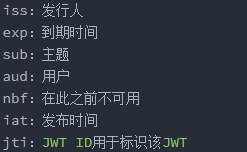
JWT概述：

## **Header + Payload + Signature**

## header:

Payload: 

Signature: *HMACSHA*256(*base*64*UrlEncode*(*header*)+"."+*base*64*UrlEncode*(*payload*),*secret*)

*HMACSHA*256为默认的签名算法

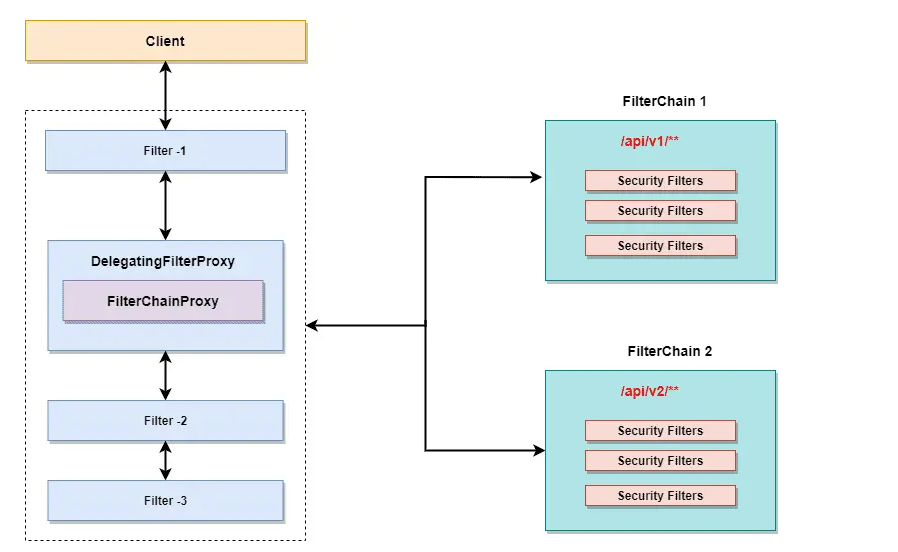
传统的session认证有如下的问题:

* 每个用户的登录信息都会保存到服务器的session中，****随着用户的增多，服务器开销会明显增大****
* 由于session是存在与服务器的物理内存中，所以****在分布式系统中，这种方式将会失效****。虽然可以将session统一保存到Redis中，但是这样做无疑增加了系统的复杂性，对于不需要redis的应用也会白白多引入一个缓存中间件
* ****对于非浏览器的客户端、手机移动端等不适用****，因为session依赖于cookie，而移动端经常没有cookie
* 由于基于Cookie，而****cookie无法跨域，所以session的认证也无法跨域，对单点登录不适用****
* 因为session认证本质基于cookie，所以如果cookie被截获，用户很容易收到跨站请求伪造攻击

### **JWT认证的优势**

* 简洁：JWT Token数据量小，传输速度也很快
* 因为JWT Token是以JSON加密形式保存在客户端的，所以JWT是****跨语言****的
* 不需要在服务端保存会话信息，也就是说****不依赖于cookie和session，所以没有了传统session认证的弊端，特别适用于分布式微服务****
* ****单点登录友好****：使用Session进行身份认证的话，由于cookie无法跨域，难以实现单点登录
* ****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

## How Security Filters Works



More details, see links:

1. <https://www.javadevjournal.com/spring-security/spring-security-filters/>
2. https://www.bezkoder.com/spring-boot-jwt-authentication/