# 发展史

## session id

起初Web应用主要是内容浏览，服务器不需要记录用户的浏览行为，每次请求都是一个新的等同的HTTP协议已，就是请求-相应的过程。

随着交互式网站（比如购物网站）的兴起，需要解决管理会话的问题，即必须记住哪些用户登录系统，哪些用户设置了购物车。这是个不小的挑战：因为HTTP协议请求本身是无状态的，所以需要给用户发一个会话表示（session id），即一个随机的字符串，每个用户都会收到一个随机且各不相同的字符串，每次用户向服务器发起HTTP请求的时候，就把这个随机的字符串一起带过去，这样服务器就能通感知到不同的用户了。

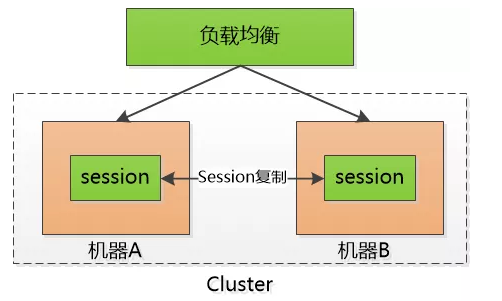
上述方案解决了HTTP无状态时，服务器如何识别不同客户端的问题。但是，对于服务器而言需要保存所有客户的session id，如果访问服务器的用户太多，那么对于服务器本身的性能是个不小的挑战，且会严重限制服务器的拓展能力（比如，客户J通过机器A登录了系统，对应客户J的session id保存在机器A上，那么客户J下一次请求被转发到机器B上怎么办？此时，机器B上是没有保存客户J的session id的）。

此时，对于服务器却存在问题了。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。

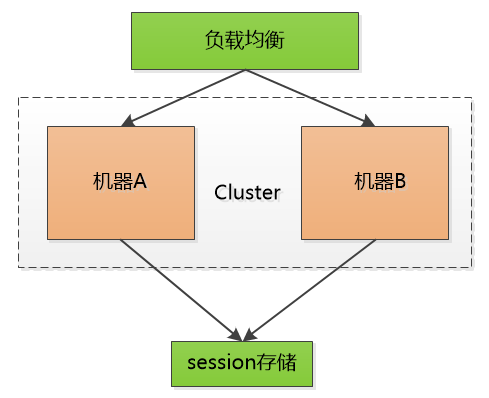
## session sticky

针对上述问题，可以采用session sticky的方案：让客户J的请求一直粘连在机器A上，但是这样机器A宕机了，还是需要转移到机器B上。

## session复制

还可以采用session 复制，在不同机器之间进行会话的复制。

## 集中存储session

 内存数据存储memcached提供了一种解决方案：把session id集中存储在一个地方，所有的机器都统一到这里获取数据。这样就不需要进行会话的复制了，但是增加了单点失败的可能（如果负责session的机器宕机了，那么所有的人都必须重新登录一遍）。

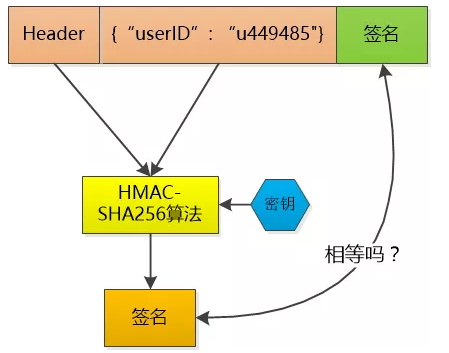
当然，可以把单点的session机器做成集群，增加可靠性，但是这样就相对复杂了。

## 令牌token

会话session的保存是个棘手的问题，如果只让客户端自行保存就好了，但是服务器还需要对session进行验证，所以关键在于验证上。

比如说，客户J已经登录了系统，服务器给客户发一个令牌(token)，包含客户J的 user id，下一次客户J再次通过HTTP请求访问服务器时，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，任何人都可以伪造，需要对数据做一个签名。比如说用HMAC-SHA256算法，加上一个只有服务器端才知道的密钥，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。

该token服务器不保存，当客户J把这个token 发送给服务器时，服务器再用同样的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，服务器就知道客户J已经登录过了，并且可以直接取到客户J的user id , 如果不相同，数据部分肯定被人篡改过，服务器就告诉发送者：没有认证。

Token中的数据是明文保存的（虽然会用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，那也没办法，服务器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，服务器无需保存session id，只是生成token，然后验证token（即用服务器CPU计算时间换取了服务器session会话存储空间，以时间换空间）。解除了session的负担，集群可以轻松实现水平拓展，用户访问量增大，直接增加服务器即可。

# 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

无状态、可扩展

支持移动设备

跨程序调用

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。

## 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**

Seesion：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。

可扩展性：在服务端的内存中使用Seesion存储登录信息，伴随而来的是可扩展性问题。

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。

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，可扩展行是最突出的。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:

用户通过用户名和密码发送请求。

程序验证。

程序返回一个签名的token 给客户端。

客户端储存token,并且每次用于每次发送请求。

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:\* ，让服务器能接受到来自所有域的请求。

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：

用户登录校验，校验成功后就返回Token给客户端。

客户端收到数据后保存在客户端

客户端每次访问API是携带Token到服务器端。

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，这些第三方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）

## 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。