

授权码模式

申请授权码:

http://localhost:8080/oauth/authorize?response_type=code&client_id=xwn&redirect_url=http://localhost:8080&scope=all

response_type:code 授权类型为授权码模式

client_id:xwn 客户端id

redirect_url=<http://localhost:8080> 重定向地址

scope:all 授权范围

拿到授权码:D7IY3W

<http://localhost:8080/?code=D7IY3W>

在访问:<http://localhost:8080/oauth/token>

以form表单提交数据

The screenshot shows a REST client interface with a POST request to `http://localhost:8080/oauth/token`. The 'Body' tab is selected, and the 'x-www-form-urlencoded' radio button is chosen. The form data is as follows:

KEY	VALUE
<input checked="" type="checkbox"/> grant_type	authorization_code
<input checked="" type="checkbox"/> code	D7IY3W
<input checked="" type="checkbox"/> client_id	xwn
<input checked="" type="checkbox"/> redirect_uri	http://localhost:8080
<input checked="" type="checkbox"/> scope	all

Below the form data, the 'Authorization' tab is selected, showing 'Basic Auth' configuration. A warning message states: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment...'. The 'Username' field contains 'xwn' and the 'Password' field contains '123'. The 'Show Password' checkbox is checked.

就可以获取到token,再根据token访问受保护的资源

路径:<http://localhost:8080/user/getUser>

POST http://localhost:8080/user/getUser

Params Authorization Headers (9) Body Pre-request Script Tests Settings

TYPE
Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Token: d4824cf7-b146-4e1e-a87a-bab3e690dc70

密码模式

直接访问:

<http://localhost:8080/oauth/token>

POST http://localhost:8080/oauth/token

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE
<input checked="" type="checkbox"/> grant_type	password
<input checked="" type="checkbox"/> scope	all
<input checked="" type="checkbox"/> username	admin
<input checked="" type="checkbox"/> password	123

TYPE
Basic Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Username: xwn
Password: 123
☒ Show Password

就能获取到token

```
1 {
2   "access_token": "d4824cf7-b146-4e1e-a87a-bab3e690dc70",
3   "token_type": "bearer",
4   "expires_in": 43199,
5   "scope": "all"
6 }
```

刷新令牌

直接访问:

<http://localhost:8080/oauth/token>

TYPE

Basic Auth

Heads up! These parameters hold sensitive data. To keep this data secure while work

Username

xwn

Password

123

☒ Show Password

none

form-data

x-www-form-urlencoded

raw

binary

urlencoded

KEY	VALUE
<input checked="" type="checkbox"/> grant_type	refresh_token
<input checked="" type="checkbox"/> refresh_token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJhZG1pbGlzIn
Key	Value

项目结构



依赖:

```

1 <parent>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-parent</artifactId>
4   <version>2.1.4.RELEASE</version>
5   <relativePath/>
6 </parent>
7
8 <dependencies>
9   <dependency>
10    <groupId>org.springframework.cloud</groupId>
11    <artifactId>spring-cloud-starter-oauth2</artifactId>
12  </dependency>
13  <dependency>
14    <groupId>org.springframework.boot</groupId>
15    <artifactId>spring-boot-starter-web</artifactId>
16  </dependency>
17  <!--redis-->
  
```

```

18     <dependency>
19         <groupId>org.springframework.boot</groupId>
20         <artifactId>spring-boot-starter-data-redis</artifactId>
21     </dependency>
22     <!--redis存储token配置-->
23     <dependency>
24         <groupId>org.apache.commons</groupId>
25         <artifactId>commons-pool2</artifactId>
26     </dependency>
27     <dependency>
28         <groupId>io.jsonwebtoken</groupId>
29         <artifactId>jjwt</artifactId>
30         <version>0.9.1</version>
31     </dependency>
32 </dependencies>
33
34 <dependencyManagement>
35     <dependencies>
36         <dependency>
37             <groupId>org.springframework.cloud</groupId>
38             <artifactId>spring-cloud-dependencies</artifactId>
39             <version>Greenwich.SR1</version>
40             <type>pom</type>
41             <scope>import</scope>
42         </dependency>
43     </dependencies>
44 </dependencyManagement>

```

授权服务器配置

```

1  package com.xwn.config;
2
3  import com.xwn.service.DetailService;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.beans.factory.annotation.Qualifier;
6  import org.springframework.context.annotation.Configuration;
7  import org.springframework.security.authentication.AuthenticationManager;
8  import org.springframework.security.core.userdetails.UserDetails;
9  import org.springframework.security.crypto.password.PasswordEncoder;
10 import
    org.springframework.security.oauth2.config.annotation.configurers.ClientDetail
    ilsServiceConfigurer;
11 import
    org.springframework.security.oauth2.config.annotation.web.configuration.Auth
    orizationServerConfigurerAdapter;
12 import
    org.springframework.security.oauth2.config.annotation.web.configuration.Enab
    leAuthorizationServer;
13 import
    org.springframework.security.oauth2.config.annotation.web.configurers.Author
    izationServerEndpointsConfigurer;
14 import org.springframework.security.oauth2.provider.token.TokenStore;
15 import
    org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConve
    rter;

```

```
16 import
17 org.springframework.security.oauth2.provider.token.store.JwtTokenStore;
18 /**
19  * @author xwn
20  * @date 2021/1/7 23:54
21  */
22
23 @Configuration
24 @EnableAuthorizationServer
25 public class AuthorizationConfig extends
26 AuthorizationServerConfigurerAdapter {
27
28     @Autowired
29     private PasswordEncoder passwordEncoder;
30
31     @Autowired
32     private AuthenticationManager authenticationManager;
33
34     @Autowired
35     private DetailService detailService;
36
37     /*@Autowired
38     private TokenStore RedisTokenStore;*/
39
40     @Autowired
41     private TokenStore tokenStore;
42
43     @Autowired
44     private JwtAccessTokenConverter jwtAccessTokenConverter;
45
46     /*
47      * 密码模式需要的!*/
48     @Override
49     public void configure(AuthorizationServerEndpointsConfigurer endpoints)
50     throws Exception {
51         endpoints.authenticationManager(authenticationManager)
52             .userDetailsService(detailService)
53             //把获取到的accessToken装成JWTtoken
54             .tokenStore(tokenStore)
55             .accessTokenConverter(jwtAccessTokenConverter);
56         //redis存储token
57         //.tokenStore(RedisTokenStore);
58     }
59
60     @Override
61     public void configure(ClientDetailsServiceConfigurer clients) throws
62     Exception {
63         clients.inMemory()
64             //客户端ID
65             .withClient("xwn")
66             //密钥
67             .secret(passwordEncoder.encode("123"))
68             //重定向地址
69             .redirectUri("http://localhost:8080")
70             //令牌过期时间
```

```

70         .accessTokenValiditySeconds(60)
71         //刷新令牌过期时间
72         .refreshTokenValiditySeconds(86400)
73         //授权范围
74         .scopes("all")
75         //授权类型(authorization_code(授权码模式),password(密码模式))
76         .authorizedGrantTypes("authorization_code",
77         "password","refresh_token");
78     }
79 }

```

资源服务器配置

```

1  package com.xwn.config;
2
3  import org.springframework.context.annotation.Configuration;
4  import
5  org.springframework.security.config.annotation.web.builders.HttpSecurity;
6  import
7  org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;
8  import
9  org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter;
10
11 /**
12  * @author xwn
13  * @date 2021/1/7 23:59
14  */
15 @Configuration
16 @EnableResourceServer
17 public class ResourceConfig extends ResourceServerConfigurerAdapter {
18
19     @Override
20     public void configure(HttpSecurity http) throws Exception {
21         http.authorizeRequests()
22             //所有请求拦截
23             .anyRequest()
24             .authenticated()
25             .and()
26             //请求路径匹配,需要token令牌才能访问
27             .requestMatchers()
28             .antMatchers("/user/**");
29     }
30 }

```

自定义登录逻辑(未连接数据库)

```

1  package com.xwn.service;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.security.core.authority.AuthorityUtils;
5  import org.springframework.security.core.userdetails.User;

```

```

6  import org.springframework.security.core.userdetails.UserDetails;
7  import org.springframework.security.core.userdetails.UserDetailsService;
8  import
org.springframework.security.core.userdetails.UsernameNotFoundException;
9  import org.springframework.security.crypto.password.PasswordEncoder;
10 import org.springframework.stereotype.Service;
11
12 /**
13  * @author xwn
14  * @date 2021/1/7 23:39
15  */
16
17 @Service
18 public class DetailService implements UserDetailsService {
19
20     @Autowired
21     private PasswordEncoder passwordEncoder;
22
23
24     @Override
25     public UserDetails loadUserByUsername(String s) throws
UsernameNotFoundException {
26
27         return new User("admin", passwordEncoder.encode("123"),
28
AuthorityUtils.commaSeparatedStringToAuthorityList("admin"));
29     }
30 }
31

```

security配置

```

1  package com.xwn.config;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Configuration;
5  import org.springframework.security.authentication.AuthenticationManager;
6  import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
7  import
org.springframework.security.config.annotation.web.configuration.EnableWebSe
curity;
8  import
org.springframework.security.config.annotation.web.configuration.WebSecurity
ConfigurerAdapter;
9  import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
10 import org.springframework.security.crypto.password.PasswordEncoder;
11
12 /**
13  * @author xwn
14  * @date 2021/1/7 23:40
15  */
16
17 @Configuration
18 @EnableWebSecurity
19 public class SecurityConfig extends WebSecurityConfigurerAdapter {
20

```

```

21     @Override
22     protected void configure(HttpSecurity http) throws Exception {
23         http.authorizeRequests()
24             .antMatchers("/oauth/**", "/login/**", "/logout/**")
25             .permitAll()
26             .anyRequest().authenticated()
27             .and()
28             .csrf().disable()
29             .formLogin()
30             .permitAll();
31     }
32
33     @Bean
34     public PasswordEncoder passwordEncoder(){
35         return new BCryptPasswordEncoder();
36     }
37
38     //密码模式的配置
39     @Bean
40     public AuthenticationManager authenticationManager() throws Exception {
41         return super.authenticationManager();
42     }
43 }
44
45

```

UserController

```

1  package com.xwn.controller;
2
3  import io.jsonwebtoken.Jwts;
4  import org.springframework.security.core.Authentication;
5  import org.springframework.web.bind.annotation.RequestMapping;
6  import org.springframework.web.bind.annotation.RestController;
7
8  import javax.servlet.http.HttpServletRequest;
9  import java.nio.charset.StandardCharsets;
10
11  /**
12   * @author xwn
13   * @date 2021/1/8 0:01
14   */
15
16
17  @RestController
18  @RequestMapping("/user")
19  public class UserController {
20
21      @RequestMapping("/getUser")
22      public Object getUser(Authentication authentication, HttpServletRequest
request) {
23          String authorization = request.getHeader("Authorization");
24          String token =
authorization.substring(authorization.lastIndexOf("bearer") + 7);
25          return Jwts.parser()
26              .setSigningKey("test_key".getBytes(StandardCharsets.UTF_8))
27              .parseClaimsJws(token)

```



```
28         .getBody();
29     }
30 }
31
```

JwtTokenStoreConfig

```
1  package com.xwn.config;
2
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Configuration;
5  import org.springframework.security.oauth2.provider.token.TokenStore;
6  import
7  org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConve
8  rter;
9  import
10 org.springframework.security.oauth2.provider.token.store.JwtTokenStore;
11
12 /**
13  * @author xwn
14  * @date 2021/1/8 18:25
15  */
16
17 @Configuration
18 public class JwtTokenStoreConfig{
19
20     @Bean
21     public TokenStore tokenStore(){
22         return new JwtTokenStore(jwtAccessTokenConverter());
23     }
24
25     @Bean
26     public JwtAccessTokenConverter jwtAccessTokenConverter(){
27         JwtAccessTokenConverter jwtAccessTokenConverter = new
28         JwtAccessTokenConverter();
29         //jwt的密钥
30         jwtAccessTokenConverter.setSigningKey("test_key");
31         return jwtAccessTokenConverter;
32     }
33 }
34
```