#### 什么是JSON Web令牌？

JSON Web令牌（JWT）是一个开放标准（RFC 7519），它定义了一种紧凑且自包含的方式，用于在各方之间作为JSON对象安全地传输信息。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。

#### 什么时候应该使用JSON Web Token？

* 授权：一旦用户登录，每个后续请求将包括JWT，从而允许用户访问该token允许的路由，服务和资源。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：JSON Web Token是在各方之间安全地传输信息的好方法。因为可以对JWT进行签名（例如，使用公钥/私钥对），所以你可以确定发件人是他们所说的人。另外，由于签名是使用标头和有效负载计算的，因此可以验证内容是否被篡改。

#### JSON Web Token结构是什么？

三部分组成Header.Payload.Signature

JWT通常如下所示：

xxxxx.yyyyy.zzzzz

**header**

header通常由两部分组成：token的类型（即JWT）和所使用的签名算法。

示例：

{

"alg": "HS256", 使用的签名算法

"typ": "JWT" token的类型

}

此JSON被Base64Url编码以形成JWT的第一部分。

**payload**

其中包含声明。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：

* 注册声明：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，可互操作的声明。其中一些是：iss（发出者），exp（到期时间），sub（主题），aud（受众）等。（注意声明名称仅是三个字符，因为JWT是紧凑的）
* 公共声明：使用JWT的人可以随意定义这些声明。但是为避免冲突，应在IANA JSON Web Token Registry中定义它们，或将其定义为包含抗冲突名称空间的URI。
* 私人声明：这些是自定义声明，旨在在同意使用它们的各方之间共享信息，既不是注册声明也不是公共声明。

示例：

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

此JSON被Base64Url编码以形成JWT的第二部分。

注意：对于已签名的token，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payload或header中。

**signature**

要创建signature部分，必须获取编码的header（Base64后的），编码的payload（Base64后的），私钥，header中指定的算法，并对其进行签名。

示例：

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，并且对使用私钥进行签名的令牌，它还可以验证JWT的发送者是它所说的真实身份。

此JSON被Base64Url编码以形成JWT的第三部分。



#### JWT工具类

* generateToken(UserDetails userDetails) :用于根据登录用户信息生成token
* getUserNameFromToken(String token)：从token中获取登录用户的信息
* validateToken(String token, UserDetails userDetails)：判断token是否有效

@Component

public class JwtTokenUtil {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(JwtTokenUtil.class);  
 private static final String *CLAIM\_KEY\_USERNAME* = "sub";  
 private static final String *CLAIM\_KEY\_CREATED* = "created";  
  
 */\*\*  
 \* 秘钥  
 \*/* @Value("${jwt.secret}")  
 private String secret;  
  
 */\*\*  
 \* 过期时长  
 \*/* @Value("${jwt.expiration}")  
 private Long expiration;  
  
 */\*\*  
 \* 1、根据用户信息生成token  
 \*/* public String generateToken(UserDetails userDetails) {  
 HashMap<String, Object> claims = new HashMap<>();  
 claims.put(*CLAIM\_KEY\_USERNAME*, userDetails.getUsername());  
 claims.put(*CLAIM\_KEY\_CREATED*, new Date());  
 return generateToken(claims);  
 }  
  
 */\*\*  
 \* 根据登录用户信息生成token  
 \*/* private String generateToken(Map<String, Object> claims) {  
 return Jwts.*builder*()  
 .setClaims(claims)  
 .setExpiration(generateExpirationDate())  
 .signWith(SignatureAlgorithm.*ES512*, secret)  
 .compact();  
 }  
  
 */\*\*  
 \* 生成token的过期时间  
 \*/* private Date generateExpirationDate() {  
 return new Date(System.*currentTimeMillis*() + expiration \* 1000);  
 }  
  
 */\*\*  
 \* 2、从token中获取登录用户名  
 \*/* public String getUserNameFromToken(String token) {  
 String username;  
 try {  
 Claims claims = getClaimsFromToken(token);  
 username = claims.getSubject();  
 } catch (Exception e) {  
 username = null;  
 }  
 return username;  
 }  
  
 */\*\*  
 \* 从token中获取JWT中的payload  
 \*/* private Claims getClaimsFromToken(String token) {  
 Claims claims = null;  
 try {  
 claims = Jwts.*parser*()  
 .setSigningKey(secret)  
 .parseClaimsJws(token)  
 .getBody();  
 } catch (Exception e) {  
 *LOGGER*.info("JWT格式验证失败", token);  
 }  
 return claims;  
 }  
  
 */\*\*  
 \* 3、验证token是否还有效  
 \*  
 \** ***@param*** *token 客户端传入的token  
 \** ***@param*** *userDetails 从数据库中查询出来的用户信息  
 \** ***@return*** *\*/* public boolean validateToken(String token, UserDetails userDetails) {  
 //从token中获取登录用户名  
 String userName = getUserNameFromToken(token);  
 return userName.equals(userDetails.getUsername()) && !isTokenExpired(token);  
 }  
  
 */\*\*  
 \* 判断token是否已经失效  
 \*/* private boolean isTokenExpired(String token) {  
 Date expiredDate = getExpiredDateFromToken(token);  
 return expiredDate.before(new Date());  
 }  
  
 */\*\*  
 \* 从token中获取过期时间  
 \*/* private Date getExpiredDateFromToken(String token) {  
 Claims claims = getClaimsFromToken(token);  
 return claims.getExpiration();  
 }  
  
 */\*\*  
 \* 判断token是否可以被刷新  
 \*/* public boolean canRefresh(String token) {  
 return !isTokenExpired(token);  
 }  
  
 */\*\*  
 \* 刷新token  
 \*/* public String refreshToken(String token) {  
 Claims claims = getClaimsFromToken(token);  
 claims.put(*CLAIM\_KEY\_CREATED*, new Date());  
 return generateToken(claims);  
 }  
  
}