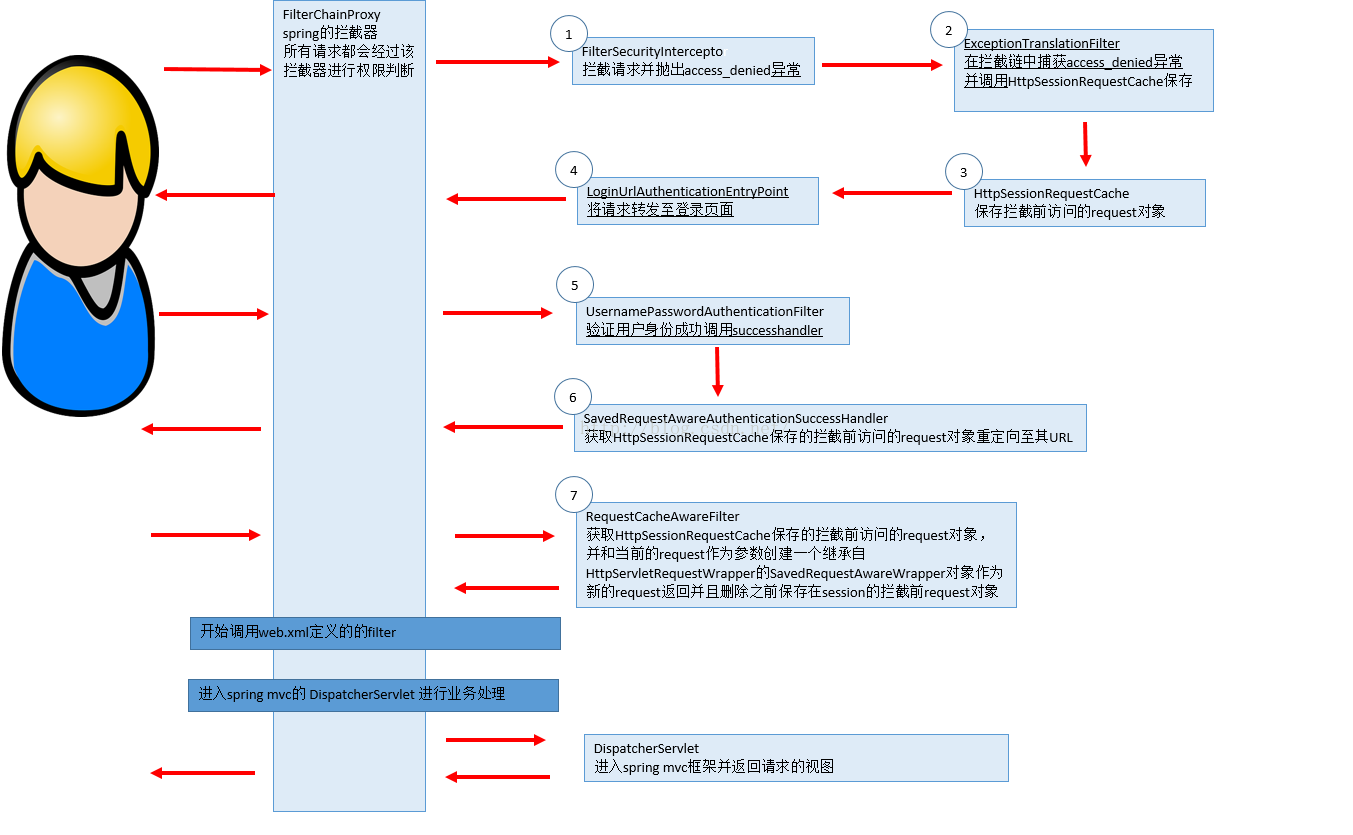
**关于spring-security登录后重定向至拦截前访问的url的实现原理**

原创 2015年10月31日 19:08:38

* 7436

首先我们来看下我们整个流程图

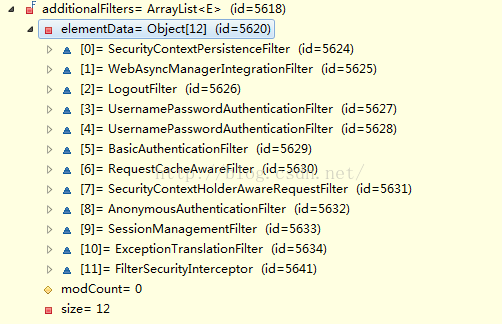


这就是我自己摸索出来的关于整个访问拦截登录重定向的流程图 其中3,6,7步是对拦截前访问的request的处理

接下来是对以上几个步骤中关键代码的分析

**1.**

首先我们先了解下关于 FilterSecurityIntercepto所在的位置



我们看到他在最后一位从这个拦截器的下面代码：

**[java]** [view plain](http://blog.csdn.net/zy_cookie/article/details/49535413) [copy](http://blog.csdn.net/zy_cookie/article/details/49535413)

1. // Attempt authorization
2. **try** {
3. **this**.accessDecisionManager .decide(authenticated , object , attributes );
4. }
5. **catch** (AccessDeniedException accessDeniedException ) {
6. publishEvent( **new** AuthorizationFailureEvent(object , attributes , authenticated, accessDeniedException ));
8. **throw** accessDeniedException;
9. }

**2.**

我们能够看出当其在身份的权限验证失败的情况下会抛出accessDeniedException异常，这个时候上一层的拦截器也就是倒数第二个拦截器ExceptionTranslationFilter捕获了该异常并进一步处理具体代码如下

**[java]** [view plain](http://blog.csdn.net/zy_cookie/article/details/49535413) [copy](http://blog.csdn.net/zy_cookie/article/details/49535413)

1. **private** **void** handleSpringSecurityException(HttpServletRequest request, HttpServletResponse response , FilterChain chain,
2. RuntimeException exception) **throws** IOException, ServletException {
3. **if** (exception **instanceof** AuthenticationException) {
4. logger.debug( "Authentication exception occurred; redirecting to authentication entry point", exception);
6. sendStartAuthentication( request, response, chain, (AuthenticationException) exception );
7. }
8. **else** **if** (exception **instanceof** AccessDeniedException ) {
9. **if** (authenticationTrustResolver .isAnonymous(SecurityContextHolder.getContext().getAuthentication())) {
10. logger.debug( "Access is denied (user is anonymous); redirecting to authentication entry point",
11. exception);
13. sendStartAuthentication( request, response, chain, **new** InsufficientAuthenticationException(
14. "Full authentication is required to access this resource"));
15. }
16. **else** {
17. logger.debug( "Access is denied (user is not anonymous); delegating to AccessDeniedHandler", exception);
19. accessDeniedHandler.handle(request , response , (AccessDeniedException) exception);
20. }
21. }
22. }

**3.**

进入sendStartAuthentication这个函数后我们看到了

**[java]** [view plain](http://blog.csdn.net/zy_cookie/article/details/49535413) [copy](http://blog.csdn.net/zy_cookie/article/details/49535413)

1. **protected** **void** sendStartAuthentication(HttpServletRequest request, HttpServletResponse response , FilterChain chain,
2. AuthenticationException reason) **throws** ServletException, IOException {
3. // SEC-112: Clear the SecurityContextHolder's Authentication, as the
4. // existing Authentication is no longer considered valid
5. SecurityContextHolder. getContext().setAuthentication(**null**);
6. requestCache.saveRequest(request , response );
7. logger.debug( "Calling Authentication entry point." );
8. authenticationEntryPoint.commence(request , response , reason );
9. }

这里就有requestCache.saveRequest(request , response );对对象进行保存的业务代码。这个是保存的key

**static** **final** String ***SAVED\_REQUEST*** = "SPRING\_SECURITY\_SAVED\_REQUEST";

**6.**

登录请求验证完成之后 UsernamePasswordAuthenticationFilter会调用SavedRequestAwareAuthenticationSuccessHandler的实例loginsuccesshandler来处理登录成功后的处理步骤

**public** **void** onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response ,

            Authentication authentication) **throws** ServletException, IOException {

        SavedRequest savedRequest = requestCache.getRequest( request, response);

**if** (savedRequest == **null**) {

**super**.onAuthenticationSuccess(request , response , authentication );

**return**;

        }

        String targetUrlParameter = getTargetUrlParameter();

**if** (isAlwaysUseDefaultTargetUrl() || (targetUrlParameter != **null** && StringUtils.*hasText*(request.getParameter( targetUrlParameter)))) {

            requestCache.removeRequest(request , response );

**super**.onAuthenticationSuccess(request , response , authentication );

**return**;

        }

        clearAuthenticationAttributes( request);

        // Use the DefaultSavedRequest URL

        String targetUrl = savedRequest .getRedirectUrl();

        logger.debug( "Redirecting to DefaultSavedRequest Url: " + targetUrl );

        getRedirectStrategy().sendRedirect( request, response, targetUrl);

    }

我们在这里看到该方法从session中获取缓存的拦截前request对象,并进行除重定向之外无其他的操作

**7.**

我们看到 RequestCacheAwareFilter在拦截器链中排第7位，它的作用就是获取session中的保存的request并对当前的request进行替换工作

**[java]** [view plain](http://blog.csdn.net/zy_cookie/article/details/49535413) [copy](http://blog.csdn.net/zy_cookie/article/details/49535413)

1. **public** **void** doFilter(ServletRequest request , ServletResponse response, FilterChain chain)
2. **throws** IOException, ServletException {
4. HttpServletRequest wrappedSavedRequest =
5. requestCache.getMatchingRequest((HttpServletRequest) request, (HttpServletResponse)response );
7. chain.doFilter( wrappedSavedRequest == **null** ? request : wrappedSavedRequest, response );
8. }

这个方法中很明显继续下面拦截处理的request已经不是之前的那个request的对象了而是被替换成了继承自HttpServletRequestWrapper的SavedRequestAwareWrapper。

这时我们已经清楚的发现新的request已经带上了我们被拦截前的request中保存的请求数据，而在登录步骤提交时并不存在这些数据。