一 背景

最近在做的一个项目中，需要自己开发权限与角色功能，所以就再次调研了一下认证和授权框架及方案，JWT 也是其中之一。因此做本篇整理。

二 JWT 简介及 Token 认证方案

2.1 简介

JWT，即 JSON Web Tokens（JWT官网），也就是 JSON 结构的 Web Token。完整描述参见rfc7519。

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

简单翻译一下：JWT 是一种紧凑的、URL
安全的方法，用于表示要在双方之间传输的声明。JWT 中的声明被编码为 JSON 对象，该 JSON 对象被用作 JSON Web
签名（JWS）结构的有效负载，或 JSON Web 加密（JWE）结构的明文，使得声明要求能够被数字签名或用消息认证码（MAC）与/或加密的完整性保护。

2.2 Token 鉴权机制

Token 可以理解为令牌，我们对外暴露的服务，不希望任意人员都能够访问，因为其中可能存在讨厌的、会攻击系统的不受欢迎的用户。那么怎样过滤掉这些用户？用户注册-->登录是常见的选择。但登录之后，我们希望客户端能够保存一份临时的认证信息，这样就不必再后续的每次操作中都执行一次登录，或者不停地来回传送用户名和密码，这样既影响效率又不安全。为此，就有了基于 token 的认证和 session 认证。

基于 Token 的鉴权过程如下：

1. 用户使用用户名密码来请求服务器；
2. 服务器验证用户信息；
3. 服务器通过验证后，生成一个 token 并发送给用户；
4. 客户端存储 token，并在每次请求时带上这个 token 值；
5. 服务端验证 token 值，并返回数据；

有两点需要注意：

1）这个 token
必须要在每次请求时传递给服务端，通常保存在请求头（Header）；

2）服务端要支持CORS
策略，这点可以通过在服务端设置Access-Control-Allow-Origin: *实现。

2.3 基于 session 的认证

Http 协议是一种无状态的协议，而这就意味着如果用户向我们的应用提供了用户名和密码来进行用户认证，那么下一次请求时，用户还要再一次进行用户认证才行，因为根据 http 协议，我们并不能知道是哪个用户发出的请求，所以为了应用能识别是哪个用户发出的请求，只能在服务器存储一份用户登录的信息，这份登录信息会在响应时传递给浏览器，告诉其保存为 cookie,以便下次请求时发送给我们的应用，这样应用就能识别请求来自哪个用户了，这就是传统的基于 session 认证。

2.4 session 的认证方式问题

不过 session 的认证方式也存在很多问题：

1）session 保存在内存中，当认证的用户数量越来越多时，也会给服务器带来更大的开销；

2）同样由于 session 是在内存中，在服务端单机的情况下没有问题，但一旦采用多台机器部署，不能保证每次请求都转发到同一台机器上时就会存在问题。所以会有基于 db 或分布式缓存的分布式 session 方案；

3）CSRF 攻击：是基于 cookie 来进行用户识别的, cookie
如果被截获，用户就会很容易受到跨站请求伪造的攻击。

三 JWT 详解

3.1 JWT 组成

JWT 由 Header（头部）、Payload（载荷）、Signature（签名）三部分组成，并通过.分割。示例：xxxxx.yyyyy.zzzzz。

3.1.1 Header

Header 包括两部分信息：声明类型（typ） 和
声明加密算法（alg）。示例如下（注意，没有格式化和换行）：

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

其中typ 固定，就是 JWT，算法可以自行选择，不过一般会直接使用 HNAC SHA256 算法来做加密。

之后对 Header 做 Base64 编码，得到的结果为:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9。

3.1.2 Payload

载荷是 JWT 存储身份信息的地方，包括三个部分：

1）Registered claims - 注册声明

一组预定义的声明，它
们不是强制的，但是推荐。包括 iss (issuer), exp (expiration
time), sub (subject), aud (audience)等等。

2）Public claims - 公共声明

这些可以由使用 JWT 的人随意定义。但是为了避免冲突，应该在 IANA JSON Web token Registry
中定义它们，或者将它们定义为包含防冲突命名空间的 URI。

3）Private claims - 私有声明

这些是为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，既不是注册声明，也不是公共声明。

一个 payload 的示例：

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

之后 Base64 结果为:

```
eyJzdWIiOiAiMTIzNDU2Nzg5MCIsIm5hbWUiOiAiSm9obiBEb2UiLCJhZG1pbiI6IHRydWV9
```

3.1.3 Signature

为了得到签名部分，你必须有编码过的 header、编码过的 payload、一个秘钥，签名算法是 header 中指定的算法，然后对它们签名即可。

例如我们使用 HMAC SHA256 算法，签名将会以下面的方式生成：

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

签名将用于验证消息在传递过程中有没有被更改，并且对于使用私钥签名的 token，它还可以验证 JWT 的发送方是否为它所称的发送方。

https://jwt.io/#debugger-io 给出了一个解析示例，可以把 JWT 解析回加密前的三个部分：

1. 应用或客户端向授权服务器请求获取授权；
2. 当授权被授予以后，授权服务器返回给应用一个访问 token；
3. 应用使用 access token 访问被保护的资源（例如 API）。

3.3 何时使用 JWT

3.3.1 JWT 优势

1）因为 json 的通用性，JWT 是可以进行跨语言支持的；

2）由于有 payload 部分，JWT
可以在自身存储一些其他业务逻辑所必要的非敏感信息；

3）便于传输，JWT 的构成非常简单，字节占用很小，所以便于传输；

4）它不需要在服务端保存会话信息, 所以它易于应用的扩展。

3.3.2 问题和注意事项

1）如上面的 2）所述，payload
不能存放敏感信息，因为这部分客户可解密；

2）尽可能使用 HTTPS 协议

3）secret 私钥非常重要，需要谨慎保存

4）JWT 虽然不必存在服务端，但也需要注意，JWT 的 token
默认是永久生效的！要解决这一问题，需要考虑在 JWT
中包含生效时间、创建时间信息；或采用持久化方案，每次请求时判断 token
是否已失效（超时或退出登录导致）。

3.4 JWT 与 OAuth

JWT 是一种认证协议，而 OAuth2 是一种授权框架。JWT
适用于在前后端分离, 需要简单的对后台 API
进行保护时使用。更复杂的场景，例如使用第三方账号登录的情况，基于
OAuth2 的框架更为合适。

四 JWT 防止 CSRF

2.4 章节中说过，session 认证这种基于 cookie 的方式可能会受到 CSRF 攻击。CSRF 是跨站点请求伪造(Cross—Site Request Forgery)，类似的还有 XSS 攻击。

4.1 CSRF

简单理解 CSRF 就是，攻击者盗用了你的身份，以你的名义发送恶意请求，对服务器来说这个请求是完全合法的，但是却完成了攻击者所期望的一个操作，比如以你的名义发送邮件、发消息，盗取你的账号，添加系统管理员，甚至于购买商品、虚拟货币转账等。详解参见
什么是CSRF攻击？如何防御CRSF攻击？

4.2 防御方式

一般可以通过三种方法来避免 CSRF 攻击：

4.2.1 判断请求头中的 Referer

Referer 是 HTTP 请求header 的一部分，当浏览器（或者模拟浏览器行为）向web 服务器发送请求的时候，头信息里有包含 Referer。

拦截 Referer，在 Java Servlet 中可以用 Filter；Spring 可以建拦截器；通过 request.get('referer') 来取得这个值。

但要需要注意，Referer 是浏览器设置的，在浏览器兼容性大不相同的时代中，如果存在某种浏览器允许用户修改这个值，那么 CSRF 漏洞依然存在。

4.2.2 在请求参数中加入 csrf token

需要区分一下 GET 和 POST 请求。GET 被“约定”为查询操作，且通常需要保证幂等，所以一般不需要加其他额外的参数。这里要提醒各位的是，尽量遵从这些约定，不要在 GET 请求中出现 /delete, /update, /edit 这种单词。把写操作放到 POST 中。

POST 请求，服务端在创建表单的时候可以加一个隐藏字段，也是通过某种加密算法得到的。在处理请求时，验证这个字段是否合法，如果合法就继续处理，否则就认为是恶意操作。

```
<form method="post" action="/update">  <!-- ???? -->  <input
 type="hidden" name="csrftoken" value="?????"/></form>
```

这个 html 片段由服务端生成。但对于一些新兴网站，很多都采用了“单页”的设计，或者 HTML 可能是由 JavaScript
拼接而成，并且表单也都是异步提交。所以这个办法也有局限性。

4.2.3 新增 HTTP Header

将 token 放在请求头中，服务端可以像获取 Referer
一样获取这个请求头，不同的是，这个 token
是由服务端生成的，所以攻击者难以猜测这个 token。

到这里我们应该清楚了，JWT 本身就是对新增的 HTTP Header
的约定。所以就可以通过这种方式规避 CSRF 攻击。

五 小结

本篇整理了 JWT 的定义、组成、工作机制和优缺点，并介绍了相关的诸如 token 认证机制、CSRF 攻击及防御方式，对 JWT
有一个整体了解。下一篇将给出 JWT
的代码示例和一些问题的解决方案，敬请期待。