Tutorial: Building a Web Application with Struts

This tutorial describes how OTN developers built a Web application for shop owners and customers of the Virtual Shopping Mall (VSM) sample application. OTN developers used Oracle9*i* JDeveloper and the Jakarta Project's Struts framework to build the application.

Contents

Concepts

The Jakarta Project's Struts framework, version 1.1b2, from Apache Software Organization is an open source framework for building web applications that integrate with standard technologies, such as Java Servlets, JavaBeans, and JavaServer Pages. Struts offers many benefits to the web application developer, including Model 2 implementation of Model-View-Controller (MVC) design patterns in JSP web applications. The MVC Model 2 paradigm applied to web applications lets you separate display code (for example, HTML and tag libraries) from flow control logic (action classes).

Following is a brief overview of the MVC Model 2 design pattern. For complete information about how Struts implements the MVC design patterns, see the *Introduction to the Struts User's Guide* on the Jakarta Project's Web site: http://jakarta.apache.org/struts/userGuide/index.html.

- The **Model** portion of an MVC-based system typically comprises JavaBean classes that define the internal state of the system; they also specify the actions that can be taken to change that state. If you use the BC4J data access framework, this layer implements the model entirely for you. Otherwise, you will need to create the classes that implement your model.

- The **View** portion of a Struts-based application is generally constructed using JSP technology. JSP pages can contain static HTML (or XML) text called "template text", plus the ability to insert dynamic content based on the interpretation (at page request time) of special action tags. The JSP environment includes a set of custom JSP tag libraries (such as the Struts tag libraries), standard JSP action tags (such as those described in the JavaServer Pages Specification), and a facility to install your own JSP custom tag libraries. If you use the BC4J data access framework, you can take advantage of JDeveloper's JSP generation wizards and the custom tag libraries that allow your JSP pages to display databound dynamic content.

- The **Controller** portion of the application is focused on receiving requests from the client (typically a user running a web browser), deciding what business logic function is to be performed, and then delegating responsibility for producing the next phase of the user interface to an appropriate View component. In Struts, the primary components of the Controller is a servlet of class ActionServlet and the class RequestProcessor. If you use the BC4J data access framework, the RequestProcessor is extended for you and is known as the BC4JRequestProcessor.

JDeveloper helps you implement the MVC Model 2 design patterns using core technology familiar to all web developers:

- You can create JSP pages with HTML and custom tag libraries to implement the View of the data. You use links to let the user trigger actions on the HTTP Request.

- You can enhance your JSP pages using a large set of custom JSP tag libraries that work with the Struts framework. All of the Struts tag libraries are accessible from the JDeveloper Component Palette, when you open a JSP in the Code Editor. For example, the Struts Form tag works closely with the Struts actions and form bean to retain the state of a data-entry form and validate entered data.

- Unlike non-Struts JSPs, when you run your application, action requests do not invoke another JSP or Servlet directly. Instead, the request URI specifies a logical page request, which the request processor (RequestProcessor class) provided by the Struts controller handles. The Struts servlet may direct the responsibility for displaying the action results to the appropriate JSP page of your application, where the page may vary according to the exit code.

The Struts framework includes custom JSP tag libraries that you can use to create JSP pages that work with the rest of the Struts framework objects in your web application:

Tag library	Description
Struts HTML	Used to create Struts input forms, as well as other tags generally useful in the creation of HTML-based user interfaces.

Struts Bean	Useful in accessing beans and their properties, as well as defining new beans (based on these accesses) that are accessible to the remainder of the page via scripting variables and page scope attributes. Convenient mechanisms to create new beans based on the value of request cookies, headers, and parameters are also provided.
Struts Logic	Useful in managing conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management.
Struts Nested	Brings a nested context to the functionality of the Struts custom tag library. The purpose of this tag library is to enable the tags to be aware of the tags which surround them so they can correctly provide the nesting property reference to the Struts system.
Struts Tiles	Provides tiles tags. Tiles were previously called Components.
Struts Templates	Three tags: `put`, `get`, and `insert`. A `put` tag moves content into request scope, which is retrieved by a `get` tag in a different JSP page (the template). That template is included with the `insert` tag.

Tags from the various Struts custom JSP tag libraries appear in JDeveloper on the Component Palette. The JDeveloper Help system lets you display the Struts Developer's Guide (obtained from the Apache Software Organization) for specific tags on the palette.

Design

While it is possible to develop Web applications that do not follow the MVC paradigm, in general the Struts framework provides significant advantages to the developer:

- Struts makes it possible for JSP pages to externalize flow control. Rather than specify physical links to various JSP pages within the JSP file, the JSP file contains a Struts-defined logical URI. The Struts URI defines a logical page request mapped to actions that may return different physical JSP pages depending on the context of the HTTP request.

- Struts simplifies the development of the actual JSP file content by limiting it to user interface generation only. Java that would otherwise appear inside the JSP files appears in separate servlet action classes that the JSP page invokes at runtime.

- Struts helps to separate the development roles into user interface designer (HTML or tag library user) and JSP action-handler developer. For example, one person can write JSP page using only HTML or suitable tag libraries, while another person works independently to create the page action handling classes in Java.

- Struts externalizes JSP actions that would otherwise appear inside all the JSP pages of your project into a single configuration file. This greatly simplifies debugging and promotes reuse.

- Struts consolidates String resources that would otherwise appear inside all the JSP pages of your project (for example, form labels) into a single file. This greatly simplifies the task of localizing JSP applications.

Required Software

This tutorial presents several code examples. If you want to study them in context, download and install the VSM source code. If you also want to build and run the VSM application, you will need the software listed in the Required Software section of *About the Virtual Shopping Mall*.

JDeveloper ships the source for the Struts framework in `<jdev_install>/jakarta-struts/` directory. This directory contains the same Struts package and Web application samples that you can download from the Jakarta Project's home page.

Setup

No special setup steps are required to use Struts with JDeveloper.

For information about setting up the VSM sample application, see the Setup section of the *Overview* tutorial.

Implementation

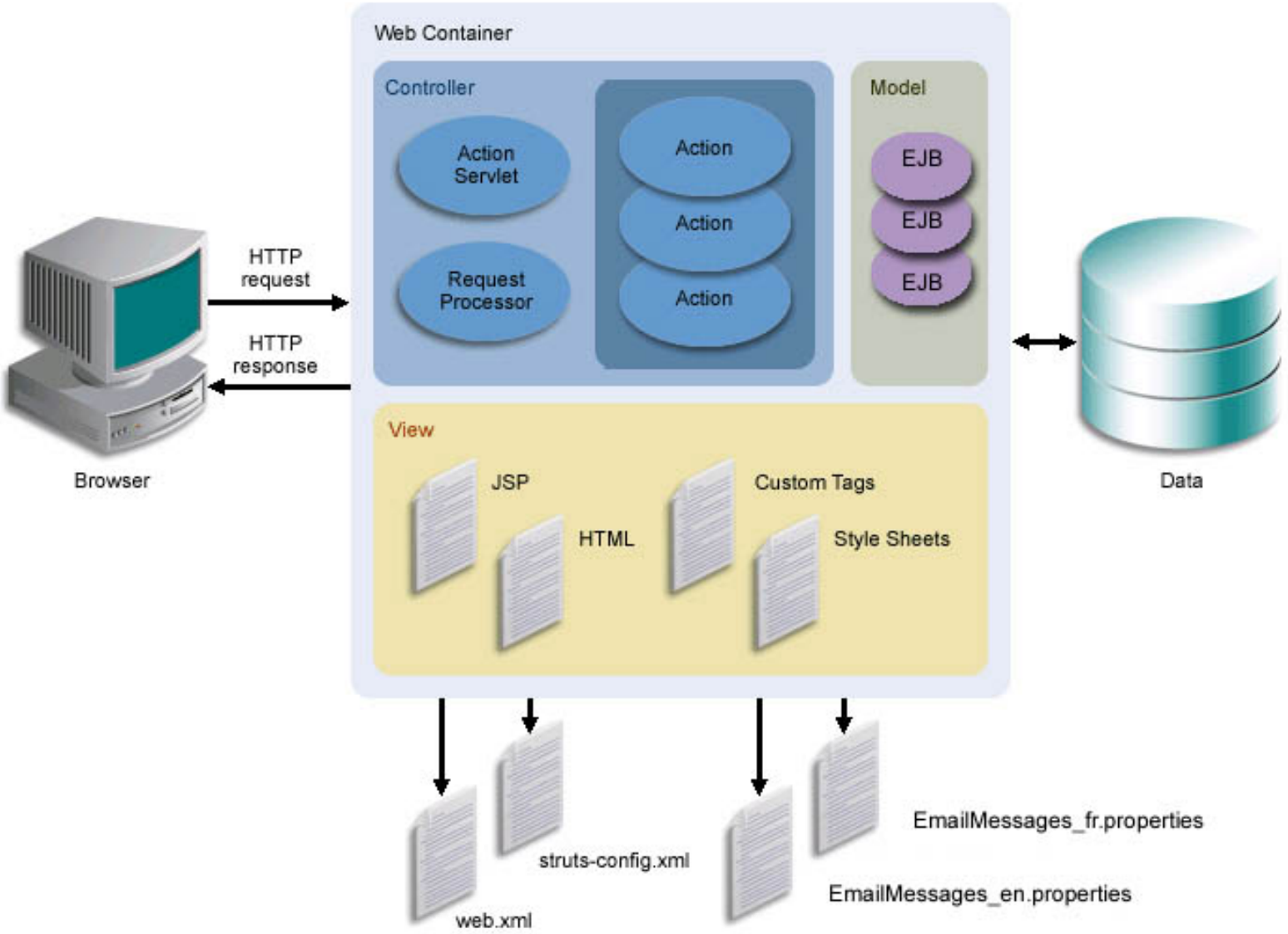
This section describes how the VSM uses Struts to display a list of orders to a mall customer.

The Struts technologies used are:

- Struts Controller Sevlet - MVC Controller servlet which controls all access to application.
- Struts Action - A java class designated to handle a Struts application Action which is in the form of a URL request.
- Struts ActionForm (Form Bean) - A Java class which represents the contents of an HTML form. It also provides a validation method.
- Struts "Bean" and "HTML" tag libraries - JSP tag libraries which allow for interaction with Struts components from JSP.
- Struts-Config.xml - The master configuration file for all Struts applications.
- ApplicationResources.properties - A resource file which can store multi-lingual messages and data for a Struts application.

JDeveloper provides these facilities to develop a web application based on Jakarta Project's Struts framework that you deploy as either Java servlets or JavaServer Pages.

- The Struts Configuration Editor lets you manage the struts-config.xml file, which defines the ActionMappings for your application. The Struts controller uses the mappings to turn HTTP requests into application actions.
- The various Struts dialogs in the Web Tier - Struts category of the New Gallery let you add the Struts framework components to a generic JSP web application.



All Struts actions begin with a URL that is submitted to the JSP request object when the end user clicks a Struts-defined link. The link must be of the form *`actionPathName`*`.do`, where the extension `.do` causes the `ActionServlet` to locate the corresponding action in the `struts-config.xml` configuration file.

The following HTML code (with `jbo` tags) comes from `mallUsers/allOrders.jsp`. It includes an HTML anchor tag that defines a link to `authusershoworders.do` and specifies parameters (e.g., `command=display`).

```
...
    <TD align="center" class="BlackText" width="25%">
      <A
         href="authusershoworders.do?command=display&orderID=
         <%=order.getId()%>" class="Link">
         <bean:message key="prompt.orders.details"/>
      </A>
    </TD>
...
```

The following XML code comes form `struts-config.xml`. It defines the action `authusershoworders`, specifying a Java class (`DisplayOrdersAction`) to handle the action, and associated JSPs (including `myOrders.jsp`).

```
...
    <action path="/authusershoworders"
            name="orderForm"
            parameter="command"
            type="oracle.otnsamples.vsm.actions.orders.DisplayOrdersAction">
      <forward name="display" path="/jsps/malluser/myOrders.jsp"/>
      <forward name="displayAll" path="/jsps/malluser/allOrders.jsp"/>
    </action>

...
```

The following Java code comes from `DisplayOrdersAction.java`. It implements the action method defined in the Struts configuration file (shown above). It imports Struts action classes (`org.apache.struts.action.*`) to gain access to Struts functionality. It returns an ActionForward object that the Struts framework will dispatch as defined in `struts-config.xml`. For example, if the code below executes without an error, it sets the return value to "display", and the Struts framework will invoke `/jsps/malluser/myOrders.jsp` as specified in the `struts-config.xml` file above.

```
package oracle.otnsamples.vsm.actions.orders;
...
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.actions.DispatchAction;
...
public class DisplayOrdersAction extends DispatchAction {
  /**
   * This is the action called from the Struts framework to get the list of
   * latest three orders for a given user with the complete information about
   * the most recent order
   *
   * @param mapping The ActionMapping used to select this instance.
   * @param form The optional ActionForm bean for this request.
   * @param request The HTTP Request we are processing.
   * @param response The HTTP Response we are processing.
   */
  public ActionForward display(
                              ActionMapping mapping, ActionForm form,
                              HttpServletRequest request,
                              HttpServletResponse response)
                              throws IOException, ServletException {
    OrderManagementHome orderHome = null;
```

```
    OrderManagement orderBean     = null;
    Order[] orders                = null;
    Order[] latestOrder           = new Order[ 1 ];
    ActionForward forward         = mapping.findForward("display");
    try {
      orderHome =
        (OrderManagementHome)
ServiceLocator.getLocator().getService("OrderManagement");
      orderBean = orderHome.create();
      String userName = (String) request.getSession().getAttribute("userName");
      // Get the latest three orders for the user and put the same in request scope
      orders = orderBean.getLatestOrdersForUser(userName, 3);
      request.setAttribute("orders", orders);
      /* If an order has been specified, get information about it, else fetch
         information about the most recent order for user */
      if(request.getParameter("orderID") != null) {
        latestOrder [ 0 ] =
          orderBean.getOrder((String) request.getParameter("orderID"));
      } else {
        latestOrder = orderBean.getLatestOrdersForUser(userName, 1);
      }
      request.setAttribute("latestOrder", latestOrder);
      // More each of the order items in the above order, get the item information
      HashMap itemInfo = new HashMap();
      List ordersList  = null;
      if(latestOrder != null && latestOrder.length > 0) {
        ordersList = latestOrder [ 0 ].getOrderDetail();
        String itemID = null;
        String langID = request.getLocale().getLanguage();
        for(int i = 0; i < ordersList.size(); i++) {
          itemID = ((OrderDetail) ordersList.get(i)).getItemID();
          itemInfo.put(itemID, orderBean.getItemName(itemID, langID));
        }
        request.setAttribute("itemInfo", itemInfo);
      }
    } catch(OrderException ex) {
      servlet.log("DisplayOrdersError", ex);
      ActionErrors errors = new ActionErrors();
      errors.add(
                 "OrderError", new ActionError("common.error", ex.getMessage()));
      saveErrors(request, errors);
      return mapping.findForward("error");
    } catch(Exception ex) {
      servlet.log("DisplayOrdersError", ex);
      ActionErrors errors = new ActionErrors();
      errors.add(
                 "OrderError", new ActionError("common.error", ex.getMessage()));
      saveErrors(request, errors);
      return mapping.findForward("error");
    } finally {
      try {
        if(orderBean != null) {
```

```
          orderBean.remove();
        }
      } catch(Exception ex) {
      }
    }
    return forward;
  }
...
```

The other tutorials in this series describe various application features and explain how they were implemented.

Resources

This tutorial is part of a series based on the Virtual Shopping Mall (VSM) sample application. Following are links to resources that can help you understand and apply the concepts and techniques presented in the tutorials. See the Required Software section to obtain the VSM source code and related files.

Resource	URL
Struts Overview	http://jakarta.apache.org/struts/
Struts Tutorials	http://jakarta.apache.org/struts/resources/tutorials.html
JDeveloper Online Help	http://otn.oracle.com/jdeveloper903/help/
Using Struts with JDeveloper	http://otn.oracle.com/products/jdev/htdocs/handson/struts/StrutsHandson.html
OTN Sample Code	http://otn.oracle.com/sample_code/

Feedback

If you have questions or comments about this tutorial, you can:

- Post a message in the OTN Sample Code discussion forum. OTN developers and other experts monitor the forum.
- Send email to the author. mailto:Robert.Hall@oracle.com

If you have suggestions or ideas for future tutorials, you can

- Post a message in the OTN Member Feedback forum.
- Send email to mailto:Raghavan.Sarathy@oracle.com.