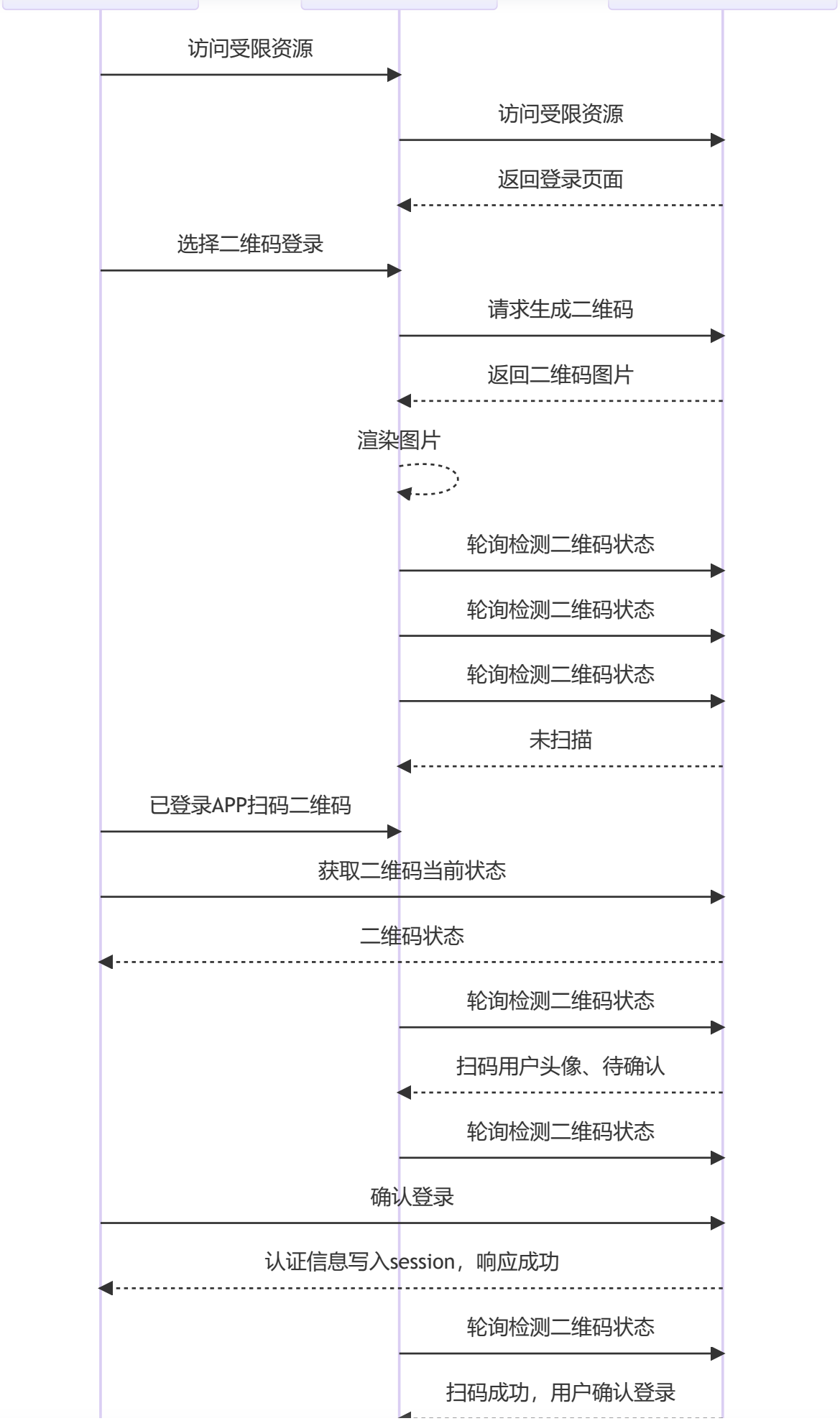
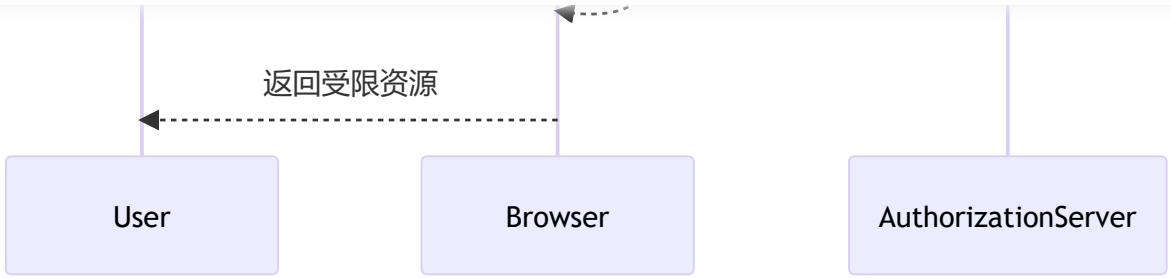
Spring Authorization Server入门 (二十) 实现二维码扫码登录

叹雪飞花 2024-01-22 937 阅读4分钟

实现原理

- 打开网页，发起授权申请/未登录被重定向到登录页面
- 选择二维码登录，页面从后端请求二维码
- 页面渲染二维码图片，并轮询请求，获取二维码的状态
- 事先登录过APP的手机扫描二维码，然后APP请求服务器端的API接口，把用户认证信息传递到服务器中
- 后端收到APP的请求后更改二维码状态，并把用户认证信息写入session
- 页面得到扫码确认的响应，并跳转回之前未登录的地址


访问受限资源
访问受限资源
返回登录页面
选择二维码登录
请求生成二维码
返回二维码图片
渲染图片
轮询检测二维码状态
轮询检测二维码状态
轮询检测二维码状态
未扫描
已登录APP扫码二维码
获取二维码当前状态
二维码状态
轮询检测二维码状态
扫码用户头像、待确认
轮询检测二维码状态
确认登录
认证信息写入session，响应成功
轮询检测二维码状态
扫码成功，用户确认登录



在这个流程中"用户认证信息写入session"后前端在重定向时能获取到认证信息是因为现在的认证服务引入了 `spring session data redis` 依赖，并且在 `application.yml` 中配置了 `server.servlet.session.cookie.domain: cdhttp.cn` 属性(演示环境域名)，这里是指定spring session的顶级域名，在该域名下的子域名服务共享session，如果你是在开发阶段可以配置为 `server.servlet.session.cookie.domain: 127.0.0.1` ，这时候前端访问使用 `127.0.0.1:5173` ，认证服务使用 `127.0.0.1:8080` ，端口可以不一样，但是认证服务和前端必须使用同一域名。

代码实现

后端实现

引入二维码依赖

xml 复制代码

```xml
<dependency>
    <groupId>com.google.zxing</groupId>
    <artifactId>core</artifactId>
    <version>3.3.0</version>
</dependency>
<dependency>
    <groupId>com.google.zxing</groupId>
    <artifactId>javase</artifactId>
    <version>3.3.0</version>
</dependency>
```

`hutool-captcha` 改为 `hutool-all`

xml 复制代码

```
    <artifactId>hutool-all</artifactId>
    <version>${hutool.version}</version>
</dependency>
```

添加二维码登录接口

提供四个接口，分别处理生成二维码、轮询、app扫码和app确认登录的逻辑。

java 复制代码

```java
package com.example.controller;

import com.example.model.Result;
import com.example.model.request.qrcode.QrCodeLoginConsentRequest;
import com.example.model.request.qrcode.QrCodeLoginScanRequest;
import com.example.model.response.qrcode.QrCodeGenerateResponse;
import com.example.model.response.qrcode.QrCodeLoginFetchResponse;
import com.example.model.response.qrcode.QrCodeLoginScanResponse;
import com.example.service.IQrCodeLoginService;
import lombok.AllArgsConstructor;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * 二维码登录接口
 *
 * @author vains
 */
@RestController
@AllArgsConstructor
@RequestMapping("/qrCode")
public class QrCodeLoginController {

    private final IQrCodeLoginService iQrCodeLoginService;

    @GetMapping("/login/generateQrCode")
    public Result<QrCodeGenerateResponse> generateQrCode() {
        // 生成二维码
        return Result.success(iQrCodeLoginService.generateQrCode());
    }
```

```
38          // 轮询二维码状态
39          return Result.success(iQrCodeLoginService.fetch(qrCodeId));
40      }
41
42
43      @PostMapping("/scan")
44      public Result<QrCodeLoginScanResponse> scan(@RequestBody QrCodeLoginScanRequest loginScan) {
45          // app 扫码二维码
46          return Result.success(iQrCodeLoginService.scan(loginScan));
47      }
48
49      @PostMapping("/consent")
50      public Result<String> consent(@RequestBody QrCodeLoginConsentRequest loginConsent) {
51
52          // app 确认登录
53          iQrCodeLoginService.consent(loginConsent);
54
55          return Result.success();
56      }
57
58  }
```

yml中放行前端访问的接口

添加匹配规则 `/qrCode/login/**`

yml 复制代码

```yml
custom:
  # 自定义认证配置
  security:
    # 登录页面路径
    login-url: http://k7fsqkhtbx.cdhttp.cn/login
    # 授权确认页面路径
    consent-page-uri: http://k7fsqkhtbx.cdhttp.cn/consent
    # 设备码验证页面
    device-activate-uri: http://k7fsqkhtbx.cdhttp.cn/activate
    # 设备码验证成功页面
    device-activated-uri: http://k7fsqkhtbx.cdhttp.cn/activated
    # 不需要认证的地址
    ignore-uri-list: assets/**, /webjars/**, /login, /getCaptcha, /getSmsCaptcha, /error, /oauth
```

编写登录service接口

java 复制代码

```java
package com.example.service;

import com.example.model.request.qrcode.QrCodeLoginConsentRequest;
import com.example.model.request.qrcode.QrCodeLoginScanRequest;
import com.example.model.response.qrcode.QrCodeGenerateResponse;
import com.example.model.response.qrcode.QrCodeLoginFetchResponse;
import com.example.model.response.qrcode.QrCodeLoginScanResponse;

/**
 * 二维码登录服务接口
 *
 * @author vains
 */
public interface IQrCodeLoginService {

    /**
     * 生成二维码
     *
     * @return 二维码
     */
    QrCodeGenerateResponse generateQrCode();

    /**
     * 扫描二维码响应
     *
     * @param loginScan 二维码id
     * @return 二维码信息
     */
    QrCodeLoginScanResponse scan(QrCodeLoginScanRequest loginScan);

    /**
     * 二维码登录确认入参
     *
     * @param loginConsent 二维码id
     */
    void consent(QrCodeLoginConsentRequest loginConsent);

    /**
     * web端轮询二维码状态处理
     *
     * @param qrCodeId 二维码id
     * @return 二维码信息
```

```
46 }
```

编写生成二维码响应类

生成二维码图片时返回二维码id和图片

java 复制代码

```java
package com.example.model.response.qrcode;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * 生成二维码响应
 *
 * @author vains
 */
@Data
@NoArgsConstructor
@AllArgsConstructor
public class QrCodeGenerateResponse {

    /**
     * 二维码id
     */
    private String qrCodeId;

    /**
     * 二维码base64值(这里响应一个链接好一些)
     */
    private String imageData;

}
```

编写web前端轮询二维码状态出参

前端根据二维码id轮询二维码状态时返回二维码状态，如果已扫描也会返回扫描者的头像、昵称。

```java
import lombok.Data;

import java.util.Set;

/**
 * web前端轮询二维码状态出参
 *
 * @author vains
 */
@Data
public class QrCodeLoginFetchResponse {

    /**
     * 二维码状态
     * 0:待扫描，1:已扫描，2:已确认
     */
    private Integer qrCodeStatus;

    /**
     * 是否已过期
     */
    private Boolean expired;

    /**
     * 扫描人头像
     */
    private String avatarUrl;

    /**
     * 扫描人昵称
     */
    private String name;

    /**
     * 待确认scope
     */
    private Set<String> scopes;

    /**
     * 跳转登录之前请求的接口
     */
    private String beforeLoginRequestUri;

    /**
     * 跳转登录之前请求参数
     */
```

编写扫描二维码入参

app扫描二维码时传入二维码id

java 复制代码

```java
package com.example.model.request.qrcode;

import lombok.Data;

/**
 * 扫描二维码入参
 *
 * @author vains
 */
@Data
public class QrCodeLoginScanRequest {

    /**
     * 二维码id
     */
    private String qrCodeId;

}
```

编写二维码响应bean

扫描二维码时生成一个临时票据返回，同时返回scope和二维码状态。

java 复制代码

```java
package com.example.model.response.qrcode;

import lombok.Data;

import java.util.Set;

/**
 * 扫描二维码响应bean
 *
```

```java
public class QrCodeLoginScanResponse {

    /**
     * 扫描临时票据
     */
    private String qrCodeTicket;

    /**
     * 二维码状态
     */
    private Integer qrCodeStatus;

    /**
     * 是否已过期
     */
    private Boolean expired;

    /**
     * 待确认scope
     */
    private Set<String> scopes;

}
```

编写二维码登录确认登录入参类

确认登录时传入二维码id和上一步生成的临时票据防篡改。

java 复制代码

```java
package com.example.model.request.qrcode;

import lombok.Data;

/**
 * 二维码登录确认入参
 *
 * @author vains
 */
@Data
public class QrCodeLoginConsentRequest {

    /**
     * 二维码id
```

```
18      /**
19       * 扫码二维码后产生的临时票据(仅一次有效)
20       */
21      private String qrCodeTicket;
22
23  }
```

编写二维码信息类

生成二维码时生成的数据bean，存入redis中，等到前端轮询或app端操作时使用。

java 复制代码

```
1   package com.example.model.qrcode;
2
3   import lombok.AllArgsConstructor;
4   import lombok.Builder;
5   import lombok.Data;
6   import lombok.NoArgsConstructor;
7
8   import java.time.LocalDateTime;
9   import java.util.Set;
10
11  /**
12   * 二维码信息
13   *
14   * @author vains
15   */
16  @Data
17  @Builder
18  @NoArgsConstructor
19  @AllArgsConstructor
20  public class QrCodeInfo {
21
22      /**
23       * 二维码id
24       */
25      private String qrCodeId;
26
27      /**
28       * 二维码状态
29       * 0:待扫描，1:已扫描，2:已确认
30       */
31      private Integer qrCodeStatus;
```

```
35      */
36     private LocalDateTime expiresTime;
37
38     /**
39      * 扫描人头像
40      */
41     private String avatarUrl;
42
43     /**
44      * 扫描人昵称
45      */
46     private String name;
47
48     /**
49      * 待确认的scope
50      */
51     private Set<String> scopes;
52
53     /**
54      * 跳转登录之前请求的接口
55      */
56     private String beforeLoginRequestUri;
57
58     /**
59      * 跳转登录之前请求参数
60      */
61     private String beforeLoginQueryString;
62
63 }
```

编写二维码登录接口实现

扫码登录实现，具体逻辑请看代码中的注释。

java 复制代码

```
1  package com.example.service.impl;
2
3  import cn.hutool.extra.qrcode.QrCodeUtil;
4  import cn.hutool.extra.qrcode.QrConfig;
5  import com.baomidou.mybatisplus.core.toolkit.IdWorker;
6  import com.example.entity.Oauth2BasicUser;
7  import com.example.model.qrcode.QrCodeInfo;
8  import com.example.model.request.qrcode.QrCodeLoginConsentRequest;
```

```java
import com.example.model.response.qrcode.QrCodeLoginScanResponse;
import com.example.property.CustomSecurityProperties;
import com.example.service.IQrCodeLoginService;
import com.example.support.RedisOperator;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.authentication.InsufficientAuthenticationException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.context.SecurityContextImpl;
import org.springframework.security.oauth2.core.OAuth2AuthenticationException;
import org.springframework.security.oauth2.core.OAuth2Error;
import org.springframework.security.oauth2.core.OAuth2ErrorCodes;
import org.springframework.security.oauth2.server.authorization.OAuth2Authorization;
import org.springframework.security.oauth2.server.authorization.OAuth2TokenType;
import org.springframework.security.oauth2.server.resource.authentication.JwtAuthenticationToken
import org.springframework.security.web.savedrequest.DefaultSavedRequest;
import org.springframework.security.web.savedrequest.HttpSessionRequestCache;
import org.springframework.security.web.savedrequest.RequestCache;
import org.springframework.security.web.util.UrlUtils;
import org.springframework.stereotype.Service;
import org.springframework.util.Assert;
import org.springframework.util.ObjectUtils;
import org.springframework.web.context.request.RequestAttributes;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

import java.security.Principal;
import java.time.LocalDateTime;
import java.util.Objects;
import java.util.Set;

import static org.springframework.security.web.context.HttpSessionSecurityContextRepository.SPRI

/**
 * 二维码登录接口实现
 *
 * @author vains
 */
@Slf4j
@Service
@RequiredArgsConstructor
```

```
61
62    private final RedisOperator<String> stringRedisOperator;
63
64    private final CustomSecurityProperties customSecurityProperties;
65
66    private final RedisOAuth2AuthorizationService authorizationService;
67
68    private final RedisOperator<UsernamePasswordAuthenticationToken> authenticationRedisOperator;
69
70    /**
71       * 过期时间
72       */
73      private final long QR_CODE_INFO_TIMEOUT = 60 * 10;
74
75    /**
76       * 二维码信息前缀
77       */
78      private final String QR_CODE_PREV = "login:qrcode:";
79
80    private final RequestCache requestCache = new HttpSessionRequestCache();
81
82    @Override
83      public QrCodeGenerateResponse generateQrCode() {
84            // 生成二维码唯一id
85            String qrCodeId = IdWorker.getIdStr();
86            // 生成二维码并转为base64
87            String pngQrCode = QrCodeUtil.generateAsBase64(qrCodeId, new QrConfig(), "png");
88            QrCodeInfo info = QrCodeInfo.builder()
89                  .qrCodeId(qrCodeId)
90                  // 待扫描状态
91                  .qrCodeStatus(0)
92                  // 1分钟后过期
93                  .expiresTime(LocalDateTime.now().plusMinutes(2L))
94                  .build();
95
96            // 获取当前request
97            RequestAttributes requestAttributes = RequestContextHolder.getRequestAttributes();
98            if (requestAttributes != null) {
99                    // 获取当前session
100                   HttpServletRequest request = ((ServletRequestAttributes) requestAttributes).ge
101                   HttpServletResponse response = ((ServletRequestAttributes) requestAttributes).
102                   DefaultSavedRequest savedRequest = (DefaultSavedRequest) this.requestCache.get
103                   if (savedRequest != null) {
104                           if (!UrlUtils.isAbsoluteUrl(customSecurityProperties.getLoginUrl())) {
105                                   // 获取查询参数与请求路径
106                                   String queryString = savedRequest.getQueryString();
```

```
110                                 info.setBeforeLoginQueryString(queryString);
111                         }
112
113                         // 获取跳转登录之前访问url的query parameter
114                         String[] scopes = savedRequest.getParameterValues("scope");
115                         if (!ObjectUtils.isEmpty(scopes)) {
116                                 // 不为空获取第一个并设置进二维码信息中
117                                 info.setScopes(Set.of(scopes[0].split(" ")));
118                         }
119                         // 前端可以根据scope显示要获取的信息，或固定显示要获取的信息
120                 }
121         }
122
123         // 因为上边设置的过期时间是2分钟，这里设置10分钟过期，可根据业务自行调整过期时间
124         redisOperator.set(QR_CODE_PREV + qrCodeId, info, QR_CODE_INFO_TIMEOUT);
125         return new QrCodeGenerateResponse(qrCodeId, pngQrCode);
126 }
127
128 @Override
129   public QrCodeLoginScanResponse scan(QrCodeLoginScanRequest loginScan) {
130         // 应该用validation的
131         Assert.hasLength(loginScan.getQrCodeId(), "二维码Id不能为空.");
132
133         // 校验二维码状态
134         QrCodeInfo info = redisOperator.get(QR_CODE_PREV + loginScan.getQrCodeId());
135         if (info == null) {
136
137                 throw new RuntimeException("无效二维码.");
138         }
139
140         // 验证状态
141         if (!Objects.equals(info.getQrCodeStatus(), 0)) {
142
143                 throw new RuntimeException("二维码已被其他人扫描，无法重复扫描.");
144         }
145
146         // 二维码是否过期
147         boolean qrCodeExpire = info.getExpiresTime().isBefore(LocalDateTime.now());
148         if (qrCodeExpire) {
149                 throw new RuntimeException("二维码已过期.");
150         }
151
152         QrCodeLoginScanResponse loginScanResponse = new QrCodeLoginScanResponse();
153
154         // 获取登录用户信息
155         OAuth2Authorization oAuth2Authorization = this.getOAuth2Authorization();
```

```
159             }
160             // app端使用密码模式、手机认证登录，不使用三方登录的情况
161             UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken =
162                     oAuth2Authorization.getAttribute(Principal.class.getName());
163             if (usernamePasswordAuthenticationToken.getPrincipal() instanceof Oauth2BasicUser basi
164                     // 生成临时票据
165                     String qrCodeTicket = IdWorker.getIdStr();
166                     // 根据二维码id和临时票据存储，确认时根据临时票据认证
167                     String redisQrCodeTicketKey = String.format("%s%s:%s", QR_CODE_PREV, loginScan
168                     stringRedisOperator.set(redisQrCodeTicketKey, qrCodeTicket, QR_CODE_INFO_TIMEO
169
170                     // 更新二维码信息的状态
171                     info.setQrCodeStatus(1);
172                     info.setName(basicUser.getName());
173                     info.setAvatarUrl(basicUser.getAvatarUrl());
174                     redisOperator.set(QR_CODE_PREV + loginScan.getQrCodeId(), info, QR_CODE_INFO_T
175
176                     // 封装响应
177                     loginScanResponse.setQrCodeTicket(qrCodeTicket);
178                     loginScanResponse.setQrCodeStatus(0);
179                     loginScanResponse.setExpired(Boolean.FALSE);
180                     loginScanResponse.setScopes(info.getScopes());
181             }
182
183             // 其它登录方式暂不处理
184             return loginScanResponse;
185     }
186
187     @Override
188       public void consent(QrCodeLoginConsentRequest loginConsent) {
189             // 应该用validation的
190             Assert.hasLength(loginConsent.getQrCodeId(), "二维码Id不能为空.");
191
192             // 校验二维码状态
193             QrCodeInfo info = redisOperator.get(QR_CODE_PREV + loginConsent.getQrCodeId());
194             if (info == null) {
195                     throw new RuntimeException("无效二维码或二维码已过期.");
196             }
197
198             // 验证临时票据
199             String qrCodeTicketKey =
200                     String.format("%s%s:%s", QR_CODE_PREV, loginConsent.getQrCodeId(), loginConsent.
201             String redisQrCodeTicket = stringRedisOperator.get(qrCodeTicketKey);
202             if (!Objects.equals(redisQrCodeTicket, loginConsent.getQrCodeTicket())) {
203                     // 临时票据有误、临时票据失效(超过redis存活时间后确认)、redis数据有误
204                     if (log.isDebugEnabled()) {
```

```
208             }
209             // 使用后删除
210             stringRedisOperator.delete(qrCodeTicketKey);
211
212             // 获取登录用户信息
213             OAuth2Authorization authorization = this.getOAuth2Authorization();
214             if (authorization == null) {
215                     throw new OAuth2AuthenticationException(
216                                     new OAuth2Error(OAuth2ErrorCodes.INVALID_TOKEN, "登录已过期.",
217             }
218
219             // app端使用密码模式、手机认证登录，不使用三方登录的情况
220             UsernamePasswordAuthenticationToken authenticationToken = authorization.getAttribute(P
221
222             // 根据二维码id存储用户信息
223             String redisUserinfoKey = String.format("%s%s:%s", QR_CODE_PREV, "userinfo", loginCons
224             // 存储用户信息
225             authenticationRedisOperator.set(redisUserinfoKey, authenticationToken, QR_CODE_INFO_TI
226
227             // 更新二维码信息的状态
228             info.setQrCodeStatus(2);
229             redisOperator.set(QR_CODE_PREV + loginConsent.getQrCodeId(), info, QR_CODE_INFO_TIMEOU
230     }
231
232     @Override
233       public QrCodeLoginFetchResponse fetch(String qrCodeId) {
234             // 校验二维码状态
235             QrCodeInfo info = redisOperator.get(QR_CODE_PREV + qrCodeId);
236             if (info == null) {
237                     throw new RuntimeException("无效二维码或二维码已过期.");
238             }
239
240             QrCodeLoginFetchResponse loginFetchResponse = new QrCodeLoginFetchResponse();
241             // 设置二维码是否过期、状态
242             loginFetchResponse.setQrCodeStatus(info.getQrCodeStatus());
243             loginFetchResponse.setExpired(info.getExpiresTime().isBefore(LocalDateTime.now()));
244
245             if (!Objects.equals(info.getQrCodeStatus(), 0)) {
246                     // 如果是已扫描/已确认
247                     loginFetchResponse.setName(info.getName());
248                     loginFetchResponse.setAvatarUrl(info.getAvatarUrl());
249             }
250
251             // 如果是已确认，将之前扫码确认的用户信息放入当前session中
252             if (Objects.equals(info.getQrCodeStatus(), 2)) {
253
```

```
257                 if (authenticationToken != null) {
258                         // 获取当前request
259                         RequestAttributes requestAttributes = RequestContextHolder.getRequestA
260                         if (requestAttributes == null) {
261                                 throw new RuntimeException("获取当前Request失败.");
262                         }
263                         // 获取当前session
264                         HttpServletRequest request = ((ServletRequestAttributes) requestAttrib
265                         HttpSession session = request.getSession(Boolean.FALSE);
266                         if (session != null) {
267                                 // 获取到认证信息后将之前扫码确认的用户信息放入当前session中。
268                                 session.setAttribute(SPRING_SECURITY_CONTEXT_KEY, new Security
269
270                                 // 操作成功后移除缓存
271                                 redisOperator.delete(QR_CODE_PREV + qrCodeId);
272                                 // 删除用户信息，防止其它人重放请求
273                                 authenticationRedisOperator.delete(redisUserinfoKey);
274
275                                 // 填充二维码数据，设置跳转到登录之前的请求路径、查询参数和是否授权
276                                 loginFetchResponse.setBeforeLoginRequestUri(info.getBeforeLogi
277                                 loginFetchResponse.setBeforeLoginQueryString(info.getBeforeLog
278                         }
279                 } else {
280                         throw new RuntimeException("获取登录确认用户信息失败.");
281                 }
282         }
283
284         return loginFetchResponse;
285 }
286
287 /**
288    * 获取当前使用token对应的认证信息
289    *
290    * @return oauth2认证信息
291    */
292   private OAuth2Authorization getOAuth2Authorization() {
293         // 校验登录状态
294         Authentication authentication = SecurityContextHolder.getContext().getAuthentication()
295         if (authentication == null) {
296                 throw new InsufficientAuthenticationException("未登录.");
297         }
298         if (authentication instanceof JwtAuthenticationToken jwtToken) {
299                 // jwt处理
300                 String tokenValue = jwtToken.getToken().getTokenValue();
301                 // 根据token获取授权登录时的认证信息(登录用户)
302                 return authorizationService.findByToken(tokenValue, OAuth2TokenType.ACCESS_TOK
```

```
306
307 }
```

到此后端部分就完成了，全局根据二维码id将整个流程串联起来，前端轮询、app端扫码和登录确认都是通过二维码id来的，中间借助redis来缓存二维码的信息，以确保每个端都可以获取到二维码信息，这样就算集群部署也不影响。

现在默认是将app端当做oauth2登录的，扫码和确认登录都是通过access_token来获取认证信息的，根据请求的token获取oauth2流程中'登录'生成的认证信息。

前端实现

编写二维码登录请求api

编写src/api/QrCodeLogin.ts文件

ts 复制代码

```ts
import loginRequest from '../util/http/LoginRequest'

/**
 * 生成二维码
 */
export function generateQrCode() {
    return loginRequest.get<any>({
        url: '/qrCode/login/generateQrCode'
    })
}

/**
 * 获取二维码信息
 * @param qrCodeId 二维码id
 */
export function fetch(qrCodeId: string) {
    return loginRequest.get<any>({
        url: `/qrCode/login/fetch/${qrCodeId}`
    })
}
```

前端页轮询时如果发现二维码状态变为确认登录则会重定向到之前被拦截后跳转到登录的地址，例如：访问/a发现未登录然后跳转到登录/login，之后扫码登录流程走完以后会重定向至/a。

vue 复制代码

```vue
<script setup lang="ts">
import { ref } from 'vue'
import router from '../../router'
import { getQueryString } from '@/util/GlobalUtils'
import { generateQrCode, fetch } from '@/api/QrCodeLogin'
import { type CountdownProps, createDiscreteApi } from 'naive-ui'
import {
  getImageCaptcha,
  getSmsCaptchaByPhone,
  loginSubmit
} from '@/api/Login'

const { message } = createDiscreteApi(['message'])

// 登录按钮加载状态
const loading = ref(false)

// 定义登录提交的对象
const loginModel = ref({
  code: '',
  username: '',
  password: '',
  loginType: '',
  captchaId: ''
})

// 图形验证码的base64数据
let captchaImage = ref('')
// 图形验证码的值
let captchaCode = ''
// 是否开始倒计时
const counterActive = ref(false)
// 是否显示三方登录
const showThirdLogin = ref(true)

// 定义二维码信息的对象
const qrCodeInfo = ref({
  qrCodeStatus: 0,
  expired: false,
  avatarUrl: '',
```

```
44
45  // 生成二维码响应数据
46  const getQrCodeInfo = ref({
47    qrCodeId: '',
48    imageData: ''
49  })
50
51  // 是否自动提交授权确认(二维码登录自动提交)
52  const autoConsentKey: string = 'autoConsent'
53
54  /**
55   * 获取图形验证码
56   */
57  const getCaptcha = () => {
58    getImageCaptcha()
59      .then((result: any) => {
60        if (result.success) {
61          captchaCode = result.data.code
62          captchaImage.value = result.data.imageData
63          loginModel.value.captchaId = result.data.captchaId
64        } else {
65          message.warning(result.message)
66        }
67      })
68      .catch((e: any) => {
69        message.warning(`获取图形验证码失败：${e.message}`)
70      })
71  }
72
73  /**
74   * 提交登录表单
75   * @param type 登录类型，passwordLogin是密码模式，smsCaptcha短信登录
76   */
77  const submitLogin = (type: string) => {
78    loading.value = true
79    loginModel.value.loginType = type
80    loginSubmit(loginModel.value)
81      .then((result: any) => {
82        if (result.success) {
83          // 移除自动提交缓存
84          localStorage.removeItem(autoConsentKey)
85          // message.info(`登录成功`)
86          let target = getQueryString('target')
87          if (target) {
88            window.location.href = target
89          } else {
```

```
93        } else {
94          message.warning(result.message)
95        }
96      })
97      .catch((e: any) => {
98        message.warning(`登录失败：${e.message}`)
99      })
100     .finally(() => {
101       loading.value = false
102     })
103 }
104
105 /**
106  * 获取短信验证码
107  */
108 const getSmsCaptcha = () => {
109   if (!loginModel.value.username) {
110     message.warning('请先输入手机号.')
111     return
112   }
113   if (!loginModel.value.code) {
114     message.warning('请先输入验证码.')
115     return
116   }
117   if (loginModel.value.code !== captchaCode) {
118     message.warning('验证码错误.')
119     return
120   }
121   getSmsCaptchaByPhone({ phone: loginModel.value.username })
122     .then((result: any) => {
123       if (result.success) {
124         message.info(`获取短信验证码成功，固定为：${result.data}`)
125         counterActive.value = true
126       } else {
127         message.warning(result.message)
128       }
129     })
130     .catch((e: any) => {
131       message.warning(`获取短信验证码失败：${e.message}`)
132     })
133 }
134
135 /**
136  * 切换时更新验证码
137  * @param name tab的名字
138  */
```

```
142   if (!showThirdLogin.value) {
143     refreshQrCode()
144   } else {
145     getCaptcha()
146   }
147 }
148
149 /**
150  * 生成二维码
151  */
152 const refreshQrCode = () => {
153   generateQrCode()
154     .then((r) => {
155       getQrCodeInfo.value.qrCodeId = r.data.qrCodeId
156       getQrCodeInfo.value.imageData = r.data.imageData
157       // 开始轮询获取二维码信息
158       fetchQrCodeInfo(r.data.qrCodeId);
159     })
160     .catch((e: any) => {
161       message.warning(`生成二维码失败：${e.message}`)
162     })
163 }
164
165 /**
166  * 根据二维码id轮询二维码信息
167  * @param qrCodeId 二维码id
168  */
169 const fetchQrCodeInfo = (qrCodeId: string) => {
170   fetch(qrCodeId)
171     .then((r: any) => {
172       if (r.success) {
173         qrCodeInfo.value = r.data
174         if (qrCodeInfo.value.qrCodeStatus !== 0 && qrCodeInfo.value.avatarUrl) {
175           // 只要不是待扫描并且头像不为空
176           getQrCodeInfo.value.imageData = qrCodeInfo.value.avatarUrl
177         }
178
179         if (r.data.qrCodeStatus !== 2 && !qrCodeInfo.value.expired) {
180           if (!showThirdLogin.value) {
181             // 显示三方登录代表不是二维码登录，不轮询；否则继续轮询
182             // 1秒后重复调用
183             setTimeout(() => {
184               fetchQrCodeInfo(qrCodeId)
185             }, 1000);
186           }
187           return
```

```
191           return
192         }
193         if (qrCodeInfo.value.qrCodeStatus === 2) {
194           // 已确认
195           let href = getQueryString('target')
196           if (href) {
197             // 确认后将地址重定向
198             window.location.href = href
199           } else {
200             // 跳转到首页
201             router.push({ path: '/' })
202           }
203         }
204       } else {
205         message.warning(r.message)
206       }
207     })
208     .catch((e: any) => {
209       message.warning(`获取二维码信息失败: ${e.message || e.statusText}`)
210     })
211 }
212
213 /**
214  * 倒计时结束
215  */
216 const onFinish = () => {
217   counterActive.value = false
218 }
219
220 /**
221  * 倒计时显示内容
222  */
223 const renderCountdown: CountdownProps['render'] = ({ hours, minutes, seconds }) => {
224   return `${seconds}`
225 }
226
227 /**
228  * 根据类型发起OAuth2授权申请
229  * @param type 三方OAuth2登录提供商类型
230  */
231 const thirdLogin = (type: string) => {
232   window.location.href = `${import.meta.env.VITE_OAUTH_ISSUER}/oauth2/authorization/${type}`
233 }
234
235 getCaptcha()
236 </script>
```

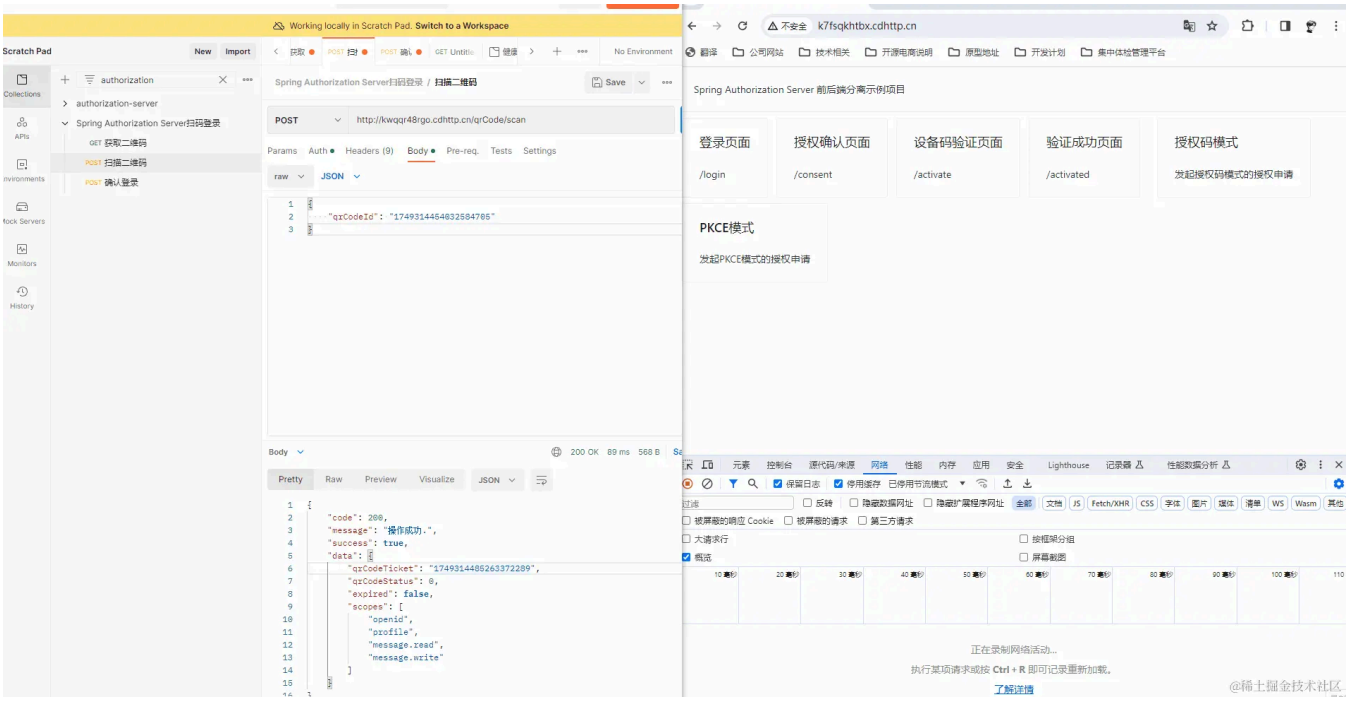
```
240     <img alt="Vue logo" class="logo" src="../../assets/logo.svg" width="125" height="125" />
241
242     <div class="wrapper">
243       <HelloWorld msg="统一认证平台" />
244     </div>
245   </header>
246
247   <main>
248     <n-card title="">
249       <n-tabs default-value="signin" size="large" justify-content="space-evenly" @update:value="
250         <n-tab-pane name="signin" tab="账号登录">
251           <n-form>
252             <n-form-item-row label="用户名">
253               <n-input v-model:value="loginModel.username" placeholder="手机号 / 邮箱" />
254             </n-form-item-row>
255             <n-form-item-row label="密码">
256               <n-input v-model:value="loginModel.password" type="password" show-password-on="mou
257                 placeholder="密码" />
258             </n-form-item-row>
259             <n-form-item-row label="验证码">
260               <n-input-group>
261                 <n-input v-model:value="loginModel.code" placeholder="请输入验证码" />
262                 <n-image @click="getCaptcha" width="130" height="34" :src="captchaImage" preview
263               </n-input-group>
264             </n-form-item-row>
265           </n-form>
266           <n-button type="info" :loading="loading" @click="submitLogin('passwordLogin')" block s
267             登录
268           </n-button>
269         </n-tab-pane>
270         <n-tab-pane name="signup" tab="短信登录">
271           <n-form>
272             <n-form-item-row label="手机号">
273               <n-input v-model:value="loginModel.username" placeholder="手机号 / 邮箱" />
274             </n-form-item-row>
275             <n-form-item-row label="验证码">
276               <n-input-group>
277                 <n-input v-model:value="loginModel.code" placeholder="请输入验证码" />
278                 <n-image @click="getCaptcha" width="130" height="34" :src="captchaImage" preview
279               </n-input-group>
280             </n-form-item-row>
281             <n-form-item-row label="验证码">
282               <n-input-group>
283                 <n-input v-model:value="loginModel.password" placeholder="请输入验证码" />
284                 <n-button type="info" @click="getSmsCaptcha" style="width: 130px" :disabled="cou
285                   获取验证码
```

```
289                       :active="counterActive" />
290                     )</span>
291                 </n-button>
292               </n-input-group>
293             </n-form-item-row>
294           </n-form>
295           <n-button type="info" :loading="loading" @click="submitLogin('smsCaptcha')" block stro
296             登录
297           </n-button>
298         </n-tab-pane>
299         <n-tab-pane name="qrcode" tab="扫码登录" style="text-align: center">
300           <div style="margin: 5.305px">
301             <n-image width="300" :src="getQrCodeInfo.imageData" preview-disabled />
302           </div>
303         </n-tab-pane>
304       </n-tabs>
305       <n-divider style="font-size: 80%; color: #909399">
306         {{ showThirdLogin ? '其它登录方式' : '使用app扫描二维码登录' }}
307       </n-divider>
308       <div class="other_login_icon" v-if="showThirdLogin">
309         <IconGitee :size="32" @click="thirdLogin('gitee')" class="icon_item" />
310         <img width="36" height="36" @click="thirdLogin('github')" src="../../assets/GitHub-Mark.
311         <img width="28" height="28" @click="thirdLogin('wechat')" src="../../assets/wechat_login
312       </div>
313     </n-card>
314   </main>
315 </template>
316
317 <style scoped>
318 .other_login_icon {
319   display: flex;
320   align-items: center;
321   justify-content: center;
322   gap: 0 10px;
323   position: relative;
324   margin-top: -5px;
325 }
326
327 .icon_item {
328   cursor: pointer;
329 }
330
331 header {
332   line-height: 1.5;
333 }
334
```

```
338 }
339
340 @media (min-width: 1024px) {
341   header {
342     display: flex;
343     place-items: center;
344     padding-right: calc(var(--section-gap) / 2);
345   }
346
347   .logo {
348     margin: 0 2rem 0 0;
349   }
350
351   header .wrapper {
352     display: flex;
353     place-items: flex-start;
354     flex-wrap: wrap;
355   }
356 }
357 </style>
```

具体修改内容请看代码仓库qrcode_login分支的二维码登录前端登录页面实现。

效果



代码仓库：Gitee、Github

参考资料

SpringBoot二维码登录(中)
反向工程解析QQ扫码登录的OAuth2流程
聊一聊二维码扫描登录原理

标签： Spring Boot Spring Java 话题： 每天一个知识点

本文收录于以下专栏

1 / 2

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

专栏目录 订阅

上一篇 Spring Authorization Server入门 (十九) ...

评论 6

平等表达，友善交流

0 / 1000 发送

最热 最新

用户807875246022

备码之后，将设备码包装成二维码，然后app扫描可以直接进入授权确认页面。就跟现在电视和手机app同样的流程

1月前 点赞 3 ···

叹雪飞花 作者：没想到设备码这一块儿，不过这样确实会更好一些

1月前 点赞 回复 ···

用户8078752... 回复 叹雪飞花 作者：大佬有时间出一期呀 到时候来学习学习，现在的产品主要都是小程序和app web相当于辅助的了 能融合进去肯定不错

1月前 点赞 回复 ···

查看全部 3 条回复

Kikyou LV.2

大佬又更新了，先赞为敬🤣

2月前 点赞 1 ···

叹雪飞花 作者：感谢支持😍

2月前 点赞 回复 ···

目录 收起

相关推荐

Spring Authorization Server入门 (一) 初识SpringAuthorizationServer和OAuth2.1协议
3.4k阅读 · 18点赞

Spring Authorization Server入门 (七) 登录添加图形验证码
2.9k阅读 · 18点赞

Spring Authorization Server入门 (二) Spring Boot整合Spring Authorization Server
6.8k阅读 · 31点赞

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
4.8k阅读 · 24点赞

Spring Authorization Server入门 (十) 添加短信验证码方式登录
3.4k阅读 · 20点赞

精选内容

面试题：Kafka如何保证高可用？有图有真相
程序员清风 · 197阅读 · 21点赞

Spring Boot集成JSch快速入门demo
HBLOG · 205阅读 · 2点赞

【C++干货基地】C++引用与指针的区别：深入理解两者特性及选择正确应用场景
鸽芷咕 · 133阅读 · 0点赞

【C++干货基地】C++入门篇：输入输出流 | 缺省函数 | 函数重载
鸽芷咕 · 139阅读 · 0点赞

【C++干货基地】namespace超越C语言的独特魅力

为你推荐

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

SpringBoot3.x最简集成SpringDoc-OpenApi

叹雪飞花 4月前 2.3k 17 评论 后端 Spring ... Java

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

叹雪飞花 8月前 1.7k 6 12 Spring Spring ... 安全

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 1.3k 8 19 Spring ... 后端 Redis

Spring Authorization Server入门 (十七) Vue项目使用授权码模式对接认证服务

叹雪飞花 6月前 762 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 767 7 评论 Spring ... Spring ... Java

Spring Authorization Server入门 (十八) Vue项目使用PKCE模式对接认证服务

叹雪飞花 6月前 687 6 4 Vue.js 安全 Spring ...

SpringDoc枚举字段处理与SpringBoot接收枚举参数处理

叹雪飞花 4月前 458 5 评论 后端 Spring ... Java

SpringDoc基础配置和集成OAuth2登录认证教程

叹雪飞花 4月前 363 4 评论 后端 Spring ... Java

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的使用

huan1993 2年前 7.2k 21 6 Spring 后端

Spring 官方发起Spring Authorization Server 项目

码农小胖哥 3年前 4.6k 7 11 Java Spring B...

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 ...