链滴

四种会话追踪技术

作者：JellyfishMIX

原文链接：https://ld246.com/article/1601479603185

来源网站：链滴

四种会话追踪技术

- Cookie
- Session
- URL重写
- 隐藏表单域

Session和Cookie

- session用来表示用户会话，session对象在服务端维护，一般tomcat设定session生命周期为30分，超时将失效，也可以主动设置无效。
- cookie存放在客户端，可以分为内存cookie和磁盘cookie。内存cookie在浏览器关闭后消失，磁盘ookie超时后消失。当浏览器发送请求时，将自动发送对应cookie信息，前提是请求url满足cookie路。
- 可以将sessionId存放在cookie中，也可以通过重写url将sessionId拼接到url上。因此可以查看浏器cookie或地址栏url看到sessionId。
- 请求到服务端时，将根据请求中的sessionId查找session，如果可以获取到则返回，否则返回null者返回新构建的session，老的session依旧存在，请参考API。

URL重写

- 可以通过url参数的形式将信息发送至服务器。
- 但是这种方式参数的大小受到浏览器限制，cookie禁用时可以继续的工作，不存在持久性，一旦页关闭则结束。
- 这种方式通过明文将信息传输，并不安全，容易被劫持。

隐藏表单域

表单隐藏域type=hidden的作用：

HTML中写表单的时候，写入这段代码

```
<input type="hidden" name="#" value="#">
```

意思是在这里增加一个隐藏域。对于用户来说，在页面上隐藏域是不可见的。

- 隐藏域的作用是帮助表单收集和发送信息，便于后端处理数据。用户点击提交数据的时候，隐藏域信息也被一起发送到了后端。
- 后端接收前端传来的数据，需要确认前端的身份，是本公司的网页传来的数据，而不是其他黑客知后端地址后传来的假数据。那么就加一个隐藏域，验证value里的值和数据库里name的值是不是对应，类似于“天王盖地虎，宝塔镇河妖”，暗号对的上，才能证明是自己人。当然这些东西也能用cooki实现，但使用隐藏域比较简单，而且不会有浏览器不支持，用户禁用cookie的烦恼。
- 有时候一个表单里有多个提交按钮，后端怎么知道用户是点击哪个按钮提交过来的呢？这时候我们要加隐藏域，对每一个按钮起个名字(value值)，后端接收到数据后，检查value值，就能知道是哪个钮提交的了。
- 有时候一个网页中有多个form，我们知道多个form是不能同时提交的，但有时这些form确实相互用，我们就可以在form中添加隐藏域来使它们联系起来。

● JavaScript不支持全局变量，但有时我们必须用全局变量，我们就可以把值先存在隐藏域里，它的就不会丢失了。

● 还有个例子，比如按一个按钮弹出四个小窗口，当点击其中的一个小窗口时其他三个自动关闭．可是E不支持小窗口相互调用，所以只有在父窗口写个隐藏域，当小窗口看到那个隐藏域的值是close时就己关掉。

例题

1.

有关会话跟踪技术描述正确的是?

● A Cookie是Web服务器发送给客户端的一小段信息，客户端请求时，可以读取该信息发送到服务端。

● B 关闭浏览器意味着临时会话ID丢失，但所有与原会话关联的会话数据仍保留在服务器上，直至会过期。

● C 在禁用Cookie时可以使用URL重写技术跟踪会话。

● D 表单隐藏域将字段添加到HTML表单并在客户端浏览器中显示。

答案

A, B, C

解析

● A 参考本二级标题知识点

● B session存在服务器，但是sessionid存在客户端，作为客户端找到session的引用。

● C 可以通过重写url将sessionId拼接到url上。

● D 表单隐藏域用来收集或发送信息的不可见元素，对于网页的访问者用户来说，表单隐藏域是看不的。当表单被提交时，表单隐藏域会将其中定义的名称和值发送到服务器上。

jsessionid

jsessionid 是 session 的标识。这就好比每个人都有身份证一样。 (jsessionid 只是 tomcat 中对 session id 的叫法，在其它容器里面，不一定就是叫 jsessionid 了)。

Q&A

1. 是不是只要一打开一个页面就会产生一个jsessionid?

2. 在不关闭浏览器的情况下，什么时候jsessionid会改变？我登陆后，登陆然后退出，jsessionid会什么变化?

3. session 和 jsessionid 有什么关系?

所谓session可以这样理解：当与服务端进行会话时，比如说登陆成功后，服务端会为用户开壁一块存区间，用以存放用户这次会话的一些内容，比如说用户名之类的。那么就需要一个东西来标志这个

存区间是你的而不是别人的，这个东西就是 session id（jsessionid 只是 tomcat 中对 session id 的法，在其它容器里面，不一定就是叫 jsessionid 了)。而这个内存区间可以理解为 session。

然后，服务器会将这个session id发回给你的浏览器，放入你的浏览器的cookies中（这个cookies是存cookies，跟一般的不一样，它会随着浏览器的关闭而消失）。

之后，只有你浏览器没有关闭，你每向服务器发请求，服务器就会从你发送过来的cookies中拿出这个ession id，然后根据这个session id到相应的内存中取你之前存放的数据。

但是，如果你退出登陆了，服务器会清掉属于你的内存区域，所以你再登的话，会产生一个新的sessin了。

jsessionid 解释的解释如下：

这是一个保险措施 因为Session默认是需要Cookie支持的，但有些客户浏览器是关闭Cookie的。而jsesionid是存储在Cookie中的，如果禁用Cookie的话，也就是说服务器那边得不到jsessionid，这样也没法根据jsessionid获得对应的session了，获得不了session就得不到session中存储的数据了。

这个时候就需要在URL中指定服务器上的session标识,也就是类似于"jsessionid=5F4771183629C984F8382E23BE13C4C"这种格式。

用一个方法(忘了方法的名字)处理URL串就可以得到这个东西，这个方法会判断你的浏览器是否开启了ookie,如果他认为应该加他就会加上去。

Q：是不是只要一打开一个页面就会产生一个jsessionid?

A：显然不是的。session是有一定作用域的，而且是有时间限制的。

Q：在不关闭浏览器的情况下，什么时候jsessionid会改变？我登陆后,登陆然后退出,jsessionid会有么变化?

A：jsessionid是服务器那边生成的，因为cookie是服务器那边送到客户端的信息。不管能不能修改jsesionid，都不应该修改，如果你修改了，这就失去了jessionid的自身意义了，你修改的话，你让服务那边如何找到对应的session？找不到的话，你存放在那个session中的数据不是取不到了吗?

登陆然后退出，我认为会重新生成一个jsessionid。因为退出的话，application作用域的数据都会丢，更何况这个比它作用域还小的session？既然session都消失了，这个jsessionid有什么用?

Q：session和jsessionid有什么关系?

A：jsessionid是session的标识。这就好比每个人都有身份证一样。

cookie和session机制区别与联系

具体来说cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状的方案。同时我们也看到，由于采用服务器端保持状态的方案在客户端也需要保存一个标识，所以sesson机制可能需要借助于cookie机制来达到保存标识的目的，但实际上它还有其他选择。

cookie机制

正统的cookie分发是通过扩展HTTP协议来实现的，服务器通过在HTTP的响应头中加上一行特殊的指以提示浏览器按照指示生成相应的cookie。然而纯粹的客户端脚本如JavaScript或者VBScript也可以成cookie。而cookie的使用是由浏览器按照一定的原则在后台自动发送给服务器的。浏览器检查所有储的cookie，如果某个cookie所声明的作用范围大于等于将要请求的资源所在的位置，则把该cooki

附在请求资源的HTTP请求头上发送给服务器。

cookie的内容主要包括：名字，值，过期时间，路径和域。路径与域一起构成cookie的作用范围。若设置过期时间，则表示这个cookie的生命期为浏览器会话期间，关闭浏览器窗口，cookie就消失。这生命期为浏览器会话期的cookie被称为会话cookie。会话cookie一般不存储在硬盘上而是保存在内里，当然这种行为并不是规范规定的。若设置了过期时间，浏览器就会把cookie保存到硬盘上，关闭再次打开浏览器，这些cookie仍然有效直到超过设定的过期时间。存储在硬盘上的cookie可以在不同浏览器进程间共享，比如两个IE窗口。而对于保存在内存里的cookie，不同的浏览器有不同的处理方。

session机制

session机制是一种服务器端的机制，服务器使用一种类似于散列表的结构（也可能就是使用散列表来保存信息。

当程序需要为某个客户端的请求创建一个session时，服务器首先检查这个客户端的请求里是否已包了一个session标识（称为session id），如果已包含则说明以前已经为此客户端创建过session，服器就按照session id把这个session检索出来使用（检索不到，会新建一个），如果客户端请求不包含session id，则为此客户端创建一个session并且生成一个与此session相关联的session id，session id值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个session id将被在本次响应中回给客户端保存。

保存这个session id的方式可以采用cookie，这样在交互过程中浏览器可以自动的按照规则把这个标发挥给服务器。一般这个cookie的名字都是类似于SEEESIONID。但cookie可以被人为的禁止，则必有其他机制以便在cookie被禁止时仍然能够把session id传递回服务器。

经常被使用的一种技术叫做URL重写，就是把session id直接附加在URL路径的后面。还有一种技术做表单隐藏字段。就是服务器会自动修改表单，添加一个隐藏字段，以便在表单提交时能够把sessiond传递回服务器。

Cookie和Token

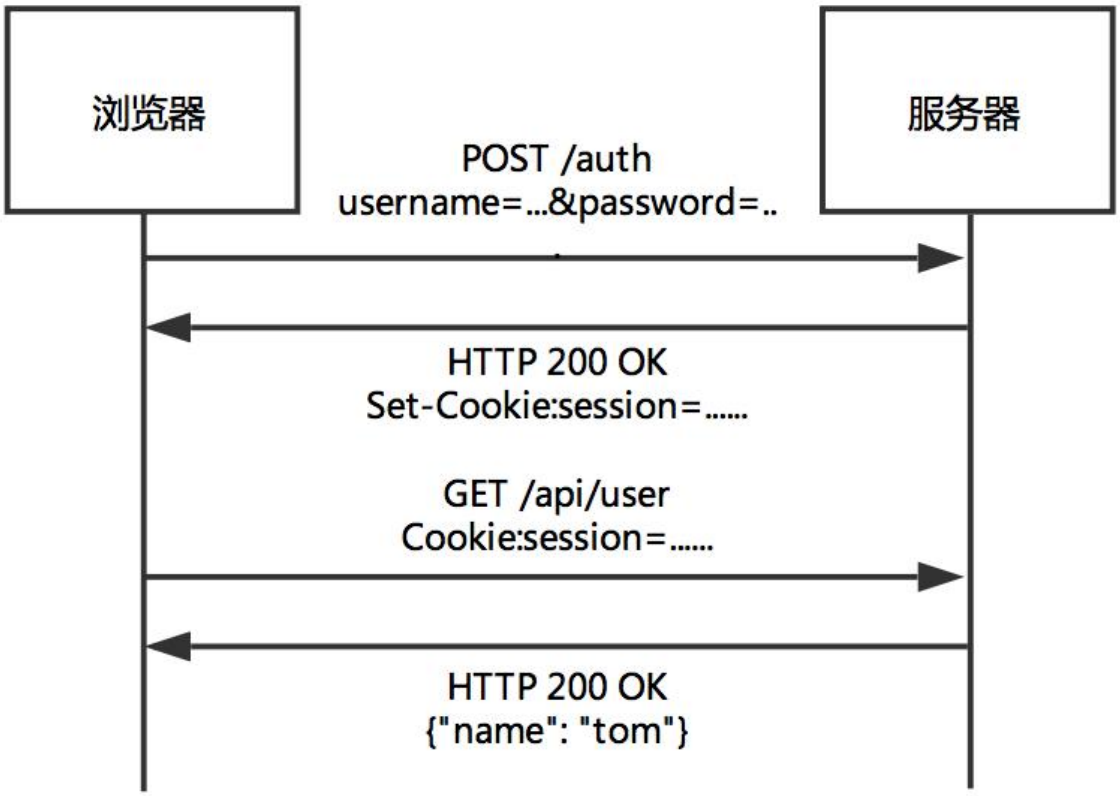
概述

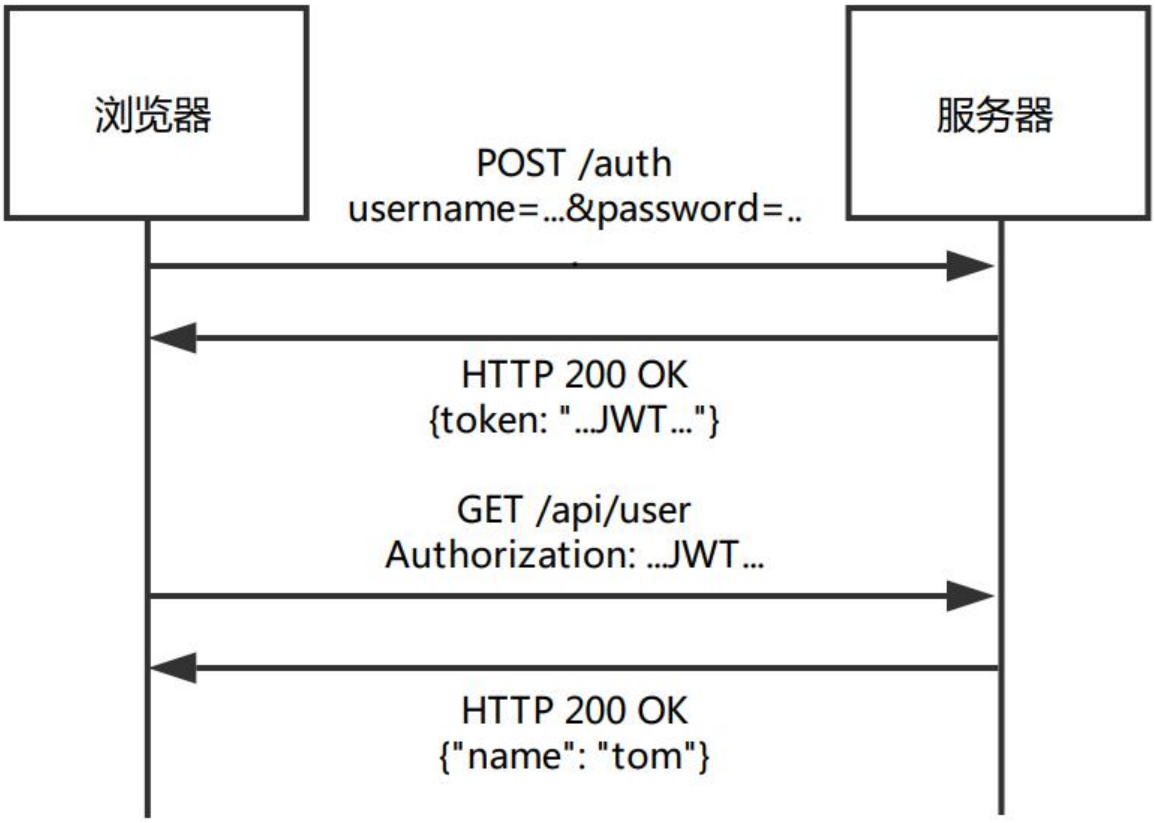
HTTP是一个“无状态”协议，这意味着Web应用程序服务器在响应客户端请求时不会将多个请求链到任何一个客户端。然而，许多Web应用程序的安全和正常运行都取决于系统能够区分用户并识别用及其权限。

这就需要一些机制来为一个HTTP请求提供状态。它们使站点能够在会话期间对各用户做出适当的响，从而保持跟踪用户在应用程序中的活动（请求和响应）。

cookie和token

下面两图大致展示了基于cookie和基于token工作流程。





原文链接：四种会话追踪技术

基于cookie的身份验证

cookie是源自站点并由浏览器存储在客户计算机上的简单文件。它们通常包含一个名称和一个值，用将客户端标识为对站点具有特定许可权的特定用户。

cookie与源域相连接的方式可以确保仅源域能够访问其中存储的信息。第三方服务器既不能读取也不更改用户计算机上该域的cookie内容。

网景公司的前雇员于1993年发明了cookie。

基于cookie的验证是有状态的，就是说验证或者会话信息必须同时在客户端和服务端保存。这个信息务端一般在数据库中记录，而前端会保存在cookie中。

验证的一般流程如下：

1. 用户输入登陆凭据；
2. 服务器验证凭据是否正确，并创建会话，然后把会话数据存储在数据库中；
3. 具有会话id的cookie被放置在用户浏览器中；
4. 在后续请求中，服务器会根据数据库验证会话id，如果验证通过，则继续处理；
5. 一旦用户登出，服务端和客户端同时销毁该会话。

基于token的身份验证

随着单页面应用程序的流行，以及Web API和物联网的兴起，基于token的身份机制越来越被大家广采用。

当讨论基于token的身份验证时，一般都是说的JSON Web Tokens（JWT）。虽然有着很多不同的式实现token，但是JWT已经成为了事实上的标准，所以后面会将JWT和token混用。

基于token的验证是无状态的。服务器不记录哪些用户已登陆或者已经发布了哪些JWT。对服务器的个请求都需要带上验证请求的token。该标记既可以加在header中，可以在POST请求的主体中发送也可以作为查询参数发送。

工作流程如下：

1. 用户输入登陆凭据；
2. 服务器验证凭据是否正确，然后返回一个经过签名的token；
3. 客户端负责存储token，可以存在local storage，或者cookie中；
4. 对服务器的请求带上这个token；
5. 服务器对JWT进行解码，如果token有效，则处理该请求；
6. 一旦用户登出，客户端销毁token。

token相对cookie的优势

无状态

基于token的验证是无状态的，这也许是它相对cookie来说最大的优点。后端服务不需要记录token每个令牌都是独立的，包括检查其有效性所需的所有数据，并通过声明传达用户信息。

服务器唯一的工作就是在成功的登陆请求上签署token，并验证传入的token是否有效。

防跨站请求伪造（CSRF）

举个CSRF攻击的例子，在网页中有这样的一个链接

，假设你已经通过银行的验证并且cookie中存在验信息，同时银行网站没有CSRF保护。一旦用户点了这个图片，就很有可能从银行向tom这个人转100块钱。

但是如果银行网站使用了token作为验证手段，攻击者将无法通过上面的链接转走你的钱。（因为攻者无法获取正确的token）

多站点使用

cookie绑定到单个域。foo.com域产生的cookie无法被bar.com域读取。使用token就没有这样的问。这对于需要向多个服务获取授权的单页面应用程序尤其有用。

使用token，使得用从myapp.com获取的授权向myservice1.com和myservice2.com获取服务成为能。

支持移动平台

好的API可以同时支持浏览器，iOS和Android等移动平台。然而，在移动平台上，cookie是不被支持。

性能

一次网络往返时间（通过数据库查询session信息）总比做一次HMACSHA256计算的Token验证和解要费时得多。

JWT

JWT是JSON Web Token的缩写。它定义了一种紧凑且独立的方式，用于将各方之间的信息安全地传为JSON对象。这是一个开放的标准，见RFC 7519。

基于JWT的信息可以通过数字签名进行校验。校验的方法即可以使用消息摘要（HMAC），或者非对加密（RSA）。

JWT具有两个特点：

1. 紧凑。由于其较小的尺寸，JWT可以通过URL，POST参数或者HTTP头发送。较小的尺寸会带来传速度的优势；

2. 自包含：token中包含了用户的所有必须信息，避免了多次查询数据库的需要。

应用场景

以下是JWT有用的一些场景

1. 验证：这是JWT最常用的场景。一旦用户登陆成功，每个后续的请求将包括JWT，服务器在对JWT行验证后，允许用户访问服务和资源。单点登陆是一个广泛使用JWT的场景，因为它的开销相对较小

并且能够在不同的域中轻松使用。

2. 信息交换：JWT是在可以安全地传输信息。因为JWT可以被签名，收信人可以确认发信人的身份，时也能够验证内容是否被篡改。

格式

JWT包括三个部分：头部、载荷和签名，这三个部分通过 .连接起来。

因此，一个典型的JWT长这样 xxxxx.yyyyy.zzzzz。

头部

头部通常包括两部分：token类型（JWT），和使用到的算法，如HMAC、SHA256或RSA，下面是个例子，说明这是一个JWT，使用的签名算法是HS256。

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

头部会通过 Base64Url 编码形成JWT的第一部分。

载荷

第二部分是载荷，要传递出去的声明，其中包含了实体（通常是用户）和附加元数据。有三种类型的明：

- 保留声明：这是一组预定义的声明，非强制性，用来帮助接收方（服务器）更好地理解这个JWT。中包括：iss（issuer，该JWT的签发者），exp（expiration time，过期时间），sub（subject，该WT所面向的用户），aud（audience，JWT的接收者），和另外一些声明
- 公共声明：这些可以用使用JWT的人随意定义。但是为了避免冲突，应在在IANA JSON WEB令牌册表中定义它们，或者将其定义为包含防冲突命名空间的URI。
- 私有声明：这些是为了在同意使用它们的各方之间共享信息而创建的自定义声明。

下面是一个例子

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

载荷会通过**Base64Url**编码形成JWT的第二部分

签名

将上面两部分编码后，使用 .连接在一起，形成了xxxxx.yyyyyy。

最后，采用头部指定的算法，和私钥对上面的字符串进行签名。

加入采用的是HMAC SHA256 算法，签名将通过下面的方式生成

```
HMACSHA256(base64UrlEncode(header) + "." +base64UrlEncode(payload),secret)
```

该签名用户验证JWT发送者的身份，并确保该消息没有被篡改。

JWT工作流程

在身份验证过程中，一旦用户使用其凭据成功登陆，服务器将返回JWT，该JWT必须在客户端本地保。这和服务器创建会话并返回cookie的传统方法不同。

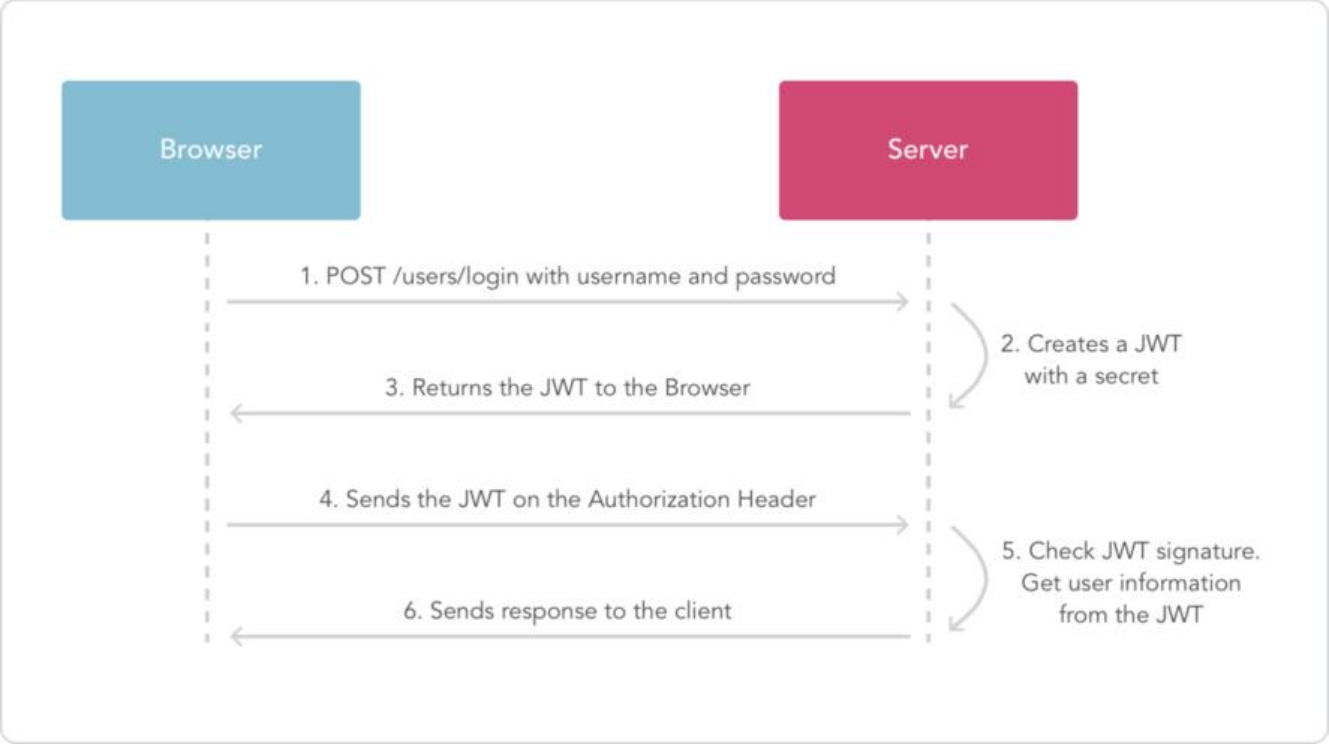
每次用户要请求受保护的资源时，必须在请求中带上JWT。通常在 Authorization头Bearer 中，如下：

```
Authorization: Bearer <token>
```

这是一种无状态的认证机制，因为用户状态永远不会保存在服务器内存中。服务器的受保护路由将在权头中检查有效的JWT，如果存在，则允许用户访问受保护的资源。由于JWT是自说明的，包含了所必要的信息，这就减少了多次查询数据库的需要。

这样可以完全依赖无状态的数据API，甚至可以向下游服务发出请求。API的作用域并不重要，因此跨资源共享（CORS）不会是一个问题，因为它不使用Cookie。

整个流程如下图：



使用JWT的理由

现在来谈谈JWT与简单网页令牌（SWT）和安全断言标记语言令牌（SAML）相比的优势。

由于JSON比XML更短小，编码时其大小也较小，使得JWT比SAML更紧凑。这使得JWT成为在HTML HTTP环境中能更快地传递。

从安全角度来说，SWT只能通过使用HMAC算法的共享密钥进行对称签名。但是，JWT和SAML令牌

以以X.509证书的形式使用公钥/私钥对进行签名。与简单的JSON签名相比，使用XML数字签名签名ML而不引入模糊的安全漏洞是非常困难的。

JSON解析器在大多数编程语言中很常见，因为它们直接映射到对象。相反，XML没有自然的文档对象映射。这使得使用JWT比SAML断言更容易。

从使用平台来说，JWT在Internet规模上使用。这突出了客户端处理多个平台上特别是移动平台上的JON Web令牌的便利性。

引用/参考

"牛客294719号"的回答 - 牛客

表单隐藏域type=hidden的作用 - 辉夜ㄟ - 简书

"半纸流年-轻描了谁的夏天"的回答 - 牛客

为什么会有jsessionid，这个东东有什么用呢？ - masanpaossa - OSCHINA

Cookie和Token - 大蟒传奇 - 简书