

Spring Authorization Server (6) 授权服务器 授权类型扩展

爱吃西瓜的胖娃 2023-09-19 👁 424 ⌚ 阅读5分钟

关注

架构版本

Spring Boot 3.1

Spring Authorization Server 1.1.1

spring-cloud 2022.0.3

spring-cloud-alibaba 2022.0.0.0

完整代码👉 [watermelon-cloud](#)

Spring Authorization Server 授权服务常见的授权模式

授权码模式 --> authorization_code

密码模式 --> password

客户端模式 --> client_credentials

Spring Authorization Server 内置的 授权模式 都定义再 `AuthorizationGrantType` 里面了

▼ java



复制代码

```
1 public final class AuthorizationGrantType implements Serializable {
2
3     private static final long serialVersionUID = SpringSecurityCoreVersion.SERIAL_VERSION_U
4
5     public static final AuthorizationGrantType AUTHORIZATION_CODE = new AuthorizationGrantT
6
7     public static final AuthorizationGrantType REFRESH_TOKEN = new AuthorizationGrantType("
8
9     public static final AuthorizationGrantType CLIENT_CREDENTIALS = new AuthorizationGrantT
10
11     @Deprecated
```

```
15 public static final AuthorizationGrantType JWT_BEARER = new AuthorizationGrantType(  
16     "urn:ietf:params:oauth:grant-type:jwt-bearer");  
17  
18 public static final AuthorizationGrantType DEVICE_CODE = new AuthorizationGrantType(  
19     "urn:ietf:params:oauth:grant-type:device_code");  
20  
21 private final String value;  
22  
23 }
```

如果有一些特定的场景，内置的授权模式不支持，那我们如何去扩展呢？

我也不知道怎么去扩展，那我们就看看 [授权码模式](#) 是怎么玩的，然后参考它的去做扩展。

还是先从入口看，前面我们也讲到了，`/oauth2/token` 对于的 `OAuth2TokenEndpointFilter` 核心代码也就是之前提到的2行代码

▼ java



复制代码

```
1 Authentication authorizationGrantAuthentication = this.authenticationConverter.convert(request);  
2  
3 OAuth2AccessTokenAuthenticationToken accessTokenAuthentication = (OAuth2AccessTokenAuthenticationToken)
```

授权码模式 `OAuth2AuthorizationCodeAuthenticationConverter`、
`OAuth2AuthorizationCodeAuthenticationToken`、
`OAuth2AuthorizationCodeAuthenticationProvider` 是怎么玩的，我们就怎么玩。

那我现在理解搞定这 `OAuth2AuthorizationCodeAuthenticationConverter`、
`OAuth2AuthorizationCodeAuthenticationToken`、`OAuth2AuthorizationCodeAuthenticationProvider`
3个就行了。

手机号+短信验证码模式扩展

SmsAuthenticationToken

▼ java



复制代码

```
1 public class SmsAuthenticationToken extends AbstractAuthenticationToken {
2
3     /**
4      * 认证类型
5      */
6     private AuthorizationGrantType authorizationGrantType;
7     /**
8      * 用户名
9      */
10    private Authentication clientPrincipal;
11    /**
12     * 手机号
13     */
14    private String phone;
15    /**
16     * 短信验证码
17     */
18    private String code;
19    /**
20     * scopes
21     */
22    private Set<String> scopes;
23    /**
24     * 扩展的参数
25     */
26    private Map<String, Object> additionalParameters;
27
28    public SmsAuthenticationToken(
29        AuthorizationGrantType authorizationGrantType,
30        Authentication clientPrincipal,
31        Set<String> scopes,
32        String phone,
33        String code,
34        Map<String, Object> additionalParameters) {
35        super(Collections.emptyList());
36
37        Assert.notNull(clientPrincipal, "clientPrincipal cannot be null");
38        this.scopes = Collections.unmodifiableSet(
```

```
42
43     this.phone = phone;
44     this.code = code;
45     this.clientPrincipal = clientPrincipal;
46     this.additionalParameters = Collections.unmodifiableMap(
47         additionalParameters != null ?
48             new HashMap<>(additionalParameters) :
49             Collections.emptyMap());
50     this.authorizationGrantType = authorizationGrantType;
51 }
52
53 /**
54  * 扩展模式一般不需要密码
55  */
56 @Override
57 public Object getCredentials() {
58     return null;
59 }
60
61 /**
62  * 获取用户名
63  */
64 @Override
65 public Object getPrincipal() {
66     return this.clientPrincipal;
67 }
68
69
70 public String getPhone() {
71     return phone;
72 }
73
74
75 public String getCode() {
76     return code;
77 }
78
79 /**
80  * 获取请求的scopes
81  */
82 public Set<String> getScopes() {
83     return this.scopes;
84 }
85
86 /**
87  * 获取请求中的 grant_type
```

```
91     }
92
93     /**
94      * 获取请求中的附加参数
95      */
96     public Map<String, Object> getAdditionalParameters() {
97         return this.additionalParameters;
98     }
99
100 }
```

SmsAuthenticationConverter

▼ java



复制代码

```
1  public final class SmsAuthenticationConverter implements AuthenticationConverter {
2
3      @Nullable
4      @Override
5      public Authentication convert(HttpServletRequest request) {
6          // grant_type (REQUIRED)
7          String grantType = request.getParameter(OAuth2ParameterNames.GRANT_TYPE);
8          if (!AuthorizationServerConfigurationConsent.GRANT_TYPE_SMS_CODE.equals(grantType)) {
9              return null;
10         }
11         Authentication clientPrincipal = SecurityContextHolder.getContext().getAuthentication();
12         //OAuth2AuthorizationUtils 是copy 源码中存在的
13         MultiValueMap<String, String> parameters = OAuth2AuthorizationUtils.getParameters(request);
14
15
16         // scope (OPTIONAL)
17         Set<String> scopes = null;
18         String scope = parameters.getFirst(OAuth2ParameterNames.SCOPE);
19         if (StringUtils.hasText(scope) &&
20             parameters.get(OAuth2ParameterNames.SCOPE).size() != 1) {
21             OAuth2AuthorizationUtils.throwError(OAuth2ErrorCodes.INVALID_REQUEST, OAuth2ParameterNames.SCOPE);
22         }
23         if (StringUtils.hasText(scope)) {
24             scopes = new HashSet<>();
25             Arrays.asList(StringUtils.delimitedListToStringArray(scope, " "));
```



```
29      // phone (REQUIRED) 手机号
30      String phone = parameters.getFirst(AuthorizationServerConfigurationConsent.OAUTH2_I
31      if (!StringUtils.hasText(phone) ||
32          parameters.get(AuthorizationServerConfigurationConsent.OAUTH2_PARAMETER_NAI
33          OAuth2AuthorizationUtils.throwError(
34              OAuth2ErrorCodes.INVALID_REQUEST,
35              "OAuth 2.0 Parameter: " + AuthorizationServerConfigurationConsent.OAUTI
36              null);
37      }
38
39      // sms_code (REQUIRED) 验证码必填
40      String smsCode = parameters.getFirst(AuthorizationServerConfigurationConsent.OAUTH:
41      if (!StringUtils.hasText(smsCode) ||
42          parameters.get(AuthorizationServerConfigurationConsent.OAUTH2_PARAMETER_NAI
43          OAuth2AuthorizationUtils.throwError(
44              OAuth2ErrorCodes.INVALID_REQUEST,
45              "OAuth 2.0 Parameter: " + AuthorizationServerConfigurationConsent.OAUTI
46              null);
47      }
48
49      //扩展参数
50      Map<String, Object> additionalParameters = new HashMap<>();
51      parameters.forEach((key, value) -> {
52          if (!key.equals(OAuth2ParameterNames.GRANT_TYPE)) {
53              additionalParameters.put(key, value.get(0));
54          }
55      });
56
57      return new SmsAuthenticationToken(new AuthorizationGrantType(AuthorizationServerCo
58          clientPrincipal,
59          scopes,
60          phone,
61          smsCode,
62          additionalParameters);
63      }
64  }
```

SmsAuthenticationProvider

```
2 public final class SmsAuthenticationProvider implements AuthenticationProvider {
3
4     private static final String ERROR_URI = "https://datatracker.ietf.org/doc/html/rfc6749";
5
6     private OAuth2AuthorizationService authorizationService;
7
8     private OAuth2TokenGenerator<? extends OAuth2Token> tokenGenerator;
9
10    private AuthenticationManager authenticationManager; //暂时没有用到 如果额外加一个 短信验证
11
12    private static final OAuth2TokenType ID_TOKEN_TOKEN_TYPE =
13        new OAuth2TokenType(OidcParameterNames.ID_TOKEN);
14
15
16    @Override
17    public Authentication authenticate(Authentication authentication) throws AuthenticationException {
18
19        SmsAuthenticationToken smsAuthenticationToken = (SmsAuthenticationToken) authentication;
20
21        OAuth2ClientAuthenticationToken clientPrincipal =
22            OAuth2AuthorizationUtils.getAuthenticatedClientElseThrowInvalidClient(smsAuthenticationToken);
23
24        RegisteredClient registeredClient = clientPrincipal.getRegisteredClient();
25        if (log.isTraceEnabled()) {
26            log.trace("Retrieved registered client");
27        }
28        if (registeredClient == null) {
29            throw new OAuth2AuthenticationException("client_id not exist");
30        }
31
32        try {
33            //todo 验证码验证逻辑
34            //验证码
35            String code = smsAuthenticationToken.getCode();
36            if (!StringUtils.hasText(code)) {
37                throw new OAuth2AuthenticationException("验证码不能为空!");
38            }
39            //todo 暂时先写000000，发送验证码的我们还没有写的
40            if (!code.equals("000000")) {
41                throw new OAuth2AuthenticationException("验证码: 【" + code + "】已过期!");
42            }
43            Authentication smsCodeValidAuthentication = smsAuthenticationToken;
44            smsAuthenticationToken.setAuthenticated(true);
45            // @formatter:off
46            DefaultOAuth2TokenContext.Builder tokenContextBuilder = DefaultOAuth2TokenContext.Builder
47                .registeredClient(registeredClient)
48                .principal(smsCodeValidAuthentication)
```

```
52         .authorizationGrant(smsAuthenticationToken);
53     // @formatter:on
54     OAuth2Authorization.Builder authorizationBuilder = OAuth2Authorization
55         .withRegisteredClient(registeredClient)
56         .principalName(smsCodeValidAuthentication.getName())
57         .authorizationGrantType(smsAuthenticationToken.getAuthorizationGrantType())
58         .authorizedScopes(smsAuthenticationToken.getScopes());
59
60     // ----- Access token -----
61     OAuth2TokenContext tokenContext = tokenContextBuilder.tokenType(OAuth2TokenType.ACCESS_TOKEN);
62     OAuth2Token generatedAccessToken = this.tokenGenerator.generate(tokenContext);
63     if (generatedAccessToken == null) {
64         OAuth2Error error = new OAuth2Error(OAuth2ErrorCodes.SERVER_ERROR,
65             "The token generator failed to generate the access token.", ERROR_URI);
66         throw new OAuth2AuthenticationException(error);
67     }
68
69     if (log.isTraceEnabled()) {
70         log.trace("Generated access token");
71     }
72
73     OAuth2AccessToken accessToken = new OAuth2AccessToken(OAuth2AccessToken.TokenType.BEARER,
74         generatedAccessToken.getTokenValue(), generatedAccessToken.getIssuedAt(),
75         generatedAccessToken.getExpiresAt(), tokenContext.getAuthorizedScopes());
76     if (generatedAccessToken instanceof ClaimAccessor) {
77         authorizationBuilder.token(accessToken, (metadata) ->
78             metadata.put(OAuth2Authorization.Token.CLAIMS_METADATA_NAME,
79                 .attribute(Principal.class.getName(), smsCodeValidAuthentication));
80     } else {
81         authorizationBuilder.accessToken(accessToken);
82     }
83
84     // ----- Refresh token -----
85     OAuth2RefreshToken refreshToken = null;
86     if (registeredClient.getAuthorizationGrantTypes().contains(AuthorizationGrantType.REFRESH_TOKEN)) {
87         // Do not issue refresh token to public client
88         !clientPrincipal.getClientAuthenticationMethod().equals(ClientAuthenticationMethod.PUBLIC_CLIENT);
89
90         tokenContext = tokenContextBuilder.tokenType(OAuth2TokenType.REFRESH_TOKEN);
91         OAuth2Token generatedRefreshToken = this.tokenGenerator.generate(tokenContext);
92         if (!(generatedRefreshToken instanceof OAuth2RefreshToken)) {
93             OAuth2Error error = new OAuth2Error(OAuth2ErrorCodes.SERVER_ERROR,
94                 "The token generator failed to generate the refresh token.", ERROR_URI);
95             throw new OAuth2AuthenticationException(error);
96         }
97     }
```




```
101
102         refreshToken = (OAuth2RefreshToken) generatedRefreshToken;
103         authorizationBuilder.refreshToken(refreshToken);
104     }
105
106     // ----- ID token -----
107     OidcIdToken idToken;
108     if (smsAuthenticationToken.getScopes().contains(OidcScopes.OPENID)) {
109
110         // @formatter:off
111         tokenContext = tokenContextBuilder
112             .tokenType(ID_TOKEN_TOKEN_TYPE)
113             .authorization(authorizationBuilder.build()) // ID token custom
114             .build();
115         // @formatter:on
116         OAuth2Token generatedIdToken = this.tokenGenerator.generate(tokenContext);
117         if (!(generatedIdToken instanceof Jwt)) {
118             OAuth2Error error = new OAuth2Error(OAuth2ErrorCodes.SERVER_ERROR,
119                 "The token generator failed to generate the ID token.", ERROR_I
120             throw new OAuth2AuthenticationException(error);
121         }
122
123         if (log.isTraceEnabled()) {
124             log.trace("Generated id token");
125         }
126
127         idToken = new OidcIdToken(generatedIdToken.getTokenValue(), generatedIdTok
128             generatedIdToken.getExpiresAt(), ((Jwt) generatedIdToken).getClaim
129         authorizationBuilder.token(idToken, (metadata) ->
130             metadata.put(OAuth2Authorization.Token.CLAIMS_METADATA_NAME, idTok
131     } else {
132         idToken = null;
133     }
134     OAuth2Authorization authorization = authorizationBuilder.build();
135     this.authorizationService.save(authorization);
136
137     if (log.isTraceEnabled()) {
138         log.trace("Saved authorization");
139     }
140     Map<String, Object> additionalParameters = Collections.emptyMap();
141     if (idToken != null) {
142         additionalParameters = new HashMap<>();
143         additionalParameters.put(OidcParameterNames.ID_TOKEN, idToken.getTokenValu
144     }
145     if (log.isTraceEnabled()) {
146         log.trace("Authenticated token request");
```

```
150         } catch (Exception e) {
151
152             OAuth2Error error = new OAuth2Error(OAuth2ErrorCodes.SERVER_ERROR,
153                 e.getMessage(), ERROR_URI);
154
155             throw new OAuth2AuthenticationException(error);
156         }
157
158     }
159     @Override
160     public boolean supports(Class<?> authentication) {
161         return SmsAuthenticationToken.class.isAssignableFrom(authentication);
162     }
163     public void setTokenGenerator(OAuth2TokenGenerator<?> tokenGenerator) {
164         Assert.notNull(tokenGenerator, "tokenGenerator cannot be null");
165         this.tokenGenerator = tokenGenerator;
166     }
167     public void setAuthenticationManager(AuthenticationManager authenticationManager) {
168         Assert.notNull(authenticationManager, "authenticationManager cannot be null");
169         this.authenticationManager = authenticationManager;
170     }
171     public void setAuthorizationService(OAuth2AuthorizationService authorizationService) {
172         Assert.notNull(authorizationService, "authorizationService cannot be null");
173         this.authorizationService = authorizationService;
174     }
175 }
```

OAuth2AuthorizationService authorizationService;

OAuth2TokenGenerator<? extends OAuth2Token> tokenGenerator;

AuthenticationManager authenticationManager; 以上3个属性，是在 `HttpSecurity` build 之后才能够从 `HttpSecurity` 中获取到，所以 `SmsAuthenticationProvider` 的注入就要在 `HttpSecurity` build 之后再添加到 `SecurityFilterChain` 中。

SmsAuthenticationProvider、SmsAuthenticationConverter 注入SecurityFilterChain中

在 `AuthorizationServerConfig` 的 `SecurityFilterChain` @Bean 中添加，因为你看官方给的demo 中就是这里添加，再有一个原因就是 `AuthorizationServerConfig` 中的 `SecurityFilterChain` 就是 `oauth2` 的配置。



▼ java



复制代码

```
1  @Configuration(proxyBeanMethods = false)
2  public class AuthorizationServerConfig {
3
4      private static final String CUSTOM_CONSENT_PAGE_URI = "/oauth2/consent";//这个是授权页
5
6      @Bean
7      @Order(Ordered.HIGHEST_PRECEDENCE)
8      public SecurityFilterChain authorizationServerSecurityFilterChain(
9          HttpSecurity http, RegisteredClientRepository registeredClientRepository,
10         AuthorizationServerSettings authorizationServerSettings) throws Exception {
11
12         OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);
13
14         DeviceClientAuthenticationConverter deviceClientAuthenticationConverter =
15             new DeviceClientAuthenticationConverter(
16                 authorizationServerSettings.getDeviceAuthorizationEndpoint());
17         DeviceClientAuthenticationProvider deviceClientAuthenticationProvider =
18             new DeviceClientAuthenticationProvider(registeredClientRepository);
19         // @formatter:off
20         http.getConfigurer(OAuth2AuthorizationServerConfigurer.class)
21             .deviceAuthorizationEndpoint(deviceAuthorizationEndpoint ->
22                 deviceAuthorizationEndpoint.verificationUri("/activate")
23             )
24             .deviceVerificationEndpoint(deviceVerificationEndpoint ->
25                 deviceVerificationEndpoint.consentPage(CUSTOM_CONSENT_PAGE_URI)
26             )
27             .clientAuthentication(clientAuthentication ->
28                 clientAuthentication
29                     .authenticationConverter(deviceClientAuthenticationConverter)
30                     .authenticationProvider(deviceClientAuthenticationProvider)
31             )
32             .authorizationEndpoint(authorizationEndpoint ->
33                 authorizationEndpoint.consentPage(CUSTOM_CONSENT_PAGE_URI))
34             .oidc(Customizer.withDefaults());    // Enable OpenID Connect 1.0
35         // @formatter:on
36
37         // @formatter:off
38         http
39             .exceptionHandling(exceptions -> exceptions
40                 .defaultAuthenticationEntryPointFor(
41                     new LoginUrlAuthenticationEntryPoint(AuthorizationServerConfig
42                         .new MediaTypeRequestMatcher(MediaType.TEXT_HTML)
43                     )
44             )
```



```
48
49     //sms off
50     SmsAuthenticationConverter smsAuthenticationConverter = new SmsAuthenticationConver
51     SmsAuthenticationProvider smsAuthenticationProvider = new SmsAuthenticationProvide
52     //sms on
53     http.getConfigurer(OAuth2AuthorizationServerConfigurer.class)
54         .tokenEndpoint(tokenEndpoint->
55             tokenEndpoint.accessTokenRequestConverter(smsAuthenticationConverter)
56                 .authenticationProvider(smsAuthenticationProvider)//选
57         );
58
59
60     DefaultSecurityFilterChain build = http.build();
61     this.initAuthenticationProviderFiled(http, smsAuthenticationProvider, smsCodeValid
62     return build;
63 }
64
65
66 @Bean
67 public OAuth2AuthorizationService authorizationService(JdbcTemplate jdbcTemplate,
68     RegisteredClientRepository regi
69     return new JdbcOAuth2AuthorizationService(jdbcTemplate, registeredClientRepository
70 }
71
72 @Bean
73 public OAuth2AuthorizationConsentService authorizationConsentService(JdbcTemplate jdbc
74     return new JdbcOAuth2AuthorizationConsentService(jdbcTemplate, registeredClientRep
75 }
76
77 @Bean
78 public OAuth2TokenCustomizer<JwtEncodingContext> idTokenCustomizer() {
79     return new FederatedIdentityIdTokenCustomizer();
80 }
81
82
83 @Bean
84 public AuthorizationServerSettings authorizationServerSettings() {
85     return AuthorizationServerSettings.builder().build();
86 }
87
88
89 /**
90  * 初始化 Provider 中的 OAuth2TokenGenerator、AuthenticationManager、OAuth2Authorizati
91  * @param http
92  * @param providers
93  */
```

手机号+验证码模式演示

POST

http://127.0.0.1:9000/oauth2/token?grant_type=sms_code&phone=70000000005&code=000000

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

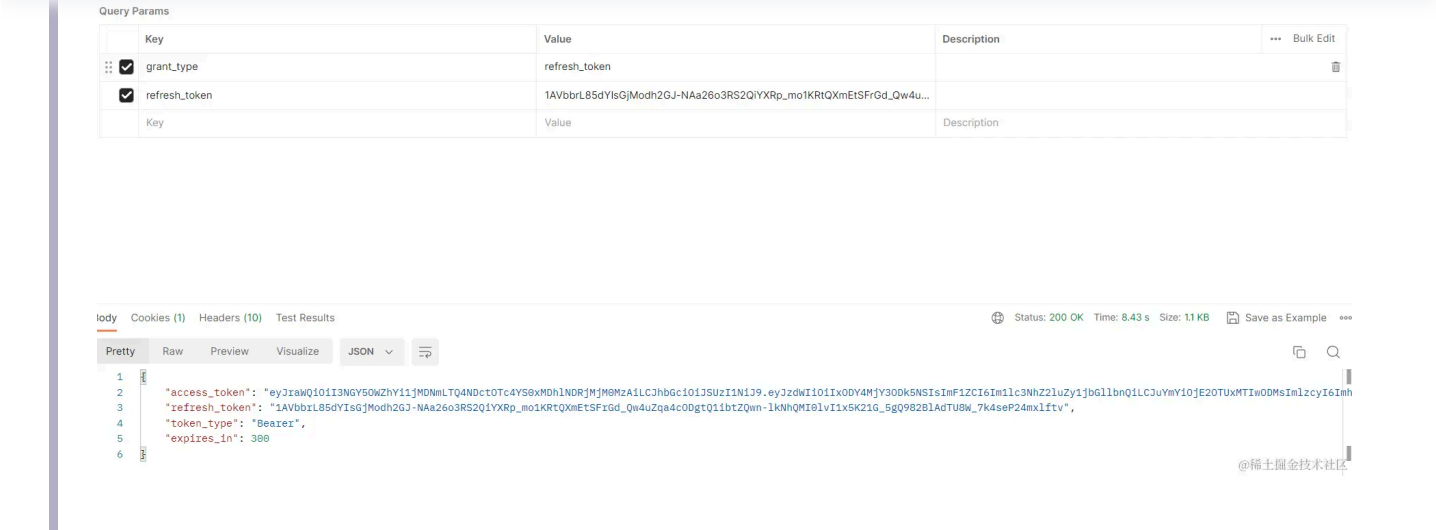
Tests

Settings

Query Params

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> grant_type	sms_code		
<input checked="" type="checkbox"/> phone	18000000000		
<input checked="" type="checkbox"/> code	000000		
Key	Value	Description	

13/17




扩展也是很so easy，不要把spring想的过于复杂了。

标签： Spring Boot Spring Cloud

话题： 1024一起掘金

本文收录于以下专栏



 Spring Authorization Server 精讲

专栏目录

本专栏将深入讲解Spring Authorization Server在实践中的扩展点，希...

56 订阅 · 11 篇文章

订阅

上一篇  Spring Authorization Ser...

下一篇  Spring Authorization Ser...

评论 0



登录 / 注册 即可发布评论！

暂无评论数据




目录

收起 ^

手机号+短信验证码模式扩展

- SmsAuthenticationToken
- SmsAuthenticationConverter
- [SmsAuthProvider](#)

相关推荐

-  Spring Authorization Server (5) 授权服务器【用户、客户端信息】扩展
497阅读 · 1点赞
-  Spring Authorization Server (8) 授权服务的默认认证方式扩展
488阅读 · 2点赞
-  Spring Authorization Server (7) 第三方平台账号存储
296阅读 · 0点赞
- Spring Authorization Server 密码模式
2.5k阅读 · 4点赞



为你推荐



Spring Authorization Server (3) so seasy 集成第三方【gitee、github】oauth2登录

爱吃西瓜的胖娃 11月前 1.1k 3 3

Spring ... Spring ...

Spring Authorization Server 授权服务器

程序员Mark 1年前 6.5k 25 9

Spring ...

Spring Authorization Server 全新授权服务器整合使用

冷冷zz 3年前 3.3k 16 3

Java Spring B...

Spring Authorization Server Password授权扩展

hundanli 5月前 386 1 2

Java Spring B...

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

叹雪飞花 1年前 9.4k 35 107

Java

Spring Authorization Server (9) 授权服务的授权信息存储方式扩展

爱吃西瓜的胖娃 9月前 877 3 3

Spring ... Spring ...

Spring Authorization Server的使用

huan1993 3年前 7.5k 21 6

Spring 后端

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 1年前 6.5k 28 74

后端 Spring ... Spring

Linkerd Service Mesh 授权策略(Server & ServerAuthorization)

为少 2年前 2.1k 4 1

云原生 Service... 后端

Spring 官方发起Spring Authorization Server 项目

码农小胖哥 4年前 4.6k 7 11

Java Spring B...

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 1年前 2.0k 14 10

Spring ... Spring Vue.js

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 11月前 1.2k 9 18

Vue.js 安全 Spring ...

鸭鸭笔记-Spring Authorization Server

Spring Authorization Server + Oauth2 配置认证服务器与资源服务器

张蕊是胖胖

1年前

👁 3.3k

👍 11

💬 6

后端