Introduction to Spring MVC HandlerInterceptor

Last modified: October 22, 2017

by baeldung (http://www.baeldung.com/author/baeldung/)

Spring (http://www.baeldung.com/category/spring/) +
Spring MVC (http://www.baeldung.com/category/spring-mvc/)

1. Introduction

In this tutorial we'll focus on understanding the Spring MVC *HandlerInterceptor* and how to use it correctly.

2. Spring MVC Handler

And in order to understand the interceptor, let's take a step back and look at the *HandlerMapping*. This maps a method to an URL, so that the *DispatcherServlet* will be able to invoke it when processing a request.

And the *DispatcherServlet* uses the *HandlerAdapter* to actually invoke the method.

Now that we understand the overall context – **this is where the handler interceptor comes in**. We'll use the *HandlerInterceptor* to perform actions before handling, after handling or after completion (when the view is rendered) of a request.

The interceptor can be used for cross-cutting concerns and to avoid repetitive handler code like: logging, changing globally used parameters in Spring model etc.

In next few sections that's exactly what we're going to be looking at – the differences between various interceptor implementations.

3. Maven Dependencies

In order to use *Interceptors*, you need to include the following section in a *dependencies* section of your *pom.xml* file:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.3.2.RELEASE</version>
</dependency>
```

Latest version can be found here (http://search.maven.org/#search%7Cga%7C1%7Ca%3A%22spring-web%22).

4. Spring Handler Interceptor

Interceptors working with the *HandlerMapping* on the framework must implement the *HandlerInterceptor* interface.

This interface contains three main methods:

- *prehandle()* – called before the actual handler is executed, but the view is not generated yet

- *postHandle()* – called after the handler is executed
- *afterCompletion()* – called after the complete request has finished and view was generated

These three methods provide flexibility to do all kinds of pre- and post-processing.

And a quick note – the main difference between *HandlerInterceptor* and *HandlerInterceptorAdapter* is that in the first one we need to override all three methods: *preHandle()*, *postHandle()* and *afterCompletion()*, whereas in the second we may implement only required methods.

A quick note before we go further – if you want to skip the theory and jump straight to examples, jump right into section 5.

Here's what a simple *preHandle()* implementation will look like:

```
@Override
public boolean preHandle(
  HttpServletRequest request,
  HttpServletResponse response,
  Object handler) throws Exception {
    // your code
    return true;
}
```

Notice the method returns a *boolean* value – which tells Spring if the request should be further processed by a handler (*true*) or not (*false*).

Next, we have an implementation of *postHandle()*:

```
@Override
public void postHandle(
  HttpServletRequest request,
  HttpServletResponse response,
  Object handler,
  ModelAndView modelAndView) throws Exception {
    // your code
}
```

This method is called immediately after the request is processed by *HandlerAdapter*, but before generating a view.

And it can of course be used in many ways – for example, we may add an avatar of a logged user into a model.

The final method we need to implement in the custom *HandlerInterceptor* implementation is *afterCompletion():*

```
@Override
public void afterCompletion(
  HttpServletRequest request,
  HttpServletResponse response,
  Object handler, Exception ex) {
    // your code
}
```

When the view is successfully generated, we can use this hook to do things like gather additional statistics related to the request.

A final note to remember is that a *HandlerInterceptor* is registered to the *DefaultAnnotationHandlerMapping* bean, which is responsible for applying interceptors to any class marked with a *@Controller* annotation. Moreover, you may specify any number of interceptors in your web application.

5. Custom Logger Interceptor

In this example we will focus on logging in our web application. First of all, our class needs to extend *HandlerInterceptorAdapter*:

```
public class LoggerInterceptor extends HandlerInterceptorAdapter {
    ...
}
```

We also need to enable logging in our interceptor:

```
private static Logger log = LoggerFactory.getLogger(LoggerInterceptor.class);
```

This allows Log4J to display logs, as well as indicate, which class is currently logging information to the specified output.

Next, let's focus on custom interceptor implementations:

5.1. Method *preHandle()*

This method is called before handling a request; it returns *true,* to allow the framework to send the request further to the handler method (or to the next interceptor). If the method returns *false*, Spring assumes that request has been handled and no further processing is needed.

We can use the hook to log information about the requests' parameters: where the request comes from, etc.

In our example, we are logging this info using a simple Log4J logger:

```java
@Override
public boolean preHandle(
  HttpServletRequest request,
  HttpServletResponse response,
  Object handler) throws Exception {

    log.info("[preHandle][" + request + "]" + "[" + request.getMethod()
      + "]" + request.getRequestURI() + getParameters(request));

    return true;
}
```

As we can see, we're logging some basic information about the request.

In case we run into a password here, we'll need to make sure we don't log that of course.

A simple option is to replace passwords, and any other sensitive type of data, with stars.

Here's a quick implementation of how that can be done:

```java
private String getParameters(HttpServletRequest request) {
    StringBuffer posted = new StringBuffer();
    Enumeration<?> e = request.getParameterNames();
    if (e != null) {
        posted.append("?");
    }
    while (e.hasMoreElements()) {
        if (posted.length() > 1) {
            posted.append("&");
        }
        String curr = (String) e.nextElement();
        posted.append(curr + "=");
        if (curr.contains("password")
          || curr.contains("pass")
          || curr.contains("pwd")) {
            posted.append("*****");
        } else {
            posted.append(request.getParameter(curr));
        }
    }
    String ip = request.getHeader("X-FORWARDED-FOR");
    String ipAddr = (ip == null) ? getRemoteAddr(request) : ip;
    if (ipAddr!=null && !ipAddr.equals("")) {
        posted.append("&_psip=" + ipAddr);
    }
    return posted.toString();
}
```

Finally, we're aiming to get the source IP address of the HTTP request.

Here's a simple implementation:

```java
private String getRemoteAddr(HttpServletRequest request) {
    String ipFromHeader = request.getHeader("X-FORWARDED-FOR");
    if (ipFromHeader != null && ipFromHeader.length() > 0) {
        log.debug("ip from proxy - X-FORWARDED-FOR : " + ipFromHeader);
        return ipFromHeader;
    }
    return request.getRemoteAddr();
}
```

5.2. Method *postHandle()*

This hook runs when the *HandlerAdapter* is invoked the handler but *DispatcherServlet* is yet to render the view.

We can use this method to add additional attributes to the *ModelAndView* or to determine the time taken by handler method to process a client's request.

In our case, we simply log a request just before *DispatcherServlet* is going to render a view.

```
@Override
public void postHandle(
  HttpServletRequest request,
  HttpServletResponse response,
  Object handler,
  ModelAndView modelAndView) throws Exception {

    log.info("[postHandle][" + request + "]");
}
```

5.3. Method *afterCompletion()*

When a request is finished and the view is rendered, we may obtain request and response data, as well as information about exceptions, if any occurred:

```
@Override
public void afterCompletion(
  HttpServletRequest request, HttpServletResponse response,Object handler, Exce
  throws Exception {
    if (ex != null){
        ex.printStackTrace();
    }
    log.info("[afterCompletion][" + request + "][exception: " + ex + "]");
}
```

6. Configuration

To add our interceptors into Spring configuration, we need to override *addInterceptors()* method inside *WebConfig* class that extends *WebMvcConfigurerAdapter:*

```
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new LoggerInterceptor());
}
```

We may achieve the same configuration by editing our XML Spring configuration file:

```
<mvc:interceptors>
    <bean id="loggerInterceptor" class="org.baeldung.web.interceptor.LoggerInte
</mvc:interceptors>
```

With this configuration active, the interceptor will be active and all requests in the application will be properly logged.

Please notice, if multiple Spring interceptors are configured, the *preHandle()* method is executed in the order of configuration,
whereas *postHandle()* and *afterCompletion()* methods are invoked in the reverse order.

7. Conclusion

This tutorial is an quick introduction to intercepting HTTP requests using Spring MVC Handler Interceptor.

All examples and configurations are available here on GitHub (https://github.com/eugenp/tutorials/tree/master/spring-security-mvc-custom).

CATEGORIES

SPRING (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)
HTTPCLIENT (HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA “BACK TO BASICS” TUTORIAL (HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (HTTP://WWW.BAELDUNG.COM/JACKSON)
HTTPCLIENT 4 TUTORIAL (HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/)
SPRING PERSISTENCE TUTORIAL (HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/)
SECURITY WITH SPRING (HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (HTTP://WWW.BAELDUNG.COM/ABOUT/)
THE COURSES (HTTP://COURSES.BAELDUNG.COM)
CONSULTING WORK (HTTP://WWW.BAELDUNG.COM/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE)
WRITE FOR BAELDUNG (HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES)
CONTACT (HTTP://WWW.BAELDUNG.COM/CONTACT)
COMPANY INFO (HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
TERMS OF SERVICE (HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
PRIVACY POLICY (HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY)