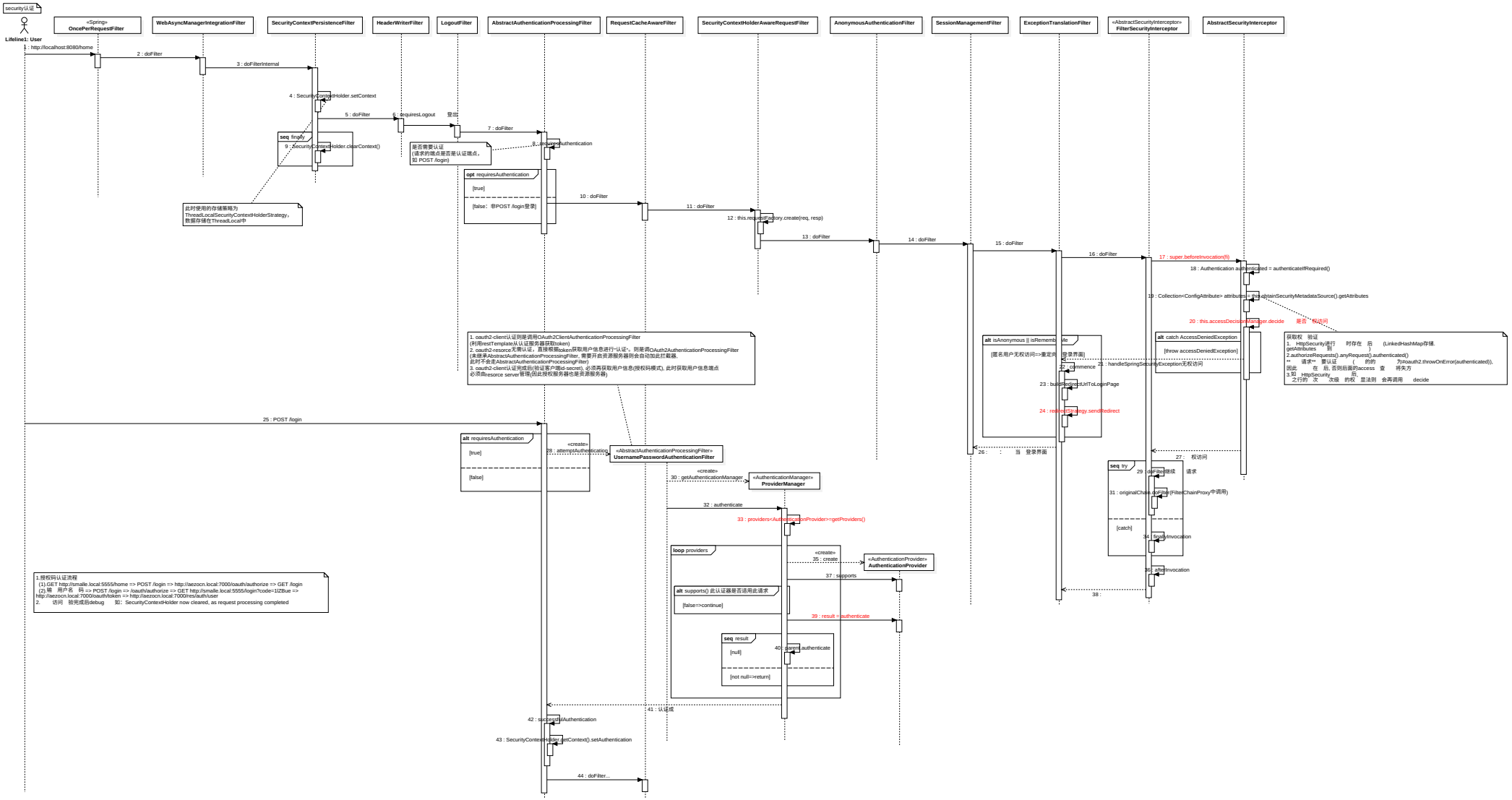
security认证
Lifeline1: User
«Spring» OncePerRequestFilter
WebAsyncManagerIntegrationFilter
SecurityContextPersistenceFilter
HeaderWriterFilter
LogoutFilter
AbstractAuthenticationProcessingFilter
RequestCacheAwareFilter
SecurityContextHolderAwareRequestFilter
AnonymousAuthenticationFilter
SessionManagementFilter
ExceptionTranslationFilter
«AbstractSecurityInterceptor» FilterSecurityInterceptor
AbstractSecurityInterceptor
1 : http://localhost:8080/home
2 : doFilter
3 : doFilterInternal
4 : SecurityContextHolder.setContext
5 : doFilter
6 : requiresLogout
登出
7 : doFilter
8 : requiresAuthentication
seq finally
9 : SecurityContextHolder.clearContext()
是否需要认证
(请求的端点是否是认证端点，
如 POST /login)
opt requiresAuthentication
[true]
[false：非POST /login登录]
10 : doFilter
11 : doFilter
12 : this.requestFactory.create(req, resp)
13 : doFilter
14 : doFilter
15 : doFilter
16 : doFilter
17 : super.beforeInvocation(fi)
18 : Authentication authenticated = authenticateIfRequired()
19 : Collection<ConfigAttribute> attributes = this.obtainSecurityMetadataSource().getAttributes
20 : this.accessDecisionManager.decide
是否 权访问
此时使用的存储策略为
ThreadLocalSecurityContextHolderStrategy，
数据存储在ThreadLocal中
alt isAnonymous || isRemember
[匿名用户无权访问=>重定向 登录界面]
21 : handleSpringSecurityException(无权访问)
22 : commence
23 : buildRedirectUrlToLoginPage
24 : redirectStrategy.sendRedirect
alt catch AccessDeniedException
[throw accessDeniedException]
获取权 验证
1 HttpSecurity进行 时存在 后 (LinkedHashMap存储.
getAttributes 到)
2.authorizeRequests().anyRequest().authenticated()
** 请求** 要认证 (的的 为#oauth2.throwOnError(authenticated)),
因此 在 后,否则后面的access 查 将失方
3.如 HttpSecurity 后,
之行的 次 次级 的权 显法则 会再调用 decide
1. oauth2-client认证则是调用OAuth2ClientAuthenticationProcessingFilter
(利用restTemplate从认证服务器获取token)
2. oauth2-resource无需认证，直接根据token获取用户信息进行认证。则是调OAuth2AuthenticationProcessingFilter
(未继承AbstractAuthenticationProcessingFilter, 需要开启资源服务器则会自动加此拦截器.
此时不会走AbstractAuthenticationProcessingFilter)
3. oauth2-client认证完成后(验证客户端id-secret), 必须再获取用户信息(授权码模式), 此时获取用户信息端点
必须由resorce server管理(即此授权服务器也是资源服务器)
25 : POST /login
alt requiresAuthentication
[true]
[false]
«create»
26 : 当 登录界面
27 : 权访问
28 : attemptAuthentication
«AbstractAuthenticationProcessingFilter» UsernamePasswordAuthenticationFilter
«create»
30 : getAuthenticationManager
«AuthenticationManager» ProviderManager
seq try
29 : doFilter(继续 请求
31 : originalChain.doFilter(FilterChainProxy中调用)
[catch]
34 : finallyInvocation
35 : afterInvocation
38 :
32 : authenticate
33 : providers<AuthenticationProvider>.getProviders()
loop providers
«create»
36 : create
«AuthenticationProvider» AuthenticationProvider
37 : supports
alt supports() 此认证器是否适用此请求
[false=>continue]
39 : result = authenticate
seq result
[null]
40 : parent.authenticate
[not null=>return]
41 : 认证成
42 : successfulAuthentication
43 : SecurityContextHolder.getContext().setAuthentication
44 : doFilter...
1.授权码认证流程
(1).GET http://smalle.local:5555/home => POST /login => http://aezocn.local:7000/oauth/authorize => GET /login
(2).输 用户名 码 => POST /login => /oauth/authorize => GET http://smalle.local:5555/login?code=1IZBue =>
http://aezocn.local:7000/oauth/token => http://aezocn.local:7000/res/auth/user
2. 访问 验完成后debug 如：SecurityContextHolder now cleared, as request processing completed

blog.aezo.cn