单点登录+第三方登录解决方案

*

修订日期	版本号	描述	修改人

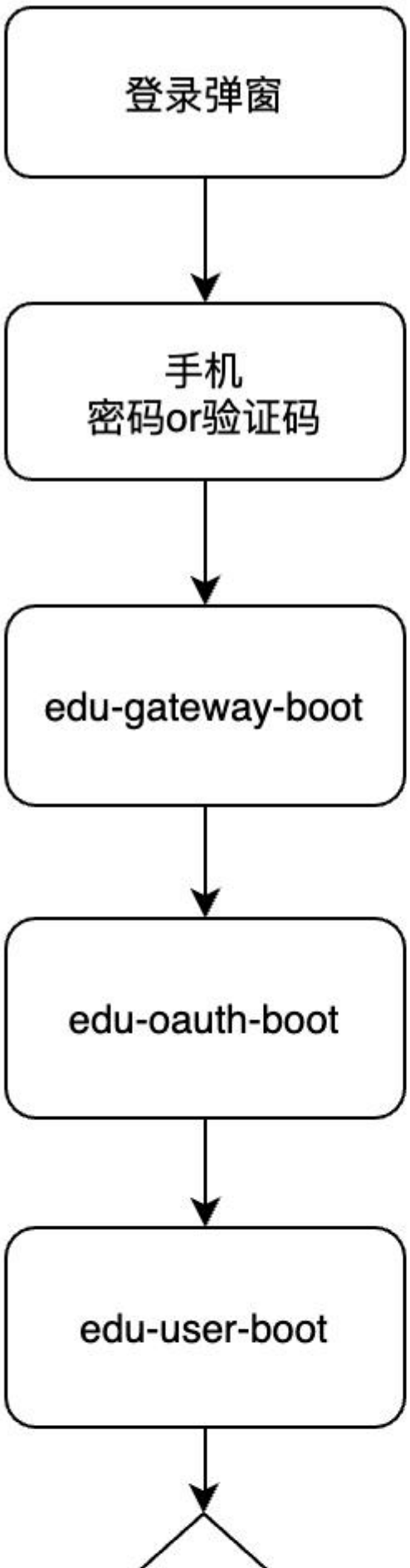
1. 单点登录

a. 支持两种登录方式

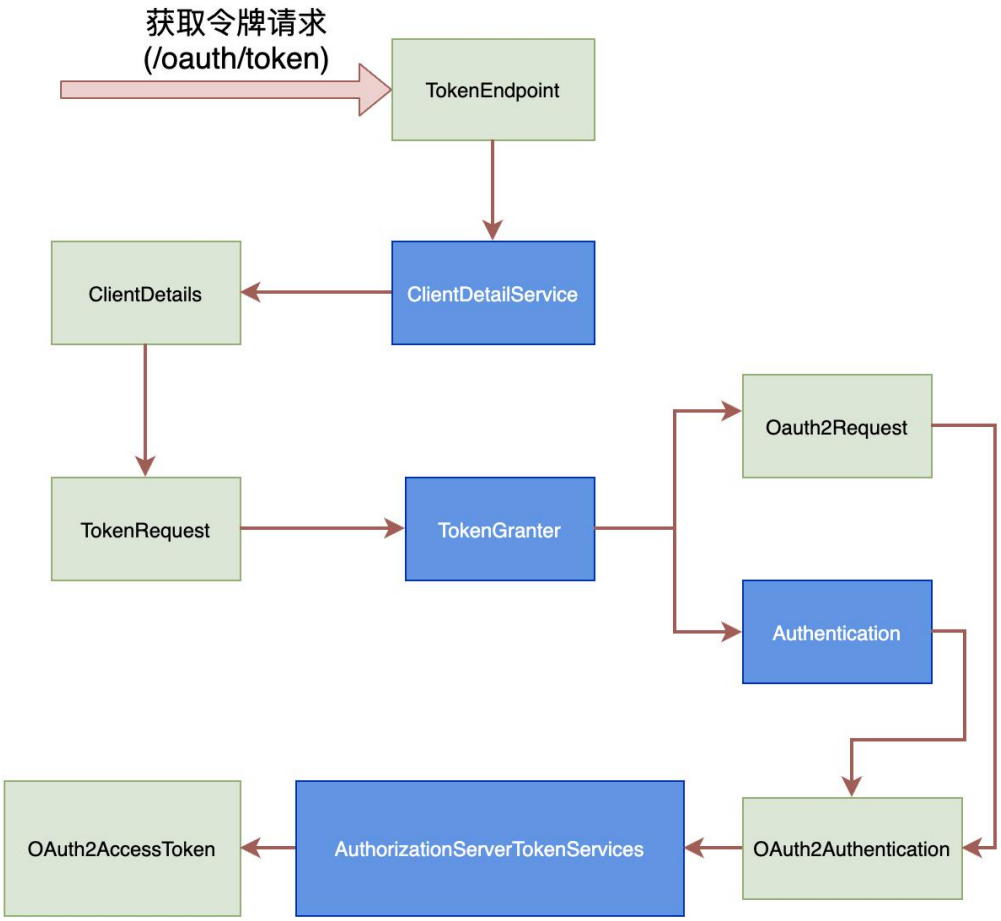
1、手机号+密码登录

2、手机号+验证码登录。整个登录过程在网关层基于 oauth2+jwt 实现的登录过程

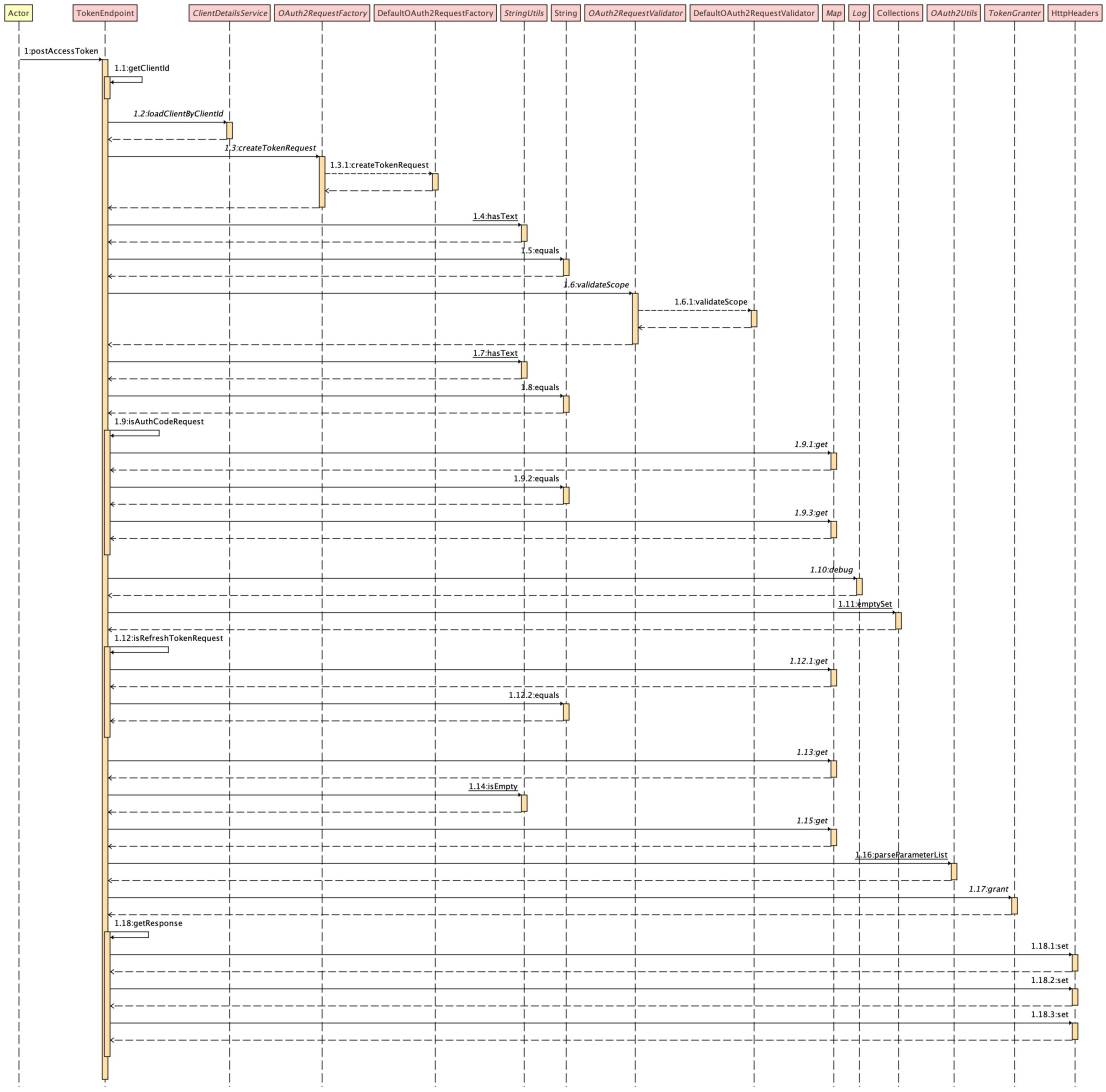
b. 整体流程如下图：


登录弹窗
手机
密码or验证码
edu-gateway-boot
edu-oauth-boot
edu-user-boot

用户输入手机号+密码或者手机号+验证码组合，点击登录，请求到/user/login 接口，开始调用 edu-oauth-boot 模块的接口(/oauth/token)，开始验证用户名、密码是否正确，同时生成 access_token、refresh_token，生成的过程如下图：



1. TokenEndpoint：认证入口，默认是只支持 POST 方法
2. ClientDetailService：JdbcClientDetailsService 实现，客户端的 client_id、client_sercet 存储在数据库中
3. ClientDetails：客户端认证凭据，包含 client_id、client_secret、scope 等信息
4. TokenRequest：根据 client_id、scope、scope、grant_type 生成 TokenRequest 对象
5. 校验 scope 是否正确、grant_type 是否为空，不能是 implicit
6. 最后通过 OAuth2Request、Authentication 生成 OAuth2AccessToken
7. 最后附一张完整调用关系图：



c. 核心配置

i. jwt 密钥配置

```
spring:
  #jwt 的密钥
  security:
    oauth2:
      jwt:
        signingKey: 123456
```

ii. AuthorizationServerConfig

```
@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {

    @Qualifier("dataSource")
    @Autowired
```

```
7.    DataSource dataSource;
8.    @Autowired
9.    @Qualifier("userDetailsService")
10.    UserDetailsService userDetailsService;
11.    @Autowired
12.    @Qualifier("authenticationManagerBean")
13.    private AuthenticationManager authenticationManager;
14.    @Autowired
15.    private IntegrationAuthenticationFilter integrationAuthenticationFilter;
16.    /**
17.     * jwt 对称加密密钥
18.     */
19.    @Value("${spring.security.oauth2.jwt.signingKey}")
20.    private String signingKey;
21.
22.    @Override
23.    public void configure(AuthorizationServerSecurityConfigurer oauthServer) {
24.        // 支持将 client 参数放在 header 或 body 中
25.        oauthServer
26.                .tokenKeyAccess("permitAll()")
27.                .checkTokenAccess("permitAll()")
28.                .allowFormAuthenticationForClients()
29.                .addTokenEndpointAuthenticationFilter(integrationAuthenticationFilter);
30.    }
31.
32.    @Override
33.    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
34.        // 配置客户端信息，从数据库中读取，对应 oauth_client_details 表
35.        clients.jdbc(dataSource);
36.    }
37.
38.    @Override
39.    public void configure(AuthorizationServerEndpointsConfigurer endpoints) {
40.        // 配置 token 的数据源、自定义的 tokenServices 等信息,配置身份认证器，配置认证
方式，TokenStore，TokenGranter，OAuth2RequestFactory
41.        endpoints.tokenStore(tokenStore())
42.                .authorizationCodeServices(authorizationCodeServices())
43.                .approvalStore(approvalStore())
44.                .exceptionTranslator(customExceptionTranslator())
45.                .tokenEnhancer(tokenEnhancerChain())
46.                .authenticationManager(authenticationManager)
47.                .userDetailsService(userDetailsService)
48.                .tokenGranter(tokenGranter(endpoints));
49.    }
50.
51.    /**
52.     * 自定义 OAuth2 异常处理
53.     */
54.    @Bean
55.    public WebResponseExceptionTranslator<OAuth2Exception> customExceptionTranslator()
```

```
{
56.         return new CustomWebResponseExceptionTranslator();
57.     }
58.
59.     /**
60.      * 授权信息持久化实现
61.      *
62.      * @return JdbcApprovalStore
63.      */
64.     @Bean
65.     public ApprovalStore approvalStore() {
66.         return new JdbcApprovalStore(dataSource);
67.     }
68.
69.     /**
70.      * 授权码模式持久化授权码 code
71.      */
72.     @Bean
73.     protected AuthorizationCodeServices authorizationCodeServices() {
74.         // 授权码存储等处理方式类，使用 jdbc，操作 oauth_code 表
75.         return new JdbcAuthorizationCodeServices(dataSource);
76.     }
77.
78.     /**
79.      * token 的持久化
80.      */
81.     @Bean
82.     public TokenStore tokenStore() {
83.         return new JdbcTokenStore(dataSource);
84.     }
85.
86.     /**
87.      * 自定义 token 增强链
88.      */
89.     @Bean
90.     public TokenEnhancerChain tokenEnhancerChain() {
91.         TokenEnhancerChain tokenEnhancerChain = new TokenEnhancerChain();
92.         tokenEnhancerChain.setTokenEnhancers(Arrays.asList(new CustomTokenEnhancer(),
accessTokenConverter()));
93.         return tokenEnhancerChain;
94.     }
95.
96.     /**
97.      * jwt token 的生成配置
98.      */
99.     @Bean
100.     public JwtAccessTokenConverter accessTokenConverter() {
101.         JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
102.         converter.setSigningKey(signingKey);
103.         return converter;
```

```
104.    }
105.
106.    /**
107.     * 配置自定义的 granter
108.     */
109.    public TokenGranter tokenGranter(final AuthorizationServerEndpointsConfigurer
endpoints) {
110.        List<TokenGranter> granters = Lists.newArrayList(endpoints.getTokenGranter());
111.        return new CompositeTokenGranter(granters);
112.    }
113.
114. }
```

iii. WebServerSecurityConfig

```
1. @Configuration
2. @EnableWebSecurity
3. public class WebServerSecurityConfig extends WebSecurityConfigurerAdapter {
4.
5.    @Autowired
6.    @Qualifier("userDetailsService")
7.    private UserDetailsService userDetailsService;
8.
9.    @Override
10.    protected void configure(HttpSecurity http) throws Exception {
11.        http.csrf().disable();
12.        http.httpBasic().disable();
13.        http.authorizeRequests().antMatchers("/actuator/**", "/oauth/token").permitAll()
14.                .anyRequest().authenticated()
15.                .and().logout().permitAll()
16.                .and().formLogin().permitAll()
17.                .and().exceptionHandling().accessDeniedHandler(new
OAuth2AccessDeniedHandler());
18.    }
19.
20.    /**
21.     * 注入自定义的 userDetailsService 实现，获取用户信息，设置密码加密方式
22.     */
23.    @Override
24.    protected void configure(AuthenticationManagerBuilder authenticationManagerBuilder)
throws Exception {
25.        authenticationManagerBuilder
26.                .userDetailsService(userDetailsService)
27.                .passwordEncoder(passwordEncoder());
28.    }
29.
30.    /**
31.     * 将 AuthenticationManager 注册为 bean，方便配置 oauth server 的时候使用
32.     */
33.    @Bean
34.    @Override
35.    public AuthenticationManager authenticationManagerBean() throws Exception {
```

```
36.         return super.authenticationManagerBean();
37.     }
38.
39.     @Bean
40.     public PasswordEncoder passwordEncoder() {
41.         return new BCryptPasswordEncoder();
42.     }
43. }
```

iv. 自定义 jwt 携带内容

```
1. /**
2.  * 自定义 token 携带内容
3.  */
4. @Slf4j
5. public class CustomTokenEnhancer implements TokenEnhancer {
6.
7.     @Override
8.     public OAuth2AccessToken enhance(OAuth2AccessToken accessToken,
OAuth2Authentication authentication) {
9.         Map<String, Object> additionalInfo = Maps.newHashMap();
10.         // 自定义 token 内容，加入组织机构信息
11.         additionalInfo.put("organization", authentication.getName());
12.         try {
13.             // 自定义 token 内容，加入 userId
14.             UserJwt details = (UserJwt) authentication.getPrincipal();
15.             if (null != details) {
16.                 additionalInfo.put("user_id", details.getId());
17.             }
18.         } catch (Exception e) {
19.             log.error("user name:{}", authentication.getName());
20.         }
21.         ((DefaultOAuth2AccessToken) accessToken).setAdditionalInformation(additionalInfo);
22.         return accessToken;
23.     }
24. }
```

v. 自定义用户验证用户信息

```
1. @Service("userDetailsService")
2. @Slf4j
3. public class IntegrationUserDetailsService implements UserDetailsService {
4.
5.     private List<IntegrationAuthenticator> authenticators;
6.
7.     @Autowired
8.     private IRoleService roleService;
9.
10.     @Autowired(required = false)
11.     public void setIntegrationAuthenticators(List<IntegrationAuthenticator> authenticators) {
12.         this.authenticators = authenticators;
13.     }
14.
15.     @Override
```

```
16.     public UserJwt loadUserByUsername(String username) throws
UsernameNotFoundException {
17.         IntegrationAuthentication integrationAuthentication =
IntegrationAuthenticationContext.get();
18.         //判断是否是集成登录
19.         if (integrationAuthentication == null) {
20.             integrationAuthentication = new IntegrationAuthentication();
21.         }
22.         integrationAuthentication.setUsername(username);
23.         UserDTO user = this.authenticate(integrationAuthentication);
24.
25.         if (user == null) {
26.             throw new UsernameNotFoundException("用户名或密码错误");
27.         }
28.
29.         return new UserJwt(
30.                 user.getName(),
31.                 user.getPassword(),
32.                 !user.getIsDel(),
33.                 user.getAccountNonExpired(),
34.                 user.getCredentialsNonExpired(),
35.                 user.getAccountNonLocked(),
36.                 this.obtainGrantedAuthorities(user), user.getId().toString());
37.
38.     }
39.
40.     /**
41.      * 获得登录者所有角色的权限集合.
42.      */
43.     protected Set<GrantedAuthority> obtainGrantedAuthorities(UserDTO user) {
44.         try {
45.             Set<Role> roles = roleService.queryUserRolesByUserId(user.getId().toString());
46.             log.info("user:{},roles:{}", user.getName(), roles);
47.             return roles.stream().map(role -> new
SimpleGrantedAuthority(role.getCode())).collect(Collectors.toSet());
48.         } catch (Exception e) {
49.             e.printStackTrace();
50.             HashSet<GrantedAuthority> grantedAuthorities = new HashSet<>();
51.             grantedAuthorities.add(new SimpleGrantedAuthority("NONE"));
52.             return grantedAuthorities;
53.         }
54.     }
55.
56.     private UserDTO authenticate(IntegrationAuthentication integrationAuthentication) {
57.         if (this.authenticators != null) {
58.             for (IntegrationAuthenticator authenticator : authenticators) {
59.                 if (authenticator.support(integrationAuthentication)) {
60.                     return authenticator.authenticate(integrationAuthentication);
61.                 }
62.             }
```

```
        }
        return null;
    }
}
```

vi. 用户退出：失效 token、同时客户端删除存储的 access_token、refresh_token

```
@FrameworkEndpoint
@Api(tags = "登出接口")
public class TokenRevokeEndpoint {

    @Autowired
    @Qualifier("consumerTokenServices")
    private ConsumerTokenServices tokenServices;

    @DeleteMapping("/oauth/token")
    @ApiOperation("退出登录")
    public ResponseDTO<String> deleteAccessToken(@RequestParam("access_token")
String accessToken) {
        tokenServices.revokeToken(accessToken);
        return ResponseDTO.success();
    }

}
```

vii. 验证码登录

```
@Override
public void configure(AuthorizationServerSecurityConfigurer oauthServer) {
    // 支持将 client 参数放在 header 或 body 中
    oauthServer
            .tokenKeyAccess("permitAll()")
            .checkTokenAccess("permitAll()")
            .allowFormAuthenticationForClients()
            .addTokenEndpointAuthenticationFilter(integrationAuthenticationFilter);
}


@Component
public class IntegrationAuthenticationFilter extends GenericFilterBean implements
ApplicationContextAware {

    private static final String AUTH_TYPE_PARAM_NAME = "auth_type";

    private static final String OAUTH_TOKEN_URL = "/oauth/token";

    private Collection<IntegrationAuthenticator> authenticators;

    private ApplicationContext applicationContext;

    private RequestMatcher requestMatcher;

    public IntegrationAuthenticationFilter() {
        this.requestMatcher = new OrRequestMatcher(
```

```
16.             new AntPathRequestMatcher(OAUTH_TOKEN_URL, "GET"),
17.             new AntPathRequestMatcher(OAUTH_TOKEN_URL, "POST")
18.         );
19.     }
20.
21.     @Override
22.     public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,
FilterChain filterChain) throws IOException, ServletException {
23.
24.         HttpServletRequest request = (HttpServletRequest) servletRequest;
25.         HttpServletResponse response = (HttpServletResponse) servletResponse;
26.
27.         if (requestMatcher.matches(request)) {
28.             //设置集成登录信息
29.             IntegrationAuthentication integrationAuthentication = new IntegrationAuthentication();
30.
integrationAuthentication.setAuthType(request.getParameter(AUTH_TYPE_PARAM_NAME));
31.             integrationAuthentication.setAuthParameters(request.getParameterMap());
32.             IntegrationAuthenticationContext.set(integrationAuthentication);
33.             try {
34.                 //预处理
35.                 this.prepare(integrationAuthentication);
36.
37.                 filterChain.doFilter(request, response);
38.
39.                 //后置处理
40.                 this.complete(integrationAuthentication);
41.             } finally {
42.                 IntegrationAuthenticationContext.clear();
43.             }
44.         } else {
45.             filterChain.doFilter(request, response);
46.         }
47.     }
48.
49.     /**
50.      * 进行预处理
51.      *
52.      * @param integrationAuthentication
53.      */
54.     private void prepare(IntegrationAuthentication integrationAuthentication) {
55.
56.         //延迟加载认证器
57.         if (this.authenticators == null) {
58.             synchronized (this) {
59.                 Map<String, IntegrationAuthenticator> integrationAuthenticatorMap =
applicationContext.getBeansOfType(IntegrationAuthenticator.class);
60.                 if (integrationAuthenticatorMap != null) {
61.                     this.authenticators = integrationAuthenticatorMap.values();
62.                 }
```

```
63.             }
64.         }
65.
66.         if (this.authenticators == null) {
67.             this.authenticators = new ArrayList<>();
68.         }
69.
70.         for (IntegrationAuthenticator authenticator : authenticators) {
71.             if (authenticator.support(integrationAuthentication)) {
72.                 authenticator.prepare(integrationAuthentication);
73.             }
74.         }
75.     }
76.
77.     /**
78.      * 后置处理
79.      *
80.      * @param integrationAuthentication
81.      */
82.     private void complete(IntegrationAuthentication integrationAuthentication) {
83.         for (IntegrationAuthenticator authenticator : authenticators) {
84.             if (authenticator.support(integrationAuthentication)) {
85.                 authenticator.complete(integrationAuthentication);
86.             }
87.         }
88.     }
89.
90.     @Override
91.     public void setApplicationContext(ApplicationContext applicationContext) throws
BeansException {
92.         this.applicationContext = applicationContext;
93.     }
94. }
```

在 tokenEndpoint 认证增加 IntegrationAuthenticationFilter 过滤器，核心逻辑如下：
根据请求参数中的 auth_type 构建 IntegrationAuthentication 对象，根据 auth_type 选择不同的 Authenticator 做用户的验证码检验，最终按照上面的流程最终生成 jwt

2. 第三方登录（微信）

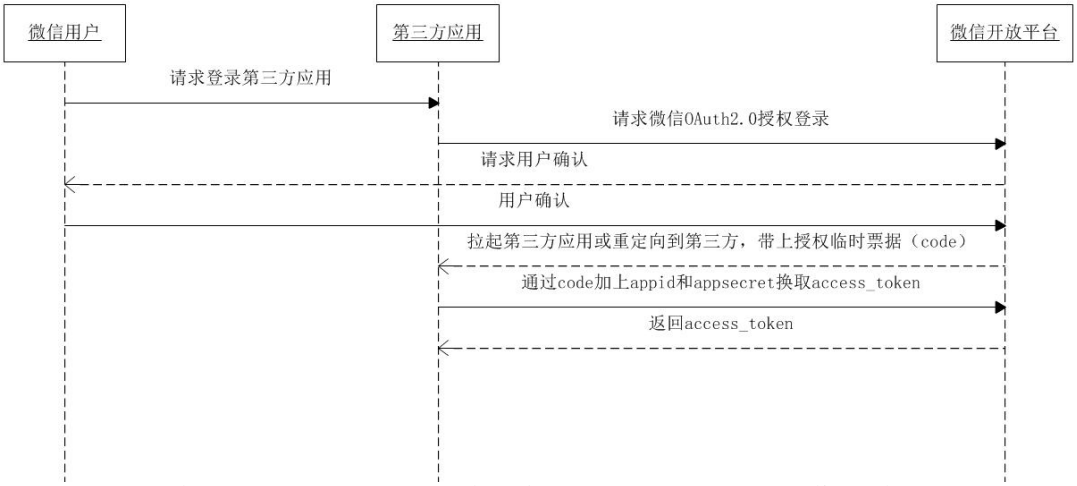
a. 微信扫码登录

微信公众号必须经过微信认证，并且与微信开放平台绑定才能做到网页授权登录

i. 用户点击微信 icon，生成授权链接跳转到微信页面
ii. 用户扫码后，浏览器跳转到回调地址
iii. 回调地址接到请求后，获取参数中的 code，调用微信服务器获取用户个人信息（昵称、openId、头像、城市、性别等信息）
iv. 获取用户信息后，查询该 openId(unionId)是否绑定了拉勾用户，如果没有则提示无法登录

v. 如果该 unionId 绑定了拉勾用户，则快速登录

vi. 微信 OAuth2.0 授权登录过程：



1. 请求 code: 授权登录前请注意已获取相应网页授权作用域（scope=snsapi_login），则可以通过在 PC 端打开以下链接：

1. https://open.weixin.qq.com/connect/qrconnect?appid=APPID&redirect_uri=REDIRECT_URI&response_type=code&scope=SCOPE&state=STATE#wechat_redirect

2. 若提示“该链接无法访问”，请检查参数是否填写错误，如 redirect_uri 的域名与审核时填写的授权域名不一致或 scope 不为 snsapi_login

3. 通过 code 获取 access_token：请求下面接口

1. https://api.weixin.qq.com/sns/oauth2/access_token?appid=APPID&secret=SECRET&code=CODE&grant_type=authorization_code

4. 通过 access_token 调用接口，对于接口作用域（scope），能调用的接口有以下：

授权作用域（scope）	接口	接口说明
snsapi_base	/sns/oauth2/access_token	通过 code 换取 access_token、refresh_token 和已授权 scope
snsapi_base	/sns/oauth2/refresh_token	刷新或续期 access_token 使用
snsapi_base	/sns/auth	检查 access_token 有效性
snsapi_userinfo	/sns/userinfo	获取用户个人信息