用户认证方式对比

实现用户登录认证的方式常见的有两种：一种是**基于 cookie 的认证**，另外一种是**基于 token 的认证**。

传统的基于 cookie 的认证方式

基本有下面几个步骤：

- 用户输入用户名和密码，发送给服务器
- 服务器验证一下用户名和密码，正确的话就创建一个会话（ session ）
- 同时会把这个会话的 ID 保存到客户端浏览器中，因为保存的地方是浏览器的 cookie ，所以这种认证方式叫做**基于 cookie** 的认证方式

- 后续的请求中，浏览器会发送会话 ID 到服务器，服务器上如果能找到对应 ID 的会话，那么服务器就会返回需要的数据给浏览器
- 当用户退出登录，会话会同时在客户端和服务器端被销毁

多年以来这种方式都是很流行的，例如，前几年我使用 Ruby On Rails 做开发，这个是 Rails 默认的认证方式。

当代的基于 token 的认证方式

对比上面的方式，有比较明显的不同：

- 用户输入用户名密码，发送给服务器
- 服务器验证一下用户名和密码，正确的话就返回一个签名过的 token（ token 可以认为就是个长长的字符串）
- 客户端浏览器拿到这个 token
- 后续每次请求中，浏览器会把 token 作为 http header 发送给服务器
- 服务器可以验证一下签名是否有效，如果有效那么认证就成功了，可以返回客户端需要的数据
- 这种方式的特点就是客户端的 token 中自己保留有大量信息，服务器没有存储这些信息，而只负责验证
- 所以一旦用户退出登录，只需要客户端销毁一下 token 即可，服务器端不需要有任何操作

我们做当代的 nodejs 开发，一般都用这样的方式进行认证。基于 token 的认证，一般会用到 JWT

技术，我们下一集里面详细介绍一下 JWT 。

为何我们的项目会采用后者？

简单来说一句话：

> *基于 Token 的认证方式才适合像 Angular, React, Vue 这样的 SPA（单页面应用）框架。*

具体来说，cookie-based 的方案有以下缺点：

- 服务器端要为每个用户保留 session 信息，连接用户多了，服务器内存压力巨大
- 适合单一域名，不适合第三方请求的形式，这个在当代是个大问题

token-based 的方案的优势是：

- 让我们的后台 API 更容易被移动 App 使用
- token 自己存储所有需要的各种信息，每次认证的时候服务器上不必进行数据库查询，执行效率大大提高

参考

- https://auth0.com/blog/cookies-vs-tokens-definitive-guide/
- https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication