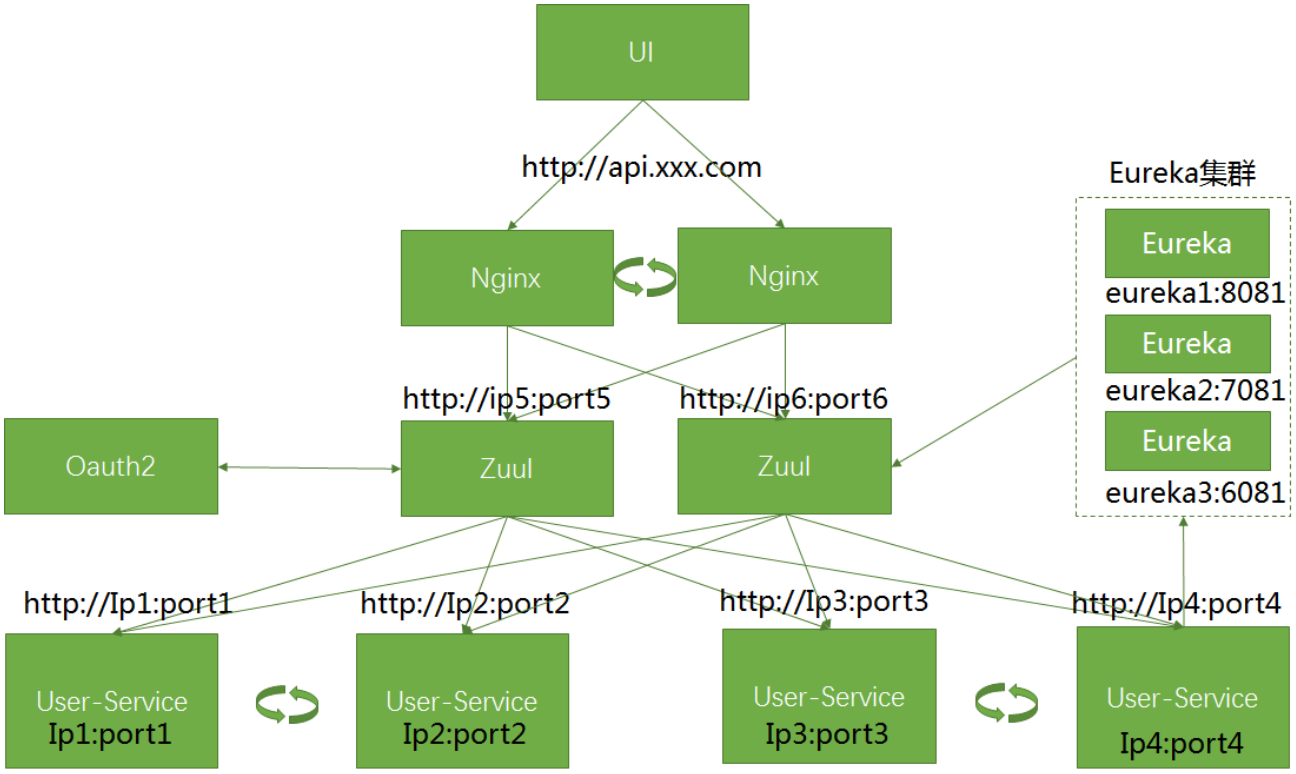
第九章 SpringCloud Oauth2认证中心-Zuul网关上添加认证

本章完整源码地址：https://github.com/kwang2003/springcloud-study-ch09.git

1.项目概要

这一章节的内容以第七章的代码为基础改造而成https://github.com/kwang2003/springcloud-study-ch08.git。

通过第八章的学习，我们已经已经基于JWT升级了OAuth2认证服务器，在这个章节中，我们将给之前的zuul网关上加上OAuth2认证功能，让所有的访问都经过一次安全认证后才能继续。系统架构和上章一样，保持不变

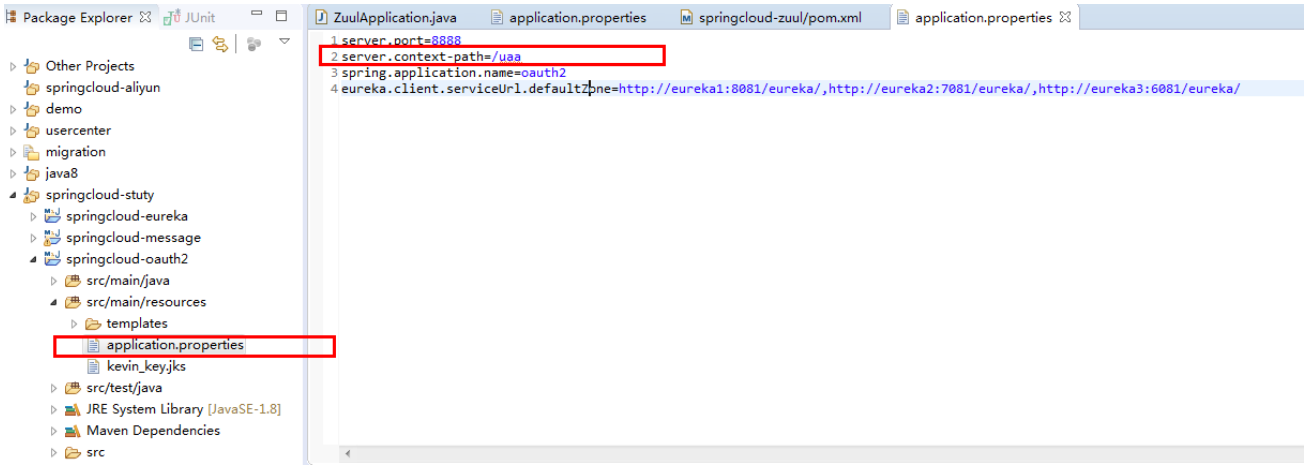


2.OAuth2项目改动

a)为oauth2项目增加/uaa访问上下文

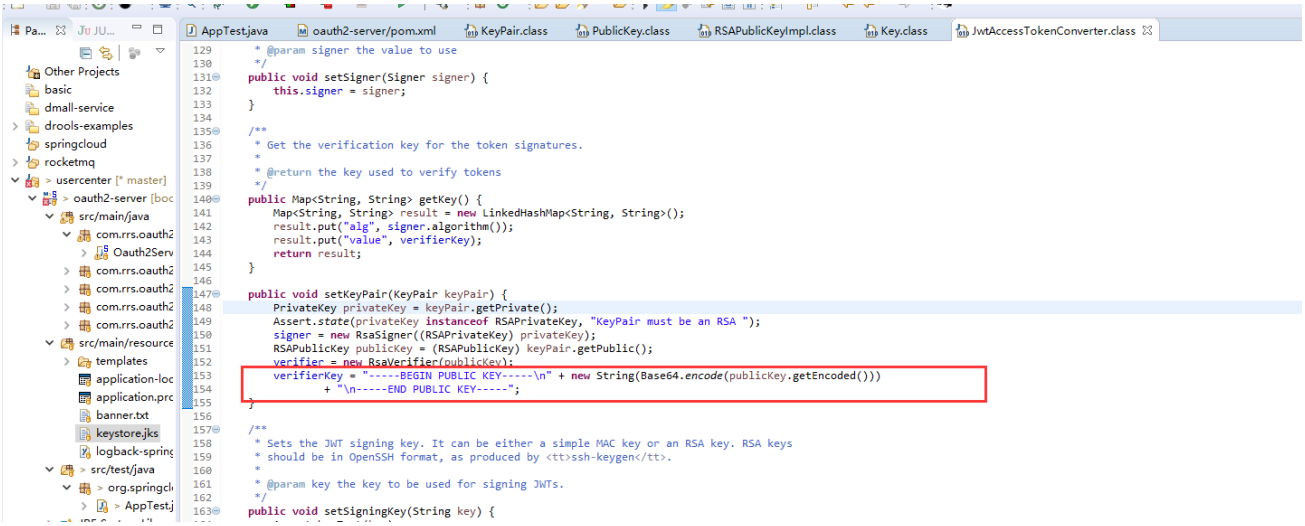
了配置以后，oauth2工程的访问地址就要加上/uaa这个上下文了，加这个上下文主要是为后一个步骤做转发规则使用

server.context-path=/uaa

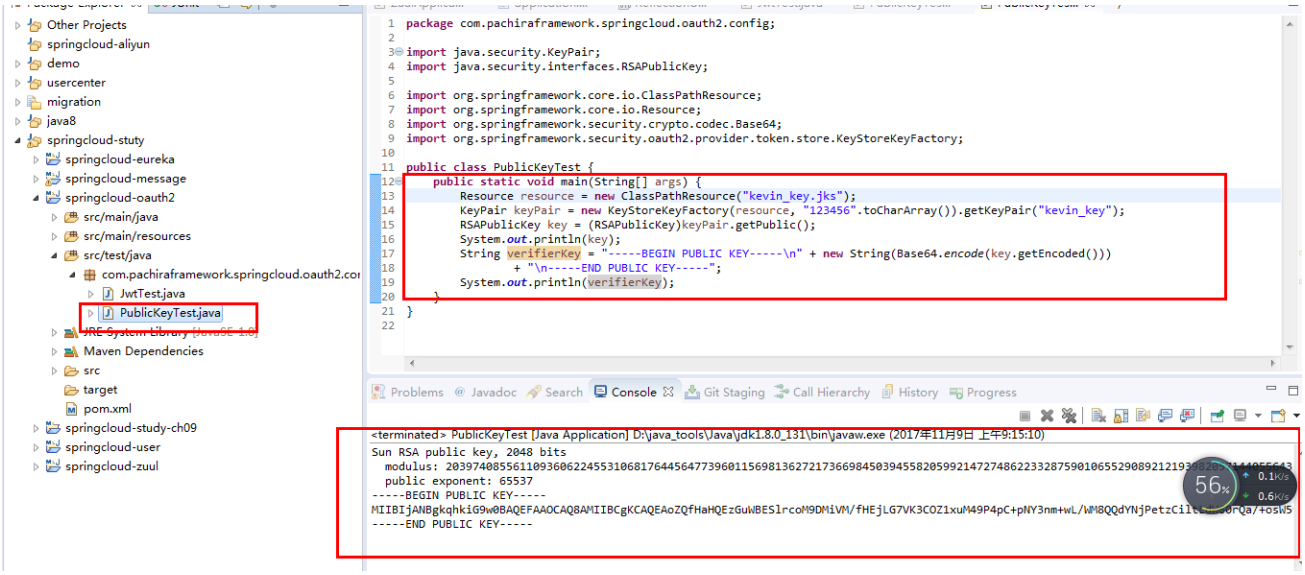


b)获取public key

通过查看JwtAccessTokenConverter的源码了解到，vertify key 是对publicKey进行Base64编码后得到的一个字符串：



增加一个测试类PublicKeyTest，按照规则生成这个字符串，代码如下：



得到输出的key结果是

-----BEGIN PUBLIC KEY-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoZQfHaHQEzGuWBESlrcoM9DMiVM/fHEjLG7VK3COZ1xuM49P4pC+pNY3nm+wL

/WM8QQdYNjPetzCiltLdW60rQa/+osW599SkmUSGF7rYXI9y4n1N4h0k9jLBdZ9n2/5miWWPDwrbsIGgLXp/NIsH4T2Gj6lUZuMj4cskIjU8P12S5T

JEQ3N+PGYBY+G8zWzYB1dr3LssqATOqv1/XH+kPEesAtaaxJsfa/SWpaxDZnM5JGSjQ1/FEjyF2e0yAKbV/NqHnIqqnucr/StFBn/sGloEs1DAj8nIMt

oiklqejqJvjCSdOC75f2N3iK72DgAVZkSNtfEdsVygHiOOtBtWQIDAQAB

-----END PUBLIC KEY-----

3.网关项目改造springcloud-zuul

a)增加spring-cloud-starter-oauth2和jwt

```
<dependency>
 <groupId>org.springframework.cloud</groupId>
 <artifactId>spring-cloud-starter-oauth2</artifactId>
</dependency>
<dependency>
 <groupId>org.springframework.security</groupId>
 <artifactId>spring-security-jwt</artifactId>
</dependency>
```

b)修改系统配置项

新增jwt和oauth2的安全认证相关配置，如下

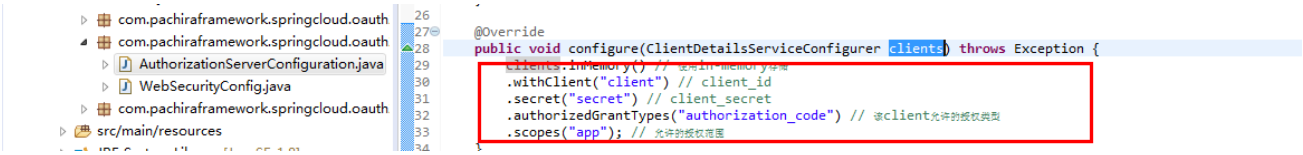
```
security.basic.enabled=false
security.oauth2.sso.loginPath=/login
security.oauth2.client.accessTokenUri=http://oauth2/uaa/oauth/token
security.oauth2.client.userAuthorizationUri=/uaa/oauth/authorize
security.oauth2.client.clientId=client
security.oauth2.client.clientSecret=secret
```

security.oauth2.resource.jwt.keyValue=\
-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAoZQfHaHQEzGuWBESlrcoM9DMiVM/fHEjLG7VK3COZ1xuM49P4pC+pNY3nm+wL/WM8QQdYNjPetzCiltLdW60rQa/+osW599SkmUSGF7rYXI9y4n1N4h0k9jLBdZ9n2/5miWWPDwrbsIGgLXp/NIsH4T2Gj6lUZuMj4cskIjU8P12S5TJEQ3N+PGYBY+G8zWzYB1dr3LssqATOqv1/XH+kPEesAtaaxJsfa/SWpaxDZnM5JGSjQ1/FEjyF2e0yAKbV/NqHnIqqnucr/StFBn/sGloEs1DAj8nIMtoiklqejqJvjCSdOC75f2N3iK72DgAVZkSNtfEdsVygHiOOtBtWQIDAQAB\
-----END PUBLIC KEY-----
security.oauth2.resource.id=zuul
security.oauth2.resource.serviceId=${PREFIX:}zuul
其中红色高亮部分为上个步骤通过PublicKeyTest输出的结果

c)设置ZuulApplication为ResourceServer

Oauth2中包含四个核心角色

- Resource Owner ----通常是用户，如张三
- Resource Server ----资源服务器，存放具体的数据，在这个项目里是Zuul（实际上是各个微服务，但是我们使用Zuul统一代理了，所以就把Resource Server设置在Zuul上了，这也是增加Zuul这一层的意义所在--我们不需要给每个微服务重复的去添加这些配置，而是在Zuul统一出口的地方添加一次就可以了）
- Authorization Server ----认证服务器，这个项目里是springcloud-oauth2服务
- Client ---需要使用OAuth2服务器进行集成的第三方应用，如我们的网站rrs.com要使用qq联合登录，那么rrs.com就是一个客户端Client,在项目中，client的信息定义在内存中了



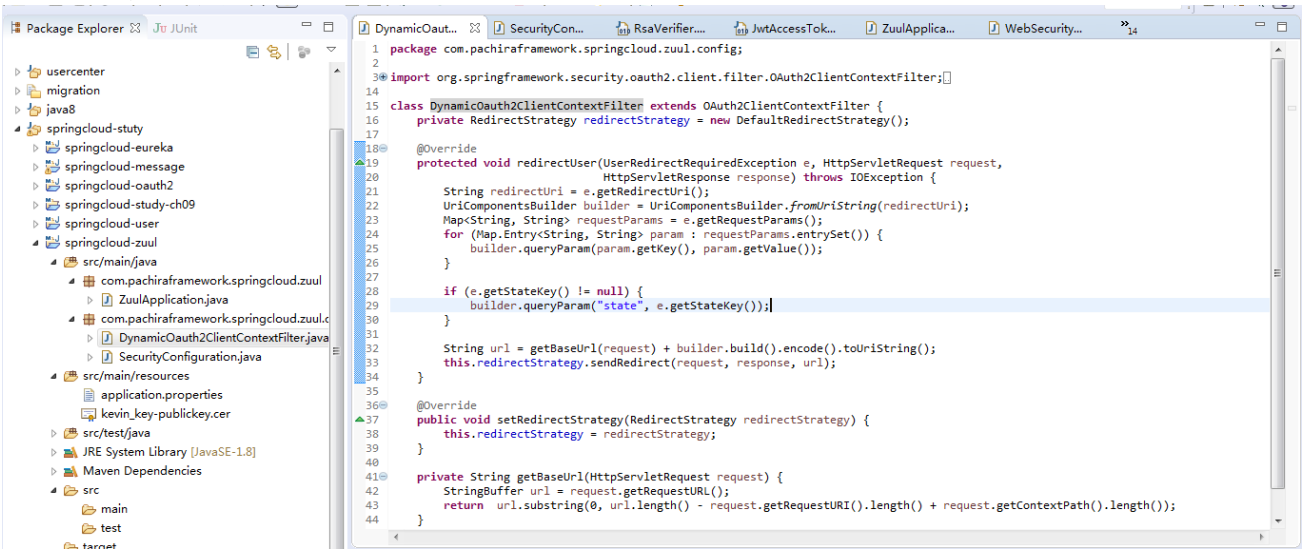
使ZuulApplication成为ResourceServer非常简单，在类上增加@EnableResourceServer注解：

```
package com.pachiraframework.springcloud.zuul;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;

@EnableZuulProxy
@EnableResourceServer
@EnableDiscoveryClient
@SpringBootApplication
public class ZuulApplication {
 public static void main(String[] args) {
  SpringApplication.run(ZuulApplication.class, args);
 }
}
```

并且要增加一个UserInfoRestTemplate的bean

```
    @Bean
    UserInfoRestTemplateCustomizer userInfoRestTemplateCustomizer(LoadBalancerInterceptor loadBalancerInterceptor) {
        return template -> {
            List<ClientHttpRequestInterceptor> interceptors = new ArrayList<>();
            interceptors.add(loadBalancerInterceptor);
            AccessTokenProviderChain accessTokenProviderChain = Stream
                    .of(new AuthorizationCodeAccessTokenProvider(), new ImplicitAccessTokenProvider(),
                            new ResourceOwnerPasswordAccessTokenProvider(), new ClientCredentialsAccessTokenProvider())
                    .peek(tp -> tp.setInterceptors(interceptors))
                    .collect(Collectors.collectingAndThen(Collectors.toList(), AccessTokenProviderChain::new));
            template.setAccessTokenProvider(accessTokenProviderChain);
        };
    }
```

d)增加一个DynamicOauth2ClientContextFilter



f)增加安全配置SecurityConfiguration

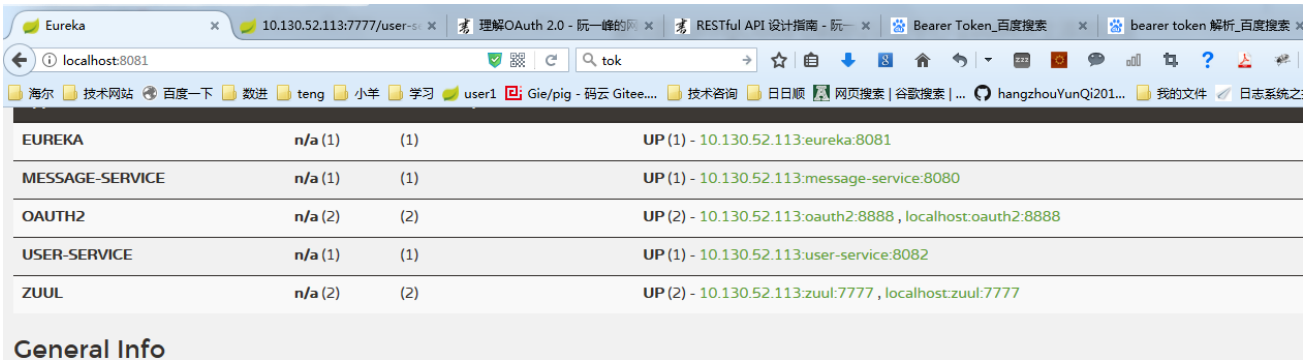
```
package com.pachiraframework.springcloud.zuul.config;

import java.io.IOException;
import java.util.regex.Pattern;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.autoconfigure.security.oauth2.client.EnableOAuth2Sso;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;
import org.springframework.core.annotation.Order;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.oauth2.client.filter.OAuth2ClientContextFilter;
import org.springframework.security.oauth2.provider.authentication.OAuth2AuthenticationManager;
import org.springframework.security.oauth2.provider.authentication.OAuth2AuthenticationProcessingFilter;
import org.springframework.security.oauth2.provider.token.ResourceServerTokenServices;
import org.springframework.security.web.authentication.preauth.AbstractPreAuthenticatedProcessingFilter;
```

```
import org.springframework.security.web.csrf.CsrfFilter;
import org.springframework.security.web.csrf.CsrfToken;
import org.springframework.security.web.csrf.CsrfTokenRepository;
import org.springframework.security.web.csrf.HttpSessionCsrfTokenRepository;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
import org.springframework.security.web.util.matcher.RequestMatcher;
import org.springframework.web.filter.OncePerRequestFilter;

@Configuration
@EnableOAuth2Sso
@Order(value = 0)
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    private static final String CSRF_COOKIE_NAME = "XSRF-TOKEN";
    private static final String CSRF_HEADER_NAME = "X-XSRF-TOKEN";

    @Autowired
    private ResourceServerTokenServices resourceServerTokenServices;

    @Bean
    @Primary
    public OAuth2ClientContextFilter dynamicOauth2ClientContextFilter() {
        return new DynamicOauth2ClientContextFilter();
    }

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests().antMatchers("/uaa/**", "/login").permitAll().anyRequest().authenticated()
            .and()
            .csrf().requireCsrfProtectionMatcher(csrfRequestMatcher()).csrfTokenRepository(csrfTokenRepository())
            .and()
            .addFilterAfter(csrfHeaderFilter(), CsrfFilter.class)
            .addFilterAfter(oAuth2AuthenticationProcessingFilter(), AbstractPreAuthenticatedProcessingFilter.class)
            .logout().permitAll()
            .logoutSuccessUrl("/");
    }

    private OAuth2AuthenticationProcessingFilter oAuth2AuthenticationProcessingFilter() {
        OAuth2AuthenticationProcessingFilter oAuth2AuthenticationProcessingFilter =
                new OAuth2AuthenticationProcessingFilter();
        oAuth2AuthenticationProcessingFilter.setAuthenticationManager(oauthAuthenticationManager());
        oAuth2AuthenticationProcessingFilter.setStateless(false);

        return oAuth2AuthenticationProcessingFilter;
    }


    private AuthenticationManager oauthAuthenticationManager() {
        OAuth2AuthenticationManager oAuth2AuthenticationManager = new OAuth2AuthenticationManager();
        oAuth2AuthenticationManager.setResourceId("zuul");
        oAuth2AuthenticationManager.setTokenServices(resourceServerTokenServices);
        oAuth2AuthenticationManager.setClientDetailsService(null);

        return oAuth2AuthenticationManager;
    }

    private RequestMatcher csrfRequestMatcher() {
```

```
        return new RequestMatcher() {
            // Always allow the HTTP GET method
            private final Pattern allowedMethods = Pattern.compile("^(GET|HEAD|OPTIONS|TRACE)$");

            // Disable CSFR protection on the following urls:
            private final AntPathRequestMatcher[] requestMatchers = { new AntPathRequestMatcher("/uaa/**") };

            @Override
            public boolean matches(HttpServletRequest request) {
                if (allowedMethods.matcher(request.getMethod()).matches()) {
                    return false;
                }

                for (AntPathRequestMatcher matcher : requestMatchers) {
                    if (matcher.matches(request)) {
                        return false;
                    }
                }
                return true;
            }
        };
    }

    private static Filter csrfHeaderFilter() {
        return new OncePerRequestFilter() {
            @Override
            protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
                                            FilterChain filterChain) throws ServletException, IOException {
                CsrfToken csrf = (CsrfToken) request.getAttribute(CsrfToken.class.getName());
                if (csrf != null) {
                    Cookie cookie = new Cookie(CSRF_COOKIE_NAME, csrf.getToken());
                    cookie.setPath("/");
                    cookie.setSecure(true);
                    response.addCookie(cookie);
                }
                filterChain.doFilter(request, response);
            }
        };
    }

    private static CsrfTokenRepository csrfTokenRepository() {
        HttpSessionCsrfTokenRepository repository = new HttpSessionCsrfTokenRepository();
        repository.setHeaderName(CSRF_HEADER_NAME);
        return repository;
    }
}
```

4.测试访问

本机地址是10.130.52.113(使用localhost访问会有问题， 具体原因尚未弄清楚)



http://10.130.52.113:7777/user-service/user/regist/sms?mobile=1234567895