

## 彻底理解cookie, session, token

墨颜、 Java团长 2019-04-16



来源: [cnblogs.com/moyand/p/9047978.html](http://cnblogs.com/moyand/p/9047978.html)

### 发展史

1、很久很久以前，Web 基本上就是文档的浏览而已，既然是浏览，作为服务器，不需要记录谁在某一段时间里都浏览了什么文档，每次请求都是一个新的HTTP协议，就是请求加响应，尤其是我不用记住是谁刚刚发了HTTP请求，每个请求对我来说都是全新的。这段时间很嗨皮。

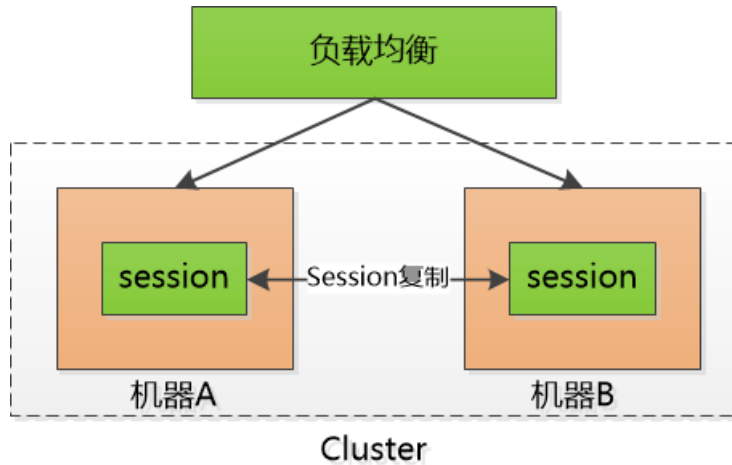
2、但是随着交互式Web应用的兴起，像在线购物网站，需要登录的网站等等，马上就面临一个问题，那就是要管理会话，必须记住哪些人登录系统，哪些人往自己的购物车中放商品，也就是说我必须把每个人区分开，这就是一个不小的挑战，因为HTTP请求是无状态的，所以想出的办法就是给大家发一个会话标识(session id), 说白了就是一个随机的字符串，每个人收到的都不一样，每次大家向我发起HTTP请求的时候，把这个字符串给一并捎过来，这样我就能区分开是谁了

3、这样大家很嗨皮了，可是服务器就不嗨皮了，每个人只需要保存自己的session id，而服务器要保存所有人的session id！如果访问服务器多了，就得由成千上万，甚至几十万个。

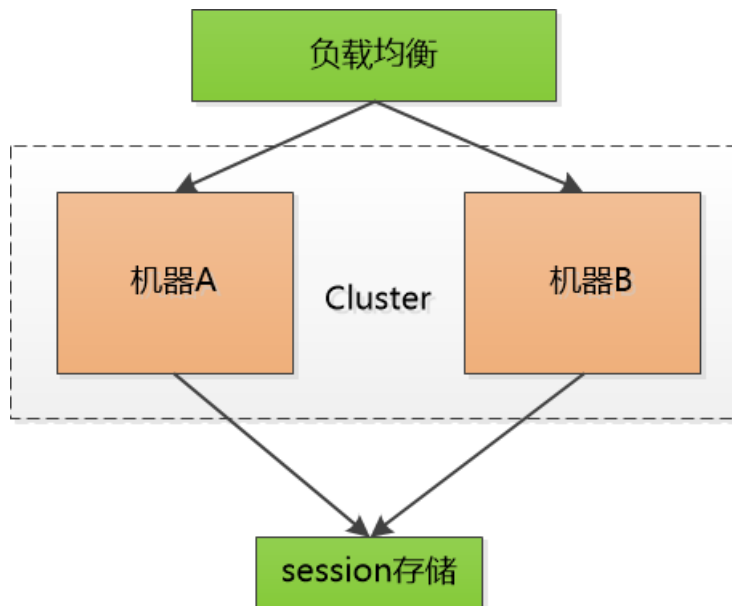
这对服务器说是一个巨大的开销，严重的限制了服务器扩展能力，比如说我用两个机器组成了一个集群，小F通过机器A登录了系统，那session id会保存在机器A上，假设小F的下一次请求被转发到机器B怎么办？机器B可没有小F的 session id啊。

有时候会采用一点小伎俩：**session sticky**，就是让小F的请求一直粘连在机器A上，但是这也不管用，要是机器A挂掉了，还得转到机器B去。

那只好做**session**的复制了，把**session id**在两个机器之间搬来搬去，快累死了。



后来有个叫Memcached的支了招：把**session id**集中存储到一个地方，所有的机器都来访问这个地方的数据，这样一来，就不用复制了，但是增加了单点失败的可能性，要是那个负责**session**的机器挂了，所有人都得重新登录一遍，估计得被人骂死。



也尝试把这个单点的机器也搞出集群，增加可靠性，但不管如何，这小小的**session**对我来说是一个沉重的负担

4、于是有人就一直在思考，我为什么要保存这可恶的**session**呢，只让每个客户端去保存该多好？

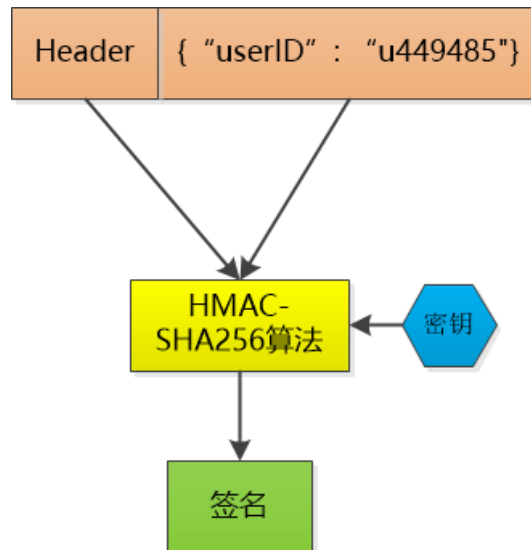
可是如果不保存这些 **session id** , 怎么验证客户端发给我的 **session id** 的确是我生成的呢? 如果不去验证, 我们都不知道他们是不是合法登录的用户, 那些不怀好意的家伙们就可以伪造 **session id** , 为所欲为了。

嗯, 对了, 关键点就是验证!

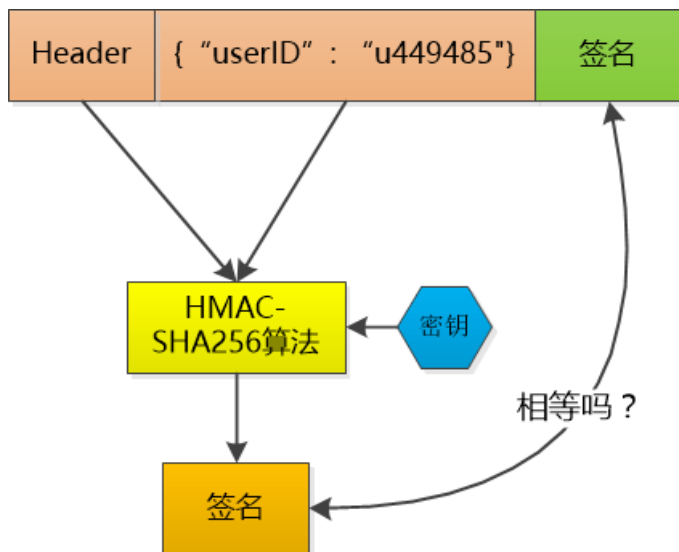
比如说, 小F已经登录了系统, 我给他发一个令牌(token), 里边包含了小F的 **user id**, 下一次小F再次通过Http 请求访问我的时候, 把这个token 通过Http header 带过来不就可以了。

不过这和**session id**没有本质区别啊, 任何人都可以可以伪造, 所以我得想点儿办法, 让别人伪造不了。

那就对数据做一个签名吧, 比如说我用HMAC-SHA256 算法, 加上一个只有我才知道的密钥, 对数据做一个签名, 把这个签名和数据一起作为token, 由于密钥别人不知道, 就无法伪造token了。



这个token 我不保存, 当小F把这个token 给我发过来的时候, 我再用同样的HMAC-SHA256 算法和同样的密钥, 对数据再计算一次签名, 和token 中的签名做个比较, 如果相同, 我就知道小F已经登录过了, 并且可以直接取到小F的**user id** , 如果不相同, 数据部分肯定被人篡改过, 我就告诉发送者: 对不起, 没有认证。



Token 中的数据是明文保存的（虽然我会用Base64做下编码，但那不是加密），还是可以被别人看到的，所以我不能在其中保存像密码这样的敏感信息。

当然，如果一个人的token被别人偷走了，那我也没办法，我也会认为小偷就是合法用户，这其实和一个人的session id被别人偷走是一样的。

这样一来，我就不保存session id了，我只是生成token，然后验证token，我用我的CPU计算时间获取了我的session存储空间！

解除了session id这个负担，可以说是无事一身轻，我的机器集群现在可以轻松地做水平扩展，用户访问量增大，直接加机器就行。这种无状态的感觉实在是太好了！

## Cookie

cookie 是一个非常具体的东西，指的就是浏览器里面能永久存储的一种数据，仅仅是浏览器实现的一种数据存储功能。

cookie由服务器生成，发送给浏览器，浏览器把cookie以kv形式保存到某个目录下的文本文件内，下一次请求同一网站时会把该cookie发送给服务器。由于cookie是存在客户端上的，所以浏览器加入了一些限制确保cookie不会被恶意使用，同时不会占据太多磁盘空间，所以每个域的cookie数量是有限的。

## Session

session 从字面上讲，就是会话。这个就类似于你和一个人交谈，你怎么知道当前和你交谈的是张三而不是李四呢？对方肯定有某种特征（长相等）表明他就是张三。

session 也是类似的道理，服务器要知道当前发请求给自己的是谁。为了做这种区分，服务器就要给每个客户端分配不同的“身份标识”，然后客户端每次向服务器发请求的时候，都带上这个“身份标识”，服务器就知道这个请求来自于谁了。至于客户端怎么保存这个“身份标识”，可以有很多种方式，对于浏览器客户端，大家都默认采用 cookie 的方式。

服务器使用session把用户的信息临时保存在了服务器上，用户离开网站后session会被销毁。这种用户信息存储方式相对cookie来说更安全，可是session有一个缺陷：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。

## Token

在Web领域基于Token的身份验证随处可见。在大多数使用Web API的互联网公司中，tokens 是多用户下处理认证的最佳方式。

以下几点特性会让你在程序中使用基于Token的身份验证

1. 无状态、可扩展
2. 支持移动设备
3. 跨程序调用
4. 安全

那些使用基于Token的身份验证的大佬们

大部分你见到过的API和Web应用都使用tokens。例如Facebook, Twitter, Google+, GitHub等。

## Token的起源

在介绍基于Token的身份验证的原理与优势之前，不妨先看看之前的认证都是怎么做的。

### 基于服务器的验证

我们都是知道HTTP协议是无状态的，这种无状态意味着程序需要验证每一次请求，从而辨别客户端的身份。

在这之前，程序都是通过在服务端存储的登录信息来辨别请求的。这种方式一般都是通过存储Session来完成。

随着Web，应用程序，已经移动端的兴起，这种验证的方式逐渐暴露出了问题。尤其是在可扩展性方面。

### 基于服务器验证方式暴露的一些问题

1. **Session:** 每次认证用户发起请求时，服务器需要去创建一个记录来存储信息。当越来越多的用户发请求时，内存的开销也会不断增加。
2. **可扩展性:** 在服务端的内存中使用Session存储登录信息，伴随而来的是可扩展性问题。
3. **CORS(跨域资源共享):** 当我们需要让数据跨多台移动设备上使用时，跨域资源的共享会是一个让人头疼的问题。在使用Ajax抓取另一个域的资源，就可以会出现禁止请求的情况。
4. **CSRF(跨站请求伪造):** 用户在访问银行网站时，他们很容易受到跨站请求伪造的攻击，并且能够被利用其访问其他的网站。

在这些问题中，可扩展性是最突出的。因此我们有必要去寻求一种更有行之有效的方法。

### 基于Token的验证原理

基于Token的身份验证是无状态的，我们不将用户信息存在服务器或Session中。

这种概念解决了在服务端存储信息时的许多问题

NoSession意味着你的程序可以根据需要去增减机器，而不用去担心用户是否登录。

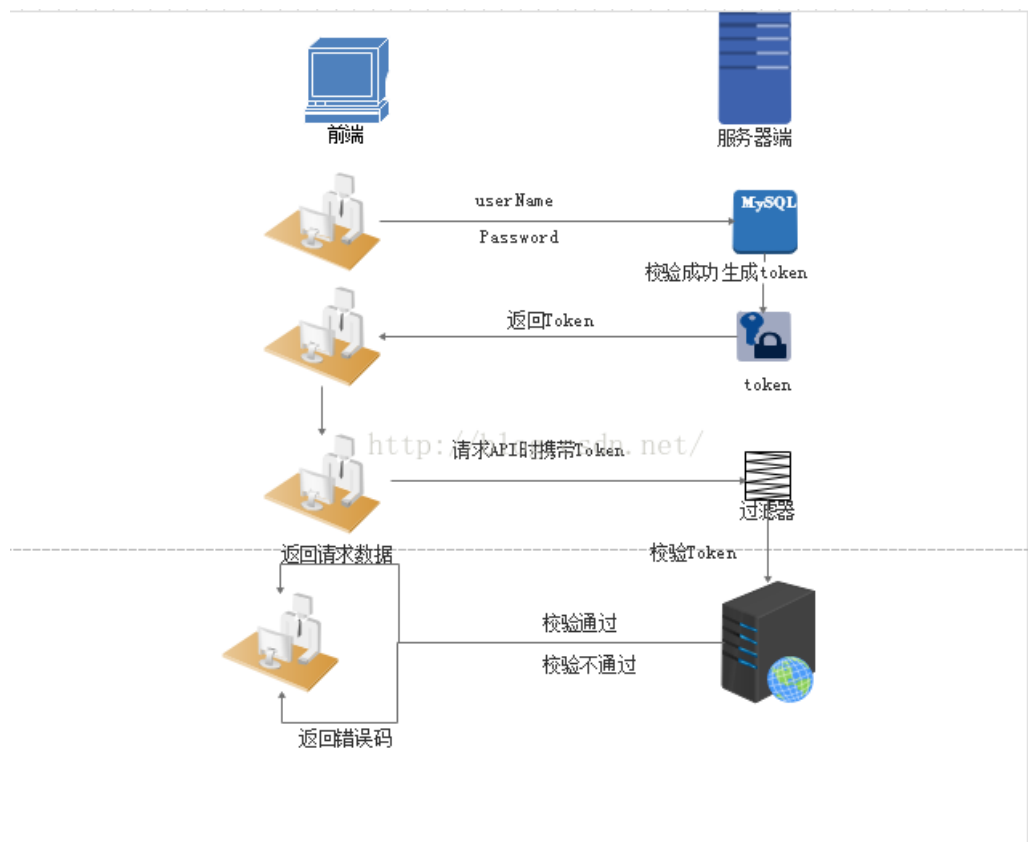
基于Token的身份验证的过程如下：

1. 用户通过用户名和密码发送请求。
2. 程序验证。
3. 程序返回一个签名的token给客户端。
4. 客户端储存token,并且每次用于每次发送请求。
5. 服务端验证token并返回数据。

每一次请求都需要token。token应该在HTTP的头部发送从而保证了Http请求无状态。我们同样通过设置服务器属性Access-Control-Allow-Origin:\*，让服务器能接受来自所有域的请求。

需要主要的是，在ACAO头部标明(designating)\*时，不得带有像HTTP认证，客户端SSL证书和cookies的证书。

实现思路：



1. 用户登录校验，校验成功后就返回Token给客户端。
2. 客户端收到数据后保存在客户端



3. 客户端每次访问API是携带Token到服务器端。
4. 服务器端采用filter过滤器校验。校验成功则返回请求数据，校验失败则返回错误码

当我们在程序中认证了信息并取得token之后，我们便能通过这个Token做许多的事情。

我们甚至能基于创建一个基于权限的token传给第三方应用程序，这些第三程序能够获取到我们的数据（当然只有在我们允许的特定的token）

## Tokens的优势

### 无状态、可扩展

在客户端存储的Tokens是无状态的，并且能够被扩展。基于这种无状态和不存储Session信息，负载均衡器能够将用户信息从一个服务传到其他服务器上。

如果我们将已验证的用户的信息保存在Session中，则每次请求都需要用户向已验证的服务器发送验证信息(称为Session亲和性)。用户量大时，可能会造成一些拥堵。

但是不要着急。使用tokens之后这些问题都迎刃而解，因为tokens自己hold住了用户的验证信息。

### 安全性

请求中发送token而不再是发送cookie能够防止CSRF(跨站请求伪造)。即使在客户端使用cookie存储token，cookie也仅仅是一个存储机制而不是用于认证。不将信息存储在Session中，让我们少了对session操作。

token是有时效的，一段时间之后用户需要重新验证。我们也不一定需要等到token自动失效，token有撤回的操作，通过token revocataion可以使一个特定的token或是一组有相同认证的token无效。

### 可扩展性

Tokens能够创建与其它程序共享权限的程序。例如，能将一个随便的社交帐号和自己的大号(Fackbook或是Twitter)联系起来。当通过服务登录Twitter(我们将这个过程Buffer)时，我们可以将这些Buffer附到Twitter的数据流上(we are allowing Buffer to post to our Twitter stream)。

使用tokens时，可以提供可选的权限给第三方应用程序。当用户想让另一个应用程序访问它们的数据，我们可以通过建立自己的API，得出特殊权限的tokens。

### 多平台跨域

我们提前先来谈论一下CORS(跨域资源共享)，对应用程序和服务进行扩展的时候，需要介入各种各种的设备和应用程序。

Having our API just serve data, we can also make the design choice to serve assets from a CDN. This eliminates the issues that CORS brings up after we set a quick header configuration for our application.

只要用户有一个通过了验证的token，数据和资源就能够在任何域上被请求到。

```
Access-Control-Allow-Origin: *
```

基于标准创建token的时候，你可以设定一些选项。我们在后续的文章中会进行更加详尽的描述，但是标准的用法会在JSON Web Tokens体现。

最近的程序和文档是供给JSON Web Tokens的。它支持众多的语言。这意味在未来的使用中你可以真正的转换你的认证机制。

(完)



## 本公众号粉丝 专属超级福利

看完下面资料若2个月内工资没有翻倍，我  
立马赔给你**1000元**！



web前端全集



Java全集



Python全集



node





数据库



angular.js

jQuery

jQuery.js



Vue.js



Recat.js



c++

C#

C#



PHP

JD 京东  
.COM

京东项目实战

阿里巴巴

阿里项目实战

腾讯网  
qq.com

腾讯项目实战

MI

小米项目实战

这才是2019年最新最全的资料！

共**2000G**！

仅限前**10**名免费拿！

原价**5999**元

3小时后恢复原价！

长按下面微信二维码，速度找我免费拿！





注意！添加时记得备注：

**大神，求全套资料**

没有备注，微信不通过的！

## Java团长

专注于Java干货分享



扫描上方二维码获取更多Java干货

吾爱破解论坛