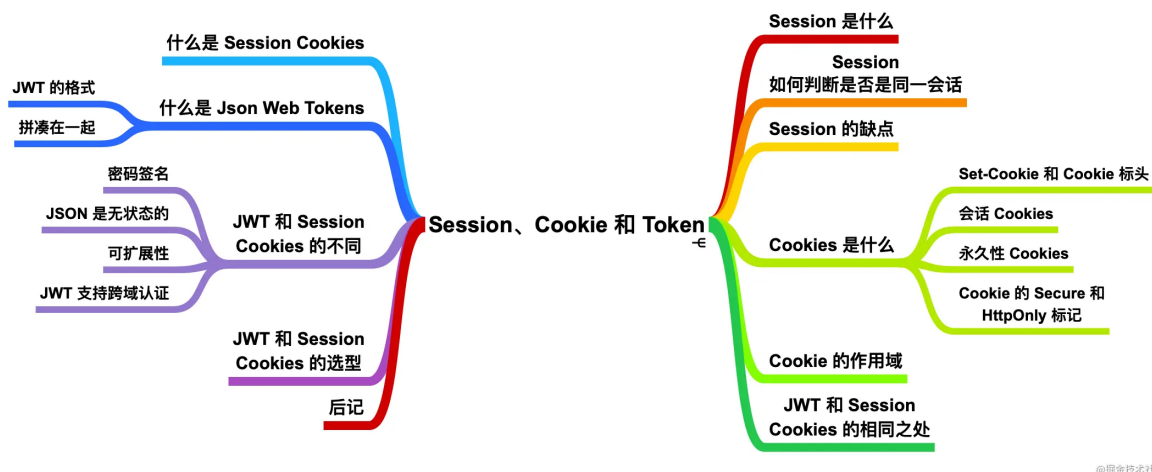


# 看完这篇 Session、Cookie、Token，和面试官扯皮就没问题了



@掘金技术社区

## Cookie 和 Session

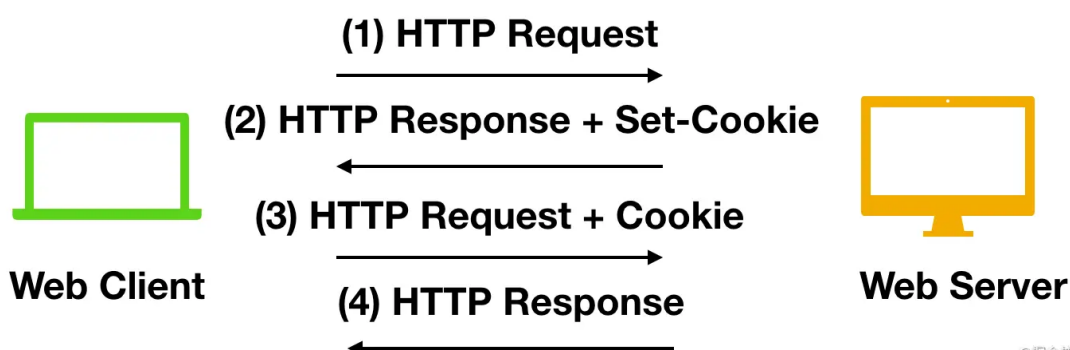
HTTP 协议是一种 **无状态协议**，即每次服务端接收到客户端的请求时，都是一个全新的请求，服务器并不知道客户端的历史请求记录；Session 和 Cookie 的主要目的就是为了解决 HTTP 的无状态特性。

### Session 是什么

客户端请求服务端，服务端会为这次请求开辟一块 **内存空间**，这个对象便是 Session 对象，存储结构为 **ConcurrentHashMap**。Session 弥补了 HTTP 无状态特性，服务器可以利用 Session 存储客户端在同一个会话期间的一些操作记录。

### Session 如何判断是否是同一会话

服务器第一次接收到请求时，开辟了一块 Session 空间（创建了 Session 对象），同时生成一个 **sessionId**，并通过响应头的 **Set-Cookie: JSESSIONID=XXXXXXX** 命令，向客户端发送要求设置 Cookie 的响应；客户端收到响应后，在本机客户端设置了一个 **JSESSIONID=XXXXXXX** 的 Cookie 信息，该 Cookie 的过期时间为浏览器会话结束；



@掘金技术社区

接下来客户端每次向同一个网站发送请求时，请求头都会带上该 Cookie 信息（包含 sessionId），然后，服务器通过读取请求头中的 Cookie 信息，获取名称为 JSESSIONID 的值，得到此次请求的 sessionId。

## Session 的缺点

Session 机制有个缺点，比如 A 服务器存储了 Session，就是做了负载均衡后，假如一段时间内 A 的访问量激增，会转发到 B 进行访问，但是 B 服务器并没有存储 A 的 Session，会导致 Session 的失效。

## Cookies 是什么



HTTP 协议中的 Cookie 包括 `web cookie` 和 `浏览器 cookie`，它是服务器发送到 Web 浏览器的一小块数据。服务器发送到浏览器的 Cookie，浏览器会进行存储，并与下一个请求一起发送到服务器。通常，它用于判断两个请求是否来自于同一个浏览器，例如用户保持登录状态。

HTTP Cookie 机制是 HTTP 协议无状态的一种补充和改良

Cookie 主要用于下面三个目的

- `会话管理`

登陆、购物车、游戏得分或者服务器应该记住的其他内容

- `个性化`

用户偏好、主题或者其他设置

- `追踪`

记录和分析用户行为

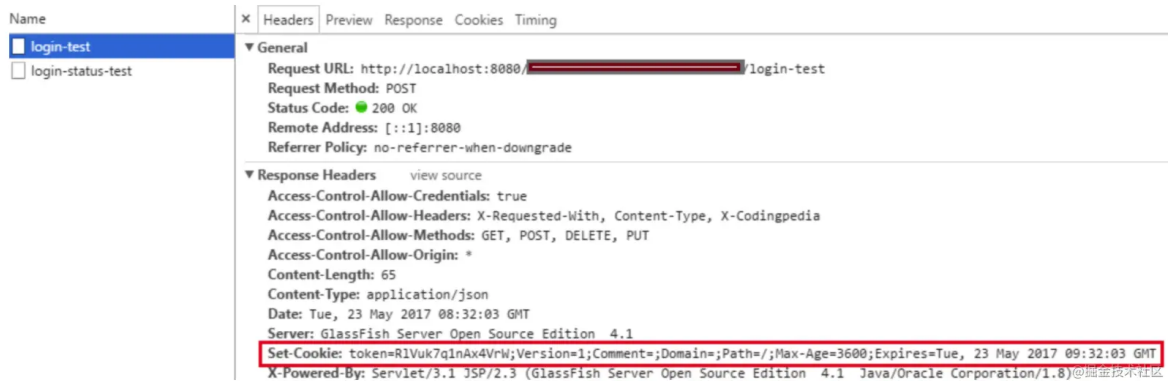
Cookie 曾经用于一般的客户端存储。虽然这是合法的，因为它们是在客户端上存储数据的唯一方法，但如今建议使用现代存储 API。Cookie 随每个请求一起发送，因此它们可能会降低性能（尤其是对于移动数据连接而言）。

# 创建 Cookie

当接收到客户端发出的 HTTP 请求时，服务器可以发送带有响应的 `Set-Cookie` 标头，Cookie 通常由浏览器存储，然后将 Cookie 与 HTTP 标头一同向服务器发出请求。

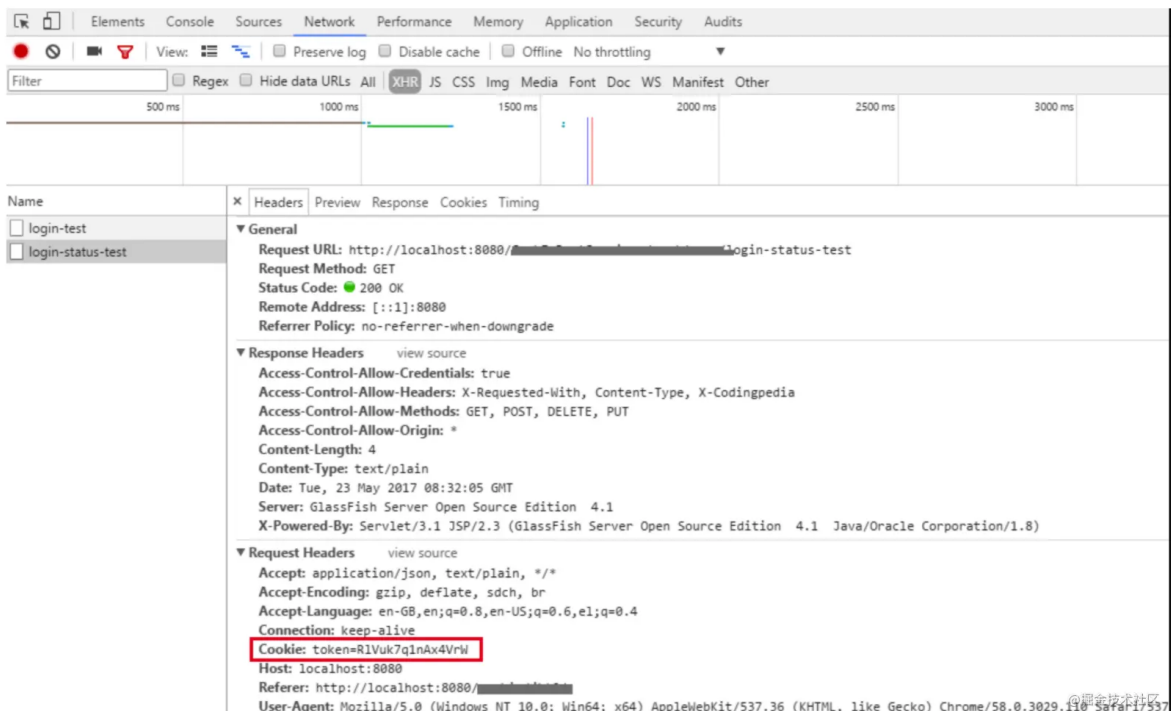
## Set-Cookie 和 Cookie 标头

`Set-Cookie` HTTP 响应标头将 cookie 从服务器发送到用户代理。下面是一个发送 Cookie 的例子



此标头告诉客户端存储 Cookie

现在，随着对服务器的每个新请求，浏览器将使用 Cookie 头将所有以前存储的 Cookie 发送回服务器。



有两种类型的 Cookies，一种是 Session Cookies，一种是 Persistent Cookies，如果 Cookie 不包含到期日期，则将其视为会话 Cookie。会话 Cookie 存储在内存中，永远不会写入磁盘，当浏览器关闭时，此后 Cookie 将永久丢失。如果 Cookie 包含 有效期，则将其视为持久性 Cookie。在到期指定的日期，Cookie 将从磁盘中删除。

还有一种是 cookie 的 `secure` 和 `HttpOnly` 标记，下面依次来介绍一下

## 会话 Cookies

上面的示例创建的是会话 Cookie，会话 Cookie 有个特征，客户端关闭时 Cookie 会删除，因为它没有指定 Expires 或 Max-Age 指令。

但是，Web 浏览器可能会使用会话还原，这会使大多数会话 Cookie 保持永久状态，就像从未关闭过浏览器一样。

## 永久性 Cookies

永久性 Cookie 不会在客户端关闭时过期，而是在特定日期（Expires）或特定时间长度（Max-Age）外过期。例如

```
Set-Cookie: id=a3fwa; Expires=Wed, 21 Oct 2015 07:28:00 GMT;
```

## Cookie 的 Secure 和 HttpOnly 标记

安全的 Cookie 需要经过 HTTPS 协议通过加密的方式发送到服务器。即使是安全的，也不应该将敏感信息存储在 cookie 中，因为它们本质上是不安全的，并且此标志不能提供真正的保护。

### HttpOnly 的作用

- 会话 Cookie 中缺少 HttpOnly 属性会导致攻击者可以通过程序(JS脚本、Applet等)获取到用户的 Cookie 信息，造成用户 Cookie 信息泄露，增加攻击者的跨站脚本攻击威胁。
- HttpOnly 是微软对 Cookie 做的扩展，该值指定 Cookie 是否可通过客户端脚本访问。
- 如果在 Cookie 中没有设置 HttpOnly 属性为 true，可能导致 Cookie 被窃取。窃取的 Cookie 可以包含标识站点用户的敏感信息，如 ASP.NET 会话 ID 或 Forms 身份验证票证，攻击者可以重播窃取的 Cookie，以便伪装成用户或获取敏感信息，进行跨站脚本攻击等。

## Cookie 的作用域

Domain 和 Path 标识定义了 Cookie 的作用域：即 Cookie 应该发送给哪些 URL。

Domain 标识指定了哪些主机可以接受 Cookie。如果不指定，默认为当前主机(不包种子域名)。如果指定了 Domain，则一般包含子域名。

例如，如果设置 Domain=mozilla.org，则 Cookie 也包含在子域名中（如 developer.mozilla.org）。

例如，设置 Path=/docs，则以下地址都会匹配：

- /docs
- /docs/web/
- /docs/web/HTTP

## JSON Web Token 和 Session Cookies 的对比

JSON Web Token，简称 JWT，它和 Session 都可以为网站提供用户的身份认证，但是它们不是一回事。

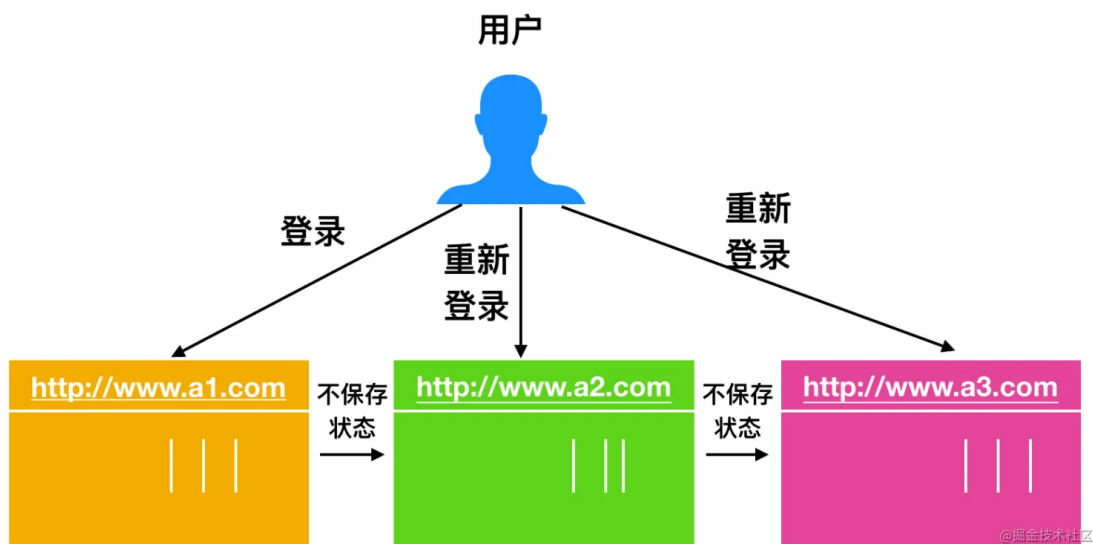
下面是 JWT 和 Session 不同之处的研究

## JWT 和 Session Cookies 的相同之处

在探讨 JWT 和 Session Cookies 之前，有必要需要先去理解一下它们的相同之处。

它们既可以对用户进行身份验证，也可以用来在用户单击进入不同页面时以及登陆网站或应用程序后进行身份验证。

如果没有这两者，那你可能需要在每个页面切换时都需要进行登录了。因为 HTTP 是一个无状态的协议。这也就意味着当你访问某个网页，然后单击同一站点上的另一个页面时，服务器的内存中 将不会记住你之前的操作。



因此，如果你登录并访问了你有权访问的另一个页面，由于 HTTP 不会记录你刚刚登录的信息，因此你将再次登录。

**JWT 和 Session Cookies 就是用来处理在不同页面之间切换，保存用户登录信息的机制。**

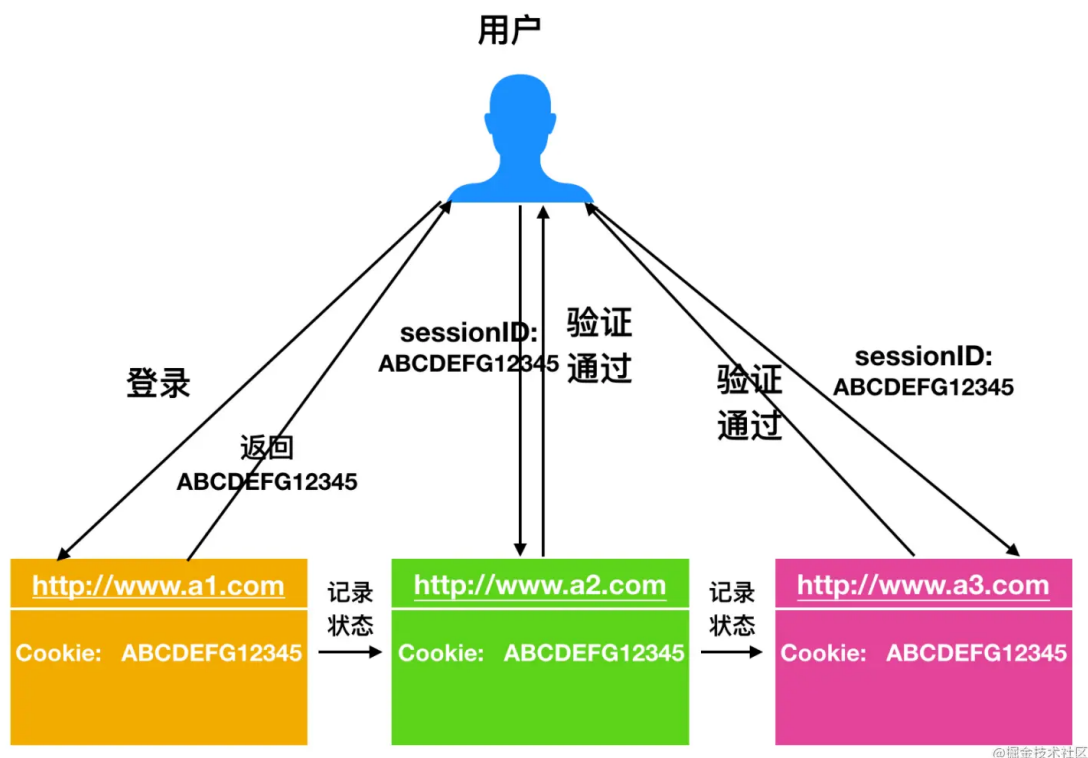
也就是说，这两种技术都是用来保存你的登录状态，能够让你在浏览任意受密码保护的网站。通过在每次产生新的请求时对用户数据进行身份验证来解决此问题。

所以 JWT 和 Session Cookies 的相同之处是什么？那就是它们能够支持你在发送不同请求之间，记录并验证你的登录状态的一种机制。

## 什么是 Session Cookies

Session Cookies 也称为 会话 Cookies，在 Session Cookies 中，用户的登录状态会保存在 服务器的内存中。当用户登录时，Session 就被服务端安全的创建。

在每次请求时，服务器都会从会话 Cookie 中读取 SessionId，如果服务端的数据和读取的 SessionId 相同，那么服务器就会发送响应给浏览器，允许用户登录。



## 什么是 Json Web Tokens

Json Web Token 的简称就是 JWT，通常可以称为 **Json 令牌**。它是 RFC 7519 中定义的用于安全的将信息作为 **Json 对象** 进行传输的一种形式。JWT 中存储的信息是经过 **数字签名** 的，因此可以被信任和理解。可以使用 HMAC 算法或使用 RSA/ECDSA 的公用/专用密钥对 JWT 进行签名。

使用 JWT 主要用来下面两点

- **认证(Authorization)**：这是使用 JWT 最常见的一种情况，一旦用户登录，后面每个请求都会包含 JWT，从而允许用户访问该令牌所允许的路由、服务和资源。**单点登录** 是当今广泛使用 JWT 的一项功能，因为它的开销很小。
- **信息交换(Information Exchange)**：JWT 是能够安全传输信息的一种方式。通过使用公钥/私钥对 JWT 进行签名认证。此外，由于签名是使用 **head** 和 **payload** 计算的，因此你还可以验证内容是否遭到篡改。

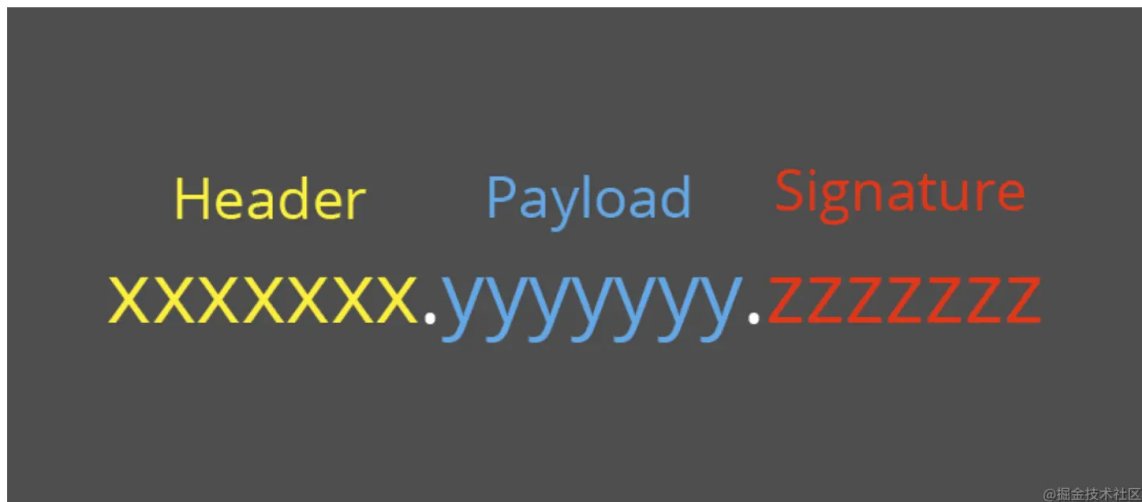
## JWT 的格式

下面，我们会探讨一下 JWT 的组成和格式是什么

JWT 主要由三部分组成，每个部分用 **.** 进行分割，各个部分分别是

- **Header**
- **Payload**
- **Signature**

因此，一个非常简单的 JWT 组成会是下面这样



然后我们分别对不同的部分进行探讨。

## Header

Header 是 JWT 的标头，它通常由两部分组成：令牌类型（即 JWT）和使用的签名算法，例如 HMAC SHA256 或 RSA。

例如

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

指定类型和签名算法后，json 块被 Base64Url 编码形成 JWT 的第一部分。

## Payload

Token 的第二部分是 Payload，Payload 中包含一个声明。声明是有关实体（通常是用户）和其他数据的声明。共有三种类型的声明：**registered**、**public** 和 **private** 声明。

- **registered** 声明：包含一组建议使用的预定义声明，主要包括

ISS	签发人
iss (issuer)	签发人
exp (expiration time)	过期时间
sub (subject)	主题
aud (audience)	受众
nbf (Not Before)	生效时间
iat (Issued At)	签发时间
jti (JWT ID)	编号

- **public** 声明：公共的声明，可以添加任何的信息，一般添加用户的相关信息或其他业务需要的必要信息，但不建议添加敏感信息，因为该部分在客户端可解密。

- **private 声明**：自定义声明，旨在在同意使用它们的各方之间共享信息，既不是注册声明也不是公共声明。

例如

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

然后 payload json 块会被 `Base64Url` 编码形成 JWT 的第二部分。

## signature

JWT 的第三部分是一个签证信息，这个签证信息由三部分组成

- header (base64后的)
- payload (base64后的)
- secret

比如我们需要 HMAC SHA256 算法进行签名

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

签名用于验证消息在此过程中没有更改，并且对于使用私钥进行签名的令牌，它还可以验证 JWT 的发送者的真实身份

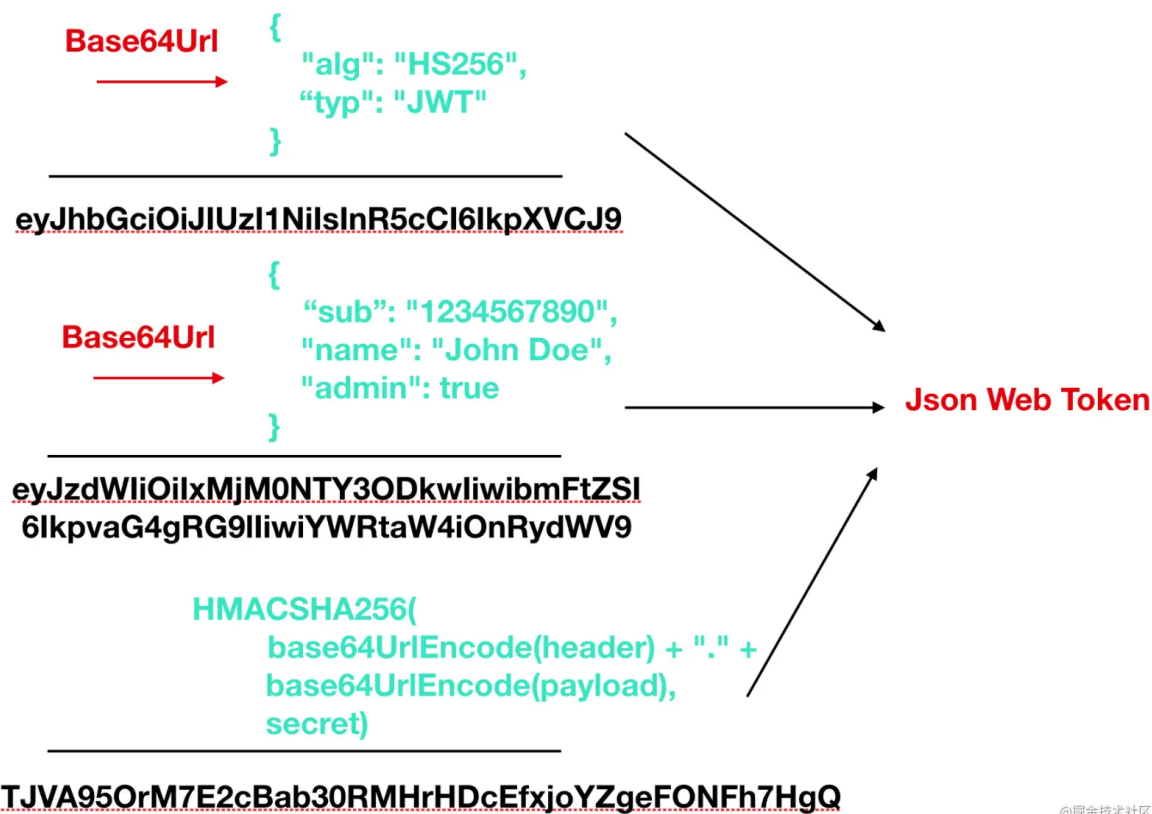
## 拼凑在一起

现在我们把上面的三个由点分隔的 Base64-URL 字符串部分组成在一起，这个字符串可以在 HTML 和 HTTP 环境中轻松传递这些字符串。

下面是一个完整的 JWT 示例，它对 header 和 payload 进行编码，然后使用 signature 进行签名

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjMONTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0Ij0iOnRydWV9.TjVA95OrM7E2cBab30RMhRHDcEfxjoYZgeFONFh7HgQ
```





如果想自己测试编写的话，可以访问 JWT 官网 [jwt.io/#debugger-i...](https://jwt.io/#debugger-i...)

## JWT 和 Session Cookies 的不同

JWT 和 Session Cookies 都提供安全的用户身份验证，但是它们有以下几点不同

## 密码签名

JWT 具有加密签名，而 Session Cookies 则没有。

## JSON 是无状态的

IWT 是无状态的，因为声明被存储在客户端，而不是服务端内存中。

身份验证可以在本地进行，而不是在请求必须通过服务器数据库或类似位置中进行。这意味着可以对用户进行多次身份验证，而无需与站点或应用程序的数据库进行通信，也无需在此过程中消耗大量资源。

## 可扩展性

Session Cookies 是存储在服务器内存中，这就意味着如果网站或者应用很大的情况下会耗费大量的资源。由于 JWT 是无状态的，在许多情况下，它们可以节省服务器资源。因此 JWT 要比 Session Cookies 具有更强的可扩展性。

## JWT 支持跨域认证

Session Cookies 只能用在 单个节点的域 或者它的 子域 中有效。如果它们尝试通过第三个节点访问, 就会被禁止。如果你希望自己的网站和其他站点建立安全连接时, 这是一个问题。

使用 JWT 可以解决这个问题，使用 JWT 能够通过多个节点进行用户认证，也就是我们常说的跨域认证。

## JWT 和 Session Cookies 的选型

我们上面探讨了 JWT 和 Cookies 的不同点，相信你也会对选型有了更深的认识，大致来说

对于只需要登录用户并访问存储在站点数据库中的一些信息的中小型网站来说，Session Cookies 通常就能满足。

如果你有企业级站点，应用程序或附近的站点，并且需要处理大量的请求，尤其是第三方或很多第三方（包括位于不同域的API），则 JWT 显然更适合。

## 后记

---

前两天面试的时候问到了这个题，所以写篇文章总结一下，还问到了一个面试题，**禁用 Cookies，如何使用 Session**？网上百度了一下，发现这是 PHP 的面试题.....

但还是选择了解了一下，如何禁用 Cookies 后，使用 Session

- 如果禁用了 Cookies，服务器仍会将 sessionId 以 cookie 的方式发送给浏览器，但是，浏览器不再保存这个cookie (即sessionId) 了。
- 如果想要继续使用 session，需要采用 `URL 重写` 的方式来实现，可以参考 [www.cnblogs.com/Renji-Fan/p...](http://www.cnblogs.com/Renji-Fan/p...)

[原文](#)