7 客户端认证方式 之 client\_secret\_jwt

这篇文章来讲讲 client\_secret\_jwt 方式。

client\_secret\_jwt

client\_secret\_jwt方式就是利用 JWT 进行认证。请求方和授权服务器，两者都知道客户端的 client\_secret，通过相同的 HMAC 算法（对称签名算法）去加签和验签 JWT ，可以达到客户端认证的目的。

请求方 通过 HMAC算法，以 client\_secret 作为密钥，将客户端信息加签生成 JWT ；

授权服务器 使用相同的 HMAC算法和client\_secret，对请求方的 JWT 进行验签以认证客户端。

了解 JWT 的签名算法

示例

请求方传参：

client\_id

client\_assertion\_type：固定值 urn:ietf:params:oauth:client-assertion-type:jwt-bearer

client\_assertion：client生成的jwt

环境准备

授权服务器

同样的，基于 快速搭建一个授权服务器 文章中的示例，修改 SecurityConfiguration 中 registeredClientRepository() 方法，如下：

@Bean

public RegisteredClientRepository registeredClientRepository() {

RegisteredClient registeredClient3 = RegisteredClient.withId(UUID.randomUUID().toString())

.clientId("client3")

// jwt方式验证，密码作为签名算法的密钥，不能配前缀！

.clientSecret("01234567890123456789012345678912")

.clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT\_SECRET\_JWT)

.clientSettings(ClientSettings.builder()

// JWT 方式必须配置，确定jwt的签名算法（CLIENT\_SECRET\_JWT 方式使用 MacAlgorithm（对称加密算法），密钥为client\_secret）

.tokenEndpointAuthenticationSigningAlgorithm(MacAlgorithm.HS256)

.build())

.authorizationGrantType(AuthorizationGrantType.CLIENT\_CREDENTIALS)

.build();

return new InMemoryRegisteredClientRepository(registeredClient3 );

}

测试

生成 JWT

public class ClientJwtTest {

public static void main(String[] args) throws JOSEException {

String clientId = "client3";

String clientSecret = "01234567890123456789012345678912";

// 至少以下四项信息

JWTClaimsSet claimsSet = new JWTClaimsSet.Builder()

// 主体：固定clientId

.subject(clientId)

// 发行者：固定clientId

.issuer(clientId)

// 授权中心的地址

.audience("http://localhost:9000")

// 过期时间

.expirationTime(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 24))

.build();

String jwt = hmacSign(claimsSet, clientSecret);

System.out.println(jwt);

// eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJjbGllbnQxIiwic3ViIjoiY2xpZW50MSIsImF1ZCI6Imh0dHA6XC9cL2xvY2FsaG9zdDo5MDAwIiwiZXhwIjoxNjYxNjk5ODAzfQ.xpv\_8w\_R8LTZFID3uoQPn3CyQK\_Bli3G-WTmrSrwayE

}

/\*\*

\* 使用 HMAC 算法加签生成jwt

\*/

private static String hmacSign(JWTClaimsSet claimsSet, String secret) throws JOSEException {

SecretKeySpec secretKeySpec = new SecretKeySpec(secret.getBytes(StandardCharsets.UTF\_8), "HmacSHA256");

JWSSigner signer = new MACSigner(secretKeySpec);

SignedJWT signedJWT = new SignedJWT(new JWSHeader(JWSAlgorithm.HS256), claimsSet);

signedJWT.sign(signer);

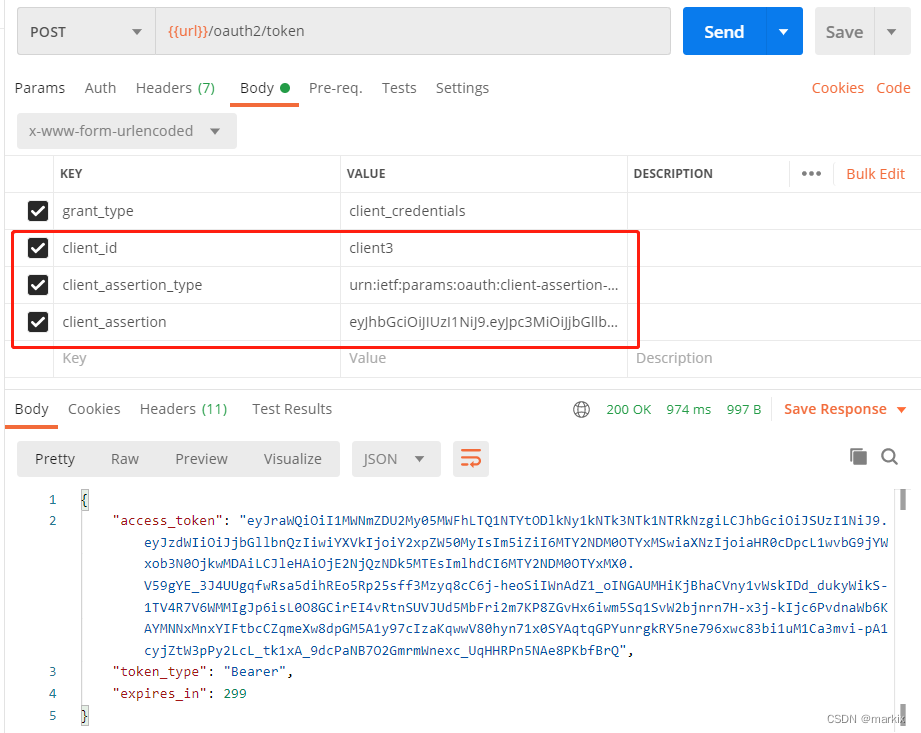
String token = signedJWT.serialize();

return token;

}

}

使用Postman测试，在 Body栏，填入’client\_id、client\_assertion\_type、client\_assertion 和grant\_type’，发送请求。



可以看到，使用此方式能成功获取到 access\_token，说明授权服务器确实支持此认证方式。

相应的curl命令如下：

curl --location --request POST 'localhost:9000/oauth2/token' \

--header 'Content-Type: application/x-www-form-urlencoded' \

--data-urlencode 'grant\_type=client\_credentials' \

--data-urlencode 'client\_id=client3' \

--data-urlencode 'client\_assertion\_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer' \

--data-urlencode 'client\_assertion=eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJjbGllbnQzIiwic3ViIjoiY2xpZW50MyIsImF1ZCI6Imh0dHA6Ly9sb2NhbGhvc3Q6OTAwMCIsImV4cCI6MTY3Mjk5OTgwM30.Cu0JobiFyhyn0mS4rc8pMogI5eOE8uM84v8w5011Gkg'

源码分析

JwtClientAssertionAuthenticationConverter

从请求中解析出 client\_id、client\_assertion\_type、client\_assertion 参数。

JwtClientAssertionAuthenticationProvider

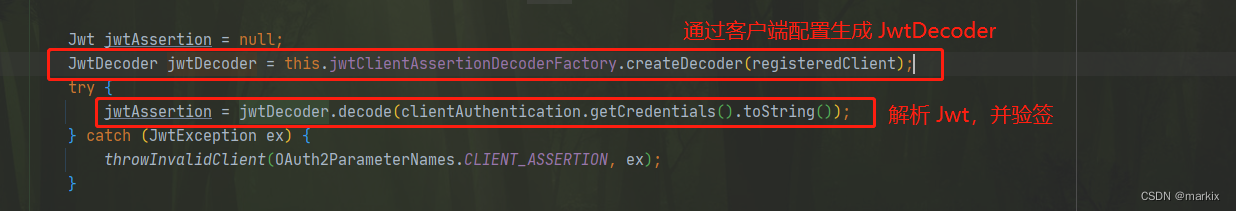
JwtClientAssertionAuthenticationProvider 的核心就是对JWT进行解析和验证。

核心流程如下：

使用请求携带的 clientId 查询客户端信息，若不存在则直接抛出异常。

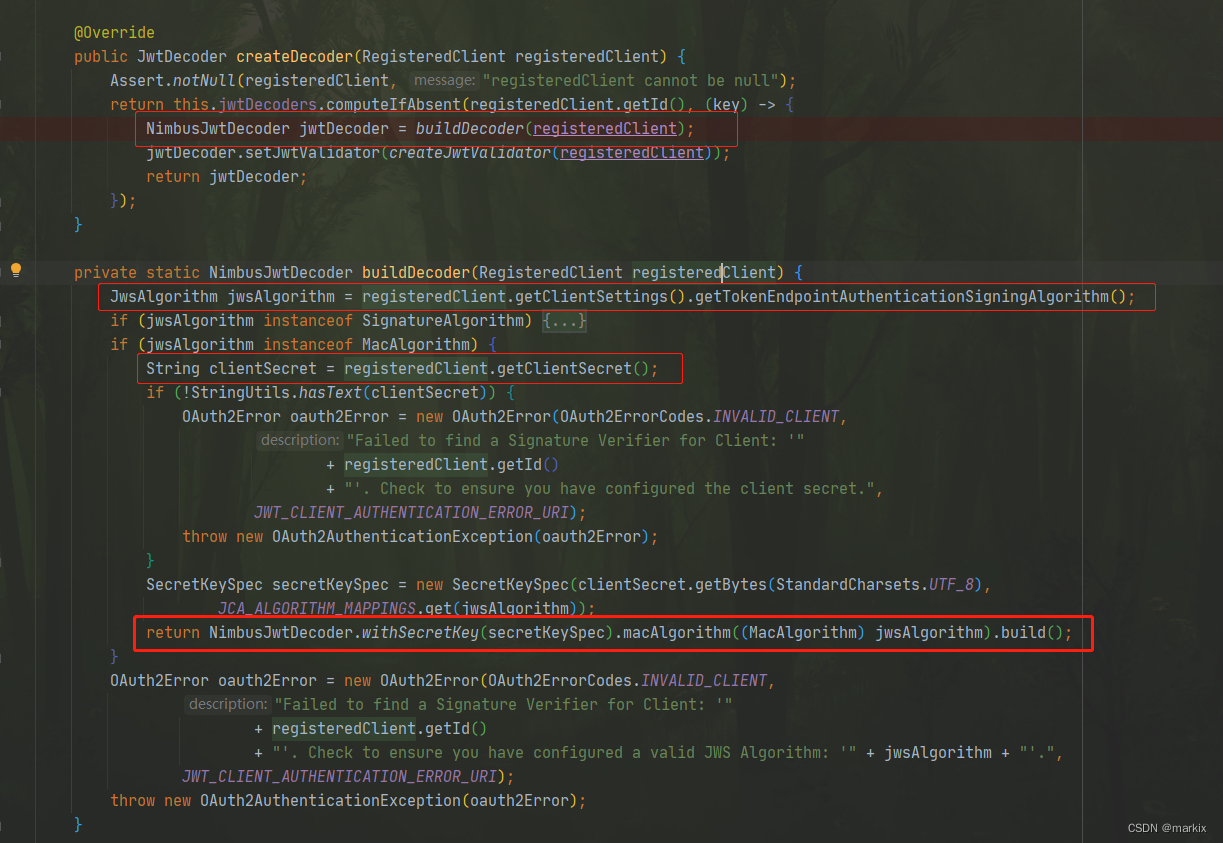
创建解析 JWT 的核心类 JwtDecoder

解析 JWT，并验签



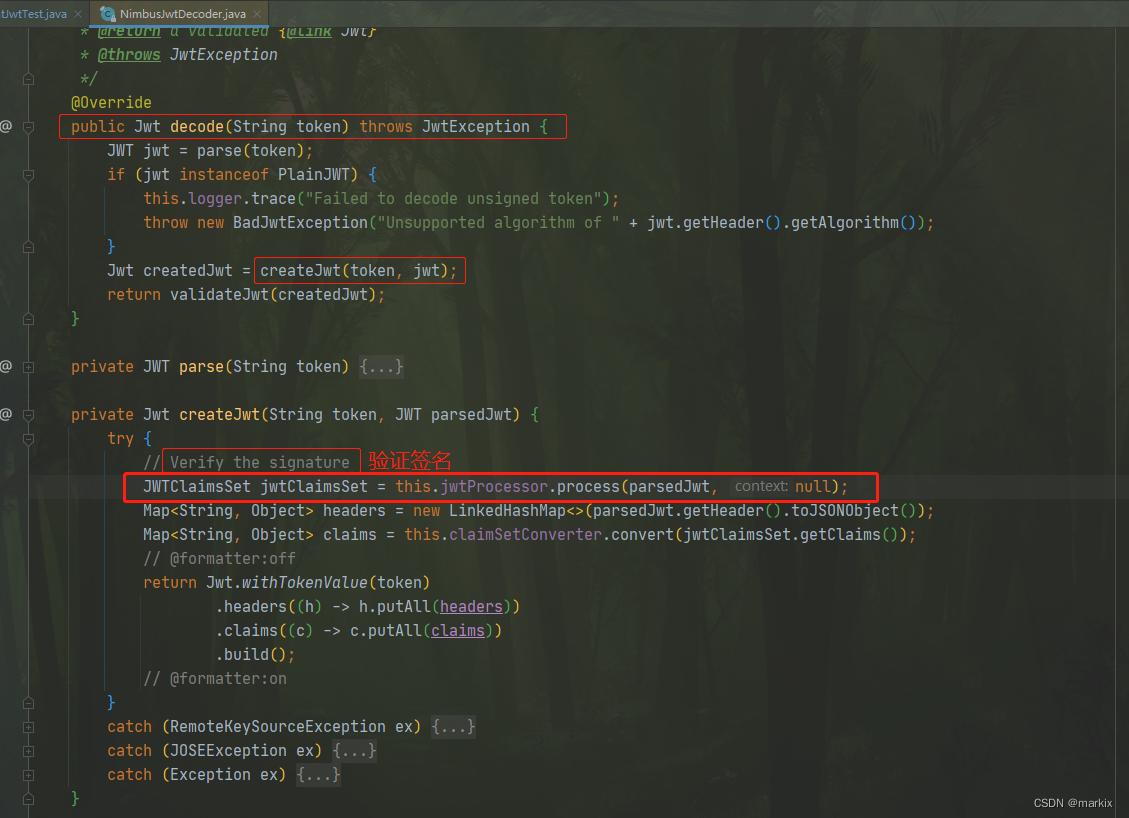
解析：

我们在文章开头就说到 授权服务器 需要使用相同的 HMAC算法和client\_secret，对请求方的 JWT 进行验签以认证客户端。事实也确实如此，一路跟踪创建 JwtDecoder 的代码就会发现，JwtDecoder 的创建过程就会用到客户端的配置（签名算法、密钥）。



创建完 JwtDecoder 之后，就要执行解析和验签了：jwtDecoder.decode(clientAuthentication.getCredentials().toString());

从上面可以看出，jwtDecoder 的实际类型为 NimbusJwtDecoder，其解析JWT和验签逻辑如下：



到这，已经梳理完整个主干流程。另外像比如一些规范性的校验逻辑，读者就可以自行去扩展了解。

下集预告：客户端认证方式 之 private\_key\_jwt，敬请期待。

end