Spring MVC Interceptors Example - HandlerInterceptor and HandlerInterceptorAdapter

Sometimes we want to intercept the HTTP Request and do some processing before handing it over to the controller handler methods. That's where Spring MVC Interceptors come handy.

Just like we have Struts2 Interceptors, we can create our own interceptors in Spring by either implementing `org.springframework.web.servlet.HandlerInterceptor` interface or by overriding abstract class `org.springframework.web.servlet.handler.HandlerInterceptorAdapter` that provides the base implementation of this interface.

HandlerInterceptor declares three methods based on where we want to intercept the HTTP request.

1. **boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)**: This method is used to intercept the request before it's handed over to the handler method. This method should return 'true' to let Spring know to process the request through another interceptor or to send it to handler method if there are no further interceptors.
   If this method returns 'false' Spring framework assumes that request has been handled by the interceptor itself and no further processing is needed. We should use response object to send response to the client request in this case.

   Object *handler* is the chosen handler object to handle the request. This method can throw Exception also, in that case Spring MVC Exception Handling should be useful to send error page as response.

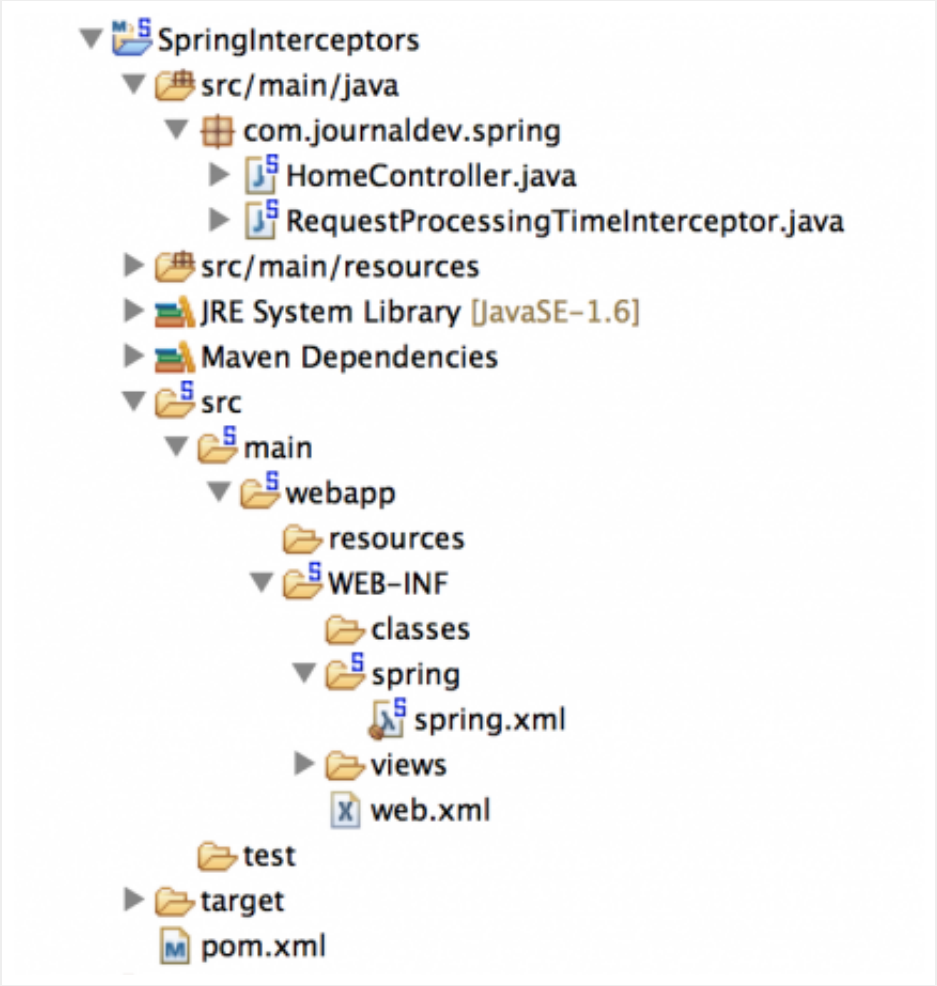
2. **void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView)**: This interceptor method is called when HandlerAdapter has invoked the handler but DispatcherServlet is yet to render the view. This method can be used to add additional attribute to the ModelAndView object to be used in the view pages. We can use this interceptor to determine the time taken by handler method to process the client request.
3. **void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex)**: This is a callback method that is called once the handler is executed and view is rendered.

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

Let's create a simple Spring MVC application where we will configure an interceptor to log timings

of controller handler method.

Our final project will look like below image, we will look into the components that we are interested in.



Controller Class

HomeController.java

```
1  package com.journaldev.spring;
2
3  import java.text.DateFormat;
4  import java.util.Date;
5  import java.util.Locale;
6
7  import org.slf4j.Logger;
8  import org.slf4j.LoggerFactory;
9  import org.springframework.stereotype.Controller;
10 import org.springframework.ui.Model;
11 import org.springframework.web.bind.annotation.RequestMapping;
12 import org.springframework.web.bind.annotation.RequestMethod;
13
14 /**
15  * Handles requests for the application home page.
16  */
17 @Controller
18 public class HomeController {
19
20     private static final Logger logger = LoggerFactory.getLogger(HomeControlle
21
```

```
    @RequestMapping(value = "/home", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        logger.info("Welcome home! The client locale is {}.", locale);
        //adding some time lag to check interceptor execution
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG

        String formattedDate = dateFormat.format(date);

        model.addAttribute("serverTime", formattedDate );
        logger.info("Before returning view page");
        return "home";
    }

}
```

I am just adding some processing time in the execution of the handler method to check our interceptor methods in action.

Spring MVC Interceptor Implementation

For simplicity, I am extending abstract class `HandlerInterceptorAdapter`.

RequestProcessingTimeInterceptor.java

```
package com.journaldev.spring;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

public class RequestProcessingTimeInterceptor extends HandlerInterceptorAdapte

    private static final Logger logger = LoggerFactory
            .getLogger(RequestProcessingTimeInterceptor.class);

    @Override
    public boolean preHandle(HttpServletRequest request,
            HttpServletResponse response, Object handler) throws Exception {
        long startTime = System.currentTimeMillis();
        logger.info("Request URL::" + request.getRequestURL().toString()
                + ":: Start Time=" + System.currentTimeMillis());
        request.setAttribute("startTime", startTime);
        //if returned false, we need to make sure 'response' is sent
        return true;
    }

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

```
        System.out.println("Request URL::" + request.getRequestURL().toString(
                + " Sent to Handler :: Current Time=" + System.currentTimeMil
        //we can add attributes in the modelAndView and use that in the view p
    }

    @Override
    public void afterCompletion(HttpServletRequest request,
            HttpServletResponse response, Object handler, Exception ex)
            throws Exception {
        long startTime = (Long) request.getAttribute("startTime");
        logger.info("Request URL::" + request.getRequestURL().toString()
                + ":: End Time=" + System.currentTimeMillis());
        logger.info("Request URL::" + request.getRequestURL().toString()
                + ":: Time Taken=" + (System.currentTimeMillis() - startTime)
    }

}
```

The logic is really simple, I am just logging the timings of handler method execution and total time taken in processing the request including rendering view page.

Spring MVC Interceptor Configuration

We have to wire the interceptor to the requests, we can use **mvc:interceptors** element to wire all the interceptors. We can also provide URI pattern to match before including the interceptor for the request through **mapping** element.

Our final spring bean configuration file looks like below.

spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:beans="http:/
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.s
        http://www.springframework.org/schema/beans http://www.springframeworl
        http://www.springframework.org/schema/context http://www.springframewo

    <!-- DispatcherServlet Context: defines this servlet's request-processing
        infrastructure -->

    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />

    <!-- Handles HTTP GET requests for /resources/** by efficiently serving
        up static resources in the ${webappRoot}/resources directory -->
    <resources mapping="/resources/**" location="/resources/" />

    <!-- Resolves views selected for rendering by @Controllers to .jsp resour
        in the /WEB-INF/views directory -->
    <beans:bean
        class="org.springframework.web.servlet.view.InternalResourceViewResol
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

    <!-- Configuring interceptors based on URI -->
```

```
    <interceptors>
        <interceptor>
            <mapping path="/home" />
            <beans:bean class="com.journaldev.spring.RequestProcessingTimeInte
        </interceptor>
    </interceptors>

    <context:component-scan base-package="com.journaldev.spring" />

</beans:beans>
```

I will not explain all other components of the web application, because we are not interested in them and they don't have any specific interceptor related configuration.

Test Spring MVC Interceptor Application

Just deploy the application in servlet container and invoke the home controller, you will see logger output something like below.

```
INFO : com.journaldev.spring.RequestProcessingTimeInterceptor - Request URL::h
INFO : com.journaldev.spring.HomeController - Welcome home! The client locale
INFO : com.journaldev.spring.HomeController - Before returning view page
Request URL::http://localhost:9090/SpringInterceptors/home Sent to Handler :: 
INFO : com.journaldev.spring.RequestProcessingTimeInterceptor - Request URL::h
INFO : com.journaldev.spring.RequestProcessingTimeInterceptor - Request URL::h
```

The output confirms that the interceptor methods are executed in the order defined.

That's all for using interceptors, you can download the sample project from below link and try to have multiple interceptors and check by different order of configuration.