# JWT 生成Token及验证


## JWT 简介

JWT是一种用于双方之间传递安全信息的简洁的、URL安全的表述性声明规范。JWT作为一个开放的标准（RFC 7519），定义了一种简洁的，自包含的方法用于通信双方之间以Json对象的形式安全的传递信息。因为数字签名的存在，这些信息是可信的，JWT可以使用HMAC算法或者是RSA的公私秘钥对进行签名。简洁(Compact): 可以通过URL，POST参数或者在HTTP header发送，因为数据量小，传输速度也很快 自包含(Self-contained)：负载中包含了所有用户所需要的信息，避免了多次查询数据库。

## JWT 的结构

JWT包含了使用.分隔的三部分： Header 头部 Payload 负载 Signature 签名
其结构看起来是这样的Header.Payload.Signature.

## Header

在header中通常包含了两部分：token类型和采用的加密算法。{ "alg": "HS256", "typ": "JWT"} 接下来对这部分内容使用 Base64Url 编码组成了JWT结构的第一部分。(用来验证签名用)

## Payload

Token的第二部分是负载，它包含了claim， Claim是一些实体（通常指的用户）的状态和额外的元数据，有三种类型的claim：reserved, public 和 private.Reserved claims: 这些claim是JWT预先定义的，在JWT中并不会强制使用它们，而是推荐使用，常用的有 iss（签发者）,exp（过期时间戳）, sub（面向的用户）, aud（接收方）, iat（签发时间）。Public claims：根据需要定义自己的字段，注意应该避免冲突 Private claims：这些是自定义的字段，可以用来在双方之间交换信息 负载使用的例子：{ "sub": "1234567890", "name": "John Doe", "admin": true} 上述的负载需要经过Base64Url编码后作为JWT结构的第二部分。

## Signature

创建签名需要使用编码后的header和payload以及一个秘钥，使用header中指定签名算法进行签名。例如如果希望使用HMAC SHA256算法，那么签名应该使用下列方式创建：HMACSHA256( base64UrlEncode(header) + "." + base64UrlEncode(payload), secret) 签名用于验证消息的发送者以及消息是没有经过篡改的。 完整的JWT 完整的JWT格式的输出是以. 分隔的三段Base64编码，与SAML等基于XML的标准相比，JWT在HTTP和HTML环境中更容易传递。 下列的JWT展示了一个完整的JWT格式，它拼接了之前的Header， Payload以及秘钥签名：

## 实例

### 生成TOKEN

```java
//公用密钥-保存在服务端，客户端是不会知道密钥的，以防被攻击
public static String SECRET = "ThisIsASecret";
/**
 * 生成Token
 * @return
 * @throws Exception
 */
public static String createToken() throws Exception {
    //签发时间
    Date iatDate = new Date();
    //过期时间 1 分钟后过期
    Calendar nowTime = Calendar.getInstance();
    nowTime.add(Calendar.MINUTE, amount: 10);
    Date expiresDate = nowTime.getTime();

    Map<String, Object> map = new HashMap<>();
    map.put("alg", "HS256");
    map.put("type", "JWT");
    String token = JWT.create()
            .withHeader(map)
            .withClaim( name: "name", value: "Free码农")
            .withClaim( name: "age", value: "12")
            .withClaim( name: "org", value: "测试")
            .withExpiresAt(expiresDate)//设置过期时间，过期时间要大于签发时间
            .withIssuedAt(iatDate)//设置签发时间
            .sign(Algorithm.HMAC256(SECRET));//加密
    return token;
}
```

## 校验TOKEN

```java
//校验token
public static boolean verifyToken(String token) throws UnsupportedEncodingException {
    JWTVerifier verifier = JWT.require(Algorithm.HMAC256(SECRET)).build();
    try {
        verifier.verify(token);
        return true;
    } catch (Exception e) {
        return false;
    }

}
```

## 获取Token信息

```java
//获取token的信息

public static DecodedJWT getTokenInfo(String token) throws UnsupportedEncodingException {
    JWTVerifier verifier = JWT.require(Algorithm.HMAC256(SECRET)).build();
    try {
        return verifier.verify(token);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

测试

```java
public static void main(String[] args) throws Exception {
    //获取token
    String token = JwtTokenUtil.createToken();
    System.out.println("token:" + token);
    //校验TOKEN
    boolean verify=JwtTokenUtil.verifyToken(token);
    System.out.println("Token 校验结果"+verify);

    //获取Token信息
    DecodedJWT tokenInfo = JwtTokenUtil.getTokenInfo(token);
    String alg=tokenInfo.getHeaderClaim( s: "alg").asString();
    System.out.println("header alg:"+alg);
    Map<String, Claim> claims = tokenInfo.getClaims();
    System.out.println("claim org:"+claims.get("org").asString());
    System.out.println("claim org:"+claims.get("exp").asDate());
}
```

## 源码解析

### 生成TOKEN主要源码

```java
private String sign() throws SignatureGenerationException {
    String header = Base64.encodeBase64URLSafeString(this.headerJson.getBytes(StandardCharsets.UTF_8));
    String payload = Base64.encodeBase64URLSafeString(this.payloadJson.getBytes(StandardCharsets.UTF_8));
    String content = String.format("%s.%s", header, payload);
    byte[] signatureBytes = this.algorithm.sign(content.getBytes(StandardCharsets.UTF_8));
    String signature = Base64.encodeBase64URLSafeString(signatureBytes);
    return String.format("%s.%s", content, signature);
}
```

从源码中可以看出 header 和 payload 结果Base64 编码后相加，sign 为 header+"."+payload+secret 加密 然后再Base64编码得到最后的token 格式为 header.payload+sign.

## 校验Token主要源码

```
public DecodedJWT verify(String token) throws JWTVerificationException {
    DecodedJWT jwt = JWT.decode(token);获取TOKEN信息
    this.verifyAlgorithm(jwt, this.algorithm)校验加密方式是否相同
    this.algorithm.verify(jwt);校验签名是否正确
    this.verifyClaims(jwt, this.claims);校验其它信息，如token是否过期
    return jwt;
}
```

JWT.decode(token);

```
String headerJson;
String payloadJson;
try {
    headerJson = StringUtils.newStringUtf8(Base64.decodeBase64(this.parts[0]));
    payloadJson = StringUtils.newStringUtf8(Base64.decodeBase64(this.parts[1]));
} catch (NullPointerException var6) {
    throw new JWTDecodeException("The UTF-8 Charset isn't initialized.", var6);
}
```

this.verifyAlgorithm(jwt, this.algorithm);

```
private void verifyAlgorithm(DecodedJWT jwt, Algorithm expectedAlgorithm) throws AlgorithmMismatchException {
    if (!expectedAlgorithm.getName().equals(jwt.getAlgorithm())) {
        throw new AlgorithmMismatchException("The provided Algorithm doesn't match the one defined in the JWT's Header.");
    }
}
```

this.algorithm.verify(jwt);

```
public void verify(DecodedJWT jwt) throws SignatureVerificationException {
    byte[] contentBytes = String.format("%s.%s", jwt.getHeader(), jwt.getPayload()).getBytes(StandardCharsets.UTF_8);
    byte[] signatureBytes = Base64.decodeBase64(jwt.getSignature());

    try {
        boolean valid = this.crypto.verifySignatureFor(this.getDescription(), this.secret, contentBytes, signatureBytes);
        if (!valid) {
            throw new SignatureVerificationException(this);
        }
    } catch (InvalidKeyException | NoSuchAlgorithmException | IllegalStateException var5) {
        throw new SignatureVerificationException(this, var5);
    }
}
```

this.verifyClaims(jwt, this.claims);

```
case 1:
    this.assertValidDateClaim(jwt.getExpiresAt(), (Long)entry.getValue(), shouldBeFuture: true);
    break;
```

```
private void assertValidDateClaim(Date date, long leeway, boolean shouldBeFuture) {
    Date today = this.clock.getToday();
    today.setTime((long)Math.floor((double)(today.getTime() / 1000L * 1000L)));
    if (shouldBeFuture) {
        this.assertDateIsFuture(date, leeway, today);
    } else {
        this.assertDateIsPast(date, leeway, today);
    }

}
```