# Spring Authorization Server (10) 授权服务的JWK密钥对生成和JWT信息扩展

爱吃西瓜的胖娃 2023-10-31 797 阅读6分钟 关注

> 架构版本
>
> Spring Boot 3.1
>
> Spring Authorization Server 1.1.1
>
> spring-cloud 2022.0.3
>
> spring-cloud-alibaba 2022.0.0.0
>
> 完整代码👉 watermelon-cloud

## JWK 密钥生成

### 为什么JWK密钥对要固定生成?

目前代码中每次重启授权服务都会重新生成密钥对，Spring Authorization Server 默认的token加密就是 jwt，所以要做这个事情，如果不用jwt生成的token，自定义去扩展，要实现很多的东西，涉及到到服务也非常多，后面再单独讲讲不用jwt，自己扩展一个玩玩。

java 复制代码

```
@Bean
public JWKSource<SecurityContext> jwkSource() throws NoSuchAlgorithmException, IOExcept
    RSAKey rsaKey = JwtKeyUtil.generateRsa();
    JWKSet jwkSet = new JWKSet(rsaKey);
    return (jwkSelector, securityContext) -> jwkSelector.select(jwkSet);
}
```

> 有这样一种场景：当我们客户端、资源服务都没有重启的情况下，授权服务器重启了，密钥对重新生成了，这个时候客户端或资源服务器到授权服务器访问 `/oauth2/jwks` ，token还是之前的token，那么密钥对都改变了，那肯定是解密失败的，这个时候就需要重新授权(生成token才行了)。

### 如何解决?

让它不变不就行了?

### 能想到有以下方案

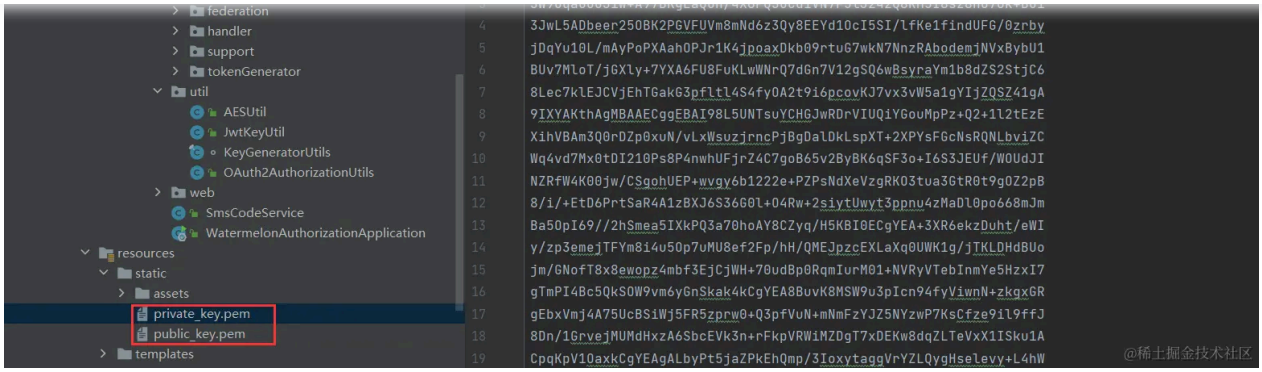
> 1.将 `JwtKeyUtil.generateRsa()` 生成的 `RSAKey` 持久化(redis)
> 2.生成密钥对文件，然后读取。

### 选择方案2

原因是jwk密钥对这样的存在就不是需要变动的，直接选择文件，简单又好维护。

### 生成密钥对文件

> win上打开小黑窗口
> 生成私钥： `openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048`
> 一定是2048，少于2048 Spring Authorization Server 是不允许的会抛异常。
> 提取公钥： `openssl rsa -pubout -in private_key.pem -out public_key.pem`




你的小黑窗口在哪个目录下打开的，就会生成在哪个目录下，我选择的实在resource的static目录下生成的。

## 读取密钥对

`private_key.pem` 里面长这样，我们只要中间的数据，将它转换为 `PrivateKey` 就可以了

text 复制代码

```
-----BEGIN PRIVATE KEY-----
MIIEwAIBADANBgkqhkiG9w0BAQEFAASCBKowggSmAgEAAoIBAQDr2aZI9TFmeS9F
3W9OqaGoG3iw+A97BKgEaQOn/4X6PQ3UCd1VN7F5lJ242Q8RM5I8sz8Ho70k+BUi
3JwL5ADbeer25OBK2PGVFUVm8mNd6z3Qy8EEYd1OcI5SI/lfKe1findUFG/0zrby
jDqYu10L/mAyPoPXAahOPJr1K4jpoaxDkb09rtuG7wkN7NnzRAbodemjNVxBybU1
BUv7MloT/jGXly+7YXA6FU8FuKLwWNrQ7dGn7V12gSQ6wBsyraYm1b8dZS2StjC6
8Lec7klEJCVjEhTGakG3pfltl4S4fyOA2t9i6pcovKJ7vx3vW5a1gYIjZQSZ41gA
9IXYAKthAgMBAAECggEBAI98L5UNTsuYCHGJwRDrVIUQiYGouMpPz+Q2+1l2tEzE
XihVBAm3Q0rDZp0xuN/vLxWsuzjrncPjBgDalDkLspXT+2XPYsFGcNsRQNLbviZC
Wq4vd7Mx0tDI210Ps8P4nwhUFjrZ4C7goB65v2ByBK6qSF3o+I6S3JEUf/WOUdJI
NZRfW4K00jw/CSgohUEP+wvgy6b1222e+PZPsNdXeVzgRKO3tua3GtR0t9gOZ2pB
8/i/+EtD6PrtSaR4A1zBXJ6S36G0l+O4Rw+2siytUwyt3ppnu4zMaDl0po668mJm
Ba5OpI69//2hSmea5IXkPQ3a70hoAY8CZyq/H5KBI0ECgYEA+3XR6ekzDuht/eWI
y/zp3emejTFYm8i4u5Op7uMU8ef2Fp/hH/QMEJpzcEXLaXq0UWK1g/jTKLDHdBUo
jm/GNofT8x8ewopz4mbf3EjCjWH+70udBp0RqmIurM01+NVRyVTebInmYe5HzxI7
gTmPI4Bc5QkSOW9vm6yGnSkak4kCgYEA8BuvK8MSW9u3pIcn94fyViwnN+zkgxGR
gEbxVmj4A75UcBSiWj5FR5zprw0+Q3pfVuN+mNmFzYJZ5NYzwP7KsCfze9il9ffJ
8Dn/1GrvejMUMdHxzA6SbcEVk3n+rFkpVRWiMZDgT7xDEKw8dqZLTeVxX1ISku1A
CpqKpV1OaxkCgYEAgALbyPt5jaZPkEhQmp/3IoxytaggVrYZLQygHselevy+L4hW
n+CqX61xBP/S7LCVqTTZ+QQr4vQTpYm76r8GJe6BvKvkCd9X3TLH1amIuVbg5EsW
9i3xt05iOoABcNqP1zGIRbLyAHrAPa8nccKulsEbCVHT4D9VjueGY+1v5RkCgYEA
rgeSzohELToyf9jKehoZ5qV4A4v7EJjSOgSxdaz9XlE8iEQcbIZH1qD/qzZRE72F
jsezAXxgA9Vf7IHo3xCNvmImk3QyzfW8cxbGu6KKUqrlDzsZI4rITS6uwcahdS/m
ylm0xnI4cvKENXhxFppvaFVN+AXXmpDFYyoiJbtcVDkCgYEArSjHW98MmSMXdEz0
```

```
-----END PRIVATE KEY-----
```

我也不知道怎么转，直接问chatgpt吧，或者科大讯飞的讯飞星火也可以，工具类方法这事就找他们代劳了。

java 复制代码

```java
public static PrivateKey getPrivateKey() throws IOException, NoSuchAlgorithmException, >I
        ClassPathResource resource = new ClassPathResource("static/private_key.pem");
        byte[] privateKeyBytes = FileCopyUtils.copyToByteArray(resource.getInputStream());
        String key = new String(privateKeyBytes, StandardCharsets.UTF_8);
        String privateStr = key
                .replace("-----BEGIN PRIVATE KEY-----", "")
                .replaceAll(System.lineSeparator(), "")
                .replace("-----END PRIVATE KEY-----", "");

        byte[] privateKeyDecodedBytes = Base64.getDecoder().decode(privateStr);

        PKCS8EncodedKeySpec keySpec = new PKCS8EncodedKeySpec(privateKeyDecodedBytes);
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        PrivateKey privateKey = keyFactory.generatePrivate(keySpec);

        return privateKey;
    }
```

public_key.pem 也同样找gpt代工。

java 复制代码

```java
public static RSAPublicKey getPublicKey() throws IOException, NoSuchAlgorithmException, I
        ClassPathResource resource = new ClassPathResource("static/public_key.pem");
        byte[] publicKeyBytes = FileCopyUtils.copyToByteArray(resource.getInputStream());
        String key = new String(publicKeyBytes, StandardCharsets.UTF_8);
        String publicStr = key
                .replace("-----BEGIN PUBLIC KEY-----", "")
                .replaceAll(System.lineSeparator(), "")
                .replace("-----END PUBLIC KEY-----", "");

        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        byte[] publicKeyBase64Bytes = Base64.getDecoder().decode(publicStr);
        X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(publicKeyBase64Bytes);
        PublicKey rsaPublicKey = keyFactory.generatePublic(publicKeySpec);
```

最后参考已有的封装一个 生成 `RSAKey` 的工具方法，这样也就完成了

java 复制代码

```java
public static RSAKey generateRsa() throws IOException, NoSuchAlgorithmException, InvalidK
        // @formatter:off
        return new RSAKey.Builder(getPublicKey())
                .privateKey(getPrivateKey())
                .build();
        // @formatter:on
    }
```

**替换工具类方法就完事了**😁

## `JWKSource<SecurityContext>` 最后是用在哪里的呢?

这个我们就要从token生成这里找入口了，因为token的加密是和它有关系的。

SmsAuthenticationProvider 种这样一段代码

java 复制代码

```java
/ ----- Access token -----
    OAuth2TokenContext tokenContext = tokenContextBuilder.tokenType(OAuth2TokenType.ACCE
    OAuth2Token generatedAccessToken = this.tokenGenerator.generate(tokenContext);
```

`tokenGenerator.generate` 就是生成token的，我们跟进去能找到对应实现就是 `JwtGenerator`

java 复制代码

```java
public final class JwtGenerator implements OAuth2TokenGenerator<Jwt> {
  private final JwtEncoder jwtEncoder;
  private OAuth2TokenCustomizer<JwtEncodingContext> jwtCustomizer;

  @Nullable
  @Override
```

```
10
11                  JwtEncodingContext jwtContext = jwtContextBuilder.build();
12                  this.jwtCustomizer.customize(jwtContext);
13          }
14
15          JwsHeader jwsHeader = jwsHeaderBuilder.build();
16          JwtClaimsSet claims = claimsBuilder.build();
17
18          Jwt jwt = this.jwtEncoder.encode(JwtEncoderParameters.from(jwsHeader, claims));
19
20          return jwt;
21  }
22
23
24 }
```

jwtEncoder 的实现继续往下 只有一个默认实现 NimbusJwtEncoder

java 复制代码

```java
1  public final class NimbusJwtEncoder implements JwtEncoder {
2
3      private final JWKSource<SecurityContext> jwkSource;
4      /**
5       * Constructs a {@code NimbusJwtEncoder} using the provided parameters.
6       * @param jwkSource the {@code com.nimbusds.jose.jwk.source.JWKSource}
7       */
8      public NimbusJwtEncoder(JWKSource<SecurityContext> jwkSource) {
9              Assert.notNull(jwkSource, "jwkSource cannot be null");
10             this.jwkSource = jwkSource;
11     }
12
13     @Override
14     public Jwt encode(JwtEncoderParameters parameters) throws JwtEncodingException {
15             Assert.notNull(parameters, "parameters cannot be null");
16
17             JwsHeader headers = parameters.getJwsHeader();
18             if (headers == null) {
19                     headers = DEFAULT_JWS_HEADER;
20             }
21             JwtClaimsSet claims = parameters.getClaims();
22
23             JWK jwk = selectJwk(headers);
24             headers = addKeyIdentifierHeadersIfNecessary(headers, jwk);
25
26             String jws = serialize(headers, claims, jwk);
```

```
30
31    private JWK selectJwk(JwsHeader headers) {
32            List<JWK> jwks;
33            try {
34                    JWKSelector jwkSelector = new JWKSelector(createJwkMatcher(headers));
35                    jwks = this.jwkSource.get(jwkSelector, null);
36            }
37            catch (Exception ex) {
38                    throw new JwtEncodingException(String.format(ENCODING_ERROR_MESSAGE_TEM
39                                    "Failed to select a JWK signing key -> " + ex.getMessag
40            }
41
42            if (jwks.size() > 1) {
43                    throw new JwtEncodingException(String.format(ENCODING_ERROR_MESSAGE_TEM
44                                    "Found multiple JWK signing keys for algorithm '" + hea
45            }
46
47            if (jwks.isEmpty()) {
48                    throw new JwtEncodingException(
49                                    String.format(ENCODING_ERROR_MESSAGE_TEMPLATE, "Failed
50            }
51
52            return jwks.get(0);
53    }
54 }
```

`NimbusJwtEncoder` 就用到了 `JWKSource<SecurityContext>` ，jwt生成token也是在这里完成的，后面如果想扩展，可以参考jwt的token生成流程。

## JWT信息扩展

我们先来看看，默认的jwt中有哪些信息

json 复制代码

```
1  {
2     "access_token": "eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiIxODY4MjY3ODk5NSIsImF1ZCI6Im1lc3NhZ2lu
3     "refresh_token": "3k17g-FjXQTGODkbIFQCkDzaRl7FENky4N0Nrz_Z53vuOcoN-Vet38FFas3ydpTqaTSKI
4     "token_type": "Bearer",
```

用jwt解密工具将 access_token 解密出来看看


**这点信息是不是有点少，如果我们想扩展呢，怎么玩呢?**

我们默认的token生成，能找到是 `JwtGenerator` 生成的， 然后直接上代码吧

java 复制代码

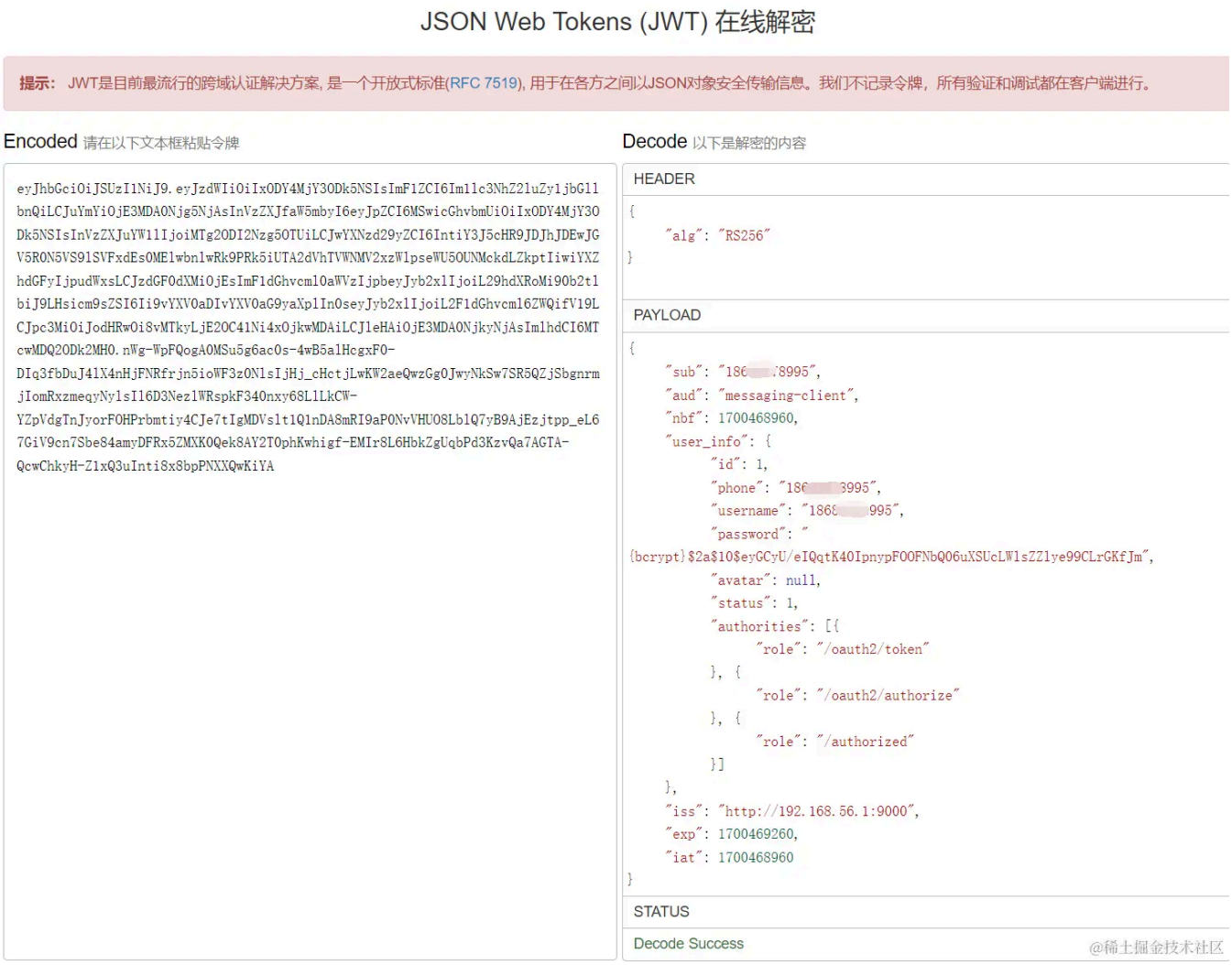
```java
public class ExtOAuth2TokenJwtCustomizer implements OAuth2TokenCustomizer<JwtEncodingCont


    @Override
    public void customize(JwtEncodingContext context) {
        JwtClaimsSet.Builder claims = context.getClaims();
        String tokenType = context.getTokenType().getValue();
        Object principal = context.getPrincipal().getPrincipal();
        if (principal instanceof DefaultOAuth2User defaultOAuth2User) {
            //todo 看后续是否需要根据第三方平台类型的map中属性不同再进行分别的一个转换
            claims.claim(AuthorizationServerConfigurationConsent.USER_INFO_PARAMETER,
                    defaultOAuth2User.getAttributes());
            return;
        }
        // 这个 Object principal 对应的是 provider 中 DefaultOAuth2TokenContext.Builder.prir
        claims.claim(AuthorizationServerConfigurationConsent.USER_INFO_PARAMETER, principa
```

然后注入到容器里面即可

java 复制代码

```java
//jwt 信息扩展
    @Bean
    public OAuth2TokenCustomizer<JwtEncodingContext> idTokenCustomizer() {
        return new ExtOAuth2TokenJwtCustomizer();
    }
```

最后来看看



## 本文收录于以下专栏

Spring Authorization Server 精讲 专栏目录

本专栏将深入讲解Spring Authorization Server在实践中的扩展点，希...

56 订阅 · 11 篇文章

订阅

上一篇 Spring Authorization Server (9) 授权...

## 评论 0

登录 / 注册 即可发布评论!

暂无评论数据

目录 收起

## 相关推荐

Spring Authorization Server (9) 授权服务的授权信息存储方式扩展
878阅读 · 3点赞

Spring Authorization Server Password授权扩展
386阅读 · 1点赞

Spring Authorization Server (5) 授权服务器【用户、客户端信息】扩展
497阅读 · 1点赞

Spring Authorization Server的使用
7.5k阅读 · 21点赞

Spring Authorization Server 定制AccessToken
387阅读 · 1点赞

## 精选内容

自从我读透了Spring的事务传播性，用户每次都兑奖成功了！！
掉头发的王富贵 · 240阅读 · 4点赞

深入理解微服务中的负载均衡算法与配置策略
努力的小雨 · 96阅读 · 6点赞

Spring Cloud全解析：注册中心之consul注册中心
isfox · 8阅读 · 0点赞

Spring Event 的幕后故事
不爱总结的麦穗 · 100阅读 · 4点赞

从单节点到集群：使用Redis解决负载均衡后WebSocket在线聊天室通信难题
翼飞 · 83阅读 · 1点赞



## 为你推荐

**Spring Authorization Server (4) 客户端、资源服务、授权服务 源码加流程细讲 再也...**

Spring Authorization Server (8) so easy 集成第三方【gitee、github】oauth2登录

爱吃西瓜的胖娃 11月前 1.1k 3 3 Spring ... Spring ...

## Spring Authorization Server (9) 授权服务的授权信息存储方式扩展

爱吃西瓜的胖娃 9月前 878 3 3 Spring ... Spring ...

## 在 React Router 中使用 JWT

D哥AI前端 1年前 310k 179 36 前端 JavaScr... React.js

## Spring Authorization Server 授权服务器

程序员Mark 1年前 6.5k 25 9 Spring ...

## 如何用 CSS 中写出超级美丽的阴影效果

非优秀程序员 2年前 1.2m 1.3k 85 前端 JavaScript

## 傻傻分不清之 Cookie、Session、Token、JWT

秋天不落叶 4年前 133k 3.4k 192 JavaScript

## Spring Authorization Server入门 (二) Spring Boot整合Spring Authorization Server

叹雪飞花 1年前 9.4k 35 107 Java

## 【1 月最新】前端 100 问：能搞懂 80% 的请把简历给我

程序员依扬 5年前 701k 11k 410 面试 前端

## 30 道 Vue 面试题，内含详细讲解（涵盖入门到精通，自测 Vue 掌握程度）

随风而逝_风逝 4年前 696k 8.4k 299 面试 Vue.js

## 中高级前端大厂面试秘籍，为你保驾护航金三银四，直通大厂(上)

郭东东 5年前 695k 7.2k 495 面试

## 一份不可多得的 TS 学习指南（1.8W字）

阿宝哥 3年前 218k 5.2k 235 前端 TypeSc...

## 「2021」高频前端面试题汇总之JavaScript篇（上）

CUGGZ 3年前 614k 3.4k 132 面试 前端

## Spring Authorization Server的使用

huan1993 3年前 7.5k 21 6 Spring 后端