参考：<http://www.jianshu.com/p/576dbf44b2ae>

# 什么是JWT

Json web token (JWT), 是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准（[(RFC 7519](https://tools.ietf.org/html/rfc7519)).该token被设计为紧凑且安全的，特别适用于分布式站点的单点登录（SSO）场景。JWT的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息（me：即服务器和应用程序-或者用户之间传递信息的一种安全方式），以便于从资源服务器获取资源，也可以增加一些额外的其它业务逻辑所必须的声明信息，该token也可直接被用于认证，也可被加密。

## 起源

说起JWT，我们应该来谈一谈基于token的认证和传统的session认证的区别。

### 传统的session认证

我们知道，http协议本身是一种无状态的协议，而这就意味着如果用户向我们的应用提供了用户名和密码来进行用户认证，那么下一次请求时，用户还要再一次进行用户认证才行，因为根据http协议，我们并不能知道是哪个用户发出的请求，所以为了让我们的应用能识别是哪个用户发出的请求，我们只能在服务器存储一份用户登录的信息，这份登录信息会在响应时传递给浏览器，告诉其保存为cookie,以便下次请求时发送给我们的应用，这样我们的应用就能识别请求来自哪个用户了,这就是传统的基于session认证。

但是这种基于session的认证使应用本身很难得到扩展，随着不同客户端用户的增加，独立的服务器已无法承载更多的用户，而这时候基于session认证应用的问题就会暴露出来.

#### 基于session认证所显露的问题

**Session**: 每个用户经过我们的应用认证之后，我们的应用都要在服务端做一次记录，以方便用户下次请求的鉴别，通常而言session都是保存在内存中（me：基于性能的考虑，也可以存储在redis中，redis也是一个内存数据库），而随着认证用户的增多，服务端的开销会明显增大。

**扩展性**: 用户认证之后，服务端做认证记录，如果认证的记录被保存在内存中的话，这意味着用户下次请求还必须要请求在这台服务器上（me：内存不能共享导致不适用于多服务器的场景，其实可以将整个web的认证作为一个服务放在单独的服务器上这样就能保证多服务器上的登录问题）,这样才能拿到授权的资源，这样在分布式的应用上，相应的限制了负载均衡器（me：导致在不同的服务器上存储用户信息不能保证用户的单点登录）的能力。这也意味着限制了应用的扩展能力。

**CSRF**: 因为是基于cookie来进行用户识别的, cookie如果被截获，用户就会很容易受到跨站请求伪造的攻击。

## 基于token的鉴权机制

基于token的鉴权机制类似于http协议也是无状态的，它不需要在服务端去保留用户的认证信息或者会话信息。这就意味着基于token认证机制的应用不需要去考虑用户在哪一台服务器登录了，这就为应用的扩展提供了便利。

流程上是这样的：

* 用户使用用户名密码来请求服务器
* 服务器进行验证用户的信息
* 服务器通过验证发送给用户一个token
* 客户端存储token，并在每次请求时附送上这个token值
* 服务端验证token值，并返回数据

这个token必须要在每次请求时传递给服务端，它应该保存在请求头里， 另外，服务端要支持CORS(跨来源资源共享)策略，一般我们在服务端这么做就可以了Access-Control-Allow-Origin: \*。

那么我们现在回到JWT的主题上。

## JWT长什么样？

JWT是由三段信息构成的，将这三段信息文本用.链接一起就构成了Jwt字符串。就像这样:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

## JWT的构成

第一部分我们称它为头部（header),第二部分我们称其为载荷（payload, 类似于飞机上承载的物品)，第三部分是签证（signature).

三部分：header 、payload 和signature

### header

jwt的头部承载两部分信息：

* 声明类型，这里是jwt
* 声明加密的算法 通常直接使用 HMAC SHA256

完整的头部就像下面这样的JSON：

{

'typ': 'JWT',

'alg': 'HS256'

}

然后将头部进行base64加密（该加密是可以对称解密的),构成了第一部分.

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9

### playload

载荷就是存放有效信息的地方。这个名字像是特指飞机上承载的货品，这些有效信息包含三个部分

* 标准中注册的声明
* 公共的声明
* 私有的声明

**标准中注册的声明** (建议但不强制使用) ：

* **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,从而回避重放攻击。

**公共的声明** ：  
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.

**私有的声明** ：  
私有声明是提供者和消费者所共同定义的声明，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。

定义一个payload:

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

然后将其进行base64加密，得到Jwt的第二部分（me：payload）。

eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9

### signature

jwt的第三部分是一个签证信息，这个签证信息由三部分组成：

* header (base64后的)
* payload (base64后的)
* secret

这个部分需要base64加密后的header和base64加密后的payload使用.连接组成的字符串(me：一般使用英文句号.连接)，然后通过header中声明的加密方式进行加盐secret组合加密，然后就构成了jwt的第三部分(me:signature)。

// javascript

var encodedString = base64UrlEncode(header) + '.' + base64UrlEncode(payload);

var signature = HMACSHA256(encodedString, 'secret'); // TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

将这三部分用.连接成一个完整的字符串,构成了最终的jwt:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

**注意：secret是保存在服务器端的，jwt的签发生成也是在服务器端的，secret就是用来进行jwt的签发和jwt的验证，所以，它就是你服务端的私钥，在任何场景都不应该流露出去。一旦客户端得知这个secret, 那就意味着客户端是可以自我签发jwt了。**

### 如何应用

一般是在请求头里加入Authorization，并加上Bearer标注：

fetch('api/user/1', {

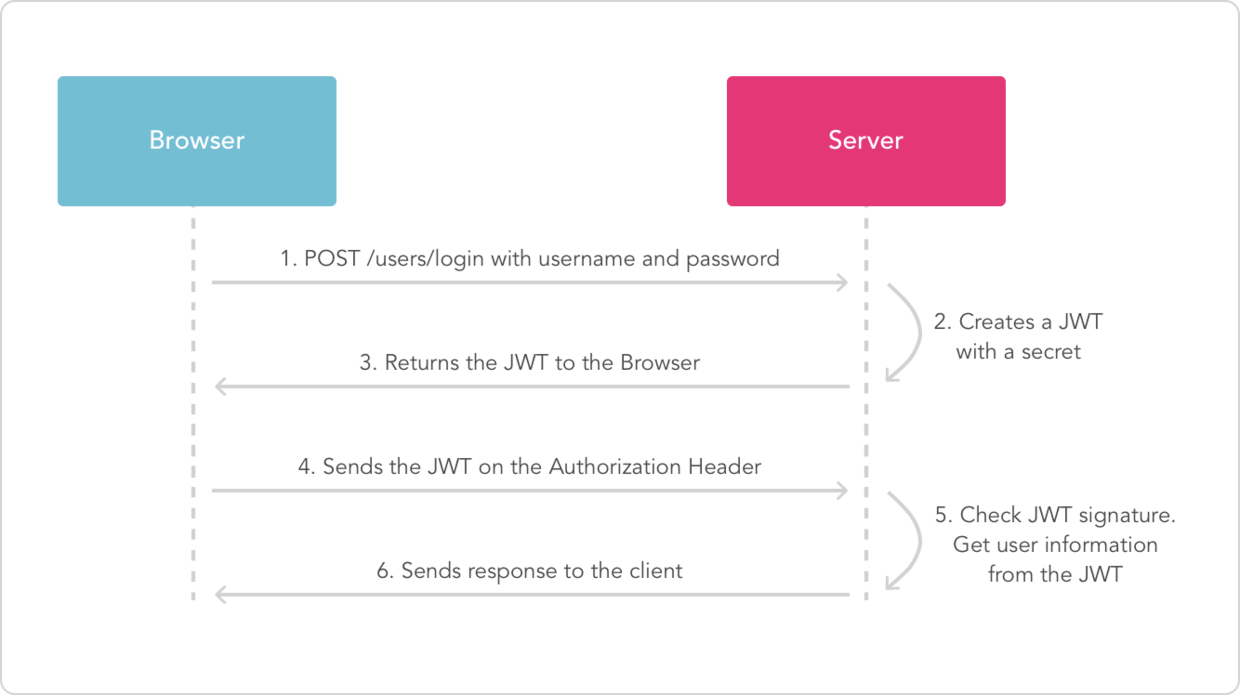
headers: {

'Authorization': 'Bearer ' + token

}

})

服务端会验证token，如果验证通过就会返回相应的资源。整个流程就是这样的:



jwt-diagram

## 总结

### 优点

* 因为json的通用性，所以JWT是可以进行跨语言支持的，像JAVA,JavaScript,NodeJS,PHP等很多语言都可以使用。
* 因为有了payload部分，所以JWT可以在自身存储一些其他业务逻辑所必要的非敏感信息。
* 便于传输，jwt的构成非常简单，字节占用很小，所以它是非常便于传输的。
* 它不需要在服务端保存会话信息, 所以它易于应用的扩展

### 安全相关

* 不应该在jwt的payload部分存放敏感信息，因为该部分是客户端可解密的部分。
* 保护好secret私钥，该私钥非常重要。
* 如果可以，请使用https协议