光星
2023, 砥砺前行, 赏花开
深 圳

文章 分类 标签
517 103 768

目录

5 年前发表 2 年前更新 架构设计 / 微服务设计 33 分钟读完 (大约4927个字

微服务之间调用的安全认证

微服务之间的相互调用，需要一套认证机制来确认调用是安全的。这不同于在 API 网关的统一认证，主要是防止在微服务暴露在外网的情况下，内部接口被外部恶意调用。

如果微服务是在内网，对外暴露的只有 API 网关，则可以不用做认证。本篇以 JWT 技术来实现安全认证。更多关于 JWT，可参考分布式应用系列(一)：详细理解 JWT（Json Web Token)。

微服务架构中，通常会将认证功能独立成一个微服务，即创建一个专门处理认证、授权、解析、核验的认证服务，可叫认证中心。

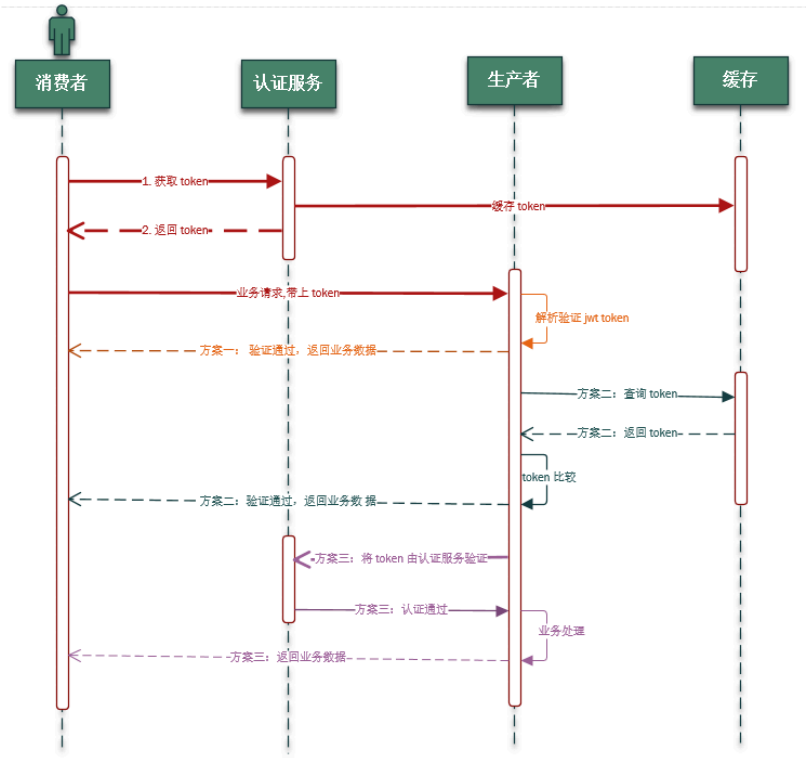
其实 API 调用安全认证与 OSS 单点登录认证，在总体流程是上是相似的的，消费者首先请求认证服务，认证服务创建签发令牌（Token）返回给客户端，消费者带着令牌发请求到生产者，接下来就是对令牌的验证，验证通过就返回到业务层。

令牌验证三种方案：

一、令牌是基于 JWT 创建的，此令牌支持自验证，可以直接在生产者端对 JWT 令牌进行解析验证。

二、认证服务在生成令牌时，存到缓存服务器，生产者从缓存取出消费者令牌，与消费者携带的令牌进行比较验证。

三、生产者拿到消费者的令牌，请求认证服务，由认证服务对签发的令牌进行验证，把验证结果返回生产者。



api-auth

认证服务

认证服务主要提供创建令牌、签发令牌、返回令牌给客户端、解析验证令牌。

创建认证服务

创建 Spring Boot Web 应用，添加 JWT 实现的 JAR 库（java-jwt 或 jjwt），这里以 **java-jwt** 库为例。

- **java-jwt**：此库是 JWT 的标准实现；
- **jjwt**：此库扩展了压缩功能，即生成的 token 是已压缩后的，非标准的，无法用标准的 JWT 实现来解析它，如果生成和解析都用此库则没有问题，若生成的 token 需要在不同开发语言的系统中解析，则不能使用，无法确保兼容。

1. 微服务信息表

   数据库创建一张表，维护微服务信息表，表字段根据实际需要进行扩充。

```
1  CREATE TABLE `app_info` (
2    `id` int(11) unsigned NOT NULL AUTO_INCR
3    `app_id` varchar(100) NOT NULL,
4    `secret_key` varchar(100) NOT NULL,
5    `app_name` varchar(50) DEFAULT NULL,
```

```
6     `app_desc` varchar(250) DEFAULT NULL,
7     PRIMARY KEY (`id`)
8   ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 CO
```

核心字段 **app_id** 和 **secret_key**，是查询条件。

2. 添加依赖

java-jwt 依赖是必须添加的，如果微服务架构的注册中心是 Eureka，可以添加 eureka-client 依赖并配置注册到注册中心，其它依赖如 fastjson、commons-lang、hutool-all 按需添加。

```
1   <dependency>
2       <groupId>com.auth0</groupId>
3       <artifactId>java-jwt</artifactId>
4       <version>3.8.1</version>
5   </dependency>
```

编写认证 API

主要两个接口，一个是生成 JWT Token 的 API，另一个是验证 Token 的 API。

生成和验证 API 都需要用到加密算法，建议抽出 JWT 工具类 和 加密算法工具类，便于复用。

1. 生成和验证 JWT Token API

AuthController.class

```
1   /**
2   * @name: tokenAuth
3   * @desc: 认证API
4   * @author: gxing
5   * @date: 2019-05-27 14:02
6   **/
7   @RestController
8   @RequestMapping("/auth")
9   public class AuthController {
10
11    private Logger logger = LogManager.getL
12
13    @Autowired
14    private AppInfoService appInfoService;
15
```

```
16      /**
17       * 签发 Token
18       *
19       * @param authQuery 认证参数
20       * @param response  响应
21       * @return ResultBean
22       */
23      @RequestMapping("/token")
24      public ResultBean getToken(AuthQuery au
25          logger.info("authQuery:{}", JSON.to
26
27          if (StringUtils.isBlank(authQuery.g
28              return new ResultBean().fialByN
29          }
30
31          //根据appId 和 secretKey 到数据库查询
32          AppInfo appInfo = appInfoService.qu
33          if (appInfo == null) {
34              return new ResultBean().fialByN
35          }
36
37          String jwtId =Long.toString(System.
38          //第二个参数是过期时间，单位:分钟,详见
39          String token = JavaJwtUtil.getToken
40
41          JwtToken jwtToken = new JwtToken(jw
42
43          return new ResultBean().success().s
44      }
45
46      /**
47       * 验证 Token
48       *
49       * @param request
50       * @return ResultBean
51       */
52      @RequestMapping("/verify")
53      public ResultBean verifyToken(HttpServl
54
55          String token = request.getHeader("A
56          String jwtId = request.getHeader("j
57          boolean verify = JavaJwtUtil.verify
58          if (!verify) {
59              return new ResultBean().fial().
60          }
61          return new ResultBean().success();
62      }
63  }
```

2. 相关实体类

- **AuthQuery**：实体类，查询数据库的条件，包含 **appId** 和 **secretKey** 两个属性。
- **AppInfo**：实体类，微服务应用信息，属性与数据库表 **app_info** 中的字段对应。
- **JwtToken**：实体类，封装生成 JWT Token 的必要信息，示例中包含基本地 **jwtId** 和 **token** 两个属性。
- **ResultBean**：实体类，封装响应结果，包含 **code、state、msg、data** 属性。

封装 JWT 工具类

抽出生成和验证 JWT Token 功能到工具类，主要方法有：

- **生成 Token**：token不要有敏感信息，通常包含用户ID，jwtId等信息。
- **验证 Token**：检查是否合法，可以指定声明验证。
- **刷新 RSA 公钥和私钥**：刷新密钥对是为了防止泄漏、公钥和私钥通常是写死的，也可以做成配置的。集成配置管理中心后，可以对公钥和私钥进行动态修改，修改后重新初始化公钥、私钥对象。

```
/**
 * @name: JavaJwtUtil
 * @desc: java_jwt 库工具类,创建签发验证token
 **/
public class JavaJwtUtil {

    private static RSAPrivateKey rsaPrivateK
    private static RSAPublicKey rsaPublicKey


    /**
     * HMAC256 算法签发Token
     *
     * @param jwtId    用户id
     * @param secret 密钥
     * @return String
     */
    public static String getTokenByHMAC256(S
        /*默认一天有效期*/
        Long endDateTime = System.currentTim

        String token = JWT.create()
```

```
23                  .withClaim("jwtId", jwtId)
24                  .withExpiresAt(new Date(endD
25                  .sign(Algorithm.HMAC256(secr
26          return token;
27      }
28
29      /**
30       * HMAC256 算法签发Token
31       *
32       * @param jwtId    用户id
33       * @param exp    过期时间,单位:分钟
34       * @param secret 密钥
35       * @return String
36       */
37      public static String getTokenByHMAC256(S
38          Long endDateTime = System.currentTim
39
40          String token = JWT.create()
41                  .withClaim("jwtId", jwtId)
42                  .withExpiresAt(new Date(endD
43                  .sign(Algorithm.HMAC256(secr
44          return token;
45      }
46
47      /**
48       * RSA512 算法签发Token
49       *
50       * @param jwtId 用户ID
51       * @return String
52       */
53      public static String getTokenByRSA512(St
54          Long endDateTime = System.currentTim
55
56          Algorithm algorithm = Algorithm.RSA5
57          String token = JWT.create()
58                  .withClaim("jwtId", jwtId)
59                  .withExpiresAt(new Date(endD
60                  .sign(algorithm);
61          return token;
62      }
63
64      /**
65       * RSA512 算法签发Token
66       *
67       * @param jwtId 用户id
68       * @param exp 有效期
69       * @return String
70       */
71      public static String getTokenByRSA512(St
```

```
72
73          Long endDateTime = System.currentTim
74
75          Algorithm algorithm = Algorithm.RSA5
76          String token = JWT.create()
77                  .withClaim("jwtId", jwtId)
78                  .withExpiresAt(new Date(endD
79                  .sign(algorithm);
80          return token;
81      }
82
83      /**
84       * 验证 HMAC256 Token
85       *
86       * @param token  令牌
87       * @param jwtId    用户id
88       * @param secret 密钥
89       * @return boolean
90       */
91      public static boolean verifyTokenByHMAC2
92
93          JWTVerifier jwtVerifier = JWT.requir
94                  .withClaim("jwtId", jwtId).b
95          return verifyToken(token, jwtVerifie
96      }
97
98      /**
99       * 验证 RSA512 Token
100      *
101      * @param token 令牌
102      * @param jwtId   用户ID
103      * @return boolean
104      */
105     public static boolean verifyTokenByRSA51
106
107         JWTVerifier jwtVerifier = JWT.requir
108                 .withClaim("jwtId", jwtId).b
109         return verifyToken(token, jwtVerifie
110     }
111
112     /**
113      * 验证 Token
114      *
115      * @param token 令牌
116      * @param jwtVerifier JWTVerifier对象
117      * @return boolean
118      */
119     private static boolean verifyToken(Strin
120         try {
```

```
121                 jwtVerifier.verify(token);
122                 return true;
123             } catch (JWTVerificationException e)
124                 e.printStackTrace();
125                 return false;
126             }
127         }
128 
129         /*
130         public static void main(String[] args) {
131 
132             String token1 = JavaJwtUtil.getToken
133             System.out.println(token1);
134 
135             String token2 = JavaJwtUtil.getToken
136             System.out.println(token2);
137 
138             boolean check = JavaJwtUtil.verifyTo
139             System.out.println(check);
140 
141             String tokenByRSA = JavaJwtUtil.getT
142             String tokenByRSA = JavaJwtUtil.getT
143 
144             System.out.println(tokenByRSA);
145 
146             boolean check = JavaJwtUtil.verifyTo
147             System.out.println(check);
148 
149         }
150         */
151     }
```

封装 RSA 加密工具类

下面工具类使用模数的指数来生成 RSA 密钥时，必须重新设置 **MODULUS** 、**PRIVATE_EXPONENT** 和 **PUBLIC_EXPONENT** 属性的值，可取消 **main** 方法的注释并运行，将打印输出的值复制到这三个对应的属性。

```
1  /**
2   * @name: RSAUtil
3   * @desc: RSA 加解密工具类
4   * @author: gxing
5   * @date: 2019-05-27 13:57
6   **/
7  public class RSAUtil {
```

```
8
9       public static String RSA_ALGORITHM = "RS
10
11      /*模数*/
12      public static String MODULUS = "";
13      /*公钥指数*/
14      public static String PUBLIC_EXPONENT = "
15      /*私钥指数*/
16      public static String PRIVATE_EXPONENT =
17
18      /**
19       * 公钥加密
20       * @param data
21       * @return
22       * @throws Exception
23       */
24      public static String encryptByPublicKey(
25          RSAPublicKey publicKey = RSAUtil.get
26          Cipher cipher = Cipher.getInstance(R
27          cipher.init(Cipher.ENCRYPT_MODE, pub
28          // 模长
29          int key_len = publicKey.getModulus()
30          // 加密数据长度 <= 模长-11
31          String[] datas = splitString(data, k
32          String mi = "";
33          // 如果明文长度大于模长-11则要分组加密
34          for (String s : datas) {
35              mi += bcd2Str(cipher.doFinal(s.g
36          }
37          return mi;
38      }
39
40      /**
41       * 私钥解密
42       * @param data
43       * @return
44       * @throws Exception
45       */
46      public static String decryptByPrivateKey
47          RSAPrivateKey privateKey = RSAUtil.g
48          Cipher cipher = Cipher.getInstance(R
49          cipher.init(Cipher.DECRYPT_MODE, pri
50          // 模长
51          int key_len = privateKey.getModulus(
52          byte[] bytes = data.getBytes();
53          byte[] bcd = ASCII_To_BCD(bytes, byt
54          // 如果密文长度大于模长则要分组解密
55          String ming = "";
56          byte[][] arrays = splitArray(bcd, ke
```

```
57          for (byte[] arr : arrays) {
58              ming += new String(cipher.doFina
59          }
60          return ming;
61      }
62
63
64      /**
65       * 生成公钥和私钥
66       * @throws NoSuchAlgorithmException
67       */
68      public static HashMap<String, Object> ge
69          HashMap<String, Object> map = new Ha
70          KeyPairGenerator keyPairGen = KeyPai
71          keyPairGen.initialize(1024);
72          KeyPair keyPair = keyPairGen.generat
73          RSAPublicKey publicKey = (RSAPublicK
74          RSAPrivateKey privateKey = (RSAPriva
75          map.put("public", publicKey);
76          map.put("private", privateKey);
77          return map;
78      }
79
80      /**
81       * 使用模和指数生成RSA公钥
82       * 注意：【此代码用了默认补位方式，为RSA/No
83       * /None/NoPadding】
84       * @param modulus  模
85       * @param exponent 指数
86       * @return
87       */
88      public static RSAPublicKey getPublicKey(
89          try {
90              BigInteger b1 = new BigInteger(m
91              BigInteger b2 = new BigInteger(e
92              KeyFactory keyFactory = KeyFacto
93              RSAPublicKeySpec keySpec = new R
94              return (RSAPublicKey) keyFactory
95          } catch (Exception e) {
96              e.printStackTrace();
97              return null;
98          }
99      }
100
101     /**
102      * 使用模和指数生成RSA私钥
103      * 注意：【此代码用了默认补位方式，为RSA/No
104      * /None/NoPadding】
105      *
```

```
106         * @param modulus  模
107         * @param exponent 指数
108         * @return
109         */
110        public static RSAPrivateKey getPrivateKe
111            try {
112                BigInteger b1 = new BigInteger(m
113                BigInteger b2 = new BigInteger(e
114                KeyFactory keyFactory = KeyFacto
115                RSAPrivateKeySpec keySpec = new
116                return (RSAPrivateKey) keyFactor
117            } catch (Exception e) {
118                e.printStackTrace();
119                return null;
120            }
121        }
122
123        /**
124         * 公钥加密
125         *
126         * @param data
127         * @param publicKey
128         * @return
129         * @throws Exception
130         */
131        public static String encryptByPublicKey(
132            Cipher cipher = Cipher.getInstance(R
133            cipher.init(Cipher.ENCRYPT_MODE, pub
134            // 模长
135            int key_len = publicKey.getModulus()
136            // 加密数据长度 <= 模长-11
137            String[] datas = splitString(data, k
138            String mi = "";
139            // 如果明文长度大于模长-11则要分组加密
140            for (String s : datas) {
141                mi += bcd2Str(cipher.doFinal(s.g
142            }
143            return mi;
144        }
145
146        /**
147         * 私钥解密
148         *
149         * @param data
150         * @param privateKey
151         * @return
152         * @throws Exception
153         */
154        public static String decryptByPrivateKey
```

```
155             Cipher cipher = Cipher.getInstance(R
156             cipher.init(Cipher.DECRYPT_MODE, pri
157             // 模长
158             int key_len = privateKey.getModulus(
159             byte[] bytes = data.getBytes();
160             byte[] bcd = ASCII_To_BCD(bytes, byt
161             // 如果密文长度大于模长则要分组解密
162             String ming = "";
163             byte[][] arrays = splitArray(bcd, ke
164             for (byte[] arr : arrays) {
165                 ming += new String(cipher.doFina
166             }
167             return ming;
168         }
169 
170         /**
171          * ASCII码转BCD码
172          */
173         public static byte[] ASCII_To_BCD(byte[]
174             byte[] bcd = new byte[asc_len / 2];
175             int j = 0;
176             for (int i = 0; i < (asc_len + 1) /
177                 bcd[i] = asc_to_bcd(ascii[j++]);
178                 bcd[i] = (byte) (((j >= asc_len)
179             }
180             return bcd;
181         }
182 
183         public static byte asc_to_bcd(byte asc)
184             byte bcd;
185 
186             if ((asc >= '0') && (asc <= '9'))
187                 bcd = (byte) (asc - '0');
188             else if ((asc >= 'A') && (asc <= 'F'
189                 bcd = (byte) (asc - 'A' + 10);
190             else if ((asc >= 'a') && (asc <= 'f'
191                 bcd = (byte) (asc - 'a' + 10);
192             else
193                 bcd = (byte) (asc - 48);
194             return bcd;
195         }
196 
197         /**
198          * BCD转字符串
199          */
200         public static String bcd2Str(byte[] byte
201             char temp[] = new char[bytes.length
202 
203             for (int i = 0; i < bytes.length; i+
```

```
204                 val = (char) (((bytes[i] & 0xf0)
205                 temp[i * 2] = (char) (val > 9 ?
206 
207                 val = (char) (bytes[i] & 0x0f);
208                 temp[i * 2 + 1] = (char) (val >
209             }
210             return new String(temp);
211         }
212 
213         /**
214          * 拆分字符串
215          */
216         public static String[] splitString(Strin
217             int x = string.length() / len;
218             int y = string.length() % len;
219             int z = 0;
220             if (y != 0) {
221                 z = 1;
222             }
223             String[] strings = new String[x + z]
224             String str = "";
225             for (int i = 0; i < x + z; i++) {
226                 if (i == x + z - 1 && y != 0) {
227                     str = string.substring(i * l
228                 } else {
229                     str = string.substring(i * l
230                 }
231                 strings[i] = str;
232             }
233             return strings;
234         }
235 
236         /**
237          * 拆分数组
238          */
239         public static byte[][] splitArray(byte[]
240             int x = data.length / len;
241             int y = data.length % len;
242             int z = 0;
243             if (y != 0) {
244                 z = 1;
245             }
246             byte[][] arrays = new byte[x + z][];
247             byte[] arr;
248             for (int i = 0; i < x + z; i++) {
249                 arr = new byte[len];
250                 if (i == x + z - 1 && y != 0) {
251                     System.arraycopy(data, i * l
252                 } else {
```

```
253                 System.arraycopy(data, i * l
254             }
255             arrays[i] = arr;
256         }
257         return arrays;
258     }
259 
260     public static void main(String[] args) t
261                 /*HashMap<String, Object> ma
262         //生成公钥和私钥
263         RSAPublicKey publicKey = (RSAPublick
264         RSAPrivateKey privateKey = (RSAPriva
265 
266         //模
267         String MODULUS = publicKey.getModulu
268         System.err.println("MODULUS:" + MODU
269         //公钥指数
270         String PUBLIC_EXPONENT = publicKey.g
271         System.err.println("PUBLIC_EXPONENT:
272         //私钥指数
273         String PRIVATE_EXPONENT = privateKey
274         System.err.println("PRIVATE_EXPONENT
275 
276 
277         //明文
278         String ming = "Hello World";
279         //使用模和指数生成公钥和私钥
280         RSAPublicKey pubKey = RSAUtil.getPub
281         RSAPrivateKey priKey = RSAUtil.getPr
282 
283         //加密后的密文
284         String mi = RSAUtil.encryptByPublick
285         System.err.println("加密后密文: "+mi)
286         //解密后的明文
287         ming = RSAUtil.decryptByPrivateKey(m
288         System.err.println("解密后明文" + min
289 
290         String encStr = encryptByPublicKey("
291         System.out.println(encStr);
292         System.out.println(decryptByPrivateK
293         */
294     }
295 }
```

消费者服务

消费者服务在请求生产者服务前必须先请求 **认证服务** 拿到到用于认证的 **Token**，然后每向生产者服务发请求，必须在

请求头 中携带此 **Token**，通常设置该请求头名为：**Authorization**。

每次向生产者服务请求前都获取 Token 是不合适的，并且 Token 是有有效期的，第一次获取后，在有效期内可继续使用，所以在拿到 Token 后可以存起来，例如存到环境变量，或存到外部缓存系统 **Redis** 中，如果 Token 过期则重新获取。

获取 Token 两种方式，一种是在应用启动时就向认证服务请求获取 Token，但不支持动态更新；另一种是使用定时器，动态获取，定时器时间必须小于 Token 的过期时间，建议使用此方式。

应用访问认证服务必须提供 **appId** 和 **secretKey** 两种参数，用于从数据库查询该应用的合法性。可以定义实体类注入配置文件中的属性值，或从环境变量(Environment 或 System)中取出。

```
1  @Component
2  @ConfigurationProperties(prefix = "common.prop
3  public class AuthQuery {
4
5      private String appId;
6      private String secretKey;
7      //-----省略 set/get 方法------
8  }
```

定时器获取 Token

如果 Token 是采用动态改变策略，可以使用定时任务的方式，定期请求认证服务获取 Token 并动态更新的环境变量，定时任务的间隔时间必须小于 Token 的有效时长。

使用定时任务，需要在 Spring Boot 启动类上添加 **@EnableScheduling** 注解开启定时任务，再编写定时任务的业务，示例如下。

```
1  /**
2   * @name: TokenScheduledTask
3   * @desc: 定时任务动态更新 Token
4   **/
5  @Component
6  public class TokenScheduledTask {
```

```
7
8       @Autowired
9       private AuthQuery authQuery;
10      private static Logger logger = LogManager
11
12      //20小时,token默认有效期是24小时
13      private final static long DELAY = 1000 *
14
15      @Autowired(required = false)
16      private AuthService authService;
17
18      @Scheduled(fixedDelay = DELAY)
19      public void reloadAuthToken() {
20          JwtToken jwtToken = this.getToken();
21          while (null == jwtToken) {
22              try {
23                  Thread.sleep(1000);
24                  jwtToken = getToken();
25              } catch (InterruptedException e)
26                  logger.info("thread sleep err
27                  e.printStackTrace();
28              }
29          }
30          System.setProperty("jwtId", jwtToken.
31          System.setProperty("token", jwtToken.
32
33      }
34
35      private JwtToken getToken() {
36          ResultBean result = authService.getTo
37          LinkedHashMap<String, String> resultD
38          if (null == resultDate) {
39              return null;
40          }
41          return new JwtToken(resultDate.get("j
42      }
43  }
```

应用启动获取 Token

如果验证的 Token 不是动态改变的，可以在应用启动时就请求获取到 Token。

编写初始化 Token 配置类，实现 CommandLineRunner 接口，重写 run 方法。可以使用 RestTemplate 发送请求，如果认证服务、消费者服务都注册到了 Eureka Server(注册中心)，也可以通过 Feign Client 来发送请求。启动初始化示例如下：

```java
/**
 * @name: InitTokenConfig
 * @desc: 应用启动时初始化 Token
 **/
@Component
public class InitTokenConfig implements Comma

    @Autowired
    private RestTemplate restTemplateOne;
    @Autowired
    private AuthQuery authQuery;

    @Override
    public void run(String... args) throws Ex

        String url = "http://localhost:9060/a

        MultiValueMap<String, Object> paramMa
        paramMap.add("appId", this.authQuery.
        paramMap.add("secretKey", this.authQu
        ResultBean resultBean = restTemplateO
        LinkedHashMap<String,String> linkedHa

        //设置到系统环境
        System.setProperty("jwtId", linkedHas
        System.setProperty("token", linkedHas

    }
}
```

缓存 Token

请求获取认证的 Token 也可以缓存到 Redis 中，这样虽然减少了请求认证的次数，但会产生网络延时，所以建议存到服务环境变量中。

请求拦截器设置请求头

HTTP 远程调用通常会用到 HttpClient 或 RestTemplate，Spring Cloud 还可以使用 Feign，在调用前每次手动设置请求头则非常麻烦，而这三种 HTTP 客户端都支持添加拦截器来统一处理请求。

Feign 拦截器设置请求头

在 Spring Cloud 中通常会用 Feign 来调用接口，Feign 提供了请求拦截器 *feign.RequestInterceptor* 来支持对请求进行统一处理。

1. Feign 请求拦截器实现 RequestInterceptor 接口

```
/**
 * @name: FeignBasicAuthRequestIntercepto
 * @desc: Feign 请求拦截器
 **/
public class FeignAuthRequestInterceptor
    @Override
    public void apply(RequestTemplate req
        requestTemplate.header("JwtId", S
        requestTemplate.header("Authoriza
    }
}
```

2. 将 FeignAuthRequestInterceptor 注册为 Bean

```
/**
 * @name: FeignCustomConfig
 * @desc: TODO
 **/
@Component
public class FeignCustomConfig {

    @Bean
    public FeignAuthRequestInterceptor fe
        return new FeignAuthRequestInterc
    }
}
```

3. 如果有多个 Feign 配置类，可通过 **@FeignClient** 注解时的 **configuration** 属性指定该配置类。

RestTemplate 拦截器设置请求头

如果使用 RestTemplate 发送请求，可以给 RestTemplate 添加拦截器来统一处理请求，需要实现 ClientHttpRequestInterceptor 接口。示例如下：

1. RestTemplate 请求拦截器实现 ClientHttpRequestInterceptor 接口

```
/**
 * @name: RestTemplateInterceptor
 * @desc: RestTemplate 请求拦截器
 **/
public class RestTemplateRequestIntercept
    @Override
    public ClientHttpResponse intercept(H

        HttpHeaders headers = request.get
        headers.add("JwtId", System.getPr
        headers.add("Authorization", Syst

        return execution.execute(request,
    }
}
```

2. 创建 RestTemplate 实例时添加请求拦截器

```
/**
 * @name: RestTemplateConfig
 * @desc: RestTemplate配置类
 **/
@Configuration
public class RestTemplateConfig {

    public RestTemplate restTemplate() {
        //设置超时时间,毫秒
        return new RestTemplateBuilder()
                .setConnectTimeout(Durati
                .setReadTimeout(Duration.
                .interceptors(new RestTem
                .build();
    }
}
```

HttpClient 拦截器设置请求头

Apache Http Client 包(org.apache.http)下提供了 HttpRequestInterceptor 拦截器，可用于统一处理请求。

1. HttpClient 请求拦截器实现 HttpRequestInterceptor 接口

```
/**
 * @name: HttpClientRequestInterceptor
 * @desc: HttpClient 请求拦截器
 **/
public class HttpClientRequestInterceptor
    @Override
    public void process(HttpRequest reque
        request.setHeader("JwtId", System
        request.setHeader("Authorization"
    }
}
```

2. 创建自定义的 httpClient 实例并添加请求拦截器

```
/**
 * @name: HttpClientConfig
 * @desc: HttpClient 自定义配置
 **/
@Component
public class HttpClientConfig {

    @Bean
    public CloseableHttpClient closeableH
        CloseableHttpClient httpclient =
                .addInterceptorLast(new H
                .build();
        return httpclient;
    }
}
```

生产者服务

生产者服务需要对消费接口请求进行身份认证，从请求头中取出 声明 和 Token，使用 JWT 进行验证。

可以使用 **过滤器** 或 **拦截器** 来对请求的身份进行认证，以下是过滤器实现示例：

1. 创建过滤器实现身份认证

```
/**
 * @name: HttpTokenAuthFilter
 * @desc: 请求身份认证(Token)
 **/
```

```
public class HttpTokenAuthFilter implemen

    @Override
    public void doFilter(ServletRequest s

        HttpServletRequest request = (Htt
        HttpServletResponse response = (H

        response.setCharacterEncoding("UT
        response.setContentType(MediaType

        String token = request.getHeader(
        String jwtId = request.getHeader(

        if (StringUtils.isBlank(token) ||
            PrintWriter printWriter = res
            Map<String, String> resultMap
            resultMap.put("state", "fail"
            resultMap.put("code", "400");
            resultMap.put("msg", "认证失败

            String resultStr = JSON.toJSO
            printWriter.write(resultStr);
        } else {
            filterChain.doFilter(request,
        }

    }
}
```

2. 注册过滤器为 Bean 来启用

```
/**
 * @name: FilterConfig
 * @desc: 过滤器配置
 **/
@Configuration
public class FilterConfig {

    @Bean
    public FilterRegistrationBean filterR
        FilterRegistrationBean registrati
        registrationBean.setFilter(new Ht

        List<String> urlPatterns = new Ar
        //针对所有请求
        urlPatterns.add("/*");
```

```
16          registrationBean.setUrlPatterns(u
17          return registrationBean;
18      }
19  }
```

网关统一身份认证

如内部微服务必须经过网关才能访问，则可以在网关统一执行身份认证。例如，Zuul 网关，可创建一个前置过滤器（pre filter），在过滤器执行统一认证，捕抓并抛出异常，阻断路由到下游服务。关于 Zuul 过滤器，可参考 Spring Cloud系列(九)：API网关 Zuul 其它详细设置。

1. 创建 Token 认证前置过滤器

```
1   /**
2    * @name: TokenAuthPreFilter
3    * @desc: 统一身份认证
4    **/
5   public class TokenAuthPreFilter extends Z
6       @Override
7       public String filterType() {
8           return FilterConstants.PRE_TYPE;
9       }
10
11      @Override
12      public int filterOrder() {
13          return 5;
14      }
15
16      @Override
17      public boolean shouldFilter() {
18
19          return true;
20      }
21
22      @Override
23      public Object run() throws ZuulExcept
24          RequestContext context = RequestC
25          HttpServletRequest request = cont
26          String jwtId = request.getHeader(
27          String authorization = request.ge
28          System.out.println("JwtId : " + j
29          System.out.println("Authorization
30          try {
31              JavaJwtUtil.verifyTokenByRSA5
```

```
            } catch (Exception e) {
                //必须抛出或打印出错误,才不会路由
//                throw e;
                Throwable throwable = context
                throwable.printStackTrace();
            }
            return null;
        }
    }
```

2. 将认证过滤器注册为 Bean

```
/**
 * @name: ZuulConfig
 * @desc: Zuul 网关配置
 **/

@Configuration
public class ZuulConfig {

    @Bean
    public TokenAuthPreFilter tokenAuthPr
        return new TokenAuthPreFilter();
    }
}
```

其它参考

1. 微服务架构之访问安全
2. 并发登录人数控制
3. SpringBoot 并发登录人数控制

JWT AUTH

喜欢这篇文章？打赏一下作者吧

支付宝 微信

非对称加密RSA工具类-RSAUtil.class 理解 Thread 线程类的方法

评论

昵称 邮箱 网址(http://)

说点什么吧

提交

来发评论吧~

Powered By Valine
v1.4.14


Rocky