# Filter

Spring Security 的底层是通过一系列的 Filter 来管理的，每个 Filter 都有其自身的功能，而且各个 Filter 在功能上还有关联关系，所以它们的顺序也是非常重要的。

## Filter 顺序

Spring Security 已经定义了一些 Filter，不管实际应用中你用到了哪些，它们应当保持如下顺序。

1. ChannelProcessingFilter，如果你访问的 channel 错了，那首先就会在 channel 之间进行跳转，如 http 变为 https。
2. SecurityContextPersistenceFilter，这样的话在一开始进行 request 的时候就可以在 SecurityContextHolder 中建立一个 SecurityContext，然后在请求结束的时候，任何对 SecurityContext 的改变都可以被 copy 到 HttpSession。
3. ConcurrentSessionFilter，因为它需要使用 SecurityContextHolder 的功能，而且更新对应 session 的最后更新时间，以及通过 SessionRegistry 获取当前的 SessionInformation 以检查当前的 session 是否已经过期，过期则会调用 LogoutHandler。
4. 认证处理机制，如 UsernamePasswordAuthenticationFilter，CasAuthenticationFilter，BasicAuthenticationFilter 等，以至于 SecurityContextHolder 可以被更新为包含一个有效的 Authentication 请求。
5. SecurityContextHolderAwareRequestFilter，它将会把 HttpServletRequest 封装成一个继承自 HttpServletRequestWrapper 的 SecurityContextHolderAwareRequestWrapper，同时使用 SecurityContext 实现了 HttpServletRequest 中与安全相关的方法。
6. JaasApiIntegrationFilter，如果 SecurityContextHolder 中拥有的 Authentication 是一个 JaasAuthenticationToken，那么该 Filter 将使用包含在 JaasAuthenticationToken 中的 Subject 继续执行 FilterChain。
7. RememberMeAuthenticationFilter，如果之前的认证处理机制没有更新 SecurityContextHolder，并且用户请求包含了一个 Remember-Me 对应的 cookie，那么一个对应的 Authentication 将会设给 SecurityContextHolder。
8. AnonymousAuthenticationFilter，如果之前的认证机制都没有更新 SecurityContextHolder 拥有的 Authentication，那么一个 AnonymousAuthenticationToken 将会设给 SecurityContextHolder。
9. ExceptionTransactionFilter，用于处理在 FilterChain 范围内抛出的 AccessDeniedException 和 AuthenticationException，并把它们转换为对应的 Http 错误码返回或者对应的页面。
10. FilterSecurityInterceptor，保护 Web URI，并且在访问被拒绝时抛出异常。

## 添加 Filter 到 FilterChain

当我们在使用 NameSpace 时，Spring Security 是会自动为我们建立对应的 FilterChain 以及其中的 Filter。但有时我们可能需要添加我们自己的 Filter 到 FilterChain，又或者是因为某些特性需要自己显示的定义 Spring Security 已经为我们提供好的 Filter，然后再把它们添加到 FilterChain。使用 NameSpace 时添加 Filter 到 FilterChain 是通过 http 元素下的 custom-filter 元素来定义的。定义 custom-filter 时需要我们通过 ref 属性指定其对应关联的是哪个 Filter，此外还需要通过 position、before 或者 after 指定该 Filter 放置的位置。诚如在上一节《Filter 顺序》中所提到的那样，Spring Security 对 FilterChain 中 Filter 顺序是有严格的规定的。Spring Security 对那些内置的 Filter 都指定了一个别名，同时指定了它们的位置。我们在定义 custom-filter 的 position、before 和 after 时使用的值就是对应着这些别名所处的位置。如 position=”CAS_FILTER” 就表示将定义的 Filter 放在 CAS_FILTER 对应的那个位置，before=”CAS_FILTER” 就表示将定义的 Filter 放在 CAS_FILTER 之前，after=”CAS_FILTER” 就表示将定义的 Filter 放在 CAS_FILTER 之后。此外还有两个特殊的位置可以指定，FIRST 和 LAST，分别对应第一个和最后一个 Filter，如你想把定义好的 Filter 放在最后，则可以使用 after=”LAST”。

接下来我们来看一下 Spring Security 给我们定义好的 FilterChain 中 Filter 对应的位置顺序、它们的别名以及将触发自动添加到 FilterChain 的元素或属性定义。下面的定义是按顺序的。

| 别名 | Filter 类 | 对应元素或属性 |
|---|---|---|
| CHANNEL_FILTER | ChannelProcessingFilter | http/intercept-url@requires-chanr |
| SECURITY_CONTEXT_FILTER | SecurityContextPersistenceFilter | http |
| CONCURRENT_SESSION_FILTER | ConcurrentSessionFilter | http/session-management/concu control |
| LOGOUT_FILTER | LogoutFilter | http/logout |
| X509_FILTER | X509AuthenticationFilter | http/x509 |
| PRE_AUTH_FILTER | AstractPreAuthenticatedProcessingFilter 的子类 | 无 |
| CAS_FILTER | CasAuthenticationFilter | 无 |
| FORM_LOGIN_FILTER | UsernamePasswordAuthenticationFilter | http/form-login |
| BASIC_AUTH_FILTER | BasicAuthenticationFilter | http/http-basic |
| SERVLET_API_SUPPORT_FILTER | SecurityContextHolderAwareRequestFilter | http@servlet-api-p |
| JAAS_API_SUPPORT_FILTER | JaasApiIntegrationFilter | http@jaas-api-prov |
| REMEMBER_ME_FILTER | RememberMeAuthenticationFilter | http/remember-me |
| ANONYMOUS_FILTER | AnonymousAuthenticationFilter | http/anonymous |
| SESSION_MANAGEMENT_FILTER | SessionManagementFilter | http/session-manag |
| EXCEPTION_TRANSLATION_FILTER | ExceptionTranslationFilter | http |
| FILTER_SECURITY_INTERCEPTOR | FilterSecurityInterceptor | http |
| SWITCH_USER_FILTER | SwitchUserFilter | 无 |

## DelegatingFilterProxy

可能你会觉得奇怪，我们在 web 应用中使用 Spring Security 时只在 web.xml 文件中定义了如下这样一个 Filter，为什么你会说是一系列的 Filter 呢？

```
<filter>
   <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
   <filter-name>springSecurityFilterChain</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

而且如果你不在 web.xml 文件声明要使用的 Filter，那么 Servlet 容器将不会发现它们，它们又怎么发生作用呢？这就是主题配置中 DelegatingFilterProxy 的作用了。

DelegatingFilterProxy 是 Spring 中定义的一个 Filter 实现类，其作用是代理真正的 Filter 实现类，也就是说在调用 DelegatingFilterProxy 的 doFilter() 方法时实际上调用的是其代理 Filter 的 doFilter() 方法。其代理 Filter 必须是一个 Spring bean 对象，所以使用 DelegatingFilterProxy 的好处就是其代理 Filter 类可以使用 Spring 的依赖注入机制方便自由的使用 ApplicationContext 中的 bean。那么 DelegatingFilterProxy 如何知道其所代理的 Filter 是哪个呢？这是通过其自身的一个叫 targetBeanName 的属性来确定的，通过该名称，DelegatingFilterProxy 可以从 WebApplicationContext 中获取指定的 bean 作为代理对象。该属性可以通过在 web.xml 中定义 DelegatingFilterProxy 时通过 init-param 来指定，如果未指定的话将默认取其在 web.xml 中声明时定义的名称。

```
<filter>
   <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
   <filter-name>springSecurityFilterChain</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

在上述配置中，DelegatingFilterProxy 代理的就是名为 SpringSecurityFilterChain 的 Filter。

需要注意的是被代理的 Filter 的初始化方法 init() 和销毁方法 destroy() 默认是不会被执行的。通过设置 DelegatingFilterProxy 的 targetFilterLifecycle 属性为 true，可以使被代理 Filter 与 DelegatingFilterProxy 具有同样的生命周期。

## FilterChainProxy

Spring Security 底层是通过一系列的 Filter 来工作的，每个 Filter 都有其各自的功能，而且各个 Filter 之间还有关联关系，所以它们的组合顺序也是非常重要的。

使用 Spring Security 时，DelegatingFilterProxy 代理的就是一个 FilterChainProxy。一个 FilterChainProxy 中可以包含有多个 FilterChain，但是某个请求只会对应一个 FilterChain，而一个 FilterChain 中又可以包含有多个 Filter。当我们使用基于 Spring Security 的 NameSpace 进行配置时，系统会自动为我们注册一个名为 springSecurityFilterChain 类型为 FilterChainProxy 的 bean（这也是为什么我们在使用 SpringSecurity 时需要在 web.xml 中声明一个 name 为 springSecurityFilterChain 类型为 DelegatingFilterProxy 的 Filter 了。），而且每一个 http 元素的定义都将拥有自己的 FilterChain，而 FilterChain 中所拥有的 Filter 则会根据定义的服务自动增减。所以我们不需要显示的再定义这些 Filter 对应的 bean 了，除非你想实现自己的逻辑，又或者你想定义的某个属性 NameSpace 没有提供对应支持等。

Spring security 允许我们在配置文件中配置多个 http 元素，以针对不同形式的 URL 使用不同的安全控制。Spring Security 将会为每一个 http 元素创建对应的 FilterChain，同时按照它们的声明顺序加入到 FilterChainProxy。所以当我们同时定义多个 http 元素时要确保将更具有特性的 URL 配置在前。

```
   <security:http pattern="/login*.jsp*" security="none"/>
   <!-- 对 http 元素的 pattern 属性指定匹配的 http 对应的 FilterChain 将匹配哪些 URL，如未指定将匹配所有的请求 -->
   <security:http pattern="/admin/**">
      <security:intercept-url pattern="/**" access="ROLE_ADMIN"/>
   </security:http>
   <security:http>
      <security:intercept-url pattern="/**" access="ROLE_USER"/>
   </security:http>
```

需要注意的是 http 拥有一个匹配 URL 的 pattern，未指定时表示匹配所有的请求，其下的子元素 intercept-url 也有一个匹配 URL 的 pattern，该 pattern 是在 http 元素对应 pattern 基础上的，也就是说一个请求必须先满足 http 对应的 pattern 才有可能满足其下 intercept-url 对应的 pattern。

## Spring Security 定义好的核心 Filter

通过前面的介绍我们知道 Spring Security 是通过 Filter 来工作的，为保证 Spring Security 的顺利运行，其内部实现了一系列的 Filter。这其中有几个是在使用 Spring Security 的 Web 应用中必定会用到的。接下来我们来简要的介绍一下 FilterSecurityInterceptor、ExceptionTranslationFilter、SecurityContextPersistenceFilter 和 UsernamePasswordAuthenticationFilter。在我们使用 http 元素时前三者会自动添加到对应的 FilterChain 中，当我们使用了 form-login 元素时 UsernamePasswordAuthenticationFilter 也会自动添加到 FilterChain 中。所以我们在利用 custom-filter 往 FilterChain 中添加自己定义的这些 Filter 时需要注意它们的位置。

## FilterSecurityInterceptor

FilterSecurityInterceptor 是用于保护 Http 资源的，它需要一个 AccessDecisionManager 和一个 AuthenticationManager 的引用。它会从 SecurityContextHolder 获取 Authentication，然后通过 SecurityMetadataSource 可以得知当前请求是否在请求受保护的资源。对于请求那些受保护的资源，如果 Authentication.isAuthenticated()返回 false 或者 FilterSecurityInterceptor 的 alwaysReauthenticate 属性为 true，那么将会使用其引用的 AuthenticationManager 再认证一次，认证之后再使用认证后的 Authentication 替换 SecurityContextHolder 中拥有的那个。然后就是利用 AccessDecisionManager 进行权限的检查。

我们在使用基于 NameSpace 的配置时所配置的 intercept-url 就会跟 FilterChain 内部的 FilterSecurityInterceptor 绑定。如果要自己定义 FilterSecurityInterceptor 对应的 bean，那么该 bean 定义大致如下所示：

```
   <bean id="filterSecurityInterceptor"
      class="org.springframework.security.web.access.intercept.FilterSecurityInterceptor">
      <property name="authenticationManager" ref="authenticationManager" />
      <property name="accessDecisionManager" ref="accessDecisionManager" />
      <property name="securityMetadataSource">
         <security:filter-security-metadata-source>
            <security:intercept-url pattern="/admin/**" access="ROLE_ADMIN" />
            <security:intercept-url pattern="/**" access="ROLE_USER,ROLE_ADMIN" />
         </security:filter-security-metadata-source>
      </property>
   </bean>
```

filter-security-metadata-source 用于配置其 securityMetadataSource 属性。intercept-url 用于配置需要拦截的 URL 与对应的权限关系。

## ExceptionTranslationFilter

通过前面的介绍我们知道在 Spring Security 的 Filter 链表中 ExceptionTranslationFilter 就放在 FilterSecurityInterceptor 的前面。而 ExceptionTranslationFilter 是捕获来自 FilterChain 的异常，并对这些异常做处理。ExceptionTranslationFilter 能够捕获来自 FilterChain 所有的异常，但是它只会处理两类异常，AuthenticationException 和 AccessDeniedException，其它的异常它会继续抛出。如果捕获到的是 AuthenticationException，那么将会使用其对应的 AuthenticationEntryPoint 的 commence()处理。如果捕获的异常是一个 AccessDeniedException，那么将视当前访问的用户是否已经登录认证做不同的处理，如果未登录，则会使用关联的 AuthenticationEntryPoint 的 commence()方法进行处理，否则将使用关联的 AccessDeniedHandler 的 handle()方法进行处理。

AuthenticationEntryPoint 是在用户没有登录时用于引导用户进行登录认证的，在实际应用中应根据具体的认证机制选择对应的 AuthenticationEntryPoint。

AccessDeniedHandler 用于在用户已经登录了，但是访问了其自身没有权限的资源时做出对应的处理。ExceptionTranslationFilter 拥有的 AccessDeniedHandler 默认是 AccessDeniedHandlerImpl，其会返回一个 403 错误码到客户端。我们可以通过显示的配置 AccessDeniedHandlerImpl，同时给其指定一个 errorPage 使其可以返回对应的错误页面。当然我们也可以实现自己的 AccessDeniedHandler。

```
   <bean id="exceptionTranslationFilter"
         class="org.springframework.security.web.access.ExceptionTranslationFilter">
      <property name="authenticationEntryPoint">
         <bean class="org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint">
            <property name="loginFormUrl" value="/login.jsp" />
         </bean>
      </property>
      <property name="accessDeniedHandler">
         <bean class="org.springframework.security.web.access.AccessDeniedHandlerImpl">
            <property name="errorPage" value="/access_denied.jsp" />
         </bean>
      </property>
   </bean>
```

在上述配置中我们指定了 AccessDeniedHandler 为 AccessDeniedHandlerImpl，同时为其指定了 errorPage，这样发生 AccessDeniedException 后将转到对应的 errorPage 上。指定了 AuthenticationEntryPoint 为使用表单登录的 LoginUrlAuthenticationEntryPoint。此外，需要注意的是如果该 filter 是作为自定义 filter 加入到由 NameSpace 自动建立的 FilterChain 中时需把它放在内置的 ExceptionTranslationFilter 后面，否则异常都将被内置的 ExceptionTranslationFilter 所捕获。

```
   <security:http>
      <security:form-login login-page="/login.jsp"
         username-parameter="username" password-parameter="password"
         login-processing-url="/login.do" />
      <!-- 退出登录时删除 session 对应的 cookie -->
      <security:logout delete-cookies="JSESSIONID" />
      <!-- 登录页面应当是不需要认证的 -->
      <security:intercept-url pattern="/login*.jsp*"
         access="IS_AUTHENTICATED_ANONYMOUSLY" />
      <security:intercept-url pattern="/**" access="ROLE_USER" />
      <security:custom-filter ref="exceptionTranslationFilter" after="EXCEPTION_TRANSLATION_FILTER"/>
   </security:http>
```

在捕获到 AuthenticationException 之后，调用 AuthenticationEntryPoint 的 commence() 方法引导用户登录之前，ExceptionTranslationFilter 还做了一件事，那就是使用 RequestCache 将当前 HttpServletRequest 的信息保存起来，以至于用户成功登录后需要跳转到之前的页面时可以获取到这些信息，然后继续之前的请求，比如用户可能在未登录的情况下发表评论，待用户提交评论的时候就会将包含评论信息的当前请求保存起来，同时引导用户进行登录认证，待用户成功登录后再利用原来的 request 包含的信息继续之前的请求，即继续提交评论，所以待用户登录成功后我们通常看到的是用户成功提交了评论之后的页面。Spring Security 默认使用的 RequestCache 是 HttpSessionRequestCache，其会将 HttpServletRequest 相关信息封装为一个 SavedRequest 保存在 HttpSession 中。

## SecurityContextPersistenceFilter

SecurityContextPersistenceFilter 会在请求开始时从配置好的 SecurityContextRepository 中获取 SecurityContext，然后把它设置给 SecurityContextHolder。在请求完成后将 SecurityContextHolder 持有的 SecurityContext 再保存到配置好的 SecurityContextRepository，同时清除 SecurityContextHolder 所持有的 SecurityContext。在使用 NameSpace 时，Spring Security 默认会给 SecurityContextPersistenceFilter 的 SecurityContextRepository 设置一个 HttpSessionSecurityContextRepository，其会将 SecurityContext 保存在 HttpSession 中。此外 HttpSessionSecurityContextRepository 有一个很重要的属性 allowSessionCreation，默认为 true。这样需要把 SecurityContext 保存在 session 中时，如果不存在 session，可以自动创建一个。也可以把它设置为 false，这样在请求结束后如果没有可用的 session 就不会保存 SecurityContext 到 session 了。SecurityContextRepository 还有一个空实现，NullSecurityContextRepository，如果在请求完成后不想保存 SecurityContext 也可以使用它。

这里再补充说明一点为什么 SecurityContextPersistenceFilter 在请求完成后需要清除 SecurityContextHolder 的 SecurityContext。SecurityContextHolder 在设置和保存 SecurityContext 都是使用的静态方法，具体操作是由其所持有的 SecurityContextHolderStrategy 完成的。默认使用的是基于线程变量的实现，即 SecurityContext 是存放在 ThreadLocal 里面的，这样各个独立的请求都将拥有自己的 SecurityContext。在请求完成后清除 SecurityContextHolder 中的 SucurityContext 就是清除 ThreadLocal，Servlet 容器一般都有自己的线程池，这可以避免 Servlet 容器下一次分发线程时线程中还包含 SecurityContext 变量，从而引起不必要的错误。

下面是一个 SecurityContextPersistenceFilter 的简单配置。

```
   <bean id="securityContextPersistenceFilter"
   class="org.springframework.security.web.context.SecurityContextPersistenceFilter">
      <property name='securityContextRepository'>
         <bean
         class='org.springframework.security.web.context.HttpSessionSecurityContextRepositor
y'>
            <property name='allowSessionCreation' value='false' />
         </bean>
      </property>
   </bean>
```

## UsernamePasswordAuthenticationFilter

UsernamePasswordAuthenticationFilter 用于处理来自表单提交的认证。该表单必须提供对应的用户名和密码，对应的参数名默认为 j_username 和 j_password。如果不想使用默认的参数名，可以通过 UsernamePasswordAuthenticationFilter 的 usernameParameter 和 passwordParameter 进行指定。表单的提交路径默认是 “j_spring_security_check”，也可以通过 UsernamePasswordAuthenticationFilter 的 filterProcessesUrl 进行指定。通过属性 postOnly 可以指定只允许登录表单进行 post 请求，默认是 true。其内部还有登录成功或失败后进行处理的 AuthenticationSuccessHandler 和 AuthenticationFailureHandler，这些都可以根据需求做相关改变。此外，它还需要一个 AuthenticationManager 的引用进行认证，这个是没有默认配置的。

```
   <bean id="authenticationFilter"
      class="org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilte
r">
      <property name="authenticationManager" ref="authenticationManager" />
      <property name="usernameParameter" value="username"/>
      <property name="passwordParameter" value="password"/>
      <property name="filterProcessesUrl" value="/login.do" />
   </bean>
```

如果要在 http 元素定义中使用上述 AuthenticationFilter 定义，那么完整的配置应该类似于如下这样子。

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:security="http://www.springframework.org/schema/security"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
          http://www.springframework.org/schema/security
          http://www.springframework.org/schema/security/spring-security-3.1.xsd">
   <!-- entry-point-ref 指定登录入口 -->
   <security:http entry-point-ref="authEntryPoint">
      <security:logout delete-cookies="JSESSIONID" />
      <security:intercept-url pattern="/login*.jsp*"
         access="IS_AUTHENTICATED_ANONYMOUSLY" />
      <security:intercept-url pattern="/**" access="ROLE_USER" />
      <!-- 添加自己定义的 AuthenticationFilter 到 FilterChain 的 FORM_LOGIN_FILTER 位置 -->
      <security:custom-filter ref="authenticationFilter" position="FORM_LOGIN_FILTER"/>
   </security:http>
   <!-- AuthenticationEntryPoint，引导用户进行登录 -->
   <bean id="authEntryPoint" class="org.springframework.security.web.authentication.LoginUrlAu
thenticationEntryPoint">
      <property name="loginFormUrl" value="/login.jsp"/>
   </bean>
   <!-- 认证过滤器 -->
   <bean id="authenticationFilter"
   class="org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilte
r">
      <property name="authenticationManager" ref="authenticationManager" />
      <property name="usernameParameter" value="username"/>
      <property name="passwordParameter" value="password"/>
      <property name="filterProcessesUrl" value="/login.do" />
   </bean>

   <security:authentication-manager alias="authenticationManager">
      <security:authentication-provider
         user-service-ref="userDetailsService">
         <security:password-encoder hash="md5"
            base64="true">
            <security:salt-source user-property="username" />
         </security:password-encoder>
      </security:authentication-provider>
   </security:authentication-manager>

   <bean id="userDetailsService"
      class="org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl">
      <property name="dataSource" ref="dataSource" />
   </bean>

</beans>
```

上一篇: intercept-url配... (/project/spring-security/intercept-url.html)

下一篇: 退出登录 logout (/project/spring-security/logout.html)

极客学院
jikexueyuan.com
(//www.jikexueyuan.com)



登录 (//passport.jikexueyuan.com/sso/login) | 注册 (//passport.jikexueyuan.com/sso/reg_phone)

只有登录了才能参与评论，快 登录 (http://passport.jikexueyuan.com/sso/login) ！如果你还没有账号你可以 注册 (http://passport.jikexueyuan.com/sso/reg_phone) 一个账号。