**JWT组件说明文档**

二零二一年三月

# JWT组件说明文档

## 编写目的

微服务架构，前后端分离目前已成为互联网项目开发的业界标准，其核心思想就是前端（APP、小程序、H5页面等）通过调用后端的API接口，提交及返回JSON数据进行交互。

在前后端分离项目中，首先要解决的就是登录及授权的问题。微服务架构下，传统的session认证限制了应用的扩展能力，无状态的JWT认证方法应运而生，该认证机制特别适用于分布式站点的单点登录（SSO）场景。

现将JWT封装为组件，供应用一站式使用，解决session跨域，session共享，单点登录等问题。

本文主要介绍JWT组件的功能及实现原理。

## JWT介绍

### JWT是什么

JSON Web Tokens（JWT）是一个开放标准（[RFC 7519](https://tools.ietf.org/html/rfc7519)），它定义了一种紧凑且自包含的方式，用于在各方之间安全地将信息作为JSON对象传输。由于此信息是经过数字签名的，因此可以被验证和信任。可以使用秘密（使用HMAC算法）或使用RSA或ECDSA的公钥/私钥对对JWT进行签名。

### JWT使用场景

**授权(**Authorization**)**：这是使用JWT的最常见方案。用户登录后，每个后续请求都将包含JWT，从而允许用户访问该令牌允许的路由，服务和资源。SSO(Single Sign On)单点登录是当今广泛使用JWT的一项功能，因为它的开销很小并且可以在不同的域中轻松使用。

**信息交换(**Information Exchange**)**：JSON Web Tokens是在各方之间安全地传输信息的一种好方法。因为可以对JWT进行签名（例如，使用公钥/私钥对），所以可以确保请求方身份。此外，由于签名是使用标头和有效负载计算的，因此还可以验证内容是否未被篡改。

### JWT结构

JWT由三部分组成，这些部分由点（.）分隔，分别是：

**标头(Header)**

**载荷(Payload)**

**签名(Signature)**

因此，一个典型的JWT通常如下所示：

xxxxx.yyyyy.zzzzz

一个JWT实际上就是一个字符串，它由三部分组成，头部、载荷与签名。

#### 3.1 标头(Header)

标头通常由两部分组成：令牌的类型（即JWT）和所使用的签名算法，例如HMAC 、SHA256或RSA。

**例如：**

**{**

**"alg": "HS256",**

**"typ": "JWT"**

**}**

**然后，此JSON被Base64Url编码以形成JWT的第一部分。**

#### 3.2 有效载荷（Payload）

令牌的第二部分是有效负载，它包含声明（Claims ）。声明是有关实体（通常是用户）和其他数据的声明。声明有以下三种类型：registered, public 和 private。

**Registered claims** : 这些是一组非强制性的但建议使用的预定义权利要求，以提供一组有用的，可互操作的权利要求。其中一些是： iss（发布者-issuer），exp（到期时间-expiration time），sub（主题-subject），aud（受众群体-audience）等。

iss: jwt签发者

sub: jwt所面向的用户

aud: 接收jwt的一方

exp: jwt的过期时间，这个过期时间必须要大于签发时间

nbf: 定义在什么时间之前，该jwt都是不可用的.

iat: jwt的签发时间

jti: jwt的唯一身份标识，主要用来作为一次性token。

**Public claims** : 公共的声明可以添加任何的信息。一般添加用户的相关信息或其他业务需要的必要信息.但不建议添加敏感信息，因为该部分在客户端可解密。

**Private claims** : 私有声明是提供者和消费者所共同定义的声明，用于在同意使用它们的各方当事人之间建立共享信息，并且不是注册的或公开的声明。一般不建议存放敏感信息，因为base64是对称解密的，意味着该部分信息可以归类为明文信息。

请注意，声明名称仅是三个字符，因为JWT的含义是紧凑的。

有效负载示例：

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

然后，对有效负载进行Base64Url编码，以形成JWT的第二部分。

请注意，对于已签名的令牌，此信息尽管可以防止篡改，但任何人都可以读取。除非将其加密，否则请勿将机密信息放入JWT的有效负载或报头元素中。

#### 3.3 签名(Signature)

要创建签名部分，你必须有编码过的header、编码过的payload、一个秘钥，签名算法是header中指定的那个，然对它们签名即可。

例如，如果要使用HMAC SHA256算法，则将通过以下方式创建签名：

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

secret)

签名用于验证消息在此过程中没有更改，并且对于使用私钥进行签名的令牌，它还可以验证JWT的发送者的身份。

**合成**

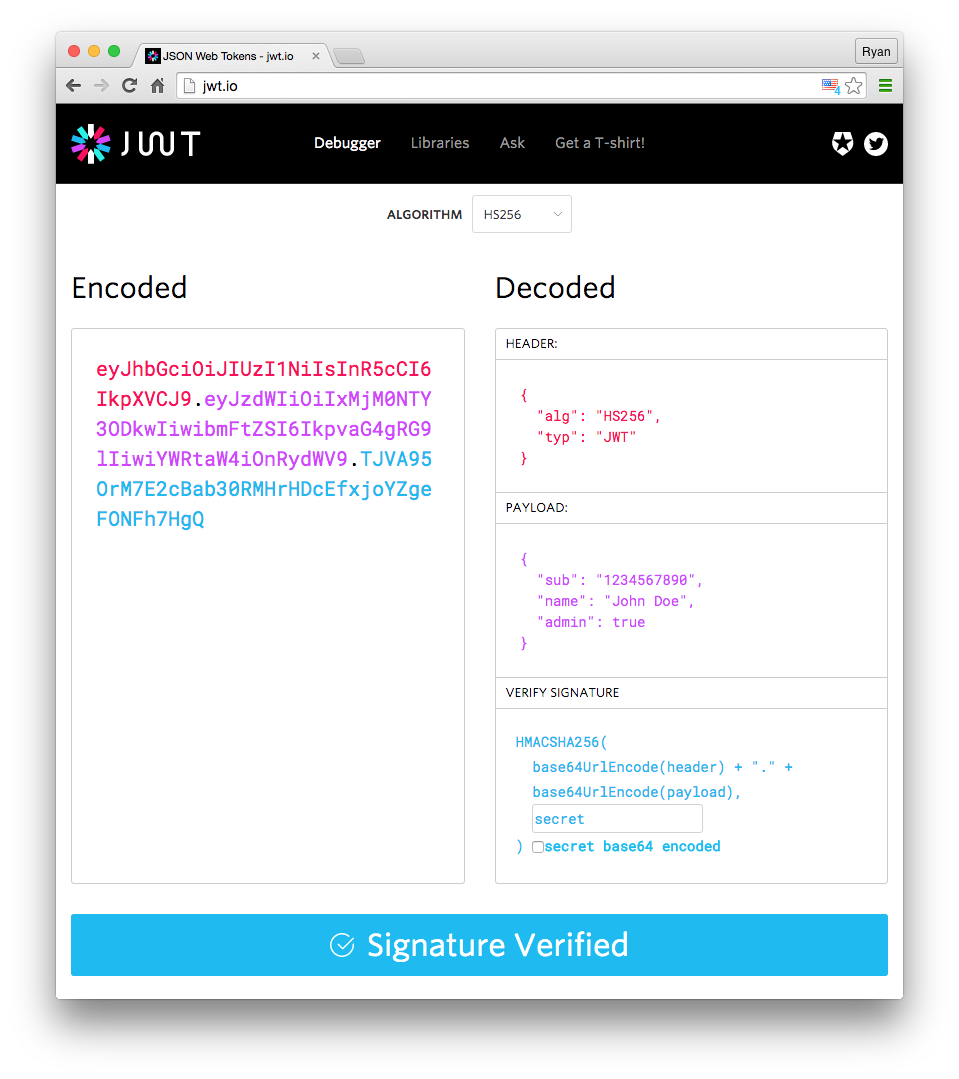
输出是由点分隔的三个Base64-URL字符串，可以在HTML和HTTP环境中轻松传递这些字符串，与基于XML的标准（例如SAML）相比，它更紧凑。

下图显示了一个JWT，它已对先前的标头和有效负载进行了编码，并用一个秘钥进行签名。

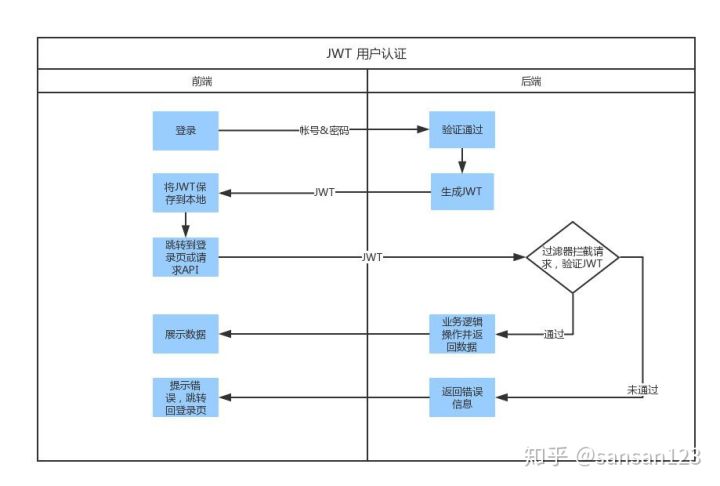


如果您想使用JWT并将这些概念付诸实践，则可以使用[jwt.io Debugger](https://jwt.io/" \l "debugger-io)解码，验证和生成JWT。

<https://jwt.io/#debugger-io>



### JWT如何工作的

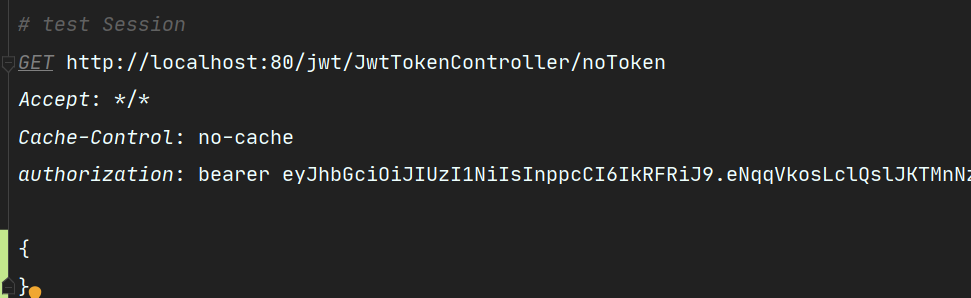


在认证的时候，当用户用他们的凭证成功登录以后，一个JWT将会被返回。此后，token就是用户凭证了，你必须非常小心以防止出现安全问题。一般而言，你保存令牌的时间不应超过你所需要它的时间。

无论何时用户想要访问受保护的路由或者资源的时候，用户代理（通常是浏览器）都应该带上JWT，典型的，通常放在Authorization header中，用Bearer schema。

header应该看起来是这样的：

Authorization: Bearer



服务器上的受保护的路由将会检查Authorization header中的JWT是否有效，如果有效，则用户可以访问受保护的资源。如果JWT包含足够多的必需的数据，那么就可以减少对某些操作的数据库查询的需要，尽管可能并不总是如此。

如果token是在授权头（Authorization header）中发送的，那么跨源资源共享(CORS)将不会成为问题，因为它不使用cookie。

## JWT组件（jwt-module）

JWT组件是基于JWT无状态认证授权模式，统一实现算法加密和密钥，方便各应用一站式无感接入。

JJWT是一个提供端到端的JWT创建和验证的Java库。

### JJWT+SpringSecurity简单实现

*<!-- jwt -->*<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-api</artifactId>  
 <version>0.10.6</version>  
</dependency>  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-impl</artifactId>  
 <version>0.10.6</version>  
</dependency>  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-jackson</artifactId>  
 <version>0.10.6</version>  
</dependency>

<!--Security框架-->  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>

### JWT util

*\*\*  
 \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* Copyright © 2020 远眺科技 Inc.All rights reserved. \* \*\*  
 \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*  
 \** ***@program:*** *redis-demo  
 \** ***@description:*** *jwt Token工具类  
 \** ***@author:*** *cnzz  
 \** ***@create:*** *2020-05-26 09:01  
 \* <p>  
 \* - jwtSecurityProperties 注入  
 \* - afterPropertiesSet jjwt secret HMAC key 生成  
 \* - createToken jwt 生成  
 \* - getExpirationDateFromToken jwt 中获取expiration期限时间  
 \* - getAuthentication 获取用户认证SpringSecuity  
 \* - validateToken jwt 校验  
 \* - getClaimsFromToken jwt获取claims jwtMap  
 \*\*/*@Slf4j  
@Component  
*public class* JwtTokenUtils *implements InitializingBean* {  
 *private final* JwtSecurityProperties jwtSecurityProperties;  
 *private static final* String *AUTHORITIES\_KEY* = "auth";  
 *private Key* key;  
 *public* JwtTokenUtils(JwtSecurityProperties jwtSecurityProperties) {  
 *this*.jwtSecurityProperties = jwtSecurityProperties;  
 }  
 @Override  
 *public void* afterPropertiesSet() {  
 *byte*[] keyBytes = Decoders.*BASE64*.decode(jwtSecurityProperties.getBase64Secret());  
 *log*.debug("keyBytes length={}", keyBytes.length);  
 *this*.key = Keys.*hmacShaKeyFor*(keyBytes);  
 }  
 */\*\*  
 \* JWT token生成  
 \* <p>  
 \* iss: jwt签发者  
 \* sub: jwt所面向的用户  
 \* aud: 接收jwt的一方  
 \* exp: jwt的过期时间，这个过期时间必须要大于签发时间  
 \* nbf: 定义在什么时间之前，该jwt都是不可用的.  
 \* iat: jwt的签发时间  
 \* jti: jwt的唯一身份标识，主要用来作为一次性token。  
 \*  
 \** ***@param userId*** *用户标识  
 \** ***@param userName*** *用户  
 \** ***@param permissionList*** *权限list  
 \** ***@return*** *String jwt  
 \*/  
 public* String createToken(String userId, String userName, *List*<String> permissionList) {  
 String newStr = permissionList.stream().collect(Collectors.*joining*(","));  
 *return* Jwts.*builder*()  
 .claim(*AUTHORITIES\_KEY*, newStr)  
 .setId(userId)  
 .setSubject(userName)  
 .setIssuedAt(*new* Date())  
 .setExpiration(*new* Date(System.*currentTimeMillis*() + jwtSecurityProperties.getTokenValidityInSeconds()))  
 .compressWith(CompressionCodecs.*DEFLATE*)  
 .signWith(key, SignatureAlgorithm.*HS256*)  
 .compact();  
 }  
  
 */\*\*  
 \* jwt 中获取expiration期限时间  
 \*  
 \** ***@param token*** *jwt  
 \** ***@return*** *Date  
 \*/  
 public* Date getExpirationDateFromToken(String token) {  
 Date expiration;  
 *try* {  
 *final Claims* claims = getClaimsFromToken(token);  
 expiration = claims.getExpiration();  
 } *catch* (Exception e) {  
 expiration = *null*;  
 }  
 *return* expiration;  
 }  
 */\*\*  
 \* 获取用户认证SpringSecuity  
 \*  
 \** ***@param token*** *jwt  
 \** ***@return*** *Authentication  
 \*/  
 public Authentication* getAuthentication(String token) {  
 *Claims* claims = Jwts.*parser*()  
 .setSigningKey(key)  
 .parseClaimsJws(token)  
 .getBody();  
 *log*.debug("我的filter|claims={}", claims);  
  
 *Collection*<? *extends GrantedAuthority*> authorities =  
 Arrays.*stream*(claims.get(*AUTHORITIES\_KEY*).toString().split(","))  
 .map(SimpleGrantedAuthority::*new*)  
 .collect(Collectors.*toList*());  
 *log*.debug("我的filter|authorities={}", authorities);  
 *//HashMap map =(HashMap) claims.get("auth");  
 //User principal = new User(map.get("user").toString(),map.get("password").toString(), authorities);* String principal = (String) claims.get("sub");  
 *return new* UsernamePasswordAuthenticationToken(principal, token, authorities);  
 }  
 */\*\*  
 \* jwt校验  
 \*  
 \** ***@param authToken*** *jwt  
 \** ***@return*** *boolean  
 \*/  
 public boolean* validateToken(String authToken) {  
 *try* {  
 Jwts.*parser*().setSigningKey(key).parseClaimsJws(authToken);  
 *return true*;  
 } *catch* (io.jsonwebtoken.security.SecurityException | MalformedJwtException e) {  
 *log*.info("Invalid JWT signature.");  
 e.printStackTrace();  
 } *catch* (ExpiredJwtException e) {  
 *log*.info("Expired JWT token.");  
 e.printStackTrace();  
 } *catch* (UnsupportedJwtException e) {  
 *log*.info("Unsupported JWT token.");  
 e.printStackTrace();  
 } *catch* (IllegalArgumentException e) {  
 *log*.info("JWT token compact of handler are invalid.");  
 e.printStackTrace();  
 }  
 *return false*;  
 }  
  
 */\*\*  
 \* jwt 获取 jwtMap  
 \*  
 \** ***@param token*** *jwt获取 Claims  
 \** ***@return*** *Claims  
 \*/  
 public Claims* getClaimsFromToken(String token) {  
 *//判断为空  
 if* (StringUtils.*isEmpty*(token)) {  
 *return null*;  
 }  
 *if* (StringUtils.*hasText*(token) && token.startsWith(JwtSecurityProperties.*TOKEN\_START\_WITH*)) {  
 token = token.substring(JwtSecurityProperties.*TOKEN\_START\_WITH*.length());  
 }  
 *Claims* claims;  
 *try* {  
 claims = Jwts.*parser*()  
 .setSigningKey(key)  
 .parseClaimsJws(token)  
 .getBody();  
 } *catch* (Exception e) {  
 e.printStackTrace();  
 claims = *null*;  
 }  
 *return* claims;  
 }  
}

### 过滤器filter

*/\*\*  
 \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* Copyright © 2020 远眺科技 Inc.All rights reserved. \* \*\*  
 \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \*  
 \** ***@program:*** *redis-demo  
 \** ***@description:*** *token验证过滤器  
 \** ***@author:*** *cnzz  
 \** ***@create:*** *2020-05-26 09:05  
 \*\*/*@Component  
@Slf4j  
*public class* JwtAuthenticationTokenFilter *extends* OncePerRequestFilter {  
 @Resource  
 *private* JwtTokenUtils jwtTokenUtils;  
 *public* JwtAuthenticationTokenFilter(JwtTokenUtils jwtTokenUtils) {  
 *this*.jwtTokenUtils = jwtTokenUtils;  
 }  
 @Override  
 *protected void* doFilterInternal(*HttpServletRequest* httpServletRequest, *HttpServletResponse* httpServletResponse, *FilterChain* filterChain) *throws* ServletException, IOException {  
 JwtSecurityProperties jwtSecurityProperties = SpringContextHolder.*getBean*(JwtSecurityProperties.*class*);  
 String requestRri = httpServletRequest.getRequestURI();  
 *//获取request token* String token = *null*;  
 String bearerToken = httpServletRequest.getHeader(JwtSecurityProperties.*HEADER*);  
 *log*.debug("从requst获取的|bearerToken={}",bearerToken);  
 *if* (StringUtils.*hasText*(bearerToken) && bearerToken.startsWith(JwtSecurityProperties.*TOKEN\_START\_WITH*)) {  
 token = bearerToken.substring(JwtSecurityProperties.*TOKEN\_START\_WITH*.length());  
 }  
 *if* (StringUtils.*hasText*(token) && jwtTokenUtils.validateToken(token)) {  
 *Authentication* authentication = jwtTokenUtils.getAuthentication(token);  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 *log*.debug("set Authentication to security context for '{}', uri: {}", authentication.getName(), requestRri);  
 } *else* {  
 *log*.debug("no valid JWT token found, uri: {}", requestRri);  
 }  
 filterChain.doFilter(httpServletRequest, httpServletResponse);  
 }  
}

## 附件

参考：

JWT官网文档 <https://jwt.io/introduction>

JWT <https://www.jianshu.com/p/99a458c62aa4>

五分钟带你了解啥是JWT <https://zhuanlan.zhihu.com/p/86937325>

Java的JJWT实现JWT <https://www.jianshu.com/p/278ad96dc7f3>

文件



Git 地址