Cookie是浏览器用来****保存用户信息的文件****，可以保存比如用户是谁，购物车有哪些商品等。

Session是一次会话，会话是指我们****访问网站的一个周期****。

* 比如用户打开一个浏览器访问某个位的站点。
* 在这个站点点击多个超链接查看各个网页，然后关闭浏览器，整个过程称之为一个会话。

token是服务器返回的一个****临时签名数据****, 可以使用这个签名数据表面用户身份.

为什么会有这三个东西呢?  都是一个目的, 服务器需要知道和自己通话的人是谁, 专业一点就是 服务器需要用某种机制来识别具体的用户.

这要从HTTP协议开始说起, ****HTTP协议是无状态的协议。一旦数据交换完毕，客户端与服务器端的连接就会关闭，再次交换数据需要建立新的连接。这就意味着服务器无法从连接上跟踪会话, 自然无法识别用户, 所以诞生了Cookie,session和token****。



早期的时候, 我们访问浏览器的时候，浏览器会发送一个HTTP请求到服务器端；

服务器会发送一个HTTP响应到客户端，其中包括Set-Cookie，意思就是浏览器建立一个cookie保存服务器指定的内容，比如用户信息和用户操作信息；

浏览器保存好信息之后，下次我们再次访问网站的时候，浏览器再发送HTTP请求到服务器端时都会携带之前保存的cookie；

服务器端会从收到的cookie中识别用户身份，就能让页面为你提供专门属于你的内容了。

cookie 可以让服务端跟踪每个客户端的访问，但是每次客户端的访问都必须传回这些 Cookie，如果 Cookie 很多，这无形地增加了客户端与服务端的数据传输量，

cookie具有以下特点

* cookie存储的数量和字符数量都有限制，只能存储几十个，不超4096左右个字符。
* ****Cookie具有不可跨域名性****

　　很多网站都会使用Cookie。例如，Google会向客户端颁发Cookie，Baidu也会向客户端颁发Cookie。那浏览器访问Google会不会也携带上Baidu颁发的Cookie呢？或者Google能不能修改Baidu颁发的Cookie呢？

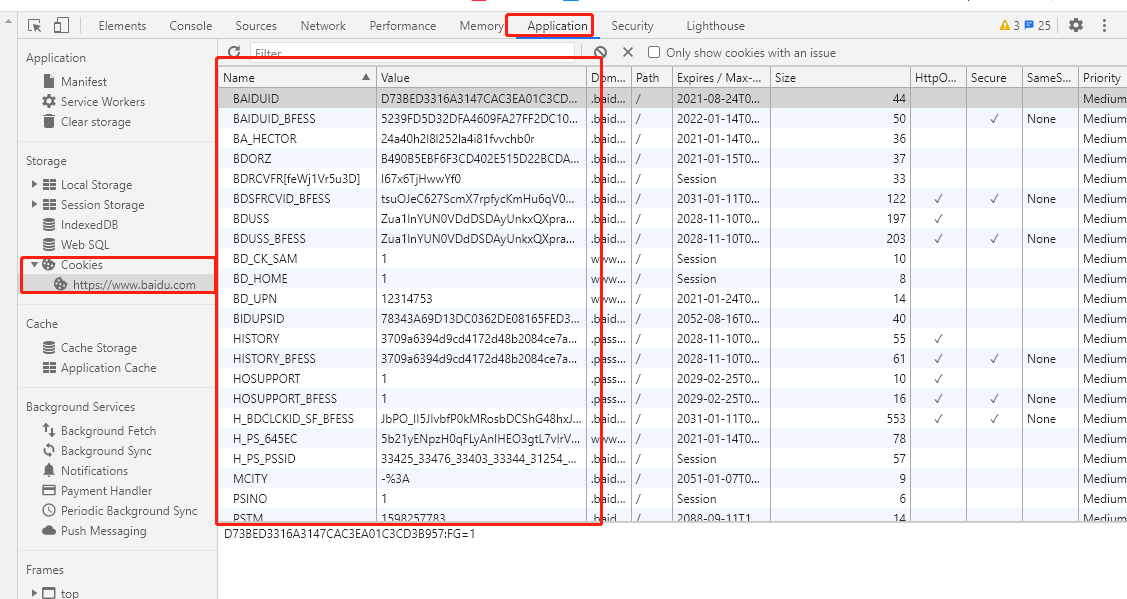
　　答案是否定的。****Cookie具有不可跨域名性****。根据Cookie规范，浏览器访问Google只会携带Google的Cookie，而不会携带Baidu的Cookie。Google也只能操作Google的Cookie，而不能操作Baidu的Cookie。

　　Cookie在客户端是由浏览器来管理的。浏览器能够保证Google只会操作Google的Cookie而不会操作Baidu的Cookie，从而保证用户的隐私安全。浏览器判断一个网站是否能操作另一个网站Cookie的依据是域名。Google与Baidu的域名不一样，因此Google不能操作Baidu的Cookie。

需要注意的是，虽然网站images.google.com与网站www.google.com同属于Google，但是域名不一样，二者同样不能互相操作彼此的Cookie。

* cookie虽然不可跨域名, 但是存储在用户浏览器里也是不安全的，任何人都能直接查看。

我们以谷歌浏览器为例查看cookie，我们右击找到检查，点击打开浏览器控制台，或者按F12



session在计算机网络应用中被称为“会话控制”。不同的是Cookie保存在客户端浏览器中，而****Session保存在服务器上****。在服务端保存Session的方法很多，内存、数据库、文件都有。

1. 客户端浏览器访问网站的时候,

2. 服务器会向客户浏览器发送一个每个用户特有的会话编号sessionID，让浏览器写入到cookie里(大多数情况)。服务器同时也把sessionID和对应的用户信息、用户操作记录在服务器上，这些记录就是session。

3. 客户端浏览器再次访问时，会发送cookie给服务器，cookie中就包含sessionID。

4. 服务器从cookie里找到sessionID，再根据sessionID找到以前记录的用户信息就可以知道他是谁, 之前操控哪些、访问过哪里。

从流程上看cookie和session看起来就是一套东西啊,  其实是不一样的, cookie是文件, 可以存任意东西, sessionID只不过是存的一种数据, 而session数据保存在服务器, 只不过session常见实现方式是借助cookie,  但是可以不用cookie, 使用URL地址重写来实现session。

随着Web，应用程序，以及移动端的兴起，session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：在服务端使用Seesion存储登录信息，伴随而来的是可扩展性问题, 多个服务器之间如何同步sessionID。



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，可扩展性是最突出的。

比如说服务端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sessionID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 sessionID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sessionID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 对服务端来说是一个沉重的负担.

因此有必要去寻求一种更有行之有效的方法。于是有人就一直在思考， 我为什么要保存这可恶的sessionID呢， 只让每个客户端去保存该多好？

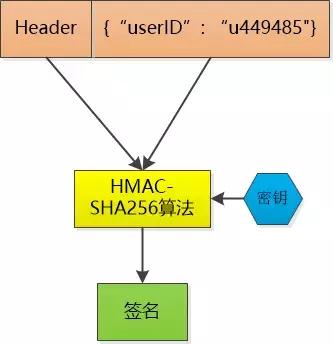
可是如果不保存这些sessionID , 怎么验证客户端发给我的sessionID 的确是服务端生成的呢？ 如果不去验证，我们都不知道他们是不是合法登录的用户， 那些不怀好意的家伙们就可以伪造sessionID , 为所欲为了。

嗯，对了，关键点就是验证 ！

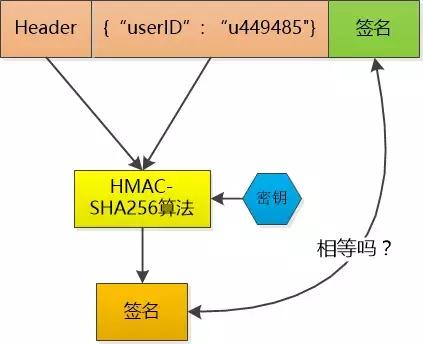
比如说， 比如说， 小F已经登录了系统， 我给他发一个令牌(token)， 里边包含了小F的 user id， 下一次小F 再次通过Http 请求访问我的时候， 把这个token 通过Http header 带过来不就可以了。

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， 那服务器也没办法， 服务器也会认为小偷就是合法用户， 这其实和一个人的sessionID 被别人偷走是一样的。

这样一来， 服务器就不保存sessionID 了，只是生成token , 然后验证token ， 用CPU计算时间获取了我的session 存储空间 ！

解除了sessionID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！

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，token是多用户下处理认证的最佳方式。

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。

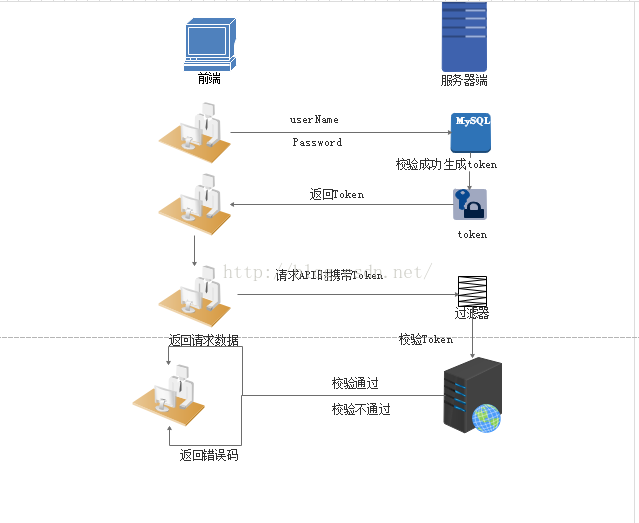
基于Token的身份验证的过程如下:

用户通过用户名和密码发送请求。

服务端验证,  返回生成的token 给客户端,  同时给数据库和Redis里关联token和用户信息。

客户端储存token,并且其后的每一次请求都添加token, token应该在HTTP的头部发送从而保证了Http请求无状态。

服务端查询Redis+数据库, 验证token并返回数据。



****Tokens的优势****

* 无状态

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。

* 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。

* 支持多平台跨服务器

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平台(Android,ios, h5)任何服务器上被请求到。

****最后做个三者的比较:****

****cookie :****

1. cookie由服务器生成，保存在客户端浏览器。

2. 容易被劫持，不安全****，****别人可以分析存放在本地的COOKIE并进行COOKIE欺骗。

3. cookie可以被用户禁止

4. 容量小, 单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

****session****

1. session是由应用服务器维持的一个服务器端的存储空间, 没有对存储的数据量的限制，可以保存更为复杂的数据类型.

2. session 默认被存在在服务器的一个文件里, 但是实际中可以放在 文件、数据库、或内存中都可以。

3. 当用户量增多时，会对服务器造成较大压力。

4. Session的实现方式大多数情况用Cookie保存的，但是也可以使用URL地址重写。

5. 较安全,用户验证这种场合一般会用 session, 比如金融银行类的产品,

****token****

1.无状态、可扩展

2.支持移动设备

3.跨服务器调用

4.安全