CSE 510
Web Data Engineering
The Struts 2 Framework
What’s The Difference?

• A new framework that implements the MVC
  – It is said to be simpler for development

Features:

• **Action**: implements an **Action** interface along with other interfaces, or extends the **ActionSupport** class

• **Validation**: through XWork validation framework

• **Action Execution**: different lifecycles
Auction MVC Workflow – Struts 1

Start
welcome

Welcome.do
forward
PageA1.jsp

Place Bid (button)
!
validate

bidForm1
CheckBid1.do

highestBid
success
PageB.jsp

Buy (button)
ccForm
Buy.do

notHighestBid

PageA2.jsp

!
validate
!
validate
Place Bid (button)

bidForm2
CheckBid2.do

notHighestBid

highestBid

Auction MVC Workflow – Struts 2

Start

show.action

result

PageA1.jsp

Place Bid (button)

input error

PageA2.jsp

checkBid.action

highestBid

success

PageB.jsp

Buy (button)

error

buy.action
Deploy *StrutsPrepareAndExecuteFilter*

**web.xml**

```xml
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>
        org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
    </filter-class>
</filter>

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```
How To Migrate From Struts 1 (cont’d)

Deploy **StrutsPrepareAndExecuteFilter**

- A filter is a lightweight servlet that doesn't generate a response, instead it executes in addition to the normal request handling process.
- Struts2 filter does not have a parameter that defines the names of the configuration files.
- The default configuration file for Struts2 is `struts.xml` and needs to be on the classpath of the web application.
How To Migrate From Struts 1 (cont’d)

• Rewrite the workflow configuration file
  - The logic itself is very similar to Struts 1

`struts.xml`

```xml
<action name="buy" class="app.actions.Buy">
  <result name="success">/pages/PageA1.jsp</result>
  <result name="error">/pages/PageB.jsp</result>
</action>
```

• By default, the extension is `.action` instead of `.do`
  - Defined in the `default.properties` file (within the Struts2 JAR file) as the `struts.action.extension` property
• Rewrite the workflow configuration file
  – The logic itself is very similar to Struts 1

**struts.xml**

```xml
<action name="checkBid" class="app.actions.CheckBid">
  <result name="input">/pages/PageA1.jsp</result>
  <result name="error">/pages/PageA1.jsp</result>
  <result name="highestBid">/pages/PageB.jsp</result>
  <result name="notHighestBid">/pages/PageA2.jsp</result>
</action>
```
• Rewrite the action
  – For the Action, there is no ActionForm bound to it anymore
  – Instead, Action itself contains the form data

```java
public class Buy extends ActionSupport {

  private String cardNum = null;
  // setter and getter for the variable

  public String execute() { ... }
}
```
• Rewrite JSP pages
  – Struts 2 uses new taglibs
  – Provides better support for client-side programming (Ajax, JavaScript)
  – Conceptually, they are the same as Struts 1

```html
<%@ taglib prefix="s" uri="/struts-tags" %>
<s:form action="checkBid" method="POST">
  <s:actionerror />
  <s:textfield name="itemID" label="Item ID"/>
  <s:textfield name="bidPrice" label="Bid"/>
  <s:submit value="Place Bid" align="center"/>
</s:form>
```
Form Validation

- Static validation
  - Write the configuration in a XML file

**CheckBid-validation.xml**

```xml
<validators>
  <field name="itemID">
    <field-validator type="requiredstring">
      <param name="trim">true</param>
      <message>Item ID is required</message>
    </field-validator>
  </field>
</validators>
```
Form Validation (cont’d)

• Dynamic Validation
  – Action implements Validateable interface

```java
public class Buy extends ActionSupport implements Validateable {

  ...

  public void validate() {
    if (...) {
      addActionError("...");
    }
  }
}
```
In many cases, you need to access objects across pages
  - Example: Pre-populate form with values
Trivial in Struts 1, as you can save a FormBean in session or request
In Struts 2, there is no FormBean concept
**Solution:** Action implements the `ModelDriven` interface, and overrides the `getModel()` method
When a form that is bound to the action is rendered, `getModel()` will be called automatically to obtain the bean object
public class MyAction implements ModelDriven, ServletRequestAware {
    private HttpSession session;
    private HttpServletRequest request;
    private FormBean bean; // objects defined by me
    public void setServletRequest(HttpServletRequest request) {
        this.request = request;
        this.session = this.getSession();
    }
    public FormBean getModel() {
        bean = (FormBean) session.getAttribute(FormBean.NAME);
        if (bean == null) {
            bean = new FormBean();
            bean.setAttribute(FormBean.NAME, bean);
        }
        return bean;
    }
    public static class FormBean {  // setter getter  }
}