一 创建auth server

##### 1,pom文件加第三方jar

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.5.2.RELEASE</version>  
 <relativePath/><!-- lookup parent from repository -->  
</parent>  
  
<dependencies>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</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>  
</dependencies>

##### 2,增加Application启动类

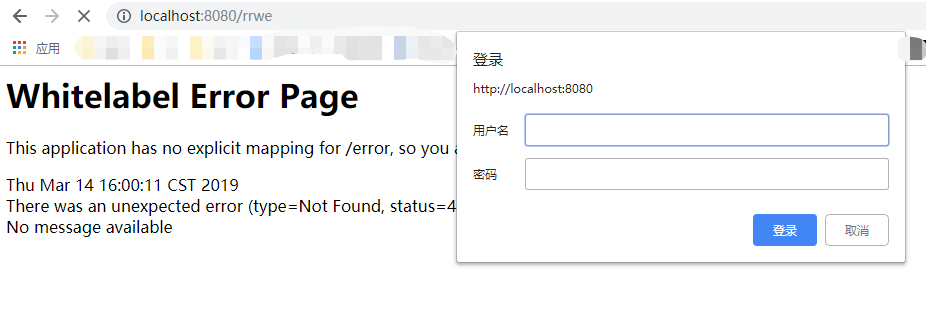
package com.zfh;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
*/\*\*  
 \** ***@author*** *zfh  
 \** ***@since*** *2019.3.14  
 \*/*@SpringBootApplication  
public class Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Application.class, args);  
 }  
}

##### 3,增加OAuth2ServerConfig

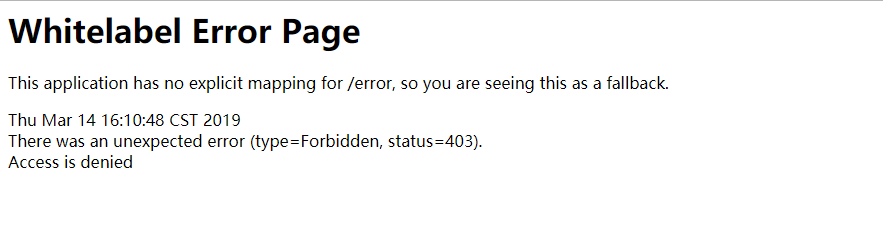
*/\*\*  
 \* 授权服务器  
 \** ***@author*** *zfh  
 \** ***@since*** *2019.3.14  
 \*/*@Configuration  
@EnableAuthorizationServer  
public class OAuth2ServerConfig extends AuthorizationServerConfigurerAdapter {  
  
 @Autowired  
 AuthenticationManager authenticationManager;  
  
 @Override  
 public void configure(ClientDetailsServiceConfigurer clients) throws Exception {  
 //super.configure(clients);  
 clients.inMemory()  
 .withClient("clientId")  
 .secret("secret") //client\_secret  
 .authorizedGrantTypes("authorization\_code","password") //授权类型  
 .scopes("\*")  
 // 加上下面代码，当为true时，OAuth Approval页面就默认放开，默认是false  
 .autoApprove("true"); //授权范围  
 }  
  
 @Override  
 public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  
 endpoints.authenticationManager(authenticationManager);  
 }  
  
 @Override  
 public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {  
 super.configure(security);  
 }  
  
  
 @Bean  
 protected UserDetailsService userDetailsService(){  
 InMemoryUserDetailsManager manager = new InMemoryUserDetailsManager();  
 manager.createUser(User.*withUsername*("user\_1").password("123456").authorities("ROLE\_USER").build());  
 manager.createUser(User.*withUsername*("user\_2").password("123456").authorities("ROLE\_USER").build());  
 return manager;  
 }  
  
  
}

#### 4,启动访问系统的任意url

现象：



这个是Spring security的最基础的认证方式，输入相应的用户名和名字，成功之后如果出现如下提示：



一般都是你登录的这个用户没有该权限去访问你刚才的url，你主要调整好增加的角色就行。调整完之后就可以正常访问你刚才的资源了

##### 5,使用auth2 code模式获取token

第一步  
url:  
localhost:8080/oauth/authorize?client\_id=clientId&response\_type=code&redirect\_uri=https://www.baidu.com  
method:GET  
结果：  
http://www.baidu.com?code=z3aNOf  
  
第二步  
  
url:  
http://localhost:8080/oauth/token?grant\_type=authorization\_code&code=3VotJO&client\_id=clientId&client\_secret=sercet&redirect\_uri=https://www.baidu.com  
method:POST  
head:clientId sercet  
结果：  
{  
 "access\_token": "6b82f6e0-b705-4136-ba6b-b2e5b72d66e3",  
 "token\_type": "bearer",  
 "expires\_in": 43199,  
 "scope": "read"  
}

##### 6,,使用密码模式获取

url:  
http://localhost:8080/oauth/token?grant\_type=password&username=user&password=67fc3e1d-b36b-4e59-ae42-089041b0e9fb  
method ： get  
  
header:  
Authorization : Basic clientId:securt 組合的base64碼

##### 7,自定义UserDeatailService

增加如下类来替换源码中默认的UserDetailService

*/\*\*  
 \* 自定义 UserDetailsService  
 \*  
 \** ***@author*** *zfh  
 \** ***@since*** *2019/2/5  
 \*/*@Configuration  
public class JdbcUserDetailService implements UserDetailsService {  
  
  
 @Override  
 public UserDetails loadUserByUsername(String userName){  
 //我这边偷懒没有用dao去查数据库  
 //UserDetails ud = super.loadUserByUsername(username);  
 List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();  
 authorities.add(new SimpleGrantedAuthority("ROLE\_ADMIN"));  
 //return new User("zfh12","$2a$10$k2/8rCh.1FGCoNt1LggQhePMPufVc.LolyZtqGSoGbPlDxLiNFKJq",authorities);  
 return new User("zfh","zfh",authorities);  
 }  
  
}

在Application增加如下代码

@Bean  
public JdbcUserDetailService jdbcUserDetailService(){  
 return new JdbcUserDetailService();  
}

将OAuth2ServerConfig类原先的代码

@Bean  
protected UserDetailsService userDetailsService(){  
 InMemoryUserDetailsManager manager = new InMemoryUserDetailsManager();  
 manager.createUser(User.*withUsername*("user\_1").password("123456").authorities("ROLE\_USER").build());  
 manager.createUser(User.*withUsername*("user\_2").password("123456").authorities("ROLE\_USER").build());  
 return manager;  
}

删除调。

然后修改代码

@Override  
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  
 endpoints.authenticationManager(authenticationManager);  
 }

变为

@Override  
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  
 endpoints.authenticationManager(authenticationManager);  
 endpoints.userDetailsService(jdbcUserDetailService);  
}

用zfh，zfh去登录成功就意味着自定义的类已经替换了

##### 8,自定义token增强器

增加如下类

*/\*\*  
 \* 自定义token 兼容原生token  
 \** ***@author*** *zfh  
 \** ***@since*** *2019/2/5  
 \*/*public class CustomTokenEnhancer implements TokenEnhancer {  
 @Override  
 public OAuth2AccessToken enhance(OAuth2AccessToken oAuth2AccessToken, OAuth2Authentication oAuth2Authentication) {  
 DefaultOAuth2AccessToken token = null;  
 if (oAuth2AccessToken instanceof DefaultOAuth2AccessToken) {  
 token = ((DefaultOAuth2AccessToken) oAuth2AccessToken);  
 //token.setValue(getNewToken()); //修改token的vakue值  
  
 Map<String, Object> additionalInformation = new HashMap<String, Object>();  
 additionalInformation.put("zfh12", "1234");  
 token.setAdditionalInformation(additionalInformation);  
 System.*out*.println(token.getValue());  
  
 String type = oAuth2AccessToken.getTokenType();  
 if ("authorization\_code".equals(type)) {  
 /\* //如果授权方式是code的方式,在兼容原先的基础上更改，增加一个过期时间  
 Map<String, Object> additionalInformation = new HashMap<String, Object>();  
 additionalInformation.put("refresh\_exp", token.getExpiration());  
 token.setAdditionalInformation(additionalInformation);\*/  
 }  
 }  
 return token;  
 }  
}

在Application增加如下代码

@Bean  
public TokenEnhancer tokenEnhancer() {  
 return new CustomTokenEnhancer();  
}

然后在OAuth2ServerConfig修改如下代码

@Override  
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  
 endpoints.authenticationManager(authenticationManager);  
 endpoints.userDetailsService(jdbcUserDetailService);  
 }

为

@Override  
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  
 endpoints.authenticationManager(authenticationManager);  
 endpoints.userDetailsService(jdbcUserDetailService);  
 endpoints.tokenEnhancer(tokenEnhancer);  
}

然后增加成员变量

@Autowired  
TokenEnhancer tokenEnhancer;

##### 9,JWT 实现

1：公钥

Pom文件中加jwt的jar

<dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-jwt</artifactId>  
</dependency>

先加一个Jwt的配置类

*/\*\*  
 \* JWT配置  
 \** ***@author*** *zfh  
 \** ***@since*** *2019/2/7  
 \*/*@Configuration  
public class JwtTokenConfig {  
  
 private final Logger logger = LoggerFactory.*getLogger*(JwtTokenConfig.class);  
  
  
 public JwtTokenConfig() {logger.info("Loading JwtTokenConfig ...");}  
  
 @Bean  
 public TokenStore jwtTokenStore() {  
 return new JwtTokenStore(jwtAccessTokenConverter());  
 }  
  
 // 用来生成jwt(JwtTokenStore需要这个类来解码转码)  
 @Bean  
 public JwtAccessTokenConverter jwtAccessTokenConverter() {  
 JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();  
  
 //生成签名的key  
 //秘签  
 jwtAccessTokenConverter.setSigningKey("zfh");  
 return jwtAccessTokenConverter;  
 }  
}

然后在OAuth2ServerConfig修改如下代码

public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  
 endpoints.authenticationManager(authenticationManager);  
 endpoints.userDetailsService(jdbcUserDetailService);  
 endpoints.tokenEnhancer(tokenEnhancer);   
}

修改为

@Override  
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  
 endpoints.authenticationManager(authenticationManager);  
 endpoints.userDetailsService(jdbcUserDetailService);  
 endpoints.tokenEnhancer(tokenEnhancer);  
  
 //设置tokenstore  
 endpoints.tokenStore(jwtTokenStore);  
 .accessTokenConverter(jwtAccessTokenConverter);  
  
   
}

**但是这个你去获取token的时候还是原来的token形式，也就是还是没有成功。**

**原因是什么呢？原来是JwtAccessTokenConverter类其实也是一个token增强器（TokenEnhancer）。因为我们之前已经有一个自定义的TokenEnhancer了，所以一直不行。但是spring security提供了一个TokenEnhancerChain，可以将两个TokenEnhancer串联起来。所以最终改成**

@Override  
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  
 endpoints.authenticationManager(authenticationManager);  
 endpoints.userDetailsService(jdbcUserDetailService);  
 // endpoints.tokenEnhancer(tokenEnhancer);  
  
 //设置tokenstore  
 endpoints.tokenStore(jwtTokenStore);  
 // .accessTokenConverter(jwtAccessTokenConverter);  
  
 TokenEnhancerChain tokenEnhancerChain = new TokenEnhancerChain();  
 tokenEnhancerChain.setTokenEnhancers(Arrays.*asList*(tokenEnhancer,jwtAccessTokenConverter));  
 endpoints.tokenEnhancer(tokenEnhancerChain);  
  
  
}

**访问公钥的链接：** /oauth/token\_key

2：私钥

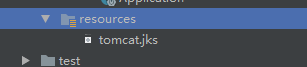
在OAuth2ServerConfig类中将以下代码

@Override  
public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {  
 super.configure(security);  
   
}

变成下面

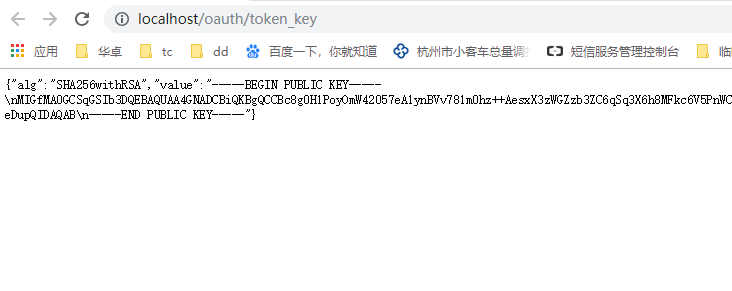
@Override  
public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {  
 super.configure(security);  
  
 security.tokenKeyAccess("permitAll()").checkTokenAccess("isAuthenticated()").allowFormAuthenticationForClients();  
}

使用jdk自带的keytool工具生成带有公钥和私钥的jks文件



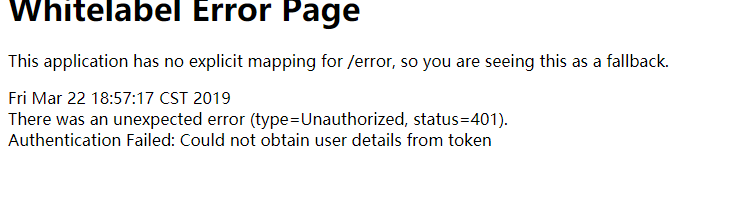
网址：<https://www.cnblogs.com/zhangzb/p/5200418.html>

获取到token的页面



然后直接在客户端的jwt.key-value的配置中写上上面获取的值

如果公钥不对那就会显示



##### 10,增加自定义UsernamePasswordAuthenticationFilter

先新增一个自定义的filter

*/\*\*  
 \** ***@auth*** *zfh  
 \** ***@since*** *2019/2/27  
 \*  
 \* 自定义UsernamePasswordAuthenticationFilter过滤器  
 \*  
 \* 自定义的需求：  
 \* 如果现在在登录页面除了username和password之类还有其它业务属性当成入参，并且要放到token里面  
 \* 我们这里以"mdid"为例子，这样最后就会在最终获取的Authentication对象中带着这个mdid  
 \*  
 \* 如果最后还需要往token加mdid，就直接在我们之前的CustomTokenEnhancer类中从Authentication中获取mdid放进去就可以了  
 \*/*public class CustomUsernamePasswordAuthenticationFilter extends UsernamePasswordAuthenticationFilter {  
 private boolean postOnly = true;  
 private String mdidParameter = "mdid";  
 public CustomUsernamePasswordAuthenticationFilter(){  
 super();  
 }  
  
 public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) throws AuthenticationException {  
 if (this.postOnly && !request.getMethod().equals("POST")) {  
 throw new AuthenticationServiceException("Authentication method not supported: " + request.getMethod());  
 } else {  
 String username = this.obtainUsername(request);  
 String password = this.obtainPassword(request);  
  
 String mdid = this.obtainMdid(request);  
 if (username == null) {  
 username = "";  
 }  
  
 if (password == null) {  
 password = "";  
 }  
  
 username = username.trim();  
 UsernamePasswordAuthenticationToken authRequest = new UsernamePasswordAuthenticationToken(username, password);  
  
 // UsernamePasswordAuthenticationToken类中还有一个detail属性，我们想要加的属性可以往这个里面加  
 HashMap<String,String> deatil = new HashMap<String, String>();  
 deatil.put("mdid",mdid);  
 authRequest.setDetails(deatil);  
  
 this.setDetails(request, authRequest);  
 return this.getAuthenticationManager().authenticate(authRequest);  
 }  
 }  
  
 protected String obtainMdid(HttpServletRequest request) {  
 return request.getParameter(this.mdidParameter);  
 }  
  
}

然后增加securityconfiguration类（其它相关配置自己自定义配置，我这边主要加filter）

在securityconfiguration类中加

@Override  
public void configure(HttpSecurity http) throws Exception {  
  
 //formLogin() 类型与spring security里xml的form-login元素，目的就是打开一个formlogin配置  
 //loginPage("登录页面的url") 自定义登录页url,默认为/login  
 // login-processing-url 登录请求拦截的url,也就是form表单提交时指定的action  
 //defaultSuccessUrl :成功登录过程后，用户将被重定向到页面 - 默认情况下，该页面是Web应用程序的根目录。  
 http.authorizeRequests().antMatchers("/\*\*").authenticated()  
 .and().formLogin();  
 /\* 增加自定义过滤器 \*/  
 CustomUsernamePasswordAuthenticationFilter filter = new CustomUsernamePasswordAuthenticationFilter();  
 filter.setAuthenticationManager(authenticationManager);  
 http.addFilter(filter);  
  
}