6 客户端认证方式 之 client\_secret\_basic & client\_secret\_post

接上篇文章，上篇文章讲了客户端认证的五种方式，这篇便来深入分析其中的 client\_secret\_basic 方式和 client\_secret\_post 方式。

client\_secret\_basic 和 client\_secret\_post 认证方式都是将客户端的 client\_id 和 client\_secret 传递给授权服务器，授权服务器接收到请求，则根据不同的认证方式从请求中解析出来客户端信息，对 client\_secret 进行验证。

两者的不同在于传参方式不一样，client\_secret\_basic 是将 clientId 和 clientSecret 通过 ‘:’ 号拼接，并使用 Base64 进行编码得到一个字符串。将此编码字符串放到请求头(Authorization) 去发送请求；

而 client\_secret\_post 是将 clientId 和 clientSecret 放到请求体(表单) 去发送请求。

示例

我们将基于 快速搭建一个授权服务器 文章中的示例进行简单修改，以满足我们讲解的目的。

基于上述项目，修改 SecurityConfiguration 中 registeredClientRepository() 方法，如下：

注册一个Client；认证方式支持 client\_secret\_basic 和 client\_secret\_post；授权方式设置为client\_credentials（客户端模式），便于测试。

@Bean

public RegisteredClientRepository registeredClientRepository() {

RegisteredClient registeredClient2 = RegisteredClient.withId(UUID.randomUUID().toString())

.clientId("client2")

.clientSecret("{noop}01234567890123456789012345678912")

// 客户端认证方式

.clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT\_SECRET\_BASIC)

.clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT\_SECRET\_POST)

// 授权方式

.authorizationGrantType(AuthorizationGrantType.CLIENT\_CREDENTIALS)

.build();

return new InMemoryRegisteredClientRepository(registeredClient2);

}

ok，进入正题。

client\_secret\_basic

传参方式是将 clientId 和 clientSecret 通过 ‘:’ 号拼接，并使用 Base64 进行编码得到一串字符。将此编码字符串放到请求头(Authorization)去发送请求。

示例测试

我们将通过客户端模式（授权方式）获取token来测试。

首先要使用clientId 和 clientSecret生成编码字符串，Java代码如下：

public static void main(String[] args) {

String clientId = "client2";

String clientSecret = "01234567890123456789012345678912";

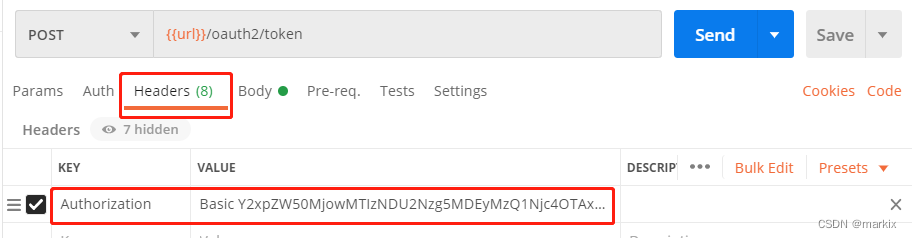
String str = clientId + ":" + clientSecret;

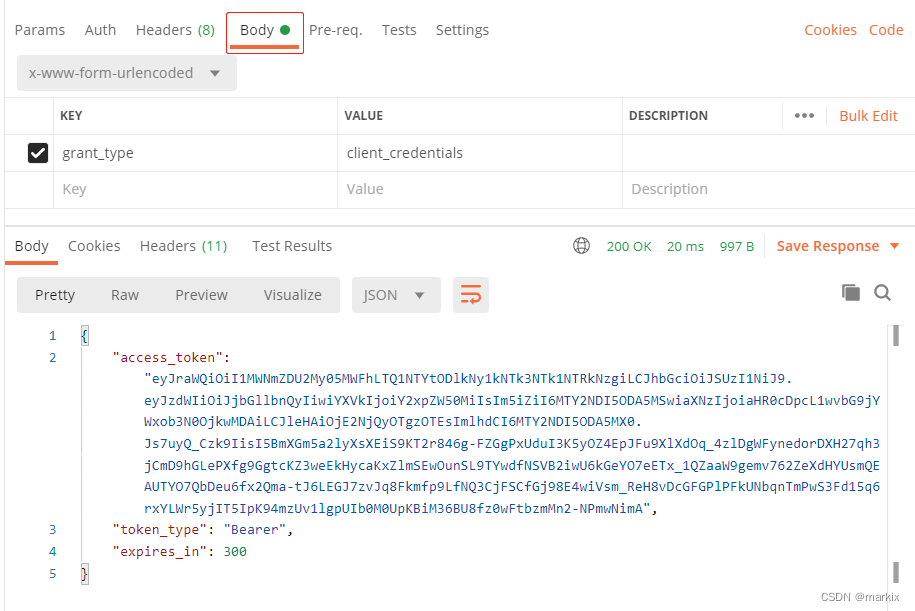
String encode = Base64.getEncoder().encodeToString(str.getBytes());

System.out.println(encode); // Y2xpZW50MjowMTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkxMg==

}

使用Postman测试，在 Header栏，key填入 Authorization，value填入步骤1生成的编码字符串，注意有个 Basic 前缀。（实际上也可以不需要我们自己生成编码字符串，只需要在Auth栏填入clientId 和 clientSecret）；再在Body栏，填入’grant\_type=client\_credentials’，发送请求。





可以看到，使用此方式能成功获取到 access\_token，说明，我们上述的配置是正确的，也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 'Authorization: Basic Y2xpZW50MjowMTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkxMg==' \

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

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

源码分析

我们在上一篇文章已经分析过，client\_secret\_basic 认证方式的核心类其实就是 ClientSecretBasicAuthenticationConverter 和 ClientSecretAuthenticationProvider。

ClientSecretBasicAuthenticationConverter

上文已经说了client信息是将 clientId 和 clientSecret 通过 ‘:’ 号拼接，并使用 Base64 进行编码，再携带在请求头中发送，那么 ClientSecretBasicAuthenticationConverter 自然也是从请求头中获取并执行相反的逻辑（解码、拆分），解析出 clientId 和 clientSecret。（源码就不贴了，逻辑比较简单，可以参照上面的生成编码字符串的Java代码进行理解）

最终生成一个 OAuth2ClientAuthenticationToken 对象，待交由 ClientSecretAuthenticationProvider 处理认证。

ClientSecretAuthenticationProvider

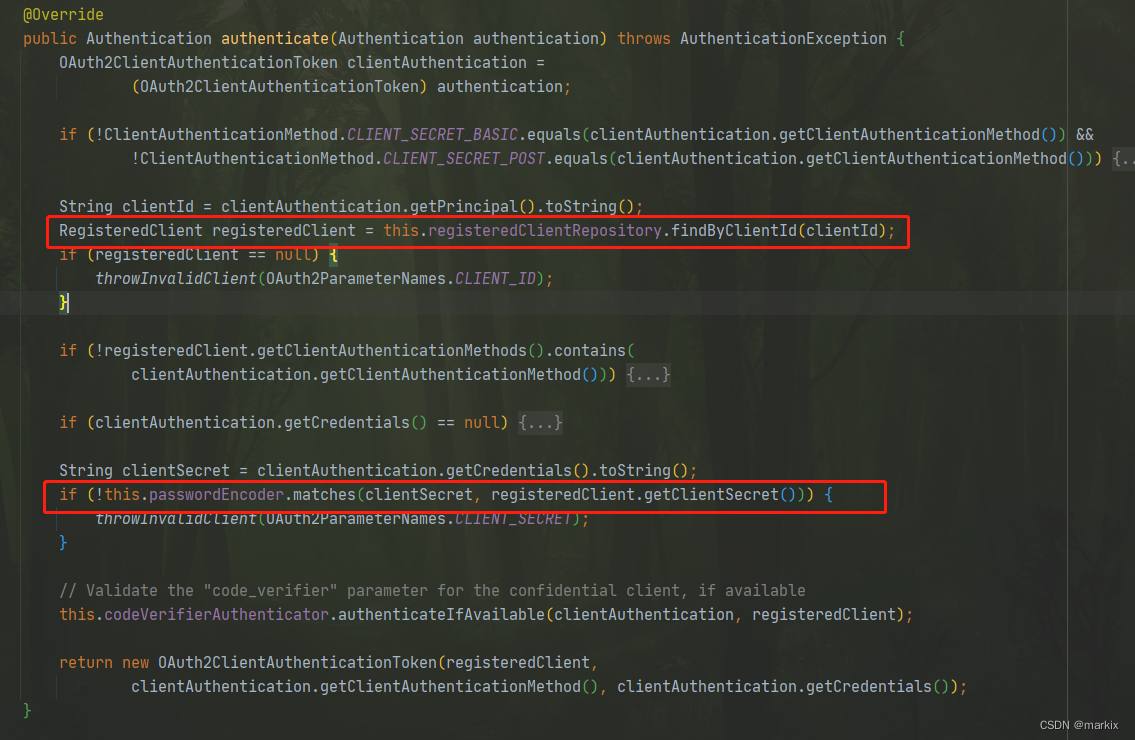
client\_secret\_basic 是通过客户端的密码（clientSecret）进行认证的，所以 ClientSecretAuthenticationProvider 内部引用了 PasswordEncoder， 可以对请求携带的 clientSecret 和 原始的 clientSecret 进行匹配验证。

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。

使用PasswordEncoder 对请求携带的 clientSecret 和 原始的 clientSecret 进行匹配验证。若验证失败则直接抛出异常。

走到return，表示认证通过了。



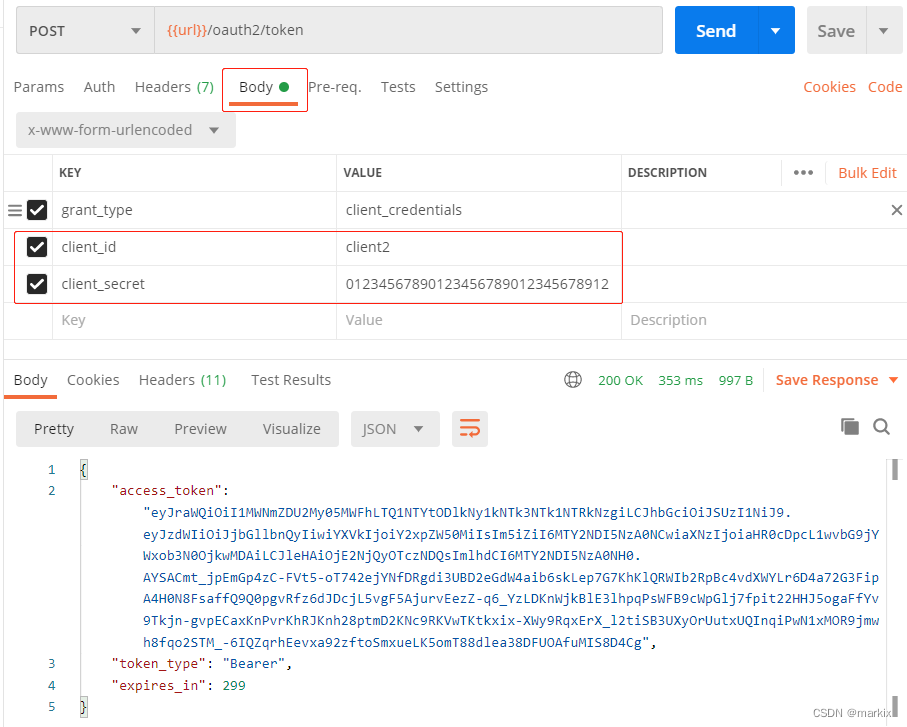
client\_secret\_post

传参方式是将 clientId 和 clientSecret 放到表单去发送请求。

示例测试

同理，我们将通过客户端模式（授权方式）获取token来测试。

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



同理，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 表单 认证方式。

源码分析

我们在上一篇文章已经分析过，client\_secret\_post 认证方式的核心类其实就是 ClientSecretPostAuthenticationConverter 和 ClientSecretAuthenticationProvider。

ClientSecretPostAuthenticationConverter

其实就是从请求表单中获取 clientId 和 clientSecret。

ClientSecretAuthenticationProvider

同 client\_secret\_basic，不赘述了。

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

end