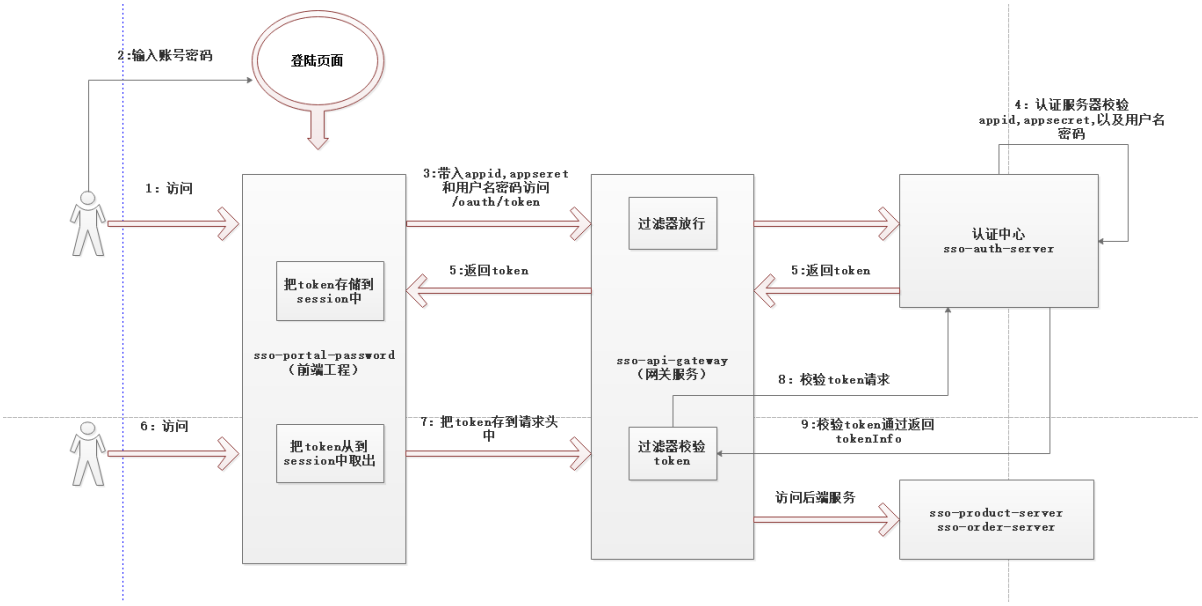
一:基于密码模式(Session)的单点登陆。

1.1）服务器规划

微服务	地址	端口	描述
sso-auth-server	auth.tuling.com	8888	认证中心
sso-api-gateway	gateway.tuling.com	8866	服务网关
sso-order-server	order.tuling.com	8899	订单微服务
sso-product-server	product.tuling.com	8877	商品微服务
sso-portal-password	portal.tuling.com	8855	前端工程

1.2)服务器架构图



1.3)服务器搭建

1.3.1)搭建认证服务器sso-auth-server

①:关键性的pom依赖

```
<dependency>
 <groupId>com.alibaba.cloud</groupId>
 <artifactId>spring-cloud-alibaba-nacos-discovery</artifactId>
</dependency>

<!--Oauth2的包-->
```

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

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

②认证服务器的配置

```
@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends AuthorizationServerConfigurerA
dapter {

 @Autowired
 private AuthenticationManager authenticationManager;

 @Autowired
 private DataSource dataSource;
 @Autowired
 private TulingUserDetailService userDetailsService;

 @Autowired
 private RedisConnectionFactory redisConnectionFactory;

 /**
 * 方法实现说明:服务器颁发的token的存储方式有四种存储方式
 * ①:基于内存的(生产环境几乎不用,因为认证服务器重启后token就消失了) 基于内存的不需
要配该组件
 * ②:基于Db存储的(生产上也机会不用，因为读取数据库的没有redis快)
 * ③:基于redis存储的(可用于生产,速度可以，但是基于redis存储的token 没有实际的业务
意义)
 * ④:基于jwt存储的(合适用于生产)
 * @author:smlz
 * @return:
 * @exception:
 * @date:2020/1/20 20:05
 */
 @Bean
 public TokenStore tokenStore(){
 return new RedisTokenStore(redisConnectionFactory);
```

```
30  //return new JdbcTokenStore(dataSource);
31  }
32
33
34
35  /**
36  * 方法实现说明:认证服务器基于配置可以给哪些第三方客户端 有二种存储方式
37  * ①:基于内存的
38  * ②:基于db的,那么需要创建数据库的表 oauth_client_details
39  * @author:smlz
40  * @return:
41  * @exception:
42  * @date:2020/1/20 20:09
43  */
44  @Override
45  public void configure(ClientDetailsServiceConfigurer clients) throws Except
ion {
46  clients.withClientDetails(clientDetails());
47  }
48
49  /**
50  * 方法实现说明:第三方客户端读取组件 专门用于读取oauth_client_details
51  * @author:smlz
52  * @return:
53  * @exception:
54  * @date:2020/1/20 20:11
55  */
56  @Bean
57  public ClientDetailsService clientDetails() {
58  return new JdbcClientDetailsService(dataSource);
59  }
60
61  /**
62  * 方法实现说明:认证服务器核心配置
63  * @author:smlz
64  * @return:
65  * @exception:
66  * @date:2020/1/20 20:13
67  */
68  @Override
69  public void configure(AuthorizationServerEndpointsConfigurer endpoints) thr
ows Exception {
70
```

```
71   endpoints.tokenStore(tokenStore())//token存储的方式
72   .authenticationManager(authenticationManager);
73   }
74
75
76   /**
77   * 方法实现说明:配置校验token需要带入clientId 和clientSeret配置
78   * @author:smlz
79   * @return:
80   * @exception:
81   * @date:2020/1/20 20:14
82   */
83   @Override
84   public void configure(AuthorizationServerSecurityConfigurer security) throw
s Exception {
85
86   security .checkTokenAccess("isAuthenticated()");
87   }
88  }
```

③:认证服务器安全配置

```
1  @Configuration
2  @EnableWebSecurity
3  public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
4   @Autowired
5   private TulingUserDetailService userDetailsService;
6
7   /**
8   * 方法实现说明:用于构建用户认证组件,需要传递userDetailsService和密码加密器
9   * @author:smlz
10   * @param auth
11   * @return:
12   * @exception:
13   * @date:2019/12/25 14:31
14   */
15   @Override
16   protected void configure(AuthenticationManagerBuilder auth) throws Exceptic
n {
17
auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
18   }
19
```

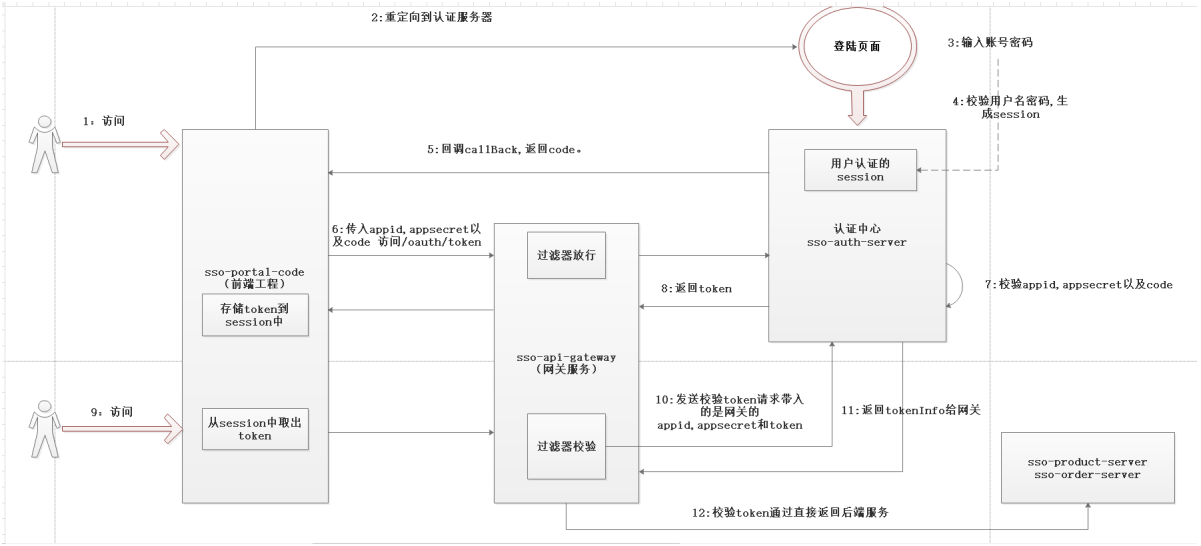
```
20  /**
21  * 方法实现说明:安全配置
22  * @author:smlz
23  * @return:
24  * @exception:
25  * @date:2020/1/20 20:34
26  */
27  protected void configure(HttpSecurity http) throws Exception {
28  http.formLogin()
29  .and()
30  .authorizeRequests()
31  .antMatchers("/user/getCurrentUser").permitAll()
32  .anyRequest()
33  .authenticated()
34  .and().csrf().disable().cors();
35  }
36
37  /**
38  * 方法实现说明:密码加密器
39  * @author:smlz
40  * @return:
41  * @exception:
42  * @date:2020/1/20 20:34
43  */
44  @Bean
45  public PasswordEncoder passwordEncoder() {
46  return new BCryptPasswordEncoder();
47  }
48
49  /**
50  * 方法实现说明:用户密码认证模式的组件
51  * @author:smlz
52  * @return:
53  * @exception:
54  * @date:2020/1/20 20:33
55  */
56  @Bean
57  public AuthenticationManager authenticationManagerBean() throws Exception {
58  return super.authenticationManagerBean();
59  }
60
61  }
```

二:基于授权码模式(Session)的单点登陆。

2.1)涉及工程

微服务	地址	端口	描述
sso-auth-code-server	www.auth.com	8888	认证中心
sso-api-gateway	www.gateway.com	8866	服务网关
sso-order-server	www.order.com	8899	订单微服务
sso-product-server	www.product.com	8877	商品微服务
sso-portal-code	www.portal.com	8855	前端工程

2.2)原理图:



2.3）认证服务器配置:sso-auth-code-server

2.3.1）认证服务器的配置(配置根据密码模式的认证服务器配置一样的)主要是安全配置有少许改动.

```
/**
 * @vlog: 高于生活，源于生活
 * @desc: 类的描述:授权中心安全配置
 * @author: smlz
 * @createDate: 2019/12/20 16:08
 * @version: 1.0
 */
```

```
@Configuration
@EnableWebSecurity
 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

  @Qualifier("userDetailsService")
  @Autowired
  private TulingUserDetailService userDetailsService;

  @Autowired
  private LogoutSuccessHandler logoutSuccessHandler;


  /**
  * 方法实现说明:用于构建用户认证组件,需要传递userDetailsService和密码加密器
  * @author:smlz
  * @param auth
  * @return:
  * @exception:
  * @date:2019/12/25 14:31
  */
  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exceptic
n {
auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
  }


  /**
  * 设置前台静态资源不拦截
  * @param web
  * @throws Exception
  */
  @Override
  public void configure(WebSecurity web) throws Exception {
  web.ignoring().antMatchers("/assets/**", "/css/**", "/images/**");
  }


  @Override
  protected void configure(HttpSecurity http) throws Exception {
  //自定义表单登陆页面
  http.formLogin()
```

```
  .loginPage("/login")
  .and()
  .logout()
  //认证服务器退出登陆回调的处理器
  .logoutSuccessHandler(logoutSuccessHandler)
  .and()
  .authorizeRequests()
  .antMatchers("/login","/login.html","/user/getCurrentUser").permitAll()
  .anyRequest()
  .authenticated()
  .and().csrf().disable().cors();
  }


  @Bean
  public PasswordEncoder passwordEncoder() {
  return new BCryptPasswordEncoder();
  }

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

}


/**
*用户回调 退出登陆后 用于跳回第三方应用的
*/
@Component
@Slf4j
public class TulingLogoutSuccessHandler implements LogoutSuccessHandler {
 @Override
  public void onLogoutSuccess(HttpServletRequest request, HttpServletResponse
response, Authentication authentication) throws IOException, ServletException {
  log.info("currentUrl:{}",request.getRequestURL());
  log.info("跳转页面:{}",request.getParameter("redirectUrl"));
  String redirectUrl = request.getParameter("redirectUrl");
  if(!StringUtils.isEmpty(redirectUrl)) {
  response.sendRedirect(redirectUrl);
  }
```

```
90    }
91 }
```

2.4）授权码模式 注意点:

2.4.1) 退出注意点: 我们在sso-portal-code中点击退出的同时 还需要去调用认证中心的退出接口。

说白了就是 失效sso-portal-code 工程中的session 也要失效sso-auth-code-server工程中的session。

2.4.2）三个有效期 作用说明

sso-portal-code 的session有效期.

该有效期是用来控制sso-portal-code多久来访问一次认证服务器的.

sso-auth-code-server Session的有效期 表示用户多久输入一次账号密码

token的有效期: 表示token 多长时间能拿着令牌来 访问服务的.若token有效期

过了怎么办？一般是通过Refresh来刷新我们的令牌.

refresh_token:有效期, 若refresh_token 实现了 表示我们要去认证服务器上

从新获取token。

3:JWT令牌 (Json token web) 一个有意义的字符串 用来做身份认证的

①:Jwt头部(签名算法)

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

②：有效载荷(pload)

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022,
  "sex":"man"
}
```

③:签名

RS256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret) 得出的字符串signature

JWT字符串的组成样例

base64UrlEncode(header).base64UrlEncode(payload).base64UrlEncode(signature)

接入jwt的认证服务器的改造sso-auth-code-jwt-server（就是基于jwt存储 与增强的代码）

```
1  @Configuration
2  @EnableAuthorizationServer
3  public class AuthServerInDbConfig extends AuthorizationServerConfigurerAdapte
r {
4
5
6   @Autowired
7   private AuthenticationManager authenticationManager;
8
9   @Autowired
```

```
10  private DataSource dataSource;
11
12  @Autowired
13  private TulingUserDetailService userDetailsService;
14
15
16  /**
17  * 方法实现说明: 使用jwt存储token,我们创建jwtTOkenStore的时候 需要一个组件
18  * jwtAccessTokenConverter 所以我们 可以通过@Bean的形式 创建一个该组件.
19  * @author:smlz
20  * @return:
21  * @exception:
22  * @date:2020/1/15 20:17
23  */
24  @Bean
25  public TokenStore tokenStore(){
26  return new JwtTokenStore(jwtAccessTokenConverter());
27  }
28
29
30  /**
31  * 这个组件 用于jwt basecode 字符串和 安全认证对象的信息转化
32  * @return
33  */
34  @Bean
35  public JwtAccessTokenConverter jwtAccessTokenConverter() {
36  JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
37  //jwt的密钥(用来保证jwt 字符串的安全性 jwt可以防止篡改 但是不能防窃听 所以jwt不
要 放敏感信息)
38  converter.setKeyPair(keyPair());
39  //converter.setSigningKey("123456");
40  return converter;
41  }
42
43  /**
44  * KeyPair是 非对称加密的公钥和私钥的保存者
45  * @return
46  */
47  @Bean
48  public KeyPair keyPair() {
49  KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(new ClassPat
hResource("jwt.jks"), "123456".toCharArray());
```

```
50  return keyStoreKeyFactory.getKeyPair("jwt", "123456".toCharArray());
51  }
52
53
54  /**
55  * 该组件就是用来给jwt令牌中添加额外信息的 来增强我们的jwt的令牌信息
56  * @return
57  */
58  @Bean
59  public TulingTokenEnhancer tulingTokenEnhancer() {
60  return new TulingTokenEnhancer();
61  }
62
63
64
65  /**
66  * 方法实现说明:认证服务器能够给哪些 客户端颁发token 我们需要把客户端的配置 存储到
67  * 数据库中 可以基于内存存储和db存储
68  * @author:smlz
69  * @return:
70  * @exception:
71  * @date:2020/1/15 20:18
72  */
73  @Override
74  public void configure(ClientDetailsServiceConfigurer clients) throws Except
ion {
75  clients.withClientDetails(clientDetails());
76  }
77
78  /**
79  * 方法实现说明:用于查找我们第三方客户端的组件 主要用于查找 数据库表 oauth_client
_details
80  * @author:smlz
81  * @return:
82  * @exception:
83  * @date:2020/1/15 20:19
84  */
85  @Bean
86  public ClientDetailsService clientDetails() {
87  return new JdbcClientDetailsService(dataSource);
88  }
89
90  /**
```

```
91   * 方法实现说明:授权服务器的配置的配置
92   * @author:smlz
93   * @return:
94   * @exception:
95   * @date:2020/1/15 20:21
96   */
97   @Override
98   public void configure(AuthorizationServerEndpointsConfigurer endpoints) thr
ows Exception {
99
100  /*
101  增加我们的令牌信息
102  */
103  TokenEnhancerChain tokenEnhancerChain = new TokenEnhancerChain();
104  tokenEnhancerChain.setTokenEnhancers(Arrays.asList(tulingTokenEnhancer(),j
wtAccessTokenConverter()));
105
106  endpoints.tokenStore(tokenStore()) //授权服务器颁发的token 怎么存储的
107  .tokenEnhancer(tokenEnhancerChain)
108  .userDetailsService(userDetailsService) //用户来获取token的时候需要 进行账号
密码
109  .authenticationManager(authenticationManager);
110  }
111
112
113  /**
114  * 方法实现说明:授权服务器安全配置
115  * @author:smlz
116  * @return:
117  * @exception:
118  * @date:2020/1/15 20:23
119  */
120  @Override
121  public void configure(AuthorizationServerSecurityConfigurer security) thro
ws Exception {
122  //第三方客户端校验token需要带入 clientId 和clientSecret来校验
123  security .checkTokenAccess("isAuthenticated()")
124  .tokenKeyAccess("isAuthenticated()");//来获取我们的tokenKey需要带入clientId,
clientSecret
125
126  security.allowFormAuthenticationForClients();
127  }
128
```

```
129 }
130
131
132
133 //////////////////////jwt增强器
134 public class TulingTokenEnhancer implements TokenEnhancer {
135  @Override
136  public OAuth2AccessToken enhance(OAuth2AccessToken accessToken, OAuth2Auth
entication authentication) {
137  TulingUser tulingUser = (TulingUser) authentication.getPrincipal();
138
139  final Map<String, Object> additionalInfo = new HashMap<>();
140  final Map<String, Object> retMap = new HashMap<>();
141
142  additionalInfo.put("email",tulingUser.getEmail());
143  additionalInfo.put("phone",tulingUser.getPhone());
144  additionalInfo.put("userId",tulingUser.getUserId());
145  additionalInfo.put("nickName",tulingUser.getNickName());
146
147  retMap.put("additionalInfo",additionalInfo);
148
149  ((DefaultOAuth2AccessToken) accessToken).setAdditionalInformation(retMap);
150
151  return accessToken;
152  }
153 }
```

怎么生成kepPair对象
keytool

-genkey

-alias tomcat(别名)

-keypass 123456(别名密码)

-keyalg RSA(生证书的算法名称，RSA是一种非对称加密算法)

-keysize 1024(密钥长度,证书大小)

-validity 365(证书有效期，天单位)

-keystore W:/tomcat.keystore(指定生成证书的位置和证书名称)

-storepass 123456(获取keystore信息的密码)

- storetype (指定密钥仓库类型)

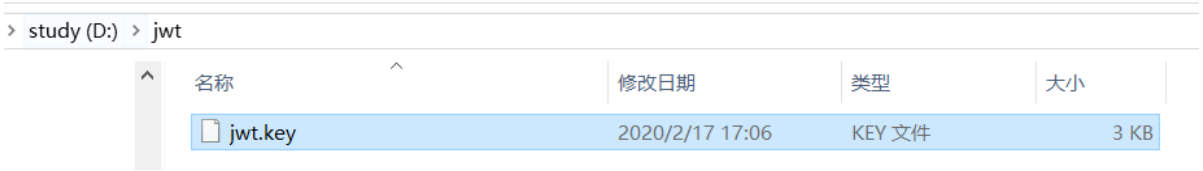
命令:

keytool -genkeypair -alias jwt -keyalg RSA -keysize 2048 -keystore D:/jwt/jwt.key

```
C:\Users\zhuwei>keytool  -genkeypair -alias jwt -keyalg RSA -keysize 2048 -keystore D:/jwt/jwt.key
输入密钥库口令:
再次输入新口令:
您的名字与姓氏是什么?
  [Unknown]:  smlz
您的组织单位名称是什么?
  [Unknown]:  tuling
您的组织名称是什么?
  [Unknown]:  tuling
您所在的城市或区域名称是什么?
  [Unknown]:  changsha
您所在的省/市/自治区名称是什么?
  [Unknown]:  hunan
该单位的双字母国家/地区代码是什么?
  [Unknown]:  china
CN=smlz, OU=tuling, O=tuling, L=changsha, ST=hunan, C=china是否正确?
  [否]:  y

输入 <jwt> 的密钥口令
        (如果和密钥库口令相同, 按回车):
再次输入新口令:

C:\Users\zhuwei>
```



网关改造:sso-api-gateway-jwt 因为网关启动的时候 需要去认证中心获取jwt解析的公钥。

1）网关启动的时候 需要通过远程调用 去获取jwt的公钥(**在这里Ribbon是不起作用的)**

在这里我们实现了InitializingBean 去获取jwt的公钥.

2)自己需要写一个负载均衡的组件 来实现负载均衡的功能.

```
1  /**
2   * 认证过滤器,根据url判断用户请求是要经过认证 才能访问
3   * Created by smlz on 2019/12/17.
4   */
5  @Component
6  @Slf4j
7  public class AuthorizationFilter implements GlobalFilter,Ordered,InitializingBean {
8
9   @Autowired
10   private RestTemplate restTemplate;
```

```
11
12  private PublicKey publicKey;
13
14  /**
15  * 请求各个微服务 不需要用户认证的URL
16  */
17  private static Set<String> shouldSkipUrl = new LinkedHashSet<>();
18
19
20  @Override
21  public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain cha
in) {
22
23  String reqPath = exchange.getRequest().getURI().getPath();
24  log.info("网关认证开始URL->:{}",reqPath);
25
26  //1:不需要认证的url
27  if(shouldSkip(reqPath)) {
28  log.info("无需认证的路径");
29  return chain.filter(exchange);
30  }
31
32  //获取请求头
33  String authHeader = exchange.getRequest().getHeaders().getFirst("Authorizat
ion");
34
35  //请求头为空
36  if(StringUtils.isEmpty(authHeader)) {
37  log.warn("需要认证的url,请求头为空");
38  throw new GateWayException(SystemErrorType.UNAUTHORIZED_HEADER_IS_EMPTY);
39  }
40
41  //交易我们的jwt 若jwt不对或者超时都会抛出异常
42  Claims claims = validateJwtToken(authHeader);
43
44  //向headers中放文件，记得build
45  ServerHttpRequest request = exchange.getRequest().mutate().header("usernam
e",claims.get("user_name").toString()).build();
46  //将现在的request 变成 change对象
47  ServerWebExchange serverWebExchange = exchange.mutate().request(request).bu
ild();
48
49  //从jwt中解析出权限集合进行判断
50  hasPremisson(claims,reqPath);
```

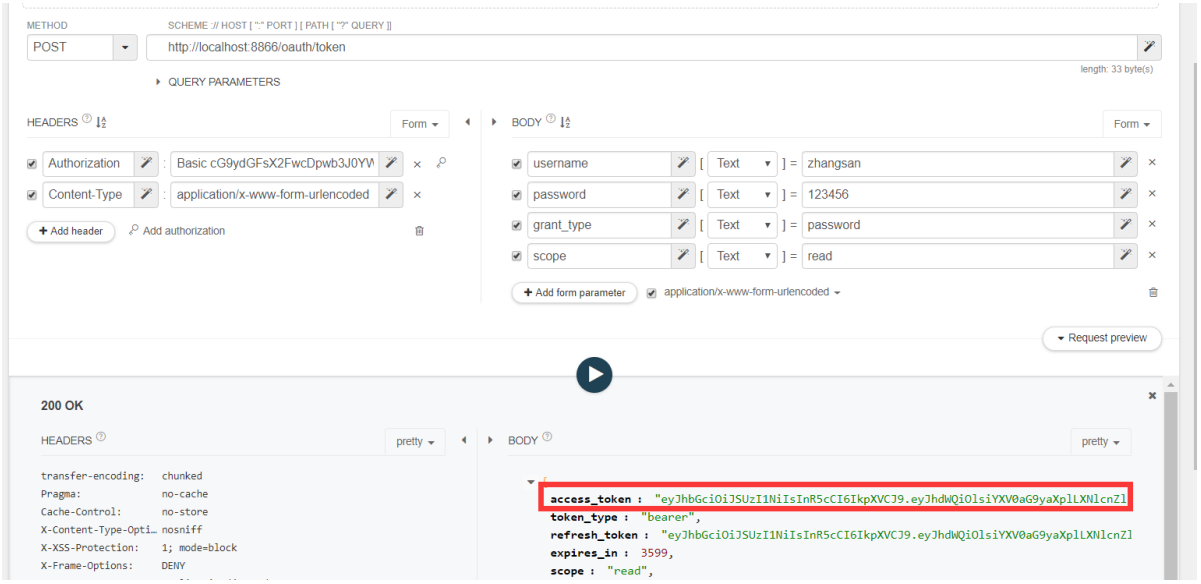
```
51
52  return chain.filter(serverWebExchange);
53
54  }
55
56  private Claims validateJwtToken(String authHeader) {
57  String token =null ;
58  try{
59  token = StringUtils.substringAfter(authHeader, "bearer ");
60
61  Jwt<JwsHeader, Claims> parseClaimsJwt = Jwts.parser().setSigningKey(publicK
ey).parseClaimsJws(token);
62
63  Claims claims = parseClaimsJwt.getBody();
64  log.info("claims:{}",claims);
65  return claims;
66  }catch(Exception e){
67  log.error("校验token异常:{},异常信息:{}",token,e.getMessage());
68  throw new GateWayException(SystemErrorType.INVALID_TOKEN);
69  }
70  }
71
72  private boolean hasPremisson(Claims claims,String currentUrl) {
73  boolean hasPremisson = false;
74  //登陆用户的权限集合判断
75  List<String> premessionList = claims.get("authorities",List.class);
76  for (String url: premessionList) {
77  if(currentUrl.contains(url)) {
78  hasPremisson = true;
79  break;
80  }
81  }
82  if(!hasPremisson){
83  log.warn("权限不足");
84  throw new GateWayException(SystemErrorType.FORBIDDEN);
85  }
86
87  return hasPremisson;
88  }
89
90
91
92
```

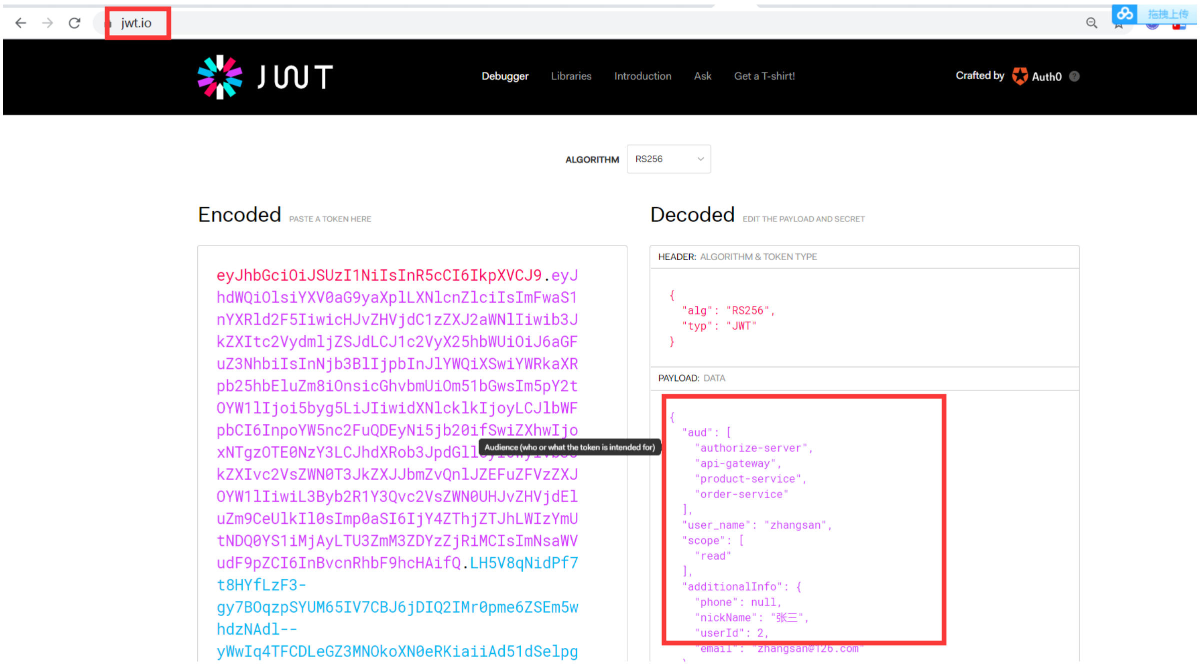
```
93  /**
94  * 方法实现说明:不需要授权的路径
95  * @author:smlz
96  * @param reqPath 当前请求路径
97  * @return:
98  * @exception:
99  * @date:2019/12/26 13:49
100  */
101  private boolean shouldSkip(String reqPath) {
102
103  for(String skipPath:shouldSkipUrl) {
104  if(reqPath.contains(skipPath)) {
105  return true;
106  }
107  }
108  return false;
109  }
110
111
112  @Override
113  public int getOrder() {
114  return 0;
115  }
116
117  @Override
118  public void afterPropertiesSet() throws Exception {
119  /**
120  *实际上，这边需要通过去数据库读取 不需要认证的URL,不需要认证的URL是各个微服务
121  * 开发模块的人员提供出来的. 我在这里没有去查询数据库了,直接模拟写死
122  */
123  shouldSkipUrl.add("/oauth/token");
124  shouldSkipUrl.add("/oauth/check_token");
125  shouldSkipUrl.add("/user/getCurrentUser");
126
127  //初始化公钥
128  this.publicKey = genPublicKeyByTokenKey();
129  }
130
131  /**
132  * 方法实现说明:去认证服务器上获取tokenKey
133  * @return:
134  * @exception:
```

```
135   * @date:2020/1/21 16:53
136   */
137   private String getTokenKey(){
138
139   HttpHeaders headers = new HttpHeaders();
140   headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
141   headers.setBasicAuth(MDA.clientId,MDA.clientSecret);
142
143   HttpEntity<MultiValueMap<String, String>> entity = new HttpEntity<>(null,
headers);
144
145   try {
146
147   ResponseEntity<Map> response = restTemplate.exchange(MDA.getTokenKey, Http
Method.GET, entity, Map.class);
148
149   String tokenKey = response.getBody().get("value").toString();
150
151   log.info("去认证服务器获取TokenKey:{}",tokenKey);
152
153   return tokenKey;
154   }catch (Exception e) {
155
156   log.error("远程调用认证服务器获取tokenKey失败:{}",e.getMessage());
157   throw new GateWayException(SystemErrorType.GET_TOKEN_KEY_ERROR);
158   }
159   }
160
161   private PublicKey genPublicKeyByTokenKey() {
162
163   try{
164   String tokenKey = getTokenKey();
165
166   String dealTokenKey =tokenKey.replaceAll("\\-*BEGIN PUBLIC KEY\\-*", "").r
eplaceAll("\\-*END PUBLIC KEY\\-*", "").trim();
167
168   java.security.Security.addProvider(new org.bouncycastle.jce.provider.Bounc
yCastleProvider());
169
170   X509EncodedKeySpec pubKeySpec = new
X509EncodedKeySpec(Base64.decodeBase64(dealTokenKey));
171
172   KeyFactory keyFactory = KeyFactory.getInstance("RSA");
173
```

```
174   PublicKey publicKey = keyFactory.generatePublic(pubKeySpec);
175
176   log.info("生成公钥:{}",publicKey);
177
178   return publicKey;
179   }catch (Exception e) {
180   log.info("生成公钥异常:{}",e.getMessage());
181   throw new GateWayException(SystemErrorType.GET_TOKEN_KEY_ERROR);
182   }
183
184   }
185
186
187 }
```

测试JWT




jwt.io
JWT
Debugger
Libraries
Introduction
Ask
Get a T-shirt!
Crafted by Auth0
ALGORITHM RS256
Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
{
"alg": "RS256",
"typ": "JWT"
}
PAYLOAD: DATA
{
"aud": [
"authorize-server",
"api-gateway",
"product-service",
"order-service"
],
"user_name": "zhangsan",
"scope": [
"read"
],
"additionalInfo": {
"phone": null,
"nickName": "张三",
"userId": 2,
Audience (who or what the token is intended for)