hash类型值{},存活时间: {}", defaultKey, userDetails, timeout);
94
95     Map<String, Object> mapHashAll = redisOperator.getMapHashAll(defaultKey);
96     log.info("根据key: {}获取到hash类型值: {}", defaultKey, mapHashAll);
97
98     Object field = redisOperator.getHash(defaultKey, name);
99     log.info("根据key: {}获取到hash类型属性: {}的值: {}", defaultKey, name, field);
100
101     Long deleteHashField = redisOperator.deleteHashField(defaultKey, name);
102     log.info("根据key: {}删除hash类型的{}属性, 删除数量: {}", defaultKey, name, deleteHashField);
103
104     // 重复获取验证删除
105     field = redisOperator.getHash(defaultKey, name);
106     log.info("根据key: {}获取到hash类型属性: {}的值: {}", defaultKey, name, field);
107     basicUser = redisOperator.getHashAll(defaultKey, OAuth2BasicUser.class);
108     log.info("根据key: {}获取到hash类型值: {}", defaultKey, basicUser);
109
110     redisOperator.setHash(defaultKey, name, userDetails.getName());
111     log.info("根据key: {}设置hash类型的{}属性, 属性值为: {}", defaultKey, name, userDetails.get);
112
113     // 重复获取验证删除
114     field = redisOperator.getHash(defaultKey, name);
115     log.info("根据key: {}获取到hash类型属性: {}的值: {}", defaultKey, name, field);
116     basicUser = redisOperator.getHashAll(defaultKey, OAuth2BasicUser.class);
117     log.info("根据key: {}获取到hash类型值: {}", defaultKey, basicUser);
118
119     // 清除key
```

```
123     log.info("根据key: {}往list类型数据中添加数据: {}", defaultKey, userDetails);
124
125     Collection<Oauth2BasicUser> users = userRedisOperator.getList(defaultKey);
126     log.info("根据key: {}获取list数据: {}", defaultKey, users);
127
128     Long listPushAll = userRedisOperator.listPushAll(defaultKey, List.of(userDetails));
129     log.info("根据key: {}往list类型数据中添加数据: {}, 成功添加{}条数据", defaultKey, List.of(us
130
131     users = userRedisOperator.getList(defaultKey);
132     log.info("根据key: {}获取list数据: {}", defaultKey, users);
133
134     userRedisOperator.listPush(defaultKey, userDetails, timeout);
135     log.info("根据key: {}往list类型数据中添加数据: {}, key的存活时间为: {}", defaultKey, userDet
136     // 睡眠, 让key失效
137     TimeUnit.SECONDS.sleep((timeout + 1));
138     // 重复获取
139     users = userRedisOperator.getList(defaultKey);
140     log.info("线程睡眠后根据失效的key: {}获取到list类型值: {}", defaultKey, users);
141
142     Long aLong = userRedisOperator.listPushAll(defaultKey, List.of(userDetails), timeout);
143     log.info("根据key: {}往list类型数据中添加数据: {}, 成功添加{}条数据, 设置过期时间: {}", defau
144 }
145
146 }
```

测试

组装url发起授权请求

http://127.0.0.1:8080/oauth2/authorize?client_id=messaging-client&response_type=code&scope=message.read&redirect_uri=http%3A%2F%2F127.0.0.1%3A8000%2Flogin%2Foauth2%2Fcode%2Fmessaging-client-oidc

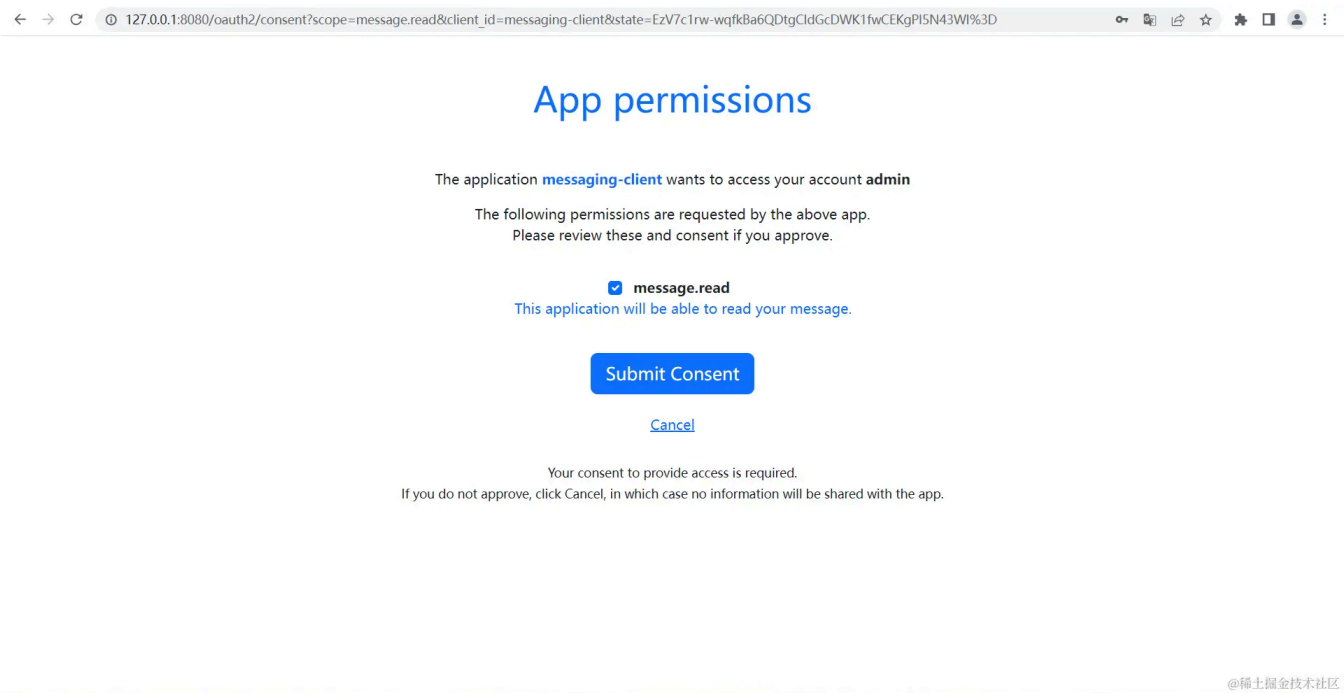
重定向到登录页面



查看redis



登录后重定向至授权确认页面





@稀土掘金技术社区

如果有什么问题请在评论区指出，如果我看到会尽快处理的，谢谢
代码已提交至Gitee：[仓库地址](#)

标签： Spring Spring Boot 安全 话题： 我的技术写作成长之路

本文收录于以下专栏

◀

1 / 2

▶



Spring Authorization Server
Spring Authorization Server系列文章
180 订阅 · 25 篇文章

专栏目录

订阅

上一篇 Spring Authorization Server... 下一篇 Spring Authorization Server...

评论 12



平等表达，友善交流



0 / 1000  发送

最热 最新




全智能废品修理师

HttpSession 直接存 redis 里面和直接使用 redis 区别是啥呀，我现在直接用的 org.springframework.session:spring-session-data-redis

4月前  点赞  2 ...

4月前

 点赞

 回复

...



全智能废品修... 回复 叹雪飞花 作者：😂好，我直接使用Spring session redis整起了

4月前

 1

 回复

...



i吧啦啦啦啦 LV.2 吧啦啦啦啦 @啊吧啊吧

为啥 我添加了@JsonSerialize 还是报错 The class with jdk.proxy2.\$Proxy202 and name of jdk.proxy2.\$Proxy202 is not in the allowlist. If you believe this class is safe to deserialize, please provide an explicit mapping using Jackson annotations or by providing a Mixin. If the serialization is only done by a trusted source, you can also enable default typing. See github.com for details

7月前

 点赞

 4

...



i吧啦啦啦啦：和redis的序列化有关吗？



7月前

 点赞

 回复

...



叹雪飞花 作者：你要确定一下你这个代理类是哪里的

7月前

 点赞

 回复

...

查看全部 4 条回复 ▾



用户4153993730335

我使用了你的配置后,能正常配合redis进行登录,但我发现redis里面的数据会有\xac\xed\x00\.....之类的乱码,所以加了一个
redisTemplate.setValueSerializer(stringRedisSerializer)。乱码是没有了,但在登录时出现了java.lang.ClassCastException: class org.springframework.security.core.context.SecurityContextImpl cannot be cast to class java.lang.String...

展开

7月前

 点赞

 3

...



Jackson的value序列化类之后会将value序列化成类似于json格式的数据，读的时候会将对象格式的value读成LinkedHashMap的实例，所以在读取时工具类的泛型可以设置为Object，读取之后使用json工具类将value转成对应的java类，而你这种是因为使用了string序列化器，在存储至redis时强转为string失败了，你需...

展开

7月前 点赞 回复 ...



用户4153993... 回复 叹雪飞花 作者：好的谢谢你的回复,我在value用了你hash用的那个序列化器后也能正常配合redis进行登录,乱码就变成被\"包着了:

\"\\abc\\\"

7月前 点赞 回复 ...

查看全部 3 条回复

目录 收起

前言

添加统一响应类

在model包下添加Result.java类

优化项目

在controller包下添加LoginController类

CaptchaResult

优化登录页面

修改CaptchaAuthenticationProvider从redis中获取验证码

修改SmsCaptchaLoginAuthenticationProvider从redis中获取短信验证码

修改Oauth2BasicUser的authorities属性，以防序列化失败

CustomGrantedAuthority

Redis常量类RedisConstants

SecurityConstants添加常量

整合Spring data redis的步骤

引入starrer

编写Redis配置文件

测试

- 组装url发起授权请求
- 重定向到登录页面
- 查看redis
- 登录后重定向至授权确认页面
- 确认后重定向至回调页面




相关推荐



- Spring Authorization Server入门 (十五) 分离授权确认与设备码校验页面
1.6k阅读 · 14点赞
- Spring Data Redis工具类
207阅读 · 3点赞
- Spring Authorization Server入门 (十四) 联合身份认证添加微信登录
1.2k阅读 · 5点赞
- Spring Authorization Server优化篇：自定义UserDetailsService实现从数据库获取用户信息
1.1k阅读 · 5点赞
- 【安全篇】Spring Boot 整合 Spring Authorization Server
3.5k阅读 · 1点赞




精选内容


- 使用Python进行自动化测试测试框架的选择与应用
CodeJourney · 255阅读 · 0点赞
- 研究一款 Java 线程编排并行框架-asyncTool
uzong · 441阅读 · 6点赞
- 并发容器之CopyOnWrite
isfox · 270阅读 · 0点赞
- Linux快速安装FFmpeg、ffprobe、ffplay以及在Linux上的使用
杰哥的技术杂货铺 · 173阅读 · 0点赞
- Redis调优-BigKey如何处理？
码易有道 · 234阅读 · 1点赞




- Spring Authorization Server入门 (二) Spring Boot整合Spring Authorization Server



叹雪飞花 9月前  6.8k  31  86 Java
- Spring Authorization Server入门 (十二) 实现授权码模式使用前后端分离的登录页面




叹雪飞花 8月前  4.8k  24  65 后端 Spring ... Spring
- Spring Authorization Server入门 (十) 添加短信验证码方式登录




叹雪飞花 9月前  3.4k  20  9 Spring Spring ...
- Spring Authorization Server入门 (八) Spring Boot引入Security OAuth2 Client对接认...




叹雪飞花 9月前  2.8k  13  43 Spring ... Spring
- Spring Authorization Server入门 (十六) Spring Cloud Gateway对接认证服务




叹雪飞花 7月前  2.6k  17  44 Spring ... Spring ... 安全
- Spring Authorization Server入门 (十三) 实现联合身份认证，集成Github与Gitee的OAu...




叹雪飞花 8月前  2.3k  13  51 Spring Spring ... 安全
- Spring Authorization Server入门 (十一) 自定义grant_type(短信认证登录)获取token



叹雪飞花 9月前  2.5k  15  39 Spring Spring ... 安全
- Spring Authorization Server入门 (七) 登录添加图形验证码

叹雪飞花 9月前  2.9k  18  4 Spring ...
- SpringBoot3.x最简集成SpringDoc-OpenApi

叹雪飞花 4月前  2.3k  16  评论 后端 Spring ... Java
- Spring Authorization Server入门 (九) Spring Boot引入Resource Server对接认证服务

叹雪飞花 9月前  1.8k  13  8 Spring Spring ...
- Spring Authorization Server入门 (十五) 分离授权确认与设备码校验页面

叹雪飞花 7月前  1.6k  14  8 Spring ... Spring Vue.js
- Spring Authorization Server入门 (十九) 基于Redis的Token、客户端信息和授权确认信...


叹雪飞花 4月前  1.2k  8  19 Spring ... 后端 Redis
- Spring Authorization Server入门 (二十) 实现二维码扫码登录

Spring Authorization Server入门(下七) vue项目使用授权时候式对接认证服务

叹雪飞花 6月前  760  9  12

Vue.js 安全 Spring ...

Spring Cloud Gateway集成SpringDoc，集中管理微服务API

叹雪飞花 3月前  761  7  评论

Spring ... Spring ... Java