Spring MVC Interceptor HandlerInterceptorAdapter, HandlerInterceptor Example

APRIL 2, 2018 BY PANKAJ — 16 COMMENTS

Spring Interceptor are used to intercept client requests and process them. 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 Interceptor come handy.

Table of Contents [hide]

Spring Interceptor

Just like we have Struts2 Interceptors, we can create our own Spring interceptor 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 HandlerInterceptor interface.

Spring Interceptor – HandlerInterceptor

Spring **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 spring interceptor or to send it to handler method if there are no further spring interceptors.
   If this method returns 'false' Spring framework assumes that request has been handled by the spring 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 HandlerInterceptor 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 spring interceptor method 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 HandlerInterceptor callback method that is called once the handler is executed and view is rendered.

If there are multiple spring 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 Spring Interceptor to log timings of controller handler method.

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

Spring Interceptor – Controller Class

```
package com.journaldev.spring;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

/**
 * Handles requests for the application home page.
 */
@Controller
public class HomeController {

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

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

Spring MVC Interceptor – HandlerInterceptorAdapter Implementation

For simplicity, I am extending abstract class `HandlerInterceptorAdapter`. HandlerInterceptorAdapter is abstract adapter class for the HandlerInterceptor interface, for simplified implementation of pre-only/post-only interceptors.

```
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 HandlerInterceptorAdapter {

        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());
```

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 spring 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 spring interceptor for the request through **mapping** element.

Our final spring bean configuration file (spring.xml) looks like below.

```
<?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://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
                http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

        <!-- 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/" />
```

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 spring interceptor related configuration.

Spring MVC Interceptor Application Testing

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::http://localhost:9090/SpringInterceptors/home:: Start Time=1396906442086
INFO : com.journaldev.spring.HomeController - Welcome home! The client locale is
en_US.
INFO : com.journaldev.spring.HomeController - Before returning view page
Request URL::http://localhost:9090/SpringInterceptors/home Sent to Handler :: Current
Time=1396906443098
INFO : com.journaldev.spring.RequestProcessingTimeInterceptor - Request
URL::http://localhost:9090/SpringInterceptors/home:: End Time=1396906443171
INFO : com.journaldev.spring.RequestProcessingTimeInterceptor - Request
URL::http://localhost:9090/SpringInterceptors/home:: Time Taken=1085
```

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

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

Download Spring Interceptors Project

« PREVIOUS
Spring Validation Example – Spring MVC Form Validator

NEXT »
Spring Interview Questions and Answers

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: SPRING

Comments

Robert says
JUNE 16, 2018 AT 8:42 AM
Hello Pankaj,
Thank you for the clear example on spring interceptors. Really appreciate it. My question is on Spring itself really and its enthusiasm for adding a plethora of additional dependencies that potentially clogs up the project over time.
I checked the pom file in your project example and there is a lot of "stuff" in there. This is not critique to you, just a question.
Do you know/have available a bare minimal amount of dependencies that are needed or is this listed dependencies in the project optimal ?
BR, Robert
Reply

Pankaj says
JUNE 16, 2018 AT 10:46 AM
Hi Robert,
There has been a lot of changes in Spring, if you are using latest versions then you can remove many of the logging related dependencies. You can always run `mvn dependency:tree` to find transitive dependencies or unused dependencies and remove them from `pom.xml` file. This will keep your pom.xml file small and clean.
Reply

Rajat Mishra says
MARCH 11, 2018 AT 1:57 PM

You are awesome. Never stop writing such awesome blogs.
Reply

akash says
MARCH 22, 2017 AT 4:36 AM
how to apply logging interceptor for all methods what should i do.
like ,for create we use following
"
"
if i want do this fr all what should i do
Reply

Akhil says
SEPTEMBER 28, 2016 AT 1:09 AM
How to use @Autowired service in handler….If i am using then null pointer exception will occur
Reply

Tejas says
MARCH 17, 2017 AT 11:32 AM
It is nothing but creating object .As suppose I have written service in Class *ServiceImpl.java . I will write autowired as
class {
@Autowired
*ServiceImpl ,
and use reference everywhere
}
Reply

lastspring says
AUGUST 28, 2016 AT 10:54 PM
How to modify the respone ?
Reply

Rita says

OCTOBER 19, 2015 AT 9:54 PM

Problem : I am calling spring controller using curl request.

As Input I am passing a json with request to controller.

Now I just want to fetch that json inside spring interceptor .But it is not working for me.

Enumeration en=request.getParameterNames();

System.out.println(Collections.list(en).size());

It is giving zero size.

Please take a look.

Reply

Pankaj says

JUNE 6, 2016 AT 8:07 AM

Could you please post your curl command? What is the HTTP method you are using, it should be POST in your case, not GET as in this example.

Reply

Pupsick says

APRIL 16, 2015 AT 9:17 AM

The presence of so many logging dependencies and exclusions is amazing. Even such a tiny app. required … 7!!! Artifacts. Surely that was not what logging was meant to be. But this is what is wrong with open source in general.

Reply

saljuama says

AUGUST 25, 2015 AT 8:43 AM

Really? this is what you get out of an Interceptor tutorial? some rant about logging dependencies? that in fact are generated automatically when creating a spring project using eclipse (STS) the old fashion way, which is tailored to give easy time for developers. This smells like open source hater to me.

Reply

Pankaj says

JUNE 6, 2016 AT 8:08 AM

You can actually clean them up and just use what you need, I am bit lazy to do that for my example code.

Reply

Evgeny says

MARCH 12, 2015 AT 1:44 AM

Thanks !

Simple and intuitive !

Reply

Pankaj says

JUNE 6, 2016 AT 8:08 AM

You are welcome friend.

Reply

Ram says

NOVEMBER 6, 2014 AT 1:57 AM

Hi,

when I am getting the parameter value from request, its a null value. so how should I get the parameter value in interceptor.

Reply

Pankaj says

JUNE 6, 2016 AT 8:09 AM

Please provide some code snippet, what is your HTTP method? How you are passing request parameters?

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

SPRING FRAMEWORK

Spring Tutorial

Spring Core

- Spring Framework
- Spring Dependency Injection
- Spring IoC and Bean
- Spring Bean Life Cycle
- Spring REST
- Spring REST XML
- Spring RestTemplate

- Spring ActiveMQ Example

Spring Integrations

- Spring JDBC
- Spring DataSource JNDI
- Spring Hibernate
- Spring Primefaces JPA
- Spring Primefaces MongoDB
- Spring Primefaces Hibernate
- SpringRoo Primefaces Hibernate
- Spring JSF
- Spring JDF Hibernate

Miscellaneous

- Spring Data MongoDB
- Spring Interview Questions

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial

Alibaba Cloud | intel

Want a 60% performance boost?

Try Alibaba Cloud next-gen cloud server

Join Us Now