Setup spring security

Reference: <http://websystique.com/spring-security/spring-security-4-hello-world-annotation-xml-example/>

Let’s assume you initially want to get up and running as quickly as possible and add authentication support and access control to an existing web application, with a few test logins. Then we’ll look at how to change over to authenticating against a database or other security repository. In later sections we’ll introduce more advanced namespace configuration options(xml file configuration). Let’s start from the xml configuration and then annotation approach. I personally prefer xml approach, but the annotation way becomes more and more fashion now a day.

Spring security provides multiple ways to configure it.

. springSecurityInitialization without spring context

. springSecurityInitialization with spring MVC

Here, I gonna focus on xml namespace strategy.

Xml namespace approach injects the beans into spring context by defining the related the security xml elements. Below are the elements security namespace has:

. Design of the spring security classes

The Spring security framework is implemented on many classes. They can be divided up into the following areas. Understanding the means of the classes may help us to customize it.

* Web/HTTP Security - the most complex part. Sets up the filters and related service beans used to apply the framework authentication mechanisms, to secure URLs, render login and error pages and much more. In another word, it works as the proxy.
* Business Object (Method) Security - options for securing the service layer.
* AuthenticationManager - handles authentication requests from other parts of the framework. AuthenticationManager contains AuthenticationProvider and UserService classes.
* AccessDecisionManager - provides access decisions for web and method security. A default one will be registered, but you can also choose to use a custom one, declared using normal Spring bean syntax.
* AuthenticationProviders - mechanisms against which the authentication manager authenticates users. The namespace provides supports for several standard options and also a means of adding custom beans declared using a traditional syntax.
* UserDetailsService - closely related to authentication providers, but often also required by other beans.

### . web.xml Configuration

The first thing you need to do is add the following filter declaration to your web.xml file:

<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>

This provides a hook into the Spring Security web infrastructure. DelegatingFilterProxy is a Spring Framework class which delegates to a filter implementation which is defined as a Spring bean in your application context. In this case, the bean is named "springSecurityFilterChain", which is an internal infrastructure bean created by the namespace to handle web security. Note that you should not use this bean name yourself. Once you’ve added this to your web.xml, you’re ready to start editing your application context file. Web security services are configured using the <http> element.

Simple example of Spring security configuration

<http>

<intercept-url pattern="/\*\*" access="hasRole('USER')" />

<form-login />

<logout />

</http>

<http> class composes many filter classes. It conceals the complexity of the filters like the order of filters etc.

<authentication-manager>

<authentication-provider>

<user-service>

<user name="jimi" password="jimispassword" authorities="ROLE\_USER, ROLE\_ADMIN" />

<user name="bob" password="bobspassword" authorities="ROLE\_USER" />

</user-service>

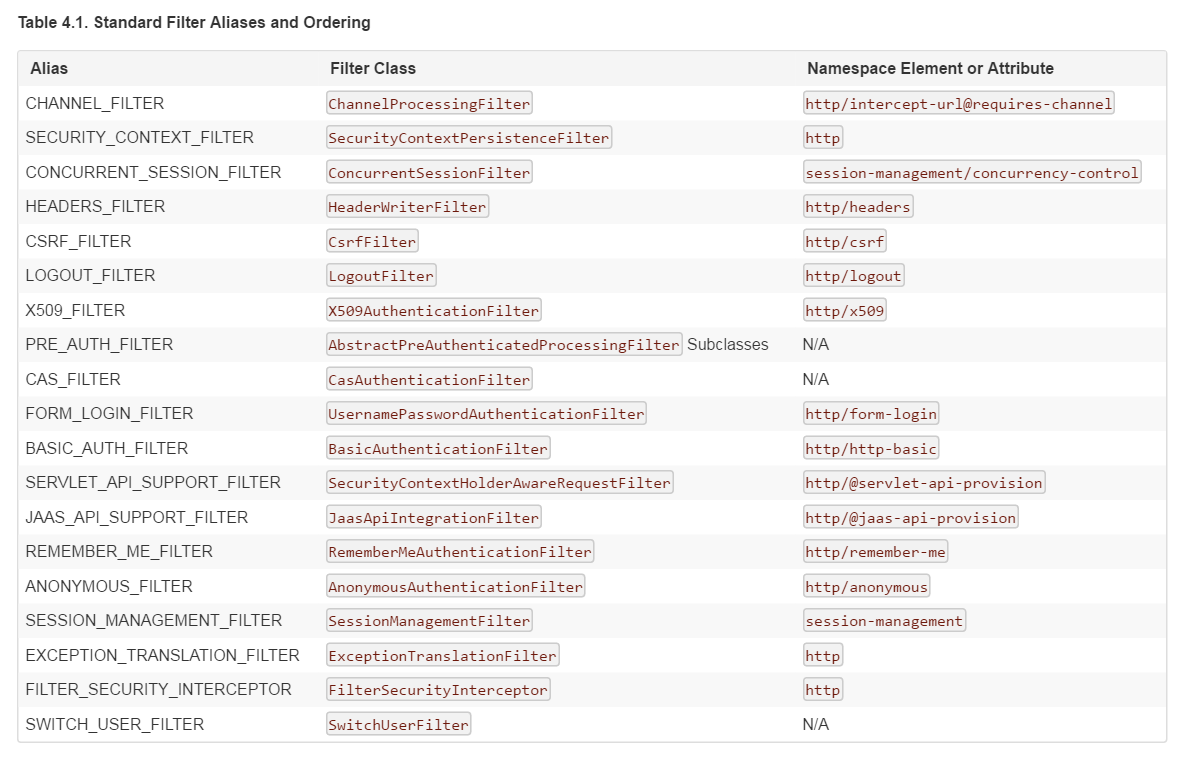
</authentication-provider>

</authentication-manager>

If you are familiar with pre-namespace versions of the framework, you can probably already guess roughly what’s going on here. The <http> element is responsible for creating a FilterChainProxy and the filter beans which it uses. Common problems like incorrect filter ordering are no longer an issue as the filter positions are predefined.

The <authentication-provider> element creates a DaoAuthenticationProvider bean and the <user-service> element creates an InMemoryDaoImpl. All authentication-provider elements must be children of the <authentication-manager> element, which creates a ProviderManager and registers the authentication providers with it. You can find more detailed information on the beans that are created in the [namespace appendix](http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#appendix-namespace). It’s worth cross-checking this if you want to start understanding what the important classes in the framework are and how they are used, particularly if you want to customize things later.

Basically, Spring security is implemented by filters. SpringSecurityProxyChain is the filter of integrating other filters. In the core filter, it can forward to other filters which are ordered. These are the default filters spring security provided.



You can add your own filter to the stack, using the custom-filter element and one of these names to specify the position your filter should appear at:

<http>

<custom-filter position="FORM\_LOGIN\_FILTER" ref="myFilter" />

</http>

<beans:bean id="myFilter" class="com.mycompany.MySpecialAuthenticationFilter"/>

You can also use the after or before attributes if you want your filter to be inserted before or after another filter in the stack. The names "FIRST" and "LAST" can be used with the position attribute to indicate that you want your filter to appear before or after the entire stack, respectively.

------Summary

Till now, what we introduced are about basic security function of spring security provides. It automatically wired some default behaviors into the workflow like default login form, error page etc. it is the great start point for jsp or other server-side UI solution. What we need to do is just redefine login-form in <http> tag and define our own AuthenticationManagerProvider, UserDetailService or add our own filter before/after/position(replace) default filters(see the table above listed the default filters by order— filter invocation order is important) etc. that’s it. However, what if we don’t want to use basic spring security function, we like to use the traditional java bean to manage the security. Or our UI is implemented by angularjs or other javascript solution framework. The security should be managed by UI itself. The spring security re-direction doesn’t make sense the requirement. In order to customize spring security, we have to understand <http> tag. (see <http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#tech-intro-auth-entry-point> 41. The security namespace).

**Must Read**

How spring security work

We know spring security is a very complex and flexible framework. It is very simple if just configured for form based application. The basic configuration can handle everything. We just need to redefine to override the default login and error pages. If any special requirement, just add a customized filter at some points(see filter spring security default filter table). but how about we want spring security to work for restful api? We expect the return is always json object or string instead of default html page. below link answered most of difficult questions. It also gave the sample code to track how the default filters created by spring security.

<http://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works>

|  |
| --- |
| I realize that Spring security build on chain of filters, which will intercept the request, detect (absence of) authentication, redirect to authentication entry point or pass the request to authorization service, and eventually let the request either hit the servlet or throw security exception (unauthenticated or unauthorized). DelegatingFitlerProxy glues these filters together. To perform their tasks, these filter access services such as UserDetailsService and AuthenticationManager.  Key filters in the chain are (in the order)   * SecurityContextPersistenceFilter (restores Authentication from JSESSIONID) * UsernamePasswordAuthenticationFilter (performs authentication) * ExceptionTranslationFilter (catch security exceptions from FilterSecurityInterceptor) * FilterSecurityInterceptor (may throw authentication and authorization exceptions)   I'm confused how these filters are used. Is it that for the spring provided form-login, UsernamePasswordAuthenticationFilter is only used for /login, and latter filters are not? Does the form-login namespace element auto-configure these filters? Does every request (authenticated or not) reach FilterSecurityInterceptor for non-login url?  **What if I want to secure my REST API with JWT-token, which is retrieved from login?** I must configure two namespace configuration http tags, rights? Other one for /login with UsernamePasswordAuthenticationFilter, and another one for REST url's, with custom JwtAuthenticationFilter.  Does configuring two http elements create two springSecurityFitlerChains? Is UsernamePasswordAuthenticationFilter turned off by default, until I declare form-login? How do I replace SecurityContextPersistenceFilter with one, which will obtain Authentication from existing JWT-token rather than JSESSIONID?  [spring](http://stackoverflow.com/questions/tagged/spring) [authentication](http://stackoverflow.com/questions/tagged/authentication) [spring-security](http://stackoverflow.com/questions/tagged/spring-security) [filter](http://stackoverflow.com/questions/tagged/filter) [jwt](http://stackoverflow.com/questions/tagged/jwt" \o "show questions tagged 'jwt') |
|  | add a comment |

## **2 Answers**

[active](http://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works?answertab=active#tab-top)[oldest](http://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works?answertab=oldest#tab-top)[votes](http://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works?answertab=votes#tab-top)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| up vote9down voteaccepted | The Spring security filter chain is a very complex and flexible engine.  Key filters in the chain are (in the order)   * SecurityContextPersistenceFilter (restores Authentication from JSESSIONID) * UsernamePasswordAuthenticationFilter (performs authentication) * ExceptionTranslationFilter (catch security exceptions from FilterSecurityInterceptor) * FilterSecurityInterceptor (may throw authentication and authorization exceptions)   Looking at the [current stable release 4.2.1 documentation](http://docs.spring.io/spring-security/site/docs/4.2.1.RELEASE/reference/htmlsingle/), section [13.3 Filter Ordering](http://docs.spring.io/spring-security/site/docs/4.2.1.RELEASE/reference/htmlsingle/#filter-ordering) you could see the whole filter chain's filter organization:  13.3 Filter Ordering  The order that filters are defined in the chain is very important. Irrespective of which filters you are actually using, the order should be as follows:   1. **ChannelProcessingFilter**, because it might need to redirect to a different protocol 2. **SecurityContextPersistenceFilter**, so a SecurityContext can be set up in the SecurityContextHolder at the beginning of a web request, and any changes to the SecurityContext can be copied to the HttpSession when the web request ends (ready for use with the next web request) 3. **ConcurrentSessionFilter**, because it uses the SecurityContextHolder functionality and needs to update the SessionRegistry to reflect ongoing requests from the principal 4. Authentication processing mechanisms - **UsernamePasswordAuthenticationFilter**, **CasAuthenticationFilter**, **BasicAuthenticationFilter** etc - so that the SecurityContextHolder can be modified to contain a valid Authentication request token 5. The **SecurityContextHolderAwareRequestFilter**, if you are using it to install a Spring Security aware HttpServletRequestWrapper into your servlet container 6. The **JaasApiIntegrationFilter**, if a JaasAuthenticationToken is in the SecurityContextHolder this will process the FilterChain as the Subject in the JaasAuthenticationToken 7. **RememberMeAuthenticationFilter**, so that if no earlier authentication processing mechanism updated the SecurityContextHolder, and the request presents a cookie that enables remember-me services to take place, a suitable remembered Authentication object will be put there 8. **AnonymousAuthenticationFilter**, so that if no earlier authentication processing mechanism updated the SecurityContextHolder, an anonymous Authentication object will be put there 9. **ExceptionTranslationFilter**, to catch any Spring Security exceptions so that either an HTTP error response can be returned or an appropriate AuthenticationEntryPoint can be launched 10. **FilterSecurityInterceptor**, to protect web URIs and raise exceptions when access is denied   Now, I'll try to go on by your questions one by one:  I'm confused how these filters are used. Is it that for the spring provided form-login, UsernamePasswordAuthenticationFilter is only used for /login, and latter filters are not? Does the form-login namespace element auto-configure these filters? Does every request (authenticated or not) reach FilterSecurityInterceptor for non-login url?  Once you are configuring a <security-http> section, for each one you must at least provide one authentication mechanism. This must be one of the filters which match group 4 in the 13.3 Filter Ordering section from the Spring Security documentation I've just referenced.  This is the minimum valid security:http element which can be configured:  <security:http authentication-manager-ref="mainAuthenticationManager"  entry-point-ref="serviceAccessDeniedHandler">  <security:intercept-url pattern="/sectest/zone1/\*\*" access="hasRole('ROLE\_ADMIN')"/>  </security:http>  Just doing it, these filters are configured in the filter chain proxy:  {  "1": "org.springframework.security.web.context.SecurityContextPersistenceFilter",  "2": "org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter",  "3": "org.springframework.security.web.header.HeaderWriterFilter",  "4": "org.springframework.security.web.csrf.CsrfFilter",  "5": "org.springframework.security.web.savedrequest.RequestCacheAwareFilter",  "6": "org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter",  "7": "org.springframework.security.web.authentication.AnonymousAuthenticationFilter",  "8": "org.springframework.security.web.session.SessionManagementFilter",  "9": "org.springframework.security.web.access.ExceptionTranslationFilter",  "10": "org.springframework.security.web.access.intercept.FilterSecurityInterceptor"  }  Note: I get them by creating a simple RestController which @Autowires the FilterChainProxy and returns it's contents:  @Autowired  private FilterChainProxy filterChainProxy;  @Override  @RequestMapping("/filterChain")  public @ResponseBody Map<Integer, Map<Integer, String>> getSecurityFilterChainProxy(){  return this.getSecurityFilterChainProxy();  }  public Map<Integer, Map<Integer, String>> getSecurityFilterChainProxy(){  Map<Integer, Map<Integer, String>> filterChains= new HashMap<Integer, Map<Integer, String>>();  int i = 1;  for(SecurityFilterChain secfc : this.filterChainProxy.getFilterChains()){  //filters.put(i++, secfc.getClass().getName());  Map<Integer, String> filters = new HashMap<Integer, String>();  int j = 1;  for(Filter filter : secfc.getFilters()){  filters.put(j++, filter.getClass().getName());  }  filterChains.put(i++, filters);  }  return filterChains;  }  Here we could see that just by declaring the <security:http> element with one minimum configuration, all the default filters are included, but none of them is of a Authentication type (4th group in 13.3 Filter Ordering section). So it actually means that just by declaring the security:http element, the SecurityContextPersistenceFilter, the ExceptionTranslationFilter and the FilterSecurityInterceptor are auto-configured.  In fact, one authentication processing mechanism should be configured, and even security namespace beans processing claims for that, throwing an error during startup, but it can be bypassed adding an entry-point-ref attribute in <http:security>  If I add a basic <form-login> to the configuration, this way:  <security:http authentication-manager-ref="mainAuthenticationManager">  <security:intercept-url pattern="/sectest/zone1/\*\*" access="hasRole('ROLE\_ADMIN')"/>  <security:form-login />  </security:http>  Now, the filterChain will be like this:  {  "1": "org.springframework.security.web.context.SecurityContextPersistenceFilter",  "2": "org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter",  "3": "org.springframework.security.web.header.HeaderWriterFilter",  "4": "org.springframework.security.web.csrf.CsrfFilter",  "5": "org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter",  "6": "org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter",  "7": "org.springframework.security.web.savedrequest.RequestCacheAwareFilter",  "8": "org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter",  "9": "org.springframework.security.web.authentication.AnonymousAuthenticationFilter",  "10": "org.springframework.security.web.session.SessionManagementFilter",  "11": "org.springframework.security.web.access.ExceptionTranslationFilter",  "12": "org.springframework.security.web.access.intercept.FilterSecurityInterceptor"  }  Now, this two filters [org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter](http://docs.spring.io/spring-security/site/docs/4.2.1.RELEASE/reference/htmlsingle/#form-login-filter) and org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter are created and confired in the FilterChainProxy.  So, now, the questions:  Is it that for the spring provided form-login, UsernamePasswordAuthenticationFilter is only used for /login, and latter filters are not?  Yes, it is used to try to complete a login processing mechanism in case the request matches the UsernamePasswordAuthenticationFilter url. This url can be configured or even changed it's behaviour to match every request.  You could too have more than one Authentication processing mechanisms configured in the same FilterchainProxy (such as HttpBasic, CAS, etc).  Does the form-login namespace element auto-configure these filters?  No, the form-login element configures the UsernamePasswordAUthenticationFilter, and in case you don't provide a login-page url, it also configures the org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter, which ends in a simple autogenerated login page.  The other filters are auto-configured by default just by creating a <security:http> element with no security:"none" attribute.  Does every request (authenticated or not) reach FilterSecurityInterceptor for non-login url?  Every request should reach it, as it is the element which takes care of whether the request has the rights to reach the requested url. But some of the filters processed before might stop the filter chain processing just not calling FilterChain.doFilter(request, response);. For example, a CSRF filter might stop the filter chain processing if the request has not the csrf parameter.  What if I want to secure my REST API with JWT-token, which is retrieved from login? I must configure two namespace configuration http tags, rights? Other one for /login with UsernamePasswordAuthenticationFilter, and another one for REST url's, with custom JwtAuthenticationFilter.  No, you are not forced to do this way. You could declare both UsernamePasswordAuthenticationFilter and the JwtAuthenticationFilter in the same http element, but it depends on the concrete behaviour of each of this filters. Both approaches are possible, and which one to choose finnally depends on own preferences.  Does configuring two http elements create two springSecurityFitlerChains?  Yes, that's true  Is UsernamePasswordAuthenticationFilter turned off by default, until I declare form-login?  Yes, you could see it in the filters raised in each one of the configs I posted  How do I replace SecurityContextPersistenceFilter with one, which will obtain Authentication from existing JWT-token rather than JSESSIONID?  You could avoid SecurityContextPersistenceFilter, just configuring [session strategy](http://docs.spring.io/spring-security/site/docs/4.2.1.RELEASE/reference/htmlsingle/#nsa-web) in <http:element>. Just configure like this:  <security:http create-session="stateless" >  Or, In this case you could overwrite it with another filter, this way inside the <security:http>element:  <security:http ...>  <security:custom-filter ref="myCustomFilter" position="SECURITY\_CONTEXT\_FILTER"/>  </security:http>  <beans:bean id="myCustomFilter" class="com.xyz.myFilter" />  EDIT:  One question about "You could too have more than one Authentication processing mechanisms configured in the same FilterchainProxy". Will the latter overwrite the authentication performed by first one, if declaring multiple (Spring implementation) authentication filters? How this relates to having multiple authentication providers?  This finally depends on the implementation of each filter itself, but it's true the fact that the latter authentication filters at least are able to overwrite any prior authentication eventually made by preceding filters.  But this won't necesarily happen. I have some production cases in secured REST services where I use a kind of authorization token which can be provided both as a Http header or inside the request body. So I configure two filters which recover that token, in one case from the Http Header and the other from the request body of the own rest request. It's true the fact that if one http request provides that authentication token both as Http header and inside the request body, both filters will try to execute the authentication mechanism delegating it to the manager, but it could be easily avoided simply checking if the request is already authenticated just at the begining of the doFilter() method of each filter.  Having more than one authentication filter is related to having more than one authentication providers, but don't force it. In the case I exposed before, I have two authentication filter but I only have one authentication provider, as both of the filters create the same type of Authentication object so in both cases the authentication manager delegates it to the same provider.  And opposite to this, I too have a scenario where I publish just one UsernamePasswordAuthenticationFilter but the user credentials both can be contained in DB or LDAP, so I have two UsernamePasswordAuthenticationToken supporting providers, and the AuthenticationManager delegates any authentication attempt from the filter to the providers secuentially to validate the credentials.  So, I think it's clear that neither the amount of authentication filters determine the amount of authentication providers nor the amount of provider determine the amount of filters.  Also, documentation states SecurityContextPersistenceFilter is responsible of cleaning the SecurityContext, which is important due thread pooling. If I omit it or provide custom implementation, I have to implement the cleaning manually, right? Are there more similar gotcha's when customizing the chain?  I did not look carefully into this filter before, but after your last question I've been checking it's implementation, and as usually in Spring, nearly everything could be configured, extended or overwrited.  The [SecurityContextPersistenceFilter](https://github.com/spring-projects/spring-security/blob/master/web/src/main/java/org/springframework/security/web/context/SecurityContextPersistenceFilter.java) delegates in a [SecurityContextRepository](https://github.com/spring-projects/spring-security/blob/master/web/src/main/java/org/springframework/security/web/context/SecurityContextRepository.java) implementation the search for the SecurityContext. By default, a [HttpSessionSecurityContextRepository](https://github.com/spring-projects/spring-security/blob/master/web/src/main/java/org/springframework/security/web/context/HttpSessionSecurityContextRepository.java) is used, but this could be changed using one of the constructors of the filter. So it may be better to write an SecurityContextRepository which fits your needs and just configure it in the SecurityContextPersistenceFilter, trusting in it's proved behaviour rather than start making all from scratch.   |  |  |  | | --- | --- | --- | | [share](http://stackoverflow.com/a/41482134)[improve this answer](http://stackoverflow.com/posts/41482134/edit) | [edited Mar 18 at 16:43](http://stackoverflow.com/posts/41482134/revisions) | answered Jan 5 at 10:07  [[https://lh3.googleusercontent.com/-P7pYwNMdimY/AAAAAAAAAAI/AAAAAAAACoM/4DCcMoDIO5Q/photo.jpg?sz=32](http://stackoverflow.com/users/4190848/jlumietu)](http://stackoverflow.com/users/4190848/jlumietu)  [jlumietu](http://stackoverflow.com/users/4190848/jlumietu)  **1,502**1414 | |
|  | |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 2 |  | | This was enlightening explanation. One question about "You could too have more than one Authentication processing mechanisms configured in the same FilterchainProxy". Will the latter overwrite the authentication performed by first one, if declaring multiple (Spring implementation) authentication filters? How this relates to having multiple authentication providers? – [Tuomas Toivonen](http://stackoverflow.com/users/5705247/tuomas-toivonen" \o "1,296 reputation) [Jan 5 at 15:47](http://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works#comment70183666_41482134) | | |  |  | | --- | --- | |  |  | | Also, documentation states SecurityContextPersistenceFilter is responsible of cleaning the SecurityContext, which is important due thread pooling. If I omit it or provide custom implementation, I have to implement the cleaning manually, right? Are there more similar gotcha's when customizing the chain? – [Tuomas Toivonen](http://stackoverflow.com/users/5705247/tuomas-toivonen" \o "1,296 reputation)[Jan 5 at 16:04](http://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works#comment70184373_41482134) | | |  |  | | --- | --- | |  |  | | @TuomasToivonen I edited my answer after the questions in your last comments – [jlumietu](http://stackoverflow.com/users/4190848/jlumietu" \o "1,502 reputation) [Jan 7 at 0:01](http://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works#comment70238819_41482134) | | |  |  | | --- | --- | |  |  | | @jlumietu There is a missing quote in the java annotation next to *("/filterChain)*. Also where do you place this method? I have tried to add it in a controller and I have : No qualifying bean of type 'org.springframework.security.web.FilterChainProxy' available: expected at least 1 bean which qualifies as autowire candidate. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(req‌​uired=true)} – [BigDong](http://stackoverflow.com/users/2127277/bigdong) [Mar 18 at 15:37](http://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works#comment72856555_41482134) | | |  |  | | --- | --- | |  |  | | @BigDong make sure you have declared the SpringSecurityFilterChain in both web.xml or java webapp config and in your spring configuration. This code snippet must be included in a Controller, just as you did. And yes, you are wright about the missing quote – [jlumietu](http://stackoverflow.com/users/4190848/jlumietu" \o "1,502 reputation) [Mar 18 at 15:43](http://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works#comment72856646_41482134) |   [show **3** more comments](http://stackoverflow.com/questions/41480102/how-spring-security-filter-chain-works) |



## **Did you find this question interesting? Try our newsletter**

Sign up for our newsletter and get our top new questions delivered to your inbox ([see an example](https://stackexchange.com/newsletters/newsletter?site=stackoverflow.com)).

Top of Form



Bottom of Form

|  |  |
| --- | --- |
| up vote1down vote | UsernamePasswordAuthenticationFilter is only used for /login, and latter filters are not?  No, UsernamePasswordAuthenticationFilter extends AbstractAuthenticationProcessingFilter, and this contains a RequestMatcher, that means you can define your own processing url, this filter only handle the RequestMatcher matches the request url, the default processing url is /login.  Later filters can still handle the request, if the UsernamePasswordAuthenticationFilter executes chain.doFilter(request, response);.  More details about [core fitlers](http://docs.spring.io/spring-security/site/docs/3.0.x/reference/core-web-filters.html)  Does the form-login namespace element auto-configure these filters?  UsernamePasswordAuthenticationFilter is created by <form-login>, these are [Standard Filter Aliases and Ordering](http://docs.spring.io/spring-security/site/docs/3.1.7.RELEASE/reference/ns-config.html#filter-stack)  Does every request (authenticated or not) reach FilterSecurityInterceptor for non-login url?  It depends on whether the before fitlers are successful, but FilterSecurityInterceptor is the last fitler normally.  Does configuring two http elements create two springSecurityFitlerChains?  Yes, every fitlerChain has a RequestMatcher, if the RequestMatcher matches the request, the request will be handled by the fitlers in the fitler chain.  The default RequestMatcher matches all request if you don't config the pattern, or you can config the specific url (<http pattern="/rest/\*\*").  If you want to konw more about the fitlers, I think you can check source code in spring security. doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) |

Reference:

Spring Security

<http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#ns-config>