# 前台框架权限认证过程简介

## 目的

我们需要实现以下几种效果：

1. 集群环境下，用户于一台服务器上某个端口下的应用登陆后，于其他任何本服务器或非本服务器上的应用，无需进行二次权限认证。且登陆信息始终共享。
2. 面对同一时间对访问用户较大的情况下，避免session存储过多而内存溢出。
3. 修改用户密码信息后，用户登陆状态主动失效，强制重登
4. 修改用户信息，如禁用、锁定等信息后，用户登陆状态主动失效
5. 支持根据用户信息的设定，能够实现一个账户多地登陆、二次登陆强制踢掉首次登陆、二次登陆强迫限制登陆等三种登陆策略
6. 记录账户所有登陆信息如IP、登陆、退出时间点
7. 支持用户一个账号多地登陆后，用户指定某已地点登陆强制退出
8. 可防御session会话劫持、CSRF攻击
9. 用户登陆后的有效会话时长，要求为最后一次服务器请求作为起始时间点

## Spring Security

### 简介

spring security是spring为解决权限认证等一系列与安全相关问题的解决方案组件。与shiro相同，但spring security功能相交shiro有更丰富的能力，并且支持其他如OAuth2、CAS、OpenID认证方式，与springmvc的结合更好，但其使用的复杂程度也较shiro更高。

### 原理

Spring security是通过构造过滤链FilterChain对所有约定的服务器请求进行过滤处理，每个过滤器各自负责不同的认证功能，如单独负责表单登陆的UsernamePasswordAuthencationFilter 、负责跨域请求的CORSFilter、负责会话攻击的CSRFFilter、负责退出的LoginOutFilter、负责session管理的SessionManagerFilter、负责OAuth2认证OAuth2ClientAuthenticationProcessingFilter等二十几种过滤器,spring 会根据你的security组件配置信息（如xml、Java Config、注解）动态创建过滤器后将其添加到过滤链FilterChain当中并根据配置信息当中约定的Filter位置对所有过滤器进行优先级排序，用于提供后续对服务器请求进行一系列的过滤行为。

### 启动过程

**1、引入组件**：WEB容器启动过程中（如Tomcat）会检测ServletContext中定义的DelegatingFilterProxy过滤器，该过滤器负责为我们的工程引入spring security组件，该过滤器在非spring boot工程中需在web.xml或动态添加到servletContext对象当中自行创建。

**2、扫描配置**：在spring容器在初始化过程中会检测security组件相关配置实例，一般我们自己会创建一个自定义的配置实例，比如我们自定义类继承WebSecurityConfigerAdapter并覆盖configure方法构造HttpSeucity，或者在web.xml当中去配置spring security相关的bean。而spring除了会识别我们自定义的security配置实例以外还会默认添加几个默认的配置实例。每一种配置实例都会创建若干个过滤器并添加到统一的FilterChain当中。Spring security会根据FilterComparator类中定义的过滤器顺序对Filter进行排序。在这里我们只需要关注我们自定义配置的配置实例如何被spring处理。

**3、过滤器相关的处理对象**：spring security中定义的每一种过滤器还包含一些与其相关的功能类，比如

* \*\*\*Handler我们可以理解为处理器，它一般被用到比如过滤器处理结果为成功或失败以后调用，用于处理如转发、重定向、修改request中的参数等等。
* \*\*\*EntryPoint我们可以理解为切入点、它的功能与Handler处理器类似，不同点在于EntryPoint会被用于匿名登陆者的处理（权限为空的用户）
* \*\*\*Provider为Filter通过AuthenticationManager调用，可提供对服务器请求的进一步处理。
* \*\*\*Strategy为session处理策略，其继承自SessionAuthenticationStrategy，一般用于过滤器过滤后调用，处理session使用。

## Agile Framework 认证过程

### 简介

Agile为满足分布式微服务架构下客户端（浏览器）统一认证问题，认证过程采用无状态话的JWT Token + Redis缓存认证，完全去除服务器端对session的依赖。原由主要包括以下几点：

1、集群情况下session需要于多个服务点上同步与共享，来保证浏览器与服务器集群之间的一次认证多处使用的问题。

2、随着登陆用户增多，session存储的用户状态信息会越来越多，极大的占用服务器内存。

3、jsessionid容易遭受会话劫持与CSRF攻击

4、跨域情况下cookie是无法共享的，token通过header传数将不存在此种限制

### JWT

JWT是一种生成加密用户身份信息Token，且提供解密与过期功能的组件，是目前最流行的一种无状态话认证方式。它的基本原理是用户登陆我们系统以后，服务器会根据用户名及过期时间与加密口令生成一串较难破译的会话令牌Token，然后将该Token已cookie或header等方式回传给浏览器。那么当浏览器再次请求服务器时，将携带该token令牌，服务器对该令牌进行过期验证并解密，用于识别用户身份。在该过程中服务器始终不存储任何用户相关的信息，也不依赖浏览器会话session。

单一的JWT Token认证方式存在以下缺点：

1、token生成后无法强迫失效，只能等待其自动过期失效

2、与jsessionid同样存在泄露问题

3、token无法延时

4、修改账户信息，比如密码，token无法同步

5、token存储的信息不能过多，否则效率很慢

### 解决方案

**关键点**：JWT + Redis缓存

**登陆**：

1. 用户登陆时，系统校验账号、密码（密码通过Spring BCryptPasswordEncoder加解密）、权限数据，未通过时将请求转发至异常处理方法抛出异常json、xml响应
2. 通过时系统根据用户数据中记录的同账号异地登陆策略，查询redis缓存中的该账号已登陆信息，并根据策略做相应处理。
   1. 限制二次登陆：直接对该登陆请求返回限制提示
   2. 强制踢掉前一次登陆：于redis缓存中删除前一次的登陆标记
   3. 多地在线：允许登陆
3. 通过时系统根据账号、密码、权限信息生成UsernamePasswordAuthenticationToken对象，与随机8位盐值作为登陆标记。直接以账号、账号+'\_SALT'为缓存KEY值存储UsernamePasswordAuthenticationToken与本次登陆标记。登陆标识为每次登陆时自动追加、退出时删除当次标记。
4. 系统根据账号、随机登陆标记、密码密文、登陆时间生成有效期30分钟的token，存放在response header/cookie中用于响应，同时存储用户登陆信息记录ip时间点与登陆标识。

**通信**：

1. 用户请求系统资源时，需携带该token值用于系统身份认证。系统未检测到token时报用户未登陆
2. token存在时检测token是否过期，过期直接报过期
3. token未过期时，系统解密token获取账号、随机登陆标记，根据账号获取缓存中的UsernamePasswordAuthenticationToken与登陆标识，对比登陆标识、密码密文、验证缓存中UsernamePasswordAuthenticationToken的合法性（包括账户禁用、锁定等），未通过时直接抛出提示信息。Token验证通过后将UsernamePasswordAuthenticationToken添加到SecurityContext，提供给业务层获取用户身份信息。
4. 业务层处理结束后，系统将已本次通信作为新的时间点与随机8位盐值作为新登陆标记，生成新的token，同时刷新数据库登陆信息中的登陆标记。以此保证每次通信的token与登陆标识将是唯一，并且由于登陆标识的限制，旧的token将在新token生成时自动失效。新token已本次通信作为新的时间节点实现自动延期。

**退出**：

1. 解析token中携带的登陆标记，并于缓存中清除该账号下的本次登陆标记，同时刷新登陆信息中的退出时间。由于缓存中本次登陆标记被清除，旧token将自动失效。

**修改**：

1. 修改用户信息时，用户信息同步到缓存中，保证通信过程的认证尽可能使用缓存数据避免过多的持久层操作。

**防御**：

1. 每一次的服务器请求，用户所获得的有效token令牌将是新的、唯一的。并且传输令牌的方式使用response header传数，所以攻击者无法通过获取cookies中的jsessionid或token来伪造身份。即便攻击者拿到我们的token令牌，由于每次服务器请求将产生新的token令牌，旧令牌立即失效，所以攻击者即便拿到本次会话的令牌，也将是无效的。

唯一的攻击缝隙时间，是在截获token后，真正的用户没有做再一次的服务器请求让被截获token失效的情况下，该缝隙时间会很小，也给攻击增加一定难度。