9 客户端认证方式 之 PKCE

开讲之前需要先补充点预备知识：客户端的类型 https://oauth.net/2/client-types

OAuth2 定义了两种类型的客户端：机密客户端（confidential clients）和公共客户端（public clients）。

机密客户端（confidential clients） 指的是能够安全地保存其 client\_secret 的应用程序。例如我们的后台服务，将 client\_secret 保存在配置中，和授权服务器交互都是后台操作，最终给到前端的只是 token，这样client\_secret是安全的。

公共客户端（public clients） 指的是无法安全地保存 client\_secret 的应用程序。例如运行在浏览器的纯前端的应用程序，无论如何，将client\_secret暴露在浏览器，都是不安全的。

ok，铺垫完了，开讲。

none

由于公共客户端没法安全的保存 client\_secret，所以在实际应用中，公共客户端 连 client\_secret 都没必要存了，所以SpringAuthorizationServer定义一个 none 方式来表示这种情况。那这意味着公共客户端就不用认证了吗？答案是否定的！OAuth2 引入了另一个验证的机制 PKCE(Proof Key for Code Exchange)。

PKCE(Proof Key for Code Exchange)

https://oauth.net/2/pkce/

PKCE 是授权码流程的扩展，用于防止 CSRF 和授权码(code)注入攻击。 所以 PKCE 一般都伴随着授权码模式使用，可称之为 增强版授权码流程，又称 Authorization Code with PKCE Flow。

原来的授权码流程 如下：

\1. 客户端发起授权请求 -> 2. 用户授权 -> 3. 客户端拿到code -> 4. 客户端通过code获取token

授权码 + PKCE 流程 对原来流程做了如下增强：

对于客户端而言

步骤1的改造：客户端随机生成一个字符串(称之为明文)，通过摘要算法生成一个密文，发起授权请求时，携带该密文和摘要算法。

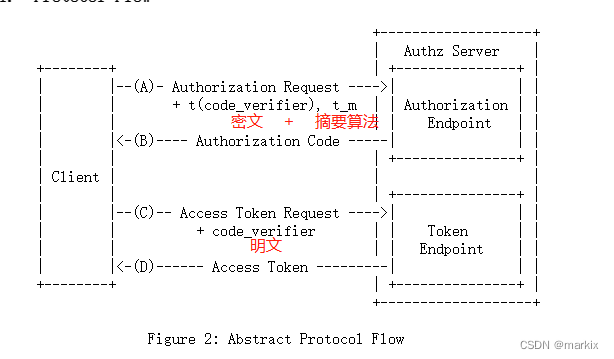
步骤4的改造：客户端通过code获取token时，需要携带上述明文。

对于授权服务器而言

步骤1的改造：授权服务器需要记录客户端此次请求的密文和摘要算法。

步骤4的改造：授权服务器收到客户端的明文，使用同样的摘要算法生成一个密文，和步骤1的密文做对比，如果相同，才算通过验证。

一图胜千言：



整个示例吧，再说可能都要晕了。

示例

步骤1（发起授权）多传的pkce参数：

code\_challenge client生成的密文

code\_challenge\_method：摘要算法，固定值 S256

步骤4（获取Token）多传的pkce参数：

code\_verifier：明文

client\_id：客户端id

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

.clientId("client1")

// 公共客户端，不需要密钥

// .clientSecret("01234567890123456789012345678912")

.clientAuthenticationMethod(ClientAuthenticationMethod.NONE)

.clientSettings(ClientSettings.builder()

// 公共客户端（NONE方式认证）必须开启 PKCE 流程

.requireProofKey(true)

// 授权码模式需要用户手动授权！false表示默认通过

.requireAuthorizationConsent(true)

.build())

.authorizationGrantType(AuthorizationGrantType.AUTHORIZATION\_CODE)

.redirectUri("https://cn.bing.com")

.scope("read")

.build();

return new InMemoryRegisteredClientRepository(registeredClient);

}

测试

客户端生成密文（这里为了便于测试，我直接将明文固定为 xx123，真正在项目用的时候得随机生成才安全！）

public class ClientPkceTest {

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

// 明文

String code\_verifier = "xx123";

// 摘要算法

String code\_challenge\_method = "SHA-256";

// 密文

String code\_challenge = calc(code\_verifier, code\_challenge\_method);

System.out.println(code\_challenge); // 3P6kopgvD5SwlNXyxCW-1DnPGJSNGGYn3H0vos0Xu4o

}

/\*\*

\* 明文 + 摘要算法，生成 密文

\* @see CodeVerifierAuthenticator#codeVerifierValid(java.lang.String, java.lang.String, java.lang.String)

\*/

private static String calc(String code\_verifier, String code\_challenge\_method) throws Exception {

byte[] bytes = code\_verifier.getBytes(StandardCharsets.US\_ASCII);

MessageDigest md = MessageDigest.getInstance(code\_challenge\_method);

byte[] digest = md.digest(bytes);

String code\_challenge = Base64.getUrlEncoder().withoutPadding().encodeToString(digest);

return code\_challenge;

}

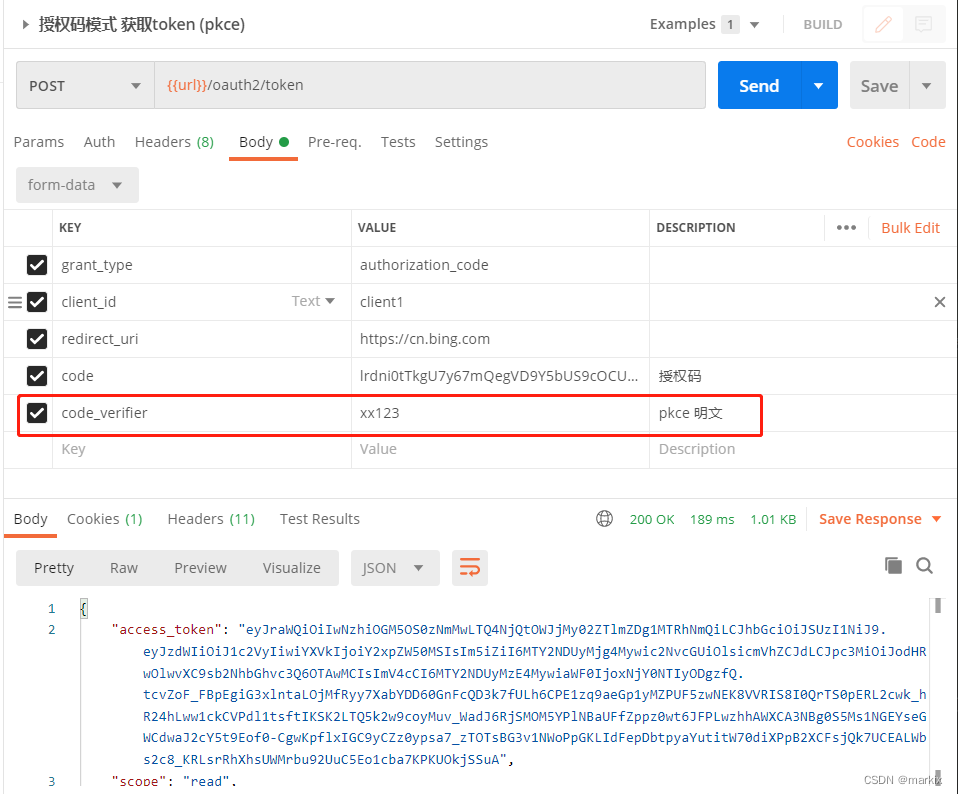
}

发起授权请求，浏览器访问如下地址（相比于原生请求，pkce多加了 code\_challenge、code\_challenge\_method 参数）

http://localhost:9000/oauth2/authorize?response\_type=code&client\_id=client1&redirect\_uri=https%3A%2F%2Fcn.bing.com&scope=read&code\_challenge=3P6kopgvD5SwlNXyxCW-1DnPGJSNGGYn3H0vos0Xu4o&code\_challenge\_method=S256

用户授权后，我们可以得到code

发起code获取token请求（相比于原生请求，pkce多加了 code\_verifier 参数）



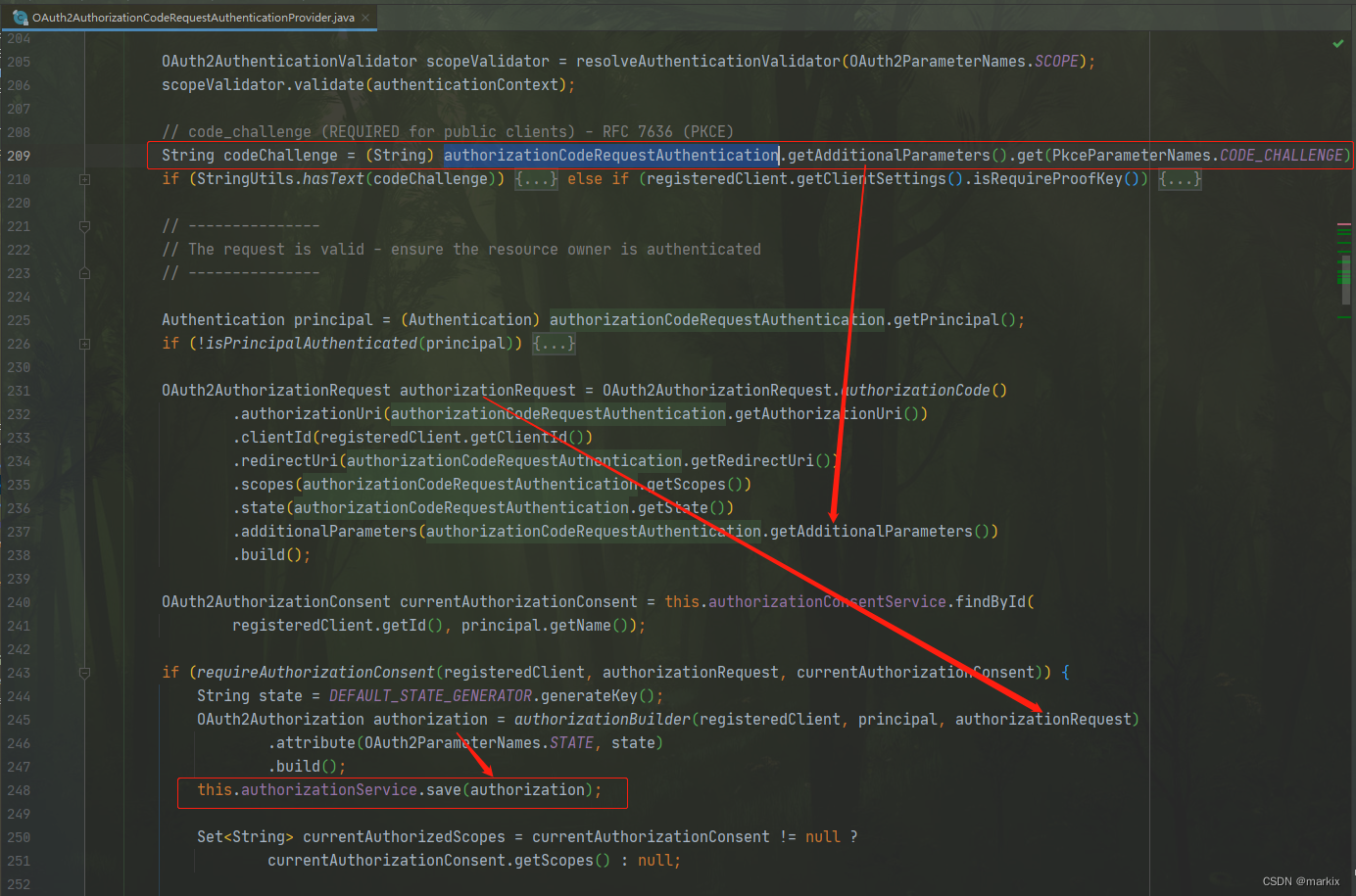
综上，便是 授权码+PKCE 的流程。

源码分析

这里涉及到有两处代码，一个授权请求是需要记录 密文和摘要算法，一个是获取token请求，计算明文和原来密文做验证。

授权请求的处理类 OAuth2AuthorizationCodeRequestAuthenticationProvider

虽然经过层层包装，但最终的效果就是 code\_challenge、code\_challenge\_method 参数会被记录起来，并和 授权码(code)关联（后续通过code才能取回密文和摘要算法做验证）。暂时不用深究其他细节。



获取Token请求

获取token请求，才是真正做校验的地方。会涉及如下关键类：

PublicClientAuthenticationConverter

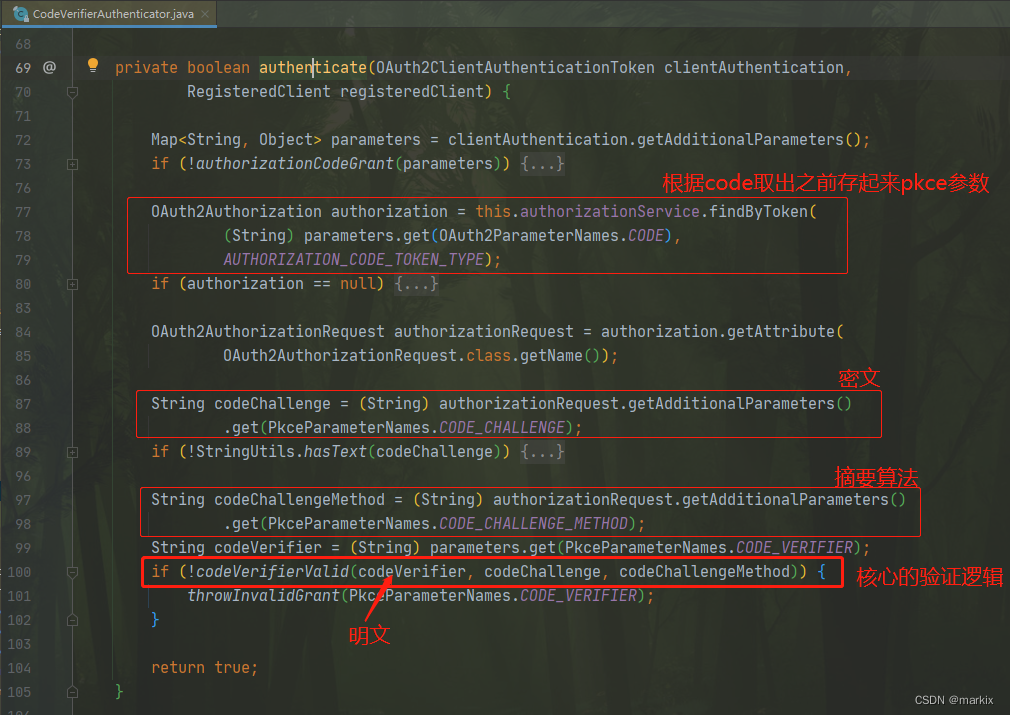
校验 code\_verifier 参数是否存在，若不存在则报错。

PublicClientAuthenticationProvider

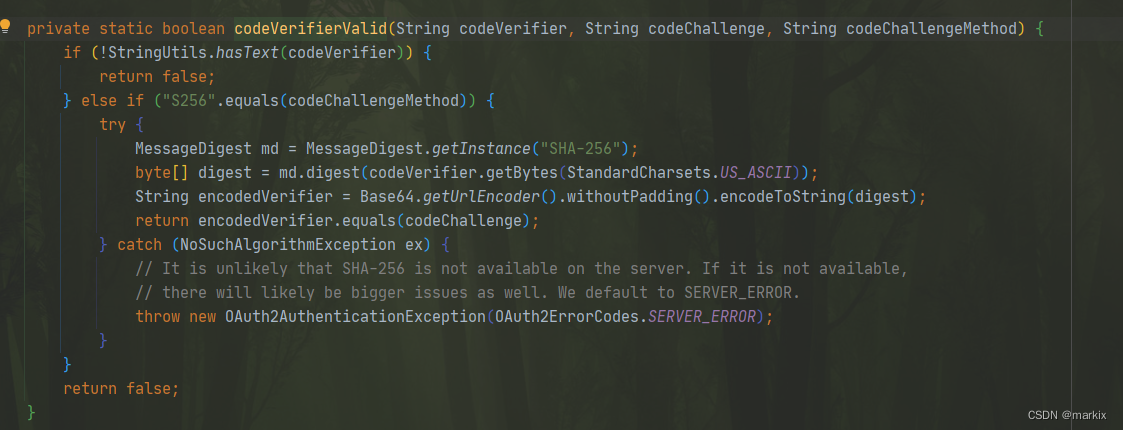
核心逻辑就是：取出授权请求保存下来的code\_challenge（密文）、code\_challenge\_method（摘要算法），和 code\_verifier（明文）做校验。其入口代码如下：



委托给 CodeVerifierAuthenticator 处理



核心验证逻辑就是用相同的摘要算法计算明文，生成一个密文和 原来的密文对比。



以上，便是 pkce 整个流程。其他细节，读者可自行扩展了解啦啦啦~

end