JWT 身份认证优缺点分析以及常见问题解决方案

之前分享了一个使用 Spring Security 实现 JWT 身份认证的 Demo，文章地址：适合初学者入门 Spring Security With JWT 的 Demo。 Demo 非常简单，没有介绍到 JWT 存在的一些问题。所以，单独抽了一篇文章出来介绍。为了完成这篇文章，我查阅了很多资料和文献，我觉得应该对大家有帮助。

相关阅读：

- 《一问带你区分清楚Authentication,Authorization以及Cookie、Session、Token》
- 适合初学者入门 Spring Security With JWT 的 Demo
- Spring Boot 使用 JWT 进行身份和权限验证

Token 认证的优势

相比于 Session 认证的方式来说，使用 token 进行身份认证主要有下面三个优势：

1.无状态

token 自身包含了身份验证所需要的所有信息，使得我们的服务器不需要存储 Session 信息，这显然增加了系统的可用性和伸缩性，大大减轻了服务端的压力。但是，也正是由于 token 的无状态，也导致了它最大的缺点：当后端在token 有效期内废弃一个 token 或者更改它的权限的话，不会立即生效，一般需要等到有效期过后才可以。另外，当用户 Logout 的话，token 也还有效。除非，我们在后端增加额外的处理逻辑。

2.有效避免了CSRF 攻击

CSRF（Cross Site Request Forgery）一般被翻译为 **跨站请求伪造**，属于网络攻击领域范围。相比于 SQL 脚本注入、XSS等等安全攻击方式，CSRF 的知名度并没有它们高。但是,它的确是每个系统都要考虑的安全隐患，就连技术帝国 Google 的 Gmail 在早些年也被曝出过存在 CSRF 漏洞，这给 Gmail 的用户造成了很大的损失。

那么究竟什么是 **跨站请求伪造** 呢？说简单用你的身份去发送一些对你不友好的请求。举个简单的例子：

小壮登录了某网上银行，他来到了网上银行的帖子区，看到一个帖子下面有一个链接写着“科学理财，年盈利率过万”，小壮好奇的点开了这个链接，结果发现自己的账户少了10000元。这是这么回事呢？原来黑客在链接中藏了一个请求，这个请求直接利用小壮的身份给银行发送了一个转账请求,也就是通过你的 Cookie 向银行发出请求。

```
<a src=http://www.mybank.com/Transfer?bankId=11&money=10000>科学理财，年盈利率过万
</>
```

导致这个问题很大的原因就是： Session 认证中 Cookie 中的 session_id 是由浏览器发送到服务端的，借助这个特性，攻击者就可以通过让用户误点攻击链接，达到攻击效果。

那为什么 token 不会存在这种问题呢?

我是这样理解的：一般情况下我们使用 JWT 的话，在我们登录成功获得 token 之后，一般会选择存放在 local storage 中。然后我们在前端通过某些方式会给每个发到后端的请求加上这个 token,这样就不会出现 CSRF 漏洞

的问题。因为，即使有个你点击了非法链接发送了请求到服务端，这个非法请求是不会携带 token 的，所以这个请求将是非法的。

但是这样会存在 XSS 攻击中被盗的风险，为了避免 XSS 攻击，你可以选择将 token 存储在标记为`httpOnly` 的 cookie 中。但是，这样又导致了你必须自己提供CSRF保护。

具体采用上面哪两种方式存储 token 呢，大部分情况下存放在 local storage 下都是最好的选择，某些情况下可能需要存放在标记为`httpOnly` 的cookie 中会更好。

3.适合移动端应用

使用 Session 进行身份认证的话，需要保存一份信息在服务器端，而且这种方式会依赖到 Cookie（需要 Cookie 保存 SessionId），所以不适合移动端。

但是，使用 token 进行身份认证就不会存在这种问题，因为只要 token 可以被客户端存储就能够使用，而且 token 还可以跨语言使用。

4.单点登录友好

使用 Session 进行身份认证的话，实现单点登录，需要我们把用户的 Session 信息保存在一台电脑上，并且还会遇到常见的 Cookie 跨域的问题。但是，使用 token 进行认证的话， token 被保存在客户端，不会存在这些问题。

Token 认证常见问题以及解决办法

1.注销登录等场景下 token 还有效

与之类似的具体相关场景有：

1. 退出登录;
2. 修改密码;
3. 服务端修改了某个用户具有的权限或者角色；
4. 用户的帐户被删除/暂停。
5. 用户由管理员注销；

这个问题不存在于 Session 认证方式中，因为在 Session 认证方式中，遇到这种情况的话服务端删除对应的 Session 记录即可。但是，使用 token 认证的方式就不好解决了。我们也说过了，token 一旦派发出去，如果后端不增加其他逻辑的话，它在失效之前都是有效的。那么，我们如何解决这个问题呢？查阅了很多资料，总结了下面几种方案：

- **将 token 存入内存数据库**：将 token 存入 DB 中，redis 内存数据库在这里是是不错的选择。如果需要让某个 token 失效就直接从 redis 中删除这个 token 即可。但是，这样会导致每次使用 token 发送请求都要先从 DB 中查询 token 是否存在的步骤，而且违背了 JWT 的无状态原则。
- **黑名单机制**：和上面的方式类似，使用内存数据库比如 redis 维护一个黑名单，如果想让某个 token 失效的话就直接将这个 token 加入到 **黑名单** 即可。然后，每次使用 token 进行请求的话都会先判断这个 token 是否存在于黑名单中。
- **修改密钥 (Secret)** : 我们为每个用户都创建一个专属密钥，如果我们想让某个 token 失效，我们直接修改对应用户的密钥即可。但是，这样相比于前两种引入内存数据库带来了危害更大，比如：1如果服务是分布式的，则每次发出新的 token 时都必须在多台机器同步密钥。为此，你需要将必须将机密存储在数据库或其他外部服务中，这样和 Session 认证就没太大区别了。2如果用户同时在两个浏览器打开系

统，或者在手机端也打开了系统，如果它从一个地方将账号退出，那么其他地方都要重新进行登录，这是不可取的。

- **保持令牌的有效期限短并经常轮换**：很简单的一种方式。但是，会导致用户登录状态不会被持久记录，而且需要用户经常登录。

对于修改密码后 token 还有效问题的解决还是比较容易的，说一种我觉得比较好的方式：**使用用户的密码的哈希值对 token 进行签名。因此，如果密码更改，则任何先前的令牌将自动无法验证。**

2.token 的续签问题

token 有效期一般都建议设置的不太长，那么 token 过期后如何认证，如何实现动态刷新 token，避免用户经常需要重新登录?

我们先来看看在 Session 认证中一般的做法：**假如 session 的有效期30分钟，如果 30 分钟内用户有访问，就把 session 有效期被延长30分钟。**

1. **类似于 Session 认证中的做法**：这种方案满足于大部分场景。假设服务端给的 token 有效期设置为30分钟，服务端每次进行校验时，如果发现 token 的有效期马上快过期了，服务端就重新生成 token 给客户端。客户端每次请求都检查新旧token，如果不一致，则更新本地的token。这种做法的问题是仅仅在快过期的时候请求才会更新 token ,对客户端不是很友好。
2. **每次请求都返回新 token** :这种方案的的思路很简单，但是，很明显，开销会比较大。
3. **token 有效期设置到半夜**： 这种方案是一种折衷的方案，保证了大部分用户白天可以正常登录，适用于对安全性要求不高的系统。
4. **用户登录返回两个 token**： 第一个是 acessToken ，它的过期时间 token 本身的过期时间比如半个小时，另外一个是 refreshToken 它的过期时间更长一点比如为1天。客户端登录后，将 accessToken和 refreshToken 保存在本地，每次访问将 accessToken 传给服务端。服务端校验 accessToken 的有效性，如果过期的话，就将 refreshToken 传给服务端。如果有效，服务端就生成新的 accessToken 给客户端。否则，客户端就重新登录即可。该方案的不足是：1需要客户端来配合；2用户注销的时候需要同时保证两个 token 都无效；3重新请求获取 token 的过程中会有短暂 token 不可用的情况（可以通过在客户端设置定时器，当accessToken 快过期的时候，提前去通过 refreshToken 获取新的accessToken）。

总结

JWT 最适合的场景是不需要服务端保存用户状态的场景，比如如果考虑到 token 注销和 token 续签的场景话，没有特别好的解决方案，大部分解决方案都给 token 加上了状态，这就有点类似 Session 认证了。

Reference

- JWT 超详细分析
- https://medium.com/devgorilla/how-to-log-out-when-using-jwt-a8c7823e8a6
- https://medium.com/@agungsantoso/csrf-protection-with-json-web-tokens-83e0f2fcbcc
- Invalidating JSON Web Tokens