Design issues

- We have seen how to develop
  - Static web pages
  - Dynamic web pages
    - Using servlets & JSP
  - Accessing databases
    - Using JDBC

How do we put these together to make a simple application??

Solution: Use Design Patterns

Topics

- Design issues
  - Design patterns
- Model View Controller
  - MVC with RequestDispatcher
- Data Access Object Pattern

Design Patterns

- Based on years of experience in developing applications
- The pattern bible:
  Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
- See also:
Sun Design patterns-1

- Sun have categorised several design patterns.

http://java.sun.com/blueprints/patterns/catalog.html

Sun Design patterns-2

- Business Delegate
  - Reduce coupling between Web and EJB tiers
- Composite Entity
  - Model a network of related business entities
- Data Access Object (DAO)
  - Abstract and encapsulate data access mechanisms
- Fast Lane Reader
  - Improve read performance of tabular data
- Front Controller
  - Centralize application request processing
- Interception Filter
  - Pre- and post-process application requests

Sun Design patterns-3

- Model-View-Controller
  - Decouple data representation, application behaviour, and presentation
- Service Locator
  - Simplify client access to enterprise business services
- Session Facade
  - Coordinate operations between multiple business objects in a workflow
- Transfer Object
  - Transfer business data between tiers
- Value List Handler
  - Efficiently iterate a virtual list
- View Helper
  - Simplify access to model state and data access logic

Popular Design patterns

- Most popular design patterns are combination of:
  - MVC + Data Transfer Object + Data Access Object + Session Façade
    - Useful toolset to run is Apache Struts
      - http://jakarta.apache.org/struts/
    - Another standard is Java Server Faces (JSF)
    - Yet another is Spring MVC
Model View Controller


Session Facade

- Also called the **Value Object**
- Design transfers related information/objects as one object
- This improves performance and provides logical grouping of objects
- Provides a façade that encapsulates other objects.
- Usually implemented as a JavaBean ie: serializable

Data Transfer Object

Variations:

- **Data Transfer hash map** – instead of getter/setter methods, use attributes keywords get/put data. Passes hash map to backend object (eg: Session EJB)
**Data Access Object**

- Hide your database implementation from business logic
- Only expose interfaces that (usually) return a Data Transfer Object
- You can then update your DAO implementation as you change your technology

**Topics**

- Design issues
  - Design patterns
- Model View Controller
  - MVC with RequestDispatcher
- Data Access Object Pattern

**Model-View-Controller**

- Model-View-Controller (MVC)
  - a design pattern for building applications that interact with a user (not web-specific, not Java-specific)
- Model
  - code for internal representation of system state (data)
- View
  - code for displaying information to user (GUI)
- Controller
  - code for changing system state (logic)

**MVC as a FSM**

- MVC is often implemented as a ‘Finite State Machine’
- Each state usually represents
  - user-interaction (eg: web page)
  - system-state (eg: error)
- Each state transitions to another state using ‘command’ actions
**Sample MVC FSM**

![MVC FSM Diagram](image)

- **Welcome**
- **List**
- **Add**
- **Delete**

**GET /**
**GET /list.jsp**
**GET /add.jsp**
**GET /del.jsp**

- **error: “missing name”**
- **confirm: Are you sure?**

---

**MVC Architecture**

![MVC Architecture Diagram](image)

- **Model 2 = MVC for web applications**

---

**MVC Arch (cont’d)**

1. Browser makes HTTP request “Action” to controller
2. Controller performs any initialisation
   - eg: instantiates JavaBeans
3. JavaBean may retrieve external data (parallel to step 4)
4. Controller forwards request to appropriate JSP
   - JSP only contains presentation/view code
5. JSP accesses JavaBean data set up by controller
6. JSP returns response to browser *(NOT controller)*
7. Web page should have form action/links back to controller

---

**When to use MVC**

- **When:**
  - A single request will result in multiple substantially different-looking results.
  - You have a large development team with different team members doing the Web development and the business logic.
  - You perform complicated data processing, but have a relatively fixed layout.
Where do I put functionality?

- **Servlet:**
  - controls workflow
  - handles request
  - no presentation logic
  - instantiates any beans/objects used by JSP
  - may do processing logic
  - uses an action parameter to forward request to appropriate JSP
  - may have > 1 controller, best partitioned by logical functionality

- **JSP:**
  - provides presentation
  - no processing logic.
  - retrieves any beans/objects created by servlet
  - uses beans to display dynamic content

- **Beans:**
  - represents data
  - may have business methods
  - may read/write persistent storage eg: database
  - no presentation logic

Topics

- **Design issues**
  - Design patterns
- **Model View Controller**
  - MVC with RequestDispatcher
- **Data Access Object Pattern**

MVC with RequestDispatcher

1. Use JavaBeans to represent data
2. Use a servlet to handle requests..
   - Reads parameters, does some checking
   - Uses an ‘action’ parameter to decide what actions to take..
3. Servlet populate the beans..
   - invokes business logic & updates beans with results.
4. Servlet stores the bean..
   - Can be in the request, session, or servletContext
   - Use `setAttribute()` method to save bean
5. Servlet forwards the request to a JSP.
   - Decides which JSP to display next & uses `RequestDispatcher.forward()` to transfer control to that page.
6. JSP uses the data from the beans.
   - JSP accesses beans with jsp:useBean and a scope matching the location of step 4. The page then uses jsp:getProperty to output the bean properties.
**RequestDispatcher Example**

```java
public void doGet(...) {
    String action = request.getParameter("action");
    if (action.equals("order")) {
        address = "order.jsp";
    } else if (action.equals("cancel")) {
        address = "cancel.jsp";
    } else {
        address = "error.jsp";
    }

    RequestDispatcher dispatcher = request.getRequestDispatcher(address);
    dispatcher.forward(request, response);
}
```

This “action” determines which page “state” to display next.

**jsp:useBean**

- The JSP page should not create the beans
  - The servlet, not the JSP page, should create all the data objects. So, to guarantee that the JSP page will not create objects, you should use `<jsp:useBean ... type="package.Class" />` instead of `<jsp:useBean ... class="package.Class" />`

- The JSP page should not modify the beans
  - So, you should use `jsp:getProperty` but not `jsp:setProperty`.

**RequestDispatcher example -3**

- JSP

```html
...<form action="controller" method="GET">
Enter your details
Name: <input name="name" type=text><br>
Address: <input name="address" type=text><br>
<input name="action" type=hidden value="order">
<input name="submit" type="submit">
</form>
```

**recall: jsp:useBean Scope**

- `request` (within page & any forwarded pages)
  - `<jsp:useBean id="..." type="..." scope="request" />`

- `session` (within particular user/browser session)
  - `<jsp:useBean id="..." type="..." scope="session" />`

- `application` (to all users)
  - `<jsp:useBean id="..." type="..." scope="application" />`

- `page` (within page only)
  - `<jsp:useBean id="..." type="..." scope="page" />`
  - or just `<jsp:useBean id="..." type="..." />`
  - **Don’t use this in MVC applications!!!**
### Session Data Sharing

- **Servlet**
  ```java
  Person p = new Person();
p.setName("Chris");
HttpSession session = request.getSession();
session.setAttribute("cw", p);
RequestDispatcher dispatcher =
    request.getRequestDispatcher("/SomePage.jsp");
dispatcher.forward(request, response);
  
- **JSP 1.1**
  ```jsp:useBean id="cw" type="myApp.Person"
scope="session" />
  <jsp:getProperty name="cw" property="name" />
```  ```jsp:useBean id="cw" type="myApp.Person"
scope="session" />
  <jsp:getProperty name="cw" property="name" />
```  
  **only current user sees** "Chris"

### ServletContext Data Sharing

- **Servlet**
  ```java
  Person p = new Person();
p.setName("Chris");
getServletContext().setAttribute("cw", p);
RequestDispatcher dispatcher =
    request.getRequestDispatcher("/SomePage.jsp");
dispatcher.forward(request, response);
  
- **JSP 1.1**
  ```jsp:useBean id="cw" type="myApp.Person"
scope="application" />
  <jsp:getProperty name="cw" property="name" />
```  ```jsp:useBean id="cw" type="myApp.Person"
scope="application" />
  <jsp:getProperty name="cw" property="name" />
```  
  **all users see** "Chris"

### Topics

- **Design issues**
  - Design patterns
- **Model View Controller**
  - MVC with RequestDispatcher
- **Data Access Object Pattern**

### Data Access Object

- Hide your implementation from the business logic!
- **General tasks:**
  1. Create a DTO or Value Object (JavaBean!!)
  2. Create an interface for your DAO.
  3. Create implementation(s) for your DAO
JavaBean

- DTO, Value objects are implemented as JavaBeans
  - As per JSP lecture/labs
  - eg: Addressbook example

DAO Interface

- A java interface not a class
- Just declare public methods
- Usually:
  - Declare CRUD (Create, Read, Update, Delete) methods
  - Declare search/find methods
- Often simply named:
  - entityDAO
  - Eg: AddressbookDAO

Data Access Object

- Important Concept: *Hide your implementation from the business logic*
- Usual setup:
  - Constructor: initialisation stuff
    - Often drivermanager, connect to database, etc
    - Can be implemented as static, private instance or use the ThreadLocal interface (best)
  - Implement CRUD (Create, Read, Update, Delete) methods
  - Implement search/find methods
  - Any helper/utility methods
- Do not expose JDBC etc. eg: don’t throw SQLExceptions!!

Simple Example:

Javabean:

```java
public class Person implements Serializable {
    private String name;
    private String favecolour;

    public String getFavecolour() {return favecolour;}
    public void setFavecolour(String favecolour) {
        this.favecolour = favecolour;
    }

    public String getName() {return name; }
    public void setName(String name) {
        this.name = name;
    }
}
```
Simple Example:

Interface:

```java
public interface PersonDAO {
    public void createPerson(Person p);
    public Person findPerson(String name);
    public void updatePerson(Person p);
    public void deletePerson(String name);
    public List<Person> searchPerson(String name);
}
```

Implementation:

```java
public class PersonDAO_Impl {
    private Connection conn;

    public PersonDAO_Impl() {
        // constructor
        // set up JDBC here, connection etc
    }

    public void createPerson(Person p) {
        // implement INSERT into PERSON
        // VALUES(p.getName, p.getFavecolour) etc
    }

    public Person findPerson(String name) {
        // implement SELECT from PERSON where name = name
    }
    // & so on
}
```

Simple Example:

Using it:

**(in your business logic)**

```java
PersonDAO dao = new PersonDAO_Impl();
Person p = dao.findPerson("chris");
session.setAttribute("p", p);
```

**(in your JSP)**

```jsp
<jsp:useBean id="p" type="Person">
You found: ${p.name} With favecolor: ${p.faveColour}
```

Questions?
Simple Example

- Notes:
- If following an MVC pattern, you

Appendix

- Struts

Topics

- Model View Controller
  - MVC with RequestDispatcher

- Apache Struts
  - Struts components
  - Struts flow of control
  - 6 steps in using struts
  - Other tricks
    - customisable messages
    - Error handling
    - automatic validations
  - Quick reference

MVC & Apache Struts

- Writing MVC applications using RequestDispatcher can be tedious and error prone
- Most MVC applications follow a common pattern
- Apache Struts is an open-source product that can automate most of the work involved with writing MVC J2EE applications.

- Main Website: http://struts.apache.org
- See also:
  - http://www.husted.com/struts/
  - http://www.onjava.com/topics/java/JSP_Servlets
### What is Apache Struts?

- **An MVC Framework**
  - Provides an unified framework for deploying MVC web applications.
- **A Collection of Utilities**
  - Provides utility classes to handle common tasks in web application development
- **A Set of JSP Custom Tag Libraries**
  - Provides custom tag libraries for outputting bean properties, generating HTML forms, logic functions, validations
  - Note that Java Standard Tag Libraries (JSTL) replace many Struts custom tags now

### Advantages of Struts

- **Dynamic file-based configuration**
  - Uses deployment descriptors – no hardcoding into Java, uses XML & .properties files
- **Provides controller servlet**
  - Java and Web developers only focus on their specific tasks (implementing business logic, presenting certain values to clients, etc.)
- **Provides Form beans**
  - Automatically populate a JavaBean component. You can use dynamically generated form beans or write your own.
- **Provides Bean tags**
  - Provides custom JSP tags which extend jsp:useBean and jsp:getProperty tags.

### Advantages of Struts -2

- **Provides HTML tags**
  - Provides JSP tags to create HTML forms automatically associated with JavaBeans.
- **Provides Logic tags**
  - Able to do conditionals, iterations etc
- **Provides Form field validation**
  - Able to validate input fields on the server side.
  - Can even auto-generate client side Javascript
- **Consistent approach**
  - Struts forces consistent use of MVC throughout your application.

### Struts MVC Architecture

![Struts MVC Architecture Diagram](image-url)
Controller – ActionServlet

- Struts includes a controller servlet, called org.apache.struts.action.ActionServlet
- ActionServlet is responsible for mapping a logical request URI to an Action class
- This URI defaults to /*.do if you use the default web.xml file

- ActionServlet uses WEB-INF/struts-config.xml to determine which Action (business logic) class to invoke for a given action

Business Logic - Action

- Action class wraps the business logic of your application.
- You must extend org.apache.struts.Action and override the execute() method
- You should write one Action class per logical request
- Your Action class then returns control via a ActionMapping.findForward() method
- Best practices: Action should control the flow only and you should keep business logic in a separate class

Model - ActionForm

- You create a JavaBean for each Struts input form
- These “Form beans” extend the org.apache.struts.ActionForm class
- They basically only contain getter/setter methods
- Usually maps 1:1 to each input field in the form (and thus each request parameter)
- Struts can perform validations on these fields
- Struts can dynamically generate a bean for you
  - use DynaValidatorForm class in struts-config.xml

Model – other classes

- You should also implement other classes to represent the model
  - eg: system state beans, business logic beans, Enterprise Java beans
- These can be called by the various Struts components
  - Not required by Struts but just good coding practice
- Struts can also provide SQL DataSources
  - Use with JDBC
**View – JSP/Taglibs**

- Struts extends the JSP model by adding extra tags
- Use the `<%@ taglib uri="/WEB-INF/struts-*.tld" JSP directive`
- Many of these tags are optional – you can still use standard JSP scriptlets, tags & expressions
  - However, Struts provides lots of useful tags which assist the developer, especially when using forms!
  - More tags later...

- Struts makes it easy to have a centralised database of messages – good for internationalisation!
  - Usually defined in `ApplicationResources.properties` and `ApplicationResources_xx.properties`
  - Where `xx` is the ISO language code eg: fr de es cn

**Topics**

- Model View Controller
  - MVC with RequestDispatcher
- Apache Struts
  - Struts components
  - **Struts flow of control**
  - 6 steps in using struts
  - Other tricks
    - Customisable messages
    - Error handling
    - Automatic validations
  - Quick reference

**Struts Flow of Control**

1. The user requests a form
   - This form is built with html:form, html:text, html:link and similar elements
   - Keeps input field names in sync with bean property names

2. The form is submitted to `/xxx.do`
   - Where `xxx` is the name of the action in html:form
   - That `xxx` is mapped by `struts-config.xml` to an Action object

3. The execute method of the Action object is invoked
   - With Servlet details + form bean as parameters
   - Form bean is automatically populated with request parameters
   - Invokes business & data-access logic
   - Stores results in JavaBeans
     - In request, session, or application scope.
   - Returns mapping.findForward() with appropriate condition
   - These conditions are mapped by `struts-config.xml` to various JSP pages.
4. Struts forwards request to the appropriate JSP page
   • (also mapped in struts-config.xml)

5. resulting JSP page displays results
   • The page can use bean:write to output bean properties
   • The page can use bean:message to output fixed strings

The Six Basic Steps in Using Struts

1. Modify `struts-config.xml`.
   Use WEB-INF/struts-config.xml to:
   - Map incoming .do addresses to Action objects
   - Map return conditions to JSP pages
     • If a mapping appears in two places, it goes in global-forwards
   - Declare any form beans that are being used.
   - You may need to redeploy webapp after modifying struts-config.xml.

2. Define a form bean.
   - extends ActionForm and has a bean property for each incoming request parameter
   - The bean will also pre-populate the initial input form
   - You can also use DynaValidateForm to dynamically generate a form bean

3. Create results beans.
   - These are normal javabeanes, use as Data Transfer Objects (ie: just to pass data between classes)

4. Define an Action class to handle requests.
   - You don’t need to use request.getParameter(), a form bean is passed as argument to execute()
The Six Basic Steps in Using Struts

5. Create form that invokes /xxx.do.
   - Don’t use HTML <FORM> and <INPUT> tags, instead use <html:form> and <html:text> (and related tags).
   - <html:form> tag associates a bean with the form
   - <html:text> uses bean property names for each input field NAME and bean property values for each input field VALUE.
   - Optionally use <bean:message> tag to output standard messages and text labels.

6. Display results in JSP.
   - Use <bean:write> to output properties of the form and result beans.
   - Optionally use <bean:message> to output standard messages and text labels
   - Use <logic:*> tags to reduce need to Java scriptlets

   Can also use EL expressions eg: $(x)

The Six Basic Steps in Using Struts

Step 1: struts-config.xml

- Located in /WEB-INF
- Main entities:
  - <form-beans>
  - Defines javabeans which represent the model
  - <action-mappings>
  - this maps a virtual path (eg: /submit.do) to a specific class (eg: myapp.Submit)
  - <global-forwards>
  - like action-mappings, but handles default actions
  - <global-exceptions>
  - generic error catching
  - <message-resources>
  - allows you to define messages (eg: prompts) in a flat file
  - <plug-in>
  - extends struts with extra features such as a validator, dynamic beans etc

<form-beans>
  - For each input form, you create a bean which holds each input field
  - <form-beans>
    - <form-bean name="loginForm" type="myapp.loginForm"/>
    - <form-bean name="colorForm" type="org.apache.struts.validator.DynaValidatorForm">
      - <form-property name="name" type="java.lang.String"/>
      - <form-property name="favecolour" type="java.lang.String"/>
    </form-bean>
  </form-beans>

<action-mappings>
  - You create a <action> for each logical request
  - <action-mappings>
    - <action path="/login" scope="request"
      type="myapp.LoginAction"
      name="loginForm"
      input="/loginForm.jsp">
      - <forward name="success" path="/confirmation.jsp"/>
      - <forward name="failure" path="/failure.jsp"/>
    </action>
  </action-mappings>

Can also use action="/xxx"
global struts-config tags

- `<global-forwards>`
  ```xml
  <forward name="index" path="/index.jsp"/>
  ```
- `<global-exceptions>`
  ```xml
  <exception handler="myapp.myExceptionHandler"
    key="global.error.message"
    path="/error.jsp"
    type="java.lang.Exception"/>
  ```

misc struts-config tags

- `<message-resources`  
  ```xml
  parameter="ApplicationResources"/>
  ```

  customisable messages are stored in this file: 
  /WEB-INF/classes/ApplicationResources.properties

- `<plug-in`  
  ```xml
  <set-property property="var" value="1"/>
  ```

  you can extend struts by adding plug-in classes

Step 2: Form beans

```java
public class LoginForm extends ActionForm {
    private String password;
    private String customerNo;

    public String getPassword() {
        return this.password;
    }
    public void setPassword(String p) {
        this.password = p;
    }

    public String getCustomerNo() {
        return this.customerNo;
    }
    public void setCustomerNo(String c) {
        this.customerNo = c;
    }
}
```

DynaActionForm

- Alternatively: in Struts-config.xml:
  ```xml
  <form-bean name="loginForm"
    type="org.apache.struts.validator.DynaActionForm">
    <set-property name="password" type="java.lang.String"/>
    <set-property name="customerNo" type="java.lang.String"/>
  </form-bean>
  ```
Step 3: result beans

- Just any old java bean
- You can also create business logic beans or POJOs

* Plain Old Java Objects

Step 4: define Action class

```java
public class loginAction extends Action {
    public ActionForward execute (ActionMapping mapping,
                                  ActionForm form,
                                  HttpServletRequest request,
                                  HttpServletResponse response) throws Exception {
        // do some control & business logic here
        // maybe save form beans in the request, session or application
        if (ok) {
            return mapping.findForward("success");
        } else {
            return mapping.findForward("failure");
        }
    }
}
```

Step 5: Create invoking JSP

```html
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<html:form action="/login.do" method="post">
    <p><html:text property="customerNo" size="16" maxlength="18" /></p>
    <p><html:text property="password" size="8" maxlength="10" /></p>
    <html:submit>
        <bean:message key="button.login" />
    </html:submit>
</form>
```

Step 6: Display results JSP

- Use `<bean:write name="beanName" property="beanProperty" scope="session">` to display stored beans

```html
<html:html>
<body>
<h1>Welcome</h1>
<bean:write name="loginForm" property="customerNo"></h1>
<bean:message key="welcome.message" />
<ol>
    <li><html:link page="/addPerson.do">Add Person</html:link></li>
    <li><html:link page="/logout.do">Log Out</html:link></li>
</ol>
</body>
</html:html>
```
Topics

- Model View Controller
- MVC with RequestDispatcher
- Apache Struts
  - Struts components
  - Struts flow of control
  - 6 steps in using struts
- Other tricks
  - customisable messages
  - Error handling
  - automatic validations
- Quick reference

Other tricks

- You can have customisable messages stored in a properties file
- You can program struts errors and display with a simple tag
- You can perform manual or automatic validation of input forms

Customisable messages

1. Create a .properties file in WEB-INF/classes
   - e.g., WEB-INF/classes/ApplicationResources.properties
2. Define key/string pairs in the properties file
   - some.key1=my first message
   - some.key2=my second message
   - some.key3=some parameterized message: {0}
3. Load the properties file in struts-config.xml
   - <message-resources parameter="ApplicationResources"/>

Customisable messages 2

4. Output the messages in JSP pages
   - Load the tag library
     <%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
   - Output the messages using bean:message
     - First message is
       <bean:message key="some.key1"/>
     - Second:
       <bean:message key="some.key2"/>
     - Third:
       <bean:message key="some.key3" arg0="replacement"/>
**Advantages of Properties Files**

- **Centralized updates**
  - If a message is used in several places, can be updated with a single change!
  - Struts philosophy: "make changes in config files, not in code"

- **I18N**
  - You can internationalize your application
  - Use multiple properties files based on locale.
    - ApplicationResources.properties ← default
    - ApplicationResources_es.properties
    - ApplicationResources_fr.properties
    - ApplicationResources_cn.properties

**Loading Alternate Properties Files**

- **More specific properties file**
  - Struts automatically looks for additional Locale files
    - xxx_es.properties, xxx_fr.properties, etc.
  - Entries from more specific file override entries from default file
  - Locale is automatically determined by browser language settings
  - Locale can also be set explicitly (e.g., based on incoming checkbox value) in an Action with setLocale()

---

**Topics**

- Model View Controller
  - MVC with RequestDispatcher
- Apache Struts
  - Struts components
  - Struts flow of control
  - 6 steps in using struts
  - Other tricks
    - customisable messages
    - Error handling
    - automatic validations
  - Quick reference

**Programming struts errors**

- You can collect one or more error messages to be displayed

```
ActionErrors errors = new ActionErrors();
errors.add("loginerror", new ActionError("error.invalid.login","request.getParameter("customerNo")"));
saveErrors(request, errors);
```

- **display this in JSP via**
  - `<html:errors/>
  - `<html:errors property="loginerror">

  This is a key in the .properties file, not a displayable text string!
Struts errors - 2

- You also need to define the text string in the .properties file
  
  `error.invalid.login = {0} is an invalid login`

- Note: `{0}` represents a parameter.
  - You can have up to 4 parameters `{0} .. `{3}`

- You should have error headers and footers defined in the .properties file eg:
  
  ```
  errors.header=<font color="red">Error!</font><ul
  errors.prefix=<li>
  errors.suffix=<li>
  errors.footer=<li>
  ```

Topics

- Model View Controller
  - MVC with RequestDispatcher
- Apache Struts
  - Struts components
  - Struts flow of control
  - 6 steps in using struts
  - Other tricks
    - customisable messages
    - Error handling
    - automatic validations
- Quick reference

Automatic validation

- Update `struts-config.xml`
  
  ```
  <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property
      property="pathnames" value="/WEB-INF/validator-rules.xml, /WEB-INF/validation.xml"/>
  </plug-in>
  ```

- Update ApplicationResources.properties file with error messages
  
  ```
  errors.invalid={0} is invalid.
  errors.maxlength={0} cannot be greater than {1} characters.
  errors.minlength={0} cannot be less than {1} characters.
  errors.range={0} is not in the range {1} through {2}.
  errors.required={0} is required.
  ```

  ```
  #-- custom prompt messages
  loginForm.password=Password
  loginForm.customerNo=Customer Number
  ```

Validation

- Manual validation:
  - put validation code in Action – just usual Java code
  - put validation code in ActionForm
    - override ActionErrors validate()
  - Client side validation – using JavaScript

- Automatic validation:
  - Struts has validator plugin
  - This does server-side and optionally client-side validations
  - uses validation.xml and validation-rules.xml files to decide on the rules
  - Nice when it works.... can be very hard to debug...
Update validation.xml

• for each field in each form, add the following

  <formset>
  <form name="loginForm">
    <field property="password"
       depends="required"><arg0 key="loginForm.password"/></field>
    ...
    <field property="customerNo"
       depends="required,integer"><arg0 key="loginForm.customerNo"/></field>
  ...
  </form></formset>

• (optional) You can edit the validation-rules.xml file with custom validations

Final validation steps:

• Form bean:
  – should extend ValidatorForm instead of ActionForm
  – need to import org.apache.struts.validator.*
  – OR use the DynaValidationForm for struts-config.xml instead of DynaActionForm

• Put <html:errors/> in input page
  – locate it where you want error messages to appear
  – You can also show field specific errors by using the property attribute
    <html:errors property="password"/>

• (optional) Enable Javascript Validation
  – Put <html:javascript formName="loginForm"/> after <body>
  – Add onsubmit="return validateBeanName(this);" to <html:form>

Topics

• Model View Controller
  • MVC with RequestDispatcher

• Apache Struts
  • Struts components
  • Struts flow of control
  • 6 steps in using struts

• Other tricks
  • customisable messages
  • Error handling
  • automatic validations

• Quick reference

Struts tag quick reference

<table>
<thead>
<tr>
<th>HTML Tag</th>
<th>Struts Equivalent</th>
<th>Struts Added Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;html&gt; &lt;/html&gt;</td>
<td><a href="">html:html</a> &lt;/html:html&gt;</td>
<td>Adds locale support for internationalization</td>
</tr>
<tr>
<td>&lt;img&gt;</td>
<td><a href="">html:img</a></td>
<td>Adds capability to load from alt text and image from message resources file</td>
</tr>
<tr>
<td>&lt;base&gt; &lt;/base&gt;</td>
<td><a href="">html:base</a></td>
<td>Automatically inserts the Web application base</td>
</tr>
<tr>
<td>&lt;a&gt; &lt;/a&gt;</td>
<td><a href="">html:link</a> &lt;/html:link&gt;</td>
<td>Allows the link to be loaded from the request or other bean</td>
</tr>
</tbody>
</table>

from BEA Weblogic 8.1 Unleashed
<table>
<thead>
<tr>
<th>Function</th>
<th>HTML</th>
<th>Struts Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text field</td>
<td><code>&lt;input type = &quot;text&quot; name = &quot;myname&quot; size = &quot;20&quot;&gt;</code></td>
<td><code>&lt;html:text property = &quot;myname&quot; size = &quot;20&quot; /&gt;</code></td>
</tr>
<tr>
<td>Text area</td>
<td><code>&lt;input type = &quot;textarea&quot; name = &quot;mytextarea&quot;&gt;</code></td>
<td><code>&lt;html:textarea property = &quot;mytextarea&quot;&gt;</code></td>
</tr>
<tr>
<td>Radio button</td>
<td><code>&lt;input type = &quot;radio&quot; name = &quot;rad1&quot; value = &quot;sel1&quot;&gt;</code></td>
<td><code>&lt;html:radio name = &quot;rad1&quot; value = &quot;sel1&quot;&gt;</code></td>
</tr>
<tr>
<td>Check box</td>
<td><code>&lt;input type = &quot;checkbox&quot; name = &quot;chk1&quot; value = &quot;sel1&quot;&gt;</code></td>
<td><code>&lt;html:checkbox name = &quot;chk1&quot; value = &quot;sel1&quot;&gt;</code></td>
</tr>
</tbody>
</table>

Struts Logic tags

- You can check conditions on beans, values, cookies, HTTP headers and parameters
  `<logic:equal parameter="password" value="secret">`
  some JSP here `<logic:equal>`
- Boolean Conditions you can check
  - equal, notEqual, greaterEqual, lessEqual, greaterThan, lessThan
- Substring matching
  - Match, notMatch – these have extra parameter location to match start or end of substring
- Parameter matching
  - present, notPresent – if the parameter exists...

Struts logic tag

- One of the most useful tags is the `<logic:iterate>` tag
  - Can display lists of information on dynamic Web pages.
  - Attributes can be:
    - id - name of bean that contains the value for each element of the collection.
    - scope - The place to look for the attribute: page, request, session, application, or anyScope is allowed. If scope isn't present, the default value is page.
    - name - name of bean (collection) to iterate on.
    - type - class of object in each row of collection
    - length - maximum number of iterations.
  - eg:
    `<logic:iterate id="mydata" name="order" type="myApp.dataBean" scope="session">`
    `<dt><bean:write name="mydata" property="item">`
    `<dd><bean:write name="mydata" property="qty">`
    `</dd>`
    `</dt>`
    `</logic:iterate>`
Summary

• A lot to learn. Most struts courses run over several days!
• We have not covered struts features such as
  – Tiles – you can re-use presentation
  – Datasources - Struts has a limited ability to define datasources independently of the web application server custom validations
  – More specialised Action classes you can extend
    • DownloadAction (file uploads), ForwardAction, IncludeAction, LocaleAction, SwitchAction, DispatchAction
  – sub-applications

References

• http://www.coreservlets.com
• http://struts.apache.org
• http://www.onjava.com/topics/java/JSP_Servlets
• http://javaboutique.internet.com/tutorials
  – (especially ../strutsform/index.html which has a great tutorial on how to use multivalued forms)
• http://www.ibm.com/developerworks/java/

• Also read up on Java Server Faces, which may replace Struts eventually...
  – http://java.sun.com/j2ee/javaserverfaces/