org.springframework.security.web

Class FilterChainProxy

java.lang.Object
  org.springframework.web.filter.GenericFilterBean
    org.springframework.security.web.FilterChainProxy

All Implemented Interfaces:

javax.servlet.Filter, Aware, BeanNameAware, DisposableBean, InitializingBean, EnvironmentAware, EnvironmentCapable, ServletContextAware

---

```
public class FilterChainProxy
extends GenericFilterBean
```

Delegates `Filter` requests to a list of Spring-managed filter beans. As of version 2.0, you shouldn't need to explicitly configure a `FilterChainProxy` bean in your application context unless you need very fine control over the filter chain contents. Most cases should be adequately covered by the default `<security:http />` namespace configuration options.

The `FilterChainProxy` is linked into the servlet container filter chain by adding a standard Spring `DelegatingFilterProxy` declaration in the application `web.xml` file.

Configuration

As of version 3.1, `FilterChainProxy` is configured using a list of `SecurityFilterChain` instances, each of which contains a `RequestMatcher` and a list of filters which should be applied to matching requests. Most applications will only contain a single filter chain, and if you are using the namespace, you don't have to set the chains explicitly. If you require finer-grained control, you can make use of the `<filter-chain>` namespace element. This defines a URI pattern and the list of filters (as comma-separated bean names) which should be applied to requests which match the pattern. An example configuration might look like this:

```
<bean id="myfilterChainProxy" class="org.springframework.security.util.FilterChainProxy">
    <constructor-arg>
        <util:list>
            <security:filter-chain pattern="/do/not/filter*" filters="none"/>
            <security:filter-chain pattern="/**" filters="filter1,filter2,filter3"/>
        </util:list>
    </constructor-arg>
</bean>
```

The names "filter1", "filter2", "filter3" should be the bean names of `Filter` instances defined in the application context. The order of the names defines the order in which the filters will be applied. As shown above, use of the value "none" for the "filters" can be used to exclude a request pattern from the security filter chain entirely. Please consult the security namespace schema file for a full list of available configuration options.

Request Handling

Each possible pattern that the `FilterChainProxy` should service must be entered. The first match for a given request will be used to define all of the `Filters` that apply to that request. This means you must put most specific matches at the top of the list, and ensure all `Filters` that should apply for a given matcher are entered against the respective entry. The `FilterChainProxy` will not iterate through the remainder of the map entries to locate additional `Filters`.

`FilterChainProxy` respects normal handling of `Filters` that elect not to call `Filter.doFilter(javax.servlet.ServletRequest, javax.servlet.ServletResponse, javax.servlet.FilterChain)`, in that the remainder of the original or `FilterChainProxy`-declared filter chain will not be called.

Request Firewalling

An `HttpFirewall` instance is used to validate incoming requests and create a wrapped request which provides consistent path values for matching against. See `DefaultHttpFirewall`, for more information on the type of attacks which the default implementation protects against. A custom implementation can be injected to provide stricter control over the request contents or if an application needs to support certain types of request which are rejected by default.

Note that this means that you must use the Spring Security filters in combination with a `FilterChainProxy` if you want this protection. Don't define them explicitly in your `web.xml` file.

`FilterChainProxy` will use the firewall instance to obtain both request and response objects which will be fed down the filter chain, so it is also possible to use this functionality to control the functionality of the response. When the request has passed through the security filter chain, the `reset` method will be called. With the default implementation this means that the original

values of `servletPath` and `pathInfo` will be returned thereafter, instead of the modified ones used for security pattern matching.

Since this additional wrapping functionality is performed by the `FilterChainProxy`, we don't recommend that you use multiple instances in the same filter chain. It shouldn't be considered purely as a utility for wrapping filter beans in a single `Filter` instance.

Filter Lifecycle

Note the `Filter` lifecycle mismatch between the servlet container and IoC container. As described in the `DelegatingFilterProxy` Javadocs, we recommend you allow the IoC container to manage the lifecycle instead of the servlet container. `FilterChainProxy` does not invoke the standard filter lifecycle methods on any filter beans that you add to the application context.

Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
static interface	**FilterChainProxy.FilterChainValidator**

Constructor Summary

Constructors

Constructor and Description
FilterChainProxy()
FilterChainProxy(**List**<**SecurityFilterChain**> filterChains)
FilterChainProxy(**SecurityFilterChain** chain)

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
void	**afterPropertiesSet**()
void	**doFilter**(javax.servlet.ServletRequest request, javax.servlet.ServletResponse response, javax.servlet.FilterChain chain)
List<**SecurityFilterChain**>	**getFilterChains**()
List<javax.servlet.Filter>	**getFilters**(**String** url) Convenience method, mainly for testing.
void	**setFilterChainValidator**(**FilterChainProxy.FilterChainValidator** filterChainValidator) Used (internally) to specify a validation strategy for the filters in each configured chain.
void	**setFirewall**(**HttpFirewall** firewall) Sets the "firewall" implementation which will be used to validate and wrap (or potentially reject) the incoming requests.
String	**toString**()

Methods inherited from class org.springframework.web.filter.GenericFilterBean

addRequiredProperty, createEnvironment, destroy, getEnvironment, getFilterConfig, getFilterName, getServletContext, init, initBeanWrapper, initFilterBean, setBeanName, setEnvironment, setServletContext

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

FilterChainProxy

```
public FilterChainProxy()
```

FilterChainProxy

```
public FilterChainProxy(SecurityFilterChain chain)
```

FilterChainProxy

```
public FilterChainProxy(List<SecurityFilterChain> filterChains)
```

Method Detail

afterPropertiesSet

```
public void afterPropertiesSet()
```

Specified by:

`afterPropertiesSet` in interface `InitializingBean`

Overrides:

`afterPropertiesSet` in class `GenericFilterBean`

doFilter

```
public void doFilter(javax.servlet.ServletRequest request,
                     javax.servlet.ServletResponse response,
                     javax.servlet.FilterChain chain)
              throws IOException,
                     javax.servlet.ServletException
```

Throws:

`IOException`

`javax.servlet.ServletException`

getFilters

```
public List<javax.servlet.Filter> getFilters(String url)
```

Convenience method, mainly for testing.

Parameters:

`url` - the URL

Returns:

matching filter list

getFilterChains

```
public List<SecurityFilterChain> getFilterChains()
```

Returns:

the list of SecurityFilterChains which will be matched against and applied to incoming requests.

setFilterChainValidator

```
public void setFilterChainValidator(FilterChainProxy.FilterChainValidator filterChainValidator)
```

Used (internally) to specify a validation strategy for the filters in each configured chain.

Parameters:

`filterChainValidator` - the validator instance which will be invoked on during initialization to check the FilterChainProxy instance.

setFirewall

```
public void setFirewall(HttpFirewall firewall)
```

Sets the "firewall" implementation which will be used to validate and wrap (or potentially reject) the incoming requests. The default implementation should be satisfactory for most requirements.

Parameters:

`firewall` -

toString

```
public String toString()
```

Overrides:

`toString` in class `Object`

OVERVIEW PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD