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 2023-08-01 1,586 阅读19分钟

2023-12-01修改：在session-data-redis(Github)分支中添加了基于 `spring-session-data-redis` 的实现，无需借助 `nonceId` 来保持认证状态，该分支已去除所有 `nonceId` 相关内容，需要注意的是 `axios` 在初始化时需要添加配置 `withCredentials: true` ，让请求携带cookie。当然一些响应json的处理还是使用下方的内容。

前言

在之前的文章(实现授权码模式使用前后端分离的登录页面)中实现了前后端分离的登录页面，但这篇文章中只分离了登录页面，鉴于部分读者好奇授权确认页面分离的实现，就实现一下授权确认页面的分离，同时设备码流程的授权确认页面与授权码流程的授权确认页面是同一个，这里也需要兼容一下，还有就是设备码流程中有一个校验设备码的页面，这里也需要分离出来。

前文中有提到，在前后端分离的模式下，在页面发起的请求需要响应json不能重定向了，所以需要修改相关接口调用成功后响应json。话不多说，直接上代码。

编码

需要修改的内容

1. 重定向至授权确认页面时直接携带相关参数重定向至前端项目中
2. 提供接口查询登录用户在发起授权的客户端中相关scope信息
3. 重定向至设备码校验页面时携带当前sessionId(nonceId)重定向至前端项目中
4. 编写授权确认失败处理类，在调用确认授权接口失败时响应json

7. 修改重定向至登录页面处理，兼容在请求校验设备码时登录信息过期处理
8. 将以上内容添加至认证服务配置中
9. 前端项目中编写授权确认、设备码校验、设备码校验成功页面

重定向至授权确认页面时直接携带相关参数重定向至前端项目中

在AuthorizationController中编写/oauth2/consent/redirect接口，借助认证服务跳转至前端的，跳转时携带sessionId保持登录状态

java 复制代码

```java
@SneakyThrows
@ResponseBody
@GetMapping(value = "/oauth2/consent/redirect")
public Result<String> consentRedirect(HttpSession session,
                                      HttpServletRequest request,
                                      HttpServletResponse response,
                                      @RequestParam(OAuth2ParameterNames.SCOPE) String scope,
                                      @RequestParam(OAuth2ParameterNames.STATE) String state,
                                      @RequestParam(OAuth2ParameterNames.CLIENT_ID) String clien
                                      @RequestHeader(name = NONCE_HEADER_NAME, required = false)
                                      @RequestParam(name = OAuth2ParameterNames.USER_CODE, requi

    // 携带当前请求参数与nonceId重定向至前端页面
    UriComponentsBuilder uriBuilder = UriComponentsBuilder
            .fromUriString(CONSENT_PAGE_URI)
            .queryParam(OAuth2ParameterNames.SCOPE, UriUtils.encode(scope, StandardCharsets.UTF_
            .queryParam(OAuth2ParameterNames.STATE, UriUtils.encode(state, StandardCharsets.UTF_
            .queryParam(OAuth2ParameterNames.CLIENT_ID, clientId)
            .queryParam(OAuth2ParameterNames.USER_CODE, userCode)
            .queryParam(NONCE_HEADER_NAME, ObjectUtils.isEmpty(nonceId) ? session.getId() : nonc

    String uriString = uriBuilder.build(Boolean.TRUE).toUriString();
    if (ObjectUtils.isEmpty(userCode) || !UrlUtils.isAbsoluteUrl(DEVICE_ACTIVATE_URI)) {
        // 不是设备码模式或者设备码验证页面不是前后端分离的，无需返回json，直接重定向
        redirectStrategy.sendRedirect(request, response, uriString);
        return null;
    }
    // 兼容设备码，需响应JSON，由前端进行跳转
    return Result.success(uriString);
}
```

在AuthorizationController中编写/oauth2/consent/parameters接口

java 复制代码

```java
@ResponseBody
@GetMapping(value = "/oauth2/consent/parameters")
public Result<Map<String, Object>> consentParameters(Principal principal,
                                                     @RequestParam(OAuth2ParameterNames.CLIENT_I
                                                     @RequestParam(OAuth2ParameterNames.SCOPE) S
                                                     @RequestParam(OAuth2ParameterNames.STATE) S
                                                     @RequestParam(name = OAuth2ParameterNames.U

    // 获取consent页面所需的参数
    Map<String, Object> consentParameters = getConsentParameters(scope, state, clientId, userCod

    return Result.success(consentParameters);
}
```

修改/oauth2/consent接口

java 复制代码

```java
@GetMapping(value = "/oauth2/consent")
public String consent(Principal principal, Model model,
                      @RequestParam(OAuth2ParameterNames.CLIENT_ID) String clientId,
                      @RequestParam(OAuth2ParameterNames.SCOPE) String scope,
                      @RequestParam(OAuth2ParameterNames.STATE) String state,
                      @RequestParam(name = OAuth2ParameterNames.USER_CODE, required = false) Str

    // 获取consent页面所需的参数
    Map<String, Object> consentParameters = getConsentParameters(scope, state, clientId, userCod
    // 转至model中，让框架渲染页面
    consentParameters.forEach(model::addAttribute);

    return "consent";
}
```

编写公共方法getConsentParameters

java 复制代码

```
 *
 * @param scope     scope权限
 * @param state     state
 * @param clientId  客户端id
 * @param userCode  设备码授权流程中的用户码
 * @param principal 当前认证信息
 * @return 页面所需数据
 */
private Map<String, Object> getConsentParameters(String scope,
                                                 String state,
                                                 String clientId,
                                                 String userCode,
                                                 Principal principal) {
    // Remove scopes that were already approved
    Set<String> scopesToApprove = new HashSet<>();
    Set<String> previouslyApprovedScopes = new HashSet<>();
    RegisteredClient registeredClient = this.registeredClientRepository.findByClientId(clientId)
    if (registeredClient == null) {
        throw new RuntimeException("客户端不存在");
    }
    OAuth2AuthorizationConsent currentAuthorizationConsent =
            this.authorizationConsentService.findById(registeredClient.getId(), principal.getNam
    Set<String> authorizedScopes;
    if (currentAuthorizationConsent != null) {
        authorizedScopes = currentAuthorizationConsent.getScopes();
    } else {
        authorizedScopes = Collections.emptySet();
    }
    for (String requestedScope : StringUtils.delimitedListToStringArray(scope, " ")) {
        if (OidcScopes.OPENID.equals(requestedScope)) {
            continue;
        }
        if (authorizedScopes.contains(requestedScope)) {
            previouslyApprovedScopes.add(requestedScope);
        } else {
            scopesToApprove.add(requestedScope);
        }
    }

    Map<String, Object> parameters = new HashMap<>(7);
    parameters.put("clientId", registeredClient.getClientId());
    parameters.put("clientName", registeredClient.getClientName());
    parameters.put("state", state);
    parameters.put("scopes", withDescription(scopesToApprove));
    parameters.put("previouslyApprovedScopes", withDescription(previouslyApprovedScopes));
    parameters.put("principalName", principal.getName());
```

```
    } else {
        parameters.put("requestURI", "/oauth2/authorize");
    }
    return parameters;
}
```

重定向至设备码校验页面时携带当前sessionId(nonceId)重定向至前端项目中

在AuthorizationController中编写/activate/redirect接口，由认证服务重定向，携带sessionId以保持登录状态

java 复制代码

```java
@GetMapping("/activate/redirect")
public String activateRedirect(HttpSession session,
                               @RequestParam(value = "user_code", required = false) String userC

    UriComponentsBuilder uriBuilder = UriComponentsBuilder
            .fromUriString(DEVICE_ACTIVATE_URI)
            .queryParam("userCode", userCode)
            .queryParam(NONCE_HEADER_NAME, session.getId());
    return "redirect:" + uriBuilder.build(Boolean.TRUE).toUriString();
}
```

编写授权确认失败处理类，在调用确认授权接口失败时响应json

java 复制代码

```java
package com.example.authorization.handler;

import com.example.model.Result;
import com.example.util.JsonUtils;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
```

```java
import org.springframework.security.oauth2.core.OAuth2AuthenticationException;
import org.springframework.security.oauth2.core.OAuth2Error;
import org.springframework.security.web.authentication.AuthenticationFailureHandler;
import org.springframework.security.web.util.UrlUtils;

import java.io.IOException;
import java.nio.charset.StandardCharsets;

import static com.example.constant.SecurityConstants.CONSENT_PAGE_URI;

/**
 * 授权确认失败处理
 *
 * @author vains
 */
public class ConsentAuthenticationFailureHandler implements AuthenticationFailureHandler {

    @Override
    public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response
        // 获取当前认证信息
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        // 获取具体的异常
        OAuth2AuthenticationException authenticationException = (OAuth2AuthenticationException)
        OAuth2Error error = authenticationException.getError();
        // 异常信息
        String message;
        if (authentication == null) {
            message = "登录已失效";
        } else {
            // 第二次点击“拒绝”会因为之前取消时删除授权申请记录而找不到对应的数据，导致抛出 [invalid_re
            message = error.toString();
        }

        // 授权确认页面提交的请求，因为授权申请与授权确认提交公用一个过滤器，这里判断一下
        if (request.getMethod().equals(HttpMethod.POST.name()) && UrlUtils.isAbsoluteUrl(CONSENT
            // 写回json异常
            Result<Object> result = Result.error(HttpStatus.BAD_REQUEST.value(), message);
            response.setCharacterEncoding(StandardCharsets.UTF_8.name());
            response.setContentType(MediaType.APPLICATION_JSON_VALUE);
            response.getWriter().write(JsonUtils.objectCovertToJson(result));
            response.getWriter().flush();
        } else {
            // 在地址栏输入授权申请地址或设备码流程的验证地址错误(user_code错误)
            response.sendError(HttpStatus.BAD_REQUEST.value(), error.toString());
        }

```

编写授权成功处理类，在调用授权确认接口成功时响应json

java 复制代码

```java
package com.example.authorization.handler;

import com.example.model.Result;
import com.example.util.JsonUtils;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.http.HttpMethod;
import org.springframework.http.MediaType;
import org.springframework.security.core.Authentication;
import org.springframework.security.oauth2.core.OAuth2Error;
import org.springframework.security.oauth2.core.endpoint.OAuth2ParameterNames;
import org.springframework.security.oauth2.server.authorization.authentication.OAuth2Authorizati
import org.springframework.security.oauth2.server.authorization.authentication.OAuth2Authorizati
import org.springframework.security.web.DefaultRedirectStrategy;
import org.springframework.security.web.RedirectStrategy;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.springframework.security.web.util.UrlUtils;
import org.springframework.util.ObjectUtils;
import org.springframework.util.StringUtils;
import org.springframework.web.util.UriComponentsBuilder;
import org.springframework.web.util.UriUtils;

import java.io.IOException;
import java.nio.charset.StandardCharsets;

import static com.example.constant.SecurityConstants.CONSENT_PAGE_URI;
import static org.springframework.security.oauth2.core.OAuth2ErrorCodes.INVALID_REQUEST;

/**
 * 授权确认前后端分离适配响应处理
 *
 * @author vains
 */
public class ConsentAuthorizationResponseHandler implements AuthenticationSuccessHandler {

    private final RedirectStrategy redirectStrategy = new DefaultRedirectStrategy();

```

```
        String redirectUri = this.getAuthorizationResponseUri(authentication);
        if (request.getMethod().equals(HttpMethod.POST.name()) && UrlUtils.isAbsoluteUrl(CONSENT
            // 如果是post请求并且CONSENT_PAGE_URI是完整的地址，则响应json
            Result<String> success = Result.success(redirectUri);
            response.setCharacterEncoding(StandardCharsets.UTF_8.name());
            response.setContentType(MediaType.APPLICATION_JSON_VALUE);
            response.getWriter().write(JsonUtils.objectCovertToJson(success));
            response.getWriter().flush();
            return;
        }
        // 否则重定向至回调地址
        this.redirectStrategy.sendRedirect(request, response, redirectUri);
    }

    /**
     * 获取重定向的回调地址
     *
     * @param authentication 认证信息
     * @return 地址
     */
    private String getAuthorizationResponseUri(Authentication authentication) {

        OAuth2AuthorizationCodeRequestAuthenticationToken authorizationCodeRequestAuthentication
                (OAuth2AuthorizationCodeRequestAuthenticationToken) authentication;
        if (ObjectUtils.isEmpty(authorizationCodeRequestAuthentication.getRedirectUri())) {
            String authorizeUriError = "Redirect uri is not null";
            throw new OAuth2AuthorizationCodeRequestAuthenticationException(new OAuth2Error(INVA
        }

        if (authorizationCodeRequestAuthentication.getAuthorizationCode() == null) {
            String authorizeError = "AuthorizationCode is not null";
            throw new OAuth2AuthorizationCodeRequestAuthenticationException(new OAuth2Error(INVA
        }

        UriComponentsBuilder uriBuilder = UriComponentsBuilder
                .fromUriString(authorizationCodeRequestAuthentication.getRedirectUri())
                .queryParam(OAuth2ParameterNames.CODE, authorizationCodeRequestAuthentication.ge
        if (StringUtils.hasText(authorizationCodeRequestAuthentication.getState())) {
            uriBuilder.queryParam(
                    OAuth2ParameterNames.STATE,
                    UriUtils.encode(authorizationCodeRequestAuthentication.getState(), StandardC
        }
        // build(true) -> Components are explicitly encoded
        return uriBuilder.build(true).toUriString();

    }
```

编写校验设备码成功响应类，在校验设备码成功后响应json

java 复制代码

```java
package com.example.authorization.handler;

import com.example.model.Result;
import com.example.util.JsonUtils;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.http.MediaType;
import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;

import java.io.IOException;
import java.nio.charset.StandardCharsets;

import static com.example.constant.SecurityConstants.DEVICE_ACTIVATED_URI;

/**
 * 校验设备码成功响应类
 *
 * @author vains
 */
public class DeviceAuthorizationResponseHandler implements AuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response
        // 写回json数据
        Result<Object> result = Result.success(DEVICE_ACTIVATED_URI);
        response.setCharacterEncoding(StandardCharsets.UTF_8.name());
        response.setContentType(MediaType.APPLICATION_JSON_VALUE);
        response.getWriter().write(JsonUtils.objectCovertToJson(result));
        response.getWriter().flush();
    }
}

```

修改重定向至登录页面处理，兼容在请求校验设备码时登录信息过期处理

```java

import com.example.constant.SecurityConstants;
import com.example.model.Result;
import com.example.util.JsonUtils;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.DefaultRedirectStrategy;
import org.springframework.security.web.RedirectStrategy;
import org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint;
import org.springframework.security.web.util.UrlUtils;
import org.springframework.util.ObjectUtils;

import java.io.IOException;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;

import static com.example.constant.SecurityConstants.DEVICE_ACTIVATE_URI;

/**
 * 重定向至登录处理
 *
 * @author vains
 */
@Slf4j
public class LoginTargetAuthenticationEntryPoint extends LoginUrlAuthenticationEntryPoint {

    private final RedirectStrategy redirectStrategy = new DefaultRedirectStrategy();

    /**
     * @param loginFormUrl URL where the login page can be found. Should either be
     *                     relative to the web-app context path (include a leading {@code /}) or
     *                     URL.
     */
    public LoginTargetAuthenticationEntryPoint(String loginFormUrl) {
        super(loginFormUrl);
    }

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response, Authenticatio
        String deviceVerificationUri = "/oauth2/device_verification";
```

```
                && UrlUtils.isAbsoluteUrl(DEVICE_ACTIVATE_URI)) {
            // 如果是请求验证设备激活码(user_code)时未登录并且设备码验证页面是前后端分离的那种则写回jsc
            Result<String> success = Result.error(HttpStatus.UNAUTHORIZED.value(), ("登录已失效,
            response.setCharacterEncoding(StandardCharsets.UTF_8.name());
            response.setContentType(MediaType.APPLICATION_JSON_VALUE);
            response.getWriter().write(JsonUtils.objectCovertToJson(success));
            response.getWriter().flush();
            return;
        }

        // 获取登录表单的地址
        String loginForm = determineUrlToUseForThisRequest(request, response, authException);
        if (!UrlUtils.isAbsoluteUrl(loginForm)) {
            // 不是绝对路径调用父类方法处理
            super.commence(request, response, authException);
            return;
        }

        StringBuffer requestUrl = request.getRequestURL();
        if (!ObjectUtils.isEmpty(request.getQueryString())) {
            requestUrl.append("?").append(request.getQueryString());
        }

        // 2023-07-11添加逻辑：重定向地址添加nonce参数，该参数的值为sessionId
        // 绝对路径在重定向前添加target参数
        String targetParameter = URLEncoder.encode(requestUrl.toString(), StandardCharsets.UTF_8
        String targetUrl = loginForm + "?target=" + targetParameter + "&" + SecurityConstants.NO
        log.debug("重定向至前后端分离的登录页面：{}", targetUrl);
        this.redirectStrategy.sendRedirect(request, response, targetUrl);
    }
}
```

将以上内容添加至认证服务配置中

AuthorizationConfig完整配置如下

java 复制代码

```java
package com.example.config;

import com.example.authorization.device.DeviceClientAuthenticationConverter;
```

```
import com.example.authorization.sms.SmsCaptchaGrantAuthenticationConverter;
import com.example.authorization.sms.SmsCaptchaGrantAuthenticationProvider;
import com.example.authorization.wechat.WechatAuthorizationRequestConsumer;
import com.example.authorization.wechat.WechatCodeGrantRequestEntityConverter;
import com.example.authorization.wechat.WechatMapAccessTokenResponseConverter;
import com.example.constant.RedisConstants;
import com.example.constant.SecurityConstants;
import com.example.support.RedisOperator;
import com.example.support.RedisSecurityContextRepository;
import com.example.util.SecurityUtils;
import com.nimbusds.jose.jwk.JWKSet;
import com.nimbusds.jose.jwk.RSAKey;
import com.nimbusds.jose.jwk.source.ImmutableJWKSet;
import com.nimbusds.jose.jwk.source.JWKSource;
import com.nimbusds.jose.proc.SecurityContext;
import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.MediaType;
import org.springframework.http.converter.FormHttpMessageConverter;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.security.access.annotation.Secured;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.authentication.configuration.Authenticatio
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.oauth2.client.endpoint.DefaultAuthorizationCodeTokenResponse
import org.springframework.security.oauth2.client.endpoint.OAuth2AccessTokenResponseClient;
import org.springframework.security.oauth2.client.endpoint.OAuth2AuthorizationCodeGrantRequest;
import org.springframework.security.oauth2.client.http.OAuth2ErrorResponseErrorHandler;
import org.springframework.security.oauth2.client.registration.ClientRegistrationRepository;
import org.springframework.security.oauth2.client.web.DefaultOAuth2AuthorizationRequestResolver;
import org.springframework.security.oauth2.client.web.OAuth2AuthorizationRequestRedirectFilter;
import org.springframework.security.oauth2.client.web.OAuth2AuthorizationRequestResolver;
import org.springframework.security.oauth2.core.AuthorizationGrantType;
import org.springframework.security.oauth2.core.ClientAuthenticationMethod;
import org.springframework.security.oauth2.core.http.converter.OAuth2AccessTokenResponseHttpMess
import org.springframework.security.oauth2.core.oidc.OidcScopes;
import org.springframework.security.oauth2.jwt.JwtDecoder;
import org.springframework.security.oauth2.server.authorization.JdbcOAuth2AuthorizationConsentSe
```

```
import org.springframework.security.oauth2.server.authorization.client.JdbcRegisteredClientRepos
import org.springframework.security.oauth2.server.authorization.client.RegisteredClient;
import org.springframework.security.oauth2.server.authorization.client.RegisteredClientRepositor
import org.springframework.security.oauth2.server.authorization.config.annotation.web.configurat
import org.springframework.security.oauth2.server.authorization.config.annotation.web.configurer
import org.springframework.security.oauth2.server.authorization.settings.AuthorizationServerSett
import org.springframework.security.oauth2.server.authorization.settings.ClientSettings;
import org.springframework.security.oauth2.server.authorization.token.JwtEncodingContext;
import org.springframework.security.oauth2.server.authorization.token.OAuth2TokenCustomizer;
import org.springframework.security.oauth2.server.authorization.token.OAuth2TokenGenerator;
import org.springframework.security.oauth2.server.resource.authentication.JwtAuthenticationConve
import org.springframework.security.oauth2.server.resource.authentication.JwtGrantedAuthoritiesC
import org.springframework.security.web.DefaultSecurityFilterChain;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.util.UrlUtils;
import org.springframework.security.web.util.matcher.MediaTypeRequestMatcher;
import org.springframework.util.ObjectUtils;
import org.springframework.web.client.RestTemplate;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;

import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.UUID;

import static com.example.constant.SecurityConstants.CONSENT_PAGE_URI;
import static com.example.constant.SecurityConstants.DEVICE_ACTIVATE_URI;

/**
 * 认证配置
 * {@link EnableMethodSecurity} 开启全局方法认证，启用JSR250注解支持，启用注解 {@link Secured} 支持，
 * 在Spring Security 6.0版本中将@Configuration注解从@EnableWebSecurity, @EnableMethodSecurity, @En
 * 和 @EnableGlobalAuthentication 中移除，使用这些注解需手动添加 @Configuration 注解
 * {@link EnableWebSecurity} 注解有两个作用:
 * 1. 加载了WebSecurityConfiguration配置类，配置安全认证策略。
 * 2. 加载了AuthenticationConfiguration，配置了认证信息。
 *
 * @author vains
 */
@Configuration
```

```
public class AuthorizationConfig {

    private final RedisOperator<String> redisOperator;

    /**
     * 登录地址，前后端分离就填写完整的url路径，不分离填写相对路径
     */
    private final String LOGIN_URL = "http://127.0.0.1:5173/login";

    private static final String CUSTOM_CONSENT_REDIRECT_URI = "/oauth2/consent/redirect";

    private static final String CUSTOM_DEVICE_REDIRECT_URI = "/activate/redirect";

    private final RedisSecurityContextRepository redisSecurityContextRepository;

    /**
     * 配置端点的过滤器链
     *
     * @param http spring security核心配置类
     * @return 过滤器链
     * @throws Exception 抛出
     */
    @Bean
    public SecurityFilterChain authorizationServerSecurityFilterChain(HttpSecurity http,
                                                                      RegisteredClientRepository
                                                                      AuthorizationServerSetting
        // 配置默认的设置，忽略认证端点的csrf校验
        OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);

        // 添加跨域过滤器
        http.addFilter(corsFilter());
        // 禁用 csrf 与 cors
        http.csrf(AbstractHttpConfigurer::disable);
        http.cors(AbstractHttpConfigurer::disable);

        // 新建设备码converter和provider
        DeviceClientAuthenticationConverter deviceClientAuthenticationConverter =
                new DeviceClientAuthenticationConverter(
                        authorizationServerSettings.getDeviceAuthorizationEndpoint());
        DeviceClientAuthenticationProvider deviceClientAuthenticationProvider =
                new DeviceClientAuthenticationProvider(registeredClientRepository);

        // 使用redis存储、读取登录的认证信息
        http.securityContext(context -> context.securityContextRepository(redisSecurityContextRe

        http.getConfigurer(OAuth2AuthorizationServerConfigurer.class)
```

```
154                 .authorizationEndpoint(authorizationEndpoint -> {
155                             // 校验授权确认页面是否为完整路径；是否是前后端分离的页面
156                             boolean absoluteUrl = UrlUtils.isAbsoluteUrl(CONSENT_PAGE_URI);
157                             // 如果是分离页面则重定向，否则转发请求
158                             authorizationEndpoint.consentPage(absoluteUrl ? CUSTOM_CONSENT_REDIR
159                             if (absoluteUrl) {
160                                 // 适配前后端分离的授权确认页面，成功/失败响应json
161                                 authorizationEndpoint.errorResponseHandler(new ConsentAuthentica
162                                 authorizationEndpoint.authorizationResponseHandler(new ConsentAu
163                             }
164                     }
165                 )
166                 // 设置设备码用户验证url(自定义用户验证页)
167                 .deviceAuthorizationEndpoint(deviceAuthorizationEndpoint ->
168                         deviceAuthorizationEndpoint.verificationUri(UrlUtils.isAbsoluteUrl(DEVIC
169                 )
170                 // 设置验证设备码用户确认页面
171                 .deviceVerificationEndpoint(deviceVerificationEndpoint -> {
172                             // 校验授权确认页面是否为完整路径；是否是前后端分离的页面
173                             boolean absoluteUrl = UrlUtils.isAbsoluteUrl(CONSENT_PAGE_URI);
174                             // 如果是分离页面则重定向，否则转发请求
175                             deviceVerificationEndpoint.consentPage(absoluteUrl ? CUSTOM_CONSENT_
176                             if (absoluteUrl) {
177                                 // 适配前后端分离的授权确认页面，失败响应json
178                                 deviceVerificationEndpoint.errorResponseHandler(new ConsentAuthe
179                             }
180                             // 如果授权码验证页面或者授权确认页面是前后端分离的
181                             if (UrlUtils.isAbsoluteUrl(DEVICE_ACTIVATE_URI) || absoluteUrl) {
182                                 // 添加响应json处理
183                                 deviceVerificationEndpoint.deviceVerificationResponseHandler(new
184                             }
185                     }
186                 )
187                 .clientAuthentication(clientAuthentication ->
188                         // 客户端认证添加设备码的converter和provider
189                         clientAuthentication
190                                 .authenticationConverter(deviceClientAuthenticationConverter)
191                                 .authenticationProvider(deviceClientAuthenticationProvider)
192                 );
193         http
194                 // 当未登录时访问认证端点时重定向至login页面
195                 .exceptionHandling((exceptions) -> exceptions
196                         .defaultAuthenticationEntryPointFor(
197                                 new LoginTargetAuthenticationEntryPoint(LOGIN_URL),
198                                 new MediaTypeRequestMatcher(MediaType.TEXT_HTML)
199                         )
```

```
                        .jwt(Customizer.withDefaults()));

        // 自定义短信认证登录转换器
        SmsCaptchaGrantAuthenticationConverter converter = new SmsCaptchaGrantAuthenticationConv
        // 自定义短信认证登录认证提供
        SmsCaptchaGrantAuthenticationProvider provider = new SmsCaptchaGrantAuthenticationProvid
        http.getConfigurer(OAuth2AuthorizationServerConfigurer.class)
                // 让认证服务器元数据中有自定义的认证方式
                .authorizationServerMetadataEndpoint(metadata -> metadata.authorizationServerMet
                // 添加自定义grant_type——短信认证登录
                .tokenEndpoint(tokenEndpoint -> tokenEndpoint
                        .accessTokenRequestConverter(converter)
                        .authenticationProvider(provider));

        DefaultSecurityFilterChain build = http.build();

        // 从框架中获取provider中所需的bean
        OAuth2TokenGenerator<?> tokenGenerator = http.getSharedObject(OAuth2TokenGenerator.class
        AuthenticationManager authenticationManager = http.getSharedObject(AuthenticationManager
        OAuth2AuthorizationService authorizationService = http.getSharedObject(OAuth2Authorizati
        // 以上三个bean在build()方法之后调用是因为调用build方法时框架会尝试获取这些类，
        // 如果获取不到则初始化一个实例放入SharedObject中，所以要在build方法调用之后获取
        // 在通过set方法设置进provider中，但是如果在build方法之后调用authenticationProvider(provider
        // 框架会提示unsupported_grant_type，因为已经初始化完了，在添加就不会生效了
        provider.setTokenGenerator(tokenGenerator);
        provider.setAuthorizationService(authorizationService);
        provider.setAuthenticationManager(authenticationManager);

        return build;
    }

    /**
     * 配置认证相关的过滤器链(资源服务，客户端配置)
     *
     * @param http spring security核心配置类
     * @return 过滤器链
     * @throws Exception 抛出
     */
    @Bean
    public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http, ClientRegistrationR
        // 添加跨域过滤器
        http.addFilter(corsFilter());
        // 禁用 csrf 与 cors
        http.csrf(AbstractHttpConfigurer::disable);
        http.cors(AbstractHttpConfigurer::disable);
        http.authorizeHttpRequests((authorize) -> authorize
```

```
                )
                // 指定登录页面
                .formLogin(formLogin -> {
                        formLogin.loginPage("/login");
                        if (UrlUtils.isAbsoluteUrl(LOGIN_URL)) {
                            // 绝对路径代表是前后端分离，登录成功和失败改为写回json，不重定向了
                            formLogin.successHandler(new LoginSuccessHandler());
                            formLogin.failureHandler(new LoginFailureHandler());
                        }
                    }
                );
        // 添加BearerTokenAuthenticationFilter，将认证服务当做一个资源服务，解析请求头中的token
        http.oauth2ResourceServer((resourceServer) -> resourceServer
                .jwt(Customizer.withDefaults())
                .accessDeniedHandler(SecurityUtils::exceptionHandler)
                .authenticationEntryPoint(SecurityUtils::exceptionHandler)
        );
        // 兼容前后端分离与不分离配置
        http
                // 当未登录时访问认证端点时重定向至login页面
                .exceptionHandling((exceptions) -> exceptions
                        .defaultAuthenticationEntryPointFor(
                                new LoginTargetAuthenticationEntryPoint(LOGIN_URL),
                                new MediaTypeRequestMatcher(MediaType.TEXT_HTML)
                        )
                );
        // 联合身份认证
        http.oauth2Login(oauth2Login -> oauth2Login
                .loginPage(LOGIN_URL)
                .authorizationEndpoint(authorization -> authorization
                        .authorizationRequestResolver(this.authorizationRequestResolver(clientRe
                )
                .tokenEndpoint(token -> token
                        .accessTokenResponseClient(this.accessTokenResponseClient())
                )
        );

        // 使用redis存储、读取登录的认证信息
        http.securityContext(context -> context.securityContextRepository(redisSecurityContextRe

        return http.build();
    }

    /**
     * AuthorizationRequest 自定义配置
     *
```

```
    private OAuth2AuthorizationRequestResolver authorizationRequestResolver(ClientRegistrationRe
        DefaultOAuth2AuthorizationRequestResolver authorizationRequestResolver =
                new DefaultOAuth2AuthorizationRequestResolver(
                        clientRegistrationRepository, OAuth2AuthorizationRequestRedirectFilter.D

        // 兼容微信登录授权申请
        authorizationRequestResolver.setAuthorizationRequestCustomizer(new WechatAuthorizationRe

        return authorizationRequestResolver;
    }

    /**
     * 适配微信登录适配，添加自定义请求token入参处理
     *
     * @return OAuth2AccessTokenResponseClient accessToken响应信息处理
     */
    private OAuth2AccessTokenResponseClient<OAuth2AuthorizationCodeGrantRequest> accessTokenResp
        DefaultAuthorizationCodeTokenResponseClient tokenResponseClient = new DefaultAuthorizati
        tokenResponseClient.setRequestEntityConverter(new WechatCodeGrantRequestEntityConverter(
        // 自定义 RestTemplate，适配微信登录获取token
        OAuth2AccessTokenResponseHttpMessageConverter messageConverter = new OAuth2AccessTokenRe
        List<MediaType> mediaTypes = new ArrayList<>(messageConverter.getSupportedMediaTypes());
        // 微信获取token时响应的类型为“text/plain”，这里特殊处理一下
        mediaTypes.add(MediaType.TEXT_PLAIN);
        messageConverter.setAccessTokenResponseConverter(new WechatMapAccessTokenResponseConvert
        messageConverter.setSupportedMediaTypes(mediaTypes);

        // 初始化RestTemplate
        RestTemplate restTemplate = new RestTemplate(Arrays.asList(
                new FormHttpMessageConverter(),
                messageConverter));

        restTemplate.setErrorHandler(new OAuth2ErrorResponseErrorHandler());
        tokenResponseClient.setRestOperations(restTemplate);
        return tokenResponseClient;
    }

    /**
     * 跨域过滤器配置
     *
     * @return CorsFilter
     */
    @Bean
    public CorsFilter corsFilter() {

        // 初始化cors配置对象
```

```
        configuration.addAllowedOrigin("http://127.0.0.1:5173");
        configuration.addAllowedOrigin("http://192.168.1.102:5173");
        // 设置跨域访问可以携带cookie
        configuration.setAllowCredentials(true);
        // 允许所有的请求方法 ==> GET POST PUT Delete
        configuration.addAllowedMethod("*");
        // 允许携带任何头信息
        configuration.addAllowedHeader("*");

        // 初始化cors配置源对象
        UrlBasedCorsConfigurationSource configurationSource = new UrlBasedCorsConfigurationSourc

        // 给配置源对象设置过滤的参数
        // 参数一: 过滤的路径 == > 所有的路径都要求校验是否跨域
        // 参数二: 配置类
        configurationSource.registerCorsConfiguration("/**", configuration);

        // 返回配置好的过滤器
        return new CorsFilter(configurationSource);
    }

    /**
     * 自定义jwt，将权限信息放至jwt中
     *
     * @return OAuth2TokenCustomizer的实例
     */
    @Bean
    public OAuth2TokenCustomizer<JwtEncodingContext> oAuth2TokenCustomizer() {
        return new FederatedIdentityIdTokenCustomizer();
    }

    /**
     * 自定义jwt解析器，设置解析出来的权限信息的前缀与在jwt中的key
     *
     * @return jwt解析器 JwtAuthenticationConverter
     */
    @Bean
    public JwtAuthenticationConverter jwtAuthenticationConverter() {
        JwtGrantedAuthoritiesConverter grantedAuthoritiesConverter = new JwtGrantedAuthoritiesCo
        // 设置解析权限信息的前缀，设置为空是去掉前缀
        grantedAuthoritiesConverter.setAuthorityPrefix("");
        // 设置权限信息在jwt claims中的key
        grantedAuthoritiesConverter.setAuthoritiesClaimName(SecurityConstants.AUTHORITIES_KEY);

        JwtAuthenticationConverter jwtAuthenticationConverter = new JwtAuthenticationConverter()
        jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter(grantedAuthoritiesConverter
```

```

    /**
     * 将AuthenticationManager注入ioc中，其它需要使用地方可以直接从ioc中获取
     *
     * @param authenticationConfiguration 导出认证配置
     * @return AuthenticationManager 认证管理器
     */
    @Bean
    @SneakyThrows
    public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticatio
        return authenticationConfiguration.getAuthenticationManager();
    }

    /**
     * 配置密码解析器，使用BCrypt的方式对密码进行加密和验证
     *
     * @return BCryptPasswordEncoder
     */
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    /**
     * 配置客户端Repository
     *
     * @param jdbcTemplate    db 数据源信息
     * @param passwordEncoder 密码解析器
     * @return 基于数据库的repository
     */
    @Bean
    public RegisteredClientRepository registeredClientRepository(JdbcTemplate jdbcTemplate, Pass
        RegisteredClient registeredClient = RegisteredClient.withId(UUID.randomUUID().toString()
                // 客户端id
                .clientId("messaging-client")
                // 客户端秘钥，使用密码解析器加密
                .clientSecret(passwordEncoder.encode("123456"))
                // 客户端认证方式，基于请求头的认证
                .clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT_SECRET_BASIC)
                // 配置资源服务器使用该客户端获取授权时支持的方式
                .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
                .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
                .authorizationGrantType(AuthorizationGrantType.CLIENT_CREDENTIALS)
                // 客户端添加自定义认证
                .authorizationGrantType(new AuthorizationGrantType(SecurityConstants.GRANT_TYPE_
                // 授权码模式回调地址，oauth2.1已改为精准匹配，不能只设置域名，并且屏蔽了localhost，本
```

```
                .scope(OidcScopes.OPENID)
                .scope(OidcScopes.PROFILE)
                // 自定scope
                .scope("message.read")
                .scope("message.write")
                // 客户端设置，设置用户需要确认授权
                .clientSettings(ClientSettings.builder().requireAuthorizationConsent(true).build
                .build();

        // 基于db存储客户端，还有一个基于内存的实现 InMemoryRegisteredClientRepository
        JdbcRegisteredClientRepository registeredClientRepository = new JdbcRegisteredClientRepo

        // 初始化客户端
        RegisteredClient repositoryByClientId = registeredClientRepository.findByClientId(regist
        if (repositoryByClientId == null) {
            registeredClientRepository.save(registeredClient);
        }
        // 设备码授权客户端
        RegisteredClient deviceClient = RegisteredClient.withId(UUID.randomUUID().toString())
                .clientId("device-message-client")
                // 公共客户端
                .clientAuthenticationMethod(ClientAuthenticationMethod.NONE)
                // 设备码授权
                .authorizationGrantType(AuthorizationGrantType.DEVICE_CODE)
                .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
                // 自定scope
                .scope("message.read")
                .scope("message.write")
                .build();
        RegisteredClient byClientId = registeredClientRepository.findByClientId(deviceClient.get
        if (byClientId == null) {
            registeredClientRepository.save(deviceClient);
        }

        // PKCE客户端
        RegisteredClient pkceClient = RegisteredClient.withId(UUID.randomUUID().toString())
                .clientId("pkce-message-client")
                // 公共客户端
                .clientAuthenticationMethod(ClientAuthenticationMethod.NONE)
                // 授权码模式，因为是扩展授权码流程，所以流程还是授权码的流程，改变的只是参数
                .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
                .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
                // 授权码模式回调地址，oauth2.1已改为精准匹配，不能只设置域名，并且屏蔽了localhost，本
                .redirectUri("http://127.0.0.1:8080/login/oauth2/code/messaging-client-oidc")
                .clientSettings(ClientSettings.builder().requireProofKey(Boolean.TRUE).build())
                // 自定scope
```

```
        RegisteredClient findPkceClient = registeredClientRepository.findByClientId(pkceClient.g
        if (findPkceClient == null) {
            registeredClientRepository.save(pkceClient);
        }
        return registeredClientRepository;
    }

    /**
     * 配置基于db的oauth2的授权管理服务
     *
     * @param jdbcTemplate               db数据源信息
     * @param registeredClientRepository 上边注入的客户端repository
     * @return JdbcOAuth2AuthorizationService
     */
    @Bean
    public OAuth2AuthorizationService authorizationService(JdbcTemplate jdbcTemplate, Registered
        // 基于db的oauth2认证服务，还有一个基于内存的服务实现InMemoryOAuth2AuthorizationService
        return new JdbcOAuth2AuthorizationService(jdbcTemplate, registeredClientRepository);
    }

    /**
     * 配置基于db的授权确认管理服务
     *
     * @param jdbcTemplate               db数据源信息
     * @param registeredClientRepository 客户端repository
     * @return JdbcOAuth2AuthorizationConsentService
     */
    @Bean
    public OAuth2AuthorizationConsentService authorizationConsentService(JdbcTemplate jdbcTempla
        // 基于db的授权确认管理服务，还有一个基于内存的服务实现InMemoryOAuth2AuthorizationConsentServ
        return new JdbcOAuth2AuthorizationConsentService(jdbcTemplate, registeredClientRepositor
    }

    /**
     * 配置jwk源，使用非对称加密，公开用于检索匹配指定选择器的JWK的方法
     *
     * @return JWKSource
     */
    @Bean
    @SneakyThrows
    public JWKSource<SecurityContext> jwkSource() {
        // 先从redis获取
        String jwkSetCache = redisOperator.get(RedisConstants.AUTHORIZATION_JWS_PREFIX_KEY);
        if (ObjectUtils.isEmpty(jwkSetCache)) {
            KeyPair keyPair = generateRsaKey();
            RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();
```

```
                    .keyID(UUID.randomUUID().toString())
                    .build();
            // 生成jws
            JWKSet jwkSet = new JWKSet(rsaKey);
            // 转为json字符串
            String jwkSetString = jwkSet.toString(Boolean.FALSE);
            // 存入redis
            redisOperator.set(RedisConstants.AUTHORIZATION_JWS_PREFIX_KEY, jwkSetString);
            return new ImmutableJWKSet<>(jwkSet);
        }
        // 解析存储的jws
        JWKSet jwkSet = JWKSet.parse(jwkSetCache);
        return new ImmutableJWKSet<>(jwkSet);
    }

    /**
     * 生成rsa密钥对，提供给jwk
     *
     * @return 密钥对
     */
    private static KeyPair generateRsaKey() {
        KeyPair keyPair;
        try {
            KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
            keyPairGenerator.initialize(2048);
            keyPair = keyPairGenerator.generateKeyPair();
        } catch (Exception ex) {
            throw new IllegalStateException(ex);
        }
        return keyPair;
    }

    /**
     * 配置jwt解析器
     *
     * @param jwkSource jwk源
     * @return JwtDecoder
     */
    @Bean
    public JwtDecoder jwtDecoder(JWKSource<SecurityContext> jwkSource) {
        return OAuth2AuthorizationServerConfiguration.jwtDecoder(jwkSource);
    }

    /**
     * 添加认证服务器配置，设置jwt签发者、默认端点请求地址等
     *
```

```
    public AuthorizationServerSettings authorizationServerSettings() {
        return AuthorizationServerSettings.builder()
                /*
                    设置token签发地址(http(s)://{ip}:{port}/context-path, http(s)://domain.com/co
                    如果需要通过ip访问这里就是ip，如果是有域名映射就填域名，通过什么方式访问该服务这里就
                 */
                .issuer("http://192.168.1.102:8080")
                .build();
    }

}
```

前端项目中编写授权确认、设备码校验、设备码校验成功页面

编写授权确认页面Consent.vue

html 复制代码

```html
<script setup lang="ts">
import { type Ref, ref } from 'vue'
import axios from 'axios'
import { createDiscreteApi } from 'naive-ui'

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

// 获取授权确认信息响应
const consentResult: Ref<any> = ref()
// 所有的scope
const scopes = ref()
// 已授权的scope
const approvedScopes = ref()

axios({
  method: 'GET',
  url: `http://192.168.1.102:8080/oauth2/consent/parameters${window.location.search}`
})
  .then((r) => {
    let result = r.data
    if (result.success) {
      consentResult.value = result.data
      scopes.value = [...result.data.previouslyApprovedScopes, ...result.data.scopes]
      approvedScopes.value = result.data.previouslyApprovedScopes.map((e: any) => e.scope)
```

```
  })
  .catch((e) => message.error(e.message))

/**
 * 提交授权确认
 *
 * @param cancel true为取消
 */
const submitApprove = (cancel: boolean) => {
  const data = new FormData()
  if (!cancel) {
    // 如果不是取消添加scope
    if (
      approvedScopes.value !== null &&
      typeof approvedScopes.value !== 'undefined' &&
      approvedScopes.value.length > 0
    ) {
      approvedScopes.value.forEach((e: any) => data.append('scope', e))
    }
  }
  data.append('state', consentResult.value.state)
  data.append('client_id', consentResult.value.clientId)
  data.append('user_code', consentResult.value.userCode)
  axios({
    method: 'POST',
    // @ts-ignore
    data: new URLSearchParams(data),
    headers: {
      nonceId: getQueryString('nonceId'),
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    url: `http://192.168.1.102:8080${consentResult.value.requestURI}`
  })
    .then((r) => {
      let result = r.data
      if (result.success) {
        window.location.href = result.data
      } else {
        if (result.message && result.message.indexOf('access_denied') > -1) {
          // 可以跳转至一个单独的页面提醒.
          message.warning('您未选择scope或拒绝了本次授权申请.')
        } else {
          message.warning(result.message)
        }
      }
    })
```

```
/**
 * 获取地址栏参数
 * @param name 地址栏参数的key
 */
function getQueryString(name: string) {
  var reg = new RegExp('(^|&)' + name + '=([^&]*)(&|$)', 'i')

  var r = window.location.search.substr(1).match(reg)

  if (r != null) {
    return unescape(r[2])
  }

  return null
}
</script>

<template>
  <header>
    <img alt="Vue logo" class="logo" src="../../assets/logo.svg" width="125" height="125" />

    <div class="wrapper">
      <HelloWorld msg="OAuth 授权请求" />
    </div>
  </header>

  <main>
    <n-card v-if="consentResult && consentResult.userCode">
      您已经提供了代码
      <b>{{ consentResult.userCode }}</b>
      ，请验证此代码是否与设备上显示的代码匹配。
    </n-card>
    <br />
    <n-card :title="`${consentResult.clientName} 客户端`" v-if="consentResult">
      <template #header-extra>
        账号:
        <b>{{ consentResult.principalName }}</b>
      </template>
      此第三方应用请求获得以下权限
    </n-card>
    <n-scrollbar style="max-height: 230px">
      <n-checkbox-group v-model:value="approvedScopes">
        <n-list>
          <n-list-item v-for="scope in scopes">
            <template #prefix>
              <n-checkbox :value="scope.scope"> </n-checkbox>
```

```
126         </n-list>
127       </n-checkbox-group>
128     </n-scrollbar>
129     <br />
130     <n-button type="info" @click="submitApprove(false)" strong>
131           确    定    
132     </n-button>
133         
134     <n-button type="warning" @click="submitApprove(true)">
135           拒    绝    
136     </n-button>
137   </main>
138 </template>
139
140 <style scoped>
141 header {
142   line-height: 1.5;
143 }
144
145 .logo {
146   display: block;
147   margin: 0 auto 2rem;
148 }
149
150 @media (min-width: 1024px) {
151   header {
152     display: flex;
153     place-items: center;
154     padding-right: calc(var(--section-gap) / 2);
155   }
156
157   .logo {
158     margin: 0 2rem 0 0;
159   }
160
161   header .wrapper {
162     display: flex;
163     place-items: flex-start;
164     flex-wrap: wrap;
165   }
166 }
167
168 b,
169 h3,
170 ::v-deep(.n-card-header__main) {
171   font-weight: bold !important;
```

编写设备码验证页面Activate.vue

html 复制代码

```html
<script setup lang="ts">
import { ref } from 'vue'
import axios from 'axios'
import { createDiscreteApi } from 'naive-ui'

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

const userCode = ref({
  userCode: getQueryString('userCode')
})

/**
 * 提交授权确认
 *
 * @param cancel true为取消
 */
const submit = () => {
  const data = {
    user_code: userCode.value.userCode
  }
  axios({
    method: 'POST',
    data,
    headers: {
      nonceId: getQueryString('nonceId'),
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    url: `http://192.168.1.102:8080/oauth2/device_verification`
  })
    .then((r) => {
      let result = r.data
      if (result.success) {
        window.location.href = result.data
      } else {
        message.warning(result.message)
      }
    })
    .catch((e) => message.error(e.message))
}

```

```

/**
 * 获取地址栏参数
 * @param name 地址栏参数的key
 */
function getQueryString(name: string) {
  var reg = new RegExp('(^|&)' + name + '=([^&]*)(&|$)', 'i')

  var r = window.location.search.substr(1).match(reg)

  if (r != null) {
    return unescape(r[2])
  }

  return null
}
</script>

<template>
  <header>
    <img alt="Vue logo" class="logo" src="../../assets/devices.png" width="125" height="125" />

    <div class="wrapper">
      <HelloWorld msg="设备激活" />
    </div>
  </header>

  <main>
    <n-card> 输入激活码对设备进行授权。 </n-card>
    <br />
    <n-card>
      <n-form-item-row label="Activation Code">
        <n-input
          v-model:value="userCode.userCode"
          placeholder="User Code"
          maxlength="9"
          show-count
          clearable
        />
      </n-form-item-row>
      <n-button type="info" @click="submit" block strong> 登录 </n-button>
    </n-card>
  </main>
</template>

<style scoped>
```

```

.logo {
  display: block;
  margin: 0 auto 2rem;
}

@media (min-width: 1024px) {
  header {
    display: flex;
    place-items: center;
    padding-right: calc(var(--section-gap) / 2);
  }

  .logo {
    margin: 0 2rem 0 0;
  }

  header .wrapper {
    display: flex;
    place-items: flex-start;
    flex-wrap: wrap;
  }
}

b,
h3,
::v-deep(.n-card-header__main) {
  font-weight: bold !important;
}
</style>

```

编写设备码验证成功页面

html 复制代码

```html
<script lang="ts" setup></script>
<template>
  <header>
    <img alt="Vue logo" class="logo" src="../../assets/devices.png" width="125" height="125" />

    <div class="wrapper">
      <HelloWorld msg="设备激活" />
    </div>
  </header>
```

```
      您已成功激活您的设备。
      <br />
      请返回到您的设备继续。
    </div>
  </main>
</template>
<style scoped>
header {
  line-height: 1.5;
}

.logo {
  display: block;
  margin: 0 auto 2rem;
}

@media (min-width: 1024px) {
  header {
    display: flex;
    place-items: center;
    padding-right: calc(var(--section-gap) / 2);
  }

  .logo {
    margin: 0 2rem 0 0;
  }

  header .wrapper {
    display: flex;
    place-items: flex-start;
    flex-wrap: wrap;
  }
}

b,
h3,
::v-deep(.n-card-header__main) {
  font-weight: bold !important;
}
</style>

```

vue-router路由配置index.ts

typescript 复制代码

```
const router = createRouter({
    history: createWebHistory(import.meta.env.BASE_URL),
    routes: [
        {
            path: '/login',
            name: 'login',
            component: () => import('../views/login/Login.vue')
        },
        {
            path: '/consent',
            name: 'consent',
            // route level code-splitting
            // this generates a separate chunk (About.[hash].js) for this route
            // which is lazy-loaded when the route is visited.
            component: () => import('../views/consent/Consent.vue')
        },
        {
            path: '/activate',
            name: 'activate',
            // route level code-splitting
            // this generates a separate chunk (About.[hash].js) for this route
            // which is lazy-loaded when the route is visited.
            component: () => import('../views/device/Activate.vue')
        },
        {
            path: '/activated',
            name: 'activated',
            // route level code-splitting
            // this generates a separate chunk (About.[hash].js) for this route
            // which is lazy-loaded when the route is visited.
            component: () => import('../views/device/Activated.vue')
        }
    ]
})

export default router

```

附一下常量类SecurityConstants

java 复制代码

```
package com.example.constant;

```

```
 * @author vains
 */
public class SecurityConstants {

    /**
     * 授权确认页面地址
     */
    public static final String DEVICE_ACTIVATED_URI = "http://127.0.0.1:5173/activated";

    /**
     * 授权确认页面地址
     */
    public static final String DEVICE_ACTIVATE_URI = "http://127.0.0.1:5173/activate";

    /**
     * 授权确认页面地址
     */
    public static final String CONSENT_PAGE_URI = "http://127.0.0.1:5173/consent";

    /**
     * 微信登录相关参数——openid：用户唯一id
     */
    public static final String WECHAT_PARAMETER_OPENID = "openid";

    /**
     * 微信登录相关参数——forcePopup：强制此次授权需要用户弹窗确认
     */
    public static final String WECHAT_PARAMETER_FORCE_POPUP = "forcePopup";

    /**
     * 微信登录相关参数——secret：微信的应用秘钥
     */
    public static final String WECHAT_PARAMETER_SECRET = "secret";

    /**
     * 微信登录相关参数——appid：微信的应用id
     */
    public static final String WECHAT_PARAMETER_APPID = "appid";

    /**
     * 三方登录类型——微信
     */
    public static final String THIRD_LOGIN_WECHAT = "wechat";

    /**
     * 三方登录类型——Gitee
```

```
    /**
     * 三方登录类型——Github
     */
    public static final String THIRD_LOGIN_GITHUB = "github";

    /**
     * 随机字符串请求头名字
     */
    public static final String NONCE_HEADER_NAME = "nonceId";

    /**
     * 登录方式入参名
     */
    public static final String LOGIN_TYPE_NAME = "loginType";

    /**
     * 验证码id入参名
     */
    public static final String CAPTCHA_ID_NAME = "captchaId";

    /**
     * 验证码值入参名
     */
    public static final String CAPTCHA_CODE_NAME = "code";

    /**
     * 登录方式——短信验证码
     */
    public static final String SMS_LOGIN_TYPE = "smsCaptcha";

    /**
     * 登录方式——账号密码登录
     */
    public static final String PASSWORD_LOGIN_TYPE = "passwordLogin";

    /**
     * 权限在token中的key
     */
    public static final String AUTHORITIES_KEY = "authorities";

    /**
     * 自定义 grant type —— 短信验证码
     */
    public static final String GRANT_TYPE_SMS_CODE = "urn:ietf:params:oauth:grant-type:sms_code"

    /**
```

```
    /**
     * 自定义 grant type —— 短信验证码 —— 短信验证码的key
     */
    public static final String OAUTH_PARAMETER_NAME_SMS_CAPTCHA = "sms_captcha";

}

```

到此为止编码就结束了

最后

因为理论部分在之前的文章中已经讲过了，这次就没写理论了，直接贴了一大堆的代码，本次代码写的比较仓促，测试的也不是很全面，如果发现有什么问题可以在评论区留言。
代码仓库地址

- 重定向至授权确认页面时直接携带相关参数重定向至前端项目中
- 提供接口查询登录用户在发起授权的客户端中相关scope信息
  - 在AuthorizationController中编写/oauth2/consent/parameters接口
  - 修改/oauth2/consent接口
  - 编写公共方法getConsentParameters
- 重定向至设备码校验页面时携带当前sessionId(nonceId)重定向至前端项目中
- 编写授权确认失败处理类，在调用确认授权接口失败时响应json
- 编写授权成功处理类，在调用授权确认接口成功时响应json
- 编写校验设备码成功响应类，在校验设备码成功后响应json
- 修改重定向至登录页面处理，兼容在请求校验设备码时登录信息过期处理
- 将以上内容添加至认证服务配置中
- 前端项目中编写授权确认、设备码校验、设备码校验成功页面
  - 编写授权确认页面Consent.vue
  - 编写设备码验证页面Activate.vue
  - 编写设备码验证成功页面
  - vue-router路由配置index.ts
- 附一下常量类SecurityConstants

最后

相关推荐

20个SpringSecurity框架核心组件详解
威哥爱编程 · 162阅读 · 1点赞

写一个编译器非常简单（第 2 部分）
xuejianxinokok · 69阅读 · 0点赞

MongoDB的使用（索引、聚合、整合应用、副本集、分片集群）
王空空 · 157阅读 · 0点赞

LongAdder升级版的AtomicLong
isfox · 70阅读 · 1点赞

java工具-高并发-JUC下Semaphore解密
重庆穿山甲 · 76阅读 · 1点赞

为你推荐

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

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 (二十) 实现二维码扫码登录
叹雪飞花 2月前 935 16 6 Spring ... Spring Java

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

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 684 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 362 4 评论 后端 Spring ... Java

Spring Authorization Server 授权服务器

编程说 1年前 6.2k 25 9 Spring ...

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 全新授权服务器整合使用

冷冷zz 3年前 3.2k 16 3 Java Spring B...