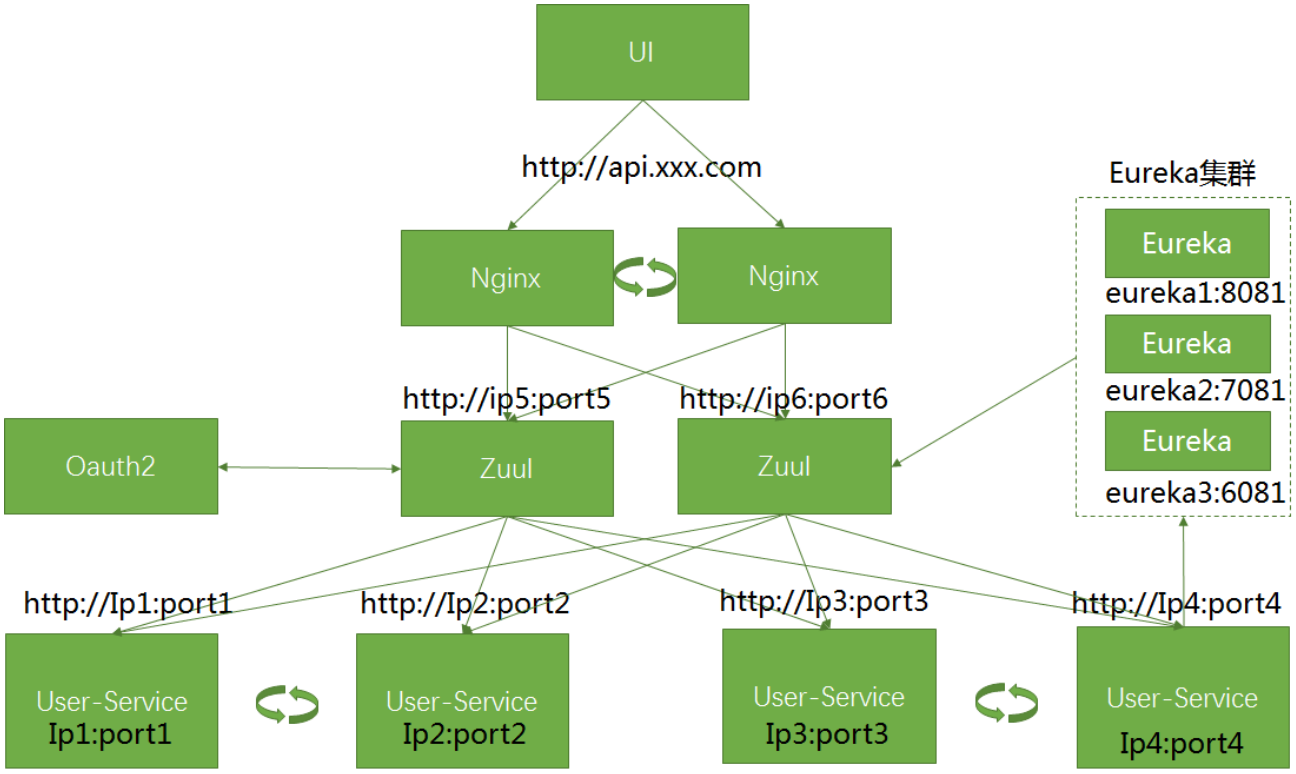
第七章 SpringCloud OAuth2认证中心-搭建认证中心

本章完整源码地址：https://github.com/kwang2003/springcloud-study-ch07.git

1.项目概要

这一章节的内容以第六章的代码为基础改造而成https://github.com/kwang2003/springcloud-study-ch06.git。

经过前几个章节的内容，我们的微服务项目架构逐渐完善了起来，这一章节的重点是通过给已有的微服务增加oauth2安全认证功能来保证服务调用的安全性，我们要实现的目标架构如下：

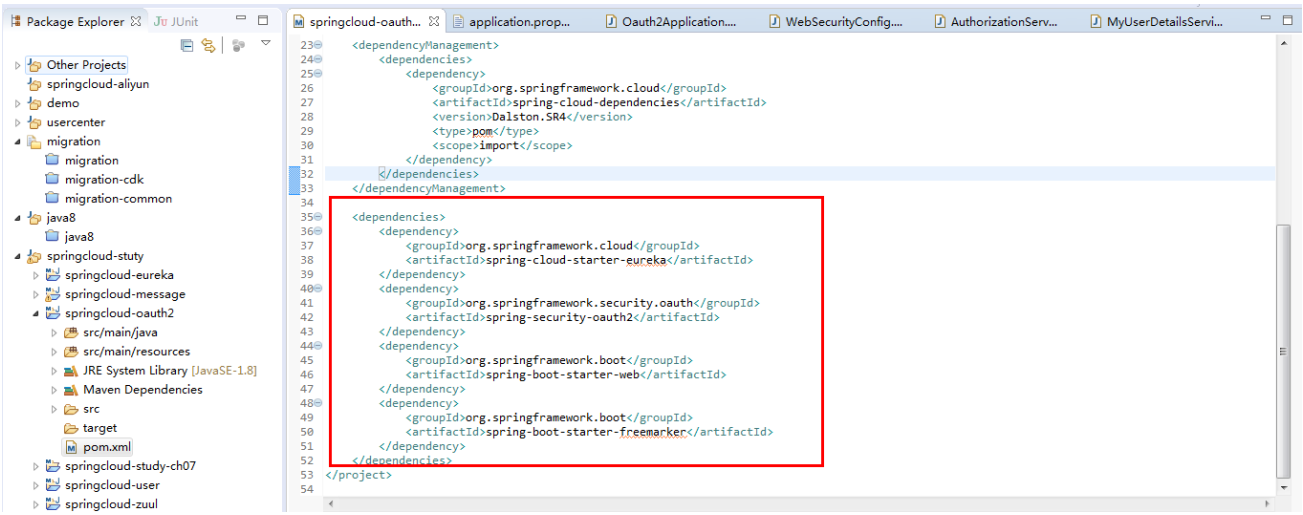


因为所有的外部请求都统一经过zuul网关，因此我们的Oauth2认证添加在Zuul这一层上，而每个微服务之间的调用则认为是项目内部模块之间的调用，不需要进行oauth2认证过程。

2.项目增加ouauth2模块

在我们的项目中增加一个新的模块springcloud-oauth2，并增加依赖

```
<dependency>
 <groupId>org.springframework.cloud</groupId>
 <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
<dependency>
 <groupId>org.springframework.security.oauth</groupId>
 <artifactId>spring-security-oauth2</artifactId>
</dependency>
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-freemarker</artifactId>
</dependency>
```



修改下工程占用的端口号

server.port=8888

eureka.instance.hostname=oauth2

eureka.client.serviceUrl.defaultZone=http://eureka1:8081/eureka/,http://eureka2:7081/eureka/,http://eureka3:6081/eureka/

3.自定义登录页面和授权确认页面

创建Oauth2Application 类，所谓项目的主函数所在类，继承自WebMvcConfigurerAdapter，自定义登录页面和授权页面的地址

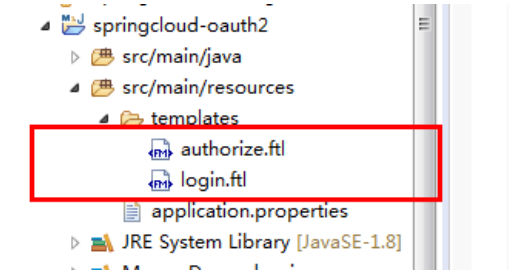
```
package com.pachiraframework.springcloud.oauth2;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@EnableDiscoveryClient
@SpringBootApplication
public class Oauth2Application extends WebMvcConfigurerAdapter{
 public static void main(String[] args) {
  SpringApplication.run(Oauth2Application.class, args);
 }

 @Override
 public void addViewControllers(ViewControllerRegistry registry) {
  registry.addViewController("/login").setViewName("login");
  registry.addViewController("/oauth/confirm_access").setViewName("authorize");
 }
}
```

同时在src/main/resources下建立templates目录，增加2个ftl文件



authorize.ftl

```
<html>
<head>
<script src="https://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.js"></script>
<link href="https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.css" rel="stylesheet">
</head>
```

```
<body>
<div class="container">
    <h2>Please Confirm</h2>
    <p>
        Do you authorize "${authorizationRequest.clientId}" at "${authorizationRequest.redirectUri}" to access your
        protected resources
        with scope ${authorizationRequest.scope?join(", ")}.
    </p>
    <form id="confirmationForm" name="confirmationForm"
          action="../oauth/authorize" method="post">
        <#list authorizationRequest.scope as scop>
      <input type="hidden" name="scope.${scop}" value="true"/>
  </#list>
        <input name="user_oauth_approval" value="true" type="hidden"/>
        <input type="hidden" id="csrf_token" name="${_csrf.parameterName}" value="${_csrf.token}"/>
        <button class="btn btn-primary" type="submit">Approve</button>
    </form>
    <form id="denyForm" name="confirmationForm"
          action="../oauth/authorize" method="post">
        <input name="user_oauth_approval" value="false" type="hidden"/>
        <input type="hidden" id="csrf_token" name="${_csrf.parameterName}" value="${_csrf.token}"/>
        <button class="btn btn-primary" type="submit">Deny</button>
    </form>
</div>
</body>
</html>
```

login.ftl

```
<html>
<head>
<link rel="stylesheet" href="https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
</head>
<body>
 <div class="container">
  <form role="form" action="login" method="post">
    <div class="form-group">
      <label for="username">Username:</label>
      <input type="text" class="form-control" id="username" name="username"/>
    </div>
    <div class="form-group">
      <label for="password">Password:</label>
      <input type="password" class="form-control" id="password" name="password"/>
    </div>
    <input type="hidden" id="csrf_token" name="${_csrf.parameterName}" value="${_csrf.token}"/>
    <button type="submit" class="btn btn-primary">Submit</button>
  </form>
 </div>
<script src="https://cdn.bootcss.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-
Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>
</body>
</html>
```

4.实现UserDetailsService

实际的项目中，我们的登录用户数据可能存在数据库中，也可能是存放在ladap或其他微服务接口中，springcloud oauth2给我们提供了一个UserDetailsService接口

UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
在项目中，我们需要自行实现这个接口来获取用户信息。在这个例子中，我们在接口中写死了一个用户名admin,并为其手动指定了一个角色admin

```
package com.pachiraframework.springcloud.oauth2.service;

import java.util.Collection;
import java.util.HashSet;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
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.stereotype.Service;

@Service
public class MyUserDetailsService implements UserDetailsService {
 @Override
 public UserDetails loadUserByUsername(String name) throws UsernameNotFoundException {
  if ("admin".equalsIgnoreCase(name)) {
   User user = mockUser();
   return user;
  }
  return null;
 }

 private User mockUser() {
  Collection<GrantedAuthority> authorities = new HashSet<>();
  authorities.add(new SimpleGrantedAuthority("admin"));//用户所拥有的角色信息
  User user = new User("admin","123456",authorities);
  return user;
 }
}
```

5.配置Oauth2Server和SecurityConfig

```
package com.pachiraframework.springcloud.oauth2.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import com.pachiraframework.springcloud.oauth2.service.MyUserDetailsService;

@Order(10)
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
 @Autowired
 private MyUserDetailsService userDetailsFitService;//注意，这个MyUserDetailsService就是上个步骤中定义的bean
 @Override
 @Bean
```

```
 public AuthenticationManager authenticationManagerBean() throws Exception {
  return super.authenticationManagerBean();
 }

 @Override
 protected void configure(HttpSecurity http) throws Exception {
  http.formLogin().loginPage("/login").permitAll().and().authorizeRequests().antMatchers("/health", "/css/**")
    .anonymous().and().authorizeRequests().anyRequest().authenticated();
 }

 @Override
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {
  auth.userDetailsService(userDetailsFitService);
  auth.parentAuthenticationManager(authenticationManagerBean());
 }
}
```

在这个例子中，我们把client的信息保存在内存中

```
package com.pachiraframework.springcloud.oauth2.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
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;
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecurityConfigurer;

import com.pachiraframework.springcloud.oauth2.service.MyUserDetailsService;

@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfiguration extends AuthorizationServerConfigurerAdapter {
 @Autowired
 private AuthenticationManager authenticationManager;
 @Autowired
 private MyUserDetailsService userDetailsService;

 @Override
 public void configure(final AuthorizationServerSecurityConfigurer oauthServer) throws Exception {
  oauthServer.tokenKeyAccess("permitAll()").checkTokenAccess("isAuthenticated()");
 }

    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
     clients.inMemory() // 使用in-memory存储
     .withClient("client") // client_id
     .secret("secret") // client_secret
     .authorizedGrantTypes("authorization_code") // 该client允许的授权类型
     .scopes("app"); // 允许的授权范围
    }

 @Override
 public void configure(final AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
  // @formatter:off
```

```
    endpoints.authenticationManager(authenticationManager)
        .userDetailsService(userDetailsService);
    // @formatter:on
  }
}
```

6.访问oauth2服务

http://localhost:8888/oauth/authorize?response_type=code&client_id=client&redirect_uri=http://baidu.com&state=123

出现登录页面，输入用户名：admin 密码：123456

点击Submit按钮，进入用户授权确认页面

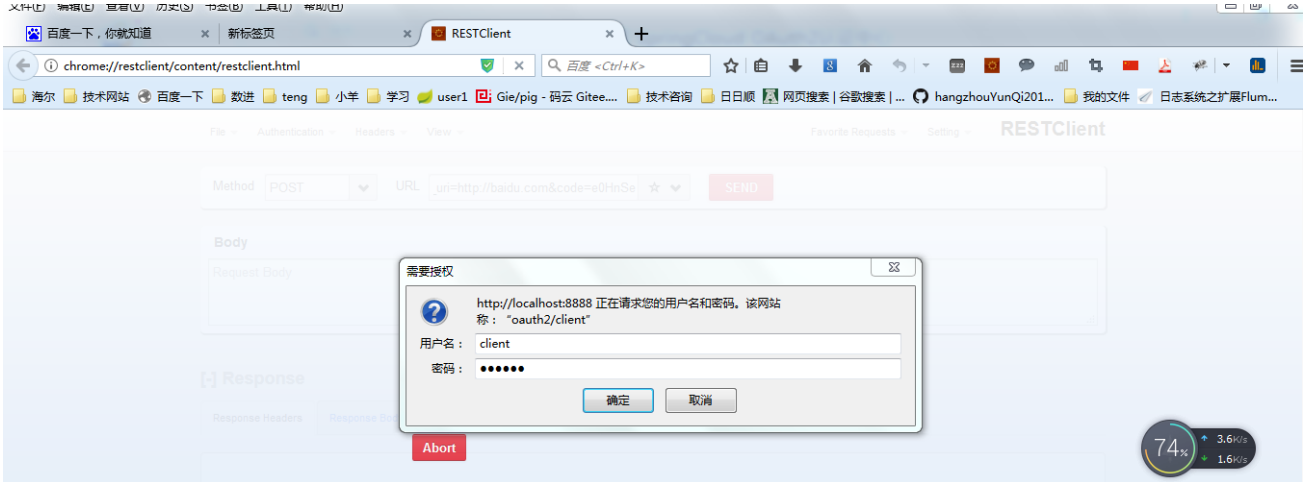
点击Approve,跳转到baidu页面，后面携带了code和state参数

https://www.baidu.com/?code=F7LsMB&state=123

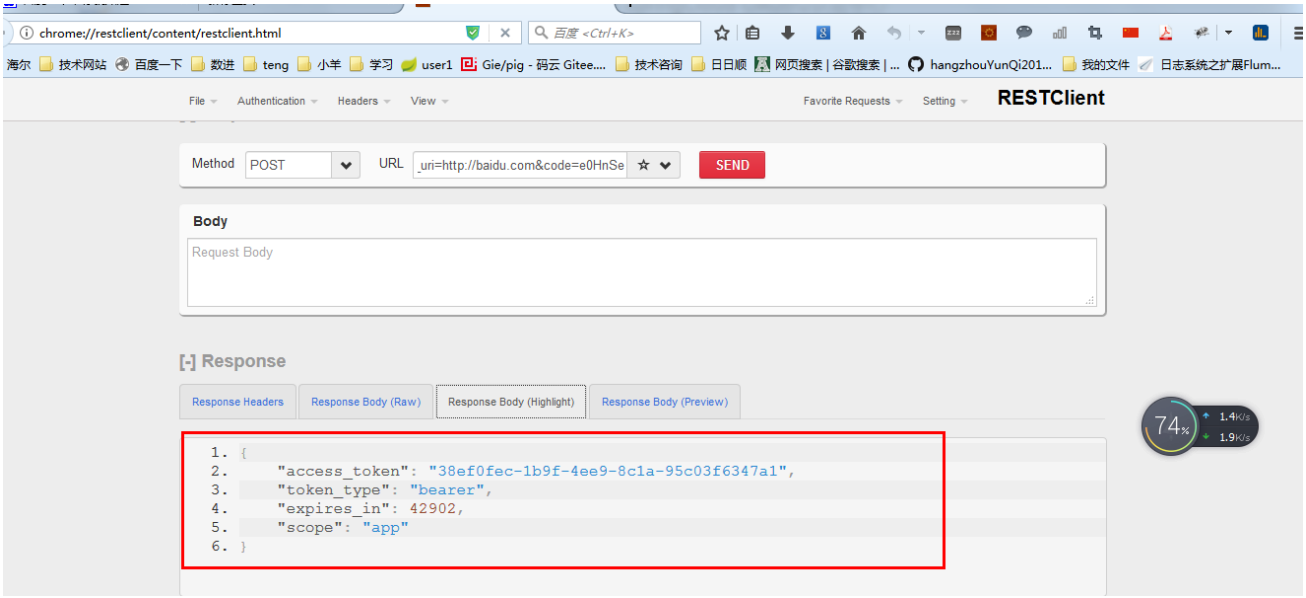
根据code换取access_code，注意使用post方法

http://localhost:8888/oauth/token?client_id=client&grant_type=authorization_code&redirect_uri=http://baidu.com&code=F7LsMB

注意这个code要和上个步骤中获得的code保持一致



用户名输入client，密码是secret,点击确定




```
{
    "access_token": "38ef0fec-1b9f-4ee9-8c1a-95c03f6347a1",
    "token_type": "bearer",
    "expires_in": 42902,
    "scope": "app"
}
```