The MVC Design Pattern & the Struts Framework

In previous episodes:
Model 1 design pattern
for every jsp page \( p \)
for every type of request \( r \) to \( p \)
insert in \( p \) code to implement the action requested by \( r \)

```
Students.jsp
```

- If request to insert student perform SQL INSERT
- If request to delete student perform SQL DELETE
- If request to update student perform SQL UPDATE

HTML part of the jsp

- INSERT STUDENT
- DELETE STUDENT
- UPDATE STUDENT

http://.../students.jsp?action=insert;
http://.../students.jsp?action=delete;
http://.../students.jsp?action=update;

Messy jsp!

MVC design pattern: Separate Model (Data Access), Controller & View

- Development “Best practice”
- Known well before web items
  - Smalltalk pioneered
- Model: Access to Underlying Databases and Info Sources
- Controller: Control Flow of Web App
- View: Look-and-Feel
The students example in MVC

The Larger Issue: Specification and Modularization

- Frictions in specification
- Inefficiencies in Large Project Management

The Process and the Frictions

- Informal, imprecise specification by customer
- Accompanied by hard-to-build demos and diagrams
- Code developed may be inconsistent with spec
- Significant effort in communicating spec formally
- Problem is even worse in evolution phase when business logic is hidden in thousands of code lines
The Problem: Communication

Struts: black-box framework that implements MVC

- Framework: reusable "partial" application
  - Struts' ActionServlet provides high level control of workflow
- You provide action classes and jsp files to customize framework into your application
  - Jsp's provide HTML presentation
  - ActionForm beans "collect" form data
  - Action beans provide details of flow
  - Struts-config.xml declares beans and jsp's

How to Develop a Struts Applications – 10 miles high

- Pass high-level control to ActionServlet
  - By appropriate URL mapping in web.xml
- Design "master plan" in diagrams and then code it in struts-config.xml
- Develop ActionForm bean(s)
- Develop Action bean(s)
- Develop model beans
- Develop html and jsp pages
Example

- The following example follows Struts 1.2
- See online notes on Struts 1.3 and Struts 2
- The example implements a dummy log-on functionality. Do NOT consider this example to be the best way to implement authorization and access control. It’s not. It’s just an example. For real authorization and access control stick to the notes of previous lecture.

WorkFlow: Automaton of pages & actions

Welcome.jsp
- "Sign in" logon
- logonForm
- username
- password
- VALIDATE

Logon.jsp
- /LogonSubmit.do
- success
- /LogOff.do
- success
- "Sign off" logoff
- UserDirectory
- getInstance()
- isValidPassword(String username, String password)
- UserDirectoryException
- !validated or UserDirectoryException
- user: logonForm
- set
- read
- welcome
- /Logon.do
- /Welcome.do
- ENTRY (http://…/logon/index.jsp)

index.jsp

```jsp
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<logic:forward name="welcome"/>
<!--
Redirect default requests to Welcome action.
--%>
```
Taglib declarations required in web.xml

<!-- Struts Tag Library Descriptors -->
<taglib>
<taglib-uri>/tags/struts-bean</taglib-uri>
<taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/tags/struts-html</taglib-uri>
<taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
<taglib>
<taglib-uri>/tags/struts-logic</taglib-uri>
<taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>

Edge names in struts-config.xml

<!-- =========Global Forward Definitions -->
<global-forwards>
<forward name="logoff" path="/Logoff.do"/>
<forward name="logon" path="/Logon.do"/>
<forward name="welcome" path="/Welcome.do"/>
</global-forwards>
logon.jsp
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<HTML>
<HEAD>
<TITLE>Sign in, Please!</TITLE>
</HEAD>
<BODY>
<html:errors/>
<html:form action="/LogonSubmit" focus="username">
<TABLE border="0" width="100">
<TR>
<TH align="right">Username:</TH>
<TD align="left"><html:text property="username"></html:text></TD>
</TR>
<TR>
<TH align="right">Password:</TH>
<TD align="left"><html:password property="password"></html:password></TD>
</TR>
<TR>
<TD align="right"><html:submit/></TD>
<TD align="left"><html:reset/></TD>
</TR>
</TABLE>
</html:form>
</BODY>
</HTML>

Declaration of an action class and its form

<action
  path="/LogonSubmit"
  type="app.LogonAction"
  name="logonForm"
  scope="request"
  validate="true"
  input="/pages/Logon.jsp">
  <forward
    name="success"
    path="/pages/Welcome.jsp"/>
</action>

Form Beans Also Provide Values

The corresponding ActionForm bean will also be used to populate html form...
public final class LogonForm extends ActionForm {
    private String password = null;
    private String username = null;

    public String getPassword() { return (this.password); }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getUsername() { return (this.username); }
    public void setUsername(String username) {
        this.username = username;
    }
    public void reset(ActionMapping mapping, HttpServletRequest request) {
        setPassword(null);
        setUsername(null);
    }
}

The Art of Balancing How Many Actions & JSP’s to Use

• Consider the “logon” application
• We could have two welcome jsp’s
  – welcomeNoneLogged.jsp, welcomeLogged.jsp
• We could have one jsp for each kind of login error
• Tags help consolidate within a few jsp’s
  – Form validation features
  – Logic tag library
• Deciding the number of actions and jsp’s is an art of design – not a science
  – Examples, practice, then more practice…

Validation

public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
    ActionErrors errors = new ActionErrors();
    if (username == null) || (username.length() < 1))
        errors.add("username", new ActionError("error.username.required");
    if (password == null) || (password.length() < 1))
        errors.add("password", new ActionError("error.password.required");
    return errors;
}
**Action Bean LogonAction**

```java
package app;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
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.action.ActionServlet;

public final class LogonAction extends Action {

/**
 * Validate credentials with business tier.
 * @param username The username credential
 * @param password The password credential
 * @returns true if credentials can be validated
 * @exception UserDirectoryException if cannot access directory
 */
    public boolean isUserLogon(String username, String password) throws UserDirectoryException {
        return (UserDirectory.getInstance().isValidPassword(username, password));
    }
}
```
public ActionForward perform(ActionMapping mapping, 
    ActionForm form, 
    HttpServletRequest request, 
    HttpServletResponse response) 
    throws IOException, ServletException {
    // Obtain username and password from web tier
    String username = ((LogonForm) form).getUsername();
    String password = ((LogonForm) form).getPassword();
    // Validate credentials with business tier
    boolean validated = false;
    try {
        validated = isUserLogon(username, password);
    } catch (UserDirectoryException ude) {
        // couldn't connect to user directory
        ActionErrors errors = new ActionErrors();
        errors.add(ActionErrors.GLOBAL_ERROR,
            new ActionError("error.logon.connect"));
        saveErrors(request, errors);
        // return to input page
        return (new ActionForward(mapping.getInput()));
    }
    if (!validated) {
        // credentials don't match
        ActionErrors errors = new ActionErrors();
        errors.add(ActionErrors.GLOBAL_ERROR,
            new ActionError("error.logon.invalid"));
        saveErrors(request, errors);
        // return to input page
        return (new ActionForward(mapping.getInput()));
    }
    // Save our logged-in user in the session,
    HttpSession session = request.getSession();
    session.setAttribute("user", form);
    // Log this event, if appropriate
    if (servlet.getDebug() >= Constants.DEBUG) {
        String message = new StringBuffer("LogonAction: User ").append(username);
        message.append(" logged on in session ").append(session.getId());
        servlet.log(message.toString());
    }
    // Return success
    return (mapping.findForward("success"));
}