**Web安全通讯之Token与JWT**

**http://blog.csdn.net/wangcantian/article/details/74199762**

# Token

## 什么是token

Token是服务端生成的一串字符串，以作客户端进行请求的一个令牌，服务端根据令牌获取客户端的身份信息。   
举个栗子：

<http://www.example.com/demo?token=15qwc87wq336scwWFSC2sc1w>

## 为什么要用token

互联网时代信息安全验证放在首要的地位，对于敏感的信息（如账号密码等等）明文的出现次数越少越好。

我们都知道，HTTP协议是一种无状态的协议，这就意味着当我们向应用服务端提供了用户名和密码进行用户认真后，下次请求还是要再进行用户认证，而且服务端又不知道发起请求是谁。按照这个思维，假如每个请求都带有敏感信息，即使进行加密，但是这就增加暴露频率，并且服务端频繁对每个请求的身份信息进行数据查询验证，这是个很大的开销，显然不是我们想要的结果。

为了我们登录后让服务端“记住”我，下次发出请求服务端识别哪个用户发送的，token令牌能解决http无状态的问题，这时候你会觉得SESSION不也一样吗？别急，下面会说到。token就像我们的身份证，客户端一旦得到服务端响应的token后本地缓存，之后每次请求带上token就行了，重要的是开发者可以在token上自定义信息（如UUID），并且是加密的，服务端就减少数据查询验证身份的开销了。

## 与传统的SESSION有什么区别

如果您还不了解session，请先自行百度学习，这里我简单介绍下：

session 是一种HTTP存储机制，目的是为无状态的HTTP提供的持久机制。

* Session一般只提供一种简单的认证ID，即JSESSIONID，用户数据只保存在服务端上，因此JSESSIONID尤为重要需要严格保密，这导致了session的弊端：如果web服务器做了负载均衡，那么下一个操作请求到了另一台服务器的时候session会丢失。
* Session：用户数据保存在服务端；Token：客户端的 cookie 或本地介质存储。
* Session 更加适合在浏览器上应用，对于开发API接口 Token 是不错的选择。
* 一旦服务器挂了 sessionid 在服务端内存中映射信息丢失了， token 具有自带性，状态还存在。
* Token 安全性好，有签名能防止信息篡改、监听、重放攻击。
* Token 能进行授权操作，Session 不可以。
* Token 不再依赖于 Cookie，所以你就不需要考虑对 CSRF（跨站请求伪造）的防范。
* HMACSHA256 计算的 Token 验证和解析的费时比数据库查询 Session 多。

## token身份验证流程

* 客户端使用用户名和密码请求登录验证
* 服务端接受请求，进行身份验证
* 验证成功后，服务端会签发一个 Token，再把这个 Token 发送给客户端
* 客户端收到 Token 后把它存起来
* 客户端每次向服务端请求时候就带上 Token
* 服务端收到 Token 后，去验证 Token ，如果验证通过，执行业务逻辑

## 服务端中跟Token有关的问题

* 或许你们会想到，Token具有自带性，用户的在线状态不可能存在Token中，那怎么知道用户是否在线呢 ？   
  其实服务端中每个Token可以与UUID形成映射对存储在服务器上面，当用户注销操作时，删除UUID与Token的键值对，所以查询不到表示下线了
* 如何存查删 Token ？   
  我们可以把Token信息缓存在内存中，比存在数据库中的好处是读取速度快，开销小，坏处是一旦断点数据全没了，不过token重新认证操作获取就有一个了。基于缓存内存中，memcached、redis，KV方式很适合对token查询的需求。

# JWT(Json Web Token)官网地址：<https://jwt.io/>

## jwt github：<https://github.com/jwtk/jjwt>

## 什么是JWT(Json Web Token)

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。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，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。也就是说JWT是Token的一种表述性声明规范。

如果你不清楚JSON请自行学习

## JWT长什么样子

eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI3OHNhd2RmZjUiLCJzdWIiOiJ4aWFvdGlhbnRpYW4iLCJpYXQiOjE0OTgwMzE0NDIsImlzcyI6IjEyMi4xMTQuMjE0LjE0NyIsImV4cCI6MTQ5ODAzMjY0Mn0.0h\_kDhyZLhnt8TRgbLsOnVT8eOUAqgFTEZP-XgIGuA

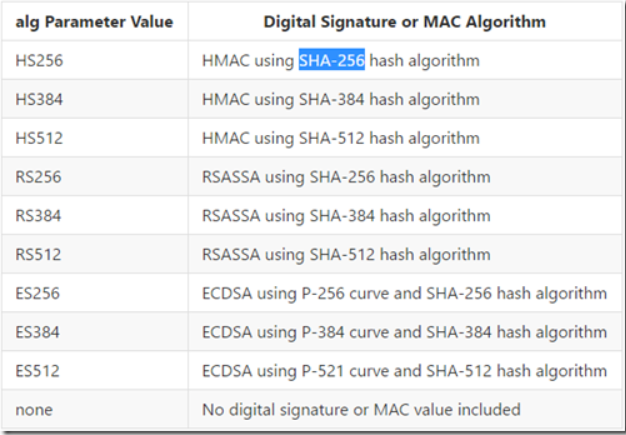
上面字符串都是用Base64编码后，发现结构类似：xxx.yyy.zzz

## JWT的结构

JWT包含了三个部分，分别用.分割开来，分别是：

* **Header 头部**
* **Payload 负载**
* **Signature 签名**
* **// 1.Header，包含JWT基础声明，加密算法与类别**
* {
* "alg": "HS256", // 加密算法
* "typ": "JWT" // 类别
* }
* **// 2.Payload，存放有效信息的地方**
* // 包含 Claim ，它可以一些实体（通常指的用户）的状态和额外的元数据，有三种类型
* // 2.1.Reserved claims JWT标准里面定好的claim，内容如下：
* // 2.2.Public claims
* // 2.3.Private claims
* // 建议的 Claims 不是强制使用的，完全可以按照自己的需求自定义playload，如果是自定义的claims名，您使用的实现库是不会主动去验证它们的
* {
* "aud": "uuu", // 接受者
* "iss": "xxx", // 签发者
* "exp": "1498499261660", // 过期时间
* "sub": "yyy", // 主题
* "iat": "1498459261660", // 签发时间
* "jti": "", // JWT的唯一身份标示
* "nbf": "1498459261660", // 定义在什么时间之前，该jwt都是不可用的
* ...
* }
* **// 3.Signature，签名，根据Header定义的算法和私钥组合加密**
* HMACSHA256(
* base64UrlEncode(header) + "." +
* base64UrlEncode(payload),
* secret)

## WT签发与验证流程

1. 服务端根据业务需求声明 Header 和 Playload
2. 将 Header 和 Playload 分别生成 Json 字符串
3. Header 和 Playload 分别进行base64编码，用 . 分隔开来，组成 JWT 的第一和第二部分，例如：   
   eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI3OHNhd2RmZjUiLCJzdWIiOiJ4aWFvdGlhbnRpYW4iLCJpYXQiOjE0OTgwMzE0NDIsImlzcyI6IjEyMi4xMTQuMjE0LjE0NyIsImV4cCI6MTQ5ODAzMjY0Mn0
4. 得到第3步生成的字符串，根据 Header 里面 alg 指定的签名算法生成出来形成 JWT 的 Signature 部分。算法不同，签名结果不同，常用的值以及对应的算法如下： 
5.  第4步生成的 Signature 组成 JWT 的第3部分，用 . 分隔组成完整的 JWT：   
   eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI3OHNhd2RmZjUiLCJzdWIiOiJ4aWFvdGlhbnRpYW4iLCJpYXQiOjE0OTgwMzE0NDIsImlzcyI6IjEyMi4xMTQuMjE0LjE0NyIsImV4cCI6MTQ5ODAzMjY0Mn0.0h\_kDhyZLhnt8TRgbLsOnVT8eOUAqgFTEZP-XgIGuA
6.  到这里服务端签发流程结束
7.  客户端得到 JWT 后存起来，每次请求带上 JWT 字符串
8.  服务端收到请求携带的 JWT ，开始进入验证流程
9.  对 JWT 的完整性进行验证，使用 base64 对 Header 进行解码，知道 JWT 使用什么签名
10.  重复第4步对 Header 和 Playload 再做一次签名
11.  比较这个签名是否与 JWT 本身携带的签名完全相同，只要不同，就可以认为该 JWT 是被篡改过的，验证失败，验证流程结束
12.  如果相同，使用 base64 对 Playload 进行解码，再进行业务逻辑处理，此时验证成功，验证结束。

## 注意

* Playload 不要存放敏感信息，因为该部分用base64编码，在客户端是可解的；
* 服务端保护好secre私钥，一旦客户端得带私匙就可以自己签发 JWT 了；
* 在网络层面上 token 明文传输的话会非常的危险，所以建议一定要使用 HTTPS ，并且把 token 放在 post body 里。PS：正在准备下一篇关于 HTTPS 的笔记。

# JWT——Token认证的两种实现和安全详解

### 前言：

最近因为项目中需要解决跨域取值的问题，所有考虑到用Token认证做技术支撑点，自己看了许多与之相关的文章，从中总结出了以下两个要点(签名和token时间)。在说这两个要点之前先大概简单说一下与之有关的一些问题。

首先，如果你对token认证的知识一点都不了解，那么我觉得这篇文章还不太适合你，因为我在这里不会在把相关的基础知识再说明一遍，因为网上有很多相关的文章，讲的都比较好，我会在文章下边参考文献中附上链接。但是还是说一下重点的几个点：

1.header(头部)，头部信息主要包括（参数的类型--JWT,签名的算法--HS256）

2.poyload(负荷)，负荷基本就是自己想要存放的信息(因为信息会暴露，不应该在载荷里面加入任何敏感的数据)，有两个形式，下边会讲到

3.sign(签名)，签名的作用就是为了防止恶意篡改数据，下边会详细说明

我的理解，在Java的实现中可以有两种方式，一种是不借助第三方jar，自己生成token，另一种的借助第三方jar，传入自己需要的负荷信息，生成token。接下来就根据这两个逐个说明。Token的组成就是header.poyload.sign。

## Token的优点

（1）相比于session，它无需保存在服务器，不占用服务器内存开销。   
（2）无状态、可拓展性强：比如有3台机器（A、B、C）组成服务器集群，若session存在机器A上，session只能保存在其中一台服务器，此时你便不能访问机器B、C，因为B、C上没有存放该Session，而使用token就能够验证用户请求合法性，并且我再加几台机器也没事，所以可拓展性好就是这个意思。   
（3）由（2）知，这样做可就支持了跨域访问。

# token和jwt存在什么区别

oken和jwt存在什么区别

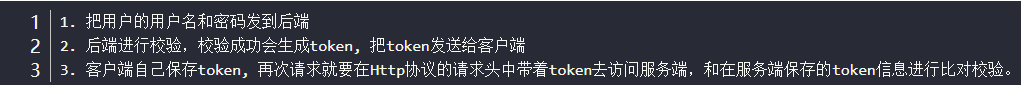
相同： 都是访问资源的令牌， 都可以记录用户信息，都是只有验证成功后

区别：

​ 服务端验证客户端发来的**token信息要进行数据的查询操作；JWT验证客户端发来的token信息就不用， 在服务端使用密钥校验就可以，不用数据库的查询。**

**Token 访问资源的令牌**

验证流程：



特点：

​ 客户端每次都要携带token, 客户端的内容比较多

**JWT**

概念: JSON WEB TOKEN 的简写。可以使用在RESTFUL接口定义， 也可以使用在普通的web

组成：  
**header、payload、签证**

验证流程：

1. 在头部信息中声明加密算法和常量， 然后把header使用json转化为字符串

2. 在载荷中声明用户信息，同时还有一些其他的内容；再次使用json 把载荷部分进行转化，转化为字符串

3. 使用在header中声明的加密算法和每个项目随机生成的secret来进行加密， 把第一步分字符串和第二部分的字符串进行加密， 生成新的字符串。词字符串是独一无二的。

4. 解密的时候，只要客户端带着JWT来发起请求，服务端就直接使用secret进行解密。

特点：

1. 三部分组成，每一部分都进行字符串的转化

2. 解密的时候没有使用数据库，仅仅使用的是secret进行解密。

3. JWT的secret千万不能泄密！！！

**JSON Web Token（JWT）是目前最流行的跨域身份验证解决方案。虫虫今天给大家介绍JWT的原理和用法。**

**1.跨域身份验证**

**Internet服务无法与用户身份验证分开。一般过程如下。**

1.用户向服务器发送用户名和密码。

2.验证服务器后，相关数据（如用户角色，登录时间等）将保存在当前会话中。

3.服务器向用户返回session\_id，session信息都会写入到用户的Cookie。

4.用户的每个后续请求都将通过在Cookie中取出session\_id传给服务器。

5.服务器收到session\_id并对比之前保存的数据，确认用户的身份。

这种模式最大的问题是，没有分布式架构，无法支持横向扩展。如果使用一个服务器，该模式完全没有问题。但是，如果它是服务器群集或面向服务的跨域体系结构的话，**则需要一个统一的session数据库库来保存会话数据实现共享，这样负载均衡下的每个服务器才可以正确的验证用户身份。**

例如虫虫举一个实际中常见的单点登陆的需求：站点A和站点B提供同一公司的相关服务。现在要求用户只需要登录其中一个网站，然后它就会自动登录到另一个网站。怎么做？

一种解决方案是**听过持久化session数据，写入数据库或文件持久层**等。收到请求后，验证服务从持久层请求数据。该解决方案的优点在于架构清晰，而缺点是架构修改比较费劲，整个服务的验证逻辑层都需要重写，工作量相对较大。而且由于依赖于持久层的数据库或者问题系统，会有单点风险，如果持久层失败，整个认证体系都会挂掉。