第一章 OAuth2 协议

1.1 OAuth2 协议简介

1.1.1 OAuth2 是什么

OAuth 2.0参考：https://oauth.net/2/

中文版参考：https://github.com/jeansfish/RFC6749.zh-cn/blob/master/SUMMARY.md

3.3. Access Token Scope 23
4. Obtaining Authorization 23
4.1. Authorization Code Grant 24
4.1.1. Authorization Request 25
4.1.2. Authorization Response 26
4.1.3. Access Token Request 29
4.1.4. Access Token Response 30
4.2. Implicit Grant 31
4.2.1. Authorization Request 33
4.2.2. Access Token Response 35
4.3. Resource Owner Password Credentials Grant 37
4.3.1. Authorization Request and Response 39
4.3.2. Access Token Request 39
4.3.3. Access Token Response 40
4.4. Client Credentials Grant 40
4.4.1. Authorization Request and Response 41
4.4.2. Access Token Request 41
4.4.3. Access Token Response 42
4.5. Extension Grants 42

OAuth 2.0 是目前最流行的授权机制，用来授权第三方应用，获取用户数据。

OAuth 协议为用户资源的授权提供了一个安全的、开放而又简易的**规范标准**。与以往的授权方式不同之处是 OAuth 的授权不会使第三方触及到用户的帐号信息（如用户名与密码），即第三方无需使用用户的用户名与密码就可以申请获得该用户资源的授权，因此 OAuth 是开放的安全的。

业界提供了 OAuth 的多种实现，如 Java、PHP、Ruby 等各种语言开发包，大大节约了程序员的时间，因而OAuth 是简易的。很多大公司如 阿里、腾讯、 Google，Yahoo，Microsoft等都提供了 OAuth 认证服务，这些都足以说明 OAuth 标准逐渐成为开放资源授权的标准。

OAuth 协议1.0版本过于复杂，目前发展到2.0版本，2.0版本已得到广泛应用。

1.1.2 OAuth2 要解决的问题

比如：我们的梦学谷官网，学员登录梦学谷官网时，不想注册帐号，而是希望通过微信进行登录。

如果微信用户同意提供帐号和密码去获取微信信息，进行登录梦学谷，这样存在很大的安全问题：

1. 提供账号和密码给第三方应用程序后，应用可以访问用户在微信上的所有数据（有什么群，好友）。

仅供购买者学习，禁止盗版、转卖、传播课程

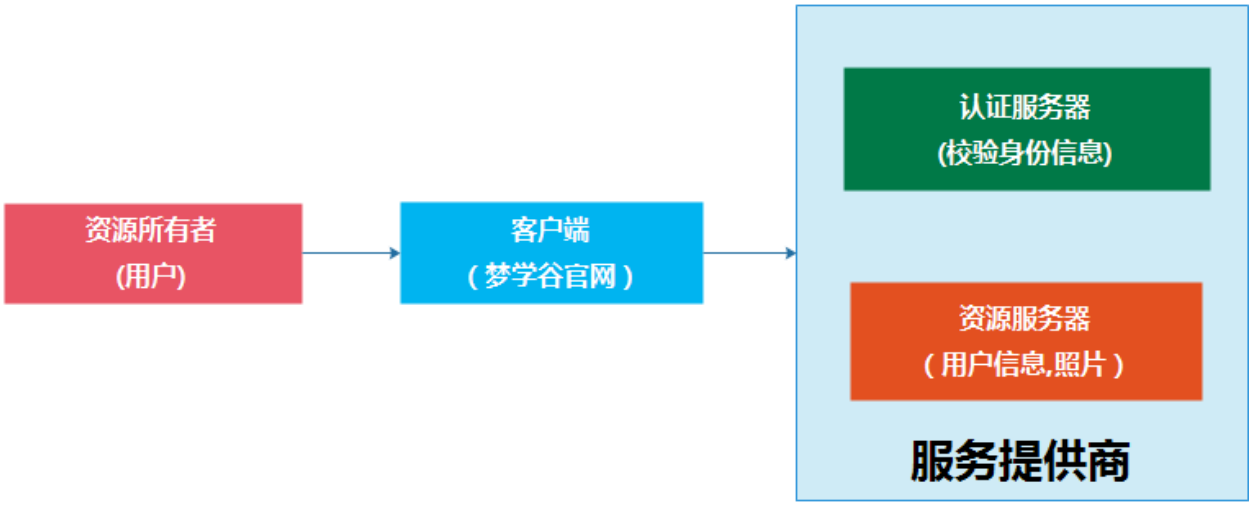
2. 用户只有修改密码后，才可以收回权限。但是如果一样将用户名和密码授权给了其他第三方应用，当你修改密码后，那么所有第三方应用程序都会被收回权限。
3. 密码泄露的可能性大大提高。因为无法控制接入应用对用户数据保护的安全系数，只要有一个接入的第三方应用遭到破解，那么用户的密码就会泄露，意味着用户的信息全部会被泄露，后果不堪设想。

OAuth 就是为了解决上面这些问题而诞生的。OAuth 协议规范了授权方式，不采用授权帐号和密码的方式，而是使用令牌方式（Token）来解决授权问题，应用通过令牌去获取被授权的用户信息，从而可以避免上面存在的安全问题。

1.1.3 OAuth2 涉及的角色

- **资源所有者（Resource Owner）：** 通常为 "用户"（user），如昵称、头像等这些资源的拥有者（用户只是将这些资源放到了服务提供商的资源服务器中）。
- **第三方应用（Third-party application）：** 又称为**客户端**（Client），比如 梦学谷官网想要使用微信的资源（昵称、头像等），梦学谷官网对于QQ、微信等系统来说是第三者，我们称梦学谷官网为第三方应用。
- **认证服务器（Authorization server）：** 专门用来对资源所有者的身份进行认证、对要访问的资源进行授权、产生令牌的服务器。想访问资源，需要通过认证服务器由资源所有者授权后才可访问。
- **资源服务器（Resource server）：** 存储用户的资源（昵称、头像等）、验证令牌有效性。比如: 微信的资源服务器存储了微信的用户信息，淘宝的资源服务器存储了淘宝的用户信息等。

  注意：*认证服务器* 和 *资源服务器* 虽然是两个解决，但其实他们可以是同一台服务器、同一个应用。
- **服务提供商（Service Provider）：** 如 QQ、微信等 （包含认证和资源服务器）。

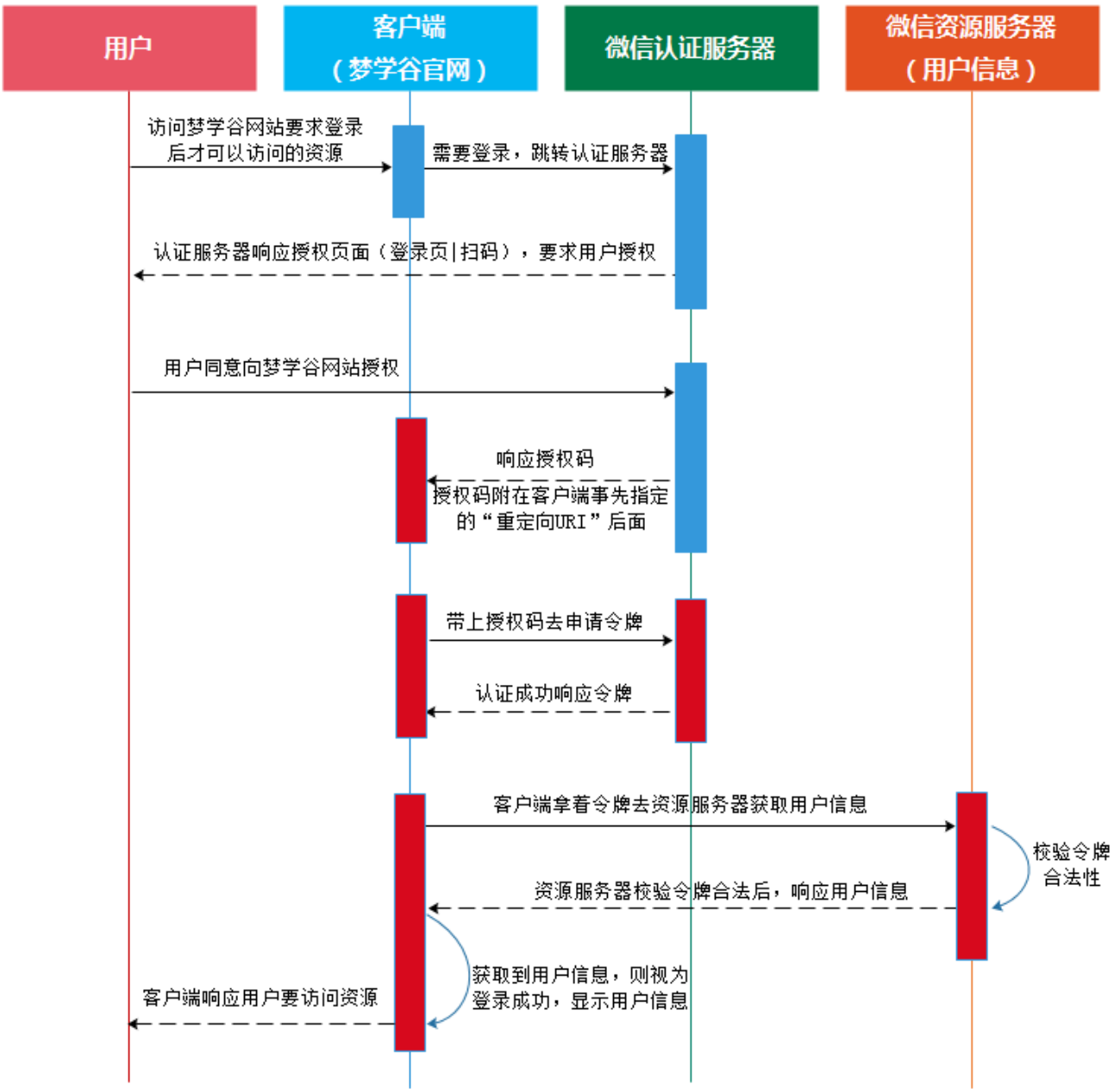


1.1.2 OAuth2 认证流程

OAuth 在 "第三方应用" 与 "服务提供商" 之间，设置了一个授权层（authorization layer）。"第三方应用" 不能直接登录 "服务提供商"，只能通过授权层将 "第三方应用" 与 用户 区分开来。"第三方应用" 通过授权层获取令牌（token），获取令牌后拿令牌去访问服务提供商。令牌和用户的密码不同，可以指定授权层令牌的权限范围和有效期，"服务提供商" 根据令牌的权限范围和有效期，向 "第三方应用" 开放用户对应的资源。

梦学谷官网使用微信认证流程：

仅供购买者学习，禁止盗版、转卖、传播课程



1.2 OAuth2 协议的授权模式

1.2.1 授权方式

授权码模式(Authorization Code)：功能最完整，流程最严密的授权模式。国内各大服务提供商（微信、QQ、微博、淘宝 、百度）都采用此模式进行授权。可以确定是用户真正同意授权；而且令牌是认证服务器发放给第三方应用的服务器，而不是浏览器上。

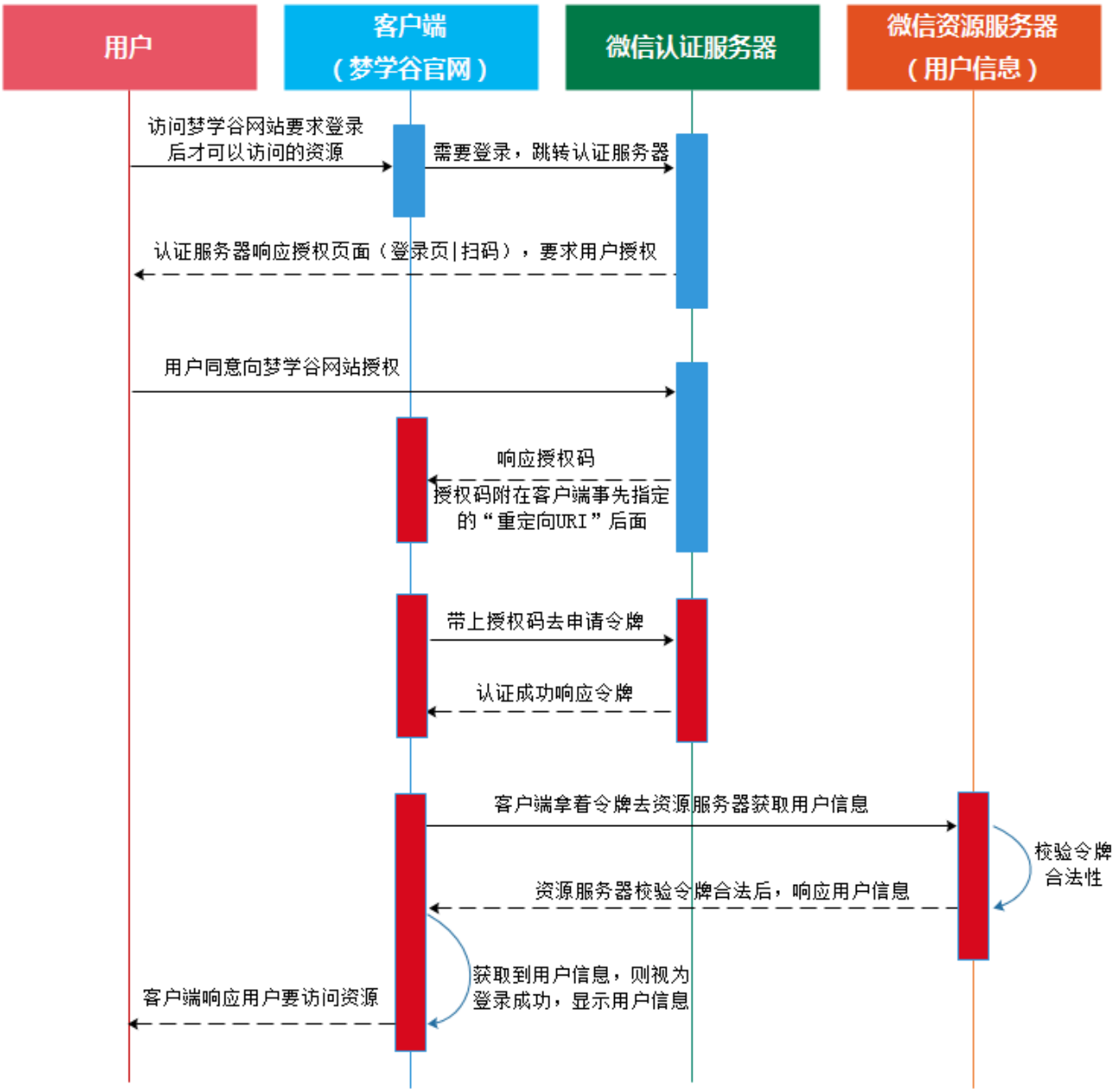
简化模式(Implicit): 令牌是发放给浏览器的，oauth客户端运行在浏览器中 ，通过JS脚本去申请令牌。而不是发放给第三方应用的服务器。

密码模式(Resource Owner Password Credentials)：将用户名和密码传过去，直接获取 access_token 。用户同意授权动作是在第三方应用上完成 ，而不是在认证服务器上。第三方应用申请令牌时，直接带着用户名密码去向认证服务器申请令牌。这种方式认证服务器无法断定用户是否真的授权了，用户名密码可能是第三方应用盗取来的。

仅供购买者学习，禁止盗版、转卖、传播课程

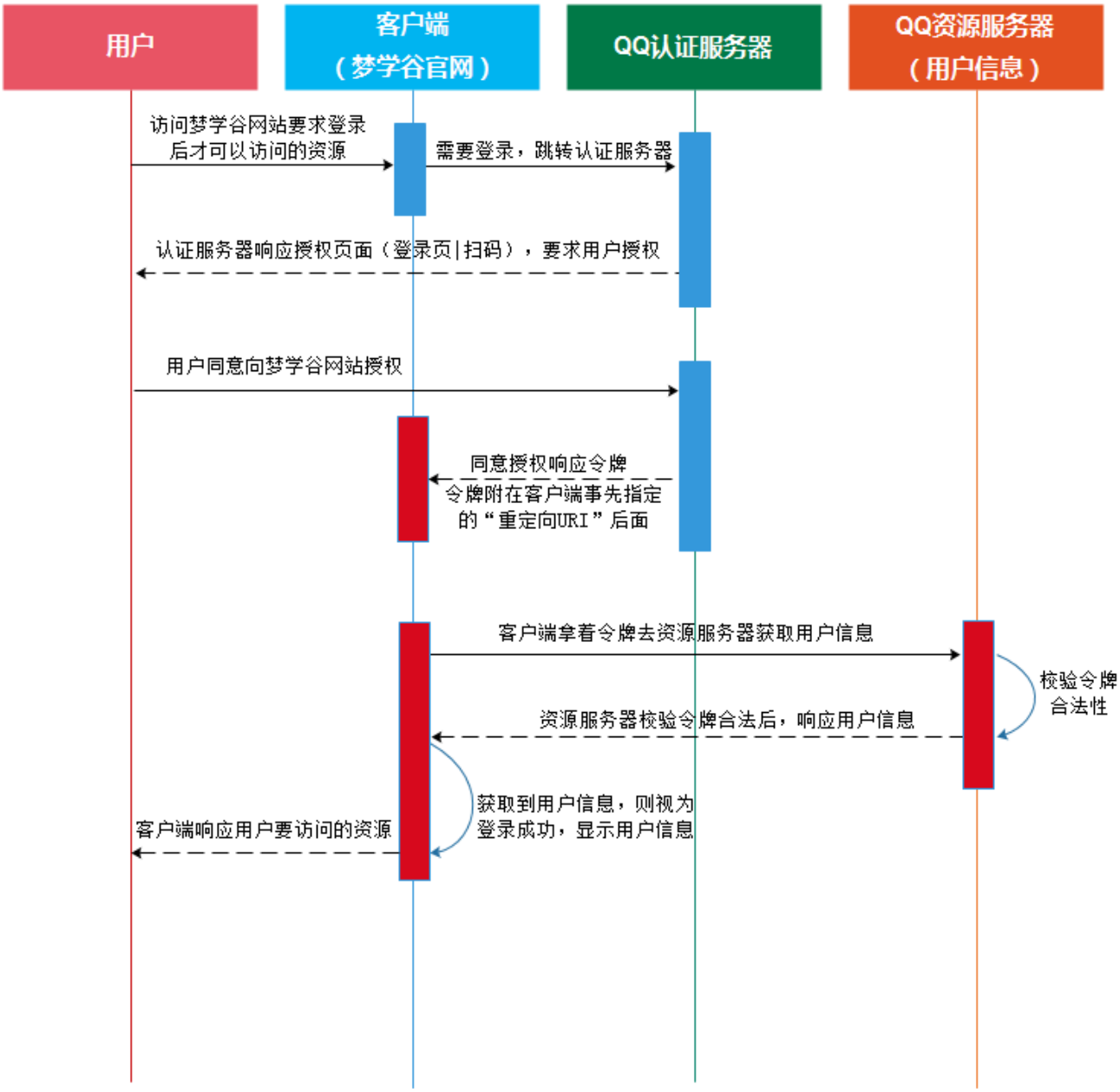
客户端证书模式(Client credentials)：用得少。当一个第三应用自己本身需要获取资源（而不是以用户的名义），而不是获取用户的资源时，客户端模式十分有用。

1.2.2 授权码模式流程



1.2.3 简化模式流程

仅供购买者学习，禁止盗版、转卖、传播课程



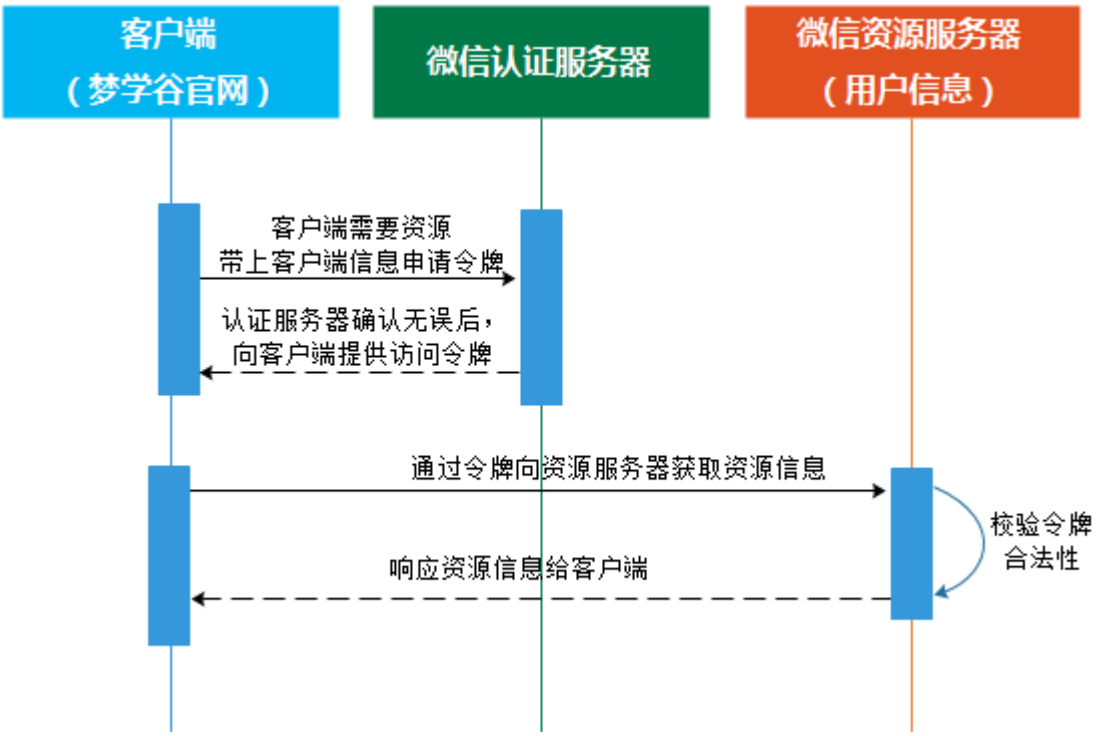
1.2.4 密码模式流程

1. 用户向客户端直接提供认证服务器平台的用户名和密码。
2. 客户端将用户名和密码发给认证服务器，向后者请求令牌。
3. 认证服务器确认无误后，向客户端提供访问令牌。

1.2.5 客户端模式流程

1. 客户端向认证服务器进行身份认证，并要求一个访问令牌。
2. 认证服务器确认无误后，向客户端提供访问令牌。

仅供购买者学习，禁止盗版、转卖、传播课程



第二章 Spring Security OAuth2 认证服务器

概述

Spring Seucrity 登录信息存在Session中，每次访问服务时，会去查看浏览器中的 Cookie 中是否存在JSESSIONID, 如果不存在会新建一个Session, 将新建的SessionID 保存到Cookie 中。每次发请求会通过浏览器的SessionID查找到对应的Session对象，从而获取用户信息。

前后端分离，前端部署在单独Web服务器，后台部署在另外一台应用服务器，浏览器先访问Web服务器，Web服务器再发送请求到应用服务器中。这样采用Cookie存储就不太合适，原因：

1. 开发复杂
2. 安全性差
3. 客户体验差
4. 有些前端技术 不支持Cookie, 如：小程序

解决：

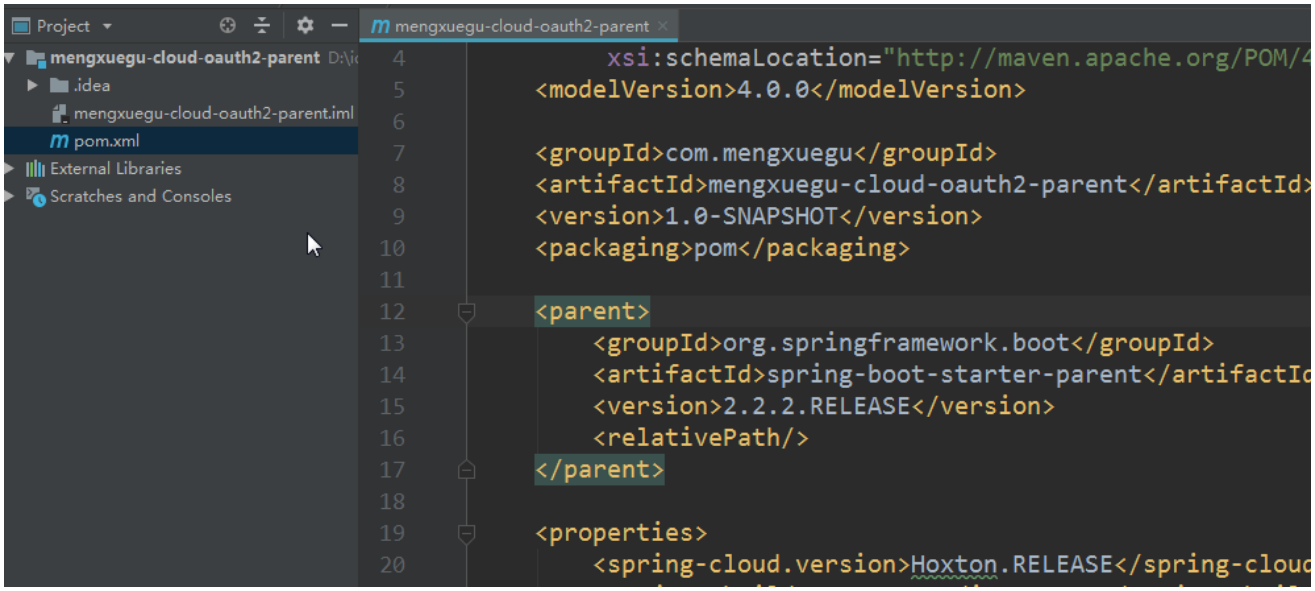
采用令牌方式进行认证就可以上面的问题，就可以使用 OAuth2 协议标准中的实现 Spring Security OAuth2 解决。

2.1 创建统一管理的父工程

2.1.1 创建 Project

`mengxuegu-cloud-oauth2-parent` 统一管理版本号

仅供购买者学习，禁止盗版、转卖、传播课程



2.1.2 pom.xml

类型为 pom, `<packaging>pom</packaging>`

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
   4.0.0.xsd">
5      <modelVersion>4.0.0</modelVersion>
6
7      <groupId>com.mengxuegu</groupId>
8      <artifactId>mengxuegu-cloud-oauth2-parent</artifactId>
9      <version>1.0-SNAPSHOT</version>
10     <packaging>pom</packaging>
11
12     <parent>
13         <groupId>org.springframework.boot</groupId>
14         <artifactId>spring-boot-starter-parent</artifactId>
15         <version>2.2.2.RELEASE</version>
16         <relativePath/>
17     </parent>
18
19     <properties>
20         <spring-cloud.version>Hoxton.RELEASE</spring-cloud.version>
21         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
22         <maven.compiler.source>1.8</maven.compiler.source>
23         <maven.compiler.target>1.8</maven.compiler.target>
24         <mybatis-plus.version>3.2.0</mybatis-plus.version>
25         <druid.version>1.1.12</druid.version>
26         <kaptcha.version>2.3.2</kaptcha.version>
27         <fastjson.version>1.2.8</fastjson.version>
28         <commons-lang.version>2.6</commons-lang.version>
29         <commons-collections.version>3.2.2</commons-collections.version>
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
30        <commons-io.version>2.6</commons-io.version>
31        <!-- 定义版本号, 子模块直接引用-->
32        <mengxuegu-security.version>1.0-SNAPSHOT</mengxuegu-security.version>
33    </properties>
34
35
36    <!--依赖声明-->
37    <dependencyManagement>
38        <dependencies>
39            <dependency>
40                <groupId>org.springframework.cloud</groupId>
41                <artifactId>spring-cloud-dependencies</artifactId>
42                <version>${spring-cloud.version}</version>
43                <type>pom</type>
44                <!--maven不支持多继承，使用import来依赖管理配置-->
45                <scope>import</scope>
46            </dependency>
47            <!--mybatis-plus启动器-->
48            <dependency>
49                <groupId>com.baomidou</groupId>
50                <artifactId>mybatis-plus-boot-starter</artifactId>
51                <version>${mybatis-plus.version}</version>
52            </dependency>
53            <!--druid连接池-->
54            <dependency>
55                <groupId>com.alibaba</groupId>
56                <artifactId>druid</artifactId>
57                <version>${druid.version}</version>
58            </dependency>
59            <!-- kaptcha 用于图形验证码 -->
60            <dependency>
61                <groupId>com.github.penggle</groupId>
62                <artifactId>kaptcha</artifactId>
63                <version>${kaptcha.version}</version>
64            </dependency>
65            <!-- 工具类依赖 -->
66            <dependency>
67                <groupId>com.alibaba</groupId>
68                <artifactId>fastjson</artifactId>
69                <version>${fastjson.version}</version>
70            </dependency>
71
72            <dependency>
73                <groupId>commons-lang</groupId>
74                <artifactId>commons-lang</artifactId>
75                <version>${commons-lang.version}</version>
76            </dependency>
77            <dependency>
78                <groupId>commons-collections</groupId>
79                <artifactId>commons-collections</artifactId>
80                <version>${commons-collections.version}</version>
81            </dependency>

82            <dependency>
```

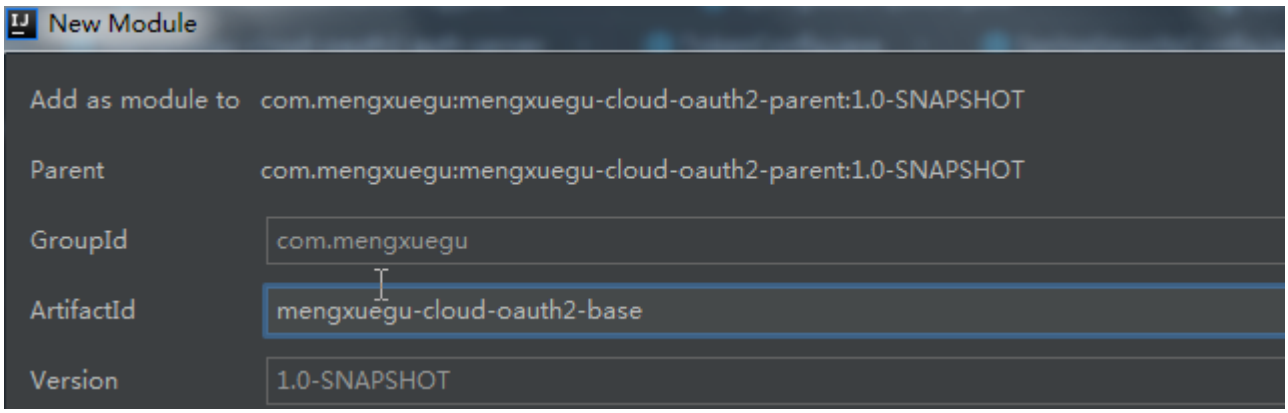
仅供购买者学习，禁止盗版、转卖、传播课程

```
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>${commons-io.version}</version>
        </dependency>
    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
        <!--springboot 打包插件-->
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```

2.2 创建基础模块

2.2.1 创建模块

与 mengxuegu-cloud-oauth2-base 一样



2.2.2 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
```

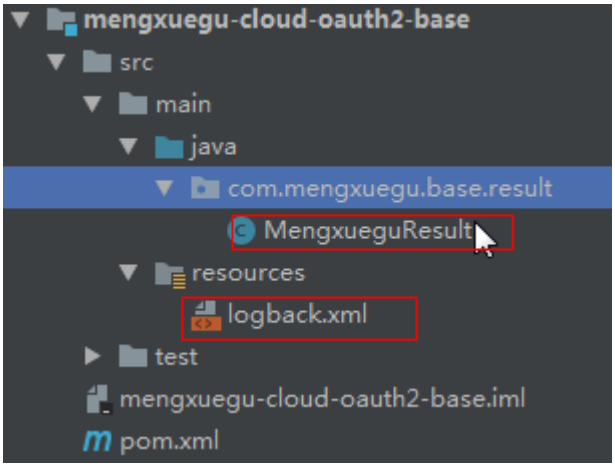
仅供购买者学习，禁止盗版、转卖、传播课程

```
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
  4.0.0.xsd">
5     <parent>
6         <artifactId>mengxuegu-cloud-oauth2-parent</artifactId>
7         <groupId>com.mengxuegu</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>mengxuegu-cloud-oauth2-base</artifactId>
13
14    <dependencies>
15        <!--类中setter/getter,使用注解-->
16        <dependency>
17            <groupId>org.projectlombok</groupId>
18            <artifactId>lombok</artifactId>
19        </dependency>
20        <!-- 工具类依赖 -->
21        <dependency>
22            <groupId>com.alibaba</groupId>
23            <artifactId>fastjson</artifactId>
24        </dependency>
25        <dependency>
26            <groupId>commons-lang</groupId>
27            <artifactId>commons-lang</artifactId>
28        </dependency>
29        <dependency>
30            <groupId>commons-collections</groupId>
31            <artifactId>commons-collections</artifactId>
32        </dependency>
33        <dependency>
34            <groupId>commons-io</groupId>
35            <artifactId>commons-io</artifactId>
36        </dependency>
37    </dependencies>
38 </project>
```

2.2.3 添加工具类与日志配置

将 mengxuegu-cloud-oauth2-base 的 MengxueguResult 类和 logback.xml 拷贝到对应位置

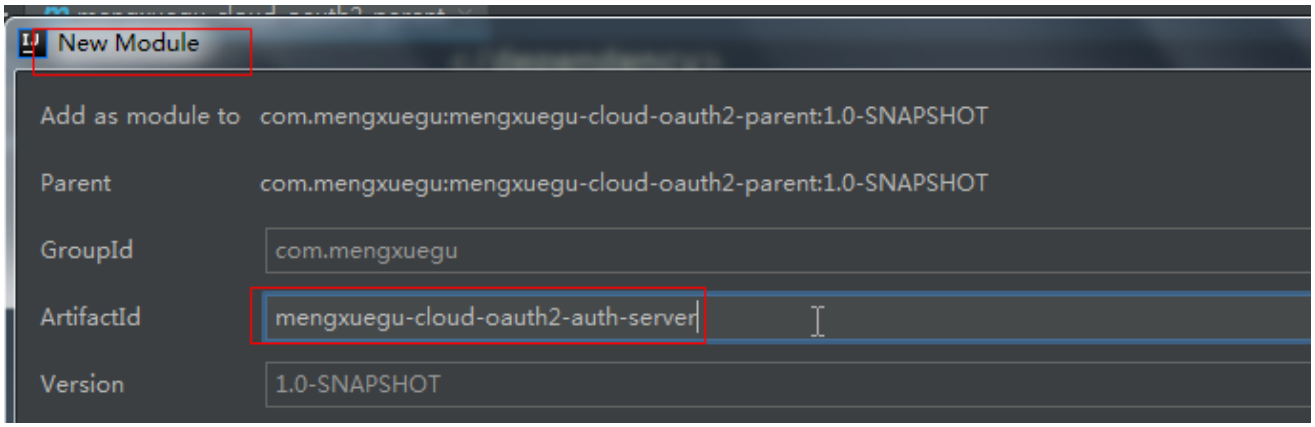
仅供购买者学习，禁止盗版、转卖、传播课程



2.3 创建认证服务器模块

2.3.1 创建模块

创建模块认证服务器 `mengxuegu-cloud-oauth2-auth-server`



2.3.2 添加依赖 pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>mengxuegu-cloud-oauth2-parent</artifactId>
        <groupId>com.mengxuegu</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mengxuegu-cloud-oauth2-auth-server</artifactId>

    <dependencies>
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
15      <dependency>
16          <groupId>com.mengxuegu</groupId>
17          <artifactId>mengxuegu-cloud-oauth2-base</artifactId>
18          <version>${mengxuegu-security.version}</version>
19      </dependency>
20      <!--spring mvc相关的-->
21      <dependency>
22          <groupId>org.springframework.boot</groupId>
23          <artifactId>spring-boot-starter-web</artifactId>
24      </dependency>
25      <!-- Spring Security、OAuth2 和JWT等 -->
26      <dependency>
27          <groupId>org.springframework.cloud</groupId>
28          <artifactId>spring-cloud-starter-oauth2</artifactId>
29      </dependency>
30
31      <!-- 注册到 Eureka
32      <dependency>
33          <groupId>org.springframework.cloud</groupId>
34          <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
35      </dependency>
36      -->
37      <!-- redis
38      <dependency>
39          <groupId>org.springframework.boot</groupId>
40          <artifactId>spring-boot-starter-data-redis</artifactId>
41      </dependency>
42       -->
43      <!--mybatis-plus启动器
44      <dependency>
45          <groupId>com.baomidou</groupId>
46          <artifactId>mybatis-plus-boot-starter</artifactId>
47      </dependency>
48      <dependency>
49          <groupId>org.springframework.boot</groupId>
50          <artifactId>spring-boot-starter-jdbc</artifactId>
51      </dependency>
52      -->
53      <!--druid连接池
54      <dependency>
55          <groupId>com.alibaba</groupId>
56          <artifactId>druid</artifactId>
57      </dependency>
58      <dependency>
59          <groupId>mysql</groupId>
60          <artifactId>mysql-connector-java</artifactId>
61      </dependency>
62      -->
63      <!-- thymeleaf 模块启动器
64      <dependency>
65          <groupId>org.springframework.boot</groupId>
66          <artifactId>spring-boot-starter-thymeleaf</artifactId>

67      </dependency>
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
        -->
        <!--对Thymeleaf添加Spring Security标签支持
         <dependency>
            <groupId>org.thymeleaf.extras</groupId>
            <artifactId>thymeleaf-extras-springsecurity5</artifactId>
         </dependency>
         -->

        <!-- springboot 单元测试 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>
        <!--热部署 ctrl+f9-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
        </dependency>

    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <!--启动类
                <configuration>
                    <mainClass>com.mengxuegu.oauth2.AuthServerApplication</mainClass>
                </configuration>
                -->
            </plugin>
        </plugins>
    </build>
</project>
```

2.3.3 配置 application.yml

注意：配置了项目的上下文路径 `/auth` ，请求时要加上，即 http://localhost:8090/auth 作为请求前缀

```yml
server:
  port: 8090
  servlet:
    context-path: /auth # 上下文路径，请求时要加上，后面网关时有用
```

2.3.4 创建启动类

创建认证服务器启动类 `com.mengxuegu.oauth2.AuthServerApplication`

仅供购买者学习，禁止盗版、转卖、传播课程

```java
package com.mengxuegu.oauth2;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * 认证服务器启动类
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@SpringBootApplication
public class AuthServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(AuthServerApplication.class, args);
    }
}
```

2.4 配置认证服务器-授权码模式

2.4.1 创建认证服务器配置类

作用：

1. 配置被允许访问此认证服务器的客户端信息, 没有在此配置的客户端是不允许访问的
2. 管理令牌:
    - 配置令牌管理策略(如：JDBC/ Redis/JWT)
    - 配置令牌生成策略
    - 配置令牌端点
    - 令牌端点的安全配置

步骤：

在 mengxuegu-oauth2-auth-server 模块创建认证配置类:

1. 创建 `com.mengxuegu.oauth2.server.config.AuthorizationServerConfig` 类继承 `AuthorizationServerConfigurerAdapter`
2. 在类上添加注解：
    - `@Configuration` 标识配置类
    - `@EnableAuthorizationServer` 开启 OAuth2 认证服务器功能，
3. 配置说明：
    - withClient：允许访问此认证服务器的客户端id , 如：PC、APP、小程序各不同的的客户端id。
    - secret：客户端密码，要加密存储，不然获取不到令牌一直要求登录,, 而且一定不能被泄露。
    - authorizedGrantTypes: 授权类型, 可同时支持多种授权类型：

      > 可配置："authorization_code", "password", "implicit","client_credentials","refresh_token"

    - scopes：授权范围标识，如指定微服务名称，则只能访问指定的微服务。
    - autoApprove：false 跳转到授权页面手动点击授权，true 不用手动授权，直接响应授权码。

仅供购买者学习，禁止盗版、转卖、传播课程

- redirectUris 当获取授权码后，认证服务器会重定向到这个URI，并且带着一个授权码 `code` 响应回来。

```
package com.mengxuegu.oauth2.server.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;

/**
 * 认证服务器配置
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@Configuration
@EnableAuthorizationServer // 开启认证服务器功能
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {

    @Autowired // 在 SpringSecurityBean 添加到容器了
    private PasswordEncoder passwordEncoder;

    /**
     * 配置被允许访问此认证服务器的客户端详情信息
     * 方式1：内存方式管理
     * 方式2：数据库管理
     */
    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
        // 使用内存方式
        clients.inMemory()
                .withClient("mengxuegu-pc") // 客户端id
                // 客户端密码，要加密,不然一直要求登录, 获取不到令牌, 而且一定不能被泄露
                .secret(passwordEncoder.encode("mengxuegu-secret"))
                // 资源id, 如商品资源
                .resourceIds("product-server")
                // 授权类型, 可同时支持多种授权类型
                .authorizedGrantTypes("authorization_code", "password", "implicit","client_credentials","refresh_token")
                // 授权范围标识，哪部分资源可访问（all是标识，不是代表所有）
                .scopes("all")
                // false 跳转到授权页面手动点击授权，true 不用手动授权，直接响应授权码，
                .autoApprove(false)
                .redirectUris("http://www.mengxuegu.com/"); // 客户端回调地址
    }

}
```

仅供购买者学习，禁止盗版、转卖、传播课程

2.4.2 统一管理Bean配置类

`PasswordEncoder` 加密方式因为多个地方要使用，所以单独提出来

创建 `com.mengxuegu.oauth2.server.config.SpringSecurityBean` 类，向容器中添加加密方式 BCrypt

```
package com.mengxuegu.oauth2.server.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@Configuration
public class SpringSecurityBean {

    @Bean // 加密方式
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

}
```

2.4.3 创建安全配置类

概要

指定要进行认证用户的用户名和密码，这个用户名和密码是资源所有者的。

和上面指定的客户端id和密码是不一样的，客户端的id和密码是应用系统的标识，每个应用系统就对应一个客户端ip和密码。

而这里指定的用户名和密码是客户的，就是每个应用系统的用户，即资源所有者。

步骤

在 `mengxuegu-cloud-oauth2-auth-server` 模块创建安全配置类:

1. 创建 `com.mengxuegu.oauth2.server.config.SpringSecurityConfig` 类继承 `WebSecurityConfigurerAdapter`
2. 在类上添加注解：
    - `@EnableWebSecurity` ，它包含了 `@Configuration` 注解所以不用加

```
package com.mengxuegu.oauth2.server.config;

import org.springframework.beans.factory.annotation.Autowired;

import
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
    org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
5   import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
6   import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
7   import org.springframework.security.crypto.password.PasswordEncoder;
8
9   /**
10   * 安全配置类
11   * @Auther: 梦学谷 www.mengxuegu.com
12   */
13  @EnableWebSecurity // 包含了@Configuration注解
14  public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
15
16      @Autowired // 在 SpringSecurityBean 添加到容器了
17      private PasswordEncoder passwordEncoder;
18
19      @Override
20      protected void configure(AuthenticationManagerBuilder auth) throws Exception {
21          // 内存方式存储用户信息
22          auth.inMemoryAuthentication().withUser("admin")
23                  .password(passwordEncoder.encode("1234")).authorities("product");
24      }
25
26  }
```

2.4.4 运行认证服务器

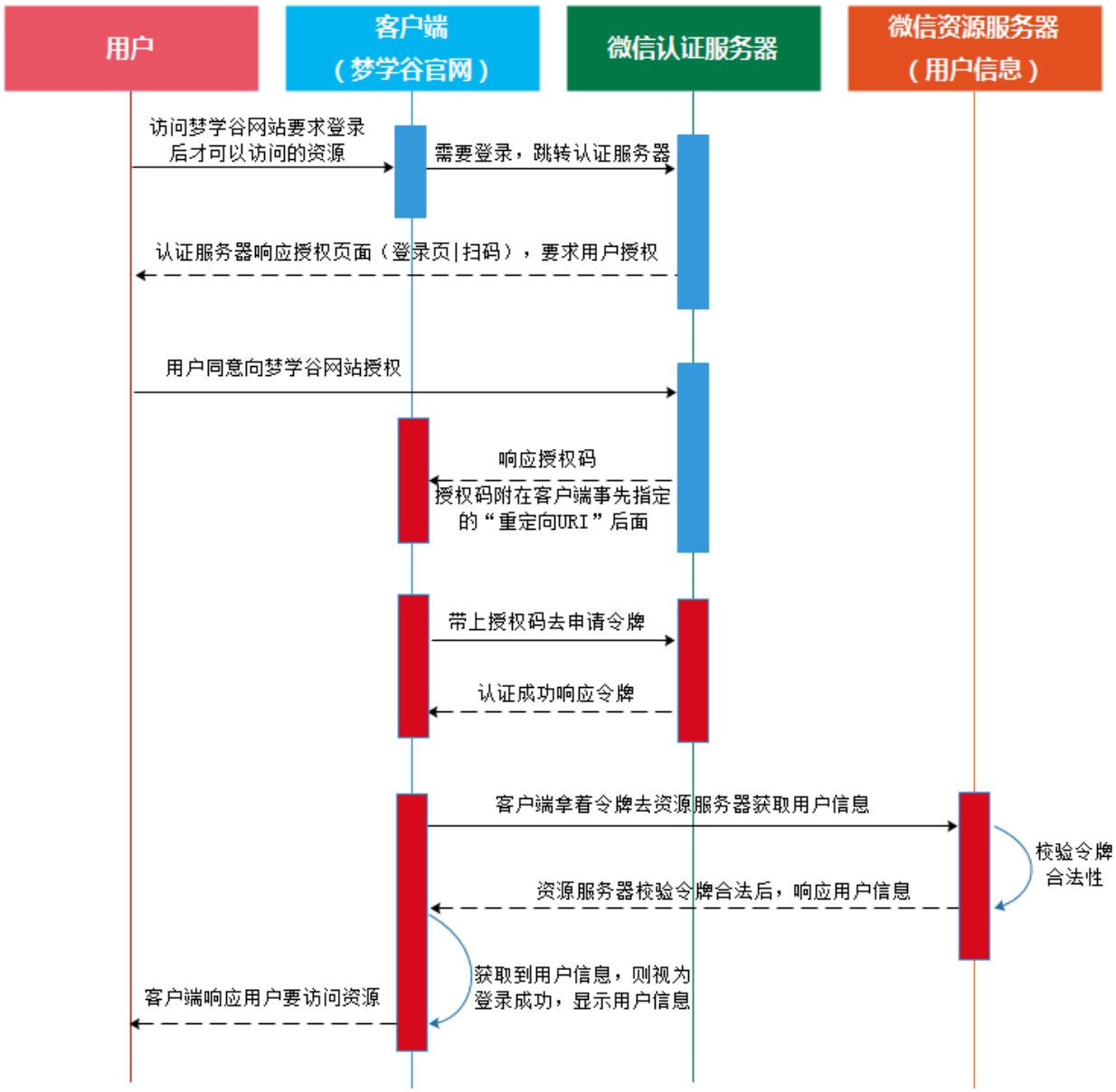
运行 com.mengxuegu.AuthServerApplication

2.4.5 令牌访问端点

Spring Security 对 OAuth2 默认提供了可直接访问端点，即URL：

- `/oauth/authorize`：申请授权码 code，涉及的类 `AuthorizationEndpoint`
- `/oauth/token`：获取令牌 token，涉及的类 `TokenEndpoint`
- `/oauth/check_token`：用于资源服务器请求端点来检查令牌是否有效，涉及的类 `CheckTokenEndpoint`
- `/oauth/confirm_access`：用户确认授权提交，涉及的类 `WhitelabelApprovalEndpoint`
- `/oauth/error`：授权服务错误信息，涉及的类 `WhitelabelErrorEndpoint`
- `/oauth/token_key`：提供公有密匙的端点，使用 JWT 令牌时会使用，涉及的类 `TokenKeyEndpoint`

仅供购买者学习，禁止盗版、转卖、传播课程



2.4.6 发送请求获取授权码 code

涉及类: `org.springframework.security.oauth2.provider.endpoint.AuthorizationEndpoint`

1. 请求如下地址申请授权码(注意：路径加上下文路径 `/auth`)

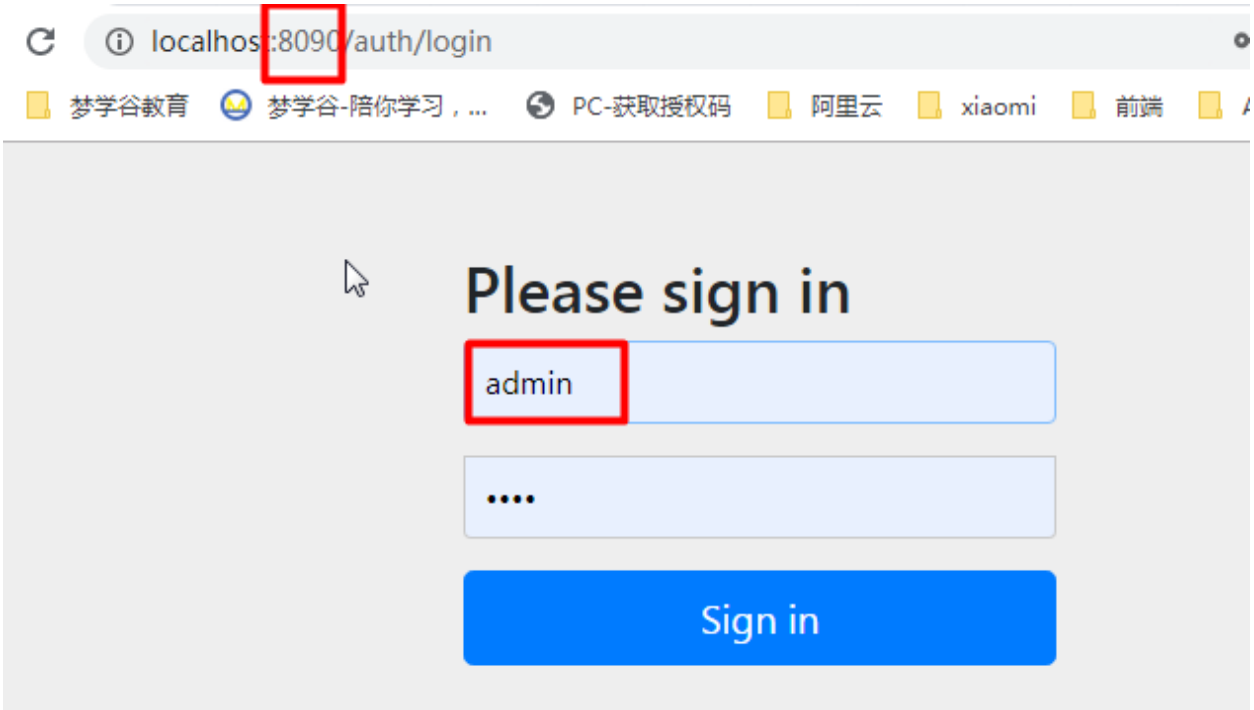
```
http://localhost:8090/auth/oauth/authorize?client_id=mengxuegu-pc&response_type=code
```

2. 当请求到达授权中心的 `AuthorizationEndpoint` 后，授权中心将会要求资源所有者做身份验证 ，

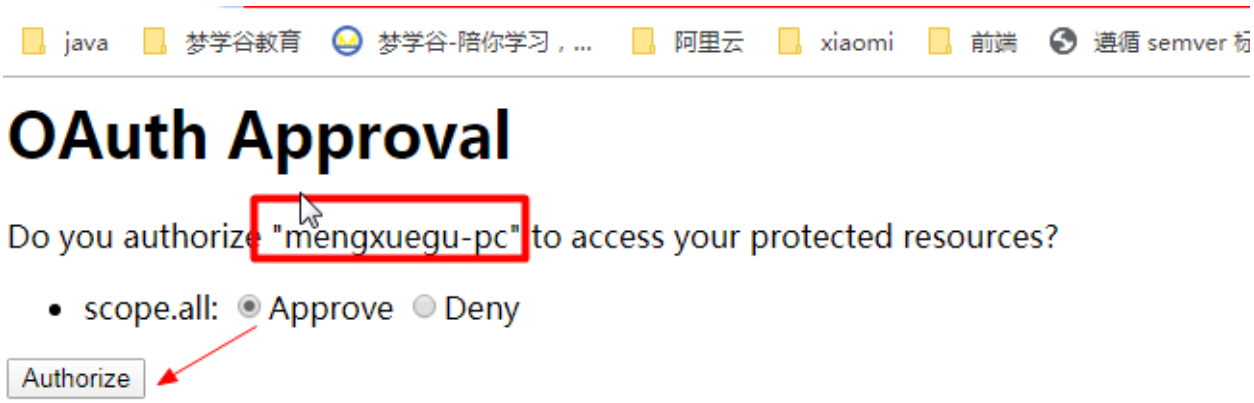
   注意：

   - 此处输入的用户名、密码是在认证服务器输入的（看端口8090），而不是在客户端上输入的，这样更加安全，因为客户端不知道用户的用户名和密码。
   - 而密码模式中，输入的用户名、密码不是在认证服务器（不是8090端口）上输入的，而是在客户端（第三方应用）输入的，这样客户端知识了就不太安全。

仅供购买者学习，禁止盗版、转卖、传播课程



3. 输入用户名密码后，登录后会重新跳转授权页面，询问资源所有者：是否将受保护的资源授权给 `mengxuegu-pc` 客户端：



4. 选择 `Approve` 后，点击 `authorize` 同意授权 scope.all 资源后，会跳转到指定的 `redirect_uri` ，回调路径携带了带着一个授权码（ `code=V3ENnC` ），如下图：

5. 获取到授权码(code) 后，就可以通过它来获取访问令牌(access_token)。

2.4.7 通过授权码获取令牌 token

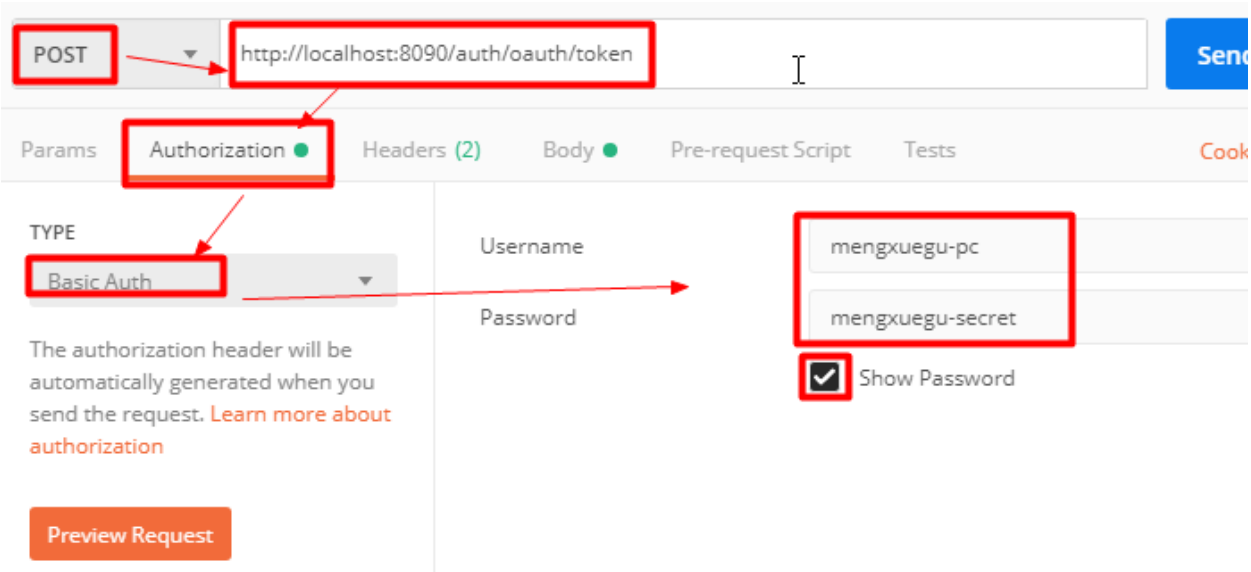
涉及类：`TokenEndpoint`

`POST` 方式请求：http://localhost:8090/auth/oauth/token

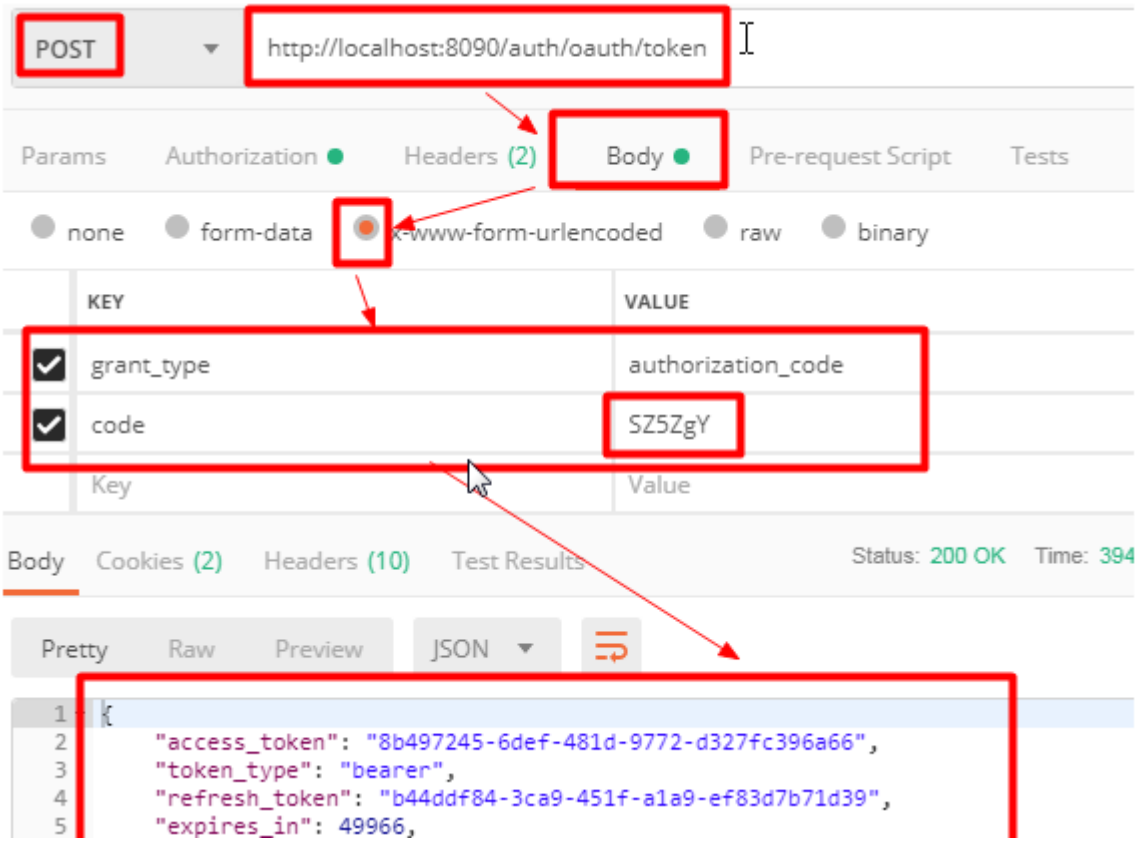
仅供购买者学习，禁止盗版、转卖、传播课程

1. 安装 Postman(你可以想象成就是客户端) ，安装文件位于： `03-配套软件\Postman-win64-5.5.2-Setup.exe`
2. 请求头: `Authorization: Basic bWVuZ3h1ZWd1LXBjOm1lbmd4dWVndS1zZWNyZXQ=`

   是将 `client_id:client_secret` 通过 Base64 编码

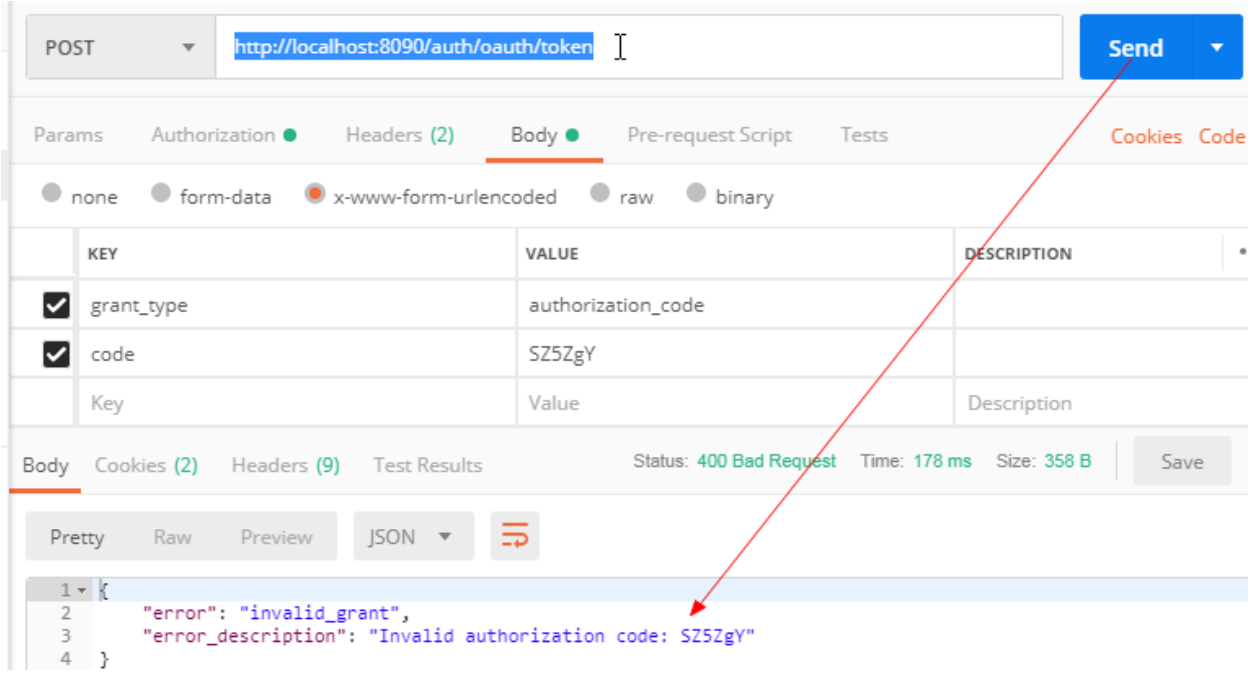


3. `post` 方式，请求体中指定 授权方式 和 授权码



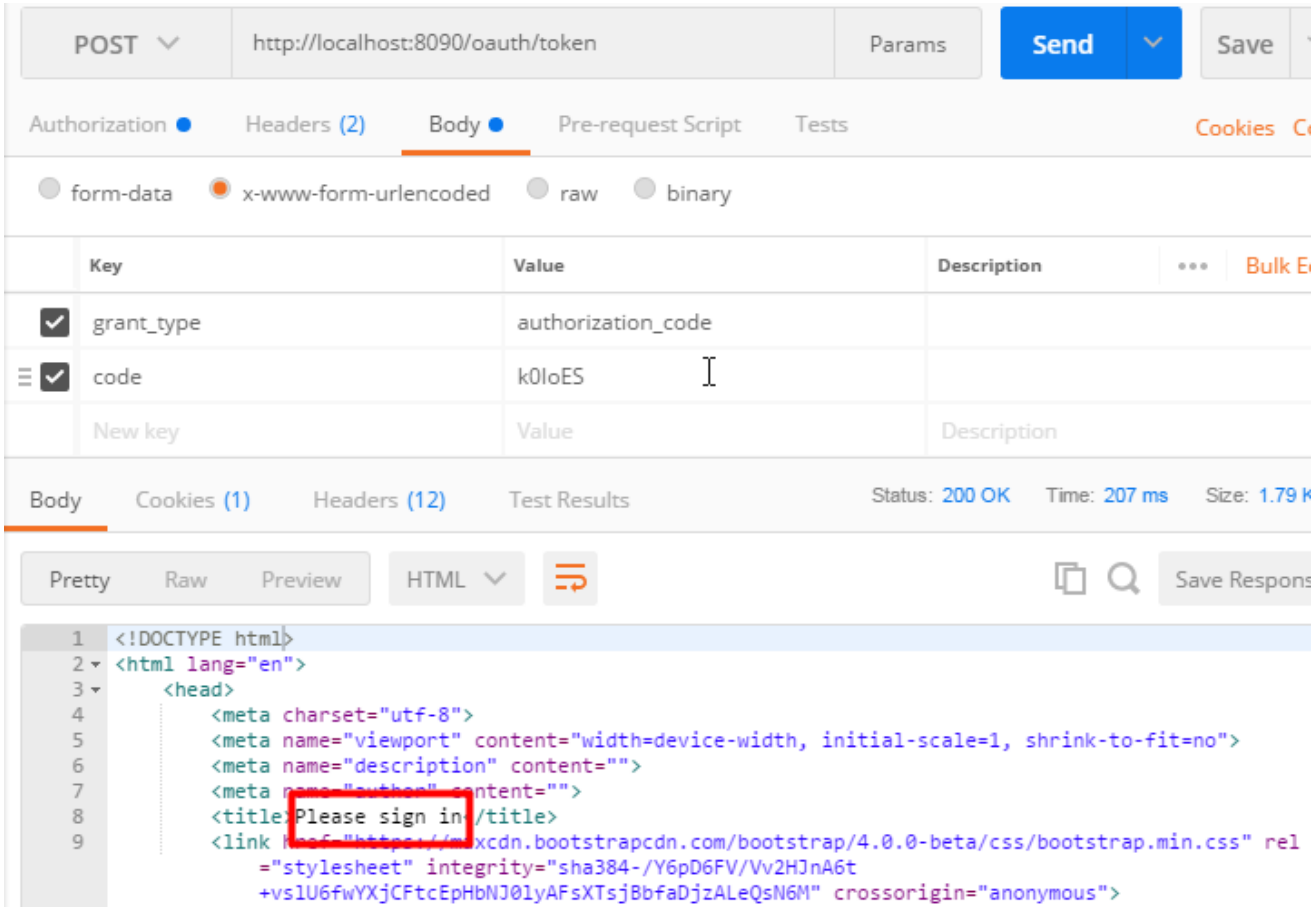
4. 每个授权码申请令牌后就会失效了,需要重新发送请求获取授权码再去认证, 如下再请求就失败了.

仅供购买者学习，禁止盗版、转卖、传播课程



2.4.8 解决申请令牌响应登录页面HTML

1. **注意: 如果响应提登录页面html , 那是因为要对 client_secret 加密**



2. 解决方式: 在 AuthorizationServerConfig 中对 `client_secret` 加密, 如下:
    1. 注入 `PasswordEncoder` 加密器

仅供购买者学习，禁止盗版、转卖、传播课程

2. 加密 `client_secret` ： .secret(`passwordEncoder.encode("mengxuegu-secret")`)
3. 重启 `mengxuegu-oauth2-auth-server` 认证服务器

```
package com.mengxuegu.oauth2.config;
....

@Configuration
@EnableAuthorizationServer // 开启认证服务器功能
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {

    @Autowired
+    private PasswordEncoder passwordEncoder;
    /**
     * 配置被允许访问此认证服务器的客户端信息
     */
    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
        // 使用内存方式
        clients.inMemory()
                .withClient("mengxuegu-pc") // 客户端id
                // 客户端密码，要加密,不然一直要求登录, 获取不到令牌, 而且一定不能被泄露
                .secret(passwordEncoder.encode("mengxuegu-secret"))
                // 资源id, 如商品资源
                .resourceIds("product-server")
                // 授权类型, 可同时支持多种授权类型
                .authorizedGrantTypes("authorization_code", "password",
"implicit","client_credentials","refresh_token")
                // 授权范围标识，哪部分资源可访问（all是标识，不是代表所有）
                .scopes("all")
                // false 跳转到授权页面手动点击授权，true 不用手动授权，直接响应授权码，
                .autoApprove(false)
                .redirectUris("http://www.mengxuegu.com/"); // 客户端回调地址
    }

}
```

2.5 密码授权模式(password)

2.5.1 概述

密码模式（Resource Owner Password Credentials Grant）中 ，用户向客户端提供自己在服务提供商（认证服务器）上的用户名和密码，然后客户端通过用户提供的用户名和密码向服务提供商（认证服务器）获取令牌。

如果用户名和密码遗漏，服务提供商（认证服务器）无法判断客户端提交的用户和密码是否盗取来的，那意味着令牌就可随时获取，数据被丢失。

适用于产品都是企业内部的，用户名密码共享不要紧。如果是第三方这种不太适合。也适用手机APP提交用户名密码

仅供购买者学习，禁止盗版、转卖、传播课程

2.5.2 配置密码模式

注入 AuthenticationManager

说明：

开启密码模式需要注入 `AuthenticationManager` ，所以要先在 SpringSecurityConfig 添加 AuthenticationManager 到容器中。

步骤：

1. 在安全配置类中 `com.mengxuegu.oauth2.server.config.SpringSecurityConfig` ，

   添加 `AuthenticationManager` 到容器中

```
/**
 * password 密码模式要使用此认证管理器
 */
@Bean
@Override
public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
}
```

完整代码：

```
package com.mengxuegu.oauth2.server.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * 安全配置类
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@EnableWebSecurity // 包含了@Configuration注解
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired // 在 SpringSecurityBean 添加到容器了
    private PasswordEncoder passwordEncoder;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // 内存方式存储用户信息
        auth.inMemoryAuthentication().withUser("admin")
                .password(passwordEncoder.encode("1234")).authorities("product");
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
    }

    /**
     * password 密码模式要使用此认证管理器
     */
    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}
```

指定密码模式

在认证服务配置中 `com.mengxuegu.oauth2.server.config.AuthorizationServerConfig` ：

1. 覆盖父类 `configure(AuthorizationServerEndpointsConfigurer endpoints)` 方法, 此方法用于配置令牌访问端点后面还会讲解，把 `authenticationManager` 注入并添加

```
@Autowired // SpringSecurityConfig添加到容器中了
private AuthenticationManager authenticationManager;


@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    // 密码模式要设置认证管理器
    endpoints.authenticationManager(authenticationManager)
}
```

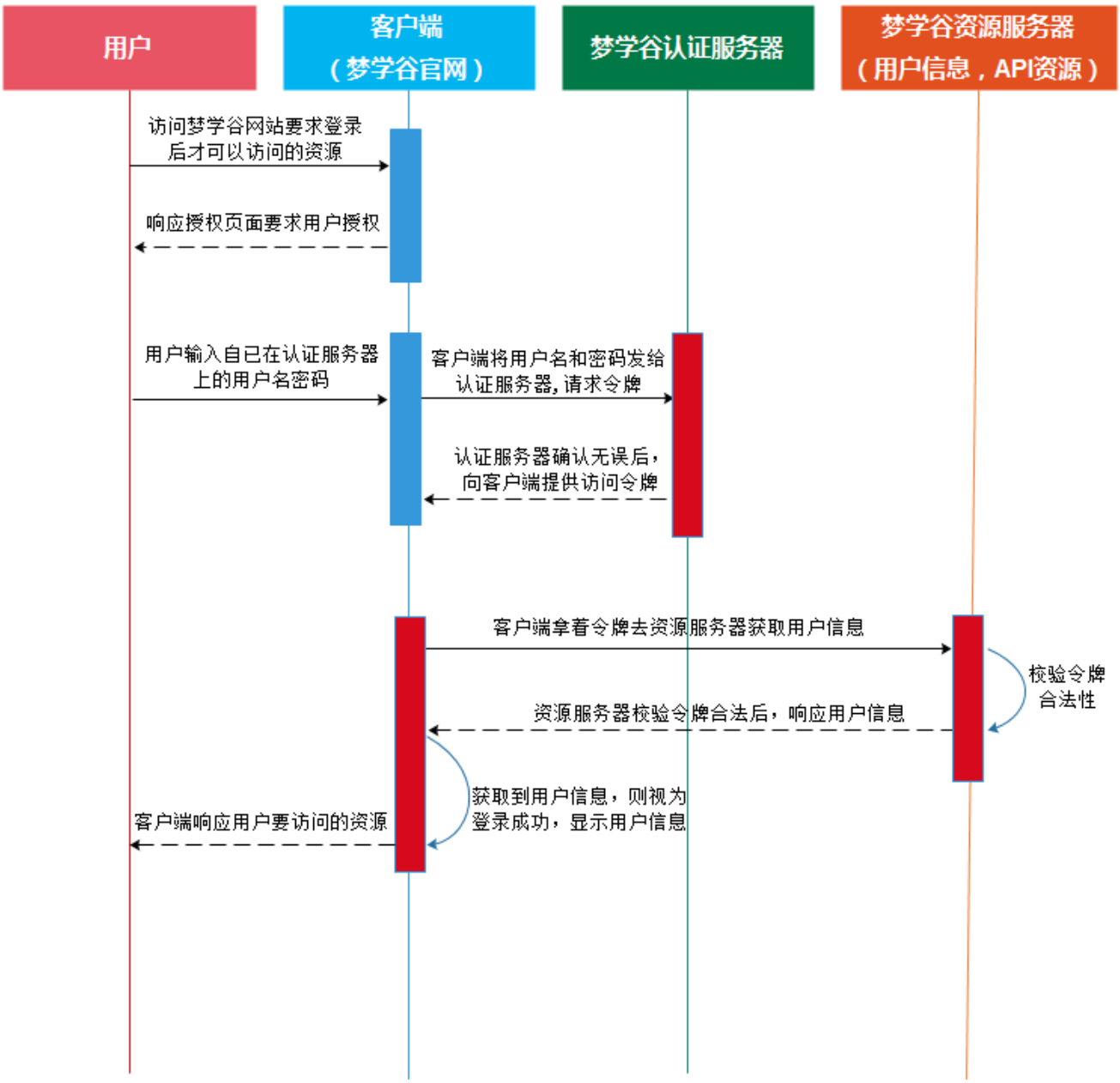
2. 针对 `mengxuegu-pc` 客户端添加 `password` 密码模式支持, 可以同时支持多个模式，如下配置即可 ：

   AuthorizationServerConfig#configure(ClientDetailsServiceConfigurer clients)

```
.authorizedGrantTypes("authorization_code", "password", "implicit","client_credentials","refresh_token")
```

2.5.3 获取令牌 token

仅供购买者学习，禁止盗版、转卖、传播课程

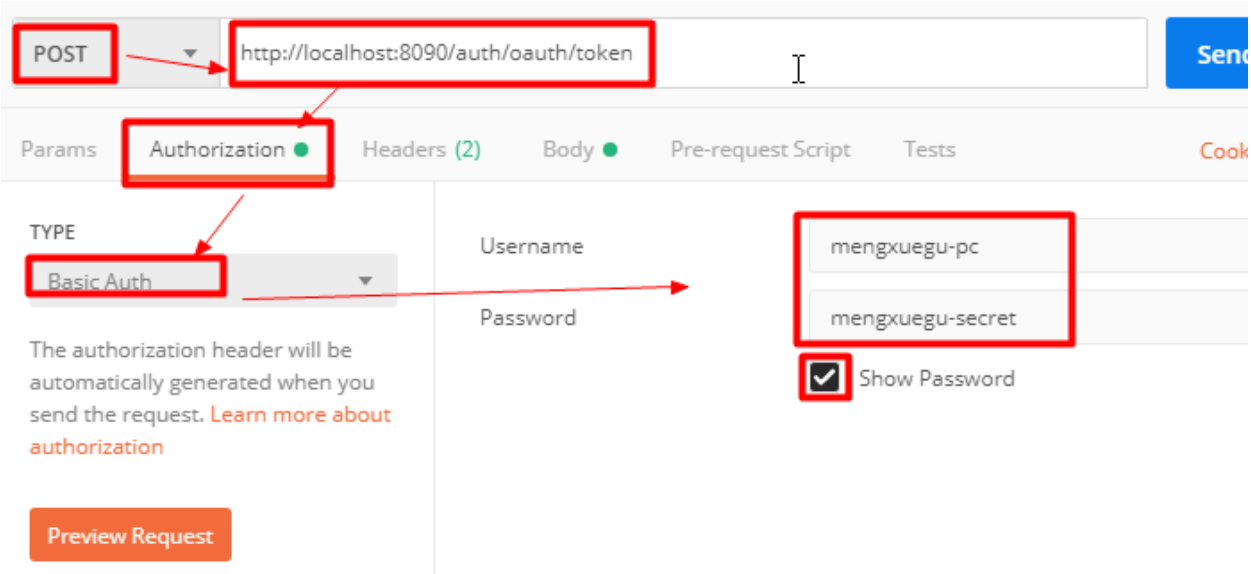


请求地址：http://localhost:8090/auth/oauth/token

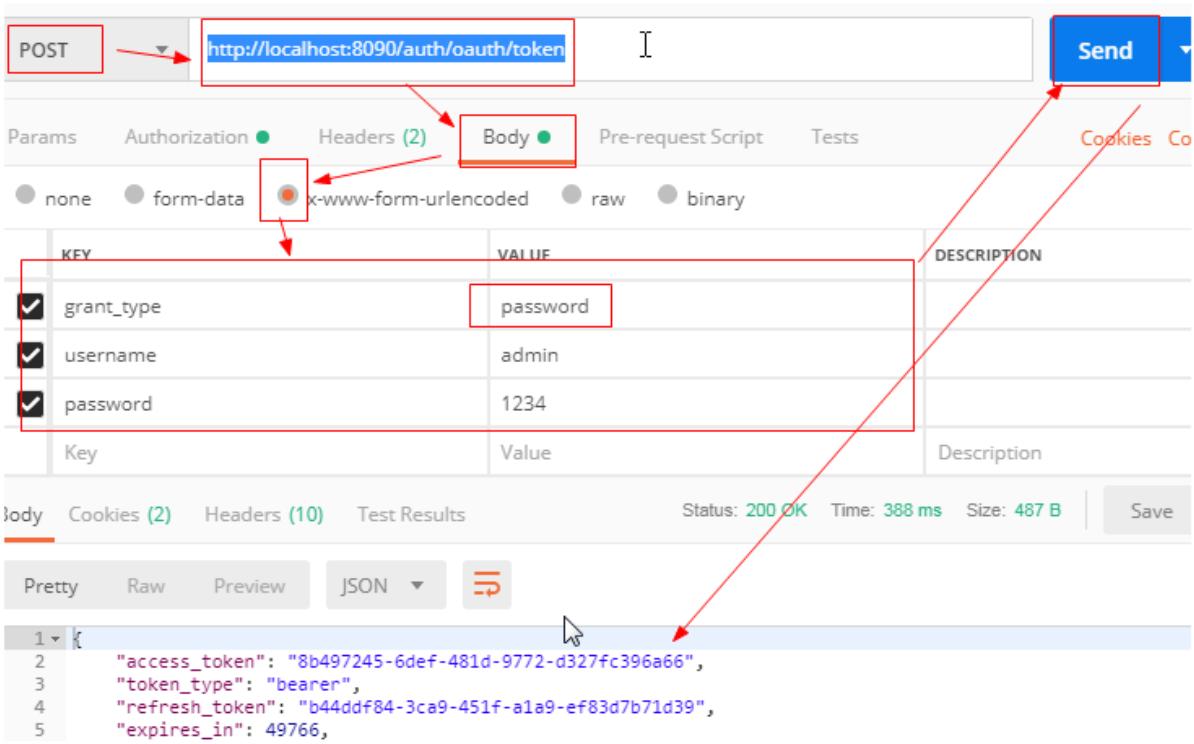
1. 重启认证服务器
2. 请求头: `Authorization: Basic bWVuZ3h1ZWd1LXBjOm1lbmd4dWVndS1zZWNyZXQ=`

   是将 `client_id:client_secret` 通过 Base64 编码

仅供购买者学习，禁止盗版、转卖、传播课程



3. post 方式，请求体中指定: 授权方式 、用户名、密码

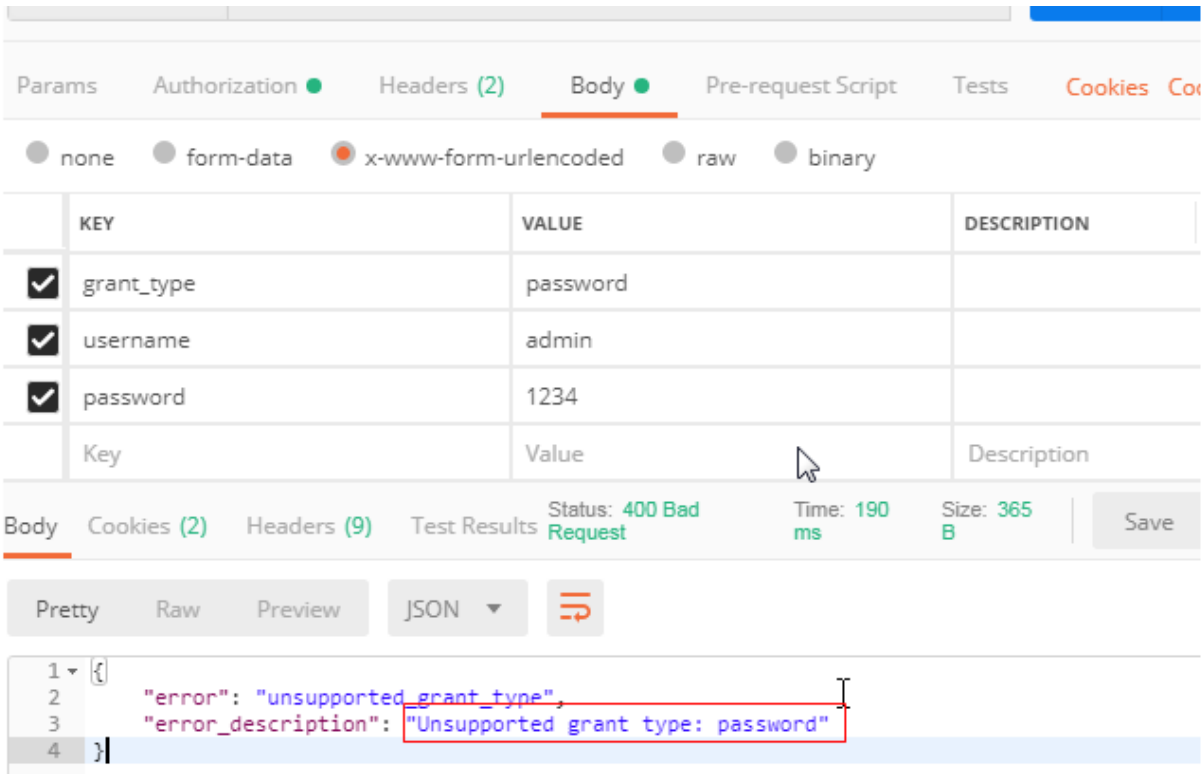


如果获取token时，报错 Unsupported grant type: password ,参见下节

2.5.4 解决提示 Unsupported grant type: password

获取token时，报错 Unsupported grant type: password 如下

仅供购买者学习，禁止盗版、转卖、传播课程



解决： 参见 `配置密码模式` 章节

1. 在 SpringSecurityConfig 添加 AuthenticationManager 到容器中
2. 在 AuthorizationServerConfig 中覆盖 configure(AuthorizationServerEndpointsConfigurer endpoints)，注入 AuthenticationManager 实例

2.6 简化授权模式(Implicit)

2.6.1 概述

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 ，不需要先获取授权码。

直接可以一次请求就可得到令牌，在 `redirect_uri` 指定的回调地址中传递令牌（ access_token ）。

该模式适合直接运行在浏览器上的应用，不用后端支持（例如 Javascript 应用）。

注意：只要客户端id即可 ，客户端密码都不需要。

2.6.2 指定简化模式

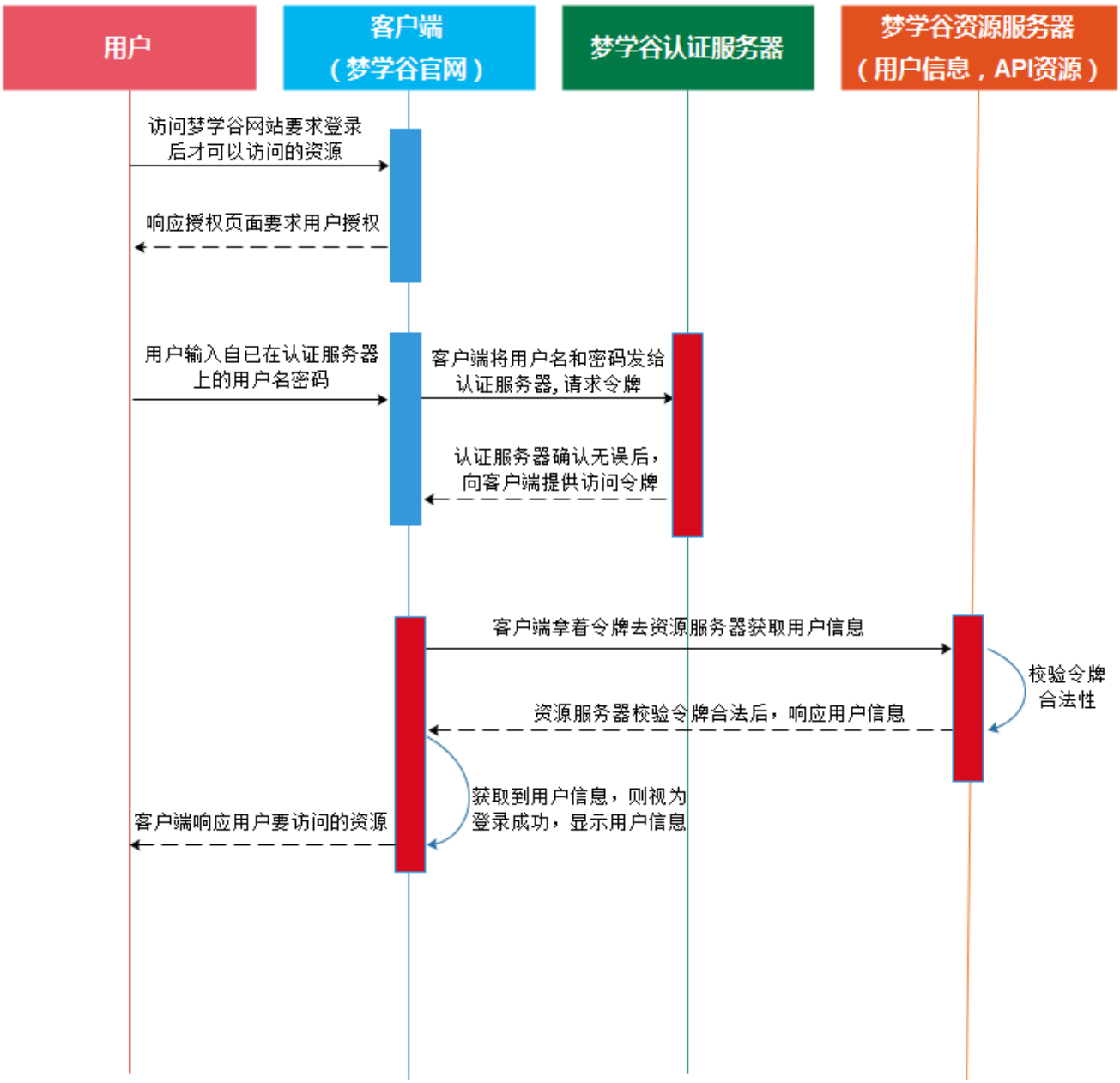
在 AuthorizationServerConfig#configure(ClientDetailsServiceConfigurer) 中指定 implicit

仅供购买者学习，禁止盗版、转卖、传播课程

```java
AuthorizationServerConfig.java
public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    // 使用内存方式
    clients.inMemory() InMemoryClientDetailsServiceBuilder
            .withClient( clientId: "mengxuegu-pc") // 客户端id
            // 客户端密码，要加密,不然一直要求登录，获取不到令牌，而且一定不能被泄露
            .secret(passwordEncoder.encode( charSequence: "mengxuegu-secret")) Clie
            // 资源id，如商品资源
            .resourceIds("product-server") ClientDetailsServiceBuilder<InMemoryClient
            // 授权类型，可同时支持多种授权类型
            .authorizedGrantTypes("authorization_code", "password", "implicit","
            // 授权范围标识，（all是标识，不是代表所有）
            .scopes("all") ClientDetailsServiceBuilder<InMemoryClientDetailsServiceBuilde
            // false 跳转到授权页面手动点击授权，true 不用哪部分资源可访问手动授权，直
            .autoApprove(false) ClientDetailsServiceBuilder<InMemoryClientDetailsService
            .redirectUris("http://www.mengxuegu.com/"); // 客户端回调地址
}
```

2.6.3 获取令牌 token

仅供购买者学习，禁止盗版、转卖、传播课程



1. 打开浏览器，访问地址：

```
http://localhost:8090/auth/oauth/authorize?client_id=mengxuegu-pc&response_type=token
```

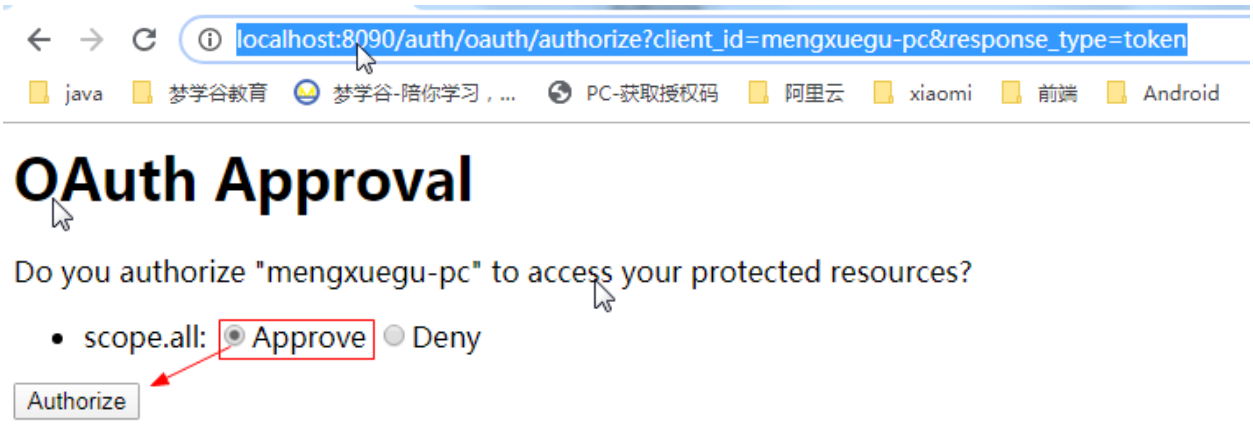
注意，此时 `response_type` 的参数值必须是 `token` 。上面第1行和第2行是连接一起的，没有换行

2. 当请求到达认证服务器的 `AuthorizationEndpoint` 后，它会要求资源所有者做身份验证：

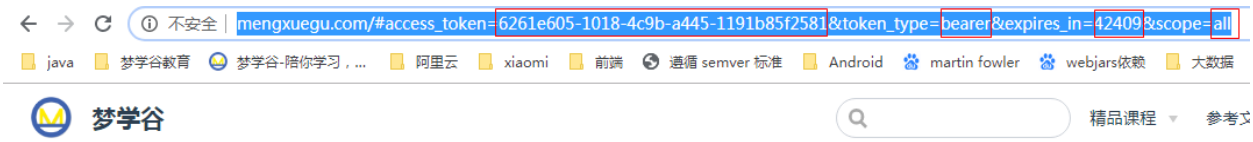
> 注意：如果没有进入到此页面，因为前面在密码授权时认证过，会直接响应令牌。

仅供购买者学习，禁止盗版、转卖、传播课程

3. 输入用户名密码后，重新跳转授权页面，询问资源所有者是否将受保护的资源，授权给 `mengxuegu-pc` 客户端：



4. 选择 `Approve` 后，点击 `authorize` 同意授权 scope.app 资源后，会跳转到指定的 `redirect_uri` ，回调路径携带着令牌 `access_token` 、 `expires_in` 、 `scope` 等 ，如下图：



5. 注意：
   - 简化模式不允许按照 OAuth2 规范发布刷新令牌（refresh token）。这种行为是有必要的，它要求在使用运行在浏览器中的程序时，用户必须在场，这样可以在任何需要的时候，给第三方应用授权。
   - 当使用简化模式时，第三方应用始终需要通过重定向URI来注册，这样能确保不会将token传给不需要验证的客户端。如果不这样做，一些心怀不轨的用户可能先注册一个应用，然后试图让其他的应用来顶替，接收这个 token，这样可能导致灾难性的结果。

仅供购买者学习，禁止盗版、转卖、传播课程

2.7 客户端授权模式(Client)

2.7.1 概述

客户端模式（Client Credentials Grant）指客户端以自己的名义，而不是以用户的名义，向服务提供商（认证服务器）进行认证。严格地说，客户端模式并不属于OAuth框架所要解决的问题。在这种模式中，用户直接向客户端注册，客户端以自己的名义要求服务提供商（认证服务器）提供服务，其实不存在授权问题。

- 客户端向认证服务器进行身份认证，并要求一个访问令牌。
- 认证服务器确认无误后，向客户端提供访问令牌。

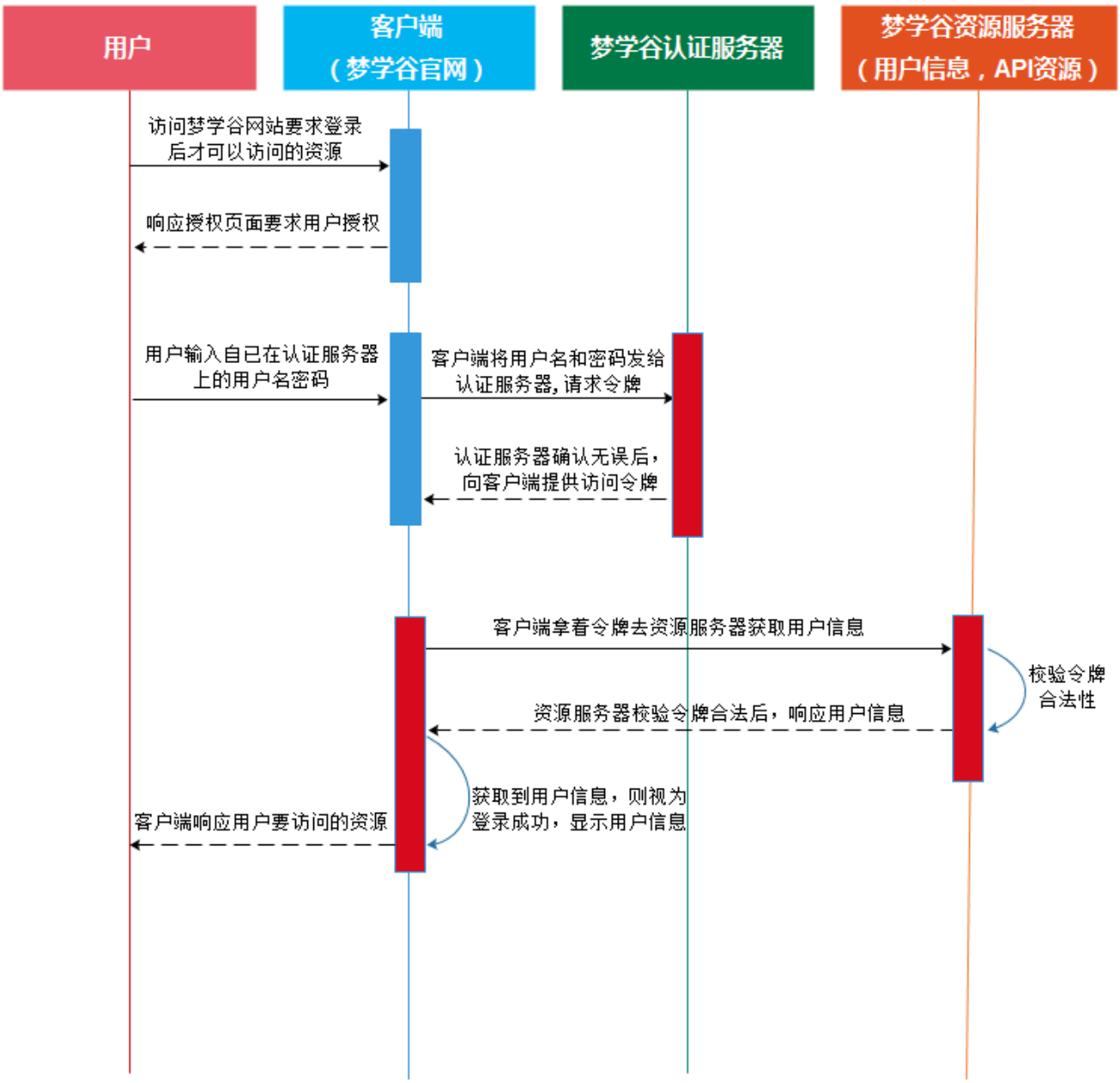
2.7.2 指定客户端模式

在 AuthorizationServerConfig#configure(ClientDetailsServiceConfigurer) 中指定 client_credentials

```
AuthorizationServerConfig.java
configure(ClientDetailsServiceConfigurer clients) throws Exception {
存方式
inMemory() InMemoryClientDetailsServiceBuilder
.withClient( clientId: "mengxuegu-pc") // 客户端id
// 客户端密码，要加密,不然一直要求登录，获取不到令牌，而且一定不能被泄露
.secret(passwordEncoder.encode( charSequence: "mengxuegu-secret")) ClientDetailsServiceBuilde
// 资源id，如商品资源
.resourceIds("product-server") ClientDetailsServiceBuilder<InMemoryClientDetailsServiceBuilder>.
// 授权类型，可同时支持多种授权类型
.authorizedGrantTypes("authorization_code", "password", "implicit","client_credentials"
// 授权范围标识，（all是标识，不是代表所有）哪部分资源可访问
.scopes("all") ClientDetailsServiceBuilder<InMemoryClientDetailsServiceBuilder>.ClientBuilder
// false 跳转到授权页面手动点击授权，true 不用手动授权，直接响应授权码，
```

2.7.3 获取令牌

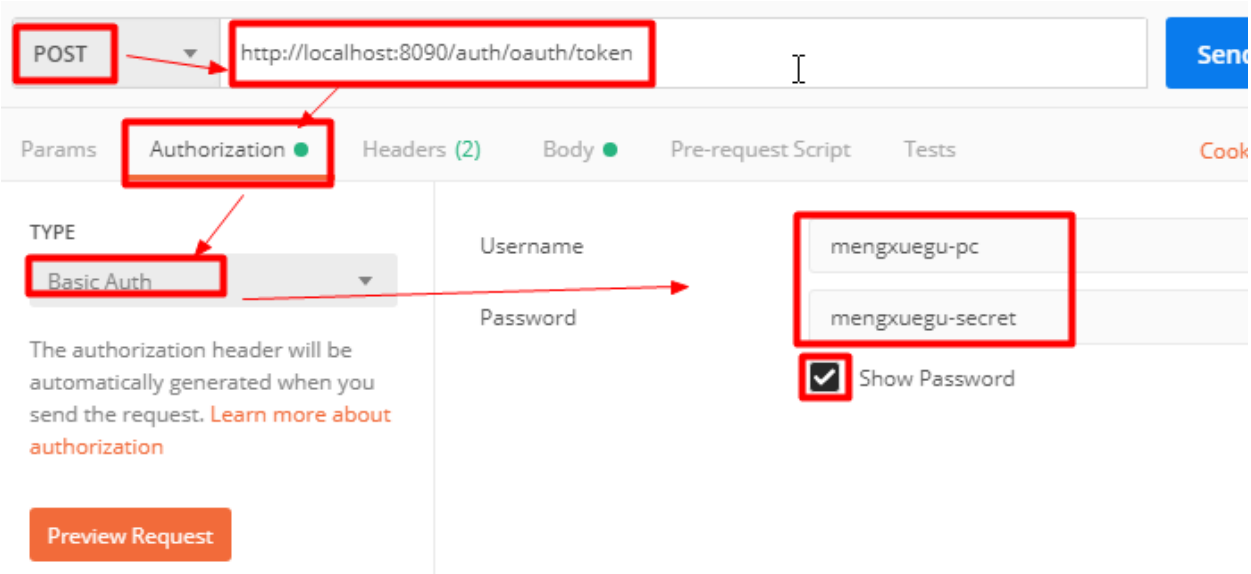
仅供购买者学习，禁止盗版、转卖、传播课程



1. 请求头: `Authorization: Basic bWVuZ3h1ZWd1LXBjOm1lbmd4dWVndS1zZWNyZXQ=`

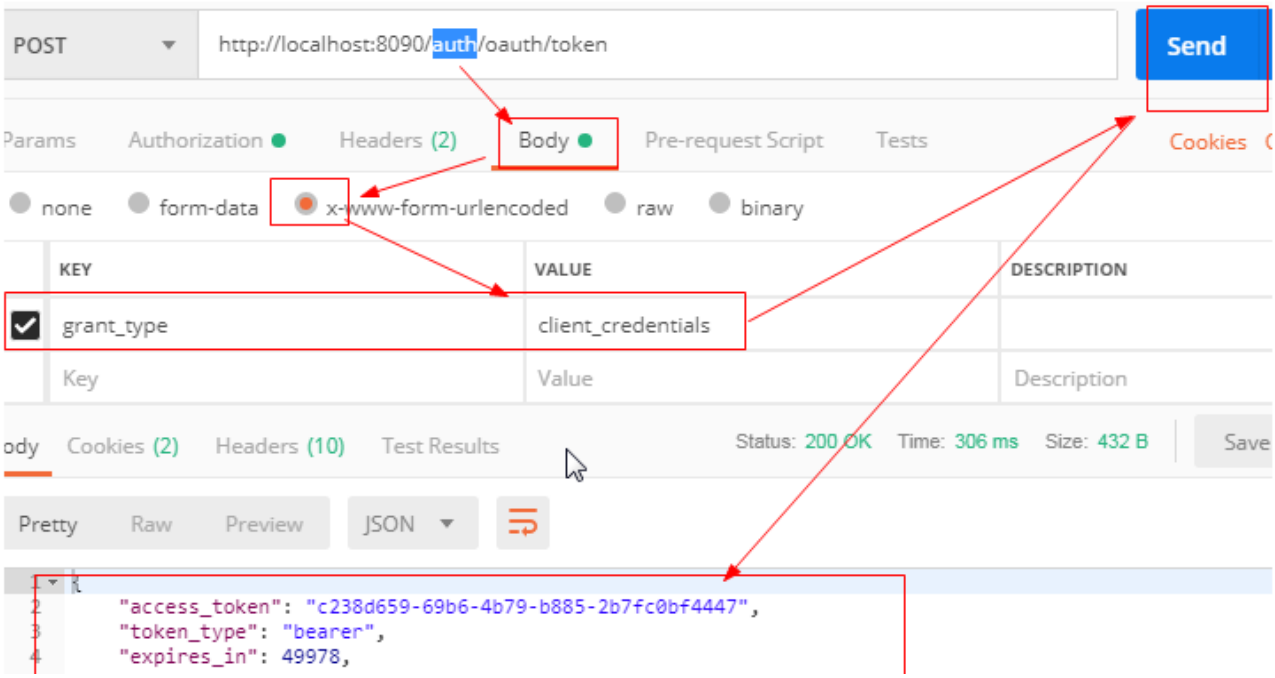
   是将 `client_id:client_secret` 通过 Base64 编码

仅供购买者学习，禁止盗版、转卖、传播课程



2. post 方式，请求体中指定授权类型grant_type ：client_credentials

   注意：响应结果中是没有刷新令牌的



第三章 认证服务器策略配置

3.1 刷新令牌

概述

5. Issuing an Access Token .....
5.1. Successful Response .....
5.2. Error Response .....
6. Refreshing an Access Token .....
7. Accessing Protected Resources .....
7.1. Access Token Types .....

如果用户访问的时候，客户端的"访问令牌"已经过期，则需要使用"更新令牌"申请一个新的访问令牌。

客户端发出更新令牌的HTTP请求，包含以下参数：

- grant_type：表示使用的授权模式，此处的值固定为 `refresh_token` ，必选项。
- refresh_token：表示早前收到的更新令牌，必选项。
- scope：表示申请的授权范围，不可以超出上一次申请的范围，如果省略该参数，则表示与上一次一致。

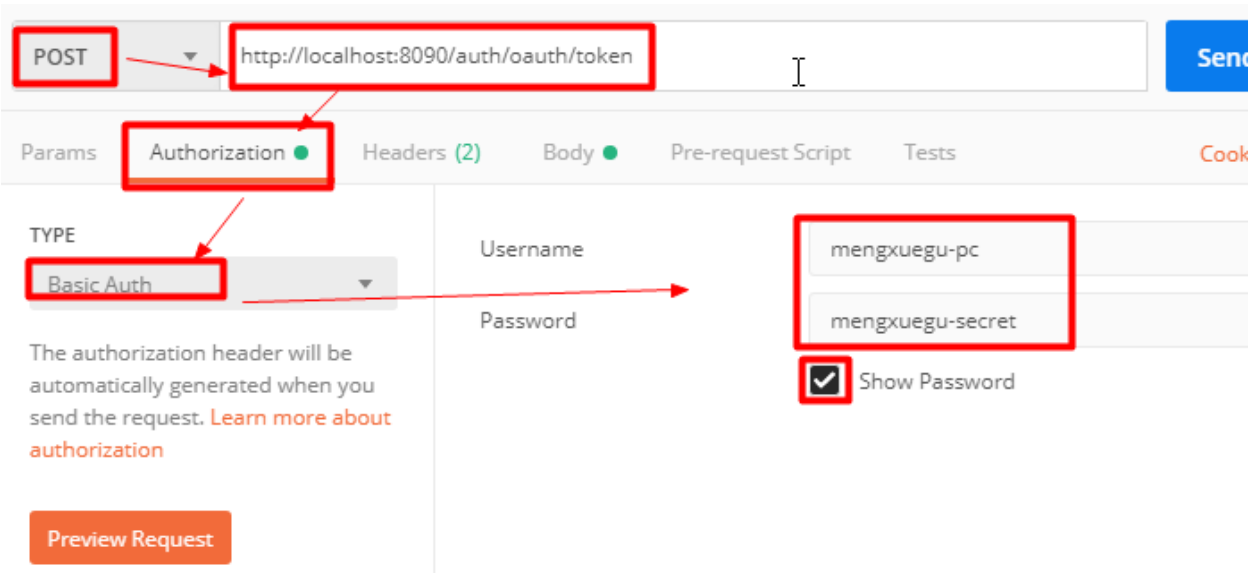
注意: 刷新令牌只在授权码模式和密码模式中才有, 对应的指定这两种模式时, 类型加上 `refresh_token` 即可

```
AuthorizationServerConfig.java
public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    // 使用内存方式
    clients.inMemory() InMemoryClientDetailsServiceBuilder
            .withClient( clientId: "mengxuegu-pc") // 客户端id
            // 客户端密码，要加密,不然一直要求登录，获取不到令牌，而且一定不能被泄露
            .secret(passwordEncoder.encode( charSequence: "mengxuegu-secret")) ClientDeta
            // 资源id，如商品资源
            .resourceIds("product-server") ClientDetailsServiceBuilder<InMemoryClientDetailsS
            // 授权类型，可同时支持多种授权类型
            .authorizedGrantTypes("authorization_code", "password","refresh_token", "i
            // 授权范围标识，（all是标识，不是代表所有）哪部分资源可访问
            .scopes("all") ClientDetailsServiceBuilder<InMemoryClientDetailsServiceBuilder>.Clie
            // false 跳转到授权页面手动点击授权，true 不用手动授权，直接响应授权码，
            .autoApprove(false) ClientDetailsServiceBuilder<InMemoryClientDetailsServiceBuilder
```

获取新令牌报错

1. 请求头: `Authorization: Basic bWVuZ3h1ZWd1LXBjOm1lbmd4dWVndS1zZWNyZXQ=`

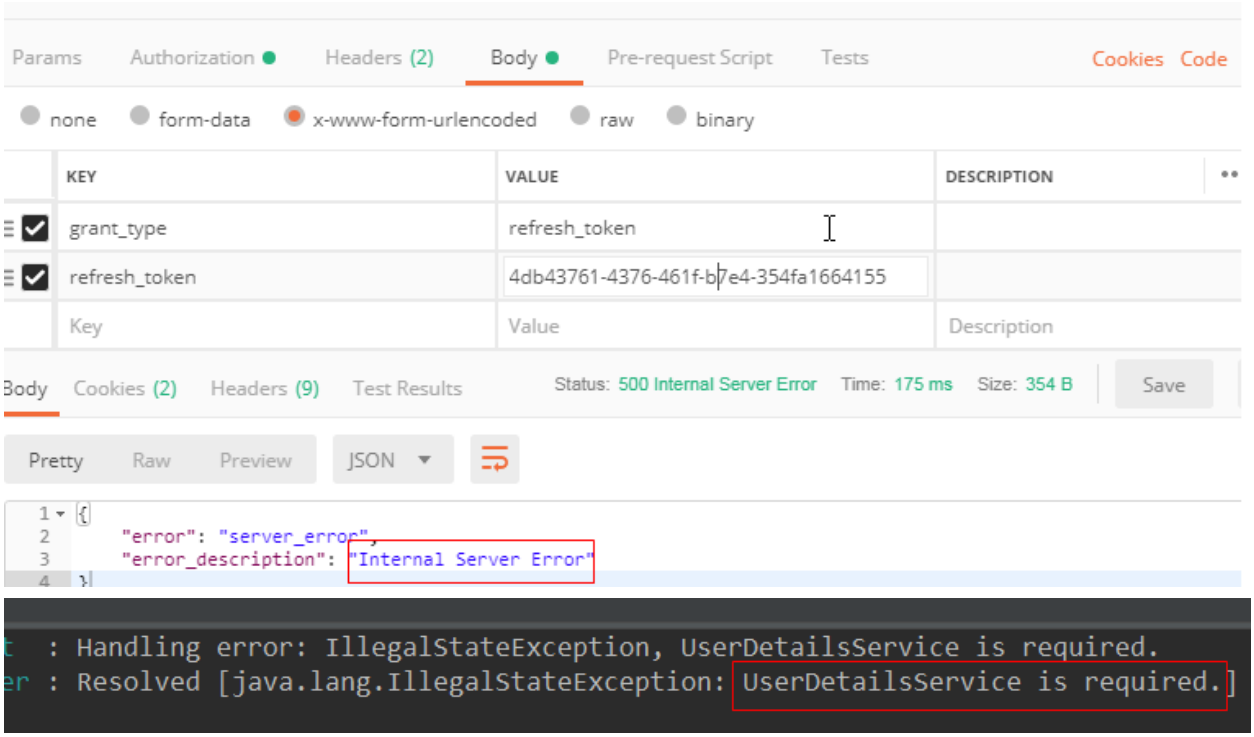
   是将 `client_id:client_secret` 通过 Base64 编码



2. `post` 方式，请求体中指定：授权类型、刷新令牌

仅供购买者学习，禁止盗版、转卖、传播课程

当前报错: Internal Server Error，对应看idea控制台也发出警告: UserDetailsService is required.



原因: 当前需要使用内存方式存储了用户令牌，应用使用 UserDetailsService 才行

创建 UserDetailsService 实现

- 创建 CustomUserDetailsService 动态获取用户令牌
- 安全配置类 SpringSecurityConfig 中注入 CustomUserDetailsService
- 认证配置类 AuthorizationServerConfig 中注入 CustomUserDetailsService ，加入到令牌端点上

具体操作：

1. 创建 com.mengxuegu.oauth2.server.service.CustomUserDetailsService 实现 UserDetailsService 接口

类上不要少了 `@Component` ，密码加密

```
package com.mengxuegu.oauth2.server.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
15 @Component
16 public class CustomUserDetailsService implements UserDetailsService {
17
18     @Autowired
19     private PasswordEncoder passwordEncoder;
20
21     @Override
22     public UserDetails loadUserByUsername(String u) throws UsernameNotFoundException {
23         return new User("admin", passwordEncoder.encode("1234"),
24                 AuthorityUtils.commaSeparatedStringToAuthorityList("product"));
25     }
26
27 }
```

2. 在 SpringSecurityConfig 中添加注入 `CustomUserDetailsService`

```
1  package com.mengxuegu.oauth2.server.config;
2
3  。。。。
4
5  /**
6   * 安全配置类
7   * @Auther: 梦学谷 www.mengxuegu.com
8   */
9  @EnableWebSecurity // 包含了@Configuration注解
10 public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
11
12     @Autowired // 在 SpringSecurityBean 添加到容器了
13     private PasswordEncoder passwordEncoder;
14
15 +   @Autowired
16 +   private UserDetailsService customUserDetailsService;
17
18     @Override
19     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
20 +       auth.userDetailsService(customUserDetailsService);
21     }
22
23     /**
24      * password 密码模式要使用此认证管理器
25      */
26     @Bean
27     @Override
28     public AuthenticationManager authenticationManagerBean() throws Exception {
29         return super.authenticationManagerBean();
30     }
31 }
```

3. 认证配置类 AuthorizationServerConfig 中注入 CustomUserDetailsService ，加入到令牌端点上

```
1  package com.mengxuegu.oauth2.server.config;
2
```

仅供购买者学习，禁止盗版、转卖、传播课程

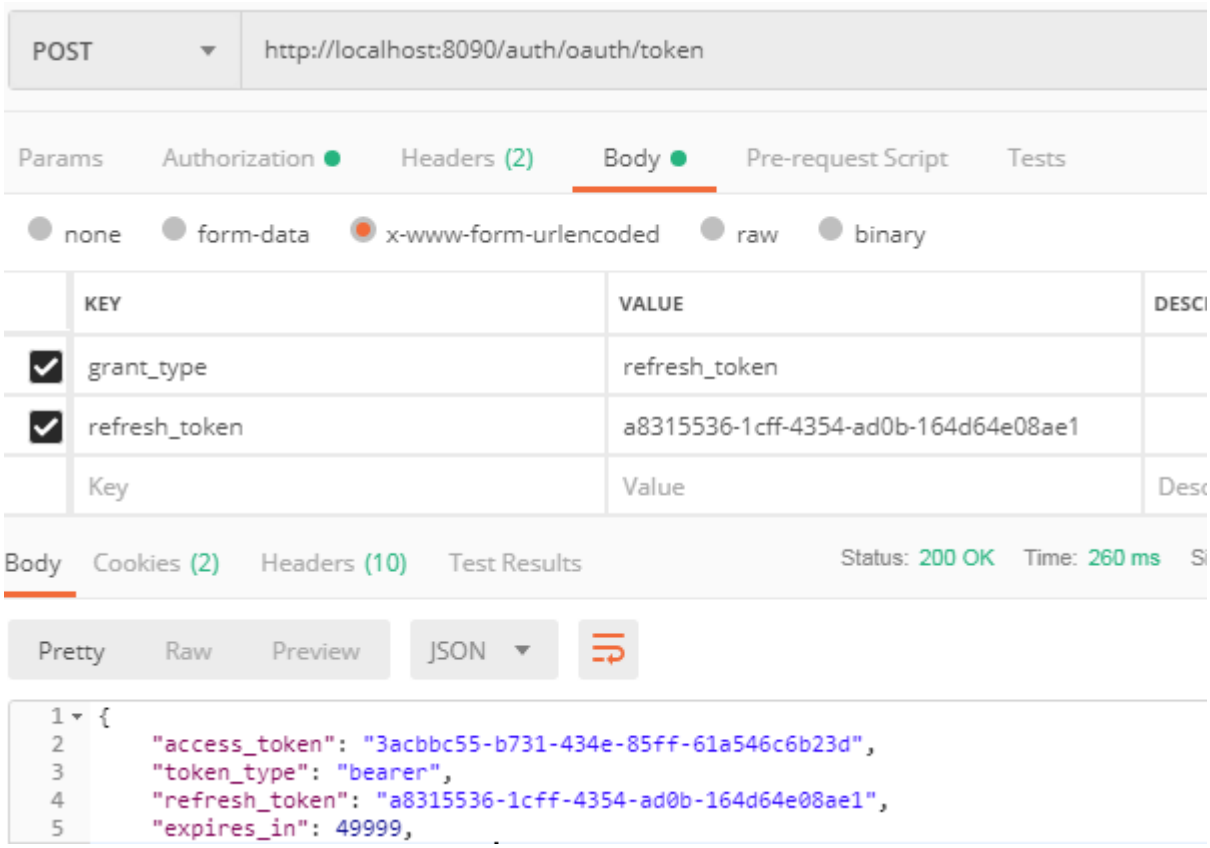
```
3   ...
4
5   /**
6    * 认证服务器配置
7    * @Auther: 梦学谷 www.mengxuegu.com
8    */
9   @Configuration
10  @EnableAuthorizationServer // 开启认证服务器功能
11  public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {
12
13      @Autowired // 在 SpringSecurityBean 添加到容器了
14      private PasswordEncoder passwordEncoder;
15
16      /**
17       * 配置被允许访问此认证服务器的客户端详情信息
18       * 方式1：内存方式管理
19       * 方式2：数据库管理
20       */
21      @Override
22      public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
23          // 使用内存方式
24          clients.inMemory()
25                  .withClient("mengxuegu-pc") // 客户端id
26                  // 客户端密码，要加密,不然一直要求登录, 获取不到令牌, 而且一定不能被泄露
27                  .secret(passwordEncoder.encode("mengxuegu-secret"))
28                  // 资源id, 如商品资源
29                  .resourceIds("product-server")
30                  // 授权类型, 可同时支持多种授权类型
31                  .authorizedGrantTypes("authorization_code", "password",
    "implicit","client_credentials","refresh_token")
32                  // 授权范围标识，（all是标识，不是代表所有）哪部分资源可访问
33                  .scopes("all")
34                  // false 跳转到授权页面手动点击授权，true 不用手动授权，直接响应授权码，
35                  .autoApprove(false)
36                  .redirectUris("http://www.mengxuegu.com/"); // 客户端回调地址
37      }
38
39      @Autowired // SpringSecurityConfig添加到容器中了
40      private AuthenticationManager authenticationManager;
41
42  +   @Autowired
43      private UserDetailsService customUserDetailsService;
44
45      @Override
46      public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
47          // 密码模式要设置认证管理器
48          endpoints.authenticationManager(authenticationManager);
49          // 刷新令牌获取新令牌时需要
50  +         endpoints.userDetailsService(customUserDetailsService);
51      }
52
53  }
```

仅供购买者学习，禁止盗版、转卖、传播课程

测试获取新令牌

1. 重启认证服务器
2. 请求头: `Authorization: Basic bWVuZ3h1ZWd1LXBjOm1lbmd4dWVndS1zZWNyZXQ=`

   是将 `client_id:client_secret` 通过 Base64 编码
3. `post` 方式，请求体中指定： 授权类型 refresh_token 、刷新令牌值



3.2 令牌管理策略 Redis & JDBC

概述

默认情况下，令牌是通过 randomUUID 产生32位随机数的来进行填充的，而产生的令牌默认是存储在内存中。

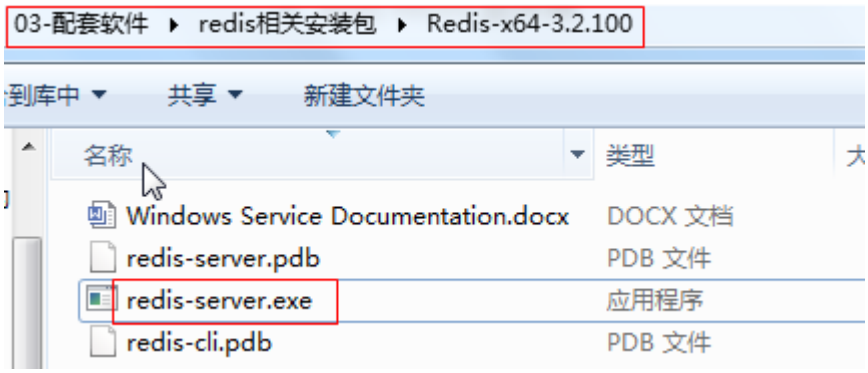
- 内存存储采用的是 `TokenStore` 接口的默认实现类 `InMemoryTokenStore`，开发时方便调试，适用单机版。
- `RedisTokenStore` 将令牌存储到 Redis 非关系型数据库中，适用于并发高的服务。
- `JdbcTokenStore` 基于 JDBC 将令牌存储到 关系型数据库中，可以在不同的服务器之间共享令牌。
- `JwtTokenStore` （JSON Web Token）将用户信息直接编码到令牌中，这样后端可以不用存储它，前端拿到令牌可以直接解析出用户信息。

Redis 管理令牌

启动 Redis 服务器

仅供购买者学习，禁止盗版、转卖、传播课程

1. 启动 Redis 服务器，以管理员身份运行 `redis-server.exe`



添加依赖 pom.xml

1. 认证服务器添加 Redis 相关依赖

```
<!-- redis -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

配置 Redis 管理 Token

1. 创建 `com.mengxuegu.oauth2.server.config.TokenConfig` 指定Redis存储Token

   添加 redis 依赖后, 容器自动就会有 RedisConnectionFactory 实例

```
package com.mengxuegu.oauth2.server.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.security.oauth2.provider.token.TokenStore;
import org.springframework.security.oauth2.provider.token.store.redis.RedisTokenStore;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@Configuration
public class TokenConfig {

    /**
     * Redis 管理令牌
     * 1. 启动 redis 服务器
     * 2. 添加 redis 相关依赖
     * 3. 添加redis 依赖后, 容器就会有 RedisConnectionFactory 实例
     */
    @Autowired
    private RedisConnectionFactory redisConnectionFactory;

```

仅供购买者学习，禁止盗版、转卖、传播课程

```
    @Bean
    public TokenStore tokenStore() {
        // Redis 管理令牌
        return new RedisTokenStore(redisConnectionFactory);
    }

}
```

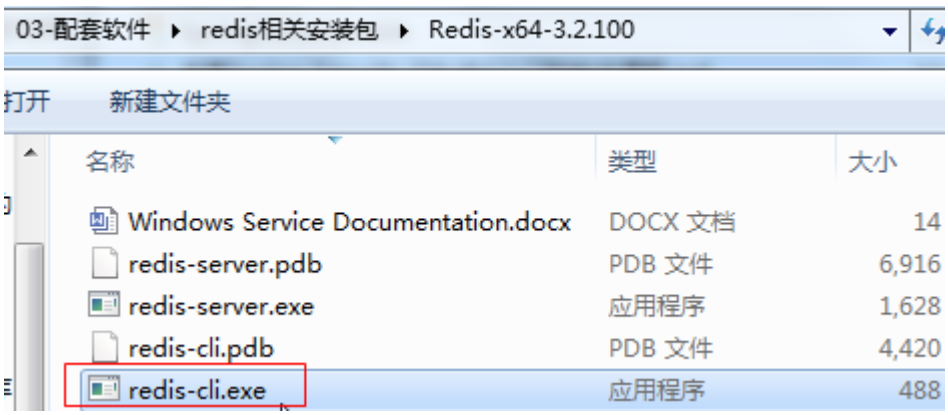
令牌管理策略添加到端点

- 将上面 令牌管理策略 作用到认证服务器端点上，这样策略就会生效。
  - 注入 TokenStore
  - 在 `AuthorizationServerConfig#configure(AuthorizationServerEndpointsConfigurer endpoints)` 方法中添加到端点上.

```
@Autowired // SpringSecurityConfig添加到容器中了
private AuthenticationManager authenticationManager;
@Autowired
private UserDetailsService customUserDetailsService;

@Autowired // 令牌管理策略
+ private TokenStore tokenStore;
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    // 密码模式要设置认证管理器
    endpoints.authenticationManager(authenticationManager);
    // 刷新令牌获取新令牌时需要
    endpoints.userDetailsService(customUserDetailsService);
    // 令牌管理策略
+   endpoints.tokenStore(tokenStore);
}
```

测试

1. 重启认证服务器
2. 打开 Redis 客户端



3. 先使用 `flushall` 命令清除所有数据，方便后面查看

仅供购买者学习，禁止盗版、转卖、传播课程

4. 使用密码模式获取令牌
5. `keys *` 查看效果如下：

```
127.0.0.1:6379> flushall
OK
127.0.0.1:6379> keys *
1) "client_id_to_access:mengxuegu-pc"
2) "auth:4e8bfb4b-1cdc-4ba3-88bd-f59b540541ef"
3) "uname_to_access:mengxuegu-pc:admin"
4) "auth_to_access:1f88a4e115b95fa6e5608f05b06214c1"
5) "access_to_refresh:4e8bfb4b-1cdc-4ba3-88bd-f59b540541ef"
6) "access:4e8bfb4b-1cdc-4ba3-88bd-f59b540541ef"
7) "refresh_to_access:3178e8fa-0f77-42a5-bedf-e84718fec9a9"
8) "refresh:3178e8fa-0f77-42a5-bedf-e84718fec9a9"
9) "refresh_auth:3178e8fa-0f77-42a5-bedf-e84718fec9a9"
127.0.0.1:6379>
```

梦学谷 www.mengxuegu.com | IT在线教育 高端品牌
陪你学习，伴你梦想
仅供购买者学习，禁止盗版、转卖、传播课程

JDBC 管理令牌

创建相关数据表

Spring 官方提供了存储 OAuth2 相关信息的数据库表结构

```
https://github.com/spring-projects/spring-security-oauth/blob/master/spring-security-oauth2/src/test/resources/schema.sql
```

当前使用了 MySQL 数据库，要修改下数据类型：

1. 官方提供的表结构主键类型为 `VARCHAR(256)` ，超过了MySQL限制的长度 `128` ，需要修改为 `VARCHAR(128)`
2. 将 `LONGVARBINARY` 类型修改为 `BLOB` 类型。

本地SQL脚本位于： `02-配套资料\03-数据库脚本\OAuth2.sql`

表字段说明参见本文档章节: `附录>>Oauth2 表字段说明`

修改后的表结构如下：

```sql
create table oauth_client_details (
  client_id VARCHAR(128) PRIMARY KEY,
  resource_ids VARCHAR(128),
  client_secret VARCHAR(128),
  scope VARCHAR(128),
  authorized_grant_types VARCHAR(128),
  web_server_redirect_uri VARCHAR(128),
  authorities VARCHAR(128),
  access_token_validity INTEGER,
  refresh_token_validity INTEGER,
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
11   additional_information VARCHAR(4096),
12   autoapprove VARCHAR(128)
13 );
14 INSERT INTO `oauth_client_details` VALUES ('mengxuegu-pc', 'product-server',
   '$2a$10$VcUHxmgnxBTdB9XNdhoOWujC.lFZ0rO1UizqMAS0GU6WAerFviX.a', 'all',
   'authorization_code,password,implicit,client_credentials,refresh_token', 'http://www.mengxuegu.com/', null,
   '50000', null, null, 'false');
15
16 create table oauth_client_token (
17   token_id VARCHAR(128),
18   token BLOB,
19   authentication_id VARCHAR(128) PRIMARY KEY,
20   user_name VARCHAR(128),
21   client_id VARCHAR(128)
22 );
23
24 create table oauth_access_token (
25   token_id VARCHAR(128),
26   token BLOB,
27   authentication_id VARCHAR(128) PRIMARY KEY,
28   user_name VARCHAR(128),
29   client_id VARCHAR(128),
30   authentication BLOB,
31   refresh_token VARCHAR(128)
32 );
33
34 create table oauth_refresh_token (
35   token_id VARCHAR(128),
36   token BLOB,
37   authentication BLOB
38 );
39
40 create table oauth_code (
41   code VARCHAR(128),
42   authentication BLOB
43 );
44
45 create table oauth_approvals (
46     userId VARCHAR(128),
47     clientId VARCHAR(128),
48     scope VARCHAR(128),
49     status VARCHAR(10),
50     expiresAt TIMESTAMP,
51     lastModifiedAt TIMESTAMP
52 );
53
54
55 -- customized oauth_client_details table
56 create table ClientDetails (
57   appId VARCHAR(128) PRIMARY KEY,
58   resourceIds VARCHAR(128),
59   appSecret VARCHAR(128),

60   scope VARCHAR(128),
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
  grantTypes VARCHAR(128),
  redirectUrl VARCHAR(128),
  authorities VARCHAR(128),
  access_token_validity INTEGER,
  refresh_token_validity INTEGER,
  additionalInformation VARCHAR(4096),
  autoApproveScopes VARCHAR(128)
);
```

JDBC 相关依赖

其中有 mybatis-plus 因为后面要用,所以一起添加进来

```
<!--mybatis-plus启动器-->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<!--druid连接池-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
</dependency>
<!--mysql驱动包-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

配置数据源信息

application.yml 添加数据源

```
server:
  port: 8090

spring:
  # 数据源配置
  datasource:
    username: root
    password: root
    url: jdbc:mysql://127.0.0.1:3306/study-security?serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=utf8
    #mysql8版本以上驱动包指定新的驱动类
    driver-class-name: com.mysql.cj.jdbc.Driver
    type: com.alibaba.druid.pool.DruidDataSource
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
13    #  数据源其他配置, 在 DruidConfig配置类中手动绑定
14    initialSize: 8
15    minIdle: 5
16    maxActive: 20
17    maxWait: 60000
18    timeBetweenEvictionRunsMillis: 60000
19    minEvictableIdleTimeMillis: 300000
20    validationQuery: SELECT 1 FROM DUAL
```

配置 JDBC 管理 Token

1. 添加 `DruidDataSource` 数据源到容器中
2. 指定 JDBC 管理 Token

TokenConfig 完整代码：

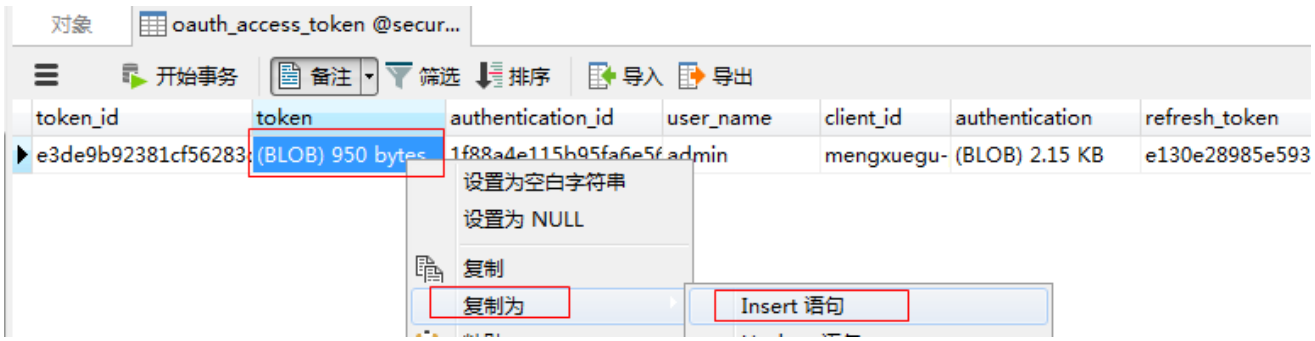
```
1   package com.mengxuegu.oauth2.server.config;
2
3   import com.alibaba.druid.pool.DruidDataSource;
4   import org.springframework.boot.context.properties.ConfigurationProperties;
5   import org.springframework.context.annotation.Bean;
6   import org.springframework.context.annotation.Configuration;
7   import org.springframework.security.oauth2.provider.token.TokenStore;
8   import org.springframework.security.oauth2.provider.token.store.JdbcTokenStore;
9
10  import javax.sql.DataSource;
11
12  /**
13   * @Auther: 梦学谷 www.mengxuegu.com
14   */
15  @Configuration
16  public class TokenConfig {
17
18     /**
19      * Redis 管理令牌
20      * 1. 启动 redis 服务器
21      * 2. 添加 redis 相关依赖
22      * 3. 添加redis 依赖后, 容器就会有 RedisConnectionFactory 实例
23      */
24     // @Autowired
25     // private RedisConnectionFactory redisConnectionFactory;
26
27     /**
28      * JDBC 管理令牌
29      * 1. 创建相关数据表
30      * 2. 添加 jdbc 相关依赖
31      * 3. 配置数据源信息
32      */
33     @Bean
34     @ConfigurationProperties(prefix = "spring.datasource")
35     public DataSource dataSource() {
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
        return new DruidDataSource();
    }

    @Bean
    public TokenStore tokenStore() {
        // Redis 管理令牌
        // return new RedisTokenStore(redisConnectionFactory);
        // JDBC 管理令牌
        return new JdbcTokenStore(dataSource());
    }

}
```

测试

1. 重启认证服务器
2. 使用密码模式进行授权操作，然后查询 `oauth_access_token` 表就存储了令牌信息:



注意：表中的 token 值是序列化 `DefaultOAuth2AccessToken` 后的串，复制为 insert 可以查看效果

```
INSERT INTO `security`.`oauth_access_token` (
    `token_id`,
    `token`,
    `authentication_id`,
    `user_name`,
    `client_id`,
    `authentication`,
    `refresh_token`
)
VALUES
    (
        'e3de9b92381cf56283df9191c9bc6146',
        '□□\0ENQsr\0Corg.springframework.security.oauth2.common.DefaultOAuth2AccessToken
        '1f88a4e115b95fa6e5608f05b06214c1',
        'admin',
        'mengxuegu-pc',
        '□□\0ENQsr\0Aorg.springframework.security.oauth2.provider.OAuth2Authentication□@
        'e130e28985e593b0980b0c20d3984b7e'
    );
```

3.3 JDBC 管理授权码

仅供购买者学习，禁止盗版、转卖、传播课程

概述

授权码主要操作oauth_code表的，只有当 grant_type 为 "authorization_code" 时,该表中才会有数据产生; 其他的grant_type没有使用该表。更多的细节请参考 JdbcAuthorizationCodeServices

默认情况下并未将授权码保存到 oauth_code 表中，原因是 JdbcAuthorizationCodeServices 没有添加到容器中。

开启后，会将授权码放到 `auth_code` 表，授权后就会删除它。

授权码切换成 JDBC 方式

- 创建JDBC管理授权码的实例: `JdbcAuthorizationCodeServices` 并添加到容器
- 在 AuthorizationServerConfig#configure(AuthorizationServerEndpointsConfigurer clients) 中添加到认证服务端点上

```
  @Autowired // SpringSecurityConfig添加到容器中了
  private AuthenticationManager authenticationManager;
  @Autowired
  private UserDetailsService customUserDetailsService;

  @Autowired // 令牌管理策略
  private TokenStore tokenStore;

+ @Autowired
+ private DataSource dataSource;
+ @Bean // 授权码管理策略
+ public AuthorizationCodeServices jdbcAuthorizationCodeServices() {
    // JDBC方式保存授权码到 oauth_code 表中,
    // 意义不大，因为获取一次令牌后，授权码就失效了
+     return new JdbcAuthorizationCodeServices(dataSource);
+ }

  @Override
  public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    // 密码模式要设置认证管理器
    endpoints.authenticationManager(authenticationManager);
    // 刷新令牌获取新令牌时需要
    endpoints.userDetailsService(customUserDetailsService);
    // 令牌管理策略
    endpoints.tokenStore(tokenStore);
    // 授权码管理策略，针对授权码模式有效，会将授权码放到 auth_code 表，授权后就会删除它
+     endpoints.authorizationCodeServices(jdbcAuthorizationCodeServices());
  }
```

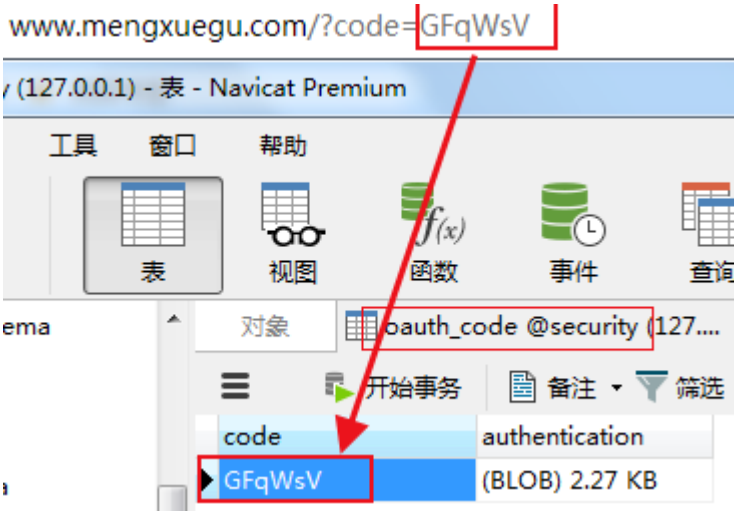
测试

1. 重启
2. 发送请求获取授权码

仅供购买者学习，禁止盗版、转卖、传播课程

```
http://localhost:8090/oauth/authorize?client_id=mengxuegu-pc&response_type=code
```

3. 查看 oauth_code 数据表数据



3.4 JDBC 存储客户端信息

查看客户端详情记录

在表 `oauth_client_details` 中有一条客户端详情记录，设置的字段如下：

- `client_id` ：客户端ID
- `resource_ids` ：可访问的资源服务器ID, 不写则不校验.
- `client_secret` ：客户端密码，**此处不能是明文，需要加密** `$2a$10$VcUHxmgnxBTdB9XNdhoOWujC.lFZ0rO1UizqMAS0GU6WAerFviX.a`
- `scope` ：客户端授权范围, 不指定默认不校验范围
- `authorized_grant_types` ：客户端授权类型，支持多个使用逗号分隔 `authorization_code,password,implicit,client_credentials,refresh_token`
- `web_server_redirect_uri` ：服务器回调地址
- `autoapprove` ：false 显示授权点击页，true不显示自动授权

注意: 要使用 `BCryptPasswordEncoder` 为 `client_secret` 客户端密码加密

仅供购买者学习，禁止盗版、转卖、传播课程

对象 | oauth_client_details @securi...

开始事务 | 备注 | 筛选 | 排序 | 导入 | 导出

client_id	mengxuegu-pc
resource_ids	product-server
client_secret	$2a$10$VcUHxmgnxBTdB9XNdhoOWujC.lFZ0rO1UizqMAS0G
scope	all
authorized_grant_ty...	authorization_code,password,implicit,client_credentials,refresl
web_server_redirec...	http://www.mengxuegu.com/
authorities	授权码和密码模式会从UserDetaiService获取 客户端和隐式模式要，加载这个作为用户拥有授权
access_token_validi...	50000 令牌有效时间，默认12小时
refresh_token_validi...	刷新令牌时间，默认30天
additional_informat...	预留字段
autoapprove	false false显示授权页，true不显示页面自动授权

客户端密码加密存储

注意:包名不要写错了

```
package com.mengxuegu.oauth2;


import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.test.context.junit4.SpringRunner;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@RunWith(SpringRunner.class)
@SpringBootTest
public class TestAuthApplication {

    @Autowired
    PasswordEncoder passwordEncoder;
    @Test
    public void testPwd() {
        System.out.println(passwordEncoder.encode("mengxuegu-secret"));
    }
}
```

仅供购买者学习，禁止盗版、转卖、传播课程

客户端管理切换成 JDBC 方式

在 com.mengxuegu.oauth2.server.config.AuthorizationServerConfig 中操作如下:

1. 向容器中添加 JDBC 方式管理客户端信息服务: `JdbcClientDetailsService`
2. 在AuthorizationServerConfig#configure(ClientDetailsServiceConfigurer) 切换成JDBC 方式管理客户端信息

```
1  package com.mengxuegu.oauth2.server.config;
2
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.context.annotation.Bean;
5  import org.springframework.context.annotation.Configuration;
6  import org.springframework.security.authentication.AuthenticationManager;
7  import org.springframework.security.core.userdetails.UserDetailsService;
8  import org.springframework.security.crypto.password.PasswordEncoder;
9  import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;
10 import
   org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdap
   ter;
11 import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;
12 import
   org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigu
   rer;
13 import org.springframework.security.oauth2.provider.ClientDetailsService;
14 import org.springframework.security.oauth2.provider.client.JdbcClientDetailsService;
15 import org.springframework.security.oauth2.provider.code.AuthorizationCodeServices;
16 import org.springframework.security.oauth2.provider.code.JdbcAuthorizationCodeServices;
17 import org.springframework.security.oauth2.provider.token.TokenStore;
18
19 import javax.sql.DataSource;
20
21 /**
22  * 认证服务器配置
23  * @Auther: 梦学谷 www.mengxuegu.com
24  */
25 @Configuration
26 @EnableAuthorizationServer // 开启认证服务器功能
27 public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {
28
29     @Autowired // 在 SpringSecurityBean 添加到容器了
30     private PasswordEncoder passwordEncoder;
31
32     @Autowired // SpringSecurityConfig添加到容器中了
33     private AuthenticationManager authenticationManager;
34     @Autowired
35     private UserDetailsService customUserDetailsService;
36
37     @Autowired // 令牌管理策略
38     private TokenStore tokenStore;
39
40     @Autowired
```

仅供购买者学习，禁止盗版、转卖、传播课程

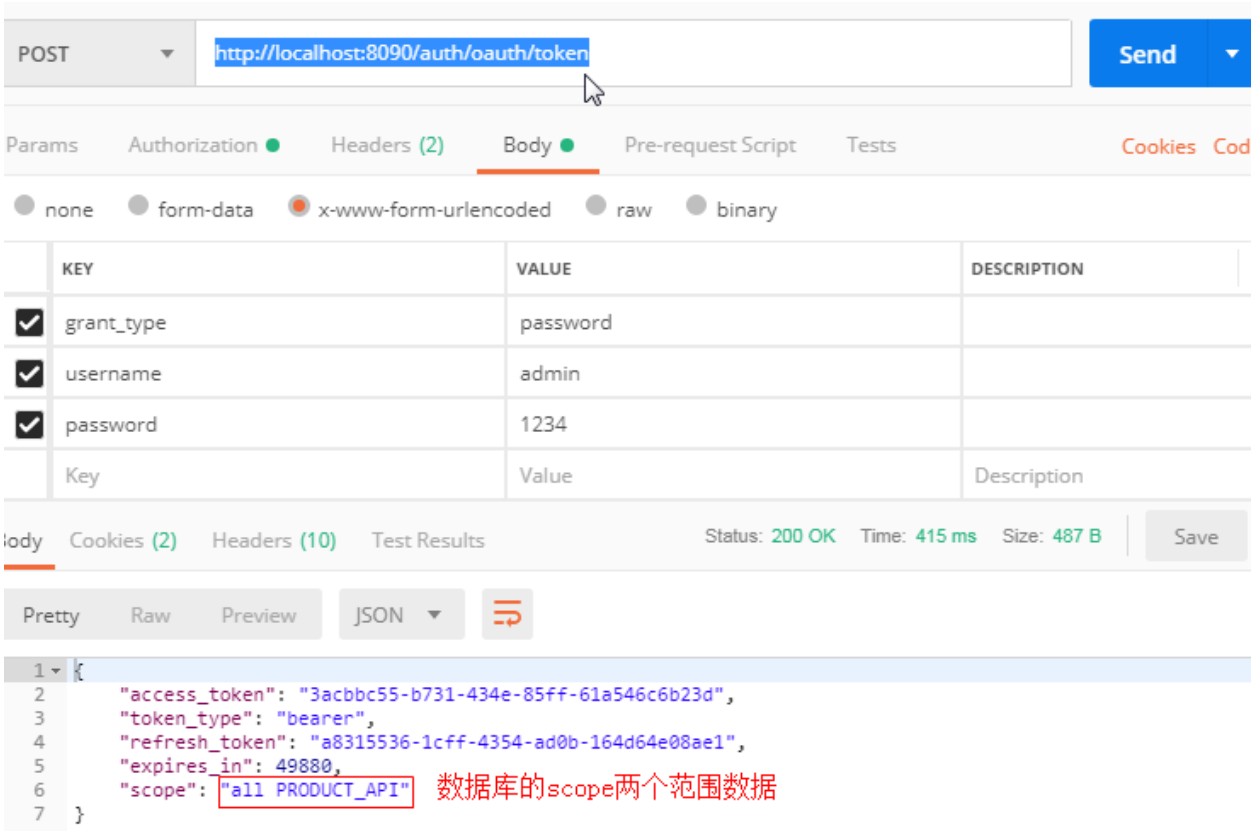
```
41      private DataSource dataSource;
42
43      @Bean // 授权码管理策略
44      public AuthorizationCodeServices jdbcAuthorizationCodeServices() {
45          // JDBC方式保存授权码到 oauth_code 表中,
46          // 意义不大，因为获取一次令牌后，授权码就失效了
47          return new JdbcAuthorizationCodeServices(dataSource);
48      }
49
50  +   @Bean // 注意:方法名为clientDetailsService
51  +   public ClientDetailsService jdbcClientDetailsService() {
52  +       // 使用 JDBC 方式管理客户端信息
53  +       return new JdbcClientDetailsService(dataSource);
54  +   }
55
56      /**
57       * 配置被允许访问此认证服务器的客户端详情信息
58       * 方式1：内存方式管理
59       * 方式2：数据库管理
60       */
61      @Override
62      public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
63          // 使用 JDBC 管理客户端信息
64  +       clients.withClientDetails(jdbcClientDetailsService());
65
66          // 使用内存方式
67          /*clients.inMemory()
68                  .withClient("mengxuegu-pc") // 客户端id
69                  // 客户端密码，要加密,不然一直要求登录, 获取不到令牌, 而且一定不能被泄露
70                  .secret(passwordEncoder.encode("mengxuegu-secret"))
71                  // 资源id, 如商品资源
72                  .resourceIds("product-server")
73                  // 授权类型, 可同时支持多种授权类型
74                  .authorizedGrantTypes("authorization_code", "password",
"implicit","client_credentials","refresh_token")
75                  // 授权范围标识，（all是标识，不是代表所有）哪部分资源可访问
76                  .scopes("all")
77                  // false 跳转到授权页面手动点击授权，true 不用手动授权，直接响应授权码，
78                  .autoApprove(false)
79                  .redirectUris("http://www.mengxuegu.com/"); // 客户端回调地址*/
80
81      }
82
83      /**
84      * 认证服务端点配置
85      */
86      @Override
87      public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
88          // 密码模式要设置认证管理器
89          endpoints.authenticationManager(authenticationManager);
90          // 刷新令牌获取新令牌时需要
91          endpoints.userDetailsService(customUserDetailsService);

92          // 令牌管理策略
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
        endpoints.tokenStore(tokenStore);
        // 授权码管理策略，针对授权码模式有效，会将授权码放到 auth_code 表，授权后就会删除它
        endpoints.authorizationCodeServices(jdbcAuthorizationCodeServices());

    }

}
```

测试

1. 重启认证服务器
2. 使用 admin 用户获取令牌看是否正常, 可以把数据库中scope值更改下，看是不是响应修改后的



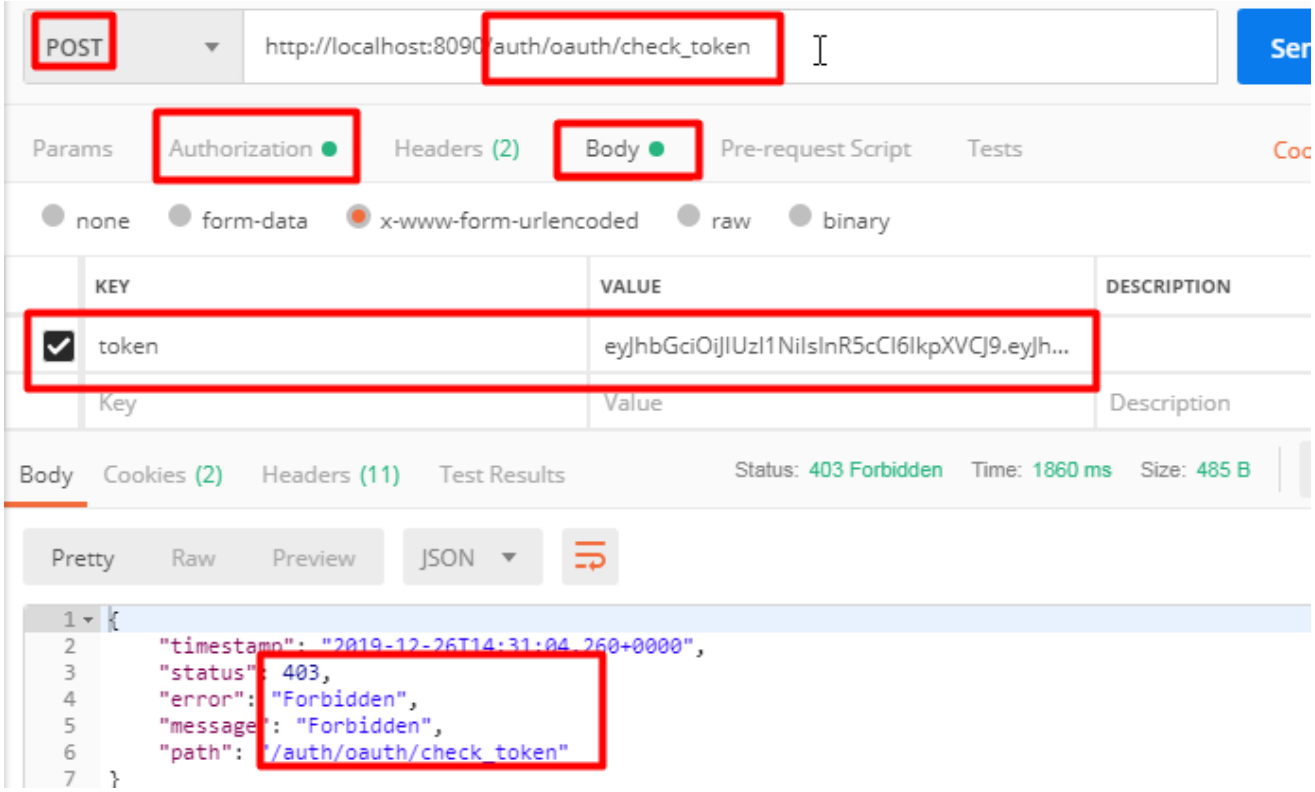
3.5 令牌端点的安全策略

端点 403 不允许访问

- `/oauth/authorize` ：申请授权码 code，涉及的类 `AuthorizationEndpoint`
- `/oauth/token` ：获取令牌 token，涉及的类 `TokenEndpoint`
- `/oauth/check_token` ：用于资源服务器请求端点来检查令牌是否有效，涉及的类 `CheckTokenEndpoint`
- `/oauth/confirm_access` ：用户确认授权提交，涉及的类 `WhitelabelApprovalEndpoint`
- `/oauth/error` ：授权服务错误信息，涉及的类 `WhitelabelErrorEndpoint`
- `/oauth/token_key` ：提供公有密匙的端点，使用 JWT 令牌时会使用，涉及的类 `TokenKeyEndpoint`

仅供购买者学习，禁止盗版、转卖、传播课程

1. 默认情况下 `/oauth/check_token` 和 `/oauth/token_key` 端点默认是 `denyAll()` 拒绝访问的权限，
   要将这两个端点认证或授权后可以访问，因为后面资源服务器，要通过此端点检验令牌是否有效。
2. 请求头还是要设置 `client_id:client_secret` Base64编码



配置端点权限

指定 `isAuthenticated()` 认证后可以访问 `/oauth/check_token` 端点,

指定 `permitAll()` 所有人可访问 `/oauth/token_key` 端点，后面要获取公钥。

在 AuthorizationServerConfig 覆盖 configure(AuthorizationServerSecurityConfigurer security) 方法如下：

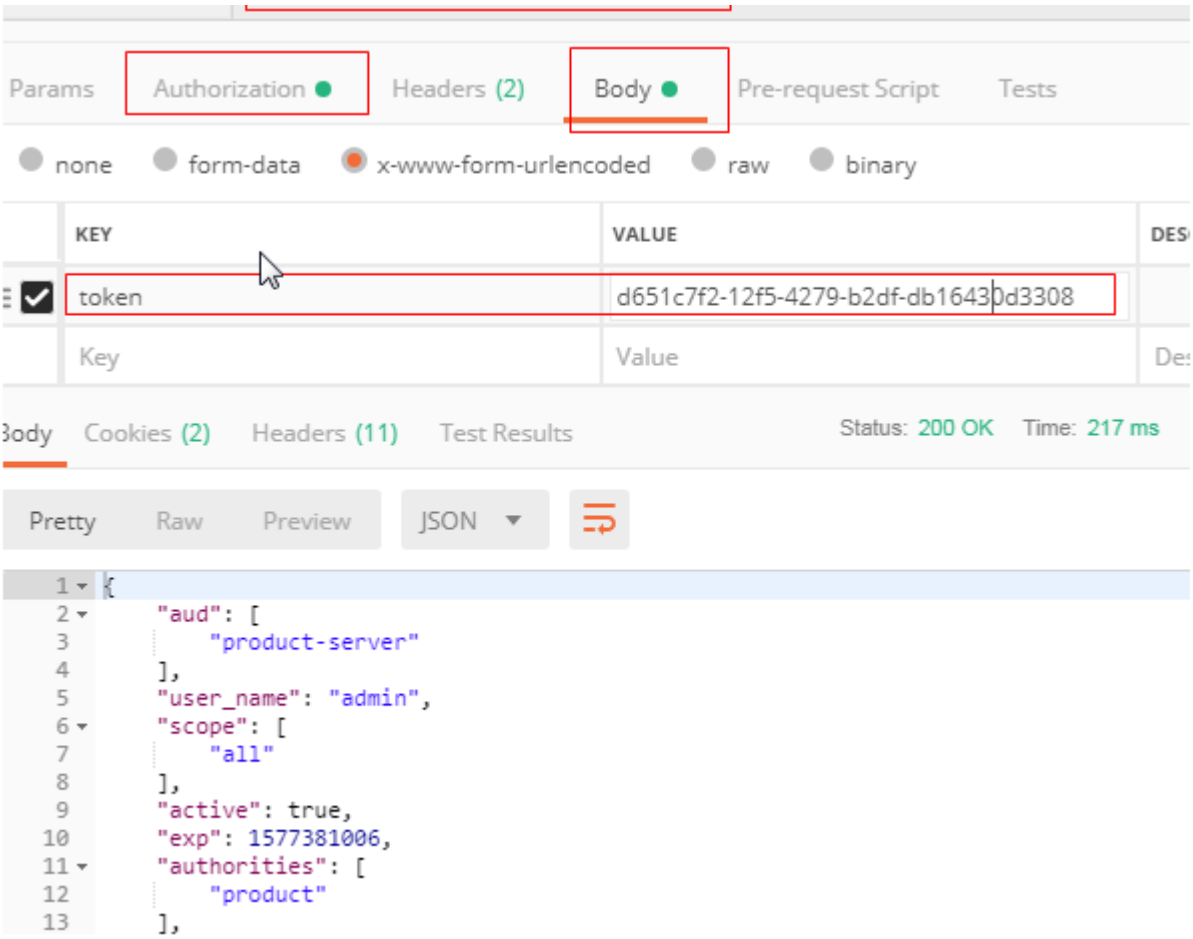
```
/**
 * 令牌端点的安全配置
 */
@Override
public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
    // 所有人可访问 /oauth/token_key 后面要获取公钥, 默认拒绝访问
    security.tokenKeyAccess("permitAll()");
    // 认证后可访问 /oauth/check_token , 默认拒绝访问
    security.checkTokenAccess("isAuthenticated()");
}
```

测试检查令牌端点

1. 重启认证服务器

仅供购买者学习，禁止盗版、转卖、传播课程

2. 重新获取令牌
3. 检查令牌对应的用户信息

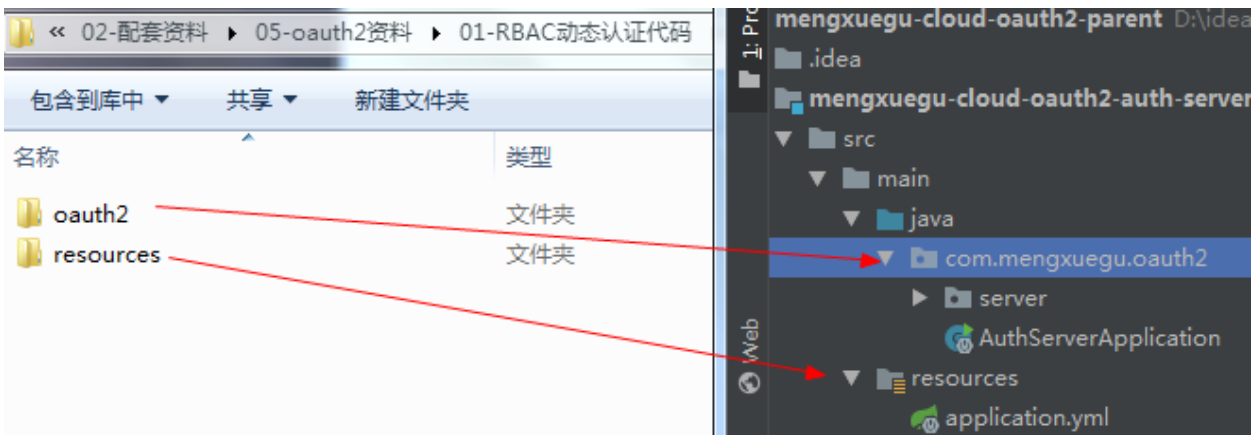


3.6 基于 RBAC 动态认证用户

准备工作

因为之前 Spring Seucrity 中讲过，所以拷贝相关代码，注意：你的包名一定和老师的一致，不然自己手动导包。

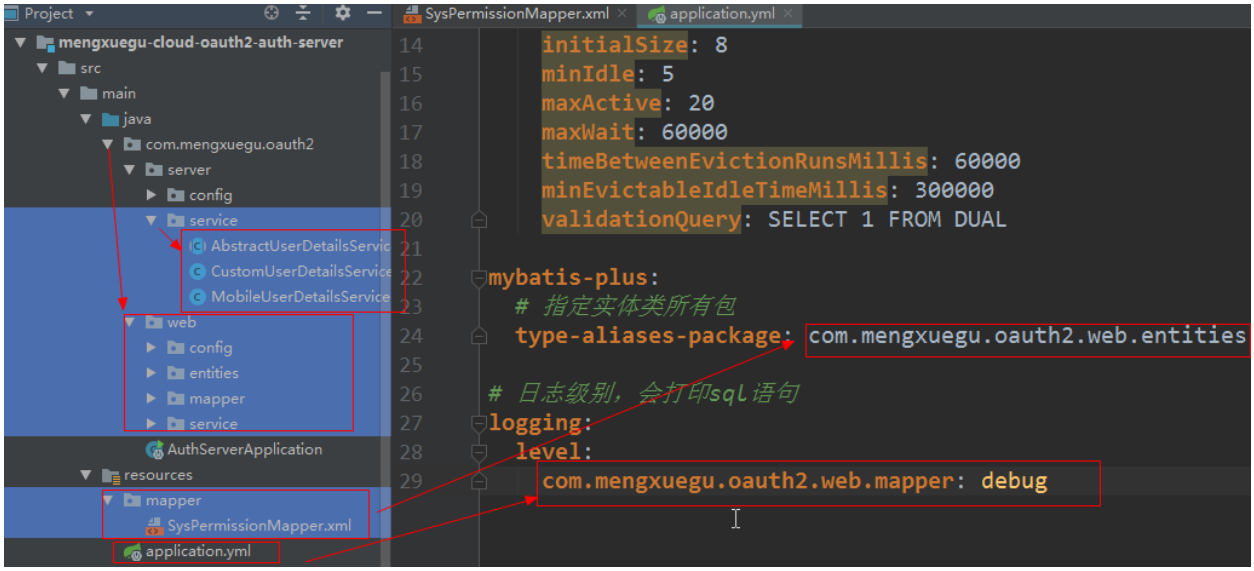
1. 找到 `02-配套资料\05-oauth2资料\01-RBAC动态认证代码` 目录文件：



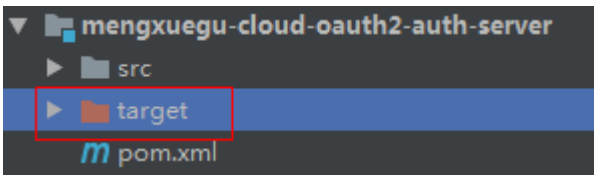
2. 将上面 `resources` 目录覆盖认证服务工程的 resources 目录文件

仅供购买者学习，禁止盗版、转卖、传播课程

3. 将上面 `oauth2` 目录粘贴到 com.mengxuegu.oauth2 包下面



4. **把 target 目录删除，然后按 ctrl + f9 编译**



5. **重启看看是否正常**

   注意：如果你的包结构和老师的不一样，自己修改，特别注意要修改的地方：

   - **com.mengxuegu.oauth2.web.config.MybatisPlusConfig 扫描Mapper的路径**
   - **SysPermissionMapper.xml 指定的 SysPermissionMapper 的全路径**
   - **application.yml 里面 entities 和 mpper 所在包**
   - **其他自行脑补调试**

核查安全配置类

- 现在 com.mengxuegu.oauth2.server.service.CustomUserDetailsService 已经是真实查询用户信息和权限信息进行认证了。
- 检查 com.mengxuegu.oauth2.server.config.SpringSecurityConfig 中是以 CustomUserDetailsService 数据库管理用户

```
    @Autowired
    private UserDetailsService customUserDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // 内存方式存储用户信息
//      auth.inMemoryAuthentication().withUser("admin")
//              .password(passwordEncoder.encode("1234")).authorities("product");
        auth.userDetailsService(customUserDetailsService);
    }
```

仅供购买者学习，禁止盗版、转卖、传播课程

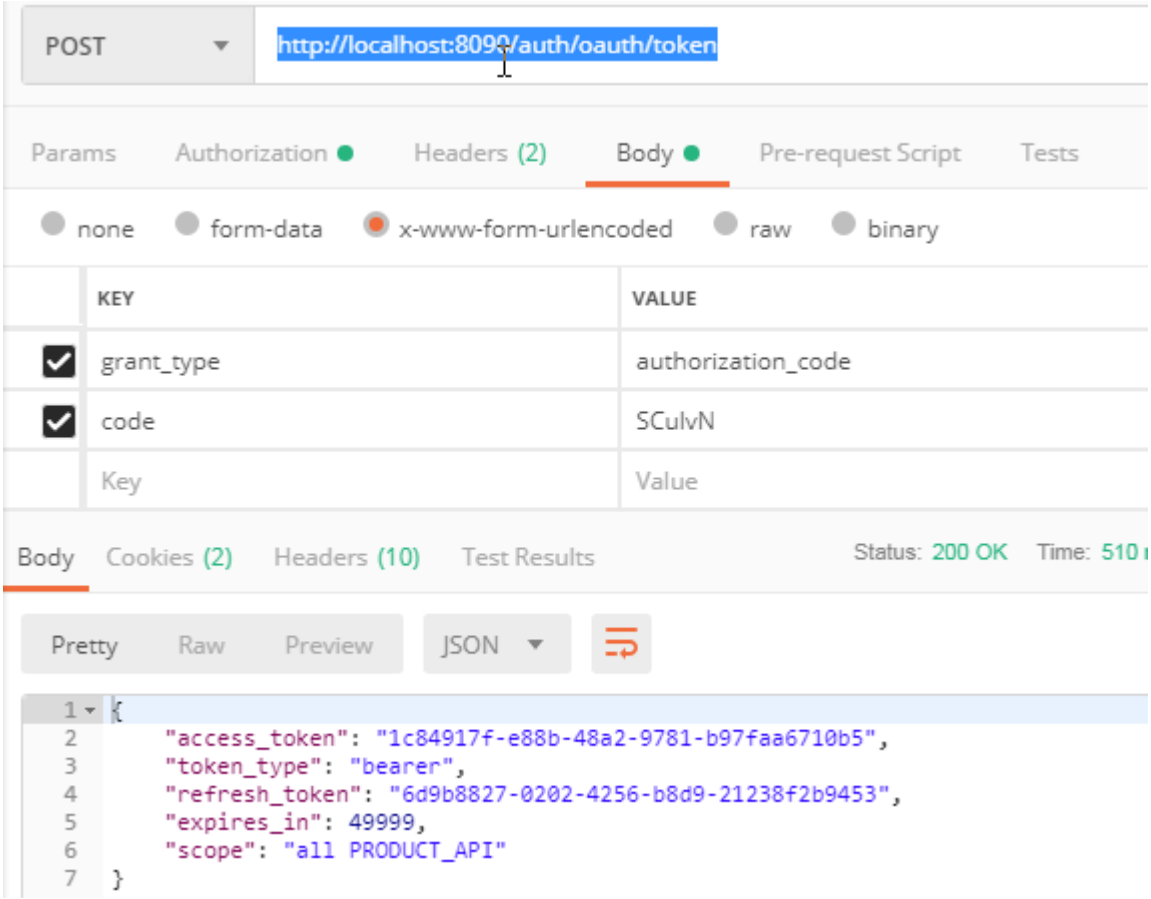
测试

1. 重启认证服务器
2. 请求获取授权码：
   http://localhost:8090/auth/oauth/authorize?client_id=mengxuegu-pc&response_type=code
3. 输入 `t_user` 表中有的用户名 `test` ，密码 `1234`

4. 进入同意授权页面

5. 通过授权码 `XE8Jp6` 获取令牌

仅供购买者学习，禁止盗版、转卖、传播课程



注意：其他定制操作就跟前面 Spring Security 方式一样, 如修改登录页面

第四章 Spring Security OAuth2 资源服务器

4.1 概述

资源服务器实际上就是对系统功能的增删改查，比如：商品管理、订单管理、积分管理、会员管理等资源，而在微服务架构中，而这每个资源实际上就是每一个微服务。当用户请求某个微服务资源时，首先通过认证服务器进行认证与授权，通过后再才可访问到对应资源。

实现的功能：

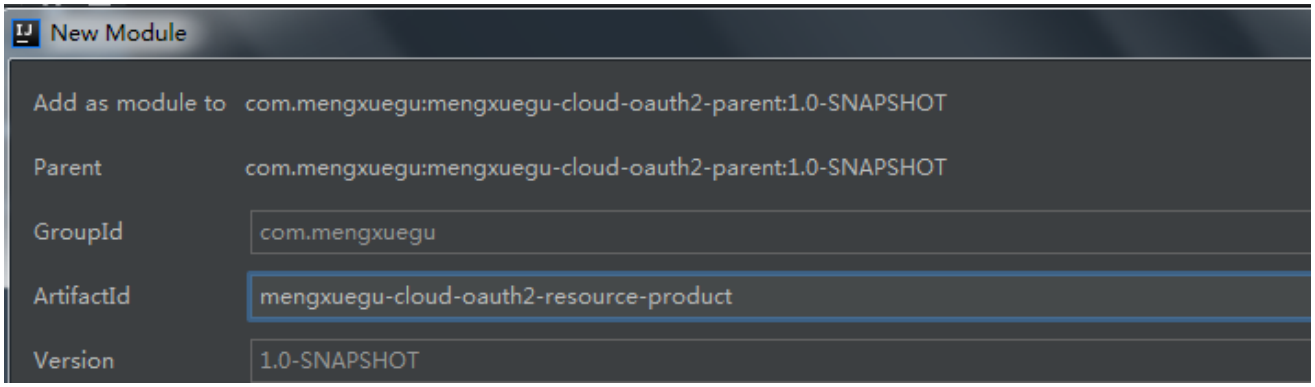
1. 要让他知道自己是资源服务器，他知道这件事后，才会在前边加一个过滤器去验令牌（配置 @EnableResourceServer 配置类）
2. 要让他知道自己是什么资源服务器（配置资源服务器ID），配置去哪里验令牌，怎么验令牌,要带什么信息去验
3. 进行资源的安全配置，让他知道资源的每个访问权限是什么

4.2 创建商品资源模块

右击 mengxuegu-cloud-oauth2-parent 创建Mobule，

模块名为：`mengxuegu-cloud-oauth2-resource-product`

仅供购买者学习，禁止盗版、转卖、传播课程



4.3 添加依赖 pom.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
   4.0.0.xsd">
5      <parent>
6          <artifactId>mengxuegu-cloud-oauth2-parent</artifactId>
7          <groupId>com.mengxuegu</groupId>
8          <version>1.0-SNAPSHOT</version>
9      </parent>
10     <modelVersion>4.0.0</modelVersion>
11
12     <artifactId>mengxuegu-cloud-oauth2-resource-product</artifactId>
13
14     <dependencies>
15         <dependency>
16             <groupId>com.mengxuegu</groupId>
17             <artifactId>mengxuegu-cloud-oauth2-base</artifactId>
18             <version>${mengxuegu-security.version}</version>
19         </dependency>
20         <!--spring mvc相关的-->
21         <dependency>
22             <groupId>org.springframework.boot</groupId>
23             <artifactId>spring-boot-starter-web</artifactId>
24         </dependency>
25         <!-- Spring Security、OAuth2 和JWT等 -->
26         <dependency>
27             <groupId>org.springframework.cloud</groupId>
28             <artifactId>spring-cloud-starter-oauth2</artifactId>
29         </dependency>
30
31         <!-- 注册到 Eureka
32         <dependency>
33             <groupId>org.springframework.cloud</groupId>
34             <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
35         </dependency>
36         -->
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
37
38        <!-- springboot 单元测试 -->
39        <dependency>
40            <groupId>org.springframework.boot</groupId>
41            <artifactId>spring-boot-starter-test</artifactId>
42        </dependency>
43        <!--热部署 ctrl+f9-->
44        <dependency>
45            <groupId>org.springframework.boot</groupId>
46            <artifactId>spring-boot-devtools</artifactId>
47        </dependency>
48
49    </dependencies>
50
51 </project>
```

4.4 创建启动类

```
1  package com.mengxuegu.oauth2;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  /**
7   * 资源服务器启动类
8   * @Auther: 梦学谷 www.mengxuegu.com
9   */
10 @SpringBootApplication
11 public class ProductResourceApplication {
12     public static void main(String[] args) {
13         SpringApplication.run(ProductResourceApplication.class, args);
14     }
15 }
```

4.5 创建商品资源

创建商品资源访问类 `com.mengxuegu.oauth2.web.controller.ProductController` ，

并配置访问权限 `product:list`

```
1  package com.mengxuegu.oauth2.web.controller;
2
3  import com.mengxuegu.base.result.MengxueguResult;
4  import org.springframework.security.access.prepost.PreAuthorize;
5  import org.springframework.web.bind.annotation.GetMapping;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RestController;
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
import java.util.ArrayList;
import java.util.List;

@RestController
@RequestMapping("/product")
public class ProductController {

    @GetMapping("/list")
    @PreAuthorize("hasAuthority('product:list')")
    public MengxueguResult list() {
        List<String> list = new ArrayList<>();
        list.add("眼镜");
        list.add("格子衬衣");
        list.add("双肩包");
        return MengxueguResult.ok(list);
    }

}
```

4.6 配置资源服务器

1. 创建 com.mengxuegu.oauth2.resource.ResourceServerConfig 类，然后继承 `ResourceServerConfigurerAdapter` 资源服务器配置适配器
2. 在类上加上以下注解：
   - `@Configuration`
   - `@EnbleResourceServer` ：标识为资源服务器，所有发往这个服务的请求，都会去请求头里找 token，找不到或者通过认证服务器验证不合法，则不允许访问。
   - `@EnableGlobalMethodSecurity(prePostEnabled = true)` ：开启方法级权限控制
3. 重写资源服务器相关配置方法 `configure(ResourceServerSecurityConfigurer resources)`
   - 配置当前资源服务器ID
   - 添加校验令牌服务
     - 创建 `RemoteTokenServices` 远程校验令牌服务，去校验令牌有效性，原因：

       因为当前认证和资源服务器不是在同一工程中，所以要通过远程调用认证服务器校验令牌是否有效。
     - 如果认证和资源服务器在同一工程中，可以使用 DefaultTokenServices 配置校验令牌。

```
package com.mengxuegu.oauth2.resource;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
9  import
org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAda
pter;
10 import
org.springframework.security.oauth2.config.annotation.web.configurers.ResourceServerSecurityConfigur
er;
11 import org.springframework.security.oauth2.provider.token.RemoteTokenServices;
12 import org.springframework.security.oauth2.provider.token.ResourceServerTokenServices;
13
14 /**
15  * 资源服务器相关配置
16  * @Auther: 梦学谷 www.mengxuegu.com
17  */
18 @Configuration
19 @EnableResourceServer // 标识为资源服务器, 所有发往当前服务的请求，都会去请求头里找token，找不到或
验证不通过不允许访问
20 @EnableGlobalMethodSecurity(prePostEnabled = true) // 开启方法级权限控制
21 public class ResourceServerConfig extends ResourceServerConfigurerAdapter {
22
23     //配置当前资源服务器的ID
24     public static final String RESOURCE_ID = "product-server";
25
26     /**
27      * 当前资源服务器的一些配置, 如 资源服务器ID
28      * @param resources
29      * @throws Exception
30      */
31     @Override
32     public void configure(ResourceServerSecurityConfigurer resources) throws Exception {
33         resources.resourceId(RESOURCE_ID) // 配置当前资源服务器的ID, 会在认证服务器验证(客户端表的
resources配置了就可以访问这个服务)
34                 .tokenServices(tokenService()); // 实现令牌服务, ResourceServerTokenServices实例
35     }
36
37     /**
38      * 配置资源服务器如何验证token有效性
39      * 1. DefaultTokenServices
40      *   如果认证服务器和资源服务器同一服务时,则直接采用此默认服务验证即可
41      * 2. RemoteTokenServices (当前采用这个)
42      *   当认证服务器和资源服务器不是同一服务时, 要使用此服务去远程认证服务器验证
43      */
44     @Bean
45     public ResourceServerTokenServices tokenService() {
46         // 资源服务器去远程认证服务器验证 token 是否有效
47         RemoteTokenServices service = new RemoteTokenServices();
48         // 请求认证服务器验证URL，注意：默认这个端点是拒绝访问的，要设置认证后可访问
49         service.setCheckTokenEndpointUrl("http://localhost:8090/oauth/check_token");
50         // 在认证服务器配置的客户端id
51         service.setClientId("mengxuegu-pc");
52         // 在认证服务器配置的客户端密码
53         service.setClientSecret("mengxuegu-secret");
54         return service;

55     }
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
}
```

4. 核实下是否已经配置了放行校验令牌端点 `/oauth/check_token`

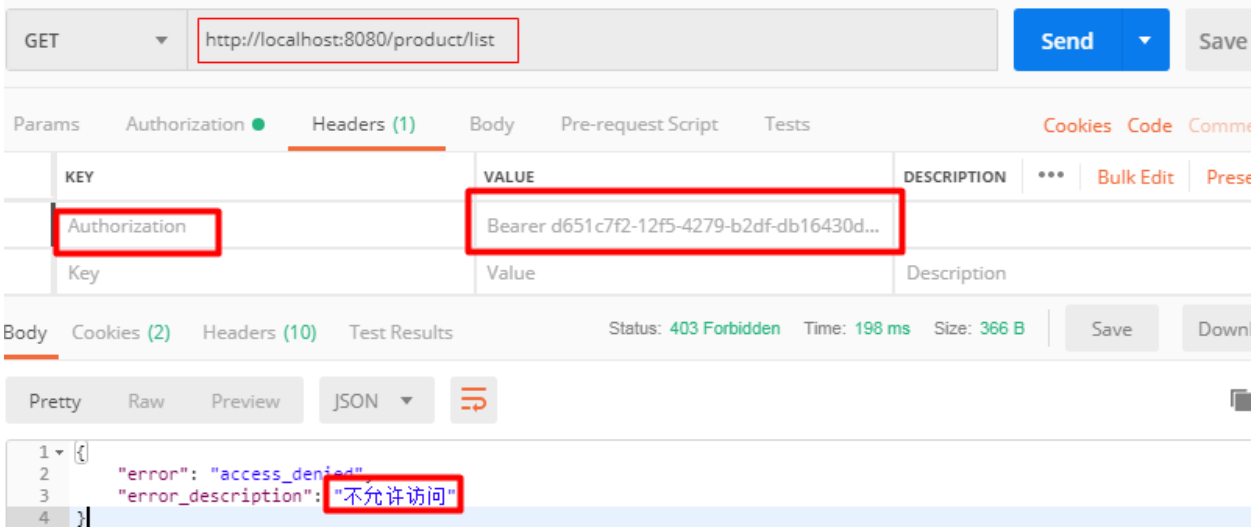
> 在认证服务器 `ResourceServerConfig` 已经配置过, 核实下保证已经配置了，如下:

```java
/**
 * 令牌端点的安全配置
 */
@Override
public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
    // 所有人可访问 /oauth/token_key 后面要获取公钥, 默认拒绝访问
    security.tokenKeyAccess("permitAll()");
    // 认证后可访问 /oauth/check_token , 默认拒绝访问
    security.checkTokenAccess("isAuthenticated()");
}
```

4.7 测试

1. 启动认证服务器和资源服务器
2. 通过密码方式获取令牌
3. 请求头带上令牌请求 `/product/list` 资源，发现响应不允许访问：

   请求头： Key: `Authorization` Value: `Bearer 令牌字符串`



   原因：当前用户令牌没有访问 `/product/list` 资源的权限 `product:list`

仅供购买者学习，禁止盗版、转卖、传播课程

```java
public class ProductController {

    @GetMapping("/list")
    @PreAuthorize("hasAuthority('product:list')")
    public MengxueguResult list() {
        List<String> list = new ArrayList<>();
        list.add("眼镜");
        list.add("格子衬衣");
        list.add("双肩包");
        return MengxueguResult.ok(list);
    }
```

如果没有权限，会提示 '不允许访问' ，

解决：要给用户授权才行，就是RBAC相关表中添加对应权限数据

Body Cookies (2) Headers (10) Test Results Status: 403

Pretty Raw Preview JSON

```
1 {
2     "error": "access_denied",
3     "error_description": "不允许访问"
4 }
```

4.8 控制令牌范围权限和授权规则

1. 资源服务器通过 `ResourceServerConfigurerAdapter#configure(HttpSecurity http)` 指定授权规则。
    - 禁用 `Session` ，因为是基于 token 认证，所以不需要 HttpSession 了
    - 指定资源的授权规则，与 SpringSecurity 中的指定方式一样
    - 用 `#oauth2` 表达式控制令牌范围 scope，如果令牌的没有对应scope权限，则对应资源不允许访问

2. 在 `ResourceServerConfig` 添加如下代码：

仅供购买者学习，禁止盗版、转卖、传播课程

```java
@Override
public void configure(HttpSecurity http) throws Exception {
    http.sessionManagement()
            // SpringSecurity 不会创建也不会使用 HttpSession
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
        .and()
            .authorizeRequests()
            // 资源授权规则
            .antMatchers("/product/**").hasAuthority("product")
            // 所有的请求对应访问的用户都要有 all 范围权限
            .antMatchers("/**").access("#oauth2.hasScope('all')")
        ;
}
```

4.9 测试

1. 当前用户令牌的范围是 `all` ，而资源服务器要求是 `xxxxx` 范围

```java
ResourceServerConfig.java
     * @throws Exception
     */
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.sessionManagement() SessionManagementConfigurer<HttpSecurity>
                // SpringSecurity 不会创建也不会使用 HttpSession
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS) SessionManagementCon
            .and() HttpSecurity
                .authorizeRequests() ExpressionInterceptUrlRegistry
                // 用 #oauth2表达式控制令牌范围是否为 `all`，不是则所有资源不允许访问
                .antMatchers( ...antPatterns: "/**").access( attribute: "#oauth2.hasScope('xxxxx')")
        ;
    }
```

不是all

如果范围错误时，会提示 `Insufficient scope for this resource` ，

解决：在 oauth_client_details 表为 `mengxuegu-pc` 客户端加上 `xxxx` 范围，多个逗号分隔

Pretty Raw Preview JSON

```{
    "error": "insufficient_scope",
    "error_description": "Insufficient scope for this resource",
    "scope": "xxxxx",
}
```

第五章 JWT 令牌

5.1 JWT 解决什么问题

当认证服务器和资源服务器不是在同一工程时，要使用 `ResourceServerTokenServices` 去远程请求认证服务器来校验令牌的合法性，如果用户访问量较大时将会影响系统的性能。

解决方式：

仅供购买者学习，禁止盗版、转卖、传播课程

生成令牌采用 JWT 格式就可以解决上面的问题。

因为当用户认证后获取到一个JWT令牌，而这个 JWT 令牌包含了用户基本信息，客户端只需要携带JWT访问资源服务器，资源服务器会通过事先约定好的算法进行解析出来，然后直接对 JWT 令牌校验，不需要每次远程请求认证服务器完成授权。

5.2 JWT 是什么

JSON Web Token（JWT）是一个开放的行业标准（RFC 7519），它定义了一种紧凑且独立的方式，用于在各方之间作为JSON对象安全地传输信息。此信息可以通过数字签名进行验证和信任。JWT可以使用秘密（使用**HMAC**算法）或使用**RSA**或**ECDSA**的公钥/私钥对进行**签名**，防止被篡改。

JWT 官网： https://jwt.io 想深入了解的可网站查看

JWT 的构成：

JWT 有三部分构成：头部、有效载荷、签名

> 例如：aaaaa.bbbbbb.cccccccc

- 头部：包含令牌的类型（JWT）与加密的签名算法（（如 SHA256 或 ES256），Base64编码后加入第一部分
- 有效载荷：通俗一点讲就是token中需要携带的信息都将存于此部分，比如：用户id、权限标识等信息。

  > 注：该部分信息任何人都可以读出来，所以添加的信息需要加密才会保证信息的安全性

- 签名：用于防止 JWT 内容被篡改, 会将头部和有效载荷分别进行 Base64编码，编码后用 `.` 连接组成新的字符串，然后再使用头部声明的签名算法进行签名。在具有秘钥的情况下，可以验证JWT的准确性，是否被篡改。

5.3 JWT 优缺点

JWT 的优点：

1. JWT 基于 json，非常方便解析。
2. 可以在令牌中自定义丰富的内容，易扩展。
3. 通过非对称加密算法及数字签名技术，JWT 防止篡改，安全性高。
4. 资源服务器使用 JWT 可以不依赖认证服务器，即可完成授权。

JWT 的缺点：

1. JWT令牌较长，占存储空间比较大，但是用户信息是有限的，所以在可接受范围。

5.4 认证服务器-对称加密 JWT 令牌

对称加密就是同一个密钥可以同时用作信息的加密和解密，这种加密方法称为对称加密，也称为单密钥加密。

5.4.1 JWT 管理令牌

仅供购买者学习，禁止盗版、转卖、传播课程

1. 重构 com.mengxuegu.oauth2.server.config.TokenConfig 将令牌存储方式切换为 JWT
   - 创建 JwtAccessTokenConverter 实例，定义 JWT 签名密钥
   - 创建 JwtTokenStore 实例，传入 JwtAccessTokenConverter 实例

```
1  package com.mengxuegu.oauth2.server.config;
2
3  import com.alibaba.druid.pool.DruidDataSource;
4  import org.springframework.boot.context.properties.ConfigurationProperties;
5  import org.springframework.context.annotation.Bean;
6  import org.springframework.context.annotation.Configuration;
7  import org.springframework.security.oauth2.provider.token.TokenStore;
8  import org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter;
9  import org.springframework.security.oauth2.provider.token.store.JwtTokenStore;
10
11 import javax.sql.DataSource;
12
13 /**
14  * @Auther: 梦学谷 www.mengxuegu.com
15  */
16 @Configuration
17 public class TokenConfig {
18
19     /**
20      * Redis 管理令牌
21      * 1. 启动 redis 服务器
22      * 2. 添加 redis 相关依赖
23      * 3. 添加redis 依赖后, 容器就会有 RedisConnectionFactory 实例
24      */
25     // @Autowired
26     // private RedisConnectionFactory redisConnectionFactory;
27
28     /**
29      * JDBC 管理令牌
30      * 1. 创建相关数据表
31      * 2. 添加 jdbc 相关依赖
32      * 3. 配置数据源信息
33      */
34     @Bean
35     @ConfigurationProperties(prefix = "spring.datasource")
36     public DataSource dataSource() {
37         return new DruidDataSource();
38     }
39
40     // JWT 签名秘钥
41     private static final String SIGNING_KEY = "mengxuegu-key";
42
43      @Bean // 在 JwtAccessTokenConverter 中定义 Jwt 签名密码
44      public JwtAccessTokenConverter jwtAccessTokenConverter() {
45          JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
46          // 对称密钥来签署我们的令牌，资源服务器也将使用此秘钥来验证准码性
47          converter.setSigningKey(SIGNING_KEY);

48          return converter;
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
    }

    @Bean
    public TokenStore tokenStore() {
        // Redis 管理令牌
        // return new RedisTokenStore(redisConnectionFactory);
        // JDBC 管理令牌
        // return new JdbcTokenStore(dataSource());
        // JWT 管理令牌，注意是 jwt, 是 jwk
        return new JwtTokenStore(jwtAccessTokenConverter()); // 参数不要少了括号
    }

}
```

5.4.2 JWT转换器添加到令牌端点

重构 com.mengxuegu.oauth2.server.config.AuthorizationServerConfig 认证服务器配置类

1. 注入 JwtAccessTokenConverter

```
@Autowired // jwt令牌
private JwtAccessTokenConverter jwtAccessTokenConverter;
```

2. 设置到令牌端点上 configure(AuthorizationServerEndpointsConfigurer endpoints)

```
+   @Autowired // jwt令牌
+   private JwtAccessTokenConverter jwtAccessTokenConverter;

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        // 密码模式要设置认证管理器
        endpoints.authenticationManager(authenticationManager);
        // 刷新令牌获取新令牌时需要
        endpoints.userDetailsService(customUserDetailsService);
        // 令牌管理策略
+       endpoints.tokenStore(tokenStore) // 后面没有分号
+               .accessTokenConverter(jwtAccessTokenConverter); // jwt令牌
        // 授权码管理策略，针对授权码模式有效，会将授权码放到 auth_code 表，授权后就会删除它
        endpoints.authorizationCodeServices(jdbcAuthorizationCodeServices());
    }
```

3. AuthorizationServerConfig 完整代码

```
package com.mengxuegu.oauth2.server.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
7  import org.springframework.security.core.userdetails.UserDetailsService;
8  import org.springframework.security.crypto.password.PasswordEncoder;
9  import
   org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;
10 import
   org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigure
   rAdapter;
11 import
   org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;
12 import
   org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsC
   onfigurer;
13 import
   org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecurityCon
   figurer;
14 import org.springframework.security.oauth2.provider.ClientDetailsService;
15 import org.springframework.security.oauth2.provider.client.JdbcClientDetailsService;
16 import org.springframework.security.oauth2.provider.code.AuthorizationCodeServices;
17 import org.springframework.security.oauth2.provider.code.JdbcAuthorizationCodeServices;
18 import org.springframework.security.oauth2.provider.token.TokenStore;
19 import org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter;
20
21 import javax.sql.DataSource;
22
23 /**
24  * 认证服务器配置
25  * @Auther: 梦学谷 www.mengxuegu.com
26  */
27 @Configuration
28 @EnableAuthorizationServer // 开启认证服务器功能
29 public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {
30
31     @Autowired // 在 SpringSecurityBean 添加到容器了
32     private PasswordEncoder passwordEncoder;
33
34     @Autowired // SpringSecurityConfig添加到容器中了
35     private AuthenticationManager authenticationManager;
36     @Autowired
37     private UserDetailsService customUserDetailsService;
38
39     @Autowired // 令牌管理策略
40     private TokenStore tokenStore;
41
42     @Autowired
43     private DataSource dataSource;
44
45     @Bean // 授权码管理策略
46     public AuthorizationCodeServices jdbcAuthorizationCodeServices() {
47         // JDBC方式保存授权码到 oauth_code 表中,
48         // 意义不大，因为获取一次令牌后，授权码就失效了
49         return new JdbcAuthorizationCodeServices(dataSource);
50     }
51
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
52      @Bean // 注意:方法名不能为clientDetailsService, 因为容器中已存在一个
53      public ClientDetailsService jdbcClientDetailsService() {
54          // 使用 JDBC 方式管理客户端信息
55          return new JdbcClientDetailsService(dataSource);
56      }
57
58      /**
59       * 配置被允许访问此认证服务器的客户端详情信息
60       * 方式1：内存方式管理
61       * 方式2：数据库管理
62       */
63      @Override
64      public void configure(ClientDetailsServiceConfigurer clients) throws Exception{
65          // 使用 JDBC 管理客户端信息
66          clients.withClientDetails(jdbcClientDetailsService());
67
68      }
69
70      @Autowired // jwt令牌
71      private JwtAccessTokenConverter jwtAccessTokenConverter;
72
73      @Override
74      public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
75          // 密码模式要设置认证管理器
76          endpoints.authenticationManager(authenticationManager);
77          // 刷新令牌获取新令牌时需要
78          endpoints.userDetailsService(customUserDetailsService);
79          // 令牌管理策略
80          endpoints.tokenStore(tokenStore) // 后面没有分号
81                  .accessTokenConverter(jwtAccessTokenConverter); // jwt令牌
82          // 授权码管理策略，针对授权码模式有效，会将授权码放到 auth_code 表，授权后就会删除它
83          endpoints.authorizationCodeServices(jdbcAuthorizationCodeServices());
84      }
85
86      /**
87       * 令牌端点的安全配置
88       */
89      @Override
90      public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
91          // 所有人可访问 /oauth/token_key 后面要获取公钥, 默认拒绝访问
92          security.tokenKeyAccess("permitAll()");
93          // 认证后可访问 /oauth/check_token , 默认拒绝访问
94          security.checkTokenAccess("isAuthenticated()");
95      }
96  }
```

5.4.3 测试

1. 重启 认证服务器
2. 使用密码模式获取令牌，查看响应结果为 JWT 令牌

仅供购买者学习，禁止盗版、转卖、传播课程

POST http://localhost:8090/auth/oauth/token Ser

Params Authorization Headers (2) Body Pre-request Script Tests Coo

none form-data x-www-form-urlencoded raw binary

KEY	VALUE	DESCRIPTION
grant_type	password	
username	admin	
password	1234	
Key	Value	Description

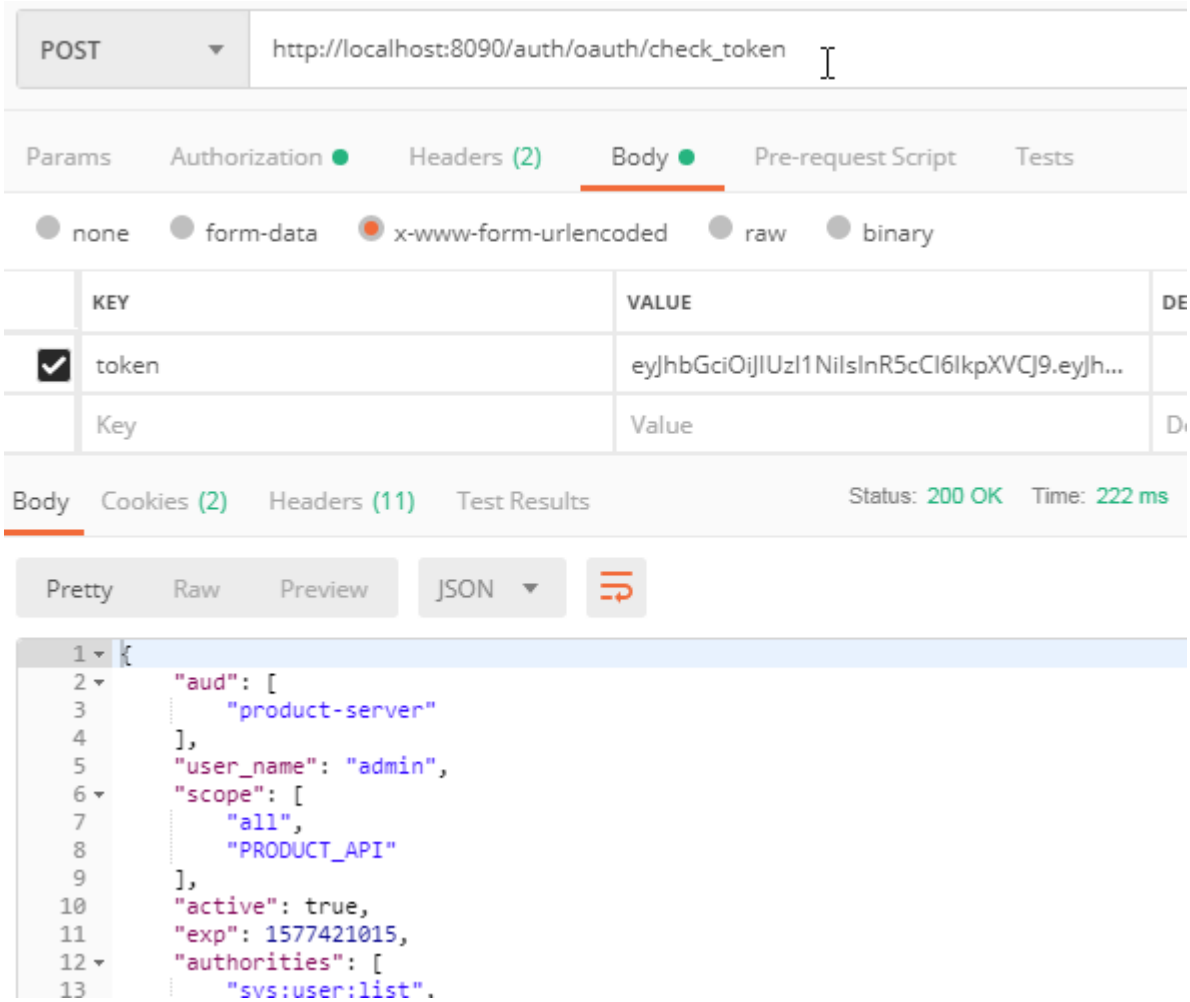
Body Cookies (2) Headers (10) Test Results Status: 200 OK Time: 1321 ms Size: 1.88 KB

Pretty Raw Preview JSON


```
{
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
        .eyJhdWQiOlsicHJvZHVjdC1zZXJ2ZXIiXSwidXNlcl9uYWl1IjoiYWRtaW4iLCJzY29wZSI6WyJhbGwiLCJQUk9
        jE1Nzc0MjEwMTUsImF1dGhvcml0aWVzIjpbInN5czp1c2VyOmxpc3QiLCJzeXM6bWFuYWdlIiwic3lzOnVzZXIiL
        hZGQiLCJzeXM6aW5kZXgiLCJzeXM6dXNlcjphZGQiLCJzeXM6cGVybWlzc2lvbjplZGl0Iiwic3lzOnJvbGU6ZWR
        W9uIiwic3lzOnVzZXI6ZWRpdCIsInN5czp1c2VyOmRlbGV0ZSIsInN5czpyb2xlOmRlbGV0ZSIsInN5czpyb2xlO
        zc2lvbjpkZWxldGUiLCJzeXM6cGVybWlzc2lvbjpsaXN0Iiwic3lzOnJvbGUiLCJzeXM6cm9sZTphZGQiXSwianR
        S00NjU2LTliMjItYzM3MzgxYTc0ODA5IiwiY2xpZW50X2lkIjoibWVuZ3h1ZWd1LXBjIn0.ab6o8BT_MmnGDtAcj
        -MLaw",
    "token_type": "bearer",
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
        .eyJhdWQiOlsicHJvZHVjdC1zZXJ2ZXIiXSwidXNlcl9uYWl1IjoiYWRtaW4iLCJzY29wZSI6WyJhbGwiLCJQUk9
        iT4YmFxOGFvMC01ZWRlLTO2NTYtOWIyMi1iMzczODFhNzO4MDkiLCJleHAiOiE1Nzk5NiMwMTUsImF1dGhvcml0a
```


3. 检查 JWT 令牌，包含了用户信息

仅供购买者学习，禁止盗版、转卖、传播课程



5.5 资源服务器-对称加密 JWT 令牌

在 JwtAccessTokenConverter 中使用了一个对称密钥来签署我们的令牌， 意味着我们需要为资源服务器使用同样的密钥来验证签名合法性。

5.5.1 JWT 管理令牌

这部分与认证服务器令牌配置是一样的, 将 JWT 令牌部分拷贝到 `mengxuegu-cloud-oauth2-resource-product` 中的 com.mengxuegu.oauth2.resource.config 包下即可.

```
package com.mengxuegu.oauth2.resource.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.oauth2.provider.token.TokenStore;
import org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter;
import org.springframework.security.oauth2.provider.token.store.JwtTokenStore;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@Configuration
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
13 public class TokenConfig {
14     // JWT 签名秘钥
15     private static final String SIGNING_KEY = "mengxuegu-key";
16
17     @Bean // 在 JwtAccessTokenConverter 中定义 Jwt 签名密码
18     public JwtAccessTokenConverter jwtAccessTokenConverter() {
19         JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
20         // 对称密钥来签署我们的令牌，资源服务器也将使用此秘钥来验证准码性
21         converter.setSigningKey(SIGNING_KEY);
22         return converter;
23     }
24
25     @Bean
26     public TokenStore tokenStore() {
27         // JWT 管理令牌，注意是 jwt, 是 jwk
28         return new JwtTokenStore(jwtAccessTokenConverter()); // 参数不要少了括号
29     }
30
31 }
```

5.5.2 资源服务器校验 JWT 令牌

重构 com.mengxuegu.oauth2.resource.config.ResourceServerConfig 资源服务器配置类

1. 注入 TokenStore

```
1     @Autowired
2     private TokenStore tokenStore;
```

2. 注释远程校验方法 ResourceServerTokenServices#tokenService() ，configure(ResourceServerSecurityConfigurer resources) 设置 tokenStore，就会自动本地校验jwt令牌

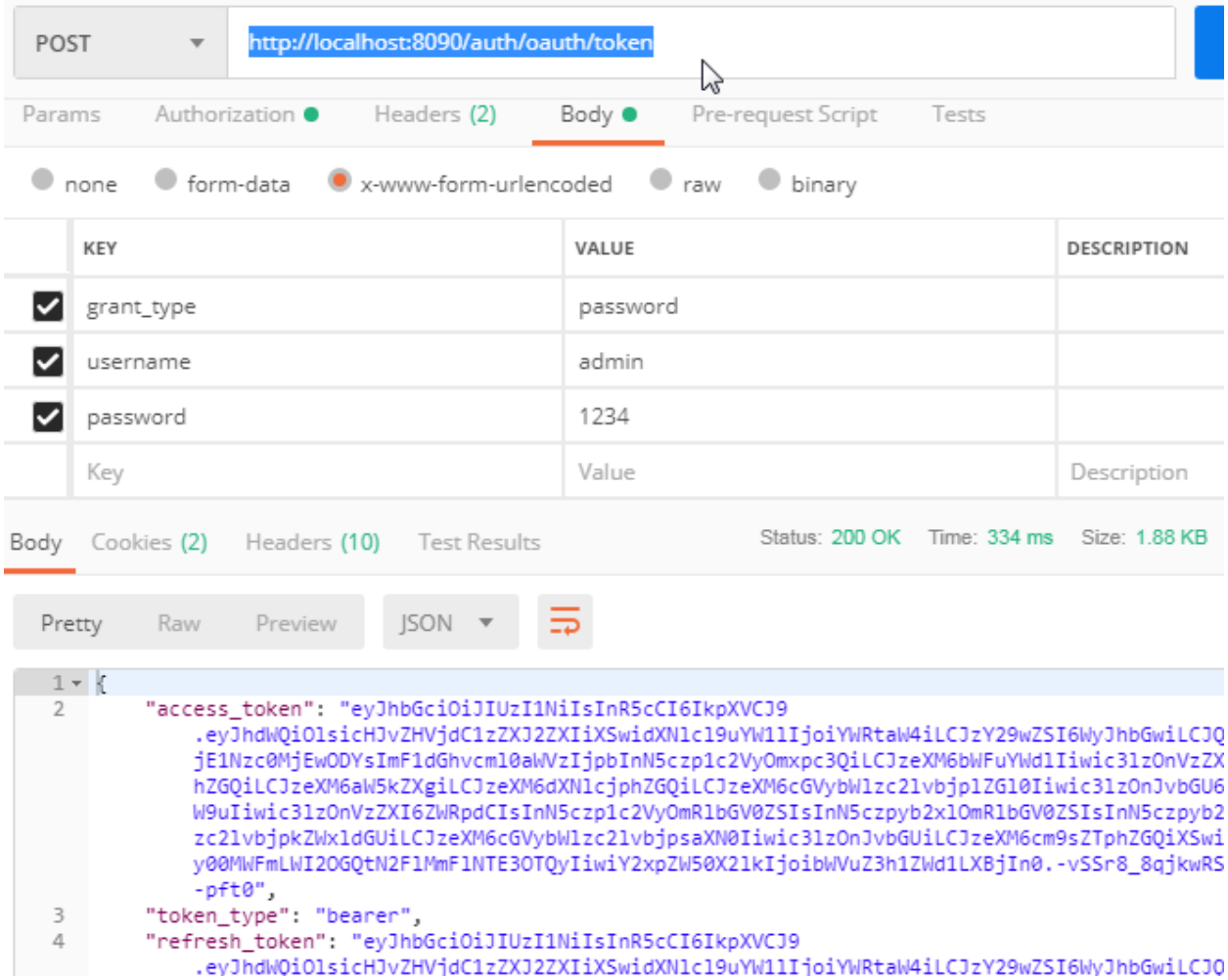
```
1  +  @Autowired
2  +  private TokenStore tokenStore;
3
4     /**
5      * 当前资源服务器的一些配置, 如 资源服务器ID
6      * @param resources
7      * @throws Exception
8      */
9     @Override
10    public void configure(ResourceServerSecurityConfigurer resources) throws Exception {
11        resources.resourceId(RESOURCE_ID)
12 +              .tokenStore(tokenStore);
13 -             //.tokenServices(tokenService());
14    }
15
16    /**
17     * 配置资源服务器如何验证token有效性
```

仅供购买者学习，禁止盗版、转卖、传播课程

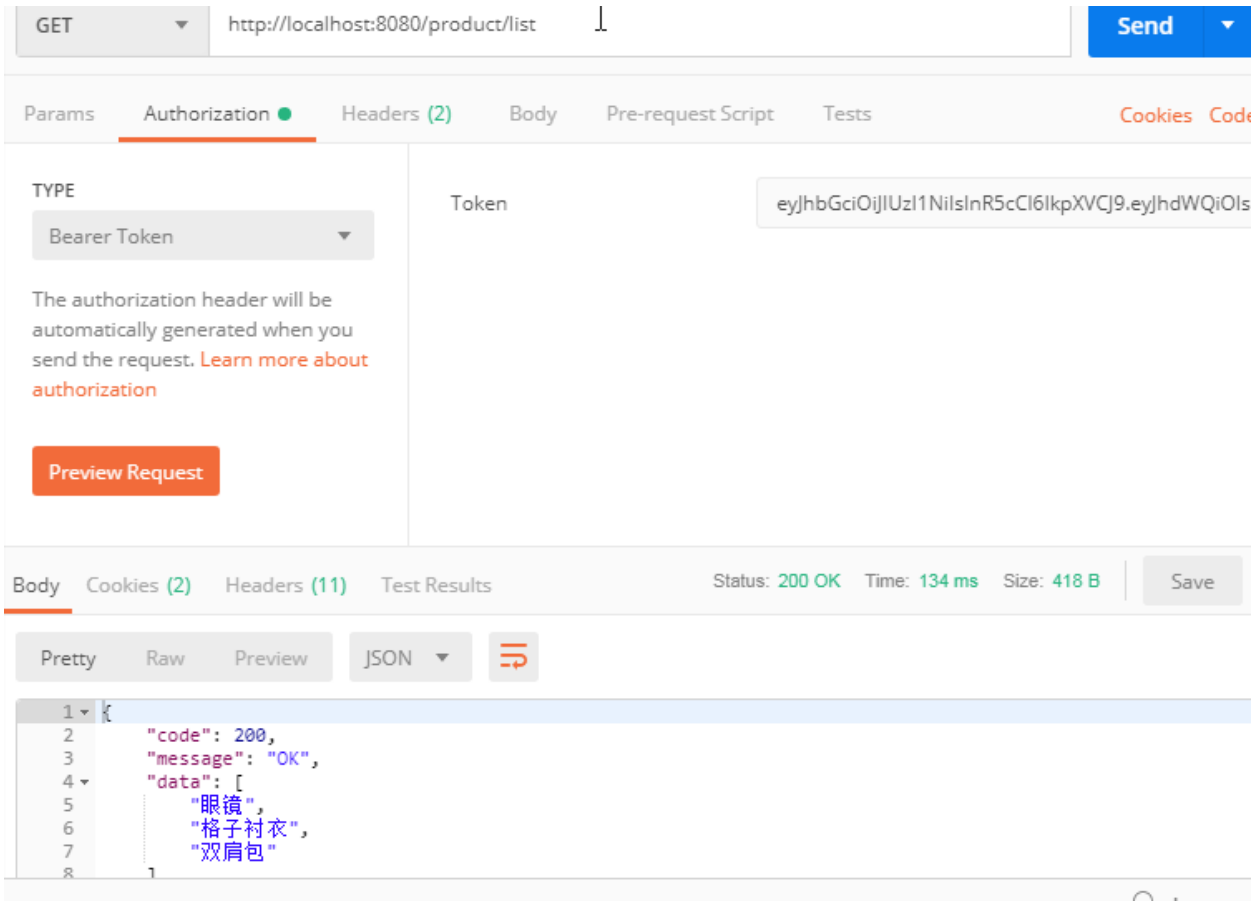
```
18      * 1. DefaultTokenServices
19      *   如果认证服务器和资源服务器同一服务时,则直接采用此默认服务验证即可
20      * 2. RemoteTokenServices (当前采用这个)
21      *   当认证服务器和资源服务器不是同一服务时, 要使用此服务去远程认证服务器验证
22     @Bean
23     public ResourceServerTokenServices tokenService() {
24       // 资源服务器去远程认证服务器验证 token 是否有效
25       RemoteTokenServices service = new RemoteTokenServices();
26       // 请求认证服务器验证URL，注意：默认这个端点是拒绝访问的，要设置认证后可访问
27       service.setCheckTokenEndpointUrl("http://localhost:8090/oauth/check_token");
28       // 在认证服务器配置的客户端id
29       service.setClientId("mengxuegu-pc");
30       // 在认证服务器配置的客户端密码
31       service.setClientSecret("mengxuegu-secret");
32       return service;
33     }*/
```

5.5.3 测试

1. 重启资源服务器
2. 秘密模式获取 JWT 令牌，如下：



3. 通过 JWT令牌查询获取商品资源



4. 为了安全性, JWT令牌每次请求获取令牌都是响应一个新令牌,因为里面已经包含用户信息, 而之前的是令牌在有效时间里每次请求都是响应一样的令牌.

5.6 认证服务器-非对称加密 JWT 令牌

非对称加密算法需要两个密钥：公开密钥（publickey:简称公钥）和私有密钥（privatekey:简称私钥）。公钥与私钥是一对，如果用私钥对数据进行加密，只有用对应的公钥才能解密。

下面 JWT令牌生成采用非对称加密算法.

5.6.1 生成密钥证书

1. 公私钥对可以使用jdk的命令 `keytool` 来生成，首先来看一下这个命令下有哪些参数：

仅供购买者学习，禁止盗版、转卖、传播课程

```
管理员: C:\Windows\system32\cmd.exe
C:\Users\Administrator>keytool
密钥和证书管理工具

命令:

 -certreq            生成证书请求
 -changealias        更改条目的别名
 -delete             删除条目
 -exportcert         导出证书
 -genkeypair         生成密钥对
 -genseckey          生成密钥
 -gencert            根据证书请求生成证书
 -importcert         导入证书或证书链
 -importpass         导入口令
 -importkeystore     从其他密钥库导入一个或所有条目
 -keypasswd          更改条目的密钥口令
 -list               列出密钥库中的条目
 -printcert          打印证书内容
 -printcertreq       打印证书请求的内容
 -printcrl           打印 CRL 文件的内容
 -storepasswd        更改密钥库的存储口令

使用 "keytool -command_name -help" 获取 command_name 的用法

C:\Users\Administrator>
```

2. 生成密钥证书文件，每个证书包含公钥和私钥, 执行以下命令：

> 执行的命令在: 03-配套软件\生成私钥工具\生成公私钥命令.txt

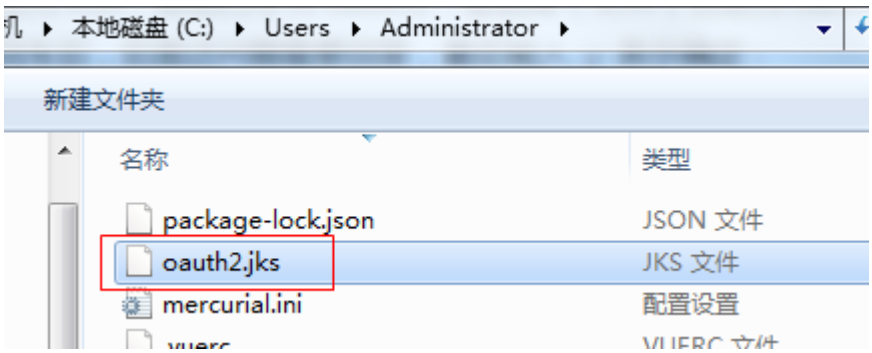
```
keytool -genkeypair -alias oauth2 -keyalg RSA -keypass oauth2 -keystore oauth2.jks
 -storepass oauth2
```

- 别名为 oauth2，秘钥算法为 RSA，秘钥口令为 oauth2，秘钥库（文件）名称为 oauth2.jks，秘钥库（文件）口令为 oauth2。输入命令回车后，后面还问题需要回答，最后输入 `y` 表示确定：

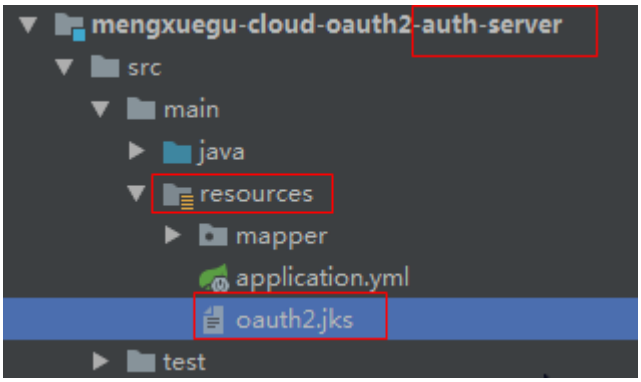
```
C:\Users\Administrator>keytool -genkeypair -alias oauth2 -keyalg RSA -keypass oa
uth2 -keystore oauth2.jks -storepass oauth2
您的名字与姓氏是什么?
  [Unknown]:  meng
您的组织单位名称是什么?
  [Unknown]:  mxg
您的组织名称是什么?
  [Unknown]:  mxg
您所在的城市或区域名称是什么?
  [Unknown]:  bj
您所在的省/市/自治区名称是什么?
  [Unknown]:  bj
该单位的双字母国家/地区代码是什么?
  [Unknown]:  cn
CN=meng, OU=mxg, O=mxg, L=bj, ST=bj, C=cn是否正确?
  [否]:  y


Warning:
JKS 密钥库使用专用格式。建议使用 "keytool -importkeystore -srckeystore oauth2.jk
s -destkeystore oauth2.jks -deststoretype pkcs12" 迁移到行业标准格式 PKCS12。
```

3. 生成后，在命令执行命令的所在目录下(教程中是 C:\Users\Administrator 目录)会有一个 `oauth2.jks` 文件

仅供购买者学习，禁止盗版、转卖、传播课程



4. 将 `oauth2.jks` 文件 拷贝到认证服务器的 `resources` 文件夹下:



5.6.2 非对称加密 JWT 令牌

1. 重构mengxuegu-cloud-oauth2-auth-server认证服务器中的 com.mengxuegu.oauth2.server.config.TokenConfig#jwtAccessTokenConverter 方法，切换为非对称加密算法

```
@Bean // 在 JwtAccessTokenConverter 中定义 Jwt 签名密码
public JwtAccessTokenConverter jwtAccessTokenConverter() {
    JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
//      // 对称密钥来签署我们的令牌，资源服务器也将使用此秘钥来验证准码性
//        converter.setSigningKey(SIGNING_KEY);
    // 读取 oauth2.jks 文件中的私钥，第2个参数是口令 oauth2
    KeyStoreKeyFactory keyFactory = new KeyStoreKeyFactory(new ClassPathResource("oauth2.jks"),
"oauth2".toCharArray());
    // 别名 oauth2
    converter.setKeyPair(keyFactory.getKeyPair("oauth2"));
    return converter;
}
```

5.7 资源服务器-非对称加密 JWT 令牌

5.6.1 根据密钥文件获取公钥

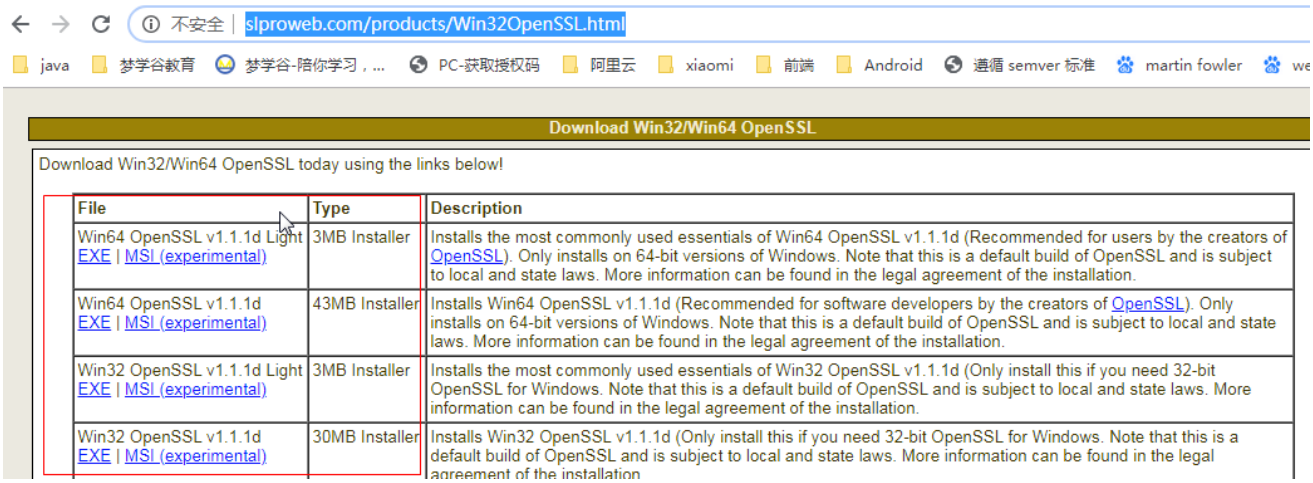
仅供购买者学习，禁止盗版、转卖、传播课程

安装OpenSSL

OpenSSL 是一个加解密工具包，可以使用 OpenSSL 来获取公钥。

1. 下载网址：http://slproweb.com/products/Win32OpenSSL.html

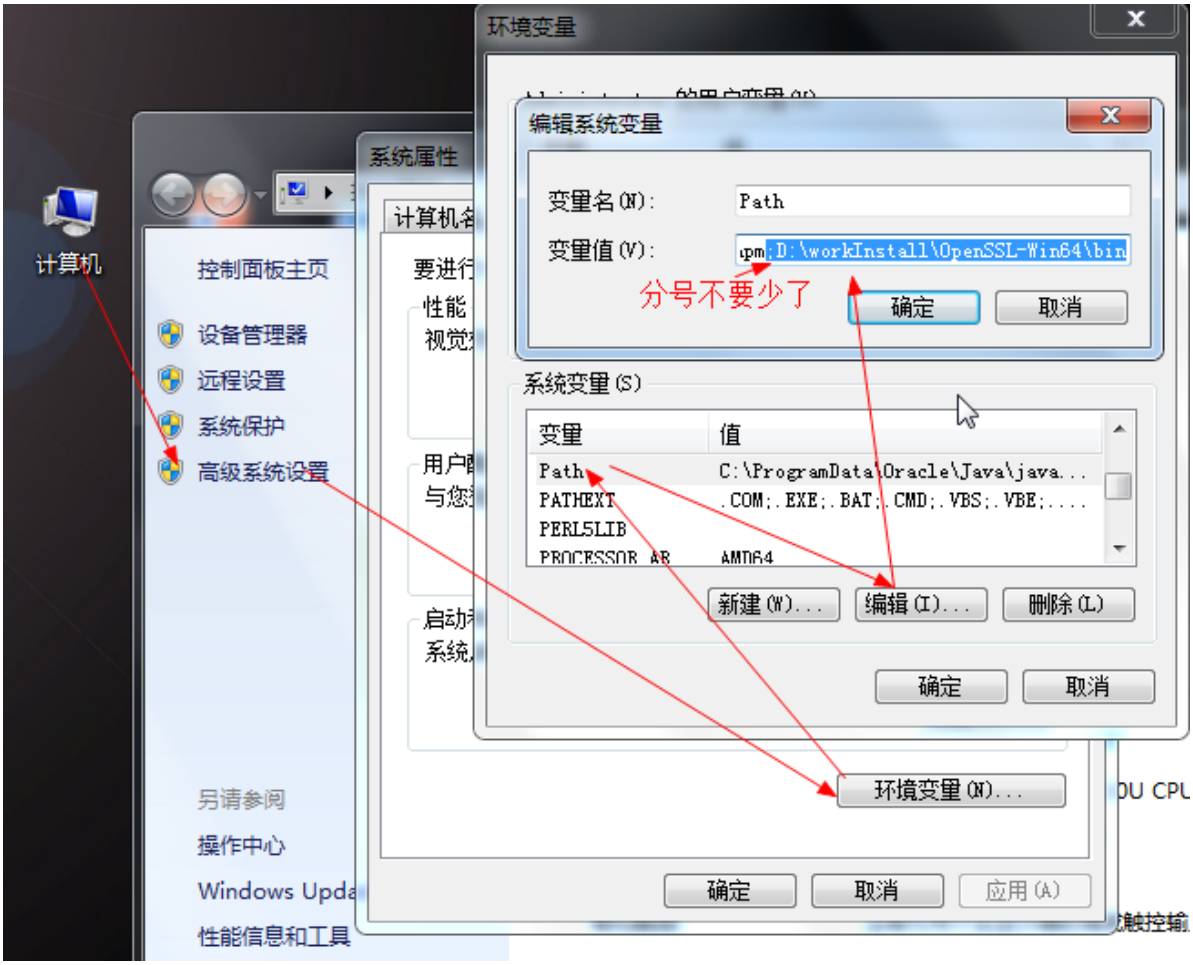
> 本地安装包位于：`03-配套软件\生成私钥工具\Win64OpenSSL_Light-1_1_1d.exe`



1. 安装 OpenSSL
   - 傻瓜式安装，但是不要安装包到中文目录，最后有个捐赠页面，随你自已。

   - 配置 OpenSSL 的环境变量，即你所安装的目录\bin，如：`D:\workInstall\OpenSSL-Win64\bin`

仅供购买者学习，禁止盗版、转卖、传播课程



获取公钥

1. 重新打开 CMD 命令行窗口，进入 `oauth2.jks` 文件所在目录执行如下命令：

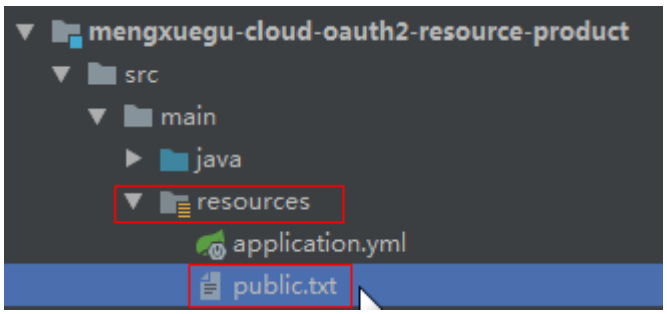
> 执行的命令在: 03-配套软件\生成私钥工具\生成公私钥命令.txt

```
keytool -list -rfc --keystore oauth2.jks | openssl x509 -inform pem -pubkey
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
管理员: C:\Windows\system32\cmd.exe
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>keytool -list -rfc --keystore oauth2.jks | openssl x509 -
inform pem -pubkey
输入密钥库口令:  oauth2                    复制下面红框所有内容
-----BEGIN PUBLIC KEY-----  ← 这个也要复制
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAw5MIvD4x06Ls89LBmJan
BoF7GyV1PYiLDJ8vHsRdGs7FkpJeMkFnbrIg2qpKkzYvhiIySpbpX4G/ODe9c1Qj
CcbJa3BQCweggswsBcDy45lPlPf3HEhJeHXuoxVcwkhw7o9oOYrBrE5YPKc+DbqO
g1Db8QPwi0G8IM0WjgbRS7aQtKeURqKxwdlQfXzevB2ndq7I31IeqSrjLNr+ILaH
HVS4NQJ8gCfOqiyfmlwg8NLcIfCdyK2dhpucaMlLVBbnvJnPB4mSWsjUm6LRHwWU
nSPtg9MG2PbA90jWuVL67xdNMcvuY6JFxjzj/XxlFURckMckI4K78GvxkcBkA9zv
+QIDAQAB
-----END PUBLIC KEY-----  ← 这个也要复制
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgAwIBAgIEPNijCzANBgkqhkiG9w0BAQsFADBRMQswCQYDVQQGEwJj
bjELMAkGA1UECBMCYmoxCzAJBgNVBAcTAmJqMQwwCgYDVQQKEwNteGcxDDAKBgNV
BAsTA214ZzEMMAoGA1UEAxMDbXhnMB4XDTE5MTIzMDA2Mzg1MVoXDTIwMDMyOTA2
Mzg1MVowUTELMAkGA1UEBhMCY24xCzAJBgNVBAgTAmJqMQswCQYDVQQHEwJiajEM
MAoGA1UEChMDbXhnMQwwCgYDVQQLEwNteGcxDDAKBgNVBAMTA214ZzCCASIwDQYJ
KoZIhvcNAQEBBQADggEPADCCAQoCggEBAMOTCLw+MdOi7PPSwZiWpwaBexsldT2I
iwyfLx7EXRrOxZKSXjJBZ26yINqqSpM2L4YiMkqW6V+Bvzg3vXNUIwnGyWtwUAsH
oILMLAXA8uOZT5T39xxISXh17qMVXMJIcO6PaDmKwaxOWDynPg26joNQ2/ED8ItB
vCDNFo4G0Uu2kLSnlEaiscHZUH183rwdp3auyN9SHqkq4yza/iC2hx1UuDUCfIAn
zqosn5pcIPDS3CHwncitnYabnGjJS1QW57yZzweJklrI1Jui0R8F1J0j7YPTBtj2
```

2. 复制打印出来的公钥，**注意：-----BEGIN PUBLIC KEY-----和-----END PUBLIC KEY-----必须要带上**。
3. 在资源服务器的 `resources` 文件夹下面，新建一个 `public.txt` 文件，将公钥粘贴进去：



5.6.2 非对称加密 JWT 令牌

1. 重构 `mengxuegu-cloud-oauth2-resource-product` 资源服务器中的 com.mengxuegu.oauth2.resource.ResourceServerConfig#jwtAccessTokenConverter 方法，切换为非对称加密算法

```
1     @Bean // 在 JwtAccessTokenConverter 中定义 Jwt 签名密码
2     public JwtAccessTokenConverter jwtAccessTokenConverter() {
3         JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
4         // 对称密钥来签署我们的令牌，资源服务器也将使用此秘钥来验证准码性
5 //        converter.setSigningKey(SIGNING_KEY);
6
7         // 非对称加密：私钥
8         ClassPathResource classPathResource = new ClassPathResource("public.txt");

9         String publicKey = null;
```

仅供购买者学习，禁止盗版、转卖、传播课程

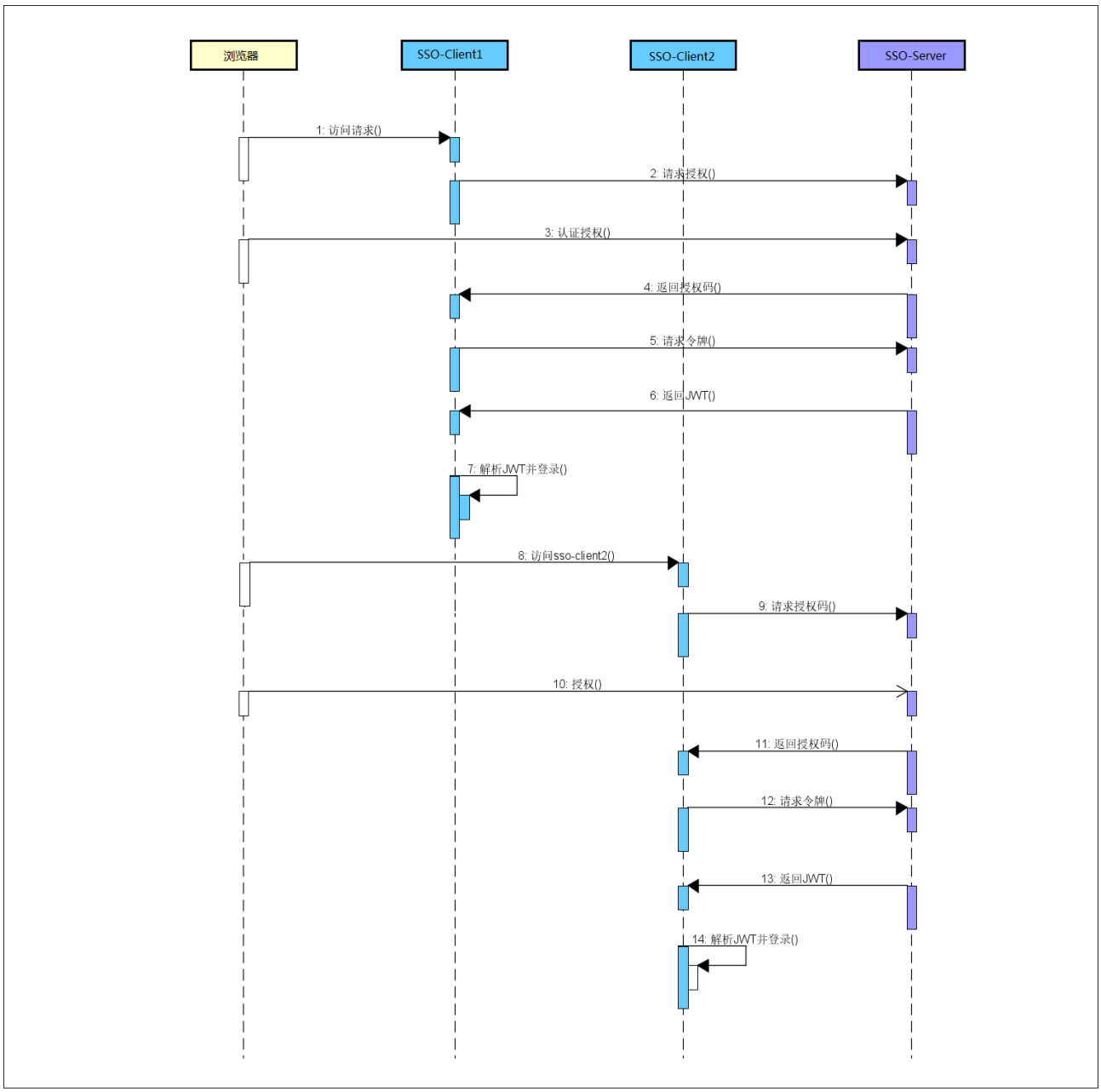
```
        try {
            publicKey=IOUtils.toString(classPathResource.getInputStream(),"UTF-8");
            System.out.println("publicKey:" + publicKey);
        } catch (IOException e) {
            e.printStackTrace();
        }
        converter.setVerifierKey(publicKey);
        return converter;
    }
```

第六章 Spring Security OAuth2 单点登录

6.1 单点登录概述

单点登录SSO（Single Sign On）说得简单点就是在一个多系统共存的环境下，用户在一处登录后，就不用在其他系统中登录，也就是用户的一次登录能得到其他所有系统的信任。单点登录在大型网站里使用得非常频繁，例如像阿里巴巴这样的网站，在网站的背后是成百上千的子系统，用户一次操作或交易可能涉及到几十个子系统的协作，如果每个子系统都需要用户认证，不仅用户会疯掉，各子系统也会为这种重复认证授权的逻辑搞疯掉。

仅供购买者学习，禁止盗版、转卖、传播课程



6.2 注册客户端信息

向 oauth_client_details 表添加两个客户端 `client1` 和 `client2` 信息

注意:

1. 单点登录中，认证服务器响应的是授权码是重定向到客户端的 `/login` 端点上, 数据库配置的重写向地址是 `/login` 端点
2. 单点登录中, 认证授权类型采用的是 授权码模式, 所以 authorized_grant_types 指定的授权码模式即可
3. `autoapprove` 值为 `true` 就不会跳转授权页, 自动授权.

仅供购买者学习，禁止盗版、转卖、传播课程

对象 | oauth_client_details @securi...

开始事务 | 备注 | 筛选 | 排序 | 导入 | 导出

client_id	client1
resource_ids	
client_secret	$2a$10$AodcXpujNILcZAQ.7cQ0b.Bm4klRumapoFBu7uyQ/ZV9Nu9F5fE3y
scope	MEMBER_READ,MEMBER_WRITE
authorized_grant_ty...	authorization_code,refresh_token
web_server_redirec...	http://localhost:9001/login
authorities	
access_token_validi...	50000
refresh_token_validi...	
additional_informat...	
autoapprove	true

对象 | oauth_client_details @securi...

开始事务 | 备注 | 筛选 | 排序 | 导入 | 导出

client_id	client2
resource_ids	
client_secret	$2a$10$AodcXpujNILcZAQ.7cQ0b.Bm4klRumapoFBu7uyQ/ZV9Nu9F5fE3y
scope	MEMBER_READ
authorized_grant_ty...	authorization_code,refresh_token
web_server_redirec...	http://localhost:9002/login 默认就是/login
authorities	
access_token_validi...	50000
refresh_token_validi...	
additional_informat...	
autoapprove	true

6.3 SSO 会员客户端1

创建客户端1模块

创建 mengxuegu-cloud-oauth2-sso-client1 模块

配置 pom.xml

仅供购买者学习，禁止盗版、转卖、传播课程

```
<dependencies>
    <dependency>
        <groupId>com.mengxuegu</groupId>
        <artifactId>mengxuegu-cloud-oauth2-base</artifactId>
        <version>${mengxuegu-security.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <!-- Spring Security、OAuth2 和JWT等 -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-oauth2</artifactId>
    </dependency>
     <!-- 注册到 Eureka
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
      -->

    <!-- thymeleaf 模块启动器-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <!--对Thymeleaf添加Spring Security标签支持-->
    <dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity5</artifactId>
    </dependency>


    <!-- springboot 单元测试 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
    </dependency>
    <!--热部署 ctrl+f9-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
    </dependency>

</dependencies>
```

配置 application.yml

```
server:
```

仅供购买者学习，禁止盗版、转卖、传播课程

```yaml
  port: 9001

spring:
  thymeleaf:
    cache: false

security:
  oauth2:
    client:
      client-id: client1
      client-secret: mengxuegu-secret
      user-authorization-uri: http://localhost:8090/auth/oauth/authorize #请求认证的地址
      access-token-uri: http://localhost:8090/auth/oauth/token #请求令牌的地址
    resource:
      jwt:
        # 当用户授权后会带着授权码重定向回客户端 localhost:9001/login?code=xxx
        # 对应/login会自动的去获取令牌，并通过key-uri指定的地址去获取公钥校验令牌有效性，
        # 然后完成本地认证与授权
        key-uri: http://localhost:8090/auth/oauth/token_key
```

检查 /oauth/token_key 所有人可访问

确保在认证授权服务器中已经将 `/oauth/token_key` 端点 `permitAll()` 所有人可以访问, 因为在启动客户端时，它就会去获取公钥 。

检查 AuthorizationServerConfig 权限服务器配置类对端点权限的控制， 如下：

```java
@Override
public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
    // 所有人可访问 /oauth/token_key 后面要获取公钥, 默认拒绝访问
    security.tokenKeyAccess("permitAll()");
    // 认证后可访问 /oauth/check_token , 默认拒绝访问
    security.checkTokenAccess("isAuthenticated()");
}
```

创建启动类

```java
package com.mengxuegu.oauth2.sso;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@SpringBootApplication
public class SsoClient1Application {

    public static void main(String[] args) {
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
        SpringApplication.run(SsoClient1Application.class, args);
    }

}
```

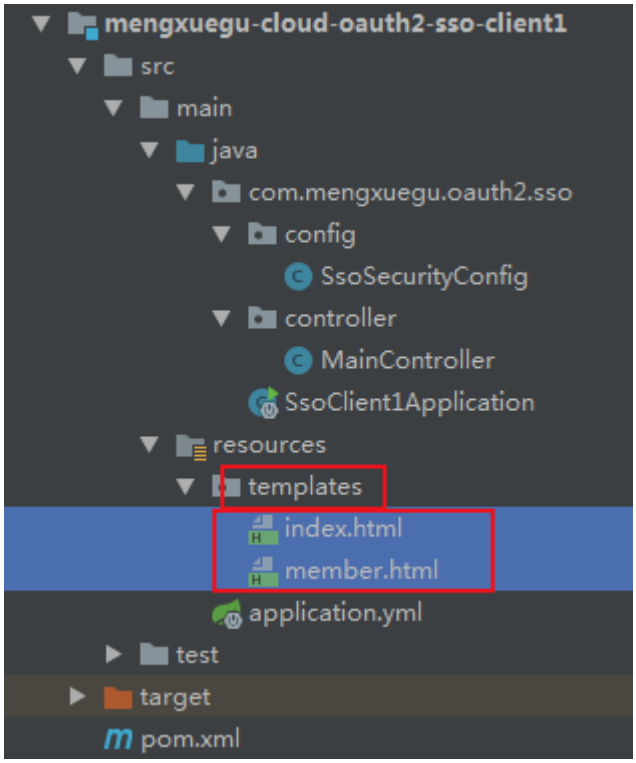
创建控制层跳转页面

```
package com.mengxuegu.oauth2.sso.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@Controller
public class MainController {

    @GetMapping("/")
    public String index() {
        return "index";
    }

    @GetMapping("/member")
    public String member() {
        return "member";
    }

}
```

创建模板页面

在 resources下创建 templates 目录，然后 templates 下创建一个 index.html 和 member.html

仅供购买者学习，禁止盗版、转卖、传播课程



1. 首页 resources/templates/index.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
    <h1>
        <a th:href="@{/member}">客户端1-查看会员</a>
    </h1>
</body>
</html>
```

2. 会员页面，登录后才可以访问，resources/templates/member.html

仅供购买者学习，禁止盗版、转卖、传播课程

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
    <div>
        <h1>客户端1，欢迎您！[[${#authentication.name}]]</h1>
        <h3><a th:href="@{/logout}">退出系统</a></h3>
    </div>
</body>
</html>
```

创建 SSO 登录配置类

```
package com.mengxuegu.oauth2.sso.config;

import org.springframework.boot.autoconfigure.security.oauth2.client.EnableOAuth2Sso;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@EnableOAuth2Sso
@Configuration
public class SsoSecurityConfig extends WebSecurityConfigurerAdapter {


    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                // 首页所有人可访问
                .antMatchers("/").permitAll()
                .anyRequest().authenticated()
            ;
    }

}
```

6.4 SSO 会员客户端2

创建 mengxuegu-cloud-oauth2-sso-client2 模块, 和上面客户端1基本上一样

仅供购买者学习，禁止盗版、转卖、传播课程

配置 pom.xml

```
<dependencies>
  <dependency>
      <groupId>com.mengxuegu</groupId>
      <artifactId>mengxuegu-cloud-oauth2-base</artifactId>
      <version>${mengxuegu-security.version}</version>
    </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <!-- Spring Security、OAuth2 和JWT等 -->
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-oauth2</artifactId>
  </dependency>
  <!-- 注册到 Eureka
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
   -->
  <!-- thymeleaf 模块启动器-->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <!--对Thymeleaf添加Spring Security标签支持-->
  <dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity5</artifactId>
  </dependency>


  <!-- springboot 单元测试 -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
  </dependency>
  <!--热部署 ctrl+f9-->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
  </dependency>

</dependencies>
```

配置 application.yml

仅供购买者学习，禁止盗版、转卖、传播课程

```yaml
server:
  port: 9002

spring:
  thymeleaf:
    cache: false

security:
  oauth2:
    client:
      client-id: client2
      client-secret: mengxuegu-secret
      user-authorization-uri: http://localhost:8090/auth/oauth/authorize #请求认证的地址
      access-token-uri: http://localhost:8090/auth/oauth/token #请求令牌的地址
    resource:
      jwt:
        # 当用户授权后会带着授权码重定向回客户端 localhost:9002/login?code=xxx
        # 对应/login会自动的去获取令牌，并通过key-uri指定的地址去获取公钥校验令牌有效性，
        # 然后完成本地认证与授权
        key-uri: http://localhost:8090/auth/oauth/token_key
```

创建启动类

```java
package com.mengxuegu.oauth2.sso;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@SpringBootApplication
public class SsoClient2Application {

    public static void main(String[] args) {
        SpringApplication.run(SsoClient2Application.class, args);
    }

}
```

创建控制层跳转页面

```java
package com.mengxuegu.oauth2.sso.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

```

仅供购买者学习，禁止盗版、转卖、传播课程

```
/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@Controller
public class MainController {

    @GetMapping("/")
    public String index() {
        return "index";
    }

    @GetMapping("/member")
    public String member() {
        return "member";
    }

}
```

创建模板页面

在 resources下创建 templates 目录，然后 templates 下创建一个 index.html 和 member.html

1. 首页 resources/templates/index.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
    <h1>
        <a th:href="@{/member}">客户端2-查看会员</a>
    </h1>
</body>
</html>
```

2. 会员页面，登录后才可以访问，resources/templates/member.html

仅供购买者学习，禁止盗版、转卖、传播课程

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
    <div>
        <h1>客户端2，欢迎您！[[${#authentication.name}]]</h1>
        <h3><a th:href="@{/logout}">退出系统</a></h3>
    </div>
</body>
</html>
```

创建 SSO 登录配置类

```
package com.mengxuegu.oauth2.sso.config;

import org.springframework.boot.autoconfigure.security.oauth2.client.EnableOAuth2Sso;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@EnableOAuth2Sso
@Configuration
public class SsoSecurityConfig extends WebSecurityConfigurerAdapter {


    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                // 首页所有人可访问
                .antMatchers("/").permitAll()
                .anyRequest().authenticated()
            ;
    }

}
```

6.5 单点登录测试

启动：

- 启动认证服务器 `mengxuegu-cloud-oauth2-auth-server`（启动于本地 `8090` 端口）

仅供购买者学习，禁止盗版、转卖、传播课程

- 启动客户端1应用 `mengxuegu-cloud-oauth2-sso-client1` （启动于本地 `9001` 端口）

  注意：一定要先启动认证服务器，因为客户端启动时会去发送请求获取公钥，不然启动报如下错误

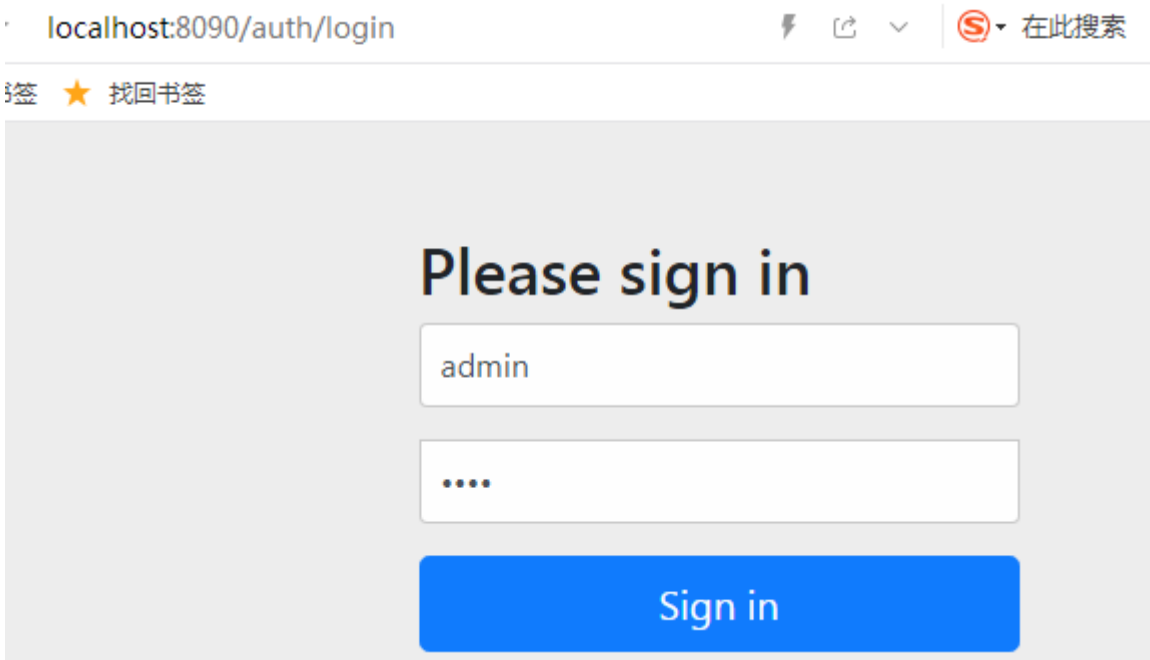
```
etWebServerApplicationContext : Exception encountered during context initia
talina.core.StandardService   : Stopping service [Tomcat]
aluationReportLoggingListener :

ons report re-run your application with 'debug' enabled.
ringApplication               : Application run failed

ception: Error creating bean with name 'jwtTokenServices' defined in class
torResolver.createArgumentArray(ConstructorResolver.java:798)
torResolver.instantiateUsingFactoryMethod(ConstructorResolver.java:539)
AutowireCapableBeanFactory.instantiateUsingFactoryMethod(AbstractAutowireCa
AutowireCapableBeanFactory.createBeanInstance(AbstractAutowireCapableBeanFa
AutowireCapableBeanFactory.doCreateBean(AbstractAutowireCapableBeanFactory.
```

- 启动客户端2应用 `mengxuegu-cloud-oauth2-sso-client2` （启动于本地 `9002` 端口）

浏览器访问：

1. 访问客户端1首页接口：http://localhost:9001/，点击 **查看会员**

2. 由于此时并没有过用户登录认证，因此会自动跳转到认证服务器的登录页面：`http://localhost:8090/auth/login`：



3. 数据库中此客户端的 `autoapprove` 值为true，所以自动同意授权，无需我们手动授权, 直接跳回会员页面

仅供购买者学习，禁止盗版、转卖、传播课程

4. 访问客户端2首页接口：http://localhost:9002/，点击 **查看会员**，

无需再重新登录，可以直接访问。

因为单点登录在客户端1已经登录过，认证服务器已经有用户授权记录，所以可以直接访问

梦学谷 www.mengxuegu.com | IT在线教育 高端品牌
陪你学习，伴你梦想
仅供购买者学习，禁止盗版、转卖、传播课程

6.6 SSO 退出系统

退出当前应用成功后，还需要向认证服务发送退出请求将所有客户端都退出。

- 分别在两个客户端 mengxuegu-cloud-oauth2-sso-client1 和 mengxuegu-cloud-oauth2-sso-client2 中的：com.mengxuegu.oauth2.sso.config.SsoSecurityConfig 添加退出处理逻辑

```
package oauth2.sso.config;

import org.springframework.boot.autoconfigure.security.oauth2.client.EnableOAuth2Sso;
```

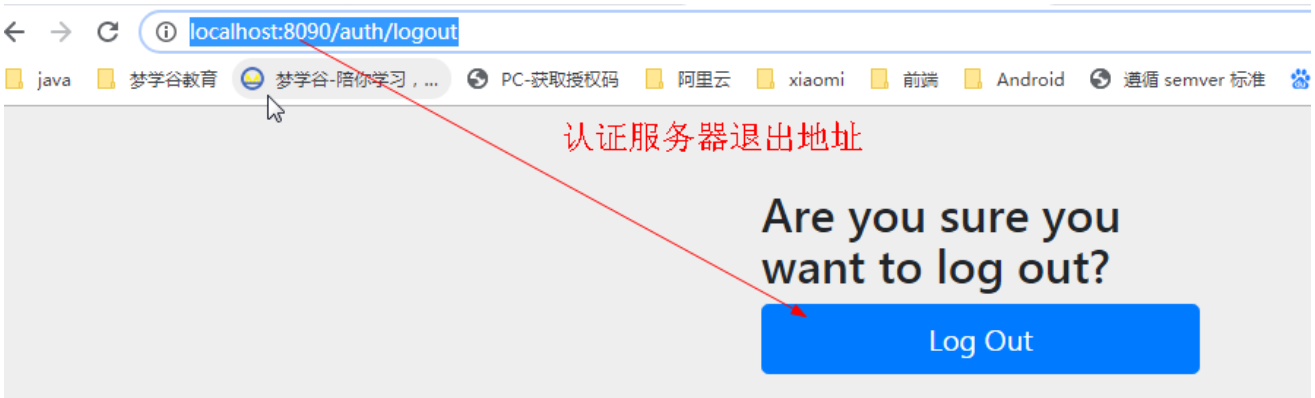
仅供购买者学习，禁止盗版、转卖、传播课程

```
4  import org.springframework.context.annotation.Configuration;
5  import org.springframework.security.config.annotation.web.builders.HttpSecurity;
6  import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
7
8  /**
9   * @Auther: 梦学谷 www.mengxuegu.com
10  */
11 @EnableOAuth2Sso
12 @Configuration
13 public class SsoSecurityConfig extends WebSecurityConfigurerAdapter {
14     //默认所有的请求都需要通过认证，
15
16     // 退出操作
17     @Override
18     protected void configure(HttpSecurity http) throws Exception {
19         http.authorizeRequests()
20                 .antMatchers("/").permitAll()
21                 .anyRequest().authenticated()
22             .and()
23                 // 当前应用退出
24                 .logout()
25                 // 当前退出成功后，跳转到认证服务器退出
26                 .logoutSuccessUrl("http://localhost:8090/auth/logout")
27                 .and().csrf().disable();
28     }
29
30 }
```

6.7 OSS 退出系统测试

上面退出成功, 还会跳转到认证服务器退出页面确认是否退出

仅供购买者学习，禁止盗版、转卖、传播课程

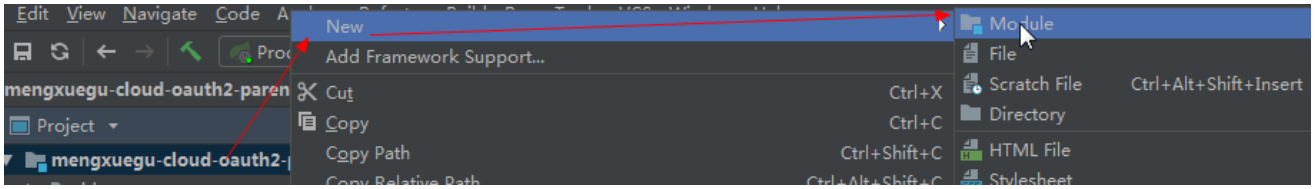


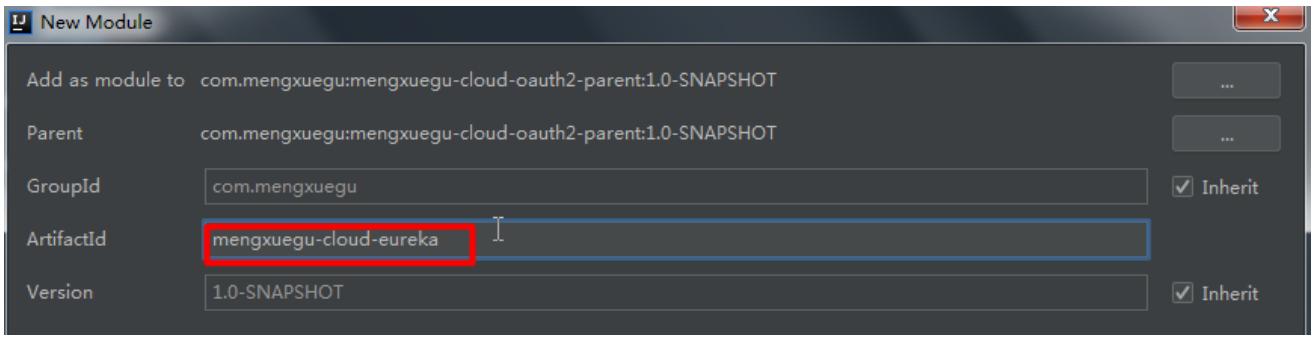
第七章 Spring Cloud OAuth2 分布式认证授权

7.1 微服务注册中心

创建 mengxuegu-cloud-eureka 模块

仅供购买者学习，禁止盗版、转卖、传播课程





配置 pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>mengxuegu-cloud-oauth2-parent</artifactId>
        <groupId>com.mengxuegu</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>mengxuegu-cloud-eureka</artifactId>

    <dependencies>
      <dependency>
          <groupId>com.mengxuegu</groupId>
          <artifactId>mengxuegu-cloud-oauth2-base</artifactId>
          <version>${mengxuegu-security.version}</version>
        </dependency>
        <!-- 导入Eureka-server 服务端依赖 -->
        <dependency>
          <groupId>org.springframework.cloud</groupId>
          <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
        </dependency>
    </dependencies>

</project>
```

配置 application.yml

仅供购买者学习，禁止盗版、转卖、传播课程

- 在 src/main/resources下新建 application.yml 文件，配置如下：

```yaml
server:
  port: 6001 # 服务端口

eureka:
  instance:
    hostname: localhost # eureka服务端的实例名称
  client:
    registerWithEureka: false # 服务注册，false表示不将自已注册到Eureka服务中
    fetchRegistry: false # 服务发现，false表示自己不从Eureka服务中获取注册信息
    serviceUrl:    # Eureka客户端与Eureka服务端的交互地址
      defaultZone: http //${eureka.instance.hostname}:${server.port}/eureka/
```

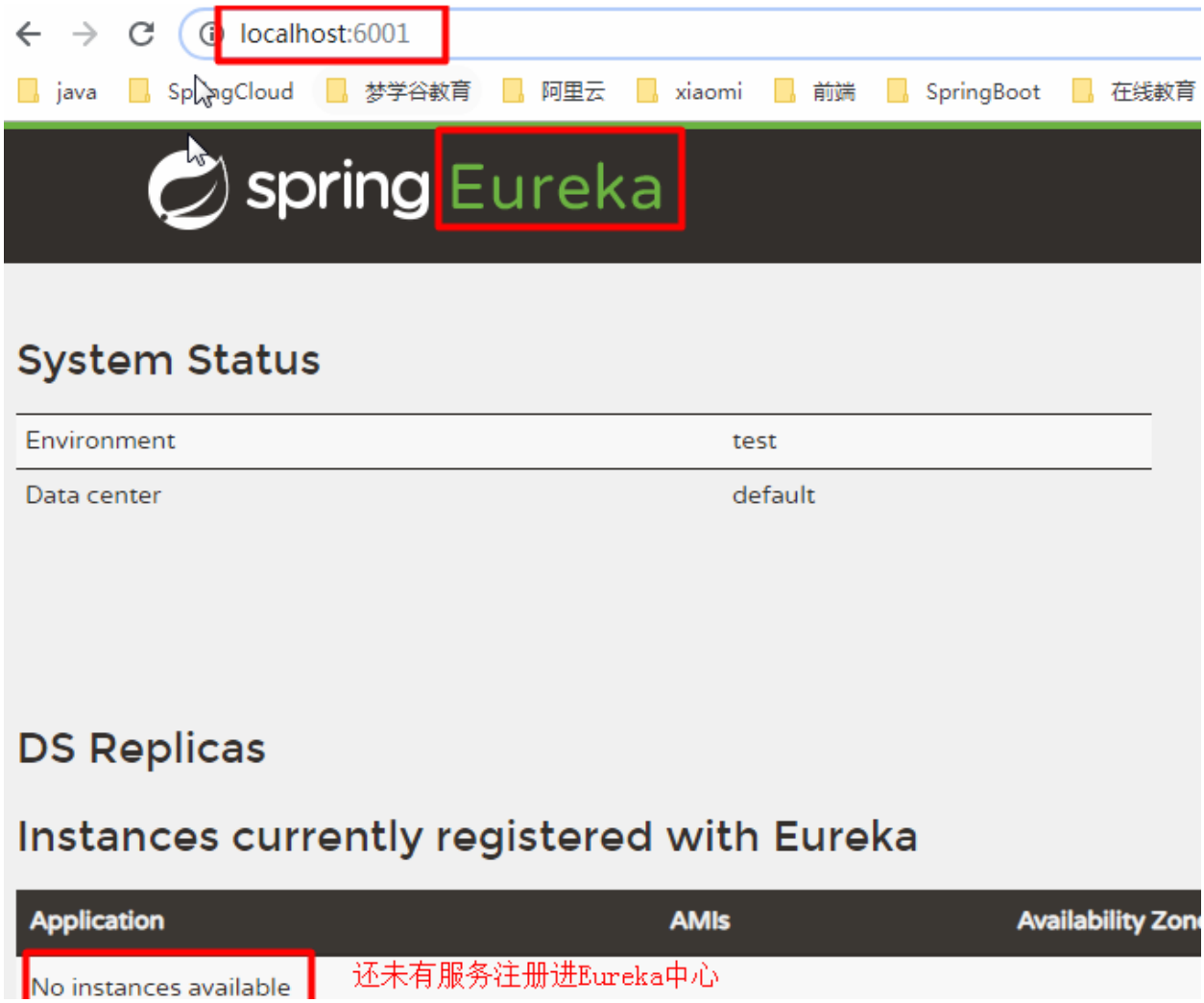
创建启动类

```java
package com.mengxuegu.oauth2;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@EnableEurekaServer //标识一个Eureka Server服务注册中心
@SpringBootApplication
public class EurekaServer_6001 {

    public static void main(String[] args) {
        SpringApplication.run(EurekaServer_6001.class, args);
    }

}
```

功能测试

- 启动：mengxuegu-cloud-eureka
- 访问：http://localhost:6001/ ，效果如下：

仅供购买者学习，禁止盗版、转卖、传播课程



7.2 认证服务器注册到注册中心

在 mengxuegu-cloud-oauth2-auth-server 认证服务器中做如下操作:

添加依赖

在 mengxuegu-cloud-oauth2-resource-product\pom.xml 中添加以下依赖:

```
<!-- 注册到 Eureka-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

配置 application.yml

```
server:
  port: 8090
```

仅供购买者学习，禁止盗版、转卖、传播课程

```yaml

eureka:
  client:
    registerWithEureka: true # 服务注册开关
    fetchRegistry: true # 服务发现开关
    serviceUrl: # 注册到哪一个Eureka Server服务注册中心，多个中间用逗号分隔
      defaultZone: http://localhost:6001/eureka
  instance:
    instanceId: ${spring.application.name}:${server.port} # 指定实例ID,页面会显示主机名
    preferIpAddress: true    #访问路径可以显示IP地址

spring:
  application:
    name: auth-server # 在Eureka页面会显示此服务名
```

修改主启动类

在启动类上添加 `@EnableEurekaClient` 注解，表示它是一个Eureka的客户端，本服务启动后会自动注册进Eureka Sever服务列表中

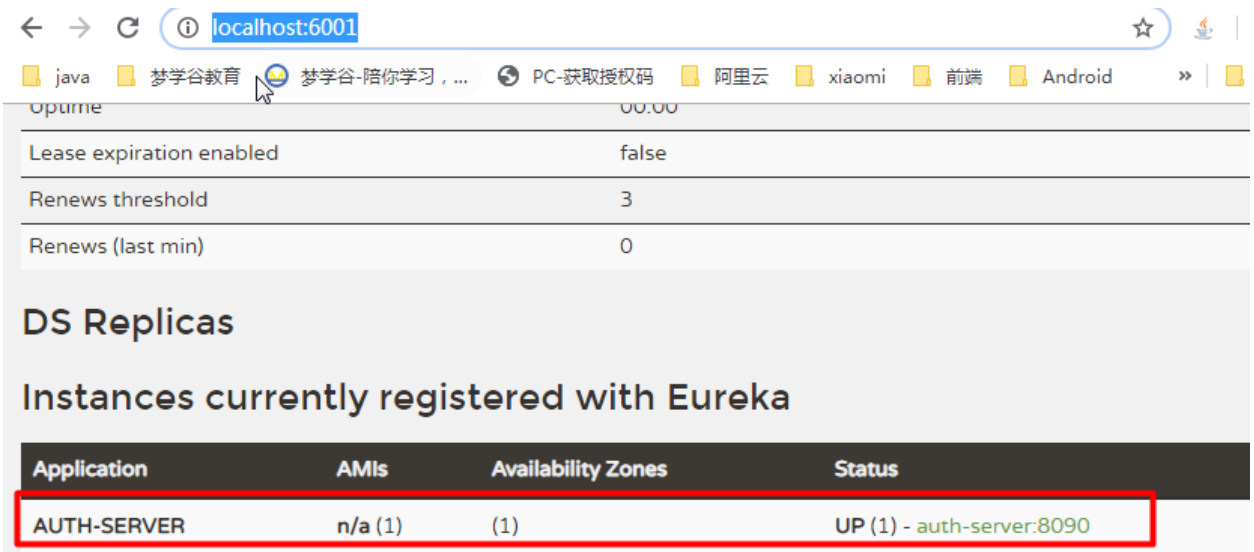
```java
package com.mengxuegu.oauth2;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

/**
 * 认证服务器启动类
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@EnableEurekaClient //本服务启动后会自动注册进Eureka中心
@SpringBootApplication
public class AuthServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(AuthServerApplication.class, args);
    }
}
```

功能测试

- 先要启动 Eureka Server: mengxuegu-cloud-eureka
- 再启动 Eureka Client：mengxuegu-cloud-oauth2-auth-server
- 访问：http://localhost:6001/

  认证服务器已经注册进 Eureka Server 中

仅供购买者学习，禁止盗版、转卖、传播课程



7.3 资源服务器注册到注册中心

在 mengxuegu-cloud-oauth2-resource-product 商品资源服务中做如下操作:

添加依赖

在 mengxuegu-cloud-oauth2-resource-product\pom.xml 中添加以下依赖:

```
<!-- 注册到 Eureka-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

配置 application.yml

```
server:
  port: 8080

eureka:
  client:
    registerWithEureka: true # 服务注册开关
    fetchRegistry: true # 服务发现开关
    serviceUrl: # 注册到哪一个Eureka Server服务注册中心，多个中间用逗号分隔
      defaultZone: http://localhost:6001/eureka
  instance:
    instanceId: ${spring.application.name}:${server.port} # 指定实例ID,页面会显示主机名
    preferIpAddress: true    #访问路径可以显示IP地址

spring:

  application:
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
16      name: product-server # 在Eureka页面会显示此服务名
```

修改主启动类

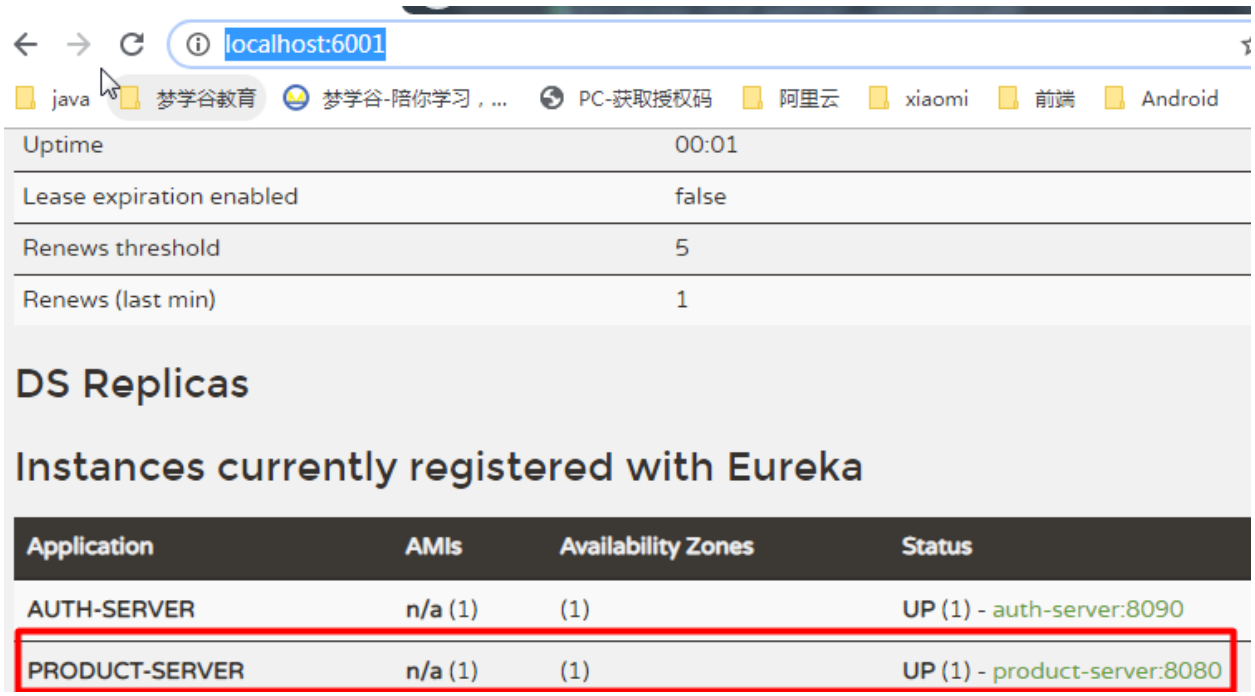
在启动类上添加 `@EnableEurekaClient` 注解，表示它是一个Eureka的客户端，本服务启动后会自动注册进Eureka Sever服务列表中

```
1  package com.mengxuegu.oauth2;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
6
7  /**
8   * 资源服务器启动类
9   * @Auther: 梦学谷 www.mengxuegu.com
10  */
11 @EnableEurekaClient //本服务启动后会自动注册进Eureka中心
12 @SpringBootApplication
13 public class ProductResourceApplication {
14     public static void main(String[] args) {
15         SpringApplication.run(ProductResourceApplication.class, args);
16     }
17 }
```

功能测试

- 先要启动 Eureka Server: mengxuegu-cloud-eureka
- 再启动 Eureka Client：mengxuegu-cloud-oauth2-resource-product
- 访问：http://localhost:6001/

  商品资源服务已经注册进 Eureka Server 中



仅供购买者学习，禁止盗版、转卖、传播课程

7.4 路由网关整合 OAuth2.0 认证授权

概述

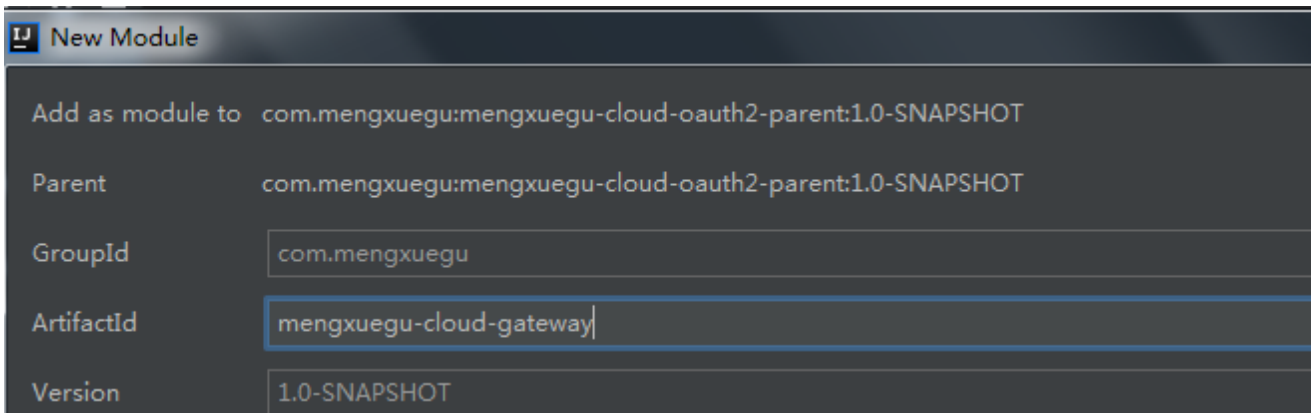
Zuul 整合 OAuth2.0，认证服务器采用 Jwt 生成令牌，统一在网关层验证请求，判断权限等操作。

核心步骤：

1. 将 Zuul 网关扮演**资源服务器**的角色，实现客户端权限拦截
2. 令牌解析和转发解析后的当前用户信息（基本信息和拥有权限）给应用级微服务
3. 应用级微服务接收到登录用户信息，可直接进行判断用户是否有访问对应资源权限。

网关环境搭建

创建 mengxuegu-cloud-gateway 模块



配置 pom.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
   4.0.0.xsd">
5      <parent>
6          <artifactId>mengxuegu-cloud-oauth2-parent</artifactId>
7          <groupId>com.mengxuegu</groupId>
8          <version>1.0-SNAPSHOT</version>
9      </parent>
10     <modelVersion>4.0.0</modelVersion>
11
12     <artifactId>mengxuegu-cloud-gateway</artifactId>
13
14     <dependencies>
15         <dependency>
16             <groupId>com.mengxuegu</groupId>
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
        <artifactId>mengxuegu-cloud-oauth2-base</artifactId>
        <version>${mengxuegu-security.version}</version>
    </dependency>
     <!-- 一样作为资源服务器，所以要引入 -->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-oauth2</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
    <!--zuul路由网关依赖-->
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
    </dependency>
  </dependencies>
</project>
```

配置 application.yml

端口号：7001，服务名：zuul-gateway

特别注意文件中的层级关系

```
server:
  port: 7001 # 服务端口

spring:
  application:
    name: zuul-gateway

eureka:
  client:
    registerWithEureka: true # 服务注册开关
    fetchRegistry: true # 服务发现开关
    serviceUrl: # 注册到哪一个Eureka Server服务注册中心，多个中间用逗号分隔
      defaultZone: http://localhost:6001/eureka
  instance:
    instanceId: ${spring.application.name}:${server.port} # 指定实例ID,Eureka页面会显示主机名
    preferIpAddress: true     #访问路径可以显示IP地址

zuul: # 网关配置
  sensitive-headers: null # 默认Zuul认为请求头中 "Cookie", "Set-Cookie", "Authorization" 是敏感信息，它不会转发请求，因为把它设置为空，就会转发了
  add-host-header: true # 正确的处理重定向操作
  routes:
    authentication: # 路由名称，名称任意，保持所有路由名称唯一
      path: /auth/** # 访问路径，转发到 auth-server 服务处理
      serviceId: auth-server # 指定服务ID，会自动从Eureka中找到此服务的ip和端口
      stripPrefix: false # 代理转发时去掉前缀，false:代理转发时不去掉前缀 例如:为true时请求 /product/get/1，代理
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
    转发到/get/1
  product:  # 商品服务路由配置
    path: /product/** # 转发到 product-server 服务处理
    serviceId: product-server
    stripPrefix: false
```

创建启动类

使用 `@EnableZuulProxy` 注解标注启动类，开启zuul的功能

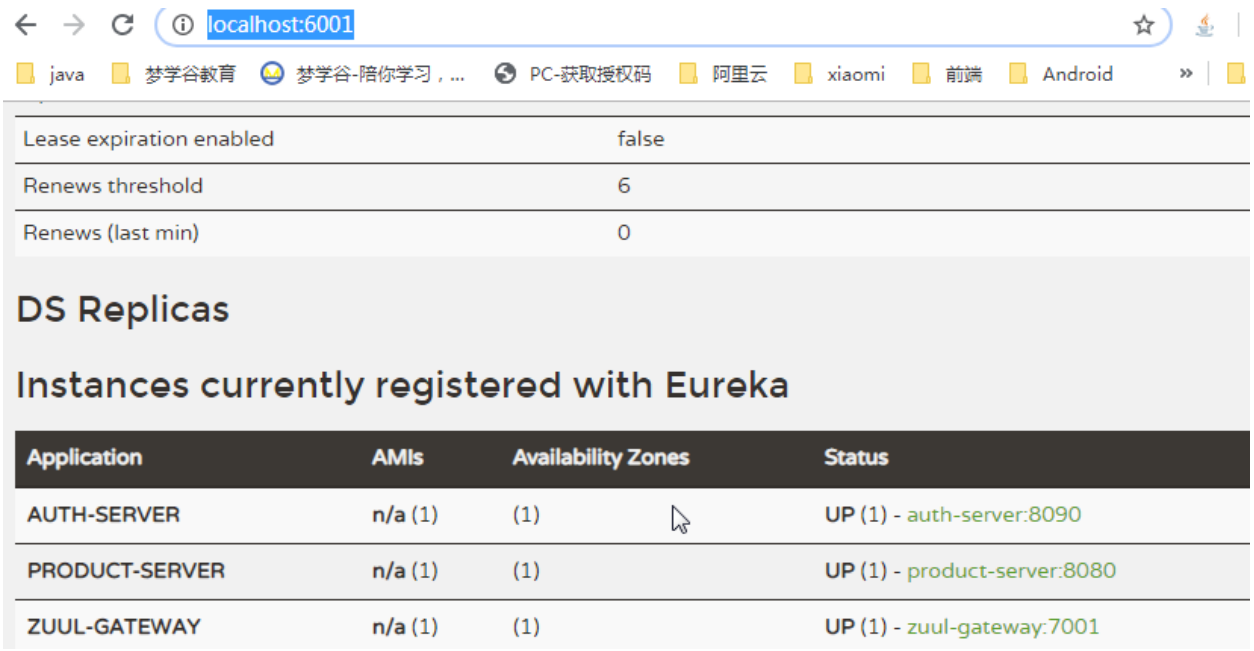
```
package com.mengxuegu.oauth2;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@EnableZuulProxy //开启zuul的功能
@EnableEurekaClient
@SpringBootApplication
public class ZuulServer_7001 {

    public static void main(String[] args) {
        SpringApplication.run(ZuulServer_7001.class, args);
    }

}
```

测试Zuul是否已被注册

查看 zuul 路由服务是否注册进Eureka注册中心

1. 启动 eureka 注册中心
2. 启动认证服务器
3. 启动商品资源服务器
4. 效果如下：http://localhost:6001/

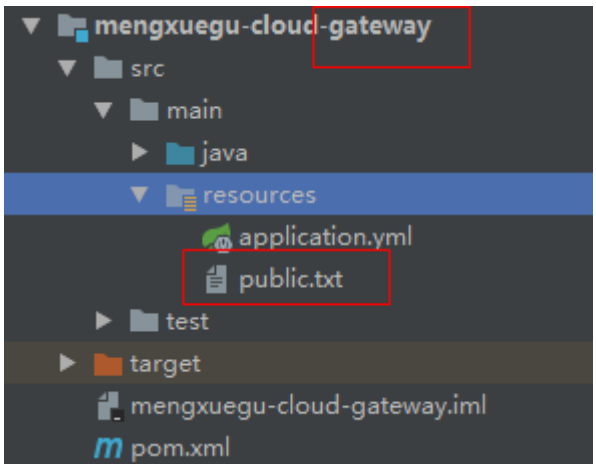
仅供购买者学习，禁止盗版、转卖、传播课程



JWT 令牌管理

资源服务器只要知道非对称加密的公钥，就可以进行本地验证并解析JWT令牌， 解析后它会将用户信息封装到 OAuth2Authentication 对象中。当前我们只要在 TokenStore 中JWT令牌管理方式即可。

1. 将商品资源服务器中的 `public.txt` 公钥文件拷贝过来：



2. 将商品资源服务器中的 TokenConfig 拷贝过来：

```
package com.mengxuegu.oauth2.config;

import org.apache.commons.io.IOUtils;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;
import org.springframework.security.oauth2.provider.token.TokenStore;
import org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter;
import org.springframework.security.oauth2.provider.token.store.JwtTokenStore;

```

仅供购买者学习，禁止盗版、转卖、传播课程

```
11 import java.io.IOException;
12
13
14 /**
15  * @Auther: 梦学谷 www.mengxuegu.com
16  */
17 @Configuration
18 public class TokenConfig {
19
20     @Bean
21     public JwtAccessTokenConverter jwtAccessTokenConverter() {
22         JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
23         // 非对称加密，资源服务器使用公钥解密 public.txt
24         ClassPathResource resource = new ClassPathResource("public.txt");
25         String publicKey = null;
26         try {
27             publicKey = IOUtils.toString(resource.getInputStream(), "UTF-8");
28         } catch (IOException e) {
29             e.printStackTrace();
30         }
31         converter.setVerifierKey(publicKey);
32         return converter;
33     }
34
35     @Bean
36     public TokenStore tokenStore() {
37         // Jwt管理令牌
38         return new JwtTokenStore(jwtAccessTokenConverter());
39     }
40
41 }
42
```

配置网关的资源服务配置类

网关也被认为是资源服务器, 因为要访问每个微服务资源，都要经过网关进行拦截判断是否允许访问。

下面就要配置每个微服务的权限规则，实现客户端权限拦截，这样只有当客户端拥有了对应权限才可以访问到对应微服务。

配置步骤：

1. 认证服务器相关配置所有请求全部放行 `.antMatchers("/**").permitAll()`
2. 商品资源服务器配置 `/product/**` 相关请求，对接入的客户端的 scope 要有 `PRODUCT_API` 范围

```
1 package com.mengxuegu.oauth2.config;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
6 import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
7  import
   org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter;
8  import
   org.springframework.security.oauth2.config.annotation.web.configurers.ResourceServerSecurityConfigurer;
9  import org.springframework.security.oauth2.provider.token.TokenStore;
10
11 /**
12  * @Auther: 梦学谷 www.mengxuegu.com
13  */
14 @Configuration
15 public class ResourceServerConfig {
16
17     //配置当前资源服务器的ID
18     public static final String RESOURCE_ID = "product-server";
19
20     @Autowired
21     private TokenStore tokenStore;
22
23     // 认证服务器资源拦截
24     @Configuration
25     @EnableResourceServer
26     public class AuthResourceServerConfig extends ResourceServerConfigurerAdapter {
27
28         public void configure(ResourceServerSecurityConfigurer resources) throws Exception {
29             resources.resourceId(RESOURCE_ID).tokenStore(tokenStore);
30         }
31         @Override
32         public void configure(HttpSecurity http) throws Exception {
33             // 认证服务器相关资源全部放行，用于处理认证
34             http.authorizeRequests().anyRequest().permitAll();
35         }
36     }
37
38     // 商品资源服务器资源拦截
39     @Configuration
40     @EnableResourceServer
41     public class ProductResourceServerConfig extends ResourceServerConfigurerAdapter {
42
43         @Override
44         public void configure(ResourceServerSecurityConfigurer resources)
45             throws Exception {
46             resources.resourceId(RESOURCE_ID).tokenStore(tokenStore);
47         }
48
49         @Override
50         public void configure(HttpSecurity http) throws Exception {
51             // 客户端要有 PRODUCT_API 范围才可访问
52             http.authorizeRequests()
53                 .antMatchers("/product/**")
54                 .access("#oauth2.hasScope('PRODUCT_API')");
55         }
56     }

57 }
```

仅供购买者学习，禁止盗版、转卖、传播课程

创建安全配置类

关于拦截功能都交给资源配置类处理，在安全配置中将所有请求放行即可，不然默认情况下所有请求都要认证。

在 `mengxuegu-cloud-gateway` 服务创建 com.mengxuegu.oauth2.config.SpringSecurityConfig 安全配置类

```
package com.mengxuegu.oauth2.config;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

/**
 * 安全配置类
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@EnableWebSecurity
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // 让资源配置类处理请求，这里放行所有的即可
        http.authorizeRequests().anyRequest().permitAll();
    }

}
```

自定义过滤器转发请求

概述

将解析后的用户信息转发给目标微服务，这样目标微服务可以判断用户可访问的权限。

- 自定义过虑器需要继承 ZuulFilter，ZuulFilter是一个抽象类，需要覆盖它的4个方法，如下：
  - filterType：返回字符串代表过滤器的类型，返回值有：
    - pre：在请求路由之前执行
    - route：在请求路由时调用
    - post：请求路由之后调用， 也就是在route和errror过滤器之后调用
    - error：处理请求发生错误时调用
  - filterOrder：此方法返回整型数值，通过此数值来定义过滤器的执行顺序，数字越小优先级越高。
  - shouldFilter：返回Boolean值，判断该过滤器是否执行。返回true表示要执行此过虑器，false不执行。
  - run：过滤器的业务逻辑。

创建过滤器

仅供购买者学习，禁止盗版、转卖、传播课程

自定义过滤器 `AuthenticationFilter` ，继承 ZuulFilter，实现抽象方法。

在类上添加 @Component 注解 (一定不要少了)

```
1  package com.mengxuegu.oauth2.filter;
2
3  import com.alibaba.fastjson.JSON;
4  import com.netflix.zuul.ZuulFilter;
5  import com.netflix.zuul.context.RequestContext;
6  import com.netflix.zuul.exception.ZuulException;
7  import org.slf4j.Logger;
8  import org.slf4j.LoggerFactory;
9  import org.springframework.security.core.Authentication;
10 import org.springframework.security.core.GrantedAuthority;
11 import org.springframework.security.core.authority.AuthorityUtils;
12 import org.springframework.security.core.context.SecurityContextHolder;
13 import org.springframework.security.oauth2.provider.OAuth2Authentication;
14 import org.springframework.stereotype.Component;
15 import org.springframework.util.Base64Utils;
16
17 import java.io.UnsupportedEncodingException;
18 import java.util.Collection;
19 import java.util.HashMap;
20 import java.util.Map;
21 import java.util.Set;
22
23 /**
24  * 请求资源前, 先通过此过滤器进行用户令牌解析与校验、转发
25  * @Auther: 梦学谷 www.mengxuegu.com
26  */
27 @Component //不要少了
28 public class AuthenticationFilter extends ZuulFilter {
29     Logger logger = LoggerFactory.getLogger(getClass());
30
31     @Override
32     public String filterType() {
33         // pre 请求路由前调用
34         return "pre";
35     }
36
37     @Override
38     public int filterOrder() {
39         // 过滤器执行顺序, 数值越小优先级越高
40         return 0;
41     }
42
43     @Override
44     public boolean shouldFilter() {
45         // true 执行下面run方法
46         return true;
47     }
48
49     @Override
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
    public Object run() throws ZuulException {


        // 获取从Security上下文中获取认证信息
        Authentication authentication =
                SecurityContextHolder.getContext().getAuthentication();


        // JWT令牌有效，则会解析用户信息封装到OAuth2Authentication对象中
        if( !(authentication instanceof OAuth2Authentication) ) {
            return null;
        }

        // 只是用户名, 用户表中手机号，邮箱等都没有
        Object principal = authentication.getPrincipal();
        // 此用户拥有权限
        Collection<? extends GrantedAuthority> authorities =
                authentication.getAuthorities();
        Set<String> authoritySet = AuthorityUtils.authorityListToSet(authorities);
        //请求详情
        Object details = authentication.getDetails();

        //封装传输的数据
        Map<String, Object> result = new HashMap<>();
        result.put("principal", principal);
        result.put("details", details);
        result.put("authorities", authoritySet);

        try {
            // 获取当前请求上下文
            RequestContext context = RequestContext.getCurrentContext();
            String base64 = Base64Utils.encodeToString(JSON.toJSONString(result).getBytes("UTF-8"));
            context.addZuulRequestHeader("auth-token", base64);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }

        return null;
    }
}
```

解决前后端分离跨域问题

所有请求都先经过 zuul 路由网关，所以要这里统一处理

```
package com.mengxuegu.oauth2.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import org.springframework.web.cors.CorsConfiguration;
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
6  import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
7  import org.springframework.web.filter.CorsFilter;
8
9  /**
10  * @Auther: 梦学谷 www.mengxuegu.com
11  */
12 @Configuration
13 public class GatewayConfig {
14
15     // 配置全局解决zuul服务中的cors跨域问题
16     @Bean
17     public CorsFilter corsFilter() {
18         final UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
19         final CorsConfiguration corsConfiguration = new CorsConfiguration();
20         corsConfiguration.addAllowedHeader("*");
21         corsConfiguration.addAllowedOrigin("*");
22         corsConfiguration.addAllowedMethod("*");
23         //↓核心代码
24         corsConfiguration.addExposedHeader("Authorization");
25         source.registerCorsConfiguration("/**", corsConfiguration);
26         return new CorsFilter(source);
27     }
28 }
```

7.5 微服务用户授权

概述

在微服务中接收到网关转发过来的 Token 后，需要我们构建一个 Authentication 对象来完成微服务认证与授权，这样这个微服务就可以根据用户所拥有的权限，来判断对应资源是否可以被用户访问。

过滤器实现授权

自定义过滤器拦截 token ，将用户信息封装到 `UsernamePasswordAuthenticationToken` 对象中，则完成认证也授权。

```
1  package com.mengxuegu.oauth2.resource.filter;
2
3  import com.alibaba.fastjson.JSON;
4  import com.alibaba.fastjson.JSONObject;
5  import org.apache.commons.lang.ArrayUtils;
6  import org.apache.commons.lang.StringUtils;
7  import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
8  import org.springframework.security.core.authority.AuthorityUtils;
9  import org.springframework.security.core.context.SecurityContextHolder;
10 import org.springframework.stereotype.Component;
11 import org.springframework.util.Base64Utils;
12 import org.springframework.web.filter.OncePerRequestFilter;
```

仅供购买者学习，禁止盗版、转卖、传播课程

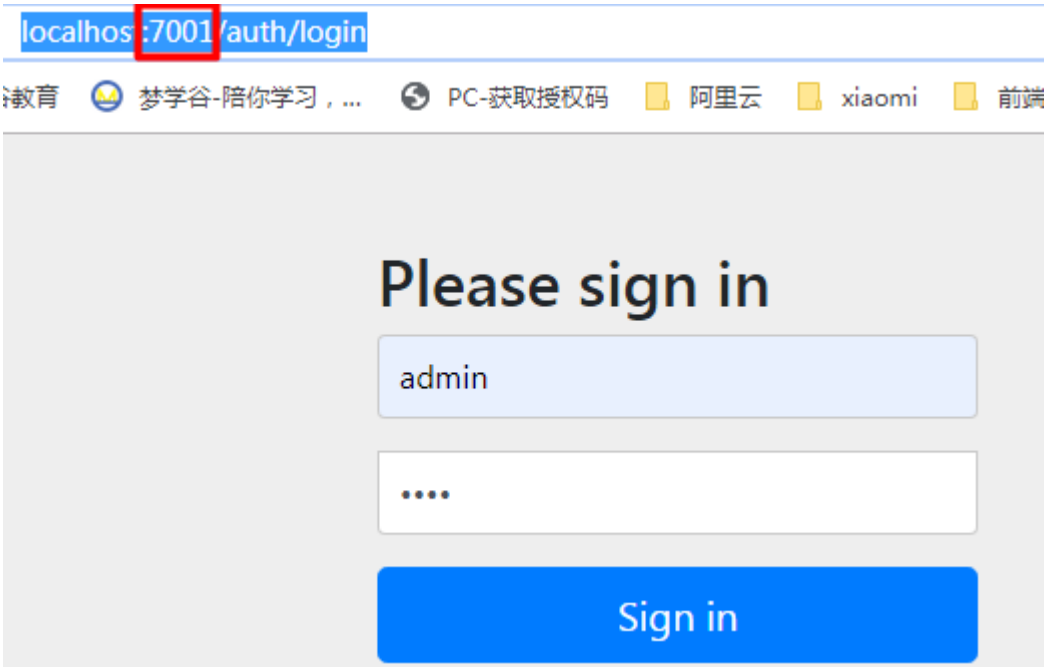
```
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@Component
public class TokenAuthenticationFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response, FilterChain filterChain) throws ServletException,
IOException {

        // 1. 从请求头中获取网关转发过来的明文 token
        String authToken = request.getHeader("auth-token");

        if(StringUtils.isNotEmpty(authToken)) {
            // Base64解码
            String authTokenJson = new String(Base64Utils.decodeFromString(authToken));
            // 转成json对象
            JSONObject jsonObject = JSON.parseObject(authTokenJson);
            // 用户权限
            String authorities = ArrayUtils.toString( jsonObject.getJSONArray("authorities").toArray() );

            // 构建一个Authentication对象, SpringSecurity 就会用于权限判断
            UsernamePasswordAuthenticationToken authenticationToken
                    = new UsernamePasswordAuthenticationToken(
                        jsonObject.get("principal"),
                        null,
                    AuthorityUtils.commaSeparatedStringToAuthorityList(authorities));
            // 请求详情
            authenticationToken.setDetails(jsonObject.get("details"));
            // 传递给上下文,这样服务可以进行获取对应数据
            SecurityContextHolder.getContext().setAuthentication(authenticationToken);
        }

        // 放行
        filterChain.doFilter(request, response);
    }
}
```

测试

1. 端口换成 7001 网关端口获取授权码，请求地址如下：

   http://localhost:7001/auth/oauth/authorize?client_id=mengxuegu-pc&response_type=code

仅供购买者学习，禁止盗版、转卖、传播课程



2. 通过授权码获取令牌：

   http://localhost:7001/auth/oauth/token

3. 通过令牌请求端口信息

   http://localhost:7001/product/list

7.6 客户端整合到网关

重构客户端1

pom.xml

添加 Eureka 客户端依赖

```
<!-- 注册到 Eureka-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

启动类

添加 @EnableEurekaClient 注解标识Eureka客户端

```
package com.mengxuegu.oauth2.sso;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

```

仅供购买者学习，禁止盗版、转卖、传播课程

```
/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@EnableEurekaClient
@SpringBootApplication
public class SsoClient1Application {

    public static void main(String[] args) {
        SpringApplication.run(SsoClient1Application.class, args);
    }

}
```

application.yml

- 添加注册到 Eureka 中的配置
- 添加服务应用名 spring.application.name=sso-client1
- 修改 `security` 下的认证服务器端口 `8090` ，改为 `7001` 网关端口

```
server:
  port: 9001

eureka:
  client:
    registerWithEureka: true # 服务注册开关
    fetchRegistry: true # 服务发现开关
    serviceUrl: # 注册到哪一个Eureka Server服务注册中心，多个中间用逗号分隔
      defaultZone: http://localhost:6001/eureka
  instance:
    instanceId: ${spring.application.name}:${server.port}
    preferIpAddress: true     #访问路径可以显示IP地址

spring:
  application:
    name: sso-client1
  thymeleaf:
    cache: false


security:
  oauth2:
    client:
      client-id: client1
      client-secret: mengxuegu-secret
      user-authorization-uri: http://localhost:7001/auth/oauth/authorize #请求认证的地址
      access-token-uri: http://localhost:7001/auth/oauth/token #请求令牌的地址
    resource:
      jwt:
        # 当用户授权后会带着授权码重定向回客户端 localhost:9001/login?code=xxx
        # 对应/login会自动的去获取令牌，并通过key-uri指定的地址去获取公钥校验令牌有效性，
        # 然后完成本地认证与授权

        key-uri: http://localhost:7001/auth/oauth/token_key
```

仅供购买者学习，禁止盗版、转卖、传播课程

修改 SsoSecurityConfig

退出系统的，认证服务器的端口 8090 改为 网关端口 7001

重构客户端2

pom.xml

添加 Eureka 客户端依赖

```
<!-- 注册到 Eureka-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

启动类

添加 @EnableEurekaClient 注解标识Eureka客户端

```
package com.mengxuegu.oauth2.sso;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
@EnableEurekaClient
@SpringBootApplication
public class SsoClient2Application {

    public static void main(String[] args) {
        SpringApplication.run(SsoClient2Application.class, args);
    }

}
```

仅供购买者学习，禁止盗版、转卖、传播课程

application.yml

- 添加注册到 Eureka 中的配置
- 添加服务应用名 spring.application.name=sso-client2
- 修改 `security` 下的认证服务器端口 `8090` ，改为 `7001` 网关端口

```yaml
server:
  port: 9002

eureka:
  client:
    registerWithEureka: true # 服务注册开关
    fetchRegistry: true # 服务发现开关
    serviceUrl: # 注册到哪一个Eureka Server服务注册中心，多个中间用逗号分隔
      defaultZone: http://localhost:6001/eureka
  instance:
    instanceId: ${spring.application.name}:${server.port}
    preferIpAddress: true    #访问路径可以显示IP地址

spring:
  application:
    name: sso-client2
  thymeleaf:
    cache: false


security:
  oauth2:
    client:
      client-id: client2
      client-secret: mengxuegu-secret
      user-authorization-uri: http://localhost:7001/auth/oauth/authorize #请求认证的地址
      access-token-uri: http://localhost:7001/auth/oauth/token #请求令牌的地址
    resource:
      jwt:
        # 当用户授权后会带着授权码重定向回客户端 localhost:9002/login?code=xxx
        # 对应/login会自动的去获取令牌，并通过key-uri指定的地址去获取公钥校验令牌有效性，
        # 然后完成本地认证与授权
        key-uri: http://localhost:7001/auth/oauth/token_key
```

修改 SsoSecurityConfig

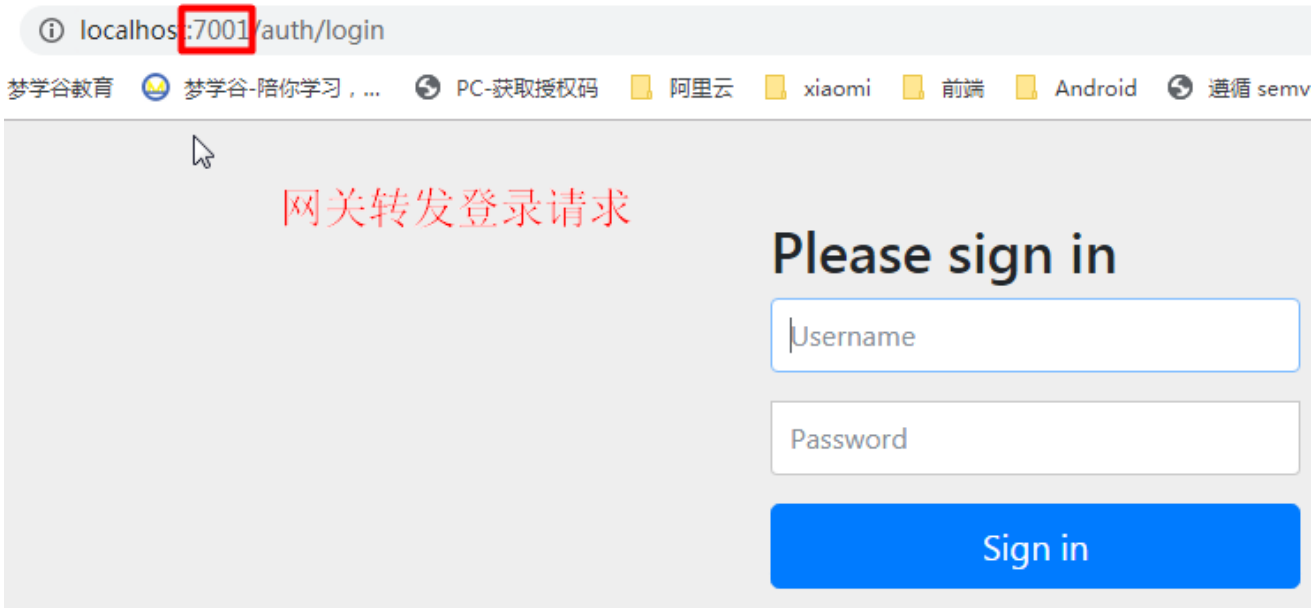
mengxuegu-cloud-oauth2-sso-client2 客户端2中的 退出系统，认证服务器的端口 `8090` 改为 网关端口 `7001`

仅供购买者学习，禁止盗版、转卖、传播课程

```
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests() ExpressionInterceptUrlRegistry
            // 首页所有人可访问
            .antMatchers( ...antPatterns: "/").permitAll() ExpressionUrlAutho
            .anyRequest().authenticated() ExpressionUrlAuthorizationConfigu
        .and() HttpSecurity
            // 退出操作
            .logout() LogoutConfigurer<HttpSecurity>
            // 当前退出成功后，跳转到认证服务器退出
            .logoutSuccessUrl("http://localhost:7001/auth/logout") Logo
            .and().csrf().disable();
}
```

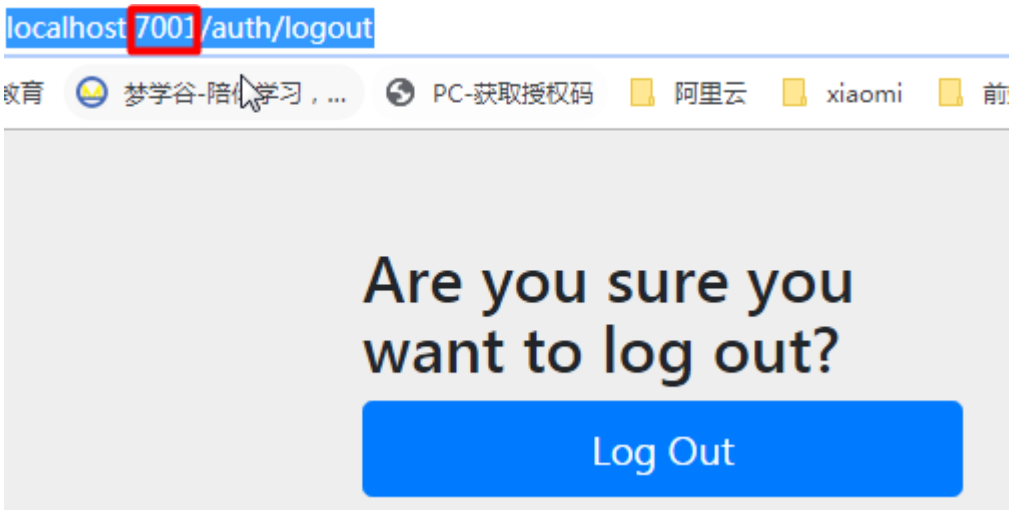
测试客户端

登录请求是通过网关转发的



退出请求也是通过网关转发的

仅供购买者学习，禁止盗版、转卖、传播课程



7.7 客户端请求资源服务器

客户端传递令牌

当客户端要请求资源服务器中的资源时，我们需要带上令牌给资源服务器，由于我们使用了 @EnableOAuth2Sso 注解，SpringBoot 会在请求上下文中添加一个 OAuth2ClientContext 对象，而我们只要在配置类中向容器中添加一个 OAuth2RestTemplate 对象，请求的资源服务器时就会把令牌带上转发过去。

- 在mengxuegu-cloud-oauth2-sso-client1 中的 com.mengxuegu.oauth2.sso.config.SsoSecurityConfig 配置中，向容器中添加 OAuth2RestTemplate 实例

```
@Bean
public OAuth2RestTemplate restTemplate(UserInfoRestTemplateFactory factory) {
    return factory.getUserInfoRestTemplate();
}
```

使用 OAuth2RestTemplate 请求资源

在客户端的 com.mengxuegu.oauth2.sso.controller.MainController 注入 OAuth2RestTemplate 实例，

我们就使用它发送带有令牌的请求到受保护的资源服务器。

```
package com.mengxuegu.oauth2.sso.controller;

import com.mengxuegu.base.result.MengxueguResult;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.oauth2.client.OAuth2RestTemplate;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

/**
 * @Auther: 梦学谷 www.mengxuegu.com
 */
```

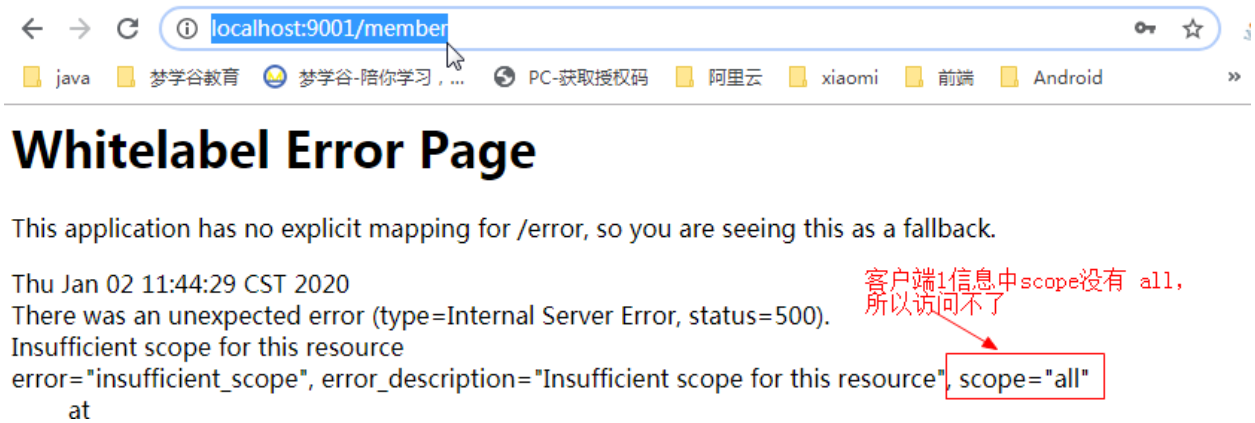
仅供购买者学习，禁止盗版、转卖、传播课程

```
12  @Controller
13  public class MainController {
14
15      @GetMapping("/")
16      public String index() {
17          return "index";
18      }
19
20  +   @Autowired
21  +   private OAuth2RestTemplate restTemplate;
22
23      @GetMapping("/member")
24      public String member() {
25  +       MengxueguResult result =
26  +           restTemplate.getForObject("http://localhost:8080/product/list",
27                                  MengxueguResult.class);
28  +       System.out.println("获取商品信息：" + result);
29          return "member";
30      }
31
32  }
33
```

上面直接请求的是资源服务器，还没有通过网关

无 scope 权限解决

1. 重启客户端：
2. 访问 http://localhost:9001/ 客户端1 ，然后进行登录，跳回到会员页面，报 500 没有 scope="all" 权限：



报错原因：

因为在 mengxuegu-cloud-oauth2-resource-product 资源服务器的 com.mengxuegu.oauth2.resource.config.ResourceServerConfig 指定了所有请求要有 `all` 范围

```
1  .antMatchers("/**").access("#oauth2.hasScope('all')")
```

解决问题：

在数据表 oauth_client_details 中为客户端1 `client1` 中的 scope 字段添加 `all` ，

仅供购买者学习，禁止盗版、转卖、传播课程

对象 | oauth_client_details @study-...

开始事务 | 备注 | 筛选 | 排序 | 导入 | 导出

client_id	client1
resource_ids	
client_secret	$2a$10$AodcXpujNILcZAQ.7cQ0b.Bm4klRumapoFBu7
scope	MEMBER_READ,MEMBER_WRITE,all
authorized_grant_ty...	authorization_code,refresh_token
web_server_redirec...	http://localhost:9001/login
authorities	
access_token_validi...	50000
refresh_token_validi...	
additional_informat...	
autoapprove	true

通过网关请求资源

客户端先通过网关，再转发请求到资源服务器

1. 在客户端的 com.mengxuegu.oauth2.sso.controller.MainController 把端口号改为7001 ：

```
@Controller
public class MainController {


    @GetMapping("/")
    public String index() { return "index"; }

    @Autowired
    private OAuth2RestTemplate restTemplate;

    @GetMapping("/member")
    public String member() {
        MengxueguResult result =
                restTemplate.getForObject( url: "http://localhost:7001/product/list", MengxueguResult.class);
        System.out.println("获取商品信息: " + result);
        return "member";
    }

}
```

2. 重启客户端
3. 访问 http://localhost:9001/ 客户端1 ，登录跳回到会员页面，报 500 没有 **scope="PRODUCT_API"** 权限：

仅供购买者学习，禁止盗版、转卖、传播课程

localhost:9001/member

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Thu Jan 02 12:46:54 CST 2020
There was an unexpected error (type=Internal Server Error, status=500).
Insufficient scope for this resource
error="insufficient_scope", error_description="Insufficient scope for this resource", scope="PRODUCT_API"

报错原因

因为在网关服务 mengxuegu-cloud-gateway 中的 com.mengxuegu.oauth2.config.ResourceServerConfig 资源配置类中对商品服务 `/product/**` 要有 `PRODUCT_API` 范围

```
resources.resourceId(RESOURCE_ID)
        .tokenStore(tokenStore)
    ;
}

@Override
public void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
            .antMatchers( ...antPatterns: "/product/**")
            .access( attribute: "#oauth2.hasScope('PRODUCT_API')");
}
```

4. **解决问题：**

在数据表 oauth_client_details 中为客户端1 `client1` 中的 scope 字段再添加 `PRODUCT_API`

client_id	client1
resource_ids	
client_secret	$2a$10$AodcXpujNILcZAQ.7cQ0b.Bm4klRumapoFBu7uyQ/ZV9Nu9F5fE3y
scope	MEMBER_READ,MEMBER_WRITE,all,PRODUCT_API
authorized_grant_ty...	authorization_code,refresh_token
web_server_redirec...	http://localhost:9001/login
authorities	
access_token_validi...	50000
refresh_token_validi...	
additional_informat...	
autoapprove	true

5. 退出重新登录可完成访问，然后查看idea控制台打印的日志

```
12:51:35.907  INFO 10184 --- [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver
获取商品信息: MengxueguResult(code=200, message=OK, data=[眼镜, 格子衬衣, 双肩包, admin])
```

附录

仅供购买者学习，禁止盗版、转卖、传播课程

Spring EL 权限表达式

Spring Security 允许我们使用 Spring EL 表达式，来进行用户权限的控制，如果对应的表达式结果返回true，则表示拥有对应的权限，反之则无。

Spring Security 可用表达式对象的基类是 `SecurityExpressionRoot` 参见此类即可。

注意: 表达式以下面为准, 黄色标注的 is 在使用时是要加上的

表达式	描述
permitAll()	总是返回true，表示允许所有访问（认证不认证都可访问 URL或方法）
denyAll()	总是返回false，表示拒绝所有访问（永远访问不到指定的 URL或方法）
isAnonymous()	当前用户是一个匿名用户（未登录用户）允许访问，返回true
isRememberMe()	当前用户是通过Remember-Me自动登录的允许访问，返回true
isAuthenticated()	当前用户是已经登录认证成功的允许访问（包含了rememberMe自动登录的），返回true
isFullyAuthenticated()	如果当前用户既不是一个匿名用户，同时也不是通过Remember-Me自动登录的，则允许访问（可以理解为通过页面输入帐户信息认证的）。
hasRole(String role)	当前用户拥有指定角色权限的允许访问，返回true。注意: 指定的角色名(如：`ADMIN`) SpringSecurity 底层会在前面拼接 `ROLE_` 字符串，所以在UserDetailsService实现类，数据库返回的角色名要有 `ROLE_ADMIN`
hasAnyRole([role1, role2])	多个角色以逗号分隔的字符串。如果当前用户拥有指定角色中的任意一个则允许访问，返回true。
hasAuthority(String authority)	当前用户拥有指定权限标识的允许访问，返回true。注意：和 `hasRole` 区别是，`hasAuthority` 不会在前面拼接 `ROLE_` 字符串, 。
hasAnyAuthority([auth1,auth2])	多个权限标识是以逗号分隔的字符串。如果当前用户拥有指定权限标识中的任意一个则允许访问，返回true
hasIpAddress("192.168.1.1/29")	限制指定IP或指定范围内的IP才可以访问

OAuth2 相关表结构

Spring 官方提供了存储 OAuth2 相关信息的数据库表结构，

```
https://github.com/spring-projects/spring-security-oauth/blob/master/spring-security-oauth2/src/test/resources/schema.sql
```

仅供购买者学习，禁止盗版、转卖、传播课程

当前使用了 MySQL 数据库，要修改下数据类型：

1. 官方提供的表结构主键类型为 `VARCHAR(256)` ，超过了MySQL限制的长度 `128` ，需要修改为 `VARCHAR(128)`
2. 将 `LONGVARBINARY` 类型修改为 `BLOB` 类型。

   修改后的表结构如下：

```
create table oauth_client_details (
  client_id VARCHAR(128) PRIMARY KEY,
  resource_ids VARCHAR(128),
  client_secret VARCHAR(128),
  scope VARCHAR(128),
  authorized_grant_types VARCHAR(128),
  web_server_redirect_uri VARCHAR(128),
  authorities VARCHAR(128),
  access_token_validity INTEGER,
  refresh_token_validity INTEGER,
  additional_information VARCHAR(4096),
  autoapprove VARCHAR(128)
);

create table oauth_client_token (
  token_id VARCHAR(128),
  token BLOB,
  authentication_id VARCHAR(128) PRIMARY KEY,
  user_name VARCHAR(128),
  client_id VARCHAR(128)
);

create table oauth_access_token (
  token_id VARCHAR(128),
  token BLOB,
  authentication_id VARCHAR(128) PRIMARY KEY,
  user_name VARCHAR(128),
  client_id VARCHAR(128),
  authentication BLOB,
  refresh_token VARCHAR(128)
);

create table oauth_refresh_token (
  token_id VARCHAR(128),
  token BLOB,
  authentication BLOB
);

create table oauth_code (
  code VARCHAR(128),
  authentication BLOB
);

create table oauth_approvals (
    userId VARCHAR(128),

    clientId VARCHAR(128),
```

仅供购买者学习，禁止盗版、转卖、传播课程

```
  scope VARCHAR(128),
  status VARCHAR(10),
  expiresAt TIMESTAMP,
  lastModifiedAt TIMESTAMP
);


-- customized oauth_client_details table
create table ClientDetails (
  appId VARCHAR(128) PRIMARY KEY,
  resourceIds VARCHAR(128),
  appSecret VARCHAR(128),
  scope VARCHAR(128),
  grantTypes VARCHAR(128),
  redirectUrl VARCHAR(128),
  authorities VARCHAR(128),
  access_token_validity INTEGER,
  refresh_token_validity INTEGER,
  additionalInformation VARCHAR(4096),
  autoApproveScopes VARCHAR(128)
);
```

Oauth2 表字段说明

oauth_client_details

仅供购买者学习，禁止盗版、转卖、传播课程

字段名	字段说明
client_id	主键,必须唯一,不能为空. 用于唯一标识每一个客户端(client); 在注册时必须填写(也可由服务端自动生成). 对于不同的grant_type,该字段都是必须的. 在实际应用中的另一个名称叫appKey,与client_id是同一个概念.
resource_ids	客户端能访问的资源id集合，多个用逗号分隔，注册客户端时，根据实际需要可选择资源id，也可以根据不同的额注册流程，赋予对应的额资源id
client_secret	必须填写 ，用于指定客户端(client)的访问密匙; 在注册时必须填写(也可由服务端自动生成). 在实际应用中的另一个名称叫appSecret,与client_secret是同一个概念.
scope	指定client的权限范围，比如移动端还是web端权限 ，哪个微服务权限
authorized_grant_types	指定客户端支持的grant_type,可选值包括authorization_code,password,refresh_token,implicit,client_credentials, 若支持多个grant_type用逗号(,)分隔,如: `authorization_code,password`，在实际应用中,当注册时,该字段是一般由服务器端指定的,而不是由申请者去选择的,最常用的grant_type组合有: "authorization_code,refresh_token"(针对通过浏览器访问的客户端); "password,refresh_token"(针对移动设备的客户端). implicit与client_credentials在实际中很少使用.
web_server_redirect_uri	客户端的重定向URI,可为空, 当grant_type为authorization_code或implicit时, 在Oauth的流程中会使用并检查与注册时填写的redirect_uri是否一致. 下面分别说明: 当grant_type=authorization_code时, 第一步 从 spring-oauth-server获取 'code'时客户端发起请求时必须有redirect_uri参数, 该参数的值必须与web_server_redirect_uri的值一致. 第二步 用 'code' 换取 'access_token' 时客户也必须传递相同的redirect_uri. 在实际应用中, web_server_redirect_uri在注册时是必须填写的, 一般用来处理服务器返回的code, 验证state是否合法与通过code去换取access_token值. 在spring-oauth-client项目中, 可具体参考AuthorizationCodeController.java中的authorizationCodeCallback方法. 当grant_type=implicit时通过redirect_uri的hash值来传递access_token值.如: http://localhost:7777/spring-oauth-client/implicit#access_token=dc891f4a-ac88-4ba6-8224-a2497e013865&token_type=bearer&expires_in=43199 然后客户端通过JS等从hash值中取到access_token值.
authorities	指定客户端所拥有的Spring Security的权限值,可选, 若有多个权限值,用逗号(,)分隔, 如: "ROLE_UNITY,ROLE_USER". 对于是否要设置该字段的值,要根据不同的grant_type来判断, 若客户端在Oauth流程中需要用户的用户名(username)与密码(password)的(authorization_code,password), 则该字段可以不需要设置值,因为服务端将根据用户在服务端所拥有的权限来判断是否有权限访问对应的API. 但如果客户端在Oauth流程中不需要用户信息的(implicit,client_credentials), 则该字段必须要设置对应的权限值, 因为服务端将根据该字段值的权限来判断是否有权限访问对应的API.
access_token_validity	设定客户端的access_token的有效时间值(单位:秒),可选, 若不设定值则使用默认的有效时间值(60 * 60 * 12, 12小时). 在服务端获取的access_token JSON数据中的expires_in字段的值即为当前access_token的有效时间值. 在项目中, 可具体参考DefaultTokenServices.java中属性accessTokenValiditySeconds. 在实际应用中, 该值一般是由服务端处理的, 不需要客户端自定义.
refresh_token_validity	设定客户端的refresh_token的有效时间值(单位:秒),可选, 若不设定值则使用默认的有效时间值(60 * 60 * 24 * 30, 30天). 若客户端的grant_type不包括refresh_token,则不用关心该字段 在项目中, 可具体参考DefaultTokenServices.java中属性refreshTokenValiditySeconds. 在实际应用中, 该值一般是由服务端处理的, 不需要客户端自定义.
additional_information	这是一个预留的字段,在Oauth的流程中没有实际的使用,可选,但若设置值,必须是JSON格式的数据,如: {"country":"CN","country_code":"086"} 按照spring-security-oauth项目中对该字段的描述 Additional information for this client, not need by the vanilla OAuth protocol but might be useful, for example,for storing descriptive information. (详见ClientDetails.java的getAdditionalInformation()方法的注释)在实际应用中, 可以用该字段来存储关于客户端的一些其他信息,如客户端的国家,地区,注册时的IP地址等等.
autoapprove	设置用户是否自动Approval操作, 默认值为 'false', 可选值包括 'true','false', 'read','write'. 该字段只适用于grant_type="authorization_code"的情况,当用户登录成功后,若该值为'true'或支持的scope值,则会跳过用户Approve的页面, 直接授权. 是 spring-security-oauth2 的 2.0 版本后添加的新属性.

仅供购买者学习，禁止盗版、转卖、传播课程

oauth_access_token

向客户端响应的访问令牌信息保存到此表中.

字段名	字段说明
create_time	数据的创建时间,精确到秒,由数据库在插入数据时取当前系统时间自动生成(扩展字段)
token_id	该字段的值是将access_token的值通过MD5加密后存储的.
token	存储将OAuth2AccessToken.java对象序列化后的二进制数据, 是真实的AccessToken的数据值.
authentication_id	该字段具有唯一性, 其值是根据当前的username(如果有),client_id与scope通过MD5加密生成的. 具体实现请参考DefaultAuthenticationKeyGenerator.java类.
user_name	登录时的用户名, 若客户端没有用户名(如grant_type="client_credentials"),则该值等于client_id
client_id	
authentication	存储将OAuth2Authentication.java对象序列化后的二进制数据.
refresh_token	该字段的值是将refresh_token的值通过MD5加密后存储的. 在项目中,主要操作oauth_access_token表的对象是JdbcTokenStore.java. 更多的细节请参考该类.

oauth_client_token

该表用于在客户端系统中存储从服务端获取的token数据, 在spring-oauth-server项目中未使用到. 对oauth_client_token表的主要操作在JdbcClientTokenServices.java类中, 更多的细节请参考该类.

字段名	字段说明
create_time	数据的创建时间,精确到秒,由数据库在插入数据时取当前系统时间自动生成(扩展字段)
token_id	从服务器端获取到的access_token的值.
token	这是一个二进制的字段, 存储的数据是OAuth2AccessToken.java对象序列化后的二进制数据.
authentication_id	该字段具有唯一性, 是根据当前的username(如果有),client_id与scope通过MD5加密生成的. 具体实现请参考DefaultClientKeyGenerator.java类.
user_name	登录时的用户名
client_id	

oauth_refresh_token

仅供购买者学习，禁止盗版、转卖、传播课程

在项目中,主要操作oauth_refresh_token表的对象是JdbcTokenStore.java. (与操作oauth_access_token表的对象一样);更多的细节请参考该类. 如果客户端的grant_type不支持refresh_token,则不会使用该表.

字段名	字段说明
create_time	数据的创建时间,精确到秒,由数据库在插入数据时取当前系统时间自动生成(扩展字段)
token_id	该字段的值是将refresh_token的值通过MD5加密后存储的.
token	存储将OAuth2RefreshToken.java对象序列化后的二进制数据.
authentication	存储将OAuth2Authentication.java对象序列化后的二进制数据.

oauth_code

在项目中,主要操作oauth_code表的对象是JdbcAuthorizationCodeServices.java. 更多的细节请参考该类. 只有当grant_type为"authorization_code"时,该表中才会有数据产生; 其他的grant_type没有使用该表.

字段名	字段说明
code	存储服务端系统生成的code的值(未加密).
authentication	存储将AuthorizationRequestHolder.java对象序列化后的二进制数据.

仅供购买者学习，禁止盗版、转卖、传播课程