Starting Spring MVC Tutorial

Part 1 of 8 Tutorial Set

By Ken Williamson

Driven Solutions

www.drivensolutions.com

Introduction

1. Who should use this tutorial?

   This tutorial is intended for entry level software engineers and developers interested in learning Spring MVC. No prior knowledge of Spring MVC is required, but this tutorial does assume a prior knowledge of Java and J2EE. It is also assumed that the reader has prior knowledge of web application software development.

   This tutorial starts from the very beginning of building a Spring MVC application and gives step by step instructions on how to build and deploy a Spring MVC application. All code and configuration files are explained to give the user an understanding of how the application functions.

2. What is Spring MVC and why should it be used?

   Spring MVC is an Open Source Java based web application framework designed for use with agile development methods and especially the XP (Extreme Programming) method. Spring MVC provides a way to quickly develop web applications that follow the Model View Controller design pattern that provides for the isolation of view code, controller code, and data access code.

   Developers and engineers might question why another MVC framework is needed. Spring MVC, however, offers the following advantages over its two closest competitors, Struts and Java Server Faces:

   - ➔ Spring MVC is very lightweight which translates to much less developed code in the web layer. This also translates to shorter development cycles.
   - ➔ Spring MVC is designed to make full use of test driven development methodologies by using POJO (Plain Old Java Object) that work well with test platforms such as JUNIT.
   - ➔ Spring MVC is very flexible which translates to the ability to easily interface other software modules into Spring MVC.
   - ➔ Spring MVC includes a collection of tag libraries for use inside web pages to help reduce the actual amount of required code inside each page. Developers can also easily incorporate tag libraries from other platforms such as Struts if needed.
   - ➔ Spring MVC is much easier to learn and use than Struts or Java Server Faces and allows developers to quickly produce web application.

3. Why use this tutorial.

   This tutorial is designed to help the reader easily understand the development processes required my Spring MVC. The reader can follow a step by step procedure to develop their first application. The reader can also refer back to this tutorial for reference when need.

4. What this tutorial contains:

   This tutorial contains the following sections:

   - What you need
   - Building your project
   - Writing the required code
   - Configuring the application
   - Deploying the application
   - Testing the application

5. How to use this tutorial.

   The best way to use this tutorial is to start at the beginning and follow each described procedure as recommended by the tutorial. Once each process is completed, the reader is left with a functional template application that serves as a foundation for future development projects.

Building Your First Spring MVC Application

What you will need

1. Java 1.5 or greater SDK
   - Download the latest full Java SE SDK from www.java.sun.com
   - Follow the installation instructions provided with the download.
2. MyEclipseIDE
   - Download MyEclipseIDE (there is a trial version available with a 30 day license) at www.myeclipseide.com
   - Follow the installation instructions provided with the download.
3. Tomcat 5.5 or later
   - Download the latest Tomcat at http://tomcat.apache.org
   - Placed the unzipped apache-tomcat-xxxx folder in the directory of you choice.
   - Follow the MyEclipseIDE instructions on how to install application servers and install Tomcat.

Building a web Project

Once you have the required components, you can start building your first application.

1. On the top menu of MyEclipse, go to: (***see figure 1.1***)
   - File->new and click ***Project***
   - Click ***Web Project***
2. Press next
3. Name the project ***helloWebWorld***
4. Select J2EE 1.4
5. Select the box labeled ***Add JSTL 1.0 Libraries***
6. Press finish

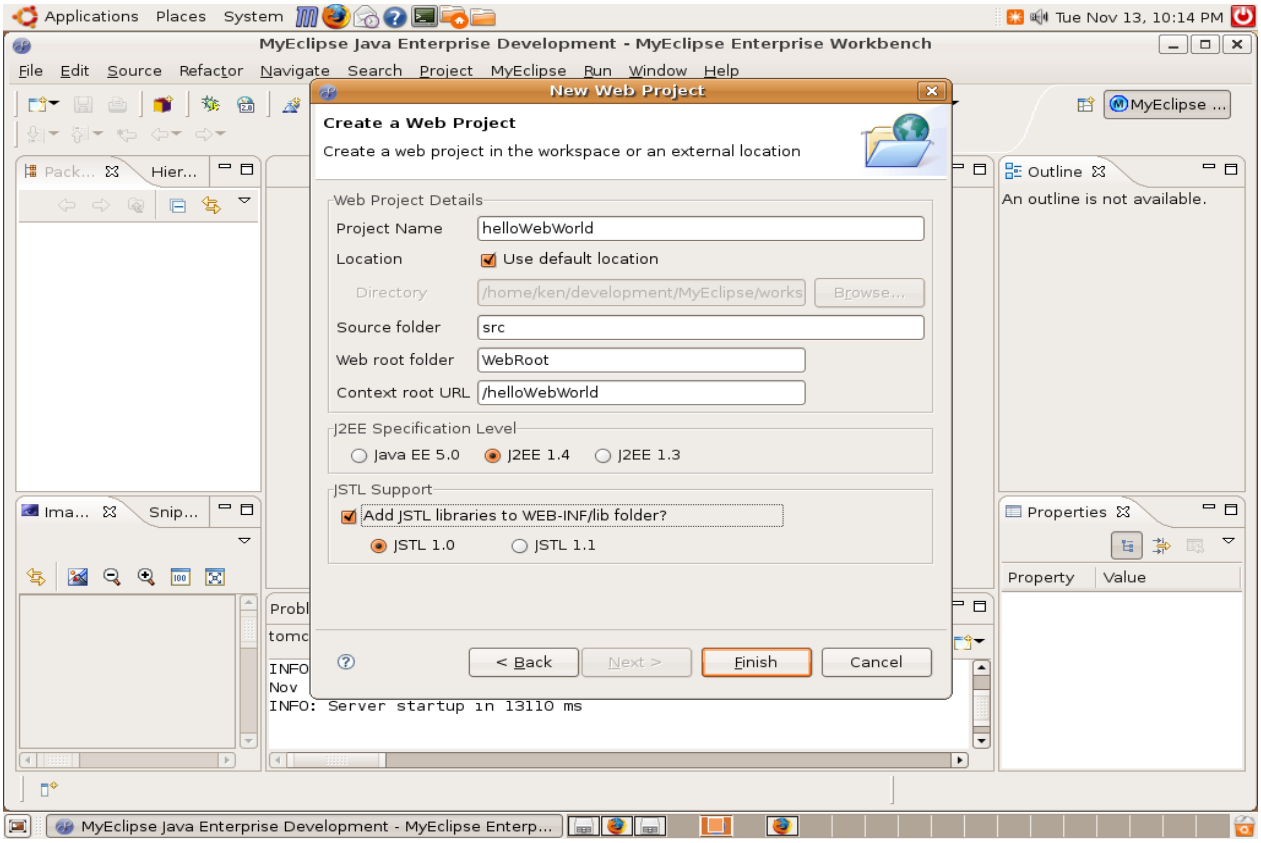

Figure 1.1

7. Click on the project helloWebWorld
8. On the top menu:
   - Click MyEclipse from the menu
   - Click Add Spring Capabilities from the context menu
9. Select the following boxes: (***see figure 1.2***)

   - Spring 2.0 J2EE Libraries
   - Spring Testing Support Libraries
   - Spring 2.0 Web Libraries
10. Leave everything else at default
11. Press finish
12. This completes the initial project setup.
13. Continue with the next section “Writing the required code”.

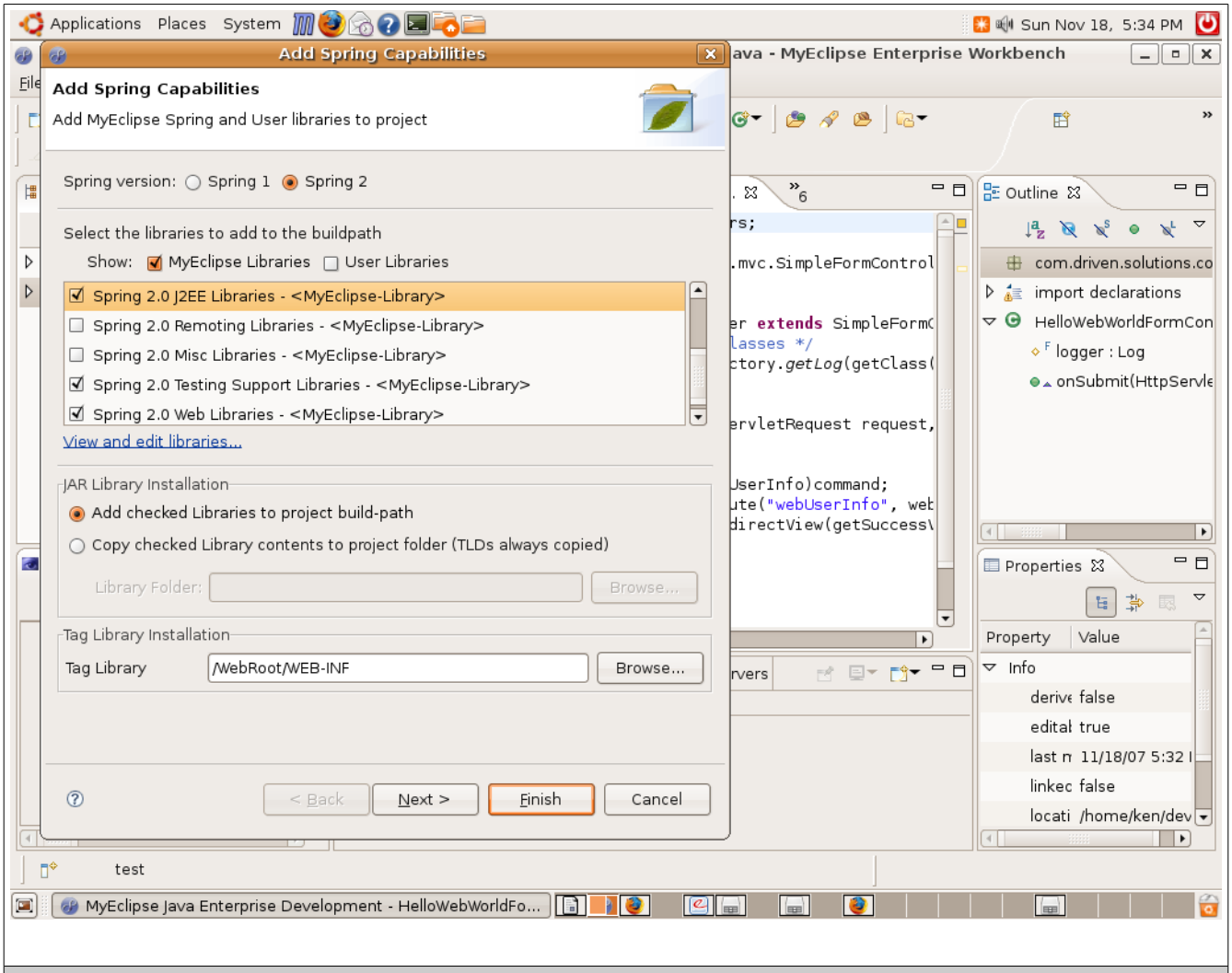

Figure 1.2

Writing the required code

In this section, you build the application structure and edit the files needed to complete the application. Build the application structure:

1. Click WebRoot/WEB-INF:
   - Double click ***web.xml*** *(This file configures the web application)*
   - Add the code listed in red below:(***see figure 1.3***)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
      xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<servlet>
    <servlet-name>helloWebWorld</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>helloWebWorld</servlet-name>
    <url-pattern>*.htm</url-pattern>  <!-- allows jsp pages to be accessed with .htm
extension -->
</servlet-mapping>
  <welcome-file-list>
    <welcome-file>/index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Figure 1.3 (web.xml)

The helloWebWorld servlet configuration is used to load the Spring context at startup of the web application. Notice the servlet url mapping. Mapping all pages to a htm extension allows you to hide the application implementation type and therefore all pages appear to be standard htm pages.

2. Click WebRoot/WEB-INF:
   - On the top menu select
   - File->new->folder
   - name the new folder ***jsp***

We place all jsp pages inside the WEB-INF folder where they are unreachable by a web browser. By doing this, we prevent users from navigating directly to a jsp page itself.

3. Click WebRoot/WEB-INF/jsp
   - On the top menu select
   - File->new->JSP
   - Name the JSP helloWebUser.jsp *(This is one of two web pages in the application)*
   - Add the code listed in ***figure 1.4***

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()
+":"+request.getServerPort()+path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>My JSP 'helloWebWorld.jsp' starting page</title>
      <meta http-equiv="pragma" content="no-cache">
      <meta http-equiv="cache-control" content="no-cache">
      <meta http-equiv="expires" content="0">
      <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
      <meta http-equiv="description" content="This is my page">
      <!--
      <link rel="stylesheet" type="text/css" href="styles.css">
      -->
  </head>
  <body>
  Hello Web User<br>
<br>
<form method='post'>First Name:
  <input name="firstName"><br>
  <br>
  <br>
Last Name: <input name="lastName"> <br>
  <br>
Age:    
       <input
 name="age"><br>
  <br>
Location:    <input name="location"><br>
  <br>
  <br>
  <input name="submit" value="Submit" type="submit"><br>
  <br><br><br>
</form>
  </body>
</html>
```

Figure 1.4 *(helloWebUser.jsp)*

Notice that the ***form tag*** has no action attribute. The action is actually mapped in the the Spring configuration file. We will see this file later.

4. Follow step 3 and create another JSP
   - Name the JSP greetings.jsp *(This is the second web page in the project)*
   - Add the code listed in ***figure 1.5***

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()
+":"+request.getServerPort()+path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>My JSP 'greetings.jsp' starting page</title>
      <meta http-equiv="pragma" content="no-cache">
      <meta http-equiv="cache-control" content="no-cache">
      <meta http-equiv="expires" content="0">
      <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
      <meta http-equiv="description" content="This is my page">
      <!--
      <link rel="stylesheet" type="text/css" href="styles.css">
      -->
  </head>
  <body>
  <table style="text-align: left; width: 727px; height: 349px;"
 border="0" cellpadding="2" cellspacing="2">
  <tbody>
    <tr>
      <td> Hello <b>${webUserInfo.firstName}</b> <b>${webUserInfo.lastName}</b>.
How are you today.<br><br>
      You are now <b>${webUserInfo.age}</b> and you live in <b>$
{webUserInfo.location}</b>.
      <br></td>
    </tr>
  </tbody>
</table>
  </body>
</html>
```

Figure 1.5 (greetings.jsp

Notice the EL tags used in the greetings.jsp page. WebUserInfo is a command bean used to hold all page variables. The command bean is configured later and is populated by the Spring MVC framework.

5. Click the ***src*** folder and:
   - On the top menu click:
   - File->new->Package
   - Name the Package ***com.driven.solutions.commandClasses***
   - Click the package added above and click:
   - File->new->class and name the class ***WebUserInfo.java*** and add the code listed in ***figure 1.6*** *(The object created from this class holds the data from helloWebUsers.jsp)*
   - Also add another package: ***com.driven.solutions.controllers***
   - Add two classes to this package, ***HelloWebWorldFormController.java*** and ***GreetingController.java*** and add the code listed in ***figures 1.7 and 1.8*** *(These classes are the Spring MVC controller classes and the objects created from these classes control the application flow)*

```
package com.driven.solutions.commandClasses;
public class WebUserInfo {
	private String firstName;//java bean used to hold page values
	private String lastName;
	private Integer age;
	private String location;
	public Integer getAge() {
		return age;
	}
	public void setAge(Integer age) {
		this.age = age;
	}
	public String getFirstName() {
		return firstName;
	}
	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}
	public String getLastName() {
		return lastName;
	}
	public void setLastName(String lastName) {
		this.lastName = lastName;
	}
	public String getLocation() {
		return location;
	}
	public void setLocation(String location) {
		this.location = location;
	}
}
```

Figure 1.6 (WebUserInfo.java)

```
package com.driven.solutions.controllers;
import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.view.RedirectView;
import com.driven.solutions.commandClasses.WebUserInfo;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Map;
import java.util.HashMap;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class GreetingsController implements Controller{// only need to implement Controller because no //form is used on the greetings.jsp page
	public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {
		WebUserInfo webUserInfo = (WebUserInfo)request.getSession().getAttribute("webUserInfo");// get web page values
		return new ModelAndView("greetings","webUserInfo",webUserInfo);//forward web page //values to new web page
	}
}
```

Figure 1.7(GreetingsController.java)

```
package com.driven.solutions.controllers;
import org.springframework.web.servlet.mvc.SimpleFormController;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.view.RedirectView;
import com.driven.solutions.commandClasses.WebUserInfo;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Map;
import java.util.HashMap;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class HelloWebWorldFormController extends SimpleFormController{//must extend the //SimpleFormController because a form is on the
        // helloWebUser.jsp page that is mapped to this controller
   protected final Log logger = LogFactory.getLog(getClass());
   public ModelAndView onSubmit(HttpServletRequest request, HttpServletResponse response, Object command,
BindException errors)
   throws ServletException {
        WebUserInfo webUserInfo = (WebUserInfo)command; // get web page values
        request.getSession().setAttribute("webUserInfo", webUserInfo); // put web page values in request //object to be forwarded to new page
        return new ModelAndView(new RedirectView(getSuccessView())); //send to the web page //configured as the successView page.
   }
}
```

Figure 1.8 (HelloWebWorldFormController)

Notice that the HelloWebWorldFormController extends the SimpleFormControaller. Doing this allows us to override the onSubmit method. Doing this also provides other functionality. For more detail, research the SimpleFormController at www.springframework.org.

Configuring the application

6. Click WebRoot/WEB-INF
   - Add the following file ***helloWebWorld-servlet.xml*** *(This is the Spring MVC configuration file where all the components of the application are tied together)* :
   - Enter the code listed in ***figure 1.9***

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<!--
  - Application context definition for "springapp" DispatcherServlet.
  -->
<beans>   <!-- configures the controller -->
    <bean id="helloWebWorldForm"
class="com.driven.solutions.controllers.HelloWebWorldFormController">
        <property name="sessionForm"><value>true</value></property>
        <property name="commandName"><value>webUserInfo</value></property>
        <property
name="commandClass"><value>com.driven.solutions.commandClasses.WebUserInfo</value>
</property>
        <property name="formView"><value>helloWebUser</value></property>
        <property name="successView"><value>greetings.htm</value></property>
    </bean>     <!-- configures the controller -->

 <bean id="greetingsController"
class="com.driven.solutions.controllers.GreetingsController"/>
    <bean id="urlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>  <!-- maps web pages to the controllers -->
                <prop key="/helloWebUser.htm">helloWebWorldForm</prop>
                 <prop key="/greetings.htm">greetingsController</prop>

            </props>
        </property>
    </bean>
     <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass">
           <value>org.springframework.web.servlet.view.JstlView</value>
        </property> <!-- configures prefix and suffix for web pages inside the
added jsp folder -->
        <property name="prefix"><value>/WEB-INF/jsp/</value></property>
        <property name="suffix"><value>.jsp</value></property>
    </bean>
</beans>
```

Figure 1.9 *(helloWebWorld-servlet.xml)*

The helloWebWorld-servlet.xml is the Spring MVC configuration file. As seen, we configure each controller and the command beans used. We also configure the input page (formView) and the output page (successView) of the form controller. Notice also that we configure the urlMapping by mapping each htm page to the controller that handles the page.

Deploying the application

7. Press the Deploy button on MyEclipse to deploy the application.

Once the application is deployed, we can start testing the operation.

Testing the application

8. Press the Start button on MyEclipse to start Tomcat.
9. Start a web browser and enter:
    - http://localhost:8080/helloWebWorld/***helloWebUser.htm***
10. Enter your first name, last name, age and location.
11. Press Submit
12. You should see the greetings page showing your first and last name, age and location.
13. Congratulation; you just completed your first Spring MVC application.

Part 2 of this tutorial will cover interfacing the MVC layer to the business logic layer.
Part 3 of this tutorial will cover using JPA and Hibernate
Part 4 of this tutorial will cover using JPA and Toplink.
Part 5 of this tutorial will cover using AOP transaction concepts
Part 6 of this tutorial will cover tag libraries for Spring MVC
Part 7 of this tutorial will cover Spring web application design concepts
Part 8 of this tutorial will tie everything together with Spring Webflow design concepts

References

Introduction To Spring Framework. (2005). Retrieved October 25, 2007 from http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework.

What's new in Spring 2.0, The Web Tier. (2007). Retrieved October 25, 2007 from http://static.springframework.org/spring/docs/2.0.x/reference/new-in-2.html#new-in-2-web

Web MVC Framework. (2007). Retrieved October 25, 2007 from http://static.springframework.org/spring/docs/2.0.x/reference/mvc.html#mvc-formtaglib