Spring Security源码分析十：初识Spring Security OAuth2

2018/01/20 Spring Security

> OAuth 是一个开放标准，允许用户让第三方应用访问该用户在某一网站上存储的私密的资源（如照片，视频，联系人列表），而不需要将用户名和密码提供给第三方应用。OAuth允许用户提供一个令牌，而不是用户名和密码来访问他们存放在特定服务提供者的数据。每一个令牌授权一个特定的网站在特定的时段内访问特定的资源。这样，OAuth让用户可以授权第三方网站访问他们存储在另外服务提供者的某些特定信息。更多`OAuth2`请参考理解OAuth 2.0

项目准备

1. 添加依赖

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.security.oauth</groupId>
            <artifactId>spring-security-oauth2</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>
```

2. 配置认证服务器

```
@Configuration
@EnableAuthorizationServer//是的，没做，就这么一个注解
public class MerryyouAuthorizationServerConfig {

}
```

3. 配置资源服务器

```
@Configuration
@EnableResourceServer//咦，没错还是一个注解
public class MerryyouResourceServerConfig {
}
```

4. 配置`application.yml`客户端信息（不配置的话，控制台会默认打印clientid和clietSecret）

```
security:
  oauth2:
    client:
      client-id: merryyou
      client-secret: merryyou
```

5. 定义 `MyUserDetailsService`

```
@Component
public class MyUserDetailsService implements UserDetailsService {

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return new User(username, "123456", AuthorityUtils.commaSeparatedStringToAuthorityList("ROLE
    }
}
```

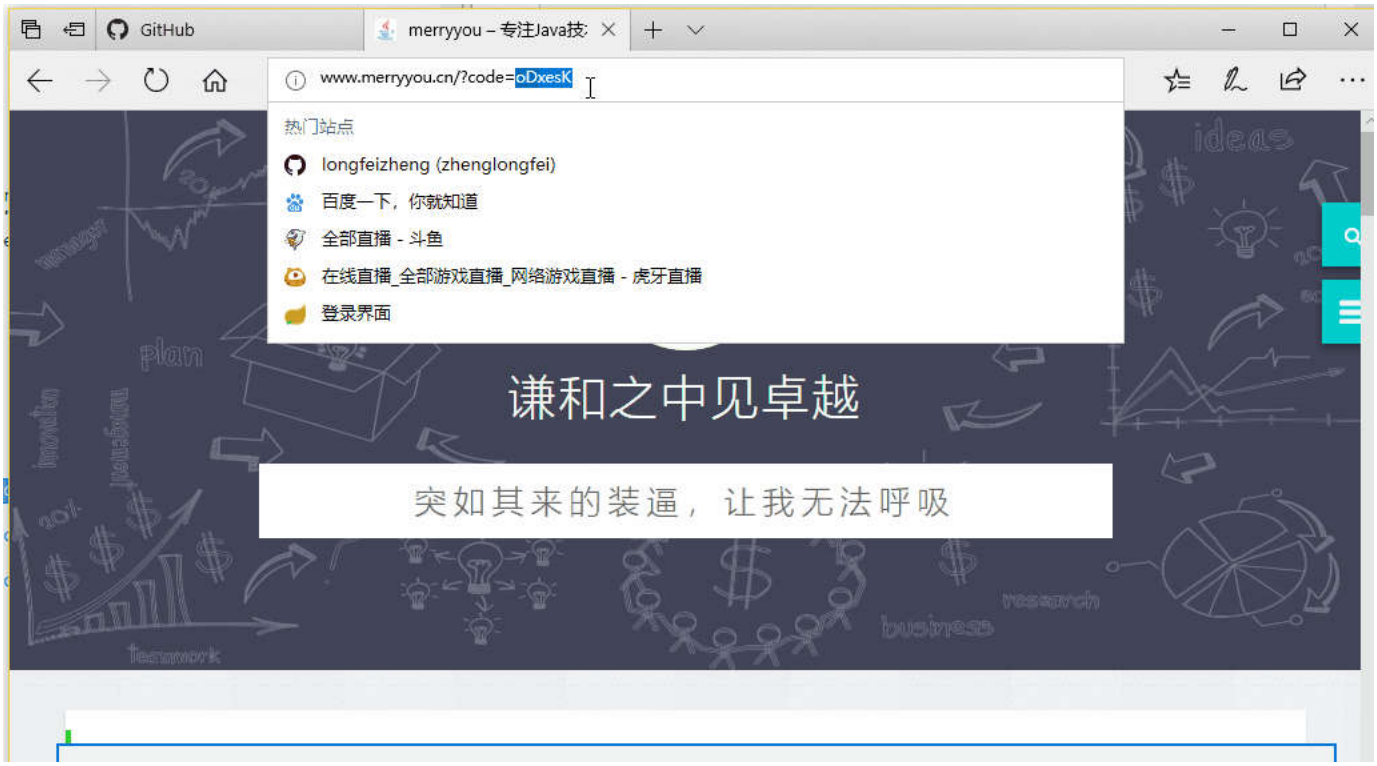
6. 添加测试类 `SecurityOauth2Test` (用户名密码模式)

```
@RunWith(SpringRunner.class)
@SpringBootTest
@Slf4j
public class SecurityOauth2Test {
    //端口
    final static long PORT = 9090;
    //clientId
    final static String CLIENT_ID = "merryyou";
    //clientSecret
    final static String CLIENT_SECRET = "merryyou";
    //用户名
    final static String USERNAME = "admin";
    //密码
    final static String PASSWORD = "123456";
    //获取accessToken得URI
    final static String TOKEN_REQUEST_URI = "http://localhost:"+PORT+"/oauth/token?grant_type=passwo
    //获取用户信息得URL
    final static String USER_INFO_URI = "http://localhost:"+PORT+"/user";

    @Test
    public void getUserInfo() throws Exception{
        RestTemplate rest = new RestTemplate();
        HttpHeaders headers = new HttpHeaders();
        headers.add( "authorization", "Bearer " + getAccessToken() );
        HttpEntity<String> entity = new HttpEntity<String>(null, headers);
        // pay attention, if using get with headers, should use exchange instead of getForEntity / g
        ResponseEntity<String> result = rest.exchange( USER_INFO_URI, HttpMethod.GET, entity, String
        log.info("用户信息返回的结果={}",JsonUtil.toJson(result));
    }

    /**
     * 获取accessToken
     * @return
     */
    private String getAccessToken(){
        RestTemplate rest = new RestTemplate();
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType( MediaType.TEXT_PLAIN );
        headers.add("authorization", getBasicAuthHeader());
        HttpEntity<String> entity = new HttpEntity<String>(null, headers);
        ResponseEntity<OAuth2AccessToken> resp = rest.postForEntity( TOKEN_REQUEST_URI, entity, OAut
        if( !resp.getStatusCode().equals( HttpStatus.OK )){
            throw new RuntimeException( resp.toString() );
        }
        OAuth2AccessToken t = resp.getBody();
        log.info("accessToken={}",JsonUtil.toJson(t));
        log.info("the response, access_token: " + t.getValue() +"; token_type: " + t.getTokenType()
                + "refresh_token: " + t.getRefreshToken() +"; expiration: " + t.getExpiresIn() +", e
        return t.getValue();
```

```
    }

    /**
     * 构建header
     * @return
     */
    private String getBasicAuthHeader(){
        String auth = CLIENT_ID + ":" + CLIENT_SECRET;
        byte[] encodedAuth = Base64.encodeBase64(auth.getBytes());
        String authHeader = "Basic " + new String(encodedAuth);
        return authHeader;
    }
}
```

授权码模式效果如下：

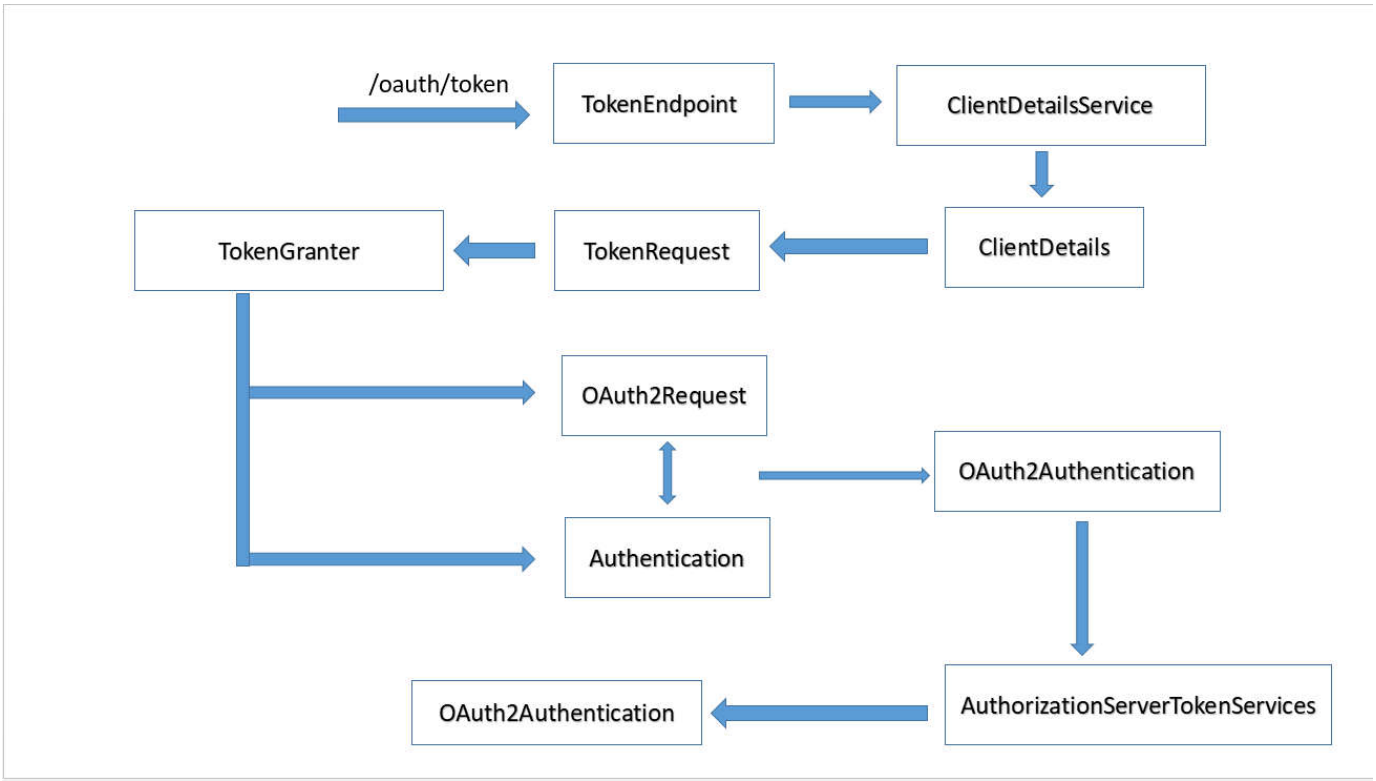
授权链接： http://localhost:9090/oauth/authorize?response_type=code&client_id=merryyou&redirect_uri=http://merryyou.cn&scope=all



测试类打印 `accessToken` 信息

```
2018-01-20 18:16:49.900  INFO 16136 --- [           main] cn.merryyou.security.SecurityOauth2Test  : acces
  "value": "8e5ea72c-d153-48f5-8ee7-9b5616fc43dc",
  "expiration": "Jan 21, 2018 6:10:25 AM",
  "tokenType": "bearer",
  "refreshToken": {
    "value": "7adfefec-c80c-4ff4-913c-4f161c47fbf1"
  },
  "scope": [
    "all"
  ],
  "additionalInformation": {}
}
```

spring security oauth2 登录核心源码

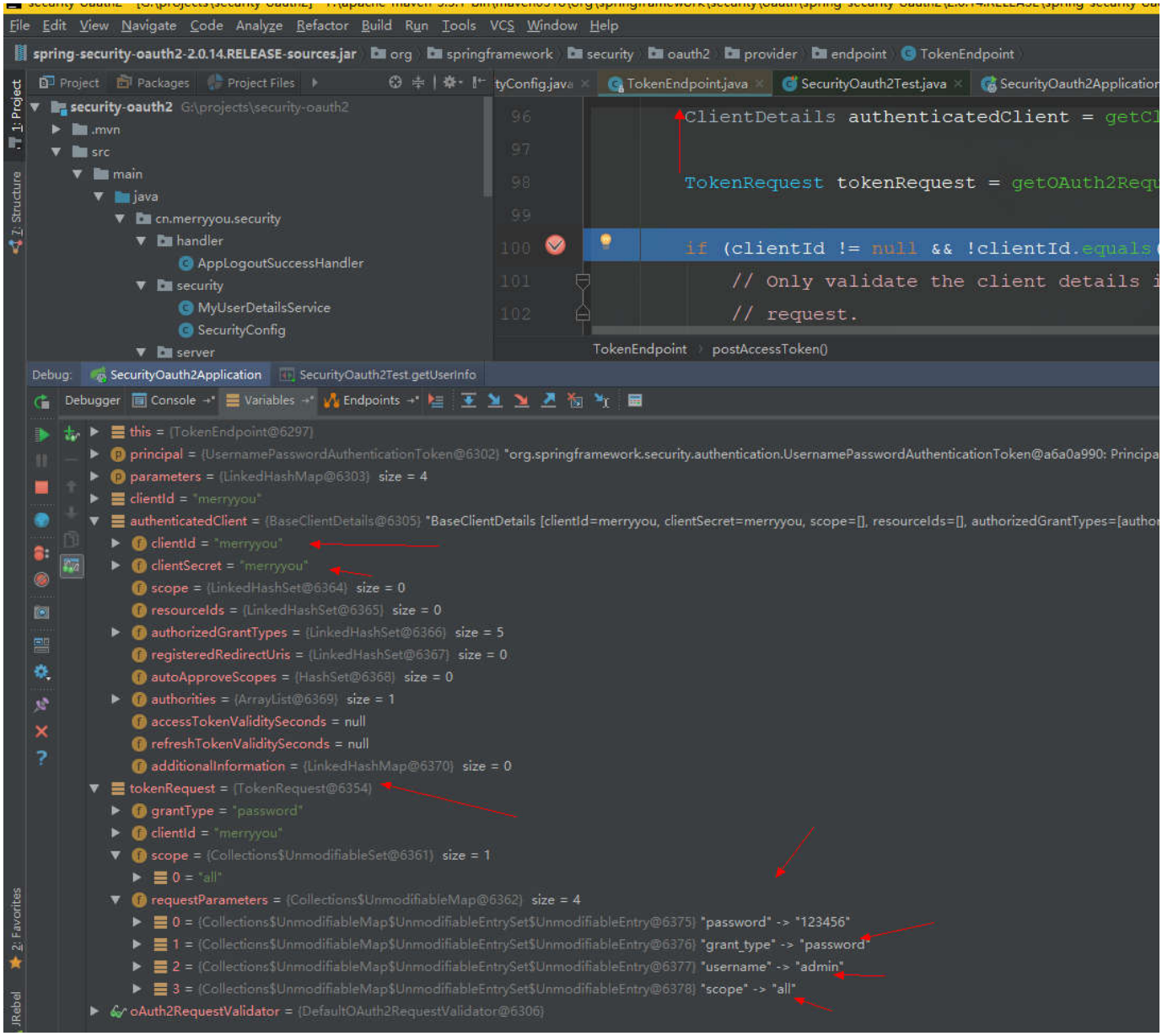


TokenEndpoint

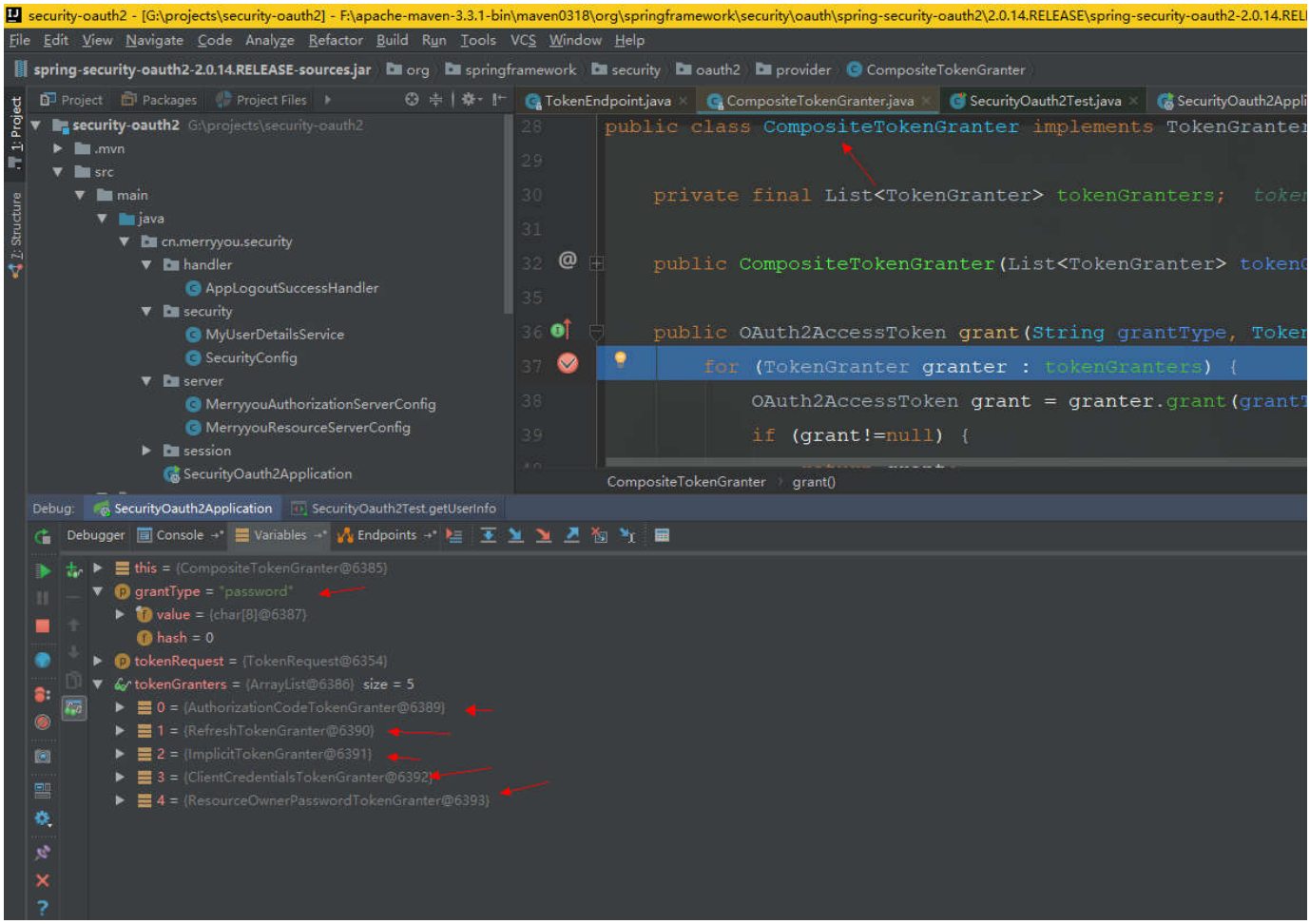
```
//#1.处理/oauth/token请求
@RequestMapping(value = "/oauth/token", method=RequestMethod.POST)
        public ResponseEntity<OAuth2AccessToken> postAccessToken(Principal principal, @RequestParam
        Map<String, String> parameters) throws HttpRequestMethodNotSupportedException {

                if (!(principal instanceof Authentication)) {
                        throw new InsufficientAuthenticationException(
                                        "There is no client authentication. Try adding an appropriate aut
                }
                //获取clientId
                String clientId = getClientId(principal);
                //获取第三方应用的详细配置信息
                ClientDetails authenticatedClient = getClientDetailsService().loadClientByClientId(client
                //使用第三方应用信息创建TokenRequest
                TokenRequest tokenRequest = getOAuth2RequestFactory().createTokenRequest(parameters, auth
                //有没有传clientId
                if (clientId != null && !clientId.equals("")) {
                        // Only validate the client details if a client authenticated during this
                        // request.
                        //与配置里面的是否匹配
                        if (!clientId.equals(tokenRequest.getClientId())) {
                                // double check to make sure that the client ID in the token request is t
                                // authenticated client
                                throw new InvalidClientException("Given client ID does not match authenti
                        }
                }
                if (authenticatedClient != null) {
                        //检查scope
                        oAuth2RequestValidator.validateScope(tokenRequest, authenticatedClient);
                }
                //grant_type是否存在值，对应四种授权模式和刷新token
                if (!StringUtils.hasText(tokenRequest.getGrantType())) {
                        throw new InvalidRequestException("Missing grant type");
```

```
                }
                //是否简化模式
                if (tokenRequest.getGrantType().equals("implicit")) {
                        throw new InvalidGrantException("Implicit grant type not supported from token endp
                }
                //是否是授权码模式
                if (isAuthCodeRequest(parameters)) {
                        // The scope was requested or determined during the authorization step
                        if (!tokenRequest.getScope().isEmpty()) {
                                logger.debug("Clearing scope of incoming token request");
                                //如果是授权码模式scope设置为空，根据获取code时的scope设置
                                tokenRequest.setScope(Collections.<String> emptySet());
                        }
                }
                //是否刷新令牌
                if (isRefreshTokenRequest(parameters)) {
                        // A refresh token has its own default scopes, so we should ignore any added by th
                        //设置scope
                        tokenRequest.setScope(OAuth2Utils.parseParameterList(parameters.get(OAuth2Utils.SC
                }
                //获取OAuth2AccessToken
                OAuth2AccessToken token = getTokenGranter().grant(tokenRequest.getGrantType(), tokenReques
                if (token == null) {
                        throw new UnsupportedGrantTypeException("Unsupported grant type: " + tokenRequest
                }

                return getResponse(token);

        }
```

ClientDetails



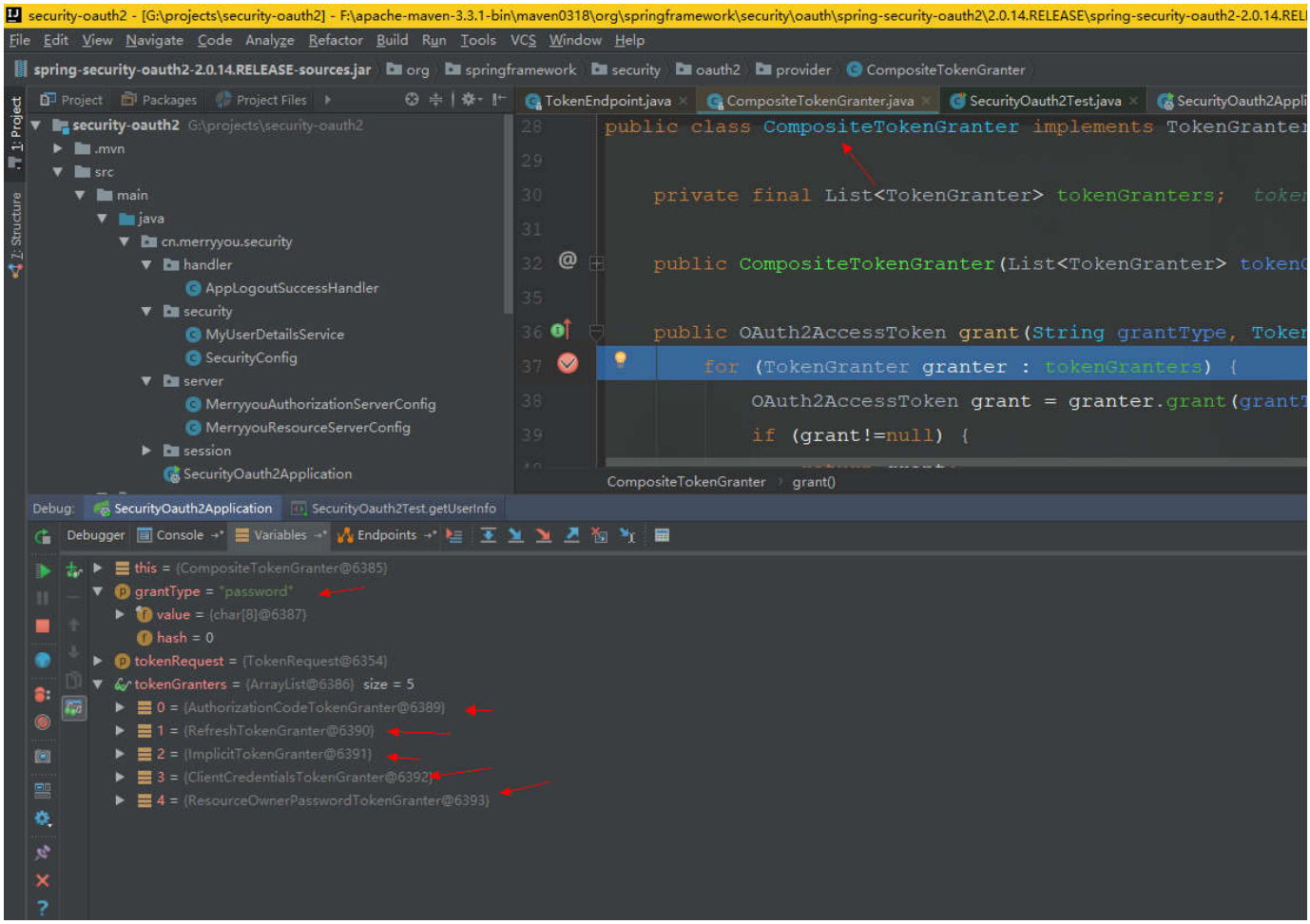
TokenRequest



CompositeTokenGranter#grant

```
    //四种授权模式+刷新令牌的模式根据grant_type判断
    public OAuth2AccessToken grant(String grantType, TokenRequest tokenRequest) {
            for (TokenGranter granter : tokenGranters) {
                    OAuth2AccessToken grant = granter.grant(grantType, tokenRequest);
                    if (grant!=null) {
                            return grant;
                    }
            }
            return null;
    }
```

tokenGranters



AbstractTokenGranter#grant

```
public OAuth2AccessToken grant(String grantType, TokenRequest tokenRequest) {
                //判断当前的授权类型和传入的是否匹配
                if (!this.grantType.equals(grantType)) {
                        return null;
                }
                //获取clientId
                String clientId = tokenRequest.getClientId();
                ClientDetails client = clientDetailsService.loadClientByClientId(clientId);
                //校验
                validateGrantType(grantType, client);

                logger.debug("Getting access token for: " + clientId);
                //产生令牌
                return getAccessToken(client, tokenRequest);

        }
```

AbstractTokenGranter#getAccessToken

```
protected OAuth2AccessToken getAccessToken(ClientDetails client, TokenRequest tokenRequest) {
                return tokenServices.createAccessToken(getOAuth2Authentication(client, tokenRequest));
        }
```

DefaultTokenServices#createAccessToken

```
public OAuth2AccessToken createAccessToken(OAuth2Authentication authentication) throws AuthenticationExce
        //从tokenStore获取OAuth2AccessToken （如果令牌存在，不同的授权模式下将返回同一个令牌）
        OAuth2AccessToken existingAccessToken = tokenStore.getAccessToken(authentication);
        OAuth2RefreshToken refreshToken = null;
        //判断是否过期
        if (existingAccessToken != null) {
                if (existingAccessToken.isExpired()) {
                        if (existingAccessToken.getRefreshToken() != null) {
                                //删除过期的令牌
                                refreshToken = existingAccessToken.getRefreshToken();
                                // The token store could remove the refresh token when the
                                // access token is removed, but we want to
                                // be sure...

                                tokenStore.removeRefreshToken(refreshToken);
                        }
                        tokenStore.removeAccessToken(existingAccessToken);
                }
                else {
                        //如果令牌存在则从新存储一下
                        // Re-store the access token in case the authentication has changed
                        tokenStore.storeAccessToken(existingAccessToken, authentication);
                        //存储完直接返回
                        return existingAccessToken;
                }
        }

        // Only create a new refresh token if there wasn't an existing one
        // associated with an expired access token.
        // Clients might be holding existing refresh tokens, so we re-use it in
        // the case that the old access token
        // expired.
        //判断刷新令牌不存在
        if (refreshToken == null) {
                //创建刷新令牌
                refreshToken = createRefreshToken(authentication);
        }
        // But the refresh token itself might need to be re-issued if it has
        // expired.
        else if (refreshToken instanceof ExpiringOAuth2RefreshToken) {
                //过期
                ExpiringOAuth2RefreshToken expiring = (ExpiringOAuth2RefreshToken) refreshToken;
                if (System.currentTimeMillis() > expiring.getExpiration().getTime()) {
                        refreshToken = createRefreshToken(authentication);
                }
        }
        //根据刷新令牌创建OAuth2AccessToken
        OAuth2AccessToken accessToken = createAccessToken(authentication, refreshToken);
        tokenStore.storeAccessToken(accessToken, authentication);
        // In case it was modified
        refreshToken = accessToken.getRefreshToken();
        if (refreshToken != null) {
                tokenStore.storeRefreshToken(refreshToken, authentication);
        }
        //返回OAuth2AccessToken
        return accessToken;

    }
```

代码下载