Spring Authorization Server入门 (十七) Vue项目使用授权码模式对接认证服务

叹雪飞花 2023-09-17 761 阅读4分钟

Vue单页面项目使用授权码模式对接流程说明

以下流程摘抄自官网

在本例中为授权代码流程。 授权码流程的步骤如下：

1. 客户端通过重定向到授权端点来发起 OAuth2 请求。
2. 如果用户未通过身份验证，授权服务器将重定向到登录页面。 身份验证后，用户将再次重定向回授权端点。
3. 如果用户未同意所请求的范围并且需要同意，则会显示同意页面。
4. 一旦用户同意，授权服务器会生成一个 `authorization_code` 并通过 `redirect_uri` 重定向回客户端。
5. 客户端通过查询参数获取 `authorization_code` 并向Token Endpoint发起请求。

Vue项目中修改内容

安装crypto-js依赖

已安装可以忽略，该依赖是为了计算 `Code Challenge` 和base64加密使用

```shell
```

TypeScript下额外添加 @types/crypto-js 依赖

shell 复制代码

```shell
npm install @types/crypto-js
```

编写公共方法

生成随机字符串

generateCodeVerifier 函数主要是为了给PKCE流程使用，下篇文章中会说明，这里借用一下生成state，因为本质上是生成随机字符串

javascript 复制代码

```javascript
/**
 * 生成 CodeVerifier
 *
 * return CodeVerifier
 */
export function generateCodeVerifier() {
    return generateRandomString(32)
}

/**
 * 生成随机字符串
 * @param length 随机字符串的长度
 * @returns 随机字符串
 */
export function generateRandomString(length: number) {
    let text = ''
    const possible = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'
    for (let i = 0; i < length; i++) {
        text += possible.charAt(Math.floor(Math.random() * possible.length))
    }
    return text
}
```

```
/**
 * 将字符串加密为Base64格式的
 * @param str 将要转为base64的字符串
 * @returns 返回base64格式的字符串
 */
export function base64Str(str: string) {
    return CryptoJS.enc.Base64.stringify(CryptoJS.enc.Utf8.parse(str));
}

```

编写获取地址栏参数方法

```
/**
 * 根据参数name获取地址栏的参数
 * @param name 地址栏参数的key
 * @returns key对用的值
 */
export function getQueryString(name: string) {
    const reg = new RegExp('(^|&)' + name + '=([^&]*)(&|$)', 'i')

    const r = window.location.search.substr(1).match(reg)

    if (r != null) {
        return decodeURIComponent(r[2])
    }

    return null
}
```

编写请求Token方法

```
/**
 * 从认证服务获取AccessToken
 * @param data 获取token入参
 * @returns 返回AccessToken对象
 */
export function getToken(data: any) {
```

```
    // 这里这么写是为了兼容PKCE与授权码模式
    if (data.client_secret) {
        // 设置客户端的basic认证
        headers.Authorization = `Basic ${base64Str(`${data.client_id}:${data.client_secret}`)}`
        // 移除入参中的key
        delete data.client_id
        delete data.client_secret
    }
    // 可以设置为AccessToken的类型
    return loginRequest.post<any>({
        url: '/oauth2/token',
        data,
        headers
    })
}
```

在环境变量配置文件中添加配置

js 复制代码

```
# 认证服务地址(token签发地址)
VITE_OAUTH_ISSUER=http://127.0.0.1:8080
# 授权码流程使用的客户端Id
VITE_OAUTH_CLIENT_ID=messaging-client
# 授权码流程使用的客户端秘钥
VITE_OAUTH_CLIENT_SECRET=123456
# 授权码模式使用的回调地址
VITE_OAUTH_REDIRECT_URI=http://127.0.0.1:5173/OAuth2Redirect
```

创建处理回调的页面 `OAuthRedirect.vue`

页面加载时会尝试从地址栏获取参数 `code` ，如果能获取到说明是从认证服务回调过来的，执行换取token流程，如果没有获取到code说明需要发起授权申请。

我这里是将获取到的token直接存储在localStorage中了，如果有需要也可以更换存储位置、存储格式等

js 复制代码

```
import { getToken } from '@/api/Login'
import { createDiscreteApi } from 'naive-ui'
import { generateCodeVerifier } from '@/util/pkce'
import { getQueryString } from '@/util/GlobalUtils'

const { message } = createDiscreteApi(['message'])

// 生成state
let state: string = generateCodeVerifier()

// 获取地址栏授权码
const code = getQueryString('code')

if (code) {
  // 从缓存中获取 codeVerifier
  const state = localStorage.getItem('state')
  // 校验state，防止cors
  const urlState = getQueryString('state')
  if (urlState !== state) {
    message.warning('state校验失败.')
  } else {
    // 从缓存中获取 codeVerifier
    getToken({
      grant_type: 'authorization_code',
      client_id: import.meta.env.VITE_OAUTH_CLIENT_ID,
      client_secret: import.meta.env.VITE_OAUTH_CLIENT_SECRET,
      redirect_uri: import.meta.env.VITE_OAUTH_REDIRECT_URI,
      code,
      state
    })
      .then((res: any) => {
        localStorage.setItem('accessToken', JSON.stringify(res))
        router.push({ path: '/' })
      })
      .catch((e) => {
        message.warning(`请求token失败：${e.data.error || e.message || e.statusText}`)
      })
  }
} else {
  // 缓存state
  localStorage.setItem('state', state)
  window.location.href = `${import.meta.env.VITE_OAUTH_ISSUER}/oauth2/authorize?client_id=${
    import.meta.env.VITE_OAUTH_CLIENT_ID
  }&response_type=code&scope=openid%20profile%20message.read%20message.write&redirect_uri=${
    import.meta.env.VITE_OAUTH_REDIRECT_URI
  }&state=${state}`
}
```

53

添加路由

js 复制代码

```
{
    path: '/OAuth2Redirect',
    name: 'OAuth2Redirect',
    component: () => import('../views/login/OAuthRedirect.vue')
}
```

认证服务修改内容

在数据库中添加对应客户端的回调地址

重要：例如文中的就需要给客户端 `messaging-client` 添加一个回调地址

`http://127.0.0.1:5173/OAuth2Redirect`

重要：例如文中的就需要给客户端 `messaging-client` 添加一个回调地址

`http://127.0.0.1:5173/OAuth2Redirect`

重要：例如文中的就需要给客户端 `messaging-client` 添加一个回调地址

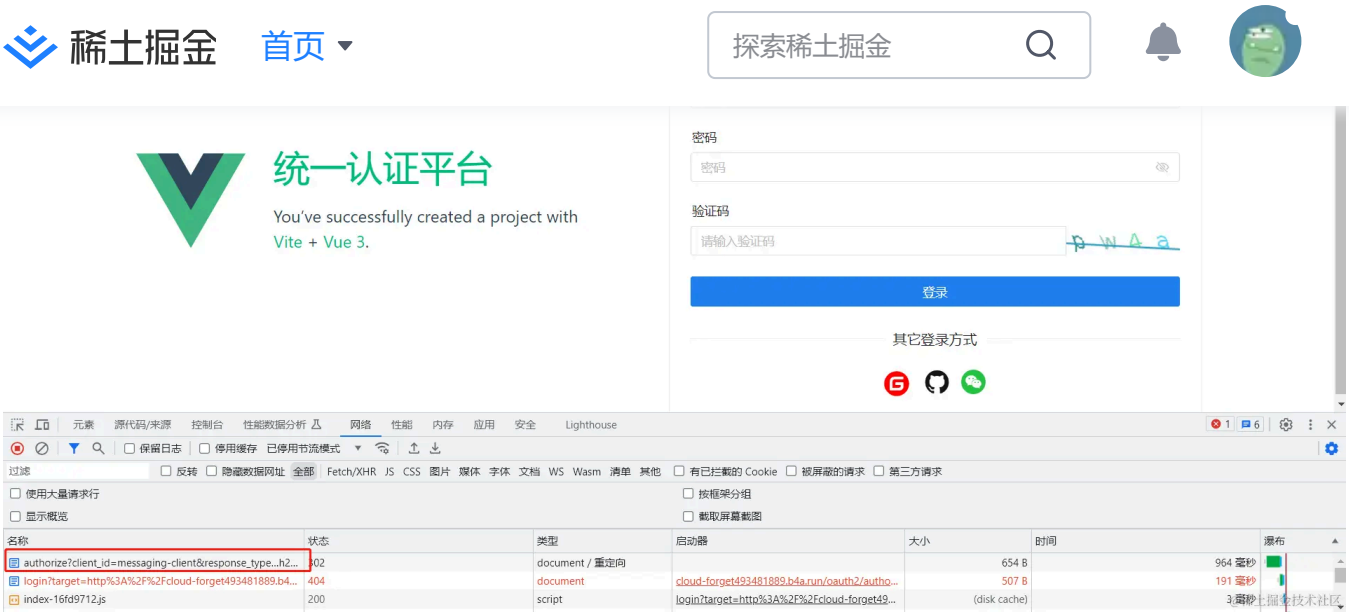
`http://127.0.0.1:5173/OAuth2Redirect`

经过以上配置授权码模式的对接就完成了，接下来就可以测试了，在首页或者需要触发登录的地方添加一个按钮，点击跳转到 `/OAuth2Redirect` 之后会自动引导发起授权申请流程。

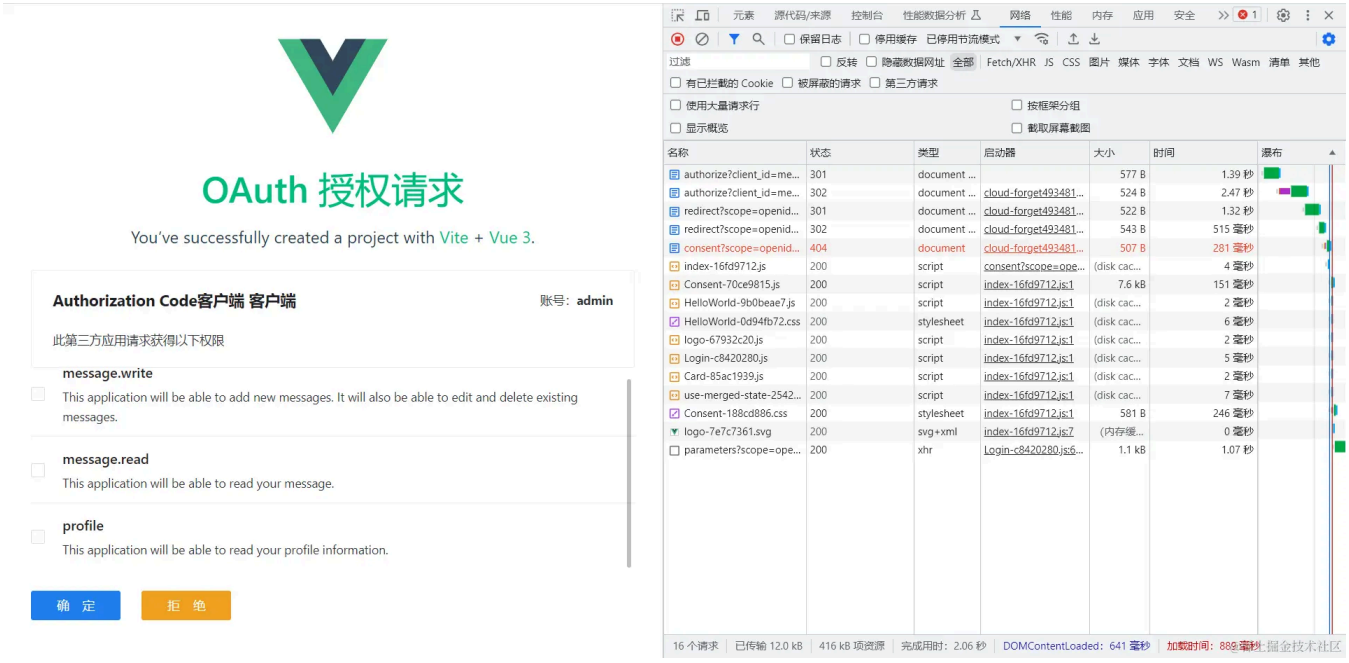
测试

点击后跳转至回调页面

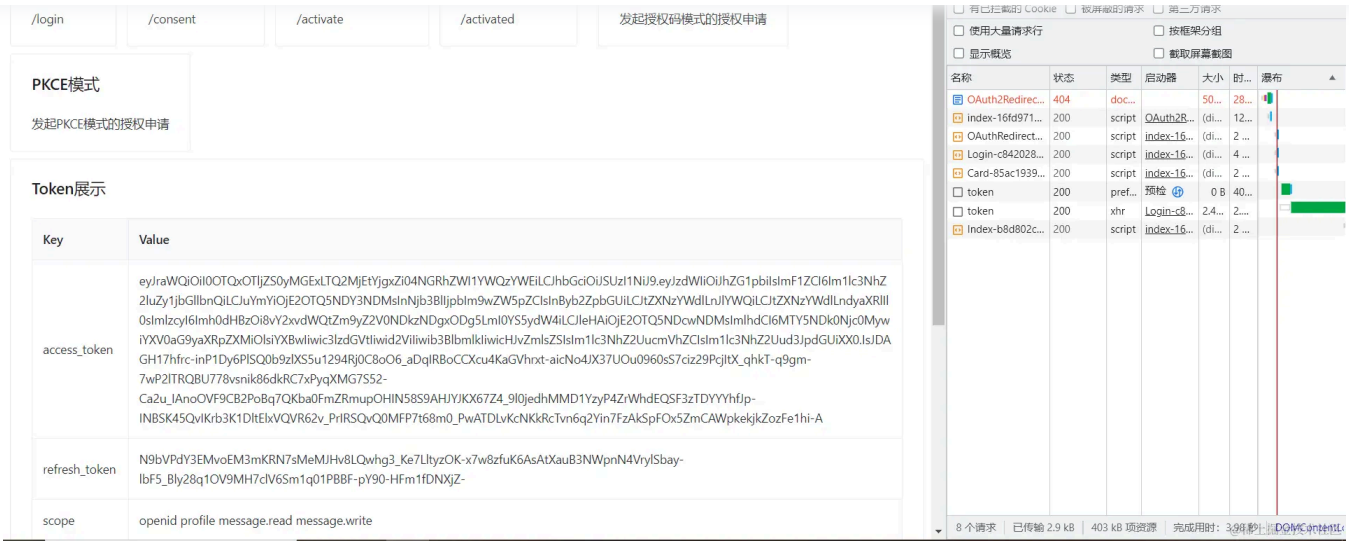
回调页面发现地址栏没有code引导发起授权申请



认证服务检测到没有登录，跳转至登录页



登录成功后发现需要授权确认，跳转至授权确认页面



授权成功后携带token重定向至回调地址，回调页根据code换取token，获取token成功后存储并返回首页

最后

实际上真正对接要写的代码并不多，逻辑也不复杂，我这里写的比较简单，并且客户端id和秘钥都是放在前端的，读者可以存放在一些更安全的地方，或加密，或从后端获取密文等，如果大家发现有什么问题可以在评论指正，谢谢

附录

代码仓库地址：Gitee

标签： Vue.js 安全 Spring Boot 话题： 日新计划更文活动

本文收录于以下专栏

上一篇 Spring Authorization Server...
下一篇 Spring Authorization Server...

评论 12

平等表达，友善交流

0 / 1000 发送

最热 最新

用户7581760356379

授权同意后携带code回调/OAuth2Redirect，前端报404，但是前端添加有路由和相关组件，是什么问题？我看上面例子也出现了404问题

3月前 点赞 1

叹雪飞花 作者：这个是因为history模式路由的问题，跳转时是通过路由处理的，但是刷新时浏览器直接去找这个路径是找不到的，这一块儿你可以百度了解下

3月前 点赞 回复

pzxm

这样是不是相当于把前端项目作为oauth2的客户端了，clientId和clientSecret不都暴露在前端了？

4月前 点赞 1

叹雪飞花 作者：是的，所以又提了PKCE的草案，使用PKCE的方式替代授权码方式，主要是使用code_verifier和code_challenge替代clientSecret，这样安全性更高

4月前 点赞 回复

flip953

端分离，试了不少方法最后也没法像单应用security那像改成SessionCreationPolicy.STATELESS的，最后只好把接口都设为permitAll()，绕过oauth2-client的认证，自己用filter另整了一套jwt认证，虽然能实现效果，但是总感觉不太好

5月前 点赞 5 ···

叹雪飞花 作者：客户端前后端分离跟认证服务没啥关系吧?

5月前 点赞 回复 ···

flip953 回复 叹雪飞花 作者：就是客户端是一个系统，引入了spring-oauth2-client包，系统前后端分离，前端调后台接口返回401（未登录）时，跳转认证服务器登录，在认证服务器登录完了后再返回到系统前端。因为网上其它单应用spring security的例子里，最后是用UsernamePasswordAuthenticationToken验证的，但是现在单点登录，密码不在客户端这边，所以犯了难。

5月前 点赞 回复 ···

查看全部 5 条回复

查看全部 12 条评论

目录 收起

测试

最后

附录

相关推荐

Spring Authorization Server入门 (十八) Vue项目使用PKCE模式对接认证服务
685阅读 · 6点赞

Spring Data Redis工具类
208阅读 · 3点赞

Spring Authorization Server入门 (六) 自定义JWT中包含的内容与资源服务jwt解析器
2.6k阅读 · 7点赞

Spring Authorization Server入门 (十四) 联合身份认证添加微信登录
1.2k阅读 · 5点赞

Spring Authorization Server优化篇：自定义UserDetailsService实现从数据库获取用户信息
1.1k阅读 · 5点赞

精选内容

Mybatis源码 - Mapper代理查询过程解析 <package>标签解析 JDK动态代理原理 代理对象创建 invoke方法
用户538661682... · 726阅读 · 2点赞

借助Numpy，优化Pandas的条件检索代码
databook · 159阅读 · 1点赞

Golang框架实战-KisFlow流式计算框架(10)-Flow多副本
刘丹冰Aceld · 173阅读 · 1点赞

探秘高并发异常背后的真相：一次排查实录
货拉拉技术 · 1.3k阅读 · 7点赞

Python办公神器：教你使用python批量制作PPT
AI小智 · 170阅读 · 1点赞

为你推荐

Spring Authorization Server入门 (十二) 实现授权码模式使用前后端分离的登录页面

叹雪飞花 8月前 4.8k 24 65 后端 Spring ... Spring

Spring Authorization Server入门 (十) 添加短信验证码方式登录

叹雪飞花 9月前 3.4k 20 9 Spring Spring ...

Spring Authorization Server入门 (八) Spring Boot引入Security OAuth2 Client对接认...

叹雪飞花 9月前 2.8k 13 43 Spring ... Spring

Spring Authorization Server入门 (十六) Spring Cloud Gateway对接认证服务

叹雪飞花 7月前 2.6k 17 44 Spring ... Spring ... 安全

Spring Authorization Server入门 (十三) 实现联合身份认证，集成Github与Gitee的OAu...

叹雪飞花 8月前 2.3k 13 51 Spring Spring ... 安全

Spring Authorization Server入门 (十一) 自定义grant_type(短信认证登录)获取token

叹雪飞花 9月前 2.5k 15 39 Spring Spring ... 安全

Spring Authorization Server入门 (七) 登录添加图形验证码

叹雪飞花 9月前 2.9k 18 4 Spring ...

SpringBoot3.x最简集成SpringDoc-OpenApi

叹雪飞花 4月前 2.3k 17 评论 后端 Spring ... Java

Spring Authorization Server入门 (九) Spring Boot引入Resource Server对接认证服务

叹雪飞花 9月前 1.8k 13 8 Spring Spring ...

Spring Authorization Server优化篇：添加Redis缓存支持和统一响应类

叹雪飞花 8月前 1.7k 6 12 Spring Spring ... 安全

Spring Authorization Server入门 (十五) 分离授权确认与设备码校验页面

叹雪飞花 7月前 1.6k 14 8 Spring ... Spring Vue.js

Spring Authorization Server入门 (十九) 基于Redis的Token、客户端信息和授权确认信...

叹雪飞花 4月前 1.2k 8 19 Spring ... 后端 Redis

Spring Cloud Gateway集成SpringDoc，集中管理微服务API

叹雪飞花 3月前 767 7 评论 Spring ... Spring ... Java