

JWT

🌱 JWT是什么？

🌱 JWT中的Header和payload

🌱 JWT中的Signature

🌱 HS256、RS256、HMAC256分别是什么？HS/RS/HMAC是什么的缩写？256是什么意思？他们之家有哪...

💣 HS512和HmacSHA512有什么差别？

💣 HMAC-SHA中的SHA是什么加密算法？

💣 HMAC和HMAC-SHA有什么区别？

💣 JWT默认header中alg属性加密算法使用的是是什么？可以切换吗？

🌱 JWT的两个第三方实现框架

💣 com.auth0:java-jwt和io.jsonwebtoken:jjwt-api对比

🌱 JJWT的生成与解析 (🚀)

🌱 Java-JWT的生成与解析 (🚀)

时间：2023年5月27日13:48:17

🌱 JWT是什么？

JWT (JSON Web Token) 是一种轻量级的身份验证和授权机制，它使用 JSON 格式来定义一种安全的、自包含的、可扩展的方式，用于在各方之间传递信息。JWT 通常用于 RESTful API 的身份验证和授权，也可以用于其他场景，如单点登录、分布式系统等。

JWT 由三部分组成：头部 (Header)、载荷 (Payload) 和签名 (Signature)。它们分别对应了 JWT 的三个组成部分：

- **头部 (Header)**：包含了 JWT 的类型 (typ) 和签名算法 (alg) 等信息，通常采用 Base64 编码。
- **载荷 (Payload)**：包含了 JWT 所携带的信息，可以自定义一些字段，如用户 ID、权限等，通常也采用 Base64 编码。
- **签名 (Signature)**：使用头部和载荷中的信息以及一个密钥生成的签名，用于验证 JWT 的真实性和完整性。

JWT 的工作流程通常如下：

1. 用户使用用户名和密码进行身份验证。

2. 服务器验证用户身份，并生成一个 JWT。
3. 服务器将 JWT 发送给客户端。
4. 客户端将 JWT 存储在本地，并在每次请求时将其发送到服务器。
5. 服务器使用密钥验证 JWT 的真实性和完整性，并根据 JWT 中携带的信息进行授权。

JWT 的优点包括：

- 轻量级：JWT 使用 JSON 格式，数据量小，传输效率高。
- 自包含：JWT 包含了所有必要的信息，无需再查询数据库或其他存储设备。
- 可扩展：JWT 可以自定义字段，满足不同场景的需求。
- 安全性高：JWT 使用签名进行验证，保证了其真实性和完整性。

需要注意的是，由于 JWT 中携带了用户的敏感信息，如用户 ID、权限等，因此在使用 JWT 时需要注意安全性问题，如密钥管理、过期时间设置、防止重放攻击等。

🌱 JWT 中的 Header 和 payload

JWT 的头部 (Header) 和载荷 (Payload) 都是使用 JSON 格式编码的字符串，它们分别对应了 JWT 的前两个组成部分。

JWT 头部 (Header) 通常包含了以下字段：

- `alg`：指定了使用的签名算法，如 HS256、RS256 等。
- `typ`：指定了 JWT 的类型，通常为 JWT。

例如，下面是一个 JWT 的头部示例：

▼ Plain Text | 复制代码

```
1  {
2    "alg": "HS256",
3    "typ": "JWT"
4  }
```

JWT 载荷 (Payload) 通常包含了以下字段：

- `iss`：表示 JWT 的签发者。
 - Issuer (`iss`) 字段表示 JWT 的颁发者。它通常是一个标识 JWT 的应用程序或服务。该字段通常用于验证 JWT 的可信度和完整性。
- `sub`：表示 JWT 所面向的用户。
 - Subject (`sub`) 字段表示 JWT 所代表的实体或用户。它通常是用户的唯一标识符，例如用户的 ID 或电

子邮件地址。该字段通常用于验证用户是否有权访问受保护的资源。

- `aud` : 表示接收 JWT 的一方。
 - Audience (aud) 字段表示JWT的受众。它通常是一个或多个接收JWT的应用程序或服务。该字段通常用于限制JWT的使用范围，以确保只有特定的应用程序或服务可以使用该JWT。
- `exp` : 表示 JWT 的过期时间。
- `nbf` : 表示 JWT 的生效时间。
- `iat` : 表示 JWT 的签发时间。
- `jti` : 表示 JWT 的唯一标识符。
- 自定义字段: 可以根据需要自定义一些字段，如用户 ID、权限等。

例如，下面是一个 JWT 的载荷示例：

▼ Plain Text 复制代码

```
1  {
2    "iss": "example.com",
3    "sub": "123456",
4    "aud": "api.example.com",
5    "exp": 1622188800,
6    "nbf": 1622102400,
7    "iat": 1622102400,
8    "jti": "a1b2c3d4",
9    "username": "user",
10   "roles": ["admin", "user"]
11 }
```

在这个示例中：

- JWT 的签发者为 `example.com`
- 面向的用户为 `123456`
- 接收 JWT 的一方为 `api.example.com`
- JWT 的过期时间为 2021 年 5 月 28 日 00:00:00
- 生效时间为 2021 年 5 月 27 日 00:00:00
- 签发时间为 2021 年 5 月 27 日 00:00:00
- JWT 的唯一标识符为 `a1b2c3d4`
- 自定义的用户名和角色信息。

需要注意的是，JWT 中的头部和载荷都是使用 Base64 编码的字符串，可以通过在线工具或者编程语言自带的库进行解码。但是，为了确保 JWT 的安全性，不应该在载荷中存储敏感信息，如密码等。

🌱 JWT中的Signature

JWT 的签名 (Signature) 是使用头部和载荷中的信息以及一个密钥生成的字符串, 用于验证 JWT 的真实性和完整性。签名通常使用头部中指定的算法进行计算, 如 HS256、RS256 等。

签名的计算过程如下:

1. 😊 将头部和载荷分别进行 Base64 编码, 并用点号 (.) 连接起来, 形成一个字符串。
2. 😊 使用指定的算法和密钥对这个字符串进行签名。
3. 😊 将签名结果也进行 Base64 编码, 并用点号 (.) 连接起来, 形成 JWT 的第三部分。

例如, 假设使用 HS256 算法和密钥 "secret" 对下面的 JWT 进行签名:

| ▼ | Plain Text | 📄 复制代码 |
|---|---|--------|
| 1 | eyJhbGciOiAiSFMyNTYiLCJkaWwIjogIkpXVCJ9.eyJzdWIiOiAiMTIzNDU2IiwgInVzZXJuYW1lIjogInVzZXIiLCJmcm9sZXMiOiBbImFkbWluIiwgInVzZXIiXX0.9Rg8kWkQ1QJ5P2R6fzHm907dQ3QqKZcUxv9zX3YgKQ0 | |

其中, 第一部分是头部, 第二部分是载荷, 第三部分是签名。

对于 **这个 JWT**, 可以按照 **以下步骤进行验证**:

1. 😊 将头部和载荷分别进行 Base64 编码, 并用点号 (.) 连接起来, 得到下面的字符串:

| ▼ | Plain Text | 📄 复制代码 |
|---|---|--------|
| 1 | eyJhbGciOiAiSFMyNTYiLCJkaWwIjogIkpXVCJ9.eyJzdWIiOiAiMTIzNDU2IiwgInVzZXJuYW1lIjogInVzZXIiLCJmcm9sZXMiOiBbImFkbWluIiwgInVzZXIiXX0 | |

2. 😊 使用密钥 "secret" 和 HS256 算法对这个字符串进行签名, 得到下面的签名字符串:

| ▼ | Plain Text | 📄 复制代码 |
|---|---|--------|
| 1 | 9Rg8kWkQ1QJ5P2R6fzHm907dQ3QqKZcUxv9zX3YgKQ0 | |

3. 😊 将签名字符串进行 Base64 编码, 得到下面的字符串:

| ▼ | Plain Text | 📄 复制代码 |
|---|--|--------|
| 1 | 0WJlNjY5M2M3NjNiNzQwMjEwZmE5ZTRlMG12YjE4ZDcyNjM5ZmU2YW== | |

4. 😊 将这个字符串与 JWT 中的第三部分进行比较, 如果相同, 则说明 JWT 的真实性和完整性得到了验证。

需要注意的是, 为了保证 JWT 的安全性, 密钥应该保密, 并且只有授权的应用程序才能获得密钥。此外, 应该定期更换密钥和设置适当的过期时间, 以提高 JWT 的安全性。

🌱 HS256、RS256、HMAC256分别是什么？HS/RS/HMAC是什么的缩写？256是什么意思？他们之家有哪些差异？

HS256、RS256、HMAC256是三种常用的加密算法。

- HS是HMAC-SHA的缩写，RS是RSA的缩写，而HMAC是Hash-based Message Authentication Code的缩写。
- 256表示密钥长度为256位。

这三种算法之间的主要差异在于它们的加密方式和使用场景。

- HS256和HMAC256都属于对称加密算法，适用于需要高效加密和解密的场景，比如API请求的签名认证。
- RS256属于非对称加密算法，适用于需要更高的安全性和数字签名的场景，比如网站的SSL证书。

举例来说，如果你想要对API请求进行签名认证，可以使用HS256或HMAC256算法。

- 其中，HS256使用相同的密钥进行加密和解密，而HMAC256使用不同的密钥进行加密和解密。
- 另一方面，如果你需要保护网站的SSL证书，可以使用RS256算法来生成公钥和私钥，并使用私钥对证书进行签名。这样，在客户端使用公钥进行验证时，就可以确保证书的完整性和真实性。

🌟 HS512和HmacSHA512有什么差别？

HS512和HmacSHA512实际上是同一个算法的不同名称，推荐使用后者。

其中，HS512是JJWT库中定义的名称，而HmacSHA512是Java标准库中定义的名称。

🍬 HMAC-SHA中的SHA是什么加密算法？

HMAC-SHA中的SHA是指Secure Hash Algorithm，即安全散列算法。SHA是一种密码哈希函数，用于将任意长度的消息转换为固定长度的哈希值。

SHA算法系列包括SHA-1、SHA-224、SHA-256、SHA-384和SHA-512等不同的变种，其中SHA-256和SHA-512是应用最为广泛的两个版本。在HMAC-SHA算法中，通常使用SHA-256或SHA-512作为哈希函数，用于生成消息认证码。

需要注意的是，由于SHA-1算法存在安全性问题，已经不再推荐使用。因此，在实际应用中，建议使用更加安全的SHA-256或SHA-512算法。

🍬 HMAC和HMAC-SHA有什么区别？

HMAC和HMAC-SHA都是基于哈希函数的消息认证码算法，用于验证消息的完整性和真实性。它们的主要区别在于所使用的哈希函数不同。

具体来说：

- HMAC使用任何可用的哈希函数，比如MD5、SHA-1、SHA-256等；
- HMAC-SHA则是基于SHA系列哈希函数的算法，比如SHA-1、SHA-256、SHA-512等。

举例来说，如果你需要对API请求进行签名认证，可以使用HMAC-SHA256算法。这样，在客户端发送请求时，可以将请求参数和密钥进行组合，并使用HMAC-SHA256算法生成签名。在服务端接收到请求后，再次使用相同的密钥和算法来生成签名，并将生成的签名与客户端发送的签名进行比对。如果两者一致，则说明请求是合法的。

另一方面，如果你需要对密码进行加密和验证，可以使用HMAC算法。比如，在用户注册时，可以将用户输入的密码和随机生成的盐值进行组合，并使用HMAC算法生成加密后的密码。在用户登录时，再次使用相同的盐值和算法来生成加密后的密码，并将生成的密码与数据库中存储的密码进行比对。如果两者一致，则说明用户输入的密码是正确的。

🍬 JWT默认header中alg属性加密算法使用的是什么？可以切换吗？



JWT默认header中的alg属性使用的是HMAC SHA256 (HS256) 算法进行签名。这是因为HMAC SHA256算法是一种常见的对称加密算法，具有高效和安全的特点，适合用于JWT的签名认证。

当然，JWT支持多种不同的加密算法，可以通过修改alg属性来切换使用的算法。常见的加密算法包括：

- HS256：使用HMAC SHA256算法进行签名。
- RS256：使用RSA SHA256算法进行签名。
- ES256：使用ECDSA SHA256算法进行签名。

需要注意的是，不同的加密算法具有不同的特点和使用场景，需要根据具体情况选择合适的算法。同时，由于JWT使用的是对称加密算法，因此在使用时需要注意保护密钥的安全性。

🌱 JWT的两个第三方实现框架

| | | |
|---|---|----------------------|
|  | 1. JJWT :: API io.jsonwebtoken » jjwt-api JJWT :: API Last Release on Apr 28, 2022 | 452 usages Apache |
|  | 1. Java JWT com.auth0 » java-jwt Java implementation of JSON Web Token (JWT) Last Release on Mar 31, 2023 | 760 usages MIT |

`com.auth0:java-jwt` 和 `io.jsonwebtoken:jjwt-api` 对比

1. 开发团队

- `com.auth0:java-jwt` 库的开发者是Auth0公司。
 - Auth0公司成立于2013年，总部位于美国华盛顿州，是一家提供全球统一身份验证平台的技术公司。Auth0的身份验证平台可以帮助开发者轻松地添加身份验证和授权功能，从而减轻开发者在这方面的负担。Auth0的客户遍布全球各个行业，包括金融、医疗、教育、媒体、零售等。Auth0的产品包括身份验证、单点登录、多因素身份验证、API安全等。
- `io.jsonwebtoken:jjwt-api` 库的开发者是Okta公司。
 - Okta公司成立于2009年，总部位于美国加利福尼亚州圣弗朗西斯科，是一家提供安全身份管理和访问管理解决方案的技术公司。Okta的产品可以帮助企业管理用户身份、控制访问权限、保护数据安全，并且可以与各种应用程序和系统进行集成。Okta的客户遍布全球各个行业，包括金融、医疗、教育、零售等。Okta的产品包括身份和访问管理、多因素身份验证、API访问管理等。

2. 依赖关系

- 

```
graph TD
    jjwt-impl["io.jsonwebtoken:jjwt-impl:0.11.5"] --> jjwt-api["io.jsonwebtoken:jjwt-api:0.11.5"]
    jjwt-gson["io.jsonwebtoken:jjwt-gson:0.11.5"] --> jjwt-api
    jjwt-gson --> gson["com.google.code.gson:gson:2.10.1"]
    jjwt-api -.-> jjwt-impl
    jjwt-api -.-> jjwt-gson
    jjwt-api -.-> gson
```
- `jjwt`需要依赖jdk中的`javax`库，`java-awt`则不。

0 个用法

```
private static JwtBuilder getJwtBuilder(String subject, Long ttl, String issuer) {  
    if (Objects.isNull(ttl)) {  
        ttl = 60 * 60 * 1000L;  
    }  
    long l1 = System.currentTimeMillis();  
    Date start = new Date(l1);  
    Date end = new Date(l1 + ttl);  
    return Jwts.builder()  
        .setSubject(subject)  
        .setIssuer(issuer)  
        .setIssuedAt(start)  
        .setExpiration(end)  
        .signWith(generateKey(), SignatureAlgorithm.HS256);  
}  
    Key接口 (java.security)
```

// signature签名密钥

1 个用法

Key子接口 (javax)

```
private static SecretKey generateKey() {  
    final String JWT_KEY = "123456788";  
    byte[] jwtKeyBytes = JWT_KEY.getBytes();  
    return new SecretKeySpec(jwtKeyBytes, offset: 0, jwtKeyBytes.length, algorithm: "AES");  
}  
    SecretKey接口实现类 (javax)
```

🌱JJWT的生成与解析 (🚀)

<https://mvnrepository.com/search?q=jjwt>

<https://mvnrepository.com/search?q=jjwt-impl>

<https://mvnrepository.com/search?q=jjwt-gson>


```
1 <dependency>
2   <groupId>io.jsonwebtoken</groupId>
3   <artifactId>jjwt-impl</artifactId>
4   <version>0.11.5</version>
5 </dependency>
6 <dependency>
7   <groupId>io.jsonwebtoken</groupId>
8   <artifactId>jjwt-gson</artifactId>
9   <version>0.11.5</version>
10 </dependency>
```

```
1  import com.alibaba.fastjson2.JSONObject;
2  import com.alibaba.fastjson2.JSONWriter;
3  import com.example.springbootmpandtk.dal.entity.User;
4  import com.google.gson.Gson;
5  import io.jsonwebtoken.Claims;
6  import io.jsonwebtoken.JwtBuilder;
7  import io.jsonwebtoken.Jwts;
8  import io.jsonwebtoken.gson.io.GsonSerializer;
9  import org.apache.commons.lang3.StringUtils;
10 import org.junit.jupiter.api.Test;
11
12 import javax.crypto.SecretKey;
13 import javax.crypto.spec.SecretKeySpec;
14 import java.time.Duration;
15 import java.time.Instant;
16 import java.time.temporal.ChronoUnit;
17 import java.util.Date;
18 import java.util.HashMap;
19 import java.util.Objects;
20
21 /**
22  * JWT(JJWT)测试类
23  *
24  * @Description JWT(JJWT)测试类
25  * @Author Trivis
26  * @Date 2023/5/27 23:36
27  * @Version 1.0
28  */
29 class JJWTTest {
30
31     // 默认30分钟过期
32     private static final Duration DEFAULT_TTL = Duration.of(30, ChronoUnit.MINUTES);
33     private static final String JWT_SUBJECT = "www.*****.com";
34     private static final String JWT_ISSUER = "www.*****.com";
35     private static final String JWT_AUDIENCE = "anonymous";
36
37     private static String generateJWT(String subject, String issuer) {
38         return generateJWT(subject, issuer, DEFAULT_TTL);
39     }
40
41     private static String generateJWT(String subject, String issuer, Duration ttlDuration) {
42         return getJwtBuilder(subject, issuer, ttlDuration)
43             .serializeToJsonWith(new GsonSerializer<>(new Gson()))
```

```

44         .compact();
45     }
46
47     private static JwtBuilder getJwtBuilder(String subject, String issuer
48 , Duration ttlDuration) {
49         if (Objects.isNull(ttlDuration)) {
50             ttlDuration = DEFAULT_TTL;
51         }
52         // 初始化jwt发布时间与过期时间
53         long l1 = System.currentTimeMillis();
54         Date start = new Date(l1);
55         Date end = new Date(l1 + ttlDuration.toMillis());
56
57         // 添加自定义payload
58         User user = new User();
59         user.setId(999999L);
60         user.setUsername("user-999999");
61         user.setPassword("password-9999999");
62         HashMap<String, Object> claimsMap = new HashMap<>();
63         claimsMap.put("userInfo", JSONObject.toJSONString(user));
64
65         return Jwts.builder()
66             .setSubject(subject)
67             .setIssuer(issuer)
68             .setIssuedAt(start)
69             .setExpiration(end)
70             // 指定接受者（可在解析时使用requireAudience进行验证）
71             .setAudience(JWT_AUDIENCE)
72             // 添加payload，前方的sub/iss/iat/exp/aud都会默认添加到payload中，请合理区分setClaims和addClaims的差异，合理使用。
73             .addClaims(claimsMap)
74             // 指定签名秘钥
75             .signWith(generateKey());
76     }
77
78     // signature签名秘钥
79     // 密钥位数必须大于256位，一个字符按照8位算，至少32个字符。
80     private static SecretKey generateKey() {
81         final String JWT_KEY = StringUtils.repeat("123456788", 8);
82         byte[] jwtKeyBytes = JWT_KEY.getBytes();
83         return new SecretKeySpec(jwtKeyBytes, 0, jwtKeyBytes.length, "Hmac
84 cSHA512");
85     }
86
87     public static Claims parseJWT(String jwt) {
88         return Jwts.parserBuilder()
89             .requireSubject(JWT_SUBJECT)
90             .requireIssuer(JWT_ISSUER)

```

```

89         .requireAudience(JWT_AUDIENCE)
90         .setSigningKey(generateKey())
91         .build()
92         .parseClaimsJws(jwt)
93         .getBody();
94     }
95
96     public static boolean isJWTExpired(Claims claims) {
97         return claims.getExpiration().before(new Date());
98     }
99
100     @Test
101     void test01() throws InterruptedException {
102         System.out.println(Instant.now().toEpochMilli());
103         System.out.println(System.currentTimeMillis());
104
105         // 生成一个JWT (1s过期)
106         Instant begin = Instant.now();
107         String jwt = generateJWT(JWT_SUBJECT, JWT_ISSUER, Duration.of(2,
108 ChronoUnit.SECONDS));
109         Instant end = Instant.now();
110         System.out.println("JWT生成耗时: " + ChronoUnit.MILLIS.between(begin, end) + "ms"); // 382ms
111
112         // 解析JWT
113         Instant begin1 = Instant.now();
114         Claims claims = parseJWT(jwt);
115         Instant end1 = Instant.now();
116         System.out.println("JWT解析耗时: " + ChronoUnit.MILLIS.between(begin1, end1) + "ms"); // 35ms
117         System.out.println(JSONObject.toJSONString(claims, JSONWriter.Feature.PrettyFormat));
118
119         // 判断JWT是否过期
120         Thread.sleep(Duration.ofSeconds(2).toMillis());
121         boolean jwtExpired = isJWTExpired(claims);
122         System.out.println("jwtExpired = " + jwtExpired);
123     }
124 }

```

```
1 package com.abc.system.common.security.helper;
2
3 import com.abc.system.common.constant.SystemRetCodeConstants;
4 import com.abc.system.common.exception.jwt.JWTException;
5 import com.abc.system.common.helper.SpringHelper;
6 import com.abc.system.common.security.config.JWTProperties;
7 import com.abc.system.common.security.util.AESUtils;
8 import com.google.gson.Gson;
9 import io.jsonwebtoken.Claims;
10 import io.jsonwebtoken.JwtBuilder;
11 import io.jsonwebtoken.Jwts;
12 import io.jsonwebtoken.gson.io.GsonSerializer;
13 import lombok.extern.slf4j.Slf4j;
14
15 import javax.crypto.SecretKey;
16 import javax.crypto.spec.SecretKeySpec;
17 import java.nio.charset.StandardCharsets;
18 import java.time.Duration;
19 import java.time.temporal.ChronoUnit;
20 import java.util.Date;
21 import java.util.HashMap;
22 import java.util.Objects;
23
24 /**
25  * JWT生成与解析工具
26  *
27  * @Description
28  * <pre>
29  * JWT生成与解析工具
30  * 1.{@code String generateJWT(String content, String currentSystemName)}
31  * 2.{@code boolean validateJWT(String jwt, String currentSystemName)}
32  * 3.{@code String parseUserInfo(String jwt, String currentSystemName)}
33  * </pre>
34  * @Author Trivis
35  * @Date 2023/5/28 8:30
36  * @Version 1.0
37  */
38 @Slf4j
39 public class JWTHelper {
40
41     /**
42      * 获取JWT字符串
43      *
44      * @param content JWT自定义payload
45      * @param currentSystemName 当前系统名, 作为Audience
```

```

46     * @return JWT字符串
47     */
48     public static String generateJWT(String content, String currentSystem
Name) {
49         JWTProperties jwtProperties = SpringHelper.getBean(JWTProperties.
class);
50         String encryptionSecret = jwtProperties.getEncryptionSecret();
51         String encryptedContent = new AESUtils(content).encrypt(encryptio
nSecret);
52         String issuer = jwtProperties.getIssuer();
53         return generateJWT(encryptedContent, issuer, currentSystemName);
54     }
55
56     /**
57     * 校验JWT字符串
58     *
59     * @param jwt            JWT字符串
60     * @param currentSystemName 当前系统名, 作为Audience
61     * @return 校验是否通过 (true是, false否)
62     */
63     public static boolean validateJWT(String jwt, String currentSystemNam
e) {
64         boolean valid;
65         try {
66             valid = isJWTExpired(parseJWT(jwt, currentSystemName));
67         } catch (Exception e) {
68             valid = false;
69             log.error(">>>>>>>|JWT校验失败|e:{}|<<<<<<<", e.getMessage()
, e);
70         }
71         return valid;
72     }
73
74     /**
75     * 解析JWT字符串中自定义payload (key="user")
76     *
77     * @param jwt            JWT字符串
78     * @param currentSystemName 当前系统名称 (作为Audience)
79     * @return 用户信息
80     */
81     public static String parseUserInfo(String jwt, String currentSystemNa
me) {
82         try {
83             return parseJWT(jwt, currentSystemName).get("user").toString(
);
84         } catch (Exception e) {
85             log.error(">>>>>>>|JWT解析用户信息失败|e:{}|<<<<<<<", e.getMe
ssage(), e);

```

```

86         throw new JWTException(SystemRetCodeConstants.JWT_PARSE_ERRORR
87     );
88     }
89 }
90 /**
91  * 解析JWT
92  *
93  * @param jwt          JWT
94  * @param currentSystemName Audience
95  * @return {@code io.jsonwebtoken.Claims}
96  */
97 private static Claims parseJWT(String jwt, String currentSystemName)
98 {
99     JWTProperties jwtProperties = SpringHelper.getBean(JWTProperties.
100 class);
101     try {
102         return Jwts.parserBuilder()
103             .requireIssuer(jwtProperties.getIssuer())
104             .requireAudience(currentSystemName)
105             .setSigningKey(generateKey())
106             .build()
107             .parseClaimsJws(jwt)
108             .getBody();
109     } catch (Exception e) {
110         log.error(">>>>>>|JWT解析异常|e:{}|<<<<<<<", e.getMessage()
111 , e);
112         throw new JWTException(SystemRetCodeConstants.JWT_PARSE_ERRORR
113 );
114     }
115 }
116 /**
117  * 判断JWT是否过期，在校验JWT通过获取Claims后，需要根据Claims检测JWT是否过期
118  *
119  * @param claims {@code io.jsonwebtoken.Claims}
120  * @return 是否过期 (true是, false否)
121  */
122 private static boolean isJWTExpired(Claims claims) {
123     return claims.getExpiration().before(new Date());
124 }
125 /**🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏*/
126 /**
127  * 生成JWT字符串
128  *
129  * @param content 自定义payload (user: content)

```

```

129     * @param issuer    JWT发布者
130     * @param audience JWT受众
131     * @return JWT字符串
132     */
133     private static String generateJWT(String content, String issuer, String audience) {
134         JWTProperties jwtProperties = SpringHelper.getBean(JWTProperties.class);
135         Duration duration = Duration.of(jwtProperties.getExpiration(), ChronoUnit.MINUTES);
136         return generateJWT(content, issuer, audience, duration);
137     }
138
139     /**
140     * 生成JWT字符串
141     *
142     * @param content    自定义payload (user: content)
143     * @param issuer     JWT发布者
144     * @param audience   JWT受众
145     * @param ttlDuration JWT存活时间范围 (Duration)
146     * @return JWT字符串
147     */
148     private static String generateJWT(String content, String issuer, String audience, Duration ttlDuration) {
149         return getJwtBuilder(content, issuer, audience, ttlDuration)
150             .serializeToJsonWith(new GsonSerializer<>(new Gson()))
151             .compact();
152     }
153
154     /**
155     * 获得JwtBuilder
156     *
157     * @param content    自定义payload (user: content)
158     * @param issuer     JWT发布者
159     * @param audience   JWT受众
160     * @param ttlDuration JWT存活时间范围 (Duration)
161     * @return {@code io.jsonwebtoken.JwtBuilder}
162     */
163     private static JwtBuilder getJwtBuilder(String content, String issuer, String audience, Duration ttlDuration) {
164         if (Objects.isNull(ttlDuration)) {
165             ttlDuration = Duration.of(30L, ChronoUnit.MINUTES);
166         }
167         // 初始化jwt发布时间与过期时间
168         long iat = System.currentTimeMillis();
169         long exp = iat + ttlDuration.toMillis();
170         Date start = new Date(iat);
171         Date end = new Date(exp);

```



```
172
173      // 添加自定义payload
174      // user: content
175      HashMap<String, Object> claimsMap = new HashMap<>(1);
176      claimsMap.put("user", content);
177
178      try {
179
```

🌱Java-JWT的生成与解析 (🚀)

<https://mvnrepository.com/search?q=java-jwt>

XML | 复制代码

```
1 <dependency>
2     <groupId>com.auth0</groupId>
3     <artifactId>java-jwt</artifactId>
4     <version>4.4.0</version>
5 </dependency>
```

```
1 package com.abc.system.common.security.helper;
2
3 import com.abc.system.common.constant.SystemRetCodeConstants;
4 import com.abc.system.common.exception.jwt.JWTException;
5 import com.abc.system.common.helper.SpringHelper;
6 import com.abc.system.common.security.config.JWTProperties;
7 import com.abc.system.common.security.util.AESUtils;
8 import com.auth0.jwt.JWT;
9 import com.auth0.jwt.JWTVerifier;
10 import com.auth0.jwt.algorithms.Algorithm;
11 import com.auth0.jwt.interfaces.Claim;
12 import com.auth0.jwt.interfaces.DecodedJWT;
13 import lombok.extern.slf4j.Slf4j;
14
15 import java.time.Duration;
16 import java.time.Instant;
17 import java.time.temporal.ChronoUnit;
18 import java.util.Date;
19 import java.util.Map;
20
21 /**
22  * JWTHelper1
23  *
24  * @Description <pre>
25  * JWT生成与解析工具(auth0:jav-jwt)
26  * 1.{@code String generateJWT(String content)}
27  *   {@code String generateJWT(String content, String currentSystemName)}
28  * 2.{@code boolean validateJWT(String jwt)}
29  *   {@code boolean validateJWT(String jwt, String currentSystemName)}
30  * 3.{@code String parseUserInfo(String jwt)}
31  *   {@code String parseUserInfo(String jwt, String currentSystemName)}
32  * </pre>
33  * @Author Trivis
34  * @Date 2023/5/28 11:26
35  * @Version 1.0
36  */
37 @Slf4j
38 public class JWTHelper1 {
39
40     /**
41      * 获取JWT字符串
42      *
43      * @param content          JWT自定义payload
44      * @param currentSystemName 当前系统名, 作为Audience
45      * @return JWT字符串
```

```

46     */
47     public static String generateJWT(String content, String currentSystem
Name) {
48         JWTProperties jwtProperties = SpringHelper.getBean(JWTProperties.
class);
49         String encryptionSecret = jwtProperties.getEncryptionSecret();
50         String encryptedContent = new AESUtils(content).encrypt(encryptio
nSecret);
51         String issuer = jwtProperties.getIssuer();
52         return generateJWT(encryptedContent, issuer, currentSystemName);
53     }
54
55     public static String generateJWT(String content) {
56         return generateJWT(content, null);
57     }
58
59     /**
60      * 校验JWT字符串
61      *
62      * @param jwt          JWT字符串
63      * @param currentSystemName 当前系统名, 作为Audience
64      * @return 校验是否通过 (true是, false否)
65      */
66     public static boolean validateJWT(String jwt, String currentSystemNam
e) {
67         boolean valid;
68         try {
69             valid = isJWTExpired(parseJWT(jwt, currentSystemName));
70         } catch (Exception e) {
71             valid = false;
72             log.error(">>>>>>>|JWT校验失败|e:{}|<<<<<<<<", e.getMessage()
, e);
73         }
74         return valid;
75     }
76
77     public static boolean validateJWT(String jwt) {
78         return validateJWT(jwt, null);
79     }
80
81     /**
82      * 解析JWT字符串中自定义payload (key="user")
83      *
84      * @param jwt          JWT字符串
85      * @param currentSystemName 当前系统名称 (作为Audience)
86      * @return 用户信息
87      */
88

```

```

89     public static String parseUserInfo(String jwt, String currentSystemName) {
90         try {
91             Map<String, Claim> stringClaimMap = parseJWT(jwt, currentSystemName);
92             if (!isJWTExpired(stringClaimMap)) {
93                 return parseJWT(jwt, currentSystemName).get("user").asString();
94             } else {
95                 throw new RuntimeException("JWT已过期");
96             }
97         } catch (Exception e) {
98             log.error(">>>>>>>|JWT解析用户信息失败|e:{}|<<<<<<<", e.getMessage(), e);
99             throw new JWTException(SystemRetCodeConstants.JWT_PARSE_ERROR);
100        };
101    }
102
103    public static String parseUserInfo(String jwt) {
104        return parseUserInfo(jwt, null);
105    }
106
107    /*🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏🍏*/
108
109    /**
110     * 解析JWT
111     *
112     * @param jwt          JWT
113     * @param currentSystemName Audience
114     * @return {@code Map<String, Claim>} com.auth0.jwt.interfaces.Claim} → iss/aud/iat/exp/自定义key
115     */
116    private static Map<String, Claim> parseJWT(String jwt, String currentSystemName) {
117        try {
118            JWTProperties jwtProperties = SpringHelper.getBean(JWTProperties.class);
119            Algorithm algorithm = Algorithm.HMAC256(jwtProperties.getSignatureSecret());
120            JWTVerifier verifier = JWT.require(algorithm)
121                .withIssuer(jwtProperties.getIssuer())
122                .withAudience(currentSystemName)
123                .build();
124            DecodedJWT decodedJWT = verifier.verify(jwt);
125            return decodedJWT.getClaims();
126        } catch (Exception e) {

```

```

127         log.error(">>>>>>>|JWT解析异常|e:{}|<<<<<<<<", e.getMessage()
128         , e);
129         throw new JWTException(SystemRetCodeConstants.JWT_PARSE_ERROR
130         );
131     }
132 }
133
134 /**
135  * 判断JWT是否过期，在校验JWT通过获取Claims后，需要根据Claims检测JWT是否过期
136  *
137  * @param claims {@code io.jsonwebtoken.Claims}
138  * @return 是否过期 (true是, false否)
139  */
140 private static boolean isJWTExpired(Map<String, Claim> claims) {
141     Instant exp = Instant.ofEpochSecond(claims.get("exp").asLong());
142     return exp.isBefore(Instant.now());
143 }
144
145 /**
146  * 生成JWT字符串
147  *
148  * @param content 自定义payload (user: content)
149  * @param issuer JWT发布者
150  * @param audience JWT受众
151  * @return JWT字符串
152  */
153 private static String generateJWT(String content, String issuer, String audience) {
154     JWTProperties jwtProperties = SpringHelper.getBean(JWTProperties.class);
155     Duration duration = Duration.of(jwtProperties.getExpiration(), ChronoUnit.MINUTES);
156     return generateJWT(content, issuer, audience, duration);
157 }
158
159 /**
160  * 生成JWT字符串
161  *
162  * @param content 自定义payload (user: content)
163  * @param issuer JWT发布者
164  * @param audience JWT受众
165  * @param ttlDuration JWT存活时间范围 (Duration)
166  * @return JWT字符串
167  */
168 private static String generateJWT(String content, String issuer, String audience, Duration ttlDuration) {
    try {

```

```
169         JWTProperties jwtProperties = SpringHelper.getBean(JWTPropert
170         ies.class);
171         Algorithm algorithm = Algorithm.HMAC256(jwtProperties.getSign
172         atureSecret());
173         long nowMillis = System.currentTimeMillis();
174         Date now = new Date(nowMillis);
175         Date expirationTime = new Date(nowMillis + ttlDuration.toMill
```

End.