



▲

赞同 2

🚩

分享

授权码 + PKCE 模式 | OIDC & OAuth2.0 认证协议最佳实践系列【03】



Authing 身份云
Authing 就是身份云

关注

2 人赞同了该文章

> 点击上方，关注我们 <



在上一篇文章中，我们介绍了 [OIDC 授权码模式](#)（点击下方链接查看），本次我们将重点围绕 **授权码 + PKCE 模式 (Authorization Code With PKCE)** 进行介绍，从而让你的系统快速具备接入用户认证的标准体系。

[OIDC & OAuth2.0 认证协议最佳实践系列 02 - 授权码模式 \(Authorization Code\) 接入 Authing](#)



为什么会有 PKCE 模式：

[PKCE](#) 是 Proof Key for Code Exchange 的缩写，PKCE 是一种用于增强授权码模式安全性的方法，它可以防止恶意应用程序通过截获授权码和重定向 URI 来获得访问令牌。PKCE 通过将随机字符串（code_verifier）和其 SHA-256 哈希值（code_challenge）与授权请求一起发送，确保访问令牌只能由具有相应 code_verifier 的应用程序使用，保障用户的安全性。

【OAuth 2.0 协议扩展】PKCE 扩展协议：为了解决公开客户端的授权安全问题

「面向对象」public 客户端，其本身没有能力保存密钥信息（恶意攻击者可以通过反编译等手段查看到客户端的密钥 client_secret，也就可以通过授权码 code 换取 access_token，到这一步，恶意应用就可以拿着 token 请求资源服务器了）

「原理」PKCE 协议本身是对 [OAuth 2.0](#) 的扩展，它和之前的授权码流程大体上是一致的，区别在于在向授权服务器的 authorize endpoint 请求时，需要额外的 code_challenge 和 code_challenge_method 参数；向 token endpoint 请求时，需要额外的 code_verifier 参数。最后授权服务器会对这三个参数进行对比验证，通过后颁发令牌

知乎

如果你的应用是一个 SPA 前端应用或移动端 App，建议使用授权码 + PKCE 模式来完成用户的认证和授权。授权码 + PKCE 模式适合不能安全存储密钥的场景（例如前端浏览器）

我们解释下 code_verifier 和 code_challenge



对于每一个 OAuth/OIDC 请求，客户端会先创建一个代码验证器 code_verifier

code_verifier: 在 [A-Z] / [a-z] / [0-9] / "-" / "." / "_" / "~" 范围内，生成 43-128 位的随机字符串。

code_challenge: 则是对 code_verifier 通过 code_challenge_method 例如 sha256 转换得来的。

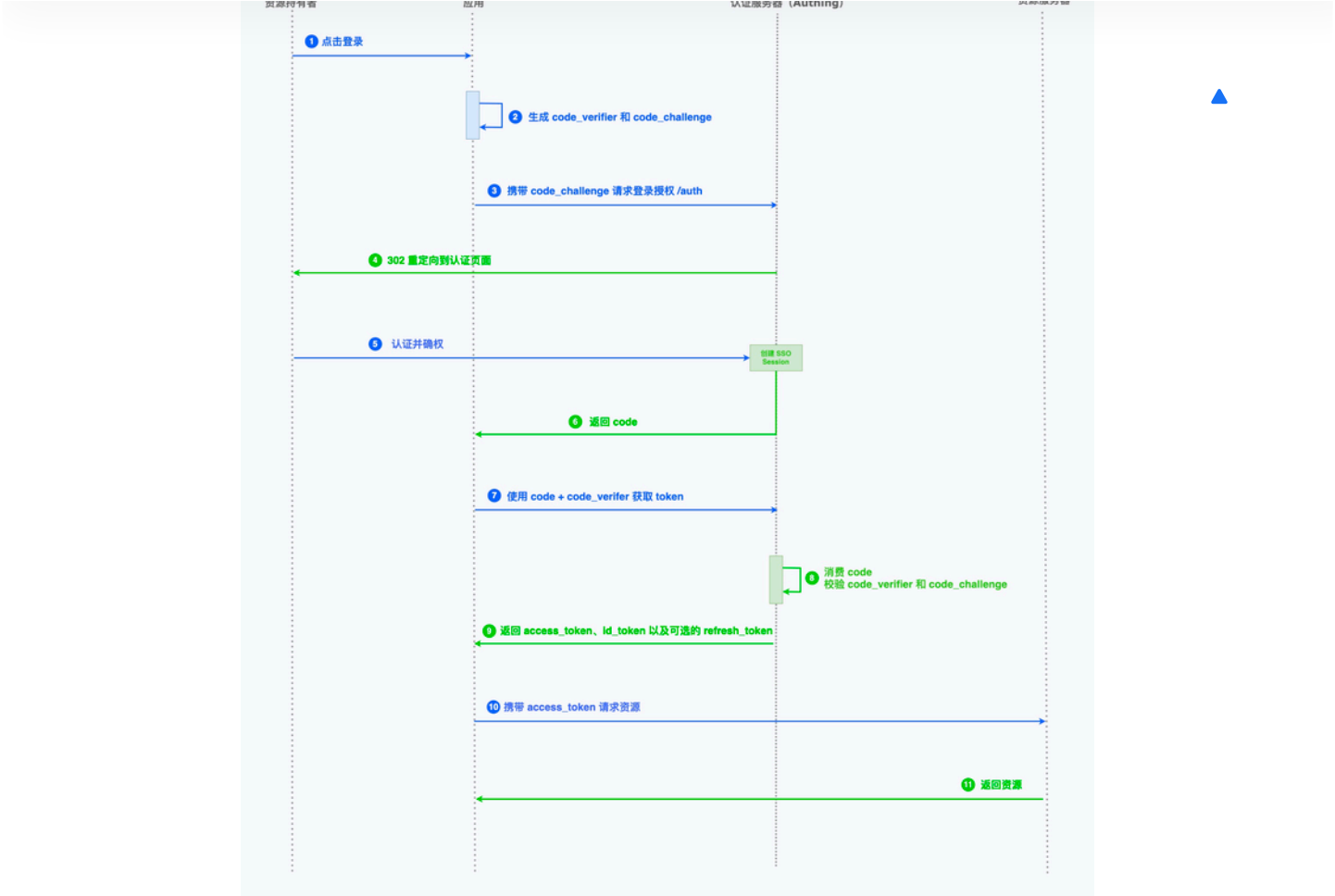
用大白话讲下就是在认证是用户携带的是加密后的 code_challenge，在用户认证成功通过 code 获取 Token 时，客户端证明自己的方式则是把 code_verifier 原文发送，认证中心收到获取 Token 请求时通过 code_verifier + code_challenge_method 进行转换，发现最终结果与 code_challenge 匹配则返回 Token，否则拒绝。

1.1 整体流程

整体上，有以下流程:

| |
|---|
| 1.用户点击登录。 |
| 2.在你的应用中，生成 code_verifier 和 code_challenge。 |
| 3. 拼接登录链接(包含 code_challenge) 跳转到 Authing 请求认证。 |
| 4. Authing 发现用户没有登录，重定向到认证页面，要求用户完成认证。 |
| 5. 用户在浏览器完成认证。 |
| 6. Authing 服务器通过浏览器通过重定向将授权码（code）发送到你的应用前端。 |
| 7. 你的应用将授权码 (code) 和 code_verifier 发送到 Authing 请求获取 Token. |
| 8. Authing 校验 code、code_verifier 和 code_challenge。 |
| 9. 校验通过，Authing 则返回 AccessToken 和 IdToken 以及可选的 RefreshToken。 |
| 10. 你的应用现在知道了用户的身份，后续使用 AccessToken 换取用户信息，调用资源方的 API 等 |

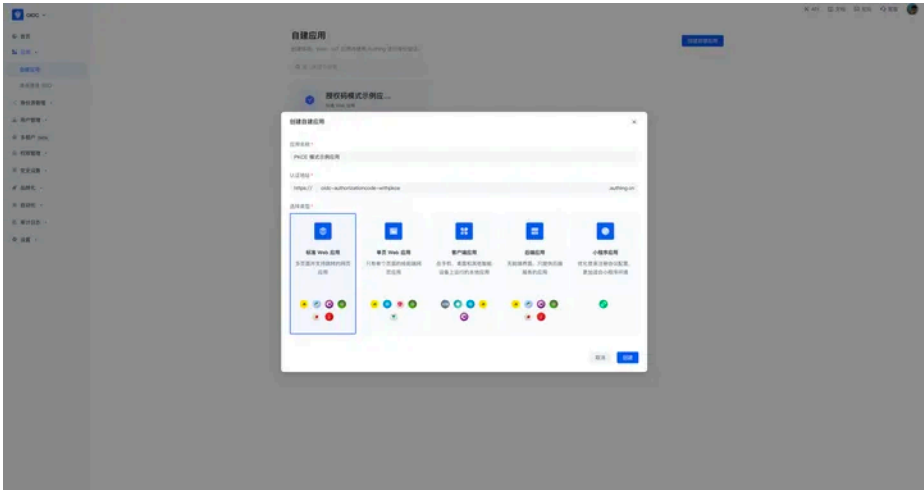
知乎



1.2 准备接入

1.2.1 整体流程

需要先在 Authing 创建应用：



配置登录回调和登出回调，配置为你实际项目的地址，我们在这里配置 localhost 用于测试。

若你想匹配多个登录/登出回调

可以使用 “*” 号进行通配，登录/登出回调可以是如下格式

知乎

| | |
|----------------------------|--|
| https://www.authing.cn/* | 路径 |
| https://*.authing.cn/index | 代表可以回调到使用 https 协议 authing.cn 任意二级域名下的首页，例如 https://abc.authing.cn/index |
| http://*.authing.cn/* | 代表可以回调到使用 https 协议 authing.cn 任意二级域名下任意子域下的任意路径，例如 |

PKCE 模式示例应用

数据概览 接入教程 体验登录

快速开始 应用配置 协议配置 登录控制 访问授权 高级配置 租户配置

基本信息


应用名称 *

PKCE 模式示例应用

应用描述

请输入应用描述

应用 logo



保存 重置

端点信息

App ID

63f305b629268cc27d93c6

JWKS 公钥端点

https://oidc-authorizationcode-withpkce.authing.cn/oidc/well-known/jwks.json

App Secret

99593***** 刷新密钥

Token 端点

https://oidc-authorizationcode-withpkce.authing.cn/oidc/token

Issuer

https://oidc-authorizationcode-withpkce.authing.cn/oidc

用户信息端点

https://oidc-authorizationcode-withpkce.authing.cn/oidc/me

服务发现地址

https://oidc-authorizationcode-withpkce.authing.cn/oidc/well-known/openid-configuration

登出端点

https://oidc-authorizationcode-withpkce.authing.cn/oidc/session/end

认证配置

认证地址 *

https:// oidc-authorizationcode-withpkce .authing.cn

登录回调 URL *

http://localhost:8080/

登出回调 URL

多个 URL 用英文逗号「,」隔开

发起登录 URL

保存 重置

在协议配置中，我们勾选 authorization_code 并且使用 code 作为返回类型，如下图所示：

PKCE 模式使用的是 code_verifier 来换取 Token，所以需要配置获取 Token 的方式为 null

PKCE 模式示例应用

数据概览 接入教程 体验登录

快速开始 应用配置 协议配置 登录控制 访问授权 高级配置 租户配置

默认协议类型

OIDC

OAuth 2.0

SAML 2.0

CAS

授权配置

OIDC

OAuth 2.0

SAML 2.0

CAS

当你使用 Authing 的身份认证功能时，默认使用 OIDC (OpenID Connect) 协议进行认证。不清楚应该选用哪种授权模式？

授权模式

☒ authorization_code

☐ implicit

☒ refresh_token

☐ password

☐ client_credentials

☐ authing_token

返回类型

☐ code_id_token

☐ id_token

☐ code_token

☒ code

☐ id_token_token

☐ id_token

id_token 签名算法

☒ HS256

☐ RS256

☐ 不强制 implicit 模式回调链接为 https

☐ 启用 id_token 加密

获取 token 身份验证方式

☐ client_secret_post

☐ client_secret_basic

☒ none

校验 token 身份验证方式

☒ client_secret_post

☐ client_secret_basic

☐ none

撤回 token 身份验证方式

☒ client_secret_post

☐ client_secret_basic

☐ none

授权码过期时间

600 秒

id_token 过期时间

1209600 秒

access_token 过期时间

1209600 秒

refresh_token 过期时间

2592000 秒

1.3.1 所需调用接口列表

1.3.2 Run in Postman所需调用接口列表

<https://app.getpostman.com/run-collection/24730905-5d29e488-719e-4ffe-af21-a7c18298d328?action=collection%2Ffork&collection-url=entityId%3D24730905-5d29e488-719e-4ffe-af21-a7c18298d328%26entityType%3Dcollection%26workspaceId%3D13ff793c-024c-459d-b1f6-87e91c4769ed#env%5BAuthing%20OIDC%5D=W3sia2V5JjoiaG9zdClzInZhbHVlIjoiaHR0cHM6Ly9kZWVwbGFuZy5hdXRoaW5nLmNuliwiZW5hYmxlZCI6dHJ1ZSwidHlwZSI6ImRlZmF1bHQifSx7ImtleSI6ImNsaVVudF9pZCIsbnZhbHVlIjoiaXNmMmNmNDg2ZTVhbjk0NDNhZjI1bHQifSx7ImtleSI6ImNsaVVudF9pZWIwNyXQqLlCJ2YWx1ZSI6Ijc3NWMyM2NiMjkwyZkwZDQwNDUxNGU3MDgyMDkzZWlziWiZW5hYmxlZCI6dHJ1ZSwidHlwZSI6ImRlZmF1bHQifSx7ImtleSI6ImFjY2Vzc190b2tlibiIsbnZhbHVlIjoilwiZW5hYmxlZCI6dHJ1ZSwidHlwZSI6ImRlZmF1bHQifSx7ImtleSI6ImlkX3Rva2VuliwidmFsdWUiOiIlLCJlbmFibGVkljp0cnVILCJ0eXBlljoiaGVmYXVsdCJ9LHsia2V5JjoicmVmcmVzaF90b2tlibiIsbnZhbHVlIjoilwiZW5hYmxlZCI6dHJ1ZSwidHlwZSI6ImRlZmF1bHQifV0=>

1.3.3 发起登录

```
GET${host}/oidc/auth
```

这是基于浏览器的 OIDC 的起点，请求对用户进行身份验证，并会在验证成功后返回授权码到您所指定的 redirect uri。

生成 code challenge 和 code verifier

在线生成

<https://tonyxu-io.github.io/pkce-generator/>

离线生成

首先，我们要生成一个 `code_challenge` 和 `code_verifier`，以下是使用 JavaScript 语言生成 PKCE 所需要的 `code verifier` 和 `code challenge` 的脚本：

```
// 生成随机字符串

function generateRandomString(length) {
    var result = '';
    var characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    var charactersLength = characters.length;
    for (var i = 0; i < length; i++) {
        result += characters.charAt(Math.floor(Math.random() * charactersLength));
    }
    return result;
}

// 生成 code_verifier
var codeVerifier = generateRandomString(128);
// 对 code_verifier 进行 SHA-256 编码，并将其转换为
var sha256 = new jsSHA("SHA-256", "TEXT");
```



```
.replace(/\/+/g, '-').replace(/\\//g, '_').replace(/=+$/ , '');
// 将 code_verifier 和 code_challenge 用对象形式返回
var pkce = {
  codeVerifier: codeVerifier,
  codeChallenge: codeChallenge
};
```

以上代码使用 jsSHA 库计算 SHA-256 哈希值，使用 base64url 编码将哈希值转换为 code_challenge。你可以将以上代码复制到你的 JavaScript 代码中，并使用 pkce.codeVerifier 和 pkce.codeChallenge 调用 OAuth 2.0 授权请求。

举例

code_verifier:

4aHg5fN1AGdbnBAfVKMf9ZMK4PUOBTwQSKKk9V8wYXOFYDZkIMI7dzDUhnQi4sYhzGb6
PWCKnQqLP70K1DNOneEDq8iyASepAdGjGBBmCs4BGCDJNwLrGpnJEfmrI66

code_verifier 的长度为 43 ~ 128，我们生成的是 128 位

code_challenge:

OhMk95M9qWkKd06--utVtRzQh8Y0Qtqo4cPqzqMJyMw

发起登录地址（浏览器中打开）

https://{host}/oidc/auth?
scope=openid+profile+offline_access+username+email+phone&redirect_uri=http://local
host:8080/&response_type=code&prompt=consent&nonce=6e187def-1a19-4067-8875-
653f024d5a9f&client_id={client_id}&state=1676881862&code_challenge=
{code_challenge}&code_challenge_method=S256

体验地址

[oidc-authorizationcode-withpkce.authing.cn...](#)

参数说明

| Parameter | Description | Param Ty | Data Type | Required |
|-----------------------|--|----------|-----------|----------|
| client_id | 客户端 ID | Query | String | TRUE |
| redirect_uri | 认证成功发送授权码或令牌的回调地址，必须与在 Authing 应用中预先注册的值相匹配 | Query | String | TRUE |
| response_type | 可选值 code、token 和 id_token | Query | String | TRUE |
| scope | 参考 https://docs.authing.cn/v2/concepts/oidc-common-questions.html#scope-%E5%8F%82%E6%95%B0%E5%AF%B9%E5%BA%94%E7%9A%84%E7%94%A8%E6%88%B7%E4%BF%A1%E6%81%AF | Query | String | TRUE |
| nonce | idToken 中返回的值。它用于减轻重放攻击 | Query | String | FALSE |
| prompt | 有效值：none、enroll_authenticator、consent、login、或 consent and login 以任意顺序。 | Query | String | FALSE |
| state | 推荐，用于维护请求和回调之间状态的不透明值。通常，跨站点请求伪造（CSRF、XSRF）缓解是通过将此参数的值与浏览器 cookie 进行加密绑定来实现的。 | Query | String | FALSE |
| code_challenge | PKCE 的挑战，在使用 PKCE 模式时必须 | Query | String | FALSE |
| code_challenge_method | 用于生成 PKCE code_challenge 的加密方法。有效值：S256 | Query | String | FALSE |

1.3.4 获取 Token

POST\${host}/oidc/token

用户在 Authing 侧完成登录操作后，Authing 会将生成的 code 作为参数回调到 redirect_uri 地址，此时通过 code 换 token 接口即可拿到对应的访问令牌 access_token

请求参数



| | | | |
|---------------|---|--------|------|
| | 10 分钟，且只能使用一次。 | | |
| redirect_uri | 当 grant_type 是 authorization_code 则为必填项，指定发送授权的重定向地址，与发起登录时的 redirect_uri 应当保持一致。 | String | TRUE |
| grant_type | 可以是以下内容之一：authorization_code、password、client_credentials、refresh_token。 | String | TRUE |
| code_verifier | 如果 grant_type 是 authorization_code，并且在原始/授权请求中指定了 code_challenge，则为必填项。此值是 PKCE 的代码验证器。 | String | TRUE |

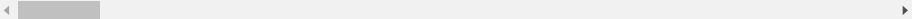


请求示例

```
curl --location --request POST 'https://{host}/oidc/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'client_id={应用ID}' \
--data-urlencode 'client_secret={应用密钥}' \
--data-urlencode 'grant_type=authorization_code' \
--data-urlencode 'redirect_uri={发起登录时指定的 redirect_uri}' \
--data-urlencode 'code={/oidc/auth 返回的代码}' \
--data-urlencode 'code_verifier={code_verifier}'
```

响应示例（成功）

```
{
  "scope": "openid username email phone offline_access profile",
  "token_type": "Bearer",
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6ImVtSzBGbVRlYX0x1QWFjeS1YW",
  "expires_in": 1209600,
  "id_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI2M2ViNTNjNDQxYTVjMmYwNWYy",
  "refresh_token": "KanvCEmonS_FgCRdFft0Cwka2f8Qjj4tcsIfJF-VC1W"
}
```



响应示例（失败）

```
{
  "error": "invalid_grant",
  "error_description": "授权码无效或已过期"
}
```

1.3.5 所需调用接口列表

GET\${host}/oidc/me 获取用户信息

此端点是 OIDC 获取用户端点，可以通过 AccessToken 获取有关当前登录用户的信息。

请求参数

| Parameter | Description |
|--------------|--------------------------------|
| access_token | /oidc/token 端点返回的 access_token |

请求示例

```
curl --location --request GET 'https://{host}/oidc/me?access_token={access_token}'
```

响应示例（成功）

```
{
  "name": null,
  "given_name": null,
  "middle_name": null,
  "family_name": null,
  "nickname": null,
  "preferred_username": null,
  "profile": null,
  "picture": "https://files.authing.co/authing-coi
```



```
"gender": "U",
"zoneinfo": null,
"locale": null,
"updated_at": "2023-02-14T09:26:28.068Z",
"email": "xxx@authing.cn",
"email_verified": true,
"phone_number": "185xxxx9995",
"phone_number_verified": true,
"username": "neo",
"sub": "63eb53c441a5c2f05f24bb03"
}
```

响应示例（失败）

```
{
  "error": "invalid_grant",
  "error_description": "Access Token 无效"
}
```

1.3.6 校验 Token

```
POST${host}/oidc/auth
```

此端点接受 access_token、id_token、refresh_token，并返回一个布尔值，指示它是否处于活动状态。如果令牌处于活动状态，还将返回有关令牌的其他数据。如果 token 无效、过期或被吊销，则认为它处于非活动状态。

access_token 可以使用 RS256 签名算法或 HS256 签名算法进行签名。下面是这两种签名算法的区别：

RS256 是使用 RSA 算法的一种数字签名算法，它使用公钥/私钥对来加密和验证信息。RS256 签名生成的令牌比 HS256 签名生成的令牌更加安全，因为使用 RSA 密钥对进行签名可以提供更高的保护级别。使用 RS256 签名算法的令牌可以使用公钥进行验证，公钥可以通过 JWK 端点获取。

HS256 是使用对称密钥的一种数字签名算法。它使用同一个密钥进行签名和验证。HS256 签名算法在性能方面比 RS256 签名算法更快，因为它使用的是对称密钥，而不是使用 RSA 公钥/私钥对来签名和验证。使用 HS256 签名算法的令牌可以通过 shared secret（应用密钥）进行验证。

在实际应用中，RS256 算法更加安全，但同时也更加消耗资源，如果系统需要高性能，可以选择 HS256 签名算法。

验证 Token 分为两种方式

本地验证与使用 Authing 在线验证。我们建议在本地验证 JWT Token，因为可以节省你的服务器带宽并加快验证速度。你也可以选择将 Token 发送到 Authing 的验证接口由 Authing 进行验证并返回结果，但这样会造成网络延迟，而且在网络拥塞时可能会有慢速请求。

以下是本地验证和在线验证的优劣对比：

| | 验证速度 | 代码复杂度 | 可靠程度 |
|------|------|-------|--------|
| 在线验证 | 慢 | 简单 | 单点故障风险 |
| 本地验证 | 快 | 一般 | 分布式 |

在线校验

需要注意的是，id_token 目前无法在线校验，因为 id_token 只是一个标识，若需要校验 id_token 则需要您在离线自行校验

请求参数

知乎

| token | access token 、 refresh token | string |
|-----------------|--|---------------|
| token_type_hint | 传递的令牌类型 Valid values: access_token refresh_token | String (Enum) |
| client_id | 应用 ID | String |
| client_secret | 应用密钥 | String |

请求示例

```
curl --location --request POST 'https://{host}/oidc/token/introspection' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'client_id={应用ID}' \
--data-urlencode 'client_secret={应用密钥}' \
--data-urlencode 'token={ token }' \
--data-urlencode 'token_type_hint={token_type_hint}'
```

校验 access_token 响应示例 (校验通过)

```
{
  "active": true,
  "sub": "63eb53c441a5c2f05f24bb03",
  "client_id": "63eb4585156d977101dd3750",
  "exp": 1677648467,
  "iat": 1676438867,
  "iss": "https://oidc-authorization-code.authing.cn/oidc",
  "jti": "ObgavGBUocr1wsrUvtDLHmuFSgoebxsi0Y4JNRqIhaQ",
  "scope": "offline_access username profile openid phone email",
  "token_type": "Bearer"
}
```

校验 access_token 响应示例 (校验未通过)

```
{
  "active": false
}
```

校验 refresh_token 响应示例 (校验通过)

```
{
  "active": true,
  "sub": "63eb53c441a5c2f05f24bb03",
  "client_id": "63eb4585156d977101dd3750",
  "exp": 1679030867,
  "iat": 1676438867,
  "iss": "https://oidc-authorization-code.authing.cn/oidc",
  "jti": "6F2T01v1YZ1_N7I3jXYHjK-vZzKtLD0IiP5KP0UFUCQ",
  "scope": "offline_access username profile openid phone email"
}
```

校验 refresh_token 响应示例 (校验未通过)

```
{
  "active": false
}
```

离线校验

可参考文档 (Authing 开发者文档) :



我们简单说下，若您使用离线校验应该对 token 进行如下规则的校验

- 1.格式校验 - 校验 token 格式是否是 JWT 格式
- 2.类型校验 - 校验 token 是否是目标 token 类型，比如 access_token 、 id_token、refresh_token
- 3.issuer 校验 - 校验 token 是否为信赖的 issuer 颁发
- 4.签名校验 - 校验 token 签名是否由 issuer 签发，防止伪造
- 5.有效期校验 - 校验 token 是否在有效期内
- 6.claims 校验 - 是否符合与预期的一致

示例代码

下面是一个示例 Java 代码，可以用于在本地校验 OIDC RS256 和 HS256 签发的 access_token

```
import com.nimbusds.jose.JWSObject;
import com.nimbusds.jwt.JWTClaimsSet;
import com.nimbusds.jwt.SignedJWT;
import java.net.URL;
import java.text.ParseException;
import java.util.Date;
public class OIDCValidator {
    private static final String ISSUER = "https://your-issuer.com";
    private static final String AUDIENCE = "your-client-id";
    private final URL jwkUrl;
    public OIDCValidator(final URL jwkUrl) {
        this.jwkUrl = jwkUrl;
    }
    public JWTClaimsSet validateToken(final String accessToken) throws ParseException {
        final SignedJWT signedJWT = SignedJWT.parse(accessToken);
        if (signedJWT == null) {
            throw new RuntimeException("Access token is null or empty");
        }
        final JWTClaimsSet claims = signedJWT.getJWTClaimsSet();
        if (claims == null) {
            throw new RuntimeException("No claims present in the access token");
        }
        if (!claims.getIssuer().equals(ISSUER)) {
            throw new RuntimeException("Invalid issuer in access token");
        }
        if (!claims.getAudience().contains(AUDIENCE)) {
            throw new RuntimeException("Invalid audience in access token");
        }
        final JWSObject jwsObject = signedJWT.getJWSObject();
        if (jwsObject == null) {
            throw new RuntimeException("No JWS object found in the access token");
        }
        // Fetch the JWKs from the JWK set URL
        final JWKSet jwkSet = JWKSet.load(jwkUrl);
        final JWK jwk = jwkSet.getKeyByKeyId(jwsObject.getHeader().getKeyID());
        if (jwk == null) {
            throw new RuntimeException("No JWK found for the access token");
        }
        if (!jwsObject.verify(jwk.getKey())) {
            throw new RuntimeException("Invalid signature in access token");
        }
        if (claims.getExpirationTime() == null || claims.getExpirationTime().before(new Date())) {
            throw new RuntimeException("Expired access token");
        }
        return claims;
    }
}
```

[illegible]

1.3.7 刷新 Token

请求参数

| Parameter | Description |
|---------------|--|
| client_id | 应用 ID |
| client_secret | 应用密钥 |
| grant_type | 可以是以下内容之一： authorization_code、password、client_credentials、refresh_token。 |
| refresh_token | 如果 grant_type 为 refresh_token ， 则为必填项。 |

```
curl --location --request POST 'https://{host}/oidc/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'client_id={应用ID}' \
--data-urlencode 'client_secret={应用密钥}' \
--data-urlencode 'refresh_token={刷新令牌}' \
--data-urlencode 'grant_type=refresh token'
```

```
{
  "refresh_token": "6F2T01v1YZ1_N7I3jXYHJk-vZzkLtD0IIP5KPoUFUCQ",
  "scope": "offline_access username profile openid phone email",
  "token_type": "Bearer",
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6ImVtSzBGbVRlIa0xLWJjeiJ9.eyJzdWIiOiI2MzViNTNjNDQxYTVjMmYwIiwiaWF0IjoiMTYyOTg0MDAwfQ.",
  "expires_in": 1209600,
  "id_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsInR5cCI6IkpXLTJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6ImVtSzBGbVRlIa0xLWJjeiJ9.eyJzdWIiOiI2MzViNTNjNDQxYTVjMmYwIiwiaWF0IjoiMTYyOTg0MDAwfQ."
}
```

```
{
  "error": "invalid_grant",
  "error_description": "Refresh Token 无效或已过时"
}
```



POST\${host}/oidc/auth

撤销 access_token / refresh_token 。



请求参数

| Parameter | Description |
|-----------------|---|
| token | access token or refresh token |
| token_type_hint | 传递的令牌类型 Valid values: access_token, refresh_token |
| client_id | 应用 ID |
| client_secret | 应用密钥 |

请求示例

```
curl --location --request POST 'https://{host}/oidc/token/revocation' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'client_id={应用ID}' \
--data-urlencode 'client_secret={应用密钥}' \
--data-urlencode 'token= {token}' \
--data-urlencode 'token_type_hint={token_type_hint}'
```

响应示例(成功)

HTTP 200 OK

响应示例(失败)

```
{
  "error": "xxxx",
  "error_description": "xxxx"
}
```

1.3.9 用户登出

GET\${host}/oidc/session/end

使用此操作通过删除用户的浏览器会话来注销用户。

post_logout_redirect_uri 可以指定在执行注销后重定向的地址。否则，浏览器将重定向到默认页面

请求参数

| Parameter | Description | Type | Required |
|--------------------------|------------------------------------|--------|----------|
| id_token_hint | id_token | String | TRUE |
| post_logout_redirect_uri | 登出后的重定向地址 | String | FALSE |
| state | 在重定向到 post_logout_redirect_uri 时作为 | String | FALSE |

请求示例(浏览器访问)

```
GET https://oidc-authorization-code.authing.cn/oidc/session/end?id_token_hint={id_toke
```

02.本章总结

模式你就基本掌握啦。

接下来我们还会介绍 OIDC 的授权码+PKCE 流程，以及接入 [Authing](#) 的方式，需要你对授权码模式的流程有一定了解哦。

往期精彩内容

[什么是事件驱动（EDA）？为什么它是技术领域的主要驱动力？](#)



编辑于 2023-05-31 17:30 · IP 属地北京

[OAuth](#) [OAuth 2.0](#) [安全认证](#)



欢迎参与讨论

2 条评论

默认 最新



小蚂蚁说故事

觉得既然是只有客户端没有服务端，code_verifier 和 code_challenge 都能够被拿到啊，还说是为了方式secret被窃取.... 感觉这绕来绕去 没有任何卵用啊 🤔

2023-10-31 · IP 属地山西

回复 喜欢



风萧萧兮、

的确，如果设想code_challenge 可能被拿到，那为啥第一次发送的code_verifier 就拿不到？

2023-11-27 · IP 属地广东

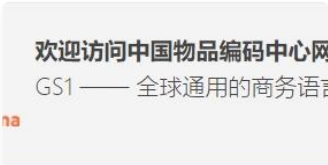
回复 喜欢

推荐阅读



OAuth系列篇之授权码的“推测”

Seide... 发表于我的三叉戟



如何申请条形码(EAN码)？

OnMyWay

一篇文章告诉你：范围

早上一个质量人在认证范围如何确定，不清楚加上这些年工作企业不知道如何确定的情况，在这里结合自己的工作经验元质量