The problems of Model 1: Mixes management of app state with application visualization & interface

Management of application state

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

HTML part of the JSP
- INSERT STUDENT
- UPDATE STUDENT
- DELETE STUDENT

Application Visualization & interface

http://.../students.jsp?action=insert&...
http://.../students.jsp?action=update&...
http://.../students.jsp?action=delete&...
for every JSP page \( p \)
for every type of request \( r \) to \( p \)
insert in \( p \) code to implement the action requested by \( r \)

The problems of Model 1: Have to account for all possible app state mgmt actions that may precede a page

for every JSP page \( p \)
for every type of request \( r \) to \( p \)
insert in \( p \) code to implement the action requested by \( r \)

The Problems of Model 1: Cannot decide dynamically the next page

Imagine a workflow where Page A is followed by either Page B or Page C, depending on the data that the user provided in page A
The problems of Model 1: Unclear wire diagram once Javascript comes in

- Are the page state changes induced by Javascript and Ajax supposed to count in a hire diagram?

Applications as Action – Page Cycles

- First, a purely server side perspective with MVC model
- Later, generalization to pages involving Javascript
The MVC Design Pattern: Separating Model, View & Controller

- Development “Best Practice”
- Known well before web items
  - Smalltalk pioneered
- **View**: Production of Pages
  - Typically templates
- **Controller**: Control Flow of Web App
  - “action” code
- **Model**: Access to Underlying Databases and Info Sources

---

Students Example

**students.jsp**

- HTML part of the JSP
- INSERT STUDENT
- UPDATE STUDENT
- DELETE STUDENT

View jsp, producing pages

Controller/Actions: Decide next step and changes of app state

Delete Student  Update Student  Insert Student

Model Java classes export methods that encapsulate SQL access

DB
Struts & Spring

- **Black-Box Framework Implementing MVC**
  - Framework: reusable “partial” application
  - **Struts** `ActionServlet` provides high level control of workflow (i.e., “what happens next”)
  - Spring is another famous Java-based MVC framework
- You provide Objects (Beans) and files to customize framework according to your application needs
  1. JSPs provide HTML presentation (View)
  2. `ActionForm` Beans “collect” form data (Part of Controller)
  3. `Action` Beans provide details of flow (Part of Controller)
  4. `struts-config.xml` declares Beans and JSPs

How To Develop Struts Applications

From 10 Miles High:
- Pass high-level control to `ActionServlet`
  - By appropriate URL mapping in `web.xml`
- Design action-page “workflow” in diagrams and then code it in `struts-config.xml`
- Develop `ActionForm` Beans responsible for collecting data
- Develop `Action` Beans responsible for changing the state of the application and deciding next step
- Develop Model Beans (not part of Struts) responsible for encapsulating database access
- Develop HTML and JSP pages for user interface
Struts Single Request Processing (cont’d)

1. When web app is loaded, ActionServlet parses struts-config.xml and associates URL paths with Action and ActionForm Beans
   - Location of struts-config.xml is given in web.xml
2. The user issues an HTTP request from an initiating page to the ActionServlet
3 The ActionServlet instantiates the ActionForm Bean associated with the HTTP request URL in struts-config.xml, and sets its properties using the HTTP request parameters (user-submitted data)

4 The ActionForm Bean validates its property values and if validation fails, ActionServlet responds with the initiating page P displaying appropriate error messages for the user to correct his/her form data

5 If validation succeeds, the ActionServlet instantiates the Action Bean associated with the HTTP request URL in struts-config.xml, and calls its execute method passing as parameters the ActionForm Bean, the HTTP request and the HTTP response objects
Within its `execute` method, the Action Bean instantiates/calls Model Beans, which open a connection to the database, execute SQL operations, and return sets of tuples. The Action Bean places the sets of tuples in the session so that JSP pages (View components) can access them.

The Action Bean returns to the ActionServlet one of the ActionForwards with which the HTTP request URL is associated in `struts-config.xml`. An ActionForward is a possible outcome of the Action Bean and represents either an JSP/HTML page or another Action that will be the response to the user’s request. Upon receiving the ActionForward, the ActionServlet responds to the user’s request with the corresponding JSP/HTML page or Action.
The example implements a dummy logon functionality
- Do **not** consider this example to be the best way to implement authorization and access control
- The example is used only to demonstrate the Struts framework
- Declarative authorization and access control for Phase 3 was covered previously
Welcome.do

Welcome User!
- Sign in
Logon.do

Sign in, Please!

Username: 
Password: 

Submit  Reset

LogonSubmit.do (no password)

Validation Error
You must correct the following error(s) before proceeding:

- Password is required

Sign in, Please!

Username: john 
Password: 

Submit  Reset
LogonSubmit.do (wrong password)

Validation Error
You must correct the following error(s) before proceeding:

- Username and password provided not found in user directory. Password must match exactly, including any lower or upper case characters.

Sign in, Please!

Username: john
Password: [hidden]

Submit  Reset

LogonSubmit.do (success)

Welcome john!

- Sign in
- Sign out
Logoff.do

```jsp
<%@ taglib uri="http://struts.apache.org/tags-logic"
    prefix="logic" %>
<logic:redirect forward="welcome"/>

<%--
Redirect default requests to Welcome global ActionForward.
--%>
```

/index.jsp

```jsp
<%@ taglib uri="http://struts.apache.org/tags-logic"
    prefix="logic" %>
<logic:redirect forward="welcome"/>

<%--
Redirect default requests to Welcome global ActionForward.
--%>
```
Welcome.jsp

```java
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic"%>
<html>
<head>
    <title><bean:message key="app.title" /></title>
    <html:base />
</head>
<body>

<logic:present name="user">
    <h3>Welcome <bean:write name="user" property="username" /></h3>
</logic:present>
...

<logic:notPresent scope="session" name="user">
    <h3><bean:message key="welcome.heading" /></h3>
</logic:notPresent>

<html:errors />

<ul>
    <li><html:link forward="logon">Sign in</html:link></li>
    <logic:present name="user">
        <li><html:link forward="logoff">Sign out</html:link></li>
    </logic:present>
</ul>

</body>
</html:html>
```

Welcome.jsp (cont'd)

```java
...
<html:errors />
<ul>
    <li><html:link forward="logon">Sign in</html:link></li>
    <logic:present name="user">
        <li><html:link forward="logoff">Sign out</html:link></li>
    </logic:present>
</ul>
```

...
Edge Labels In struts-config.xml

```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>
```

/pages/Logon.jsp

```jsp
...  
<body>
  <html:errors/>
  <h3><bean:message key="logon.heading" /></h3>
  <html:form action="/LogonSubmit" focus="username">
    <table width="100%">
      <tr><th align="right">Username:</th>
         <td><html:text property="username" /></td></tr>
      <tr><th align="right">Password:</th>
         <td><html:password property="password" /></td></tr>
      <tr><td align="right"><html:submit /></td>
         <td><html:reset /></td></tr>
    </table>
  </html:form>
</body>
..."
**Associating the ActionForm Bean with the HTML Form**

```xml
<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**

```xml
/pages/Logon.jsp
...
<h3><bean:message key="logon.heading" /></h3>
<html:form action="/LogonSubmit" focus="username">
    <table width="100%">
        <tr><th align="right">Username:<th></td>
        <td><html:text property="username" /></td></tr>
        <tr><th align="right">Password:<th></td>
        <td><html:password property="password" /></td></tr>
        <tr><td align="right"><html:submit /></td>
        <td><html:reset /></td></tr>
    </table>
</html:form>
...
### Typical Code Of A LogonForm Bean

```java
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 & JSPs to Use

- Consider the “logon” application
- We could have one JSP for each kind of login error
- However, we will see technologies that help consolidate within a few JSPs
  - Form validation features
  - Logic tag library
- Deciding the number of actions and JSPs is an art of design – not a science
  - Examples, practice, then more practice...
Validation

```java
public ActionErrors validate(ActionMapping mapping,
HttpServletRequest request) {

    ActionErrors errors = new ActionErrors();

    if ((username == null) || (username.length() < 1))
        errors.add("username", new ActionMessage("error.username.required"));

    if ((password == null) || (password.length() < 1))
        errors.add("password", new ActionMessage("error.password.required"));

    return errors;
}
```

Resource File & Internationalization

**MessageResources.properties**

- `app.title=Struts Logon Application`
- `welcome.heading=Welcome User!`
- `logon.heading=Sign in, Please!`
- `errors.header=<h3><font color="red">Validation Error</font></h3>You must...`
- `errors.prefix=<li>`
- `errors.suffix=</li>`
- `errors.footer=</ul><hr>`
- `error.username.required=Username is required`
- `error.password.required=Password is required`
- `error.logon.invalid=Username and password provided not found in user...`
- `error.logon.connect=Could not connect to user directory.
...`
package app;

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

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));
    // return true;
}
...

}
Action Bean LogonAction.java

... public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws Exception {
    // 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(…, new ActionMessage("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(…, new ActionMessage("error.logon.invalid"));
    saveErrors(request, errors);
    // return to input page
    return (new ActionForward(mapping.getInput()));
}
// Save our logged-in user in the session,
// because we use it again later.
HttpSession session = request.getSession();
session.setAttribute(Constants.USER_KEY, form);

// Return success
return mapping.findForward(Constants.SUCCESS);
}
} // End LogonAction
WorkFlow Variance

- The two variances share the same actions
- No modification in Actions source files
- Only need to change “view” (JSPs) and “workflow” (struts-config.xml)
Tag Libraries

- Collections of custom JSP tags
  - Hide Java from JSPs
  - Java classes in special format
  - Methods invoked with XML tags
  - Often looking like scripting languages
- Load one of the many on the web, or build your own
  - Unlikely given the plenty of choices available

- Struts Tag Libraries
- JSP Standard Tag Library (JSTL)

Struts Tag Libraries

- Four Struts Tag Libraries
  - html Generate HTML dynamically
  - bean Define beans, print bean properties, print localized strings
  - logic Manage conditionals, iterations, flow control
  - nested
**html Tag Library**

- Used to create input forms for your application
- A few other useful tags used in the creation and rendering of HTML-based user interfaces
  - `html:form`
  - `html:errors`
  - `html:password`
  - `html:submit`
  - `html:text`
  - `html:option`

---

**html Tag Library: Example**

```html
<html:form action="/login" method="POST">
  <h1>Login</h1>
  <html:errors/>
  <table>
    <tr>
      <td>User Name</td>
      <td><html:text property="userName"/></td>
    </tr>
    <tr>
      <td>Password</td>
      <td><html:password property="password"/></td>
    </tr>
    <tr>
      <td>&nbsp;</td>
      <td><html:submit value="Log in"/></td>
    </tr>
  </table>
</html:form>
```
**bean Tag Library**

- Used for creating and accessing Java Beans and a few other general purpose uses
- **bean:define**
  Define a scripting variable based on the value(s) of the specified bean property
- **bean:write**
  Render the value of the specified bean property
- **bean:message**
  Render an internationalized message string to the response

---

**bean Tag Library: Example**

```html
<html:html>
<head>
  <title>Bean Define, Bean Write Tags</title>
</head>
<body>
  <bean:define id="message"
              type="java.lang.String"
              value="First message string"/>
  <p><b><bean:write name="message"/></b></p>
</body>
</html:html>
```
**bean** Tag Library: Another Example

```xml
<html:html>
<head><title>
  <bean:message key="welcome.taglib.title"/>
</title></head>
<body>
  <h3>
    <bean:message key="welcome.taglib.heading"/>
  </h3>
  <p>
    <bean:message key="welcome.taglib.message"/>
  </p>
</body>
</html:html>

Message defined in:
\WEB-INF\classes\MessageResources.properties
```

**logic** Tag Library: Example

```xml
<logic:present name="itemsList">
  Items available for the selected color
  <br/>
  <logic:iterate id="item" name="itemsList">
    <br/>
  </logic:iterate>
</logic:present>

<logic:notPresent name="itemsList">
  No Items available for selected color
  <bean:write name="selectedColor"/>
</logic:notPresent>
```
JSTL

- **Core** Lib (prefix `c`): scripting language
- **Database** Lib (sql): support for DB
- **Functions** Lib (fn): string manipulation etc.
- **XML** Lib (x): XML support
- **Internationalization** Lib (fmt): formatting

Installation:
- JSTL requires `jstl.jar` and `standard.jar` located in `apache-tomcat-6.0.20/webapps/examples/WEB-INF/lib/
- Copy these two files into `apache-tomcat-6.0.20/lib/

JSTL Expressions

- Many JSTL tags have attributes whose values are JSTL expressions
  - Enclosed in `{$...}`
  - `<c:out value="${request.v}"/>`
    is shorthand for
    `<% String attr=request.getParameter("v");%>
    ...`  
    `<%= v %>`
  - Declare on top of your JSP
    `<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>`
Access to Bean Properties

- Assume your JSP uses a bean named `myBean` that has property `prop` and corresponding `getProp` and `setProp`

  ```
  <c:out value="${myBean.prop}"/>
  stands for
  <%= myBean.getProp() %>
  ```

Scripting Language Features

- Variable definition
  ```
  <c:set var="v"
    value="${...}"
    scope="session"/>
  <c:remove var="v"/>
  ```

- Weak typing
  - Strings to numbers
  - Integers to reals
  - and many more
Flow Control Tags

• `<c:if test="${booleanExpression}">`
  body of tag evaluated if boolean is true
  `</c:if>`

• `<c:choose>`
  `<c:when test="${booleanExpression}"">...`<c:when>`
  `<c:when test="${booleanExpression}"">...`<c:when>`
  `<c:otherwise>...`<c:otherwise>`
  `</c:choose>`

Iteration

• Iterate over start to end values, arrays, Collection, Iterator, Enumeration, Map, List, comma-separated string, etc.

• `<c:forEach var="i" begin="0" end="10" step="1">`
  loop body
  `</c:forEach>`

• `<c:forEach var="ck" items="${request.cookies}" items>`
  loop body
  `</c:forEach>`
### Struts Examples

- To play with Struts examples:
  - Copy struts-cookbook-1.3.10.war under \webapps
  - Access http://localhost:8080/struts-cookbook-1.3.10/

- To play with more Struts examples:
  - Copy struts-examples-1.3.10.war under \webapps
  - This automatically deploys a new web app directory
  - Access http://localhost:8080/struts-examples-1.3.10/

- To start your own Struts application:
  - Copy struts-blank-1.3.10.war under \webapps
  - Rename \struts-blank-1.3.10 to \your_app_name

### Pass Control to ActionServlet

URLs of requests ending with ".do", pass all .do requests to ActionServlet

**web.xml**

```xml
<web-app>
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```
Let’s reconsider the “students” example

Data Entry Example - 4th Attempt

Web Application Workflow
Data Entry Example - 4th Attempt

Web Application Workflow

![Diagram showing the workflow and request processing steps]

showStudents.do Request Processing

![Diagram showing the request processing steps and components]
showStudents.do Configuration

**struts-config.xml**

```xml
<struts-config>
  ...
  <action-mappings>
    <action
      path="/showStudents"
      type="dataentry.actions.ShowStudentsAction">
      <forward
        name="success"
        path="/pages/students.jsp"/>
    </action>
  ...
  </action-mappings>
  ...
</struts-config>
```

showStudents.do Action Bean

**ShowStudentsAction.java**

```java
package dataentry.actions;

import javax.sql.RowSet;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import dataentry.model.StudentModel;
...

class ShowStudentsAction extends Action {
  ...
```
showStudents.do Action Bean

ShowStudentsAction.java (cont’d)

    public ActionForward execute(
            ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) throws DBException {

            // retrieve all students
            RowSet crsStudents = StudentModel.getAllStudents();
            // store the RowSet in the request scope
            request.setAttribute("crsStudents", crsStudents);

            return mapping.findForward("success");
    }

showStudents.do Model Bean

StudentsModel.java

package dataentry.model;

    public class StudentModel {
        private static String selectStr = ...;
        private static String insertStr = ...;
        private static String updateStr = ...;
        private static String deleteStr = ...;

        public static CachedRowSet getAllStudents() {...}
        public static void insertStudent(StudentBean student) {...}
        public static void updateStudent(StudentBean student) {...}
        public static void deleteStudent(StudentBean student) {...}
    }
showStudents.do ActionForward

students.jsp
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
...
<%-- ------- Iteration Code ------- --%>
<% RowSet crsStudents = (RowSet)
      request.getAttribute("crsStudents");
      while (crsStudents.next()) { %>
<tr>
  ...
  <td>
    <html:text property="middle" size="15"
      value="<%= crsStudents.getString("middleName") %>" />
  </td>
  ...
</tr>
<% } %>
...

insertStudent.do Request Processing

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel

View
showStudents.do (Step 2 on Slide 8)

Controller
ActionServlet
InsertStudentAction
StudentFormInsertUpdate
struts-config.xml

Model
DB
StudentModel
insertStudent.do Configuration

**struts-config.xml**

```xml
...<form-bean name="studentFormInsertUpdate"
    type="dataentry.forms.StudentFormInsertUpdate"/>
...

<action
    path="/insertStudent"
    type="dataentry.actions.InsertStudentAction"
    validate="true"
    scope="request"
    input="/showStudents.do"
    name="studentFormInsertUpdate">
    <forward name="success" path="/showStudents.do"
        redirect="true"/>
</action>
```

insertStudent.do ActionForm Bean

**StudentFormInsertUpdate.java**

```java
package dataentry.forms;
...

public class StudentFormInsertUpdate extends ActionForm {

    private String id = null;
    private String first = null;
    private String middle = null;
    private String last = null;

    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
    ...
```
**insertStudent.do ActionForm Bean**

**StudentFormInsertUpdate.java (cont’d)**

```java
...  
/**  
 * Reset all properties to their default values.  
 */  
public void reset(ActionMapping mapping,  
HttpServletRequest request) {  
    setId(null);  
    setFirst(null);  
    setMiddle(null);  
    setLast(null);  
}  
...
```

**insertStudent.do ActionForm Bean**

**StudentFormInsertUpdate.java (cont’d)**

```java
...  
public ActionErrors validate(ActionMapping mapping,  
HttpServletRequest request) {  

    ActionErrors errors = new ActionErrors();  

    if ((id == null) || (id.length() < 1))  
        errors.add("idMsgTag1",  
                    new ActionMessage("errors.required", "ID"));  
...

    return errors;  
}  
```
**InsertStudentAction.java**

```java
public class InsertStudentAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws DBException {
        // cast the form
        StudentFormInsertUpdate iForm = (StudentFormInsertUpdate) form;
        // insert the student
        StudentModel.insertStudent(iForm);

        return mapping.findForward("success");
    }
}
```

**students.jsp**

```jsp
<%@ taglib uri="http://struts.apache.org/tags-html/" prefix="html" %>
...
<!-- in case form validation fails -->
<html:errors />
...
<tr>
    <html:form action="/insertStudent">
        <td><html:text property="id" size="10" /></td>
        <td><html:text property="first" size="15" /></td>
        <td><html:text property="middle" size="15" /></td>
        <td><html:text property="last" size="15" /></td>
        <td><html:submit value="Insert" /></td>
        <td><html:reset /></td>
    </html:form>
</tr>
..."
struts-config.xml Structure

```xml
<struts-config>
  <!-- Form Bean Definitions -->
  <form-beans>...</form-beans>
  <!-- Global Exception Definitions -->
  <global-exceptions>...</global-exceptions>
  <!-- Global Forward Definitions -->
  <global-forwards>...</global-forwards>
  <!-- Action Mapping Definitions -->
  <action-mappings>...</action-mappings>
  <!-- Message Resources Definitions -->
  <message-resources parameter="MessageResources" />
</struts-config>
```

Global Exceptions

struts-config.xml

```xml
<global-exceptions>
  <exception key="error.db"
    type="dataentry.db.DBException"
    path="/pages/dbException.jsp"/>
</global-exceptions>
```
Global Exceptions

**DBException.java**

```java
package dataentry.db;

public class DBException extends Exception {

    public DBException() {
        super();
    }

    public DBException(String message) {
        super(message);
    }
}
```

**StudentModel.java**

```java
public static void insertStudent(
    StudentFormInsertUpdate student) throws DBException {

    try {
        ...
    } catch (SQLException ex) {
        throw new DBException(ex);
    } catch (NamingException ex) {
        throw new DBException(ex);
    }
}
```
**Global Exceptions**

**InsertStudentAction.java**

```java
public class InsertStudentAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws DBException {
        ...
        StudentModel.insertStudent(...);
        ...
    }
}
```

**dbException.jsp**

```jsp
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>

<html>
<body>
<h2>Database Exception</h2>
...
<h3>Here is the message generated by the thrown database exception:</h3>

<p><html:errors /></p>

</body>
</html>
```
Global Forwards

struts-config.xml

<!-- ------------------------- Global Forward Definitions -->
<global-forwards>
  <forward name="showStudents" path="/showStudents.do"/>
</global-forwards>

menu.jsp

<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
<b>Data Entry Menu</b></noscript>
<ul>
  <li><html:link forward="showStudents">Students</html:link></li>
  ...
</ul>

Message Resources

MessageResources.properties

# -- app --
app.title=Struts Data Entry Application
...

students.jsp

<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
<html>
  <head>
    <title><bean:message key="app.title" /></title>
  </head>
  ...
</html>
Beyond pure server side models

Problem 1, on action/model perspective:
- Once Javascript/Ajax is available, the action may also be just Javascript code, or a combo of Javascript and server side code (Ajax).
  - From the point of view of the user, it does not matter whether the new page state is an update of the page state or it is produced from scratch by a server side jsp

Problem 2, on templating:
- Cool web sites use maps, barcharts and other Javascript/Ajax components
- Not just HTML any more

Components’ setting does not show in the jsp

...<div id="map_canvas">
<% for (int i = 0; i < list.size(); i++) {
  for (int j = 0; j < list.get(i).getDeliveries().size(); j++) {%>
<tr>
<td><% list.get(i).getId()%></td>
</tr>
<%}%>
</%>...
<table>
<thead>
<tr>
<th>Truck ID</th>
<th>Coordinates</th>
<th>Delivery ID</th>
<th>Scheduled Time</th>
<th>Delivered Time</th>
<th>Item Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>27.6635089.504395 -75.8805317405641</td>
<td>0</td>
<td>14:19</td>
<td>14:19</td>
<td>Item 3</td>
</tr>
<tr>
<td>0</td>
<td>27.6635089.504395 -75.8805317405641</td>
<td>1</td>
<td>14:19</td>
<td>14:19</td>
<td>Item 3</td>
</tr>
<tr>
<td>0</td>
<td>27.6635089.504395 -75.8805317405641</td>
<td>2</td>
<td>14:19</td>
<td>14:19</td>
<td>Item 3</td>
</tr>
<tr>
<td>0</td>
<td>27.6635089.504395 -75.8805317405641</td>
<td>3</td>
<td>14:19</td>
<td>14:19</td>
<td>Item 3</td>
</tr>
<tr>
<td>0</td>
<td>27.6635089.504395 -75.8805317405641</td>
<td>4</td>
<td>14:19</td>
<td>14:19</td>
<td>Item 3</td>
</tr>
<tr>
<td>0</td>
<td>27.6635089.504395 -75.8805317405641</td>
<td>5</td>
<td>14:19</td>
<td>14:19</td>
<td>Item 3</td>
</tr>
<tr>
<td>0</td>
<td>27.6635089.504395 -75.8805317405641</td>
<td>6</td>
<td>14:19</td>
<td>14:19</td>
<td>Item 3</td>
</tr>
</tbody>
</table>
**Action-Page Cycle**

An application executes in an **action-page cycle**.

1. To load a page, an action is invoked.
2. The action chooses a template to display.
3. The template is rendered to the browser.

**Unified Application State (UAS)**
- Page State
- Session
- REST Services
- SQL Database

**FORWARD Runtime**

**Actions**
- index.action

**Templates**
- index.template

---

**Example #1: Simple HTML**

- Displaying simple HTML is virtually identical to pure server-side frameworks.
- An action can be specified in different programming languages, such as Java and JavaScript. The example shows a JavaScript function.
- The action chooses a template to display.
- The template renders static HTML to the browser.

**index.action.js**
```javascript
function index() {
  display('index.template');
}
```

**index.template**
```html
<% template index() %>
<table>
  <tr>
    <th>Truck</th>
    <th>Latitude</th>
    <th>Longitude</th>
  </tr>
  <tr>
    <td>Truck #1</td>
    <td>32.7</td>
    <td>117.2</td>
  </tr>
  <tr>
    <td>Truck #2</td>
    <td>32.8</td>
    <td>118.1</td>
  </tr>
</table>
<% end template %>
```
Example #2: Visual Units

index.template

```ruby
<% template index() %>
<% unit google-maps %>
{
  markers : [
    {
      position : {
        latitude : 32.7,
        longitude : 117.2
      },
    },
    {
      position : {
        latitude : 32.8,
        longitude : 118.1
      },
    }
  ],
  <% end unit %>
<% end template %>
```

- Displaying visual units is similar to displaying simple HTML: provide markup and data.
- A **visual unit** is a wrapper around an existing JavaScript component library, such as charts and maps.
- The visual unit wrapper translates JSON data into the incremental rendering methods of the underlying JavaScript component.
- Notice that no JavaScript is needed to integrate the JavaScript component!

Action-Page Cycle

**FORWARD Runtime**

1. **Actions**
2. **Templates**
3. **Unified Application State (UAS)**
4. **Templates**

3. Typically, a template reads data that changes dynamically, instead of hardcoding static data.

- The Unified Application State (UAS) is a virtual database that connects to different sources, such as SQL databases, sessions, page state etc.
Example #3: Dynamic Data

A template has **views** that read data from the UAS. Each view can be specified as SQL++ queries or JavaScript.

- SQL++ queries extend SQL to support JSON data, including arrays, nesting, heterogeneity and optional schemas.
- Templates support for-loops, if-then-else and substitution for instantiating JSON data.
- Notice that there is no code for incrementally updating the visual unit. Incremental rendering is automatic and optimized within a FORWARD template.

Inlined data processing (no page model)

```plaintext
<% template index() using sqlpp %>
<% unit google-maps %>
{
    markers : [
        <% for truck in trucks %>
            {
                position : {
                    latitude : <%= truck.lat %>,
                    longitude : <%= truck.lng %>
                }
            }
        <% end for %>
    ]
}<% end unit %>
<% end template %>
```
**Action-Page Cycle**

FORWARD Runtime

1. The framework syncs the user input into the page state.
2. The action bound to the UI event is invoked.
3. The action reads and writes the UAS.
4. The rest of the action-page cycle is the same.

**Example #4: User Input**

```html
<% template index() using sqlpp %>
<% var chosen_size init 'small' %>
<% view trucks as
    select lat, lng
    from gps.trucks
    where size = chosen_size
%>
<% view sizes as
    select size, count(*)
    from gps.trucks
    group by size
%>
...
```

- A template has **variables** that store page state within the UAS.
- A variable is initialized when a template is first displayed.
- Variables can be updated by user input and actions. They are unlike views, which are read-only.
Example #4: User Input

index.template

```html
...<% unit dropdown %>
{
    options : [
        <% for size in sizes %>
            value : <%= size.size %>,
            label : <%= size.size + '(' + size.count + ')'>
        <% end for %>
    ]
    selected : <%= bind chosen_size %>
    <% event onchange chooseSize(chosen_size) %>
}<% end unit %>
...
```

- A visual unit typically displays data. For example, a dropdown box displays options.
- In addition, a visual unit accepts user input. For example, a dropdown box accepts user input for the selected option. The user's selection is bound into the variable chosen_size.
- A visual unit also supports events. For example, when the user selection changes, the onchange event fires, which invokes the action chooseSize().

chooseSize.action.js

```javascript
function chooseSize () {
    display('index.template');
}
```

- An action can simply display the corresponding template. Such trivial actions are very common, since displaying visualizations and user input can be handled exclusively by the template.
- An action can also produce side-effects, such as: charging credit cards, storing data into a data source, calling a REST service, writing log messages, etc.
High-level Abstractions and Automatic Optimizations

1. The UAS is an integrated database that abstracts away the location of data sources. FORWARD automatically optimizes for distributed queries across machines.

2. Views declaratively specify the data to be displayed. FORWARD automatically optimizes for how existing data is incrementally maintained to become current data.

3. Visual units declaratively specify the state of JavaScript components. FORWARD automatically optimizes for how to incrementally render the JavaScript components.

UAS Integrated Database and Distributed Queries

- The UAS is an integrated database that abstracts away the locations of data sources.
- In practice, a data source resides on the browser or the server.
- FORWARD automatically optimizes SQL++ queries, so that they execute efficiently when distributed between the browser and server.
**Distributed Query #1**

```sql
<% template index() using sqlpp %>
<% view server.trucks (chosen_size) as
    select lat, lng
    from gps.trucks
    where size = chosen_size
%>
<% view server.sizes
    select size, count(*)
    from gps.trucks
    group by size
%>
<% view browser.trucks (chosen_size) as
    server.trucks(chosen_size)
%>
<% view browser.sizes as
    server.sizes
%>
<% var browser.chosen_size init 'small' %>
...
```

- Based on data sizes, computation time, and network transfer, FORWARD determines the most efficient way to distribute a SQL++ query between being executed on the server versus the browser.

- For example, one common use case is to push SQL++ queries into the server. This is most efficient when `gps.trucks` is large, or when the server is significantly faster than the browser (esp. mobile devices).

---

**Distributed Query #2**

```sql
<% template index() using sqlpp %>
<% view browser.all_trucks as
gps.trucks
%>
<% view browser.trucks (chosen_size) as
    select lat, lng
    from browser.all_trucks
    where size = chosen_size
%>
<% view browser.sizes as
    select size, count(*)
    from browser.all_trucks
    group by size
%>
<% var browser.chosen_size init 'small' %>
...
```

- Another common use case is to push SQL++ queries into the browser. This is most efficient when `gps.trucks` is small enough to be cached in the browser, and network latency is relatively high.

- Between the two extremes of evaluating SQL++ queries entirely on the server or browser, there is a wide range of choices on how to distribute query evaluation between the two locations.
Views and Incremental View Maintenance (IVM)

FORWARD Runtime

- A view declaratively specifies the data to be displayed.
- In practice, naively evaluating a view from scratch is less efficient than incremental changes.
- FORWARD automatically optimizes to incrementally maintain views from existing data to current data.

Unified Application State (UAS)

- Page State
- Session
- REST Services
- SQL Database

Incremental View Maintenance #1

- FORWARD monitors data sources for data changes. For example, data changes in SQL databases are monitored through database triggers or the transaction log.
- When data changes occur, FORWARD propagates these incremental changes from the data sources all the way to the template.
- The data of the template (and of each view) are incrementally maintained with inserts, updates and deletes.

index.template

```text
<% template index() using sqlpp %>
<% view trucks as select lat, lng from gps.trucks where size = 'small' %>
<% unit google-maps %>
{
  markers : [
    <% for truck in trucks %>
    {
      position : {
        latitude : <%= truck.lat %>,
        longitude : <%= truck.lng %>
      }
    }
    <% end for %>
  ]
}
<% end unit %>
<% end template %>
```
Incremental View Maintenance #2

index.template

```html
<% template index() %>
Views can be specified in JavaScript.

<% if (isPublicHoliday()) {
    return findTrucks();
} else {
    // ...
} %>
Incremental changes on trucks have to be manually provided with a delta function delta_trucks.

<% unit google-maps %>
{
    markers : [
        <% for truck in trucks %>
            {
                position : {
                    latitude : <%= truck.lat %>,
                    longitude : <%= truck.lng %>
                }
            }
        <% end for %>
    ]
}<% end unit %>
FORWARD continues to propagate incremental changes to markers.

• For SQL++ queries, FORWARD automatically infers the delta queries necessary for IVM optimizations.

• For arbitrary JavaScript code (i.e. black box functions), IVM optimizations are semi-automatic. A delta function that produce the incremental changes has to be manually provided. FORWARD then continues to propagate these incremental changes to other views and the template.

• If no delta function is provided, FORWARD will evaluate the JavaScript code in full. Then, obtain incremental changes by comparing the old and new data, and continue to propagate these changes.

Visual Units and Incremental Rendering

• A visual unit declaratively specifies the state of a JS component.

• In practice, JS components support only API methods for incremental rendering, not declarative state.

• FORWARD automatically translates between incremental changes (of the state) and incremental rendering methods.
Visual Units #1

Google Maps Visual Unit

Construct Map
var m = new map(...);
...
setRootUiObject(m);

Destruct Map
var m = getRootUiObject();
...
$(m.getDiv()).remove();

Insert Marker
...

Delete Marker
...

Update Marker
...

FORWARD automatically finds most specific renderer.

Visual Units #2

Google Maps Visual Unit

Construct Map
var m = new map(...);
...
setRootUiObject(m);

Destruct Map
var m = getRootUiObject();
...
$(m.getDiv()).remove();

Insert Marker
...

Delete Marker
...

FORWARD simulates renderers when necessary.
Google AngularJS

- Google AngularJS is a browser-side web framework. It is one of the most popular browser-side web frameworks. Other frameworks include Facebook React, Meteor, Ember, Backbone.

- Angular follows a similar action-page cycle.

- Angular supports visual units with declarative state (using Angular directives) and incremental rendering optimizations.

- Angular supports views in JavaScript only. There is no IVM optimization: each JavaScript view is always evaluated in full. Angular compares the old and new data (using Angular watches) to produce incremental changes for incremental rendering.

- Angular resides exclusively on the browser-side, and is agnostic about server-side data. Thus, it does not support data abstractions across multiple locations, nor distributed query optimizations.