Spring MVC Tutorial – Contact List Application

Step 05: CRUD Functionality without Ajax

Step 05: CRUD Functionality without Ajax

Overview

This section of the tutorial presents another approach to building Java web applications. This approach does not use Ajax or JavaScript — it uses only Java code on the server, JSPs, and tag libraries. We'll add a new controller and new views (i.e. JSPs) to handle all of the CRUD functionality. These components will sit alongside and run in parallel with components that we have already created for our application. We do not have to re-implement either the DTO or DAO because we can reuse the DTO and DAO that we have already created. We will only create new controller and view components.

Step 01 - Add New Controller

Our first step in this process will be to create a new controller. The controller will contain all of the endpoints needed for this new approach. Like the other controllers in this application, we will inject the DAO into this controller using annotation-based constructor injection. This initial version of this controller will contain only one endpoint, which will simply return the logical name of the view we'll use for the main page for this new approach.

Create a new Java class called **HomeControllerNoAjax** in the **controller** package. Put the following code in your new controller, and pay attention to the comments for a detailed explanation of what the code is doing:

```
@Controller
public class HomeControllerNoAjax {

    // Reference to our DAO
    private ContactListDao dao;

    // Use annotation-driven constructor injection to inject a DAO implementation
    // into our controller
    @Inject
    public HomeControllerNoAjax(ContactListDao dao) {
        this.dao = dao;
    }

    // This endpoint simply returns the name of the view that will serve as
    // the main landing page for the new functionality.  The name of that view
    // is displayContactListNoAjax.jsp.
    //
    // NOTE: The RequestMapping value, the name of the method, and the name
    //       of the JSP are all the same in this case (displayContactListNoAjax).
    //       THIS IS NOT A REQUIREMENT!!!  These names can be all different if
    //       you want them to be.
    //
    // NOTE: This method does takes a Model object as a parameter.  This is
    //       because this method gets a list of all the Contact objects from the
    //       DAO.  We need to place this list on the Model so that Spring MVC
    //       can pass the list of Contacts on to the view component.  We'll
    //       use JSTL tags to iterate through the list and print the Contact
    //       information to the screen.
    @RequestMapping(value="/displayContactListNoAjax", method=RequestMethod.GET)
    public String displayContactListNoAjax(Model model) {
        return "displayContactListNoAjax";
    }
}
```

Step 02 - Add New View and Update Nav Bar

Now, we need to create the View component for the endpoint that we created in the previous step. We also have to update the nav bar in our existing view so that we can get to our new page.

Create a new JSP called **displayContactListNoAjax.jsp** in the **jsp** folder. Remove the auto-generated content of this file and replace it with the following. Pay attention to the link we have added to the nav bar (Contact List (No Ajax)):

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="s" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Company Contacts</title>
        <!-- Bootstrap core CSS -->
        <link href="${pageContext.request.contextPath}/css/bootstrap.min.css" rel="stylesheet">

        <!-- SWC Icon -->
        <link rel="shortcut icon" href="${pageContext.request.contextPath}/img/icon.png">

    </head>
    <body>
        <div class="container">
            <h1>Company Contacts</h1>
            <hr/>
            <div class="navbar">
                <ul class="nav nav-tabs">
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/home">Home</a>
                    </li>
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/search">Search</a>
                    </li>
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/stats">Stats</a>
                    </li>
                    <li role="presentation" class="active">
                        <a href="${pageContext.request.contextPath}/displayContactListNoAjax">
                            Contact List (No Ajax)
                       </a>
                    </li>
                </ul>
            </div>
        </div>


        <!-- Placed at the end of the document so the pages load faster -->
        <script src="${pageContext.request.contextPath}/js/jquery-1.11.1.min.js"></script>
        <script src="${pageContext.request.contextPath}/js/bootstrap.min.js"></script>

    </body>
</html>
```

Now add the new link to this page to the nav bar in home.jsp, search.jsp, and stats.jsp so that the nav bar looks the same for all of our pages. Copy and paste the following just below the Stats link in the nav bar for each of these JSPs:

```
<li role="presentation" class="active">
    <a href="${pageContext.request.contextPath}/displayContactListNoAjax">
        Contact List (No Ajax)
   </a>
</li>
```

Step 03 - Implement displayContactListNoAjax Controller Method

Now that we have the shells of the Controller and View in place, we'll finish implementing the displayContactListNoAjax method. Change your displayContactListNoAjax method so that it looks like the following:

```
@RequestMapping(value="/displayContactListNoAjax", method=RequestMethod.GET)
public String displayContactListNoAjax(Model model) {
    // Get the list of all Contacts
    List<Contact> cList = dao.getAllContacts();
    // Put the list of all Contacts on the Model so Spring MVC can pass it
    // along to the view
    model.addAttribute("contactList", cList);
    // Return the logical view name
    return "displayContactListNoAjax";
}
```

Step 04 - Implement displayContactListNoAjax.jsp

Now, we can implement the displayContactListNoAjax.jsp view component. This page will display a list of all the Contacts in the system. Each Contact displayed on the page will have a link that allows the user to either Edit or Delete the Contact. The page will also have a link that allows the user to create a new Contact.

Add the following <div> to your displayContactListNoAjax.jsp. This <div> should be placed just before the <script> tags that reference the jQuery and Bootstrap JavaScript files (pay attention to the numbered comments in the code):

```
<div class="container">
    <h1>Company Contacts</h1>
    <!-- #1 - Link to addContactForm -->
    <a href="displayNewContactFormNoAjax">Add a Contact</a><br/>
    <hr/>

    <!-- #2 - Iterate over contactList: forEach contact in contactList, do something -->
    <c:forEach var="contact" items="${contactList}">
        <!-- #3 - Build custom delete URL for each contact.  Use the id of the contact -->
        <!--      to specify the contact to delete or update                           -->
        <s:url value="deleteContactNoAjax"
               var="deleteContact_url">
            <s:param name="contactId" value="${contact.contactId}" />
        </s:url>
        <!-- Build custom edit URL for each contact -->
        <s:url value="displayEditContactFormNoAjax"
               var="editContact_url">
            <s:param name="contactId" value="${contact.contactId}" />
        </s:url>
        <!-- #4 - A pointless demonstration of the if tag -->
        <c:if test="${contact.lastName == 'Doe'}">
            CEO<br/>
        </c:if>
        Name: ${contact.firstName} ${contact.lastName} |
        <a href="${deleteContact_url}">Delete</a> |
        <a href="${editContact_url}">Edit</a><br/>
        Phone: ${contact.phone}<br/>
        Email: ${contact.email}<br/>
        <hr>
    </c:forEach>
</div>
```

Step 05 - Implement Controller Endpoint for Displaying the New Contact Form

In this step, we will implement the controller endpoint that causes the New Contact form to be displayed. When building a web application that doesn't use Ajax, each of our forms must have two endpoints: the first will simply display the form and the second will be POSTed to where the form is submitted and will process the form data appropriately. Add the following method to your controller:

```
// This endpoint simply returns the name of the view that will display the
// New Contact Form (newContactFormNoAjax.jsp).
//
// NOTE: This method takes no parameters because it does not need to look
//       at the incoming request or put anything on the model.
@RequestMapping(value="displayNewContactFormNoAjax", method=RequestMethod.GET)
public String displayNewContactFormNoAjax() {
    return "newContactFormNoAjax";
}
```

Step 06 - Implement JSP to Display New Contact Form

Now, we need to create newContactFormNoAjax.jsp and add code to it so that it displays our New Contact form. Create the JSP and then modify it so that it looks like the following. Pay special attention to the fact that our <form> now has both an **action** and a **method**. Also note that each of our inputs now have a **name** attribute.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="s" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Company Contacts</title>
        <!-- Bootstrap core CSS -->
        <link href="${pageContext.request.contextPath}/css/bootstrap.min.css" rel="stylesheet">

        <!-- SWC Icon -->
        <link rel="shortcut icon" href="${pageContext.request.contextPath}/img/icon.png">

    </head>
    <body>
        <div class="container">
            <h1>Company Contacts</h1>
            <hr/>
        </div>

        <div class="container">
            <h1>New Contact Form</h1>
            <!-- #1 - Link to displayContactListNoAjax -->
            <a href="displayContactListNoAjax">Contact List (No Ajax)</a><br/>
            <hr/>
            <form class="form-horizontal"
                  role="form"
                  action="addNewContactNoAjax"
                  method="POST">
                <div class="form-group">
                    <label for="add-first-name"
                           class="col-md-4 control-label">First Name:</label>
                    <div class="col-md-8">
                        <input type="text"
                               class="form-control"
                               id="add-first-name"
                               name="firstName"
                               placeholder="First Name"/>
                    </div>
                </div>
                <div class="form-group">
                    <label for="add-last-name" class="col-md-4 control-label">Last Name:</label>
                    <div class="col-md-8">
                        <input type="text"
                               class="form-control"
                               id="add-last-name"
                               name="lastName"
                               placeholder="Last Name"/>
                    </div>
                </div>

<div class="form-group">
                    <label for="add-company"
                           class="col-md-4 control-label">Company:</label>
                    <div class="col-md-8">
                        <input type="text"
                               class="form-control"
                               id="add-company"
                               name="company"
                               placeholder="Company"/>
                    </div>
                </div>
                <div class="form-group">
                    <label for="add-email" class="col-md-4 control-label">Email:</label>
                    <div class="col-md-8">
                        <input type="email"
                                class="form-control"
                                id="add-email"
                                name="email"
                                placeholder="Email"/>
                    </div>
                </div>
                <div class="form-group">
                    <label for="add-phone" class="col-md-4 control-label">Phone:</label>
                    <div class="col-md-8">
                        <input type="tel"
                               class="form-control"
                               id="add-phone"
                               name="phone"
                               placeholder="Phone"/>
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-md-offset-4 col-md-8">
                        <button type="submit"
                                id="add-button"
                                class="btn btn-default">Add New Contact</button>
                    </div>
                </div>
            </form>

        </div>
        <!-- Placed at the end of the document so the pages load faster -->
        <script src="${pageContext.request.contextPath}/js/jquery-1.11.1.min.js"></script>
        <script src="${pageContext.request.contextPath}/js/bootstrap.min.js"></script>

    </body>
</html>
```

Step 07 - Create Controller Endpoint to Process New Contact Form Submission

Now, we will create the endpoint that will process the form submission, add a new Contact to the DAO, and redirect to the view that displays all of the Contacts in the system. Add the following method to your controller:

```
    // This endpoint gets the submitted form data from the HttpServletRequest,
    // creates a new Contact object, sets the fields on the new Contact
    // object appropriately, add the Contact to the DAO, and then redirects
    // to the displayContactListNoAjax controller endpoint.
    @RequestMapping(value="/addNewContactNoAjax", method=RequestMethod.POST)
    public String addNewContactNoAjax(HttpServletRequest req) {
        // Get all of the form data from the request
        String firstName = req.getParameter("firstName");
        String lastName = req.getParameter("lastName");
        String company = req.getParameter("company");
        String email = req.getParameter("email");
        String phone = req.getParameter("phone");

        // Create a new Contact and set all the fields
        Contact contact = new Contact();
        contact.setFirstName(firstName);
        contact.setLastName(lastName);
        contact.setCompany(company);
        contact.setEmail(email);
        contact.setPhone(phone);

        // Add the Contact to the DAO
        dao.addContact(contact);

        // Redirect to the displayContactListNoAjax controller endpoint - we must
        // use the redirect: here so that Spring MVC routes us to the controller
        // endpoint and not directly to a JSP.
        return "redirect:displayContactListNoAjax";
    }
```

Step 08 - Create Controller Endpoint for Deleting a Contact

In Step 04, we worked ahead just a bit. You'll notice that the JSP code generates an Edit link and a Delete link for each of the Contacts in the list. The Delete link URL is of the following format:

/deleteContactNoAjax?contactId=<*N*>

where *N* is the id of the Contact to be deleted. With those links in place, the only thing we have to do to complete the Delete functionality is implement a controller endpoint. Add the following method to your controller:

```
// This method gets the id of the Contact to be deleted from the
// HttpServletRequest and then asks the DAO to delete the Contact.  When
// finished, it redirects to the displayContactListNoAjax controller
// endpoint.
@RequestMapping(value="/deleteContactNoAjax", method=RequestMethod.GET)
public String deleteContactNoAjax(HttpServletRequest req) {
    // Get the id of the contact to be deleted from the HttpServletRequest
    int contactId = Integer.parseInt(req.getParameter("contactId"));

    // Ask DAO to delete the given contact
    dao.removeContact(contactId);


     // Redirect to the displayContactListNoAjax controller endpoint - we must
    // use the redirect: here so that Spring MVC routes us to the controller
    // endpoint.
    return "redirect:displayContactListNoAjax";
}
```

Step 09 - Create Controller Endpoint to Display Edit Contact Form

Next, we will create an endpoint that will cause the Edit Contact form to be displayed. The Edit Contact workflow will follow the same pattern as the Add Contact workflow: first display the form; second, process the form; and third, redirect back to the displayContactListNoAjax endpoint.

In Step 04, we worked ahead for the Edit functionality as well. Each of the Contacts displayed on the page has an Edit link of the following format:

/displayEditContactFormNoAjax?contactId=<*N*>

where *N* is the id of the Contact to be edited. Add the following method to your controller:

```
// This method gets the id of the Contact to be edited from the HttpServletRequest,
// retrieves the specified Contact from the DAO, puts the retrieved Contact
// on the Model, and returns the name of the view that will display the
// Contact data in the Edit Form.
@RequestMapping(value="/displayEditContactFormNoAjax", method=RequestMethod.GET)
public String displayEditContactFormNoAjax(HttpServletRequest req, Model model) {
    // Get the id of the Contact to be edited
    int contactId = Integer.parseInt(req.getParameter("contactId"));
    // Get the Contact from the DAO
    Contact contact = dao.getContactById(contactId);
    // Put the Contact on the Model
    model.addAttribute("contact", contact);

    return "editContactFormNoAjax";
}
```

Step 10 - Create JSP to Display Edit Contact Form

Now, we need to create a JSP to display the Edit Contact form. This page will be similar to the JSP that displays the New Contact form except that the Edit Contact form will be pre-populated with the data of the Contact that is to be edited. Create a JSP called editContactForNoAjax.jsp and modify it so it has the following content. Pay special attention to the new taglib directive at the top (this allows us to use the Spring Forms tags) and the **modelAttribute** and **path** attributes on the form and form inputs. These attributes tell Spring how to grab and display the Contact information that we placed on the Model in the displayEditContactFormNoAjax controller method.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="s" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<!-- #1 - Directive for Spring Form tag libraries -->
<%@ taglib prefix="sf" uri="http://www.springframework.org/tags/form"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Company Contacts</title>
        <!-- Bootstrap core CSS -->
        <link href="${pageContext.request.contextPath}/css/bootstrap.min.css" rel="stylesheet">

        <!-- SWC Icon -->
        <link rel="shortcut icon" href="${pageContext.request.contextPath}/img/icon.png">

    </head>
    <body>
        <div class="container">
            <h1>Company Contacts</h1>
            <hr/>
        </div>

        <div class="container">
            <h1>New Contact Form</h1>
            <a href="displayContactListNoAjax">Contact List (No Ajax)</a><br/>
            <hr/>
            <sf:form class="form-horizontal" role="form" modelAttribute="contact"
                     action="editContactNoAjax" method="POST">
                <div class="form-group">
                    <label for="add-first-name" class="col-md-4 control-label">First Name:</label>
                    <div class="col-md-8">
                        <sf:input type="text" class="form-control" id="add-first-name"
                                  path="firstName" placeholder="First Name"/>
                    </div>
                </div>
                <div class="form-group">
                    <label for="add-last-name" class="col-md-4 control-label">Last Name:</label>
                    <div class="col-md-8">
                        <sf:input type="text" class="form-control" id="add-last-name"
                                  path="lastName" placeholder="Last Name"/>
                    </div>
                </div>
                <div class="form-group">
                    <label for="add-company" class="col-md-4 control-label">Company:</label>
                    <div class="col-md-8">
                        <sf:input type="text" class="form-control" id="add-company"
                                  path="company" placeholder="Company"/>
                    </div>
                </div>
                <div class="form-group">
                    <label for="add-email" class="col-md-4 control-label">Email:</label>
                    <div class="col-md-8">
                        <sf:input type="email" class="form-control" id="add-email"
                                  path="email" placeholder="Email"/>
                    </div>
                </div>
                <div class="form-group">
                    <label for="add-phone" class="col-md-4 control-label">Phone:</label>
                    <div class="col-md-8">
                        <sf:input type="tel" class="form-control" id="add-phone"
                                  path="phone" placeholder="Phone"/>
                        <sf:hidden path="contactId"/>
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-md-offset-4 col-md-8">
                        <button type="submit" id="add-button" class="btn btn-default">Add New Contact</button>
                    </div>
                </div>
            </sf:form>

        </div>

        <!-- Placed at the end of the document so the pages load faster -->
        <script src="${pageContext.request.contextPath}/js/jquery-1.11.1.min.js"></script>
        <script src="${pageContext.request.contextPath}/js/bootstrap.min.js"></script>

    </body>
</html>
```

Step 11 - Create Controller Endpoint to Process Edit Form Submission

Finally, we need to create a controller endpoint to process the Edit Contact form submission. Add the following method to your controller. Pay close attention to the @ModelAttribute annotation. Because we are using Spring Forms for the Edit Contact form, we can have Spring automagically convert the submitted form data into a Contact object for us.

```
// This method uses the @ModelAttribute annotation to tell Spring to
    // convert the submitted form data into a Contact object for us.  The method
    // then hands the Contact to the DAO for update, and finally redirects to
    // the displayContactListNoAjax controller endpoint.
    @RequestMapping(value="/editContactNoAjax", method=RequestMethod.POST)
    public String editContactNoAjax(@ModelAttribute("contact") Contact contact) {

        dao.updateContact(contact);

        return "redirect:displayContactListNoAjax";
    }
```

Wrap-up

That wraps up this demonstration of how to implement CRUD functionality Spring MVC web application without any JavaScript. Here are the key things to remember:

1. The values used for RequestMappings, the names of your JSPs, and the names of your controller methods **can** be the same, but they **do not have to be** the same.
2. If your controller method needs to get one or more values from the HTTP Request, you must have a parameter of type HttpServletRequest in your method signature.
3. If your controller method needs to pass any data to its corresponding JSP, you must have a parameter of type Model in your method signature. Your controller method is then responsible for adding the data the must be passed to the JSP on the Model object.
4. Every HTML form will have two endpoints: one that simply forwards control to the correct view so the form is displayed and one that processes the submitted form data.
5. If we use Spring Form tags, we can have Spring automagically convert Java DTOs into form values and submitted form values into Java DTOs.