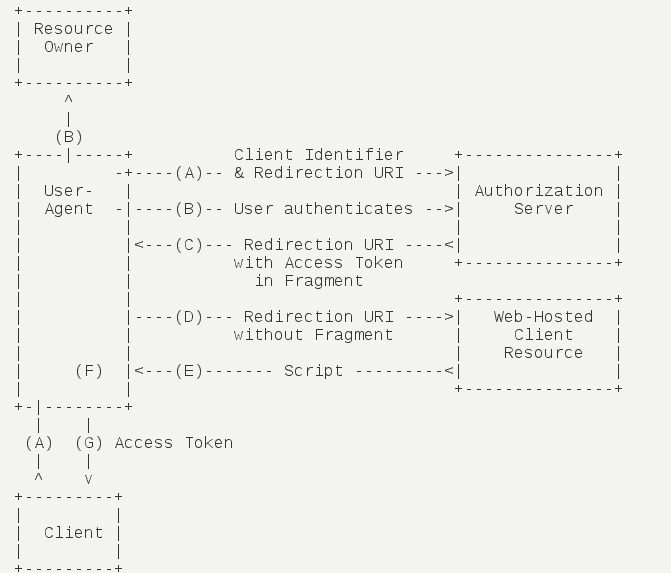
OAuth2.0 授权模式

# 授权码模式（最全模式）

**授权码模式**

* 授权码模式是功能最完整、流程最严密的授权模式，它的特点是通过客户端的后台服务器，与“服务器提供”的认证服务器进行互动



* 它的步骤如下：
  + （A）用户访问客户端，后者将前者导向认证服务器
  + （B）用户选择是否给予客户端授权
  + （C）假设用户给予授权，认证服务器将用户导向客户端事先指定的“重定向 URI”，同时附上一个授权码
  + （D）客户端收到授权码，附上早先的“重定向 URI”向认证服务器申请令牌，这一步是在客户端的后台服务器上完成的，对用户不可见
  + （E）认证服务器核对了授权码和重定向URI，确认无误后向客户端发送令牌和更新令牌
* 上述步骤中所需要的参数：
  + A步骤中，客户端申请认证的 URI，包含以下参数：
    - repsone\_type：授权类型，必选，此处固定值“code”
    - client\_id：客户端的ID，必选
    - client\_secret：客户端的密码，可选
    - redirect\_uri：重定向URI，可选
    - scope：申请的权限范围，可选
    - state：客户端当前的状态，可以指定任意值，认证服务器会原封不动的返回这个值

 GET /authorize?response\_type=code&client\_id=s6BhdRkqt3&state=xyz

&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1

Host: server.example.com

 C步骤中，服务器回应客户端的URI，包含以下参数：

* code：表示授权码，必须按，该码有效期应该很短，通常10分钟，客户端只能使用一次，否则会被授权服务器拒绝，该码与客户端 ID 和 重定向 URI 是一一对应关系
* state：如果客户端请求中包含着歌参数，认证服务器的回应也必须一模一样包含这个参数

 HTTP/1.1 302 Found

Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA

&state=xyz

 D步骤中，客户端向认证服务器申请令牌的HTTP请求，包含以下参数：

* grant\_type：表示使用的授权模式，必选，此处固定值为“authorization\_code”
* code：表示上一步获得的授权吗，必选
* redirect\_uri：重定向URI，必选，与步骤 A 中保持一致
* client\_id：表示客户端ID，必选

 POST /token HTTP/1.1

Host: server.example.com

Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

Content-Type: application/x-www-form-urlencoded

grant\_type=authorization\_code&code=SplxlOBeZQQYbYS6WxSbIA

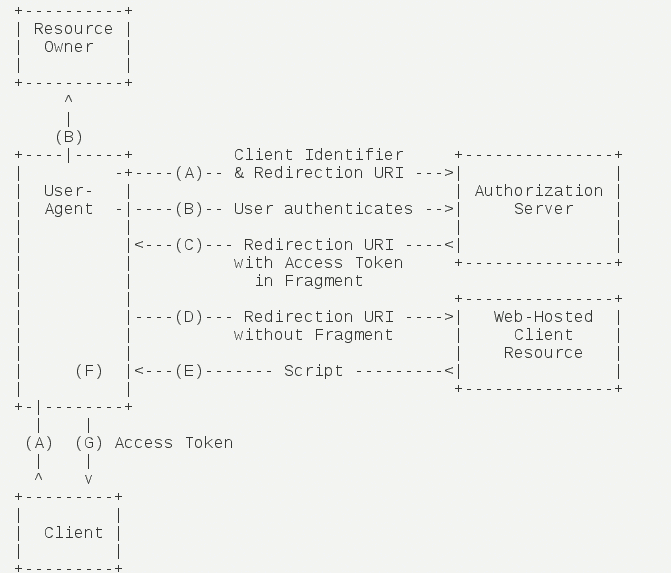
&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb

 E步骤中，认证服务器发送的HTTP回复，包含以下参数：

* access\_token：表示令牌，必选
* token\_type：表示令牌类型，该值大小写不敏感，必选，可以是 bearer 类型或 mac 类型
* expires\_in：表示过期时间，单位为秒，若省略该参数，必须设置其它过期时间
* refresh\_token：表示更新令牌，用来获取下一次的访问令牌，可选
* scope：表示权限范围
  + HTTP/1.1 200 OK
  + Content-Type: application/json;charset=UTF-8
  + Cache-Control: no-store
  + Pragma: no-cache
  + {
  + "access\_token":"2YotnFZFEjr1zCsicMWpAA",
  + "token\_type":"example",
  + "expires\_in":3600,
  + "refresh\_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  + "example\_parameter":"example\_value"
  + }
  + 从上面代码可以看到，参数使用 JSON 格式发送（Content-Type: application/json），才外，HTTP头信息中明确指定不得缓存

**简化模式**

* 简化模式不通过第三方应用程序的服务器，直接在浏览器中向认证服务器申请令牌，跳过了“授权码”这个步骤



* 它的步骤如下：
  + （A）客户端将用户导向认证服务器
  + （B）用户决定是否给予客户端授权
  + （C）假设用户给予授权，认证服务器将用户导向客户端指定的“重定向URI”，并在URI的Hash部分包含了访问令牌
  + （D）浏览器向资源服务器发出请求，其中不包括上一步收到的Hash值
  + （E）资源服务器返回一个网页，其中包含了代码可以获取Hash值中的令牌
  + （F）浏览器执行上一步获得的脚本，取出令牌
  + （G）浏览器将令牌发给客户端
* 上述步骤中所需要的参数：
  + A步骤中，客户端发出HTTP请求，包含以下参数：
    - response\_type：表示授权类型，此处固定值为"token"，必选
    - client\_id：表示客户端ID，必选
    - redirect\_uri：表示重定向URI，可选
    - scope：表示权限范围，可选
    - state：表示客户端当前状态，可指定任意值，认证服务器会原封不动返回这个值

 GET /authorize?response\_type=token&client\_id=s6BhdRkqt3&state=xyz

&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1

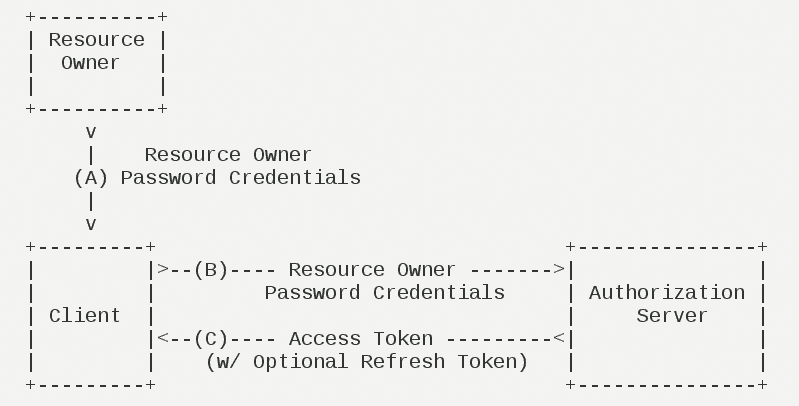
Host: server.example.com

 C步骤中，认证服务器回应客户端的URI，包含以下参数：

* access\_token：表示访问令牌，必选
* token\_type：表示令牌类型，该值大小写不敏感，必选
* expires\_in：表示过期时间，单位为秒
* scope：表示权限范围，如果与客户端申请的范围一致，可忽略
* state：如果客户端请求中包含这个参数，认证服务器也要返回一模一样的参数
  + HTTP/1.1 302 Found
  + Location: http://example.com/cb#access\_token=2YotnFZFEjr1zCsicMWpAA
  + &state=xyz&token\_type=example&expires\_in=3600
* 上面例子中，认证服务器用HTTP头信息的Location栏，指定浏览器重定向的网址，注意，这个网址的Hash部分包含了令牌
* 根据D步骤，下一步浏览器会访问Location指定的网址，但是Hash部分不会被发送，接下来的E步骤，服务提供商的资源服务器发送过来的代码，提取出Hash令牌

**密码模式**

* 密码模式中，用户向客户端提供自己的用户名和密码，客户端使用这些信息向“服务提供商”索要授权
* 在这种模式中，用户必须把密码给客户端，但客户端不得存储密码，这通常在用户对客户端高端信任的情况下，比如客户端是操作系统的一部分，由一个著名的公司出品，而认证服务器只有在其它授权模式无法执行的情况下，才考虑该模式



9434708-6165f69e2bfc8881.png

* 它的步骤如下：
  + （A）用户向客户端提供用户名和密码
  + （B）客户端将用户名密码发送认证给服务器，向后者请求令牌
  + （C）认证服务器确认无误后，向客户端提供访问令牌
* 上述步骤中所需要的参数：
  + B步骤中，客户端发出HTTP请求，包含以下参数：
    - grant\_type：授权类型，必选，此处固定值“password”
    - username：表示用户名，必选
    - password：表示用户密码，必选
    - scope：权限范围，可选

 POST /token HTTP/1.1

Host: server.example.com

Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

Content-Type: application/x-www-form-urlencoded

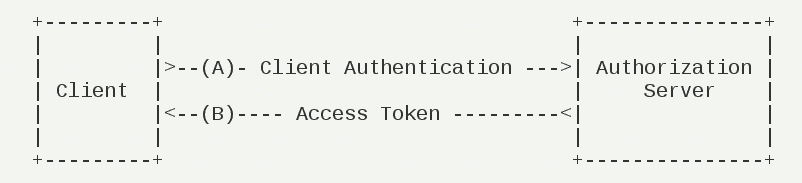
grant\_type=password&username=johndoe&password=A3ddj3w

 C步骤中，认证服务器向客户端发送访问令牌，下面是一个例子：

* + HTTP/1.1 200 OK
  + Content-Type: application/json;charset=UTF-8
  + Cache-Control: no-store
  + Pragma: no-cache
  + {
  + "access\_token":"2YotnFZFEjr1zCsicMWpAA",
  + "token\_type":"example",
  + "expires\_in":3600,
  + "refresh\_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  + "example\_parameter":"example\_value"
  + }

**客户端模式**

* 客户端模式指客户端以自己名义，而不是用户名义，向“服务提供商”进行认证，严格地说，客户端模式不属于OAuth框架要解决的问题，在这种模式中，用户直接向客户端注册，客户端以自己名义要求“服务提供商”提供服务



bg2014051207.png

* 它的步骤如下：  
  -（A）:客户端向认证服务器进行身份认证，并要求一个访问令牌  
  -（B）:认证服务器确认无误后，向客户端提供访问令牌
* 上述步骤中所需要的参数：
  + A步骤中，客户端发出HTTP请求，包含以下参数：
    - granttype：表示授权类型，此处固定值为“clientcredentials”，必选
    - scope：表示权限范围，可选

 POST /token HTTP/1.1

Host: server.example.com

Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

Content-Type: application/x-www-form-urlencoded

grant\_type=client\_credentials

 B步骤中，认证服务器向客户端发送访问令牌，下面是一个例子

* + HTTP/1.1 200 OK
  + Content-Type: application/json;charset=UTF-8
  + Cache-Control: no-store
  + Pragma: no-cache
  + {
  + "access\_token":"2YotnFZFEjr1zCsicMWpAA",
  + "token\_type":"example",
  + "expires\_in":3600,
  + "example\_parameter":"example\_value"
  + }

**更新令牌**

* 如果用户访问的时候，客户端“访问令牌”已经过期，则需要使用“更新令牌”申请一个新的令牌
* 客户端发出更新令牌请求，包含以下参数：
  + granttype：表示授权模式，此处固定值为“refreshtoken”，必选
  + refresh\_token：表示早前收到的更新令牌，必选
  + scope：表示申请权限范围，不得超出上一次申请的范围，若省略该参数，则表示与上一次一样

POST /token HTTP/1.1

Host: server.example.com

Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

Content-Type: application/x-www-form-urlencoded

grant\_type=refresh\_token&refresh\_token=tGzv3JOkF0XG5Qx2TlKWIA

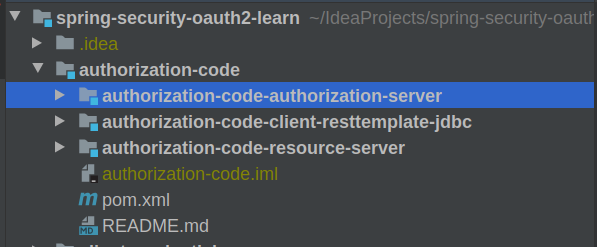
# 授权码模式 (Authorization Code) sec案例

## 授权码模式流程

授权码模式要求：用户登录并对第三方应用（客户端）进行授权，出示授权码交给客户端，客户端凭授权码换取access\_token（访问凭证）

此模式要求授权服务器与用户直接交互，在此过程中，第三方应用是无法获取到用户输入的密码等信息的，这个模式也是OAuth 2.0中最安全的一个

## Demo基本结构



这里主要关注authorization-code-authorization-server与authorization-code-resource-server这两个模块



authorization-code-client-resttemplate-jdbc这个项目是用来测试非OAuth2服务使用RestTemplate与JdbcTemplate对接OAuth2授权服务的，流程这里不讲，有兴趣可以debug看看，可能会让您对整个流程会有更清晰的感受

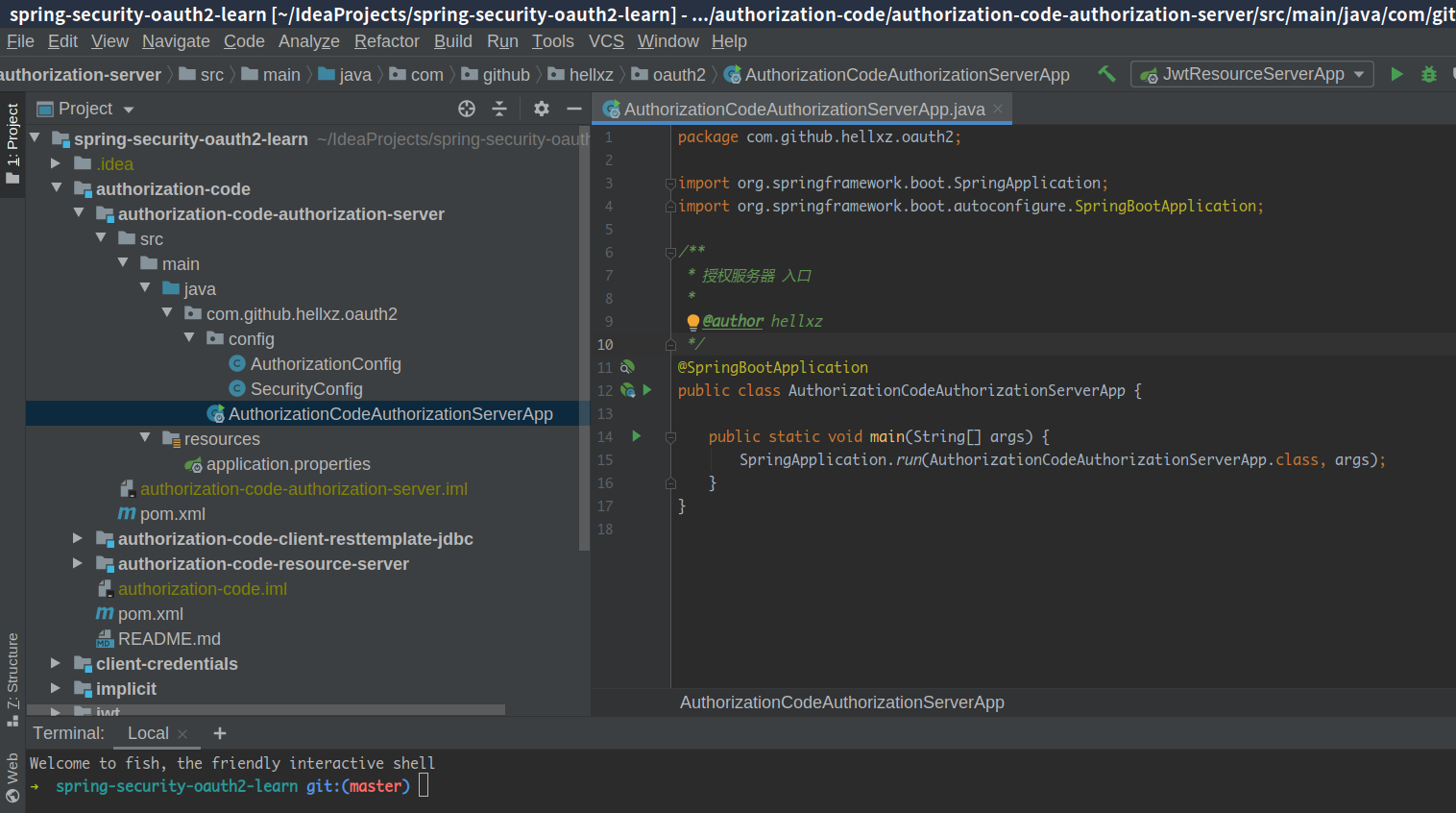
## Maven依赖

|  |
| --- |
| <!--Spring Security-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-security</artifactId>  </dependency>  <!--Spring Boot Starter Web 所有demo均使用web-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <!-- Spring Security OAuth2 -->  <dependency>  <groupId>org.springframework.security.oauth</groupId>  <artifactId>spring-security-oauth2</artifactId>  <version>${spring-security-oauth2.version}</version>  </dependency> |

## 搭建授权服务器（Authorization Server）

文中服务器均使用demo级别配置，请勿直接使用到生产环境

授权服务器结构主体如下：



通过@Configuration 和@EnableWebSecurity开启Spring Security配置，继承WebSecurityConfigurerAdapter的方法，实现个性化配置，这里我们使用内存保存一个名为hellxz、密码为xyz的用户，与授权服务器交互的用户就是他了

除了配置用户，我们需要对服务的资源进行保护，这里将所有的请求都要求通过认证才可以访问，用户登录需要使用httpBasic形式（就是那种网页弹个窗要求登录的那种😄）

Spring Security 5.x版本后，要求显示声明使用的密码器，就是PasswordEncoder了，常用BCryptPasswordEncoder，简单的可以认为它是使用时间戳和盐进行加密的一种算法，同一个密码被加密后也不会相同

接着看看授权服务器的配置

1.通过@Configuration 和EnableAuthorizationServer开启授权服务器配置，通过重写AuthorizationServerConfigurerAdapter的方法来完成自定义授权服务器

2.OAuth2授权码模式中，要求不仅仅用户需要登录，还要求客户端也需要登录，这里就需要在configure(ClientDetailsServiceConfigurer clients)这个方法中配置客户端（第三方应用）的登录信息，

* withClient中配置的是客户端id（client\_id）
* secret为客户端的密码，要求使用加密器进行加密
* 授权码的authorizedGrantTypes必须配置有"authorization\_code"（授权码模式），这里是可以同时支持多种授权模式的，为了简单只写一个
* scopes，请求资源作用域，用于限制客户端与用户无法访问没有作用域的资源
* resourceIds，可选，资源id，可以对应一个资源服务器，个人理解为某个资源服务器的所有资源标识
* redirectUris，回调地址，有两个作用：1.回调客户端地址，返回授权码； 2.校验是否是同一个客户端

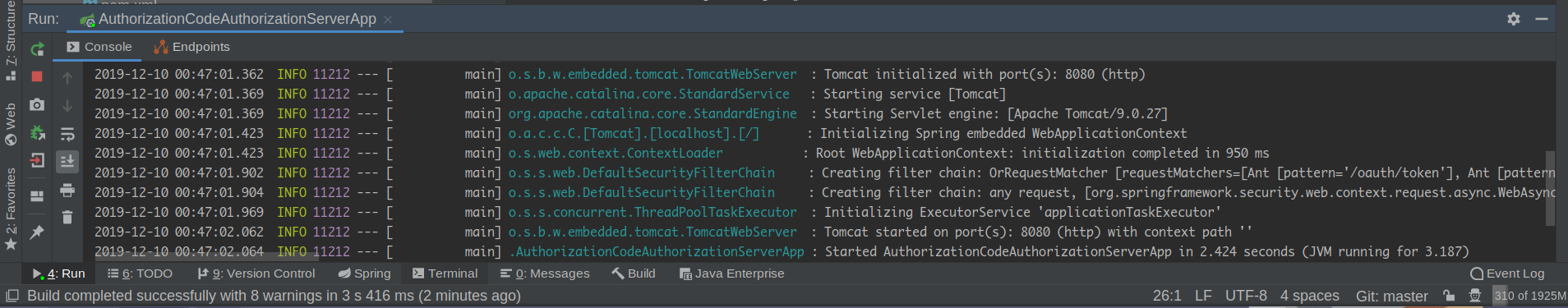
redirectUris校验是否同一个客户端这个，可能说的不是很准确，说下大体流程，我们在授权服务器上配置了这个回调地址，授权服务器在用户授权成功后，返回授权码的地址就是它，另外我们后续申请token时，也需要传递这个回调地址，所以我的理解是校验是否是同一客户端发来的第二次请求（换token时）

3.configure(AuthorizationServerSecurityConfigurer security)这里配置资源客户端（第三方应用）的表单提交权限，类似Spring Security配置的permitAll()等权限控制标识，如果不配置，客户端将无法换取token

4.application.properties

这里我只配置了server.port=8080

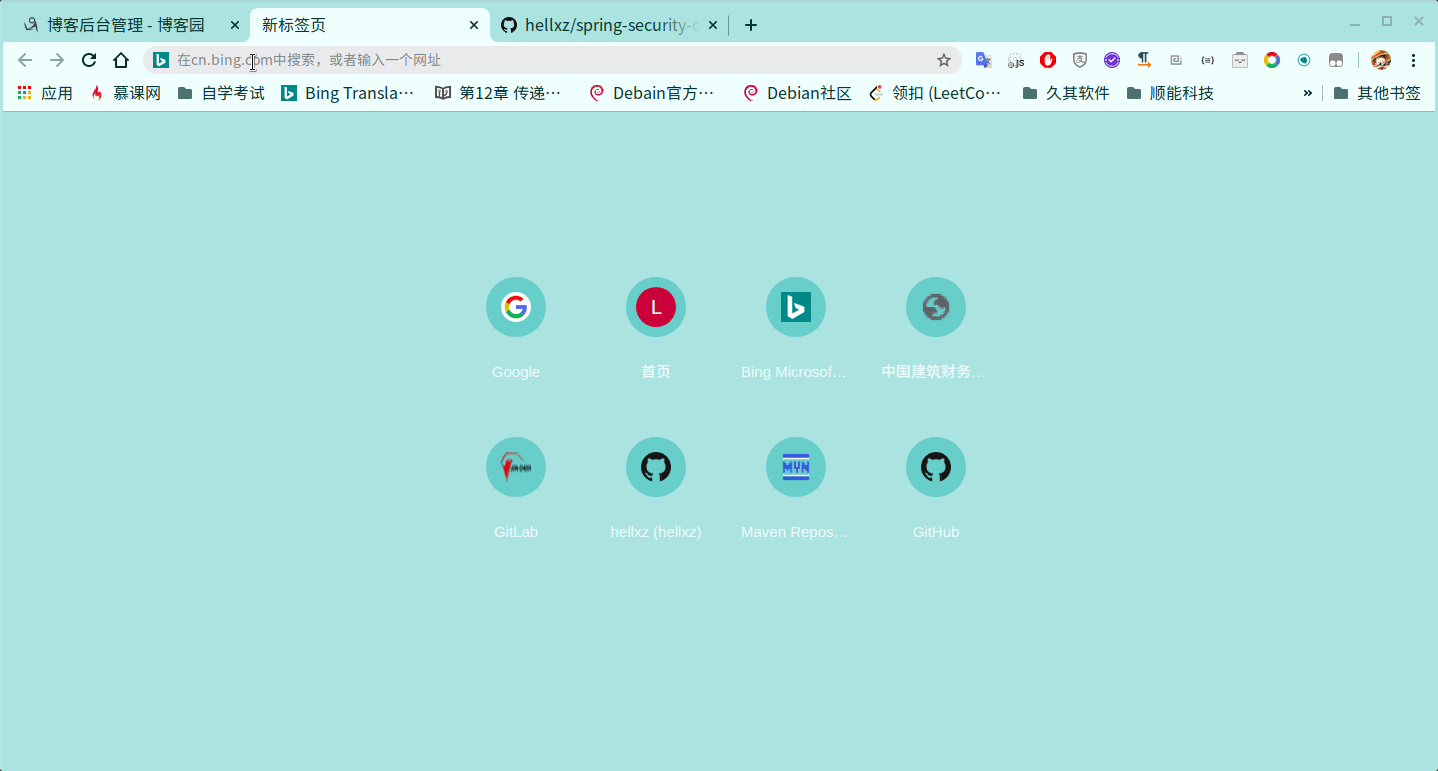
这样我们就配置了相当简易的授权服务器，**启动测试**下



**获取授权码的流程**，**一般是由客户端使用自己的client\_id与密码+response\_type=code拼接url，让浏览器跳转完成的，用户的登录与授权过程都需要在浏览器中完成**，启动项目后访问下列url

<http://localhost:8080/oauth/authorize?client_id=client-a&client_secret=client-a-secret&response_type=code>

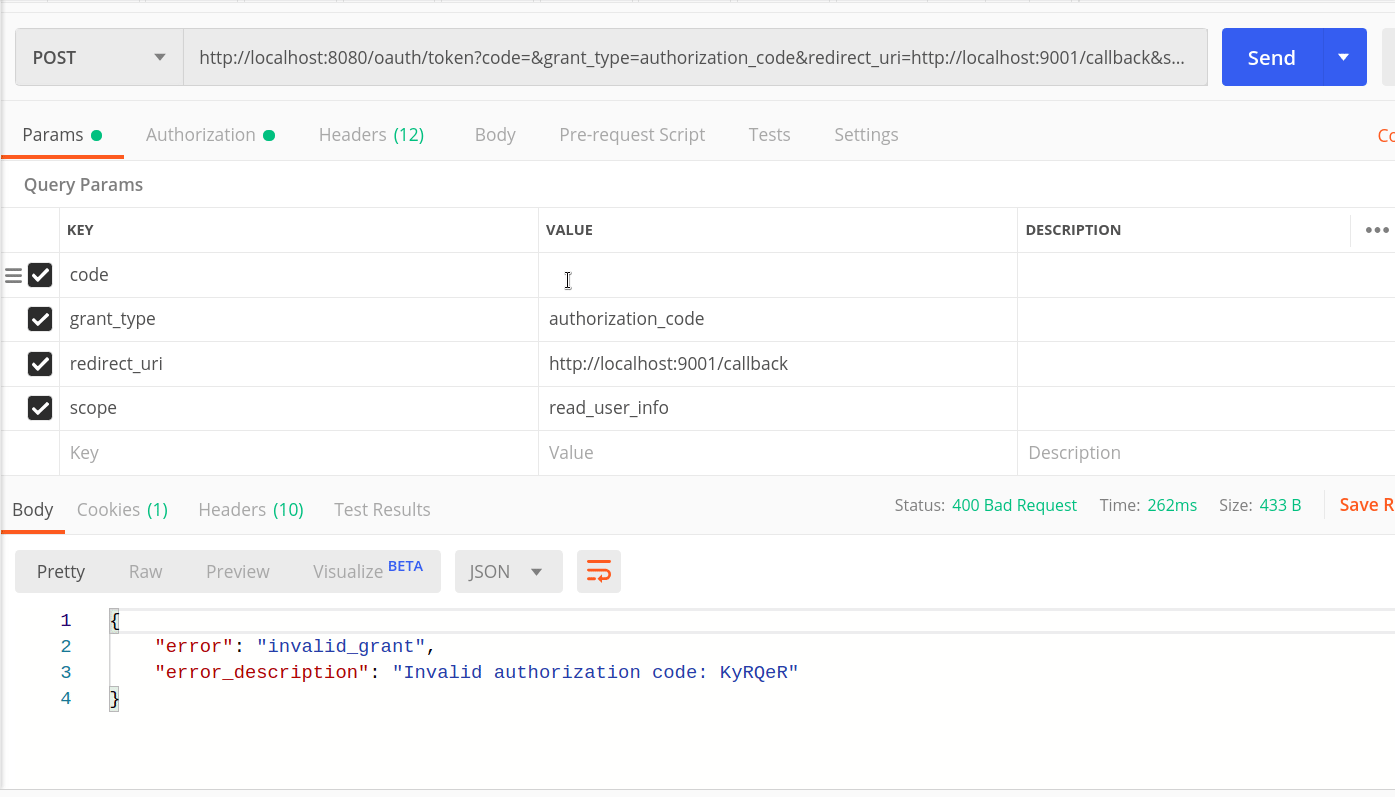
登录用户/密码： hellxz/xyz ，选择Approve表示接受授权，Deny反之，如下动图所示



最后我们得到了回调地址http://localhost:9001/callback?code=2e6450

这里的**code就是授权码**，接下来我们**使用授权码进行换取token**

POST请求，[http://localhost:8080/oauth/token，参数如图](http://localhost:8080/oauth/token%EF%BC%8C%E5%8F%82%E6%95%B0%E5%A6%82%E5%9B%BE)



BasicAuth：这里填的是客户端配置的client\_id和client\_secret的值，相当于curl --user client\_id:client\_secret，配置后会在Header中添加Authorization:Basic Y2xpZW50LWE6Y2xpZW50LWEtc2VjcmV0，Basic空格 后的是client\_id:client\_secret具体值被Base64后得到的值

请求参数列表：

* code=授权码
* grant\_type=authorization\_code
* redirect\_uri=回调url ，要与配置处和获取授权码处相同
* scope=作用域

最后我们获得了授权服务的响应，包含token的json

{

"access\_token": "99435e13-f9fe-438a-a94e-3b00d549b329", //访问token

"token\_type": "bearer", //token类型，使用时需要拼接在token前并在token前加空格

"expires\_in": 43199, //过期时间

"scope": "read\_user\_info" //作用域

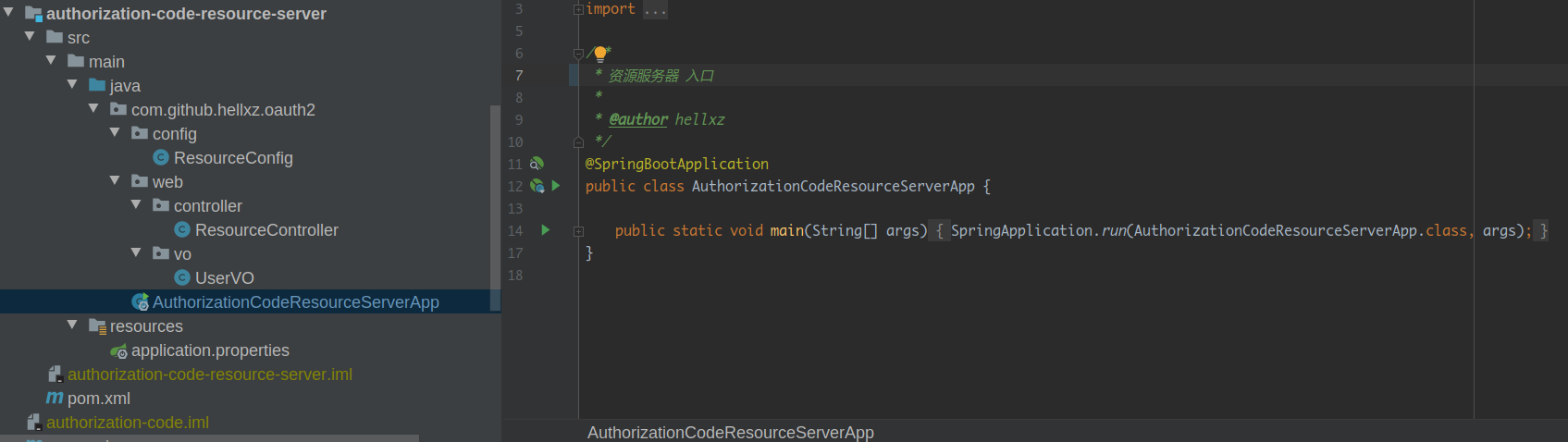
}

在access\_token未过期之前，同一个用户名使用同一个客户端访问都会是同一个access\_token

授权服务器先放在这里，不要关服，接下来搭建资源服务器

## 搭建资源服务器（Resource Server）

资源服务器结构



入口类不多说，先搭建资源服务器主要配置，这里直接使用ResourceConfig进行配置

1.通过@Configuration 和@EnableResourceServer这两个注解标识服务是一个资源服务器，重写ResourceServerConfigurerAdapter来实现自定义授权服务器

2.配置configure(HttpSecurity http)方法，这里可以代替Spring Security同名方法配置，开启所有请求需要授权才可访问

3.配置资源相关设置configure(ResourceServerSecurityConfigurer resources)，这里只设置resourceId

后续的使用redis校验token也在这里设置

4.校验token的配置，这里使用了远程调用授权服务器帮忙校验token的方式，只需要显示注入RemoteTokenServices remoteTokenServices()的Bean，就可以调用授权服务器的/oauth/check\_token端点，设置客户端配置的值，详见注释

这样一来我们就配置好了资源服务器，当然光有配置是不够的，我们搞一个资源接口做测试用

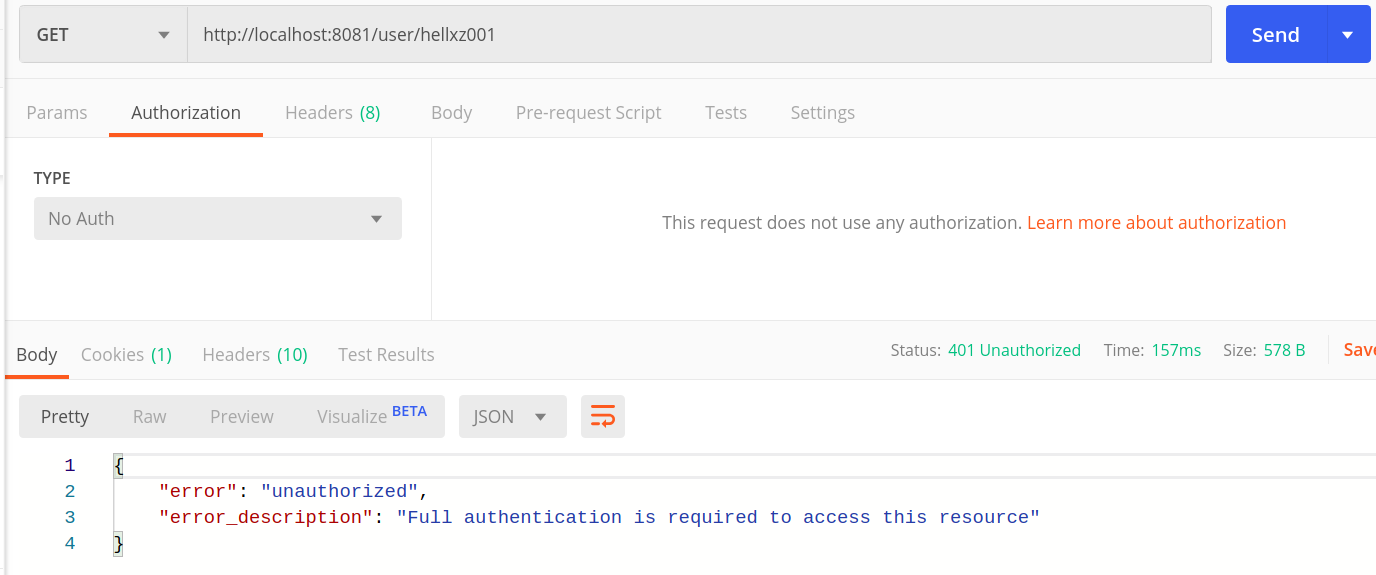
上边的ResourceController与UserVO都比较简单，传入一个名称，返回用户对象，包含用户名和邮箱信息

application.properties中配置了与授权服务器不同的端口：8081

server.port=8081

**启动资源服务测试**

什么也不传，直接访问接口，提示资源需要授权



复制之前获取到的token，添加token访问接口http://localhost:8081/user/hellxz001



Bearer Token相当于在Headers中添加Authorization：Bearer空格access\_token

至此我们成功的搭建并测试了授权码模式下的最简单的授权服务与资源服务分离的demo