Jwt

www.bzMe.work

参考：

<https://jwt.io>

<https://jwt.io/#libraries>

<https://github.com/jwt-dotnet/jwt>

<https://github.com/AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet>

<https://www.cnblogs.com/jackyfei/p/9998918.html>

<https://github.com/auth0/auth0.net>

[Introduction to JSON Web Tokens](https://jwt.io/introduction/) by Auth0

[Sessionless Authentication using JWTs (with Node + Express + Passport JS)](https://medium.com/@bryanmanuele/sessionless-authentication-withe-jwts-with-node-express-passport-js-69b059e4b22c), by Bryan Manuele

[Learn how to use JSON Web Tokens](https://github.com/dwyl/learn-json-web-tokens/blob/master/README.md), by dwyl

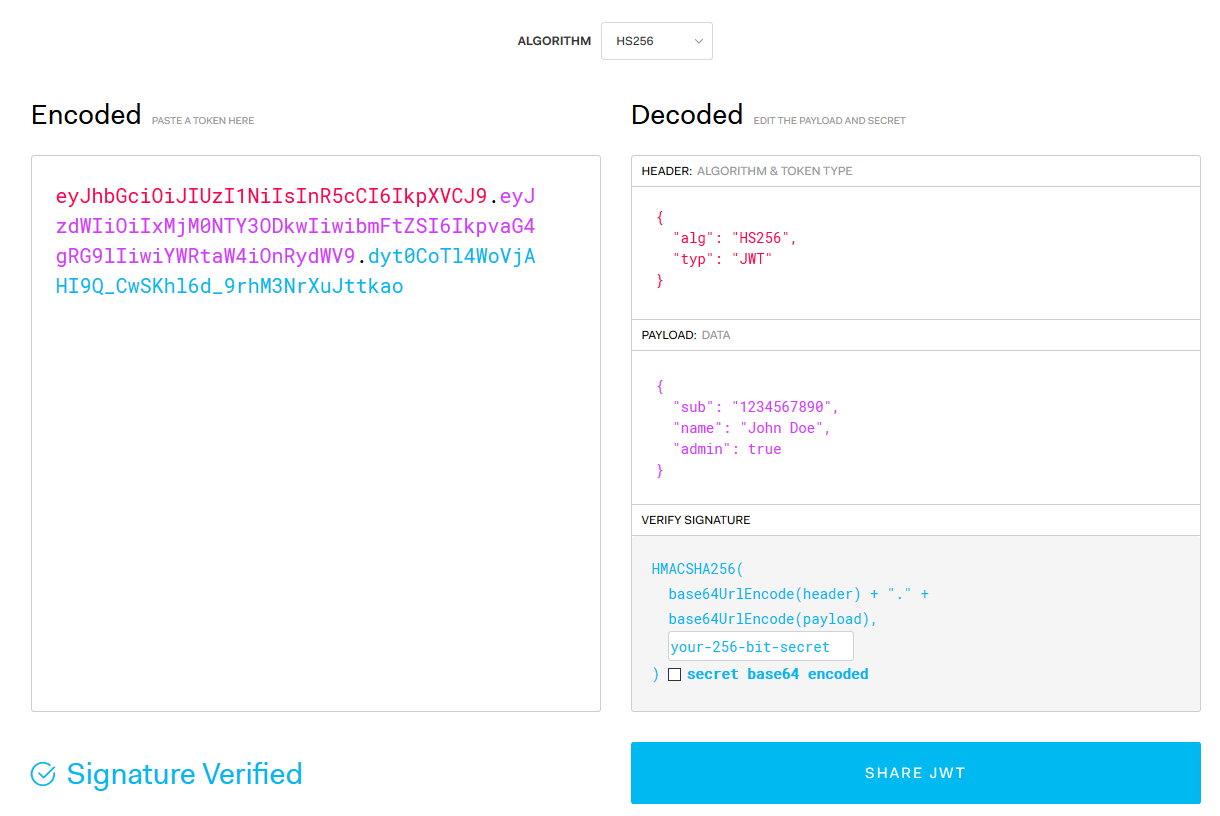
<http://www.ruanyifeng.com/blog/2018/07/json_web_token-tutorial.html>

**什么是JSON Web令牌？**

JSON Web令牌(JSON Web Token，JWT)是开放的，遵循[RFC 7519](https://tools.ietf.org/html/rfc7519)行业标准，用于在双方之间的安全认证。JSON Web令牌的格式如下所示（为可读性插入换行）

|  |
| --- |
| eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9. eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9. TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ |

下图是官方(https://jwt.io/#libraries)的JWT生成器：



**JWT的构成**

JWT由三个部分组成: 头(header)、负载(payload)、签名(signature)。

这三个部分分别编码，然后用点(.)分隔形成一个编码字符串，例如：

|  |
| --- |
| eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9. eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9. TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ |

注意：JWT内部没有换行，此处为了可读性插入了换行。

**1.头(Header):** ALGORITHM & TOKEN TYPE

Header 部分是一个JSON对象，描述JWT的元数据，格式如下：

|  |
| --- |
| {  "alg": "HS256",  "typ": "JWT"  } |

其中：

alg表示签名算法(algorithm)，默认是HMAC SHA256(写成 HS256)。其它的算法还有：

HS384,HS512,RS256,RS384,RS512,ES256,ES384,ES512,PS256,PS384,PS512.

typ表示令牌(token)的类型(type)，统一为JWT。

最后，将上面的JSON对象使用Base64URL算法编码转成字符串。

**2.**负载**(Payload): Data**

所谓负载，就像人的肩上扛着的重物，在JWT中就是要负载的数据，因此”Payload”这个词较为形象。

Payload部分也是一个JSON对象，用来存放实际需要传递的数据，JWT规定了7个官方字段供选用，分别是：

|  |
| --- |
| iss(issuer)：签发人，令牌由谁签发，例如某某公司或个人。  exp(expiration time)：过期时间，令牌的过期时间。  sub(subject)：主题，对令牌的简单描述。  aud(audience)：受众，令牌颁发给谁使用。  nbf(Not Before)：生效时间，令牌的生效时间。  iat(Issued At)：签发时间，令牌的签发时间。  jti(JWT ID)：编号，令牌的编号。 |

除了官方字段，你还可以在这个部分定义私有字段，如下例所示：

|  |
| --- |
| {  "sub": "1234567890",  "name": "John Doe",  "admin": true  } |

最后，将上面的JSON对象使用Base64URL算法编码转成字符串。

注意，负载数据默认是不加密的，任何人都可以读到，所以不要把秘密信息放在这个部分。

**3.签名(Signature)**

Signature部分是对前两部分的签名，防止数据被篡改。

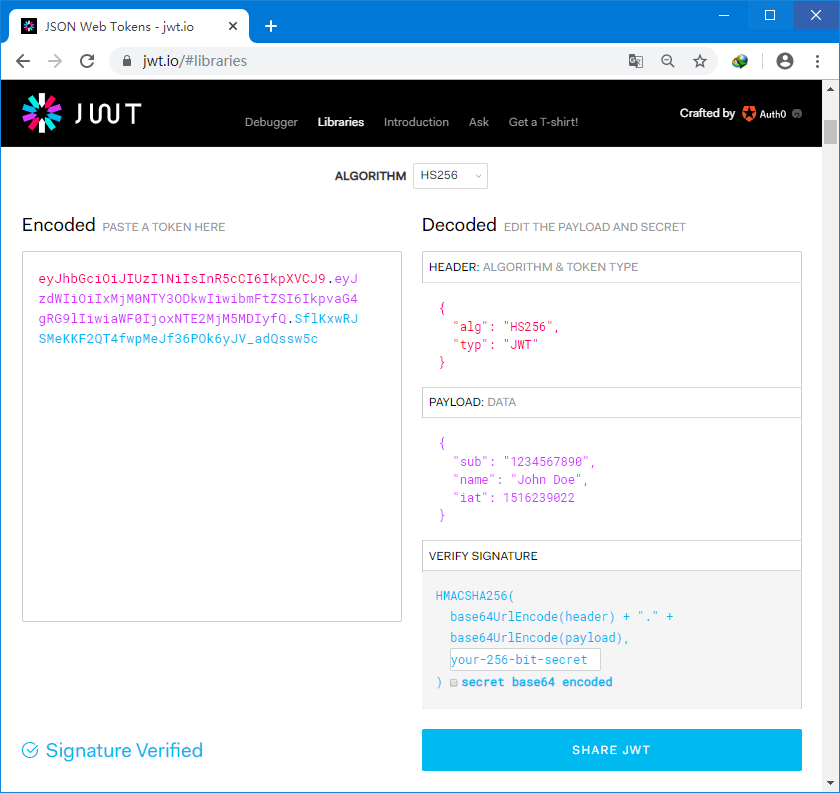
首先，需要指定一个密钥(secret)。这个密钥只有服务器才知道，不能泄露给用户。然后，使用Header里面指定的签名算法(默认是HMAC SHA256)，按照下面的公式产生签名：

|  |
| --- |
| HMACSHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  secret) |

算出签名以后，最后把Header、Payload、Signature三个部分拼成一个字符串，每个部分之间用点(.)分隔，就可以返回给客户端用户使用了。



你也可以使用官方的调试器([jwt.io Debugger](http://jwt.io))来解码、验证和生成JWT。



**4.Base64URL编码**

前面说过，Header和Payload需要编码成Base64URL。为什么编码成Base64URL而不是Base64呢？这是因为JWT作为一个令牌(token)，有时可能会放在URL(例如api.example.com/?token=xxx)中使用。而Base64有三个字符+、/和=，在URL里面有特殊含义，所以要被替换掉：=被省略、+替换成-，/替换成\_。这就是Base64URL算法。

因此Base64URL算法实质就是Base64算法，只是编码后进行了特殊处理，以便于JWT能在URL中使用。

**JWT的使用**

客户端收到服务器返回的JWT令牌，可以储存在Cookie里面，或者储存在localStorage中。此后，客户端每次与服务器通信，都带上这个JWT，把它放在Cookie里面自动发送给服务器，但是这样不能跨域。

所以更好的做法是放在HTTP请求的头信息Authorization字段里面，如下所示：

|  |
| --- |
| Authorization: Bearer <token> |

另一种方式是，跨域的时候，把JWT放在POST请求的数据体里面。如下所示：

|  |
| --- |
| <input type=hidden name=”Authorization” value=”<token>”/> |

最后一种方式就是放在URL中。如下所示：

|  |
| --- |
| https://api.example.com/?token=xxx |

**JWT的特点**

(1)JWT默认是不加密，但也是可以加密的，生成原始Token以后，可以用密钥再加密一次。

(2)JWT不加密的情况下，不要将秘密数据写入Payload部分。

(3)JWT不仅可以用于认证，也可以用于交换信息。有效使用JWT，可以降低服务器查询数据库的次数。

(4)JWT的最大缺点是，由于服务器不保存session状态，因此无法在使用过程中废止某个token，或者更改token的权限。也就是说，一旦JWT签发了，在到期之前就会始终有效，除非服务器部署额外的逻辑，例如使用Redis缓存服务器来缓存token并设置过期时间。

(5)JWT本身包含了认证信息，一旦泄露，任何人都可以获得该令牌的所有权限。为了减少盗用，JWT的有效期应该设置得比较短。对于一些比较重要的权限，使用时应该再次对用户进行认证。

(6)为了减少盗用，JWT不应该使用HTTP协议明码传输，要使用HTTPS协议传输。

**JWT的安全问题**

JWT的逻辑是先构造Header和Payload两部分，然后用一个Secret将前面两部分生成一个Signature用于验证。最后组合成JWT，如下所示：

|  |
| --- |
| eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9. eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9. TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ |

首先，用户是不知道的Secret(密钥)的，因为这保存在服务器，只有颁发者知道。

其次，Header和Payload两部分仅进行了Base64编码并没有加密，因此是可以解开的。

现在假定Header和Payload被解开了，并重新进行伪造：

|  |
| --- |
| eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkplYW4iLCJhZG1pbiI6dHJ1ZX0. （伪造部分）  TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ |

服务器在收到这个JWT以后，根据Secret(密钥)进行解码：

|  |
| --- |
| eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkplYW4iLCJhZG1pbiI6dHJ1ZX0.  zHUMniI-Kc0SU0zgAOsJ-lBfxFSHAHp0S0tGfQXqNss （解码出来的签名） |

比对Signature(签名):

TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ 不等于 zHUMniI-Kc0SU0zgAOsJ-lBfxFSHAHp0S0tGfQXqNss

告知客户端JWT是无效的Token，属于伪造。

显然客户端在不知晓Secret(密钥)的情况是伪造不了JWT令牌的，但是由于Header和Payload没有加密可以解开并查看信息，这两部分是不安全的，因此为了保证JWT不被破坏，在生成JWT以后，应该对其再次加密，这样客户端就解不开Header和Payload，且无论如何伪造都是徒劳。

**总结**

由于JWT被设计成无状态的，因此JWT主要用途是用来进行跨域认证，例如WebAPI接口调用，用来认证客户端用户是否已经登录。

如果想要用JWT来实现单点登录(Single Sign On，SSO)，这就让JWT由无状态变成了有状态了，改变了JWT的设计初衷和用途。由于JWT主要用来进行跨域认证，因此要实现单点登录，就不能用Session来存储状态信息了，而应该用Redis等缓存数据库来存储状态信息。用JWT令牌实现单点登录的逻辑如下：

1.用户发出登录请求。

2.服务器验证登录信息，重新生成JWT token，并颁发给用户。

3.服务器更新Redis中该用户的token为最新颁发的令牌，并设置过期时间。

这样该用户在其它的终端上就被挤压出去了，因为其它的终端上的该用户再用以前的token进行认证时，从Redis中已经找不到对应的token了。

由于颁发的JWT令牌可以永久存储，显然不可能允许用这个令牌无限制地进行登录认证，因此对于每个颁发的Token都必须设置一个过期时间，例如2小时。显然，除了跨域认证以外，还可以将JWT令牌用于软件的授权许可。