(/)

A Quick Struts 2 Intro

Last modified: August 22, 2019

by baeldung (https://www.baeldung.com/blog/author/baeldung/)

Java EE (https://www.baeldung.com/blog/category/java-ee/)

I just announced the new *Learn Spring* course, focused on the fundamentals of Spring 5 and Spring Boot 2:

>> CHECK OUT THE COURSE (/ls-course-start)

1. Introduction

Apache Struts 2 (https://struts.apache.org) is an MVC-based framework for developing enterprise Java web applications. It is a complete rewrite of original Struts framework. It has an open source API implementation and a rich feature set.

We use cookies to improve your experience with the site. To find out more, you can read the full Privacy and Cookie Policy (/privacy-policy)
Ok

In this tutorial, we will have a beginner's introduction to different core components of the Struts2 framework. Moreover, we will show how to use them.

2. Overview of Struts 2 Framework

Some of Struts 2 features are:

- POJO (plain old Java Objects)-based actions
- plugin support for REST, AJAX, Hibernate, Spring, etc
- convention over configuration
- support of various view-layer technologies
- ease of profiling and debugging

2.1. Different Components of Struts2

Struts2 is an MVC-based framework so the following three components will be present in all Struts2 applications:

1. **Action class –** which is a POJO class (POJO means it is not part of any type hierarchy and can be used as a standalone class); we will implement our business logic here
2. **Controller –** in Struts2, HTTP filters are used as controllers; they basically perform tasks like intercepting and validating requests/responses
3. **View –** is used for presenting processed data; it is usually a JSP file

3. Designing Our Application

Let's proceed with the development of our web app. It is an application where a user selects a particular *Car* brand and gets greeted by a customized message.

3.1. Maven Dependencies

Let's add following entries to the *pom.xml*:

```
<dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-core</artifactId>
    <version>2.5.10</version>
</dependency>
<dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-junit-plugin</artifactId>
    <version>2.5.10</version>
</dependency>
<dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-convention-plugin</artifactId>
    <version>2.5.10</version>
</dependency>
```

The latest version of the dependencies can be found here (https://search.maven.org/classic/#search%7Cga%7C1%7Cg%3A%22org.apache.struts%22).

3.2. Business Logic

We use cookies to improve your experience with the site. To find out more, you can read the full Privacy and Cookie Policy (/privacy-policy)

Ok

Let's create an action class *CarAction* which returns a message for a particular input value. The *CarAction* has two fields – *carName* (used for storing the input from the user) and *carMessage* (used for storing the custom message to be displayed):

```
public class CarAction {

    private String carName;
    private String carMessage;
    private CarMessageService carMessageService = new CarMessageService();

    public String execute() {
        this.setCarMessage(this.carMessageService.getMessage(carName));
        return "success";
    }

    // getters and setters
}
```

The *CarAction* class uses *CarMessageService* which provides the customized message for a *Car* brand:

```
public class CarMessageService {

    public String getMessage(String carName) {
        if (carName.equalsIgnoreCase("ferrari")){
            return "Ferrari Fan!";
        }
        else if (carName.equalsIgnoreCase("bmw")){
            return "BMW Fan!";
        }
        else {
            return "please choose ferrari Or bmw";
        }
    }
}
```

3.3. Accepting User Input

Let's add a *JSP* which is an entry point in our application. This is a content of the *input.jsp* file:

```
<body>
    <form method="get" action="/struts2/tutorial/car.action">
        <p>Welcome to Baeldung Struts 2 app</p>
        <p>Which car do you like !!</p>
        <p>Please choose ferrari or bmw</p>
        <select name="carName">
            <option value="Ferrari" label="ferrari" />
            <option value="BMW" label="bmw" />
         </select>
        <input type="submit" value="Enter!" />
    </form>
</body>
```

The *<form>* tag specifies the action (in our case it is a HTTP URI to which GET request has to be sent).

We use cookies to improve your experience with the site. To find out more, you can read the full Privacy and Cookie Policy (/privacy-policy)

Ok

3.4. The Controller Part

StrutsPrepareAndExecuteFilter is the controller, which will intercept all incoming requests. We need to register the following filter in the *web.xml:*

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

StrutsPrepareAndExecuteFilter will filter every incoming request as we are specifying URL matching wildcard ***<url-pattern>/*</url-pattern>***

3.5. Configuring the Application

Let's add following annotations to our action class *Car*:

```
@Namespace("/tutorial")
@Action("/car")
@Result(name = "success", location = "/result.jsp")
```

Let's understand the logic of this annotations. The **@*Namespace*** is used for logical separation of request URI for different action classes; we need to include this value in our request.

Furthermore, the ***@Action*** tells the actual end point of the request URI which will hit our *Action* class. The action class consults *CarMessageService* and initializes the value of another member variable *carMessage*. After *execute()* method returns a value, **"success"** in our case, it matches that value to invoke *result.jsp*

Finally, the ***@Result*** has two parameters. First one, *name,* specifies the value which our *Action* class will return; this value is returned from the *execute()* method of *Action* class. **This is the default method name which will be executed**.

The second part, *location,* tells which is the file to be referred to after the *execute()* method has returned a value. Here, we are specifying that when *execute()* returns a String with value "*success*", we have to forward the request to *result.jsp*.

The same configuration can be achieved by providing XML configuration file:

We use cookies to improve your experience with the site. To find out more, you can read the full Privacy and Cookie Policy (/privacy-policy)

Ok

```
<struts>
    <package name="tutorial" extends="struts-default" namespace="/tutorial">
        <action name="car" class="com.baeldung.struts.CarAction" method="execute">
            <result name="success">/result.jsp</result>
        </action>
    </package>
</struts>
```

3.6. The View

This is the content of *result.jsp* which will be used for presenting the message to the user:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<body>
    <p> Hello Baeldung User </p>
    <p>You are a <s:property value="carMessage"/></p>
</body>
```

There are two important things to notice here:

- in *<@taglib prefix="s" uri="/struts-tags %>* we are importing *struts-tags* library
- in *<s:property value="carMessage"/>* we are using *struts-tags* library to print the value of a property *carMessage*

4. Running the Application

This web app can be run in any web container, for example in Apache Tomcat. These are the required steps for accomplishing it:

1. After deploying the web app, open the browser and access the following URL: *http://www.localhost.com:8080/MyStrutsApp/input.jsp*
2. Select one of the two options and submit the request
3. You will be forwarded to the *result.jsp* page with customized message based on the selected input option

5. Conclusion

We use cookies to improve your experience with the site. To find out more, you can read the full Privacy and Cookie Policy (/privacy-policy)

Ok

In this tutorial, we walked through a step by step guide, how to create our first Struts2 web application. We covered different MVC related aspects in the Struts2 domain and showed how to put them together for development.

As always, this tutorial can be found over on Github (https://github.com/eugenp/tutorials/tree/master/struts-2) as a Maven project.

I just announced the new *Learn Spring* course, focused on the fundamentals of Spring 5 and Spring Boot 2:

>> CHECK OUT THE COURSE (/ls-course-end)

Comments are closed on this article!

CATEGORIES

SPRING (HTTPS://WWW.BAELDUNG.COM/BLOG/CATEGORY/SPRING/)
REST (HTTPS://WWW.BAELDUNG.COM/BLOG/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/BLOG/CATEGORY/JAVA/)
SECURITY (HTTPS://WWW.BAELDUNG.COM/BLOG/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTPS://WWW.BAELDUNG.COM/BLOG/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/BLOG/CATEGORY/JSON/JACKSON/)
HTTP CLIENT-SIDE (HTTPS://WWW.BAELDUNG.COM/BLOG/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/BLOG/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
CONSULTING WORK (/CONSULTING)

We use cookies to improve your experience with the site. To find out more, you can read the full Privacy and Cookie Policy (/privacy-policy)

Ok

META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)

We use cookies to improve your experience with the site. To find out more, you can read the full Privacy and Cookie Policy (/privacy-policy)

Ok