Struts And JavaServer Faces

Craig R. McClanahan
Senior Staff Engineer
Sun Microsystems

Agenda

- Background
- JavaServer Faces from a Struts Perspective
- Extending JavaServer Faces – Shale Framework
- Graphical JavaServer Faces – Java Studio Creator
- Summary

Background – The Origin of Struts

- Like many open source projects, Struts started with me “scratching my own itch”
  - Take a U.S. centric application to Europe ...
  - Supporting multiple languages ...
  - And make it available on the web
- I was familiar with Java and Open Source
  - Apache JServ (predecessor to Tomcat)
  - Apache Tomcat
- No good architectural models to follow

Background – The Origin of Struts

- Early versions of the JavaServer Pages (JSP) specification became available
- Version 0.91 described two basic approaches:
  - *Model 1* – Resource is responsible for both creating a page's markup *and* processing the subsequent submit
  - *Model 2* – Separate resources are responsible for creating a page's markup and processing the submit
- The second approach sounded better:
  - Separate resources for writing HTML and accessing DBs
  - So they can be built by different people ...
  - Perhaps using different tools

Background – The Origin of Struts

- I built a “home grown” architecture
  - Implemented a *Model 2* design
  - Based on the *model-view-controller* (MVC) design pattern
- Contributed to Apache in June, 2000
  - Release 1.0 occurred approximately one year later

Model-View-Controller Terminology

- *Model* – Persistent data and the business logic that processes that data
  - Often subdivided into *persistence* and *business logic* tiers
- *View* – The interface with which the user interacts
  - In webapps, the HTML markup displayed in the browser, as well as *state* information about input field values
- *Controller* – Management software to perform a "request processing lifecycle" on all requests:
  - Uniform enforcement of login/access restrictions
  - Consistent processing of all incoming requests

In The Mean Time ...

- A standardization effort (JSR-127) was started:
  - Harvest good ideas from existing implementations
  - Portable component API for interoperable components
  - For 1.0, focus *mostly* on components, and provide extension points for frameworks
- Goals of the original JSR:
  - Accessible to corporate developers
  - Accessible to tools
  - Client device neutral
  - Usable *with* or *without* JSP
  - Usable *with* or *without* HTML

In The Mean Time ...

- Let's look at the features of JavaServer Faces ...
- From the perspective of someone already familiar with the Apache Struts Framework

First Concept – Components

- In a rich client world, AWT and Swing demonstrate the value of a *user interface component model*
- Components help you:
  - Cleanly partition view tier requirements
  - Encapsulate complex behaviors in simple components
  - Leverage parent-child component relationships
- Struts:
  - Does not really have a user interface component model
  - Closest analog is JSP custom tags that render HTML
  - Additional responsibilities assigned to *ActionForm* and *Action* classes provided by the application developer

First Concept – Components

- JavaServer Faces components:
  - Render markup (but are not limited to HTML)
  - Process input field values
    - Including conversion and validation
  - Fire server side events
  - Save and restore view tier intermediate state
- Net result – no more form beans

Components Are Arranged In A Tree

- JavaServer Faces components are organized into a hierarchical tree structure:
  - *View Root* is singleton root component of the tree
  - Components can have arbitrary numbers of children
  - In addition, components can have *facets*
    - Special purpose “named” children such as column headers

Components Are Arranged In A Tree

- Components can take responsibility for processing their children (both input and output):

```
<h:table id="mytable" var="customer" ...>
  <h:column>
    <f:facet name="header">
      <h:outputText value="Customer Name"/>
    </f:facet>
    <h:outputText value="#{customer.name}"/>
  </h:column>
</h:table>
```

Components Are Arranged In A Tree

- Or, components can simply delegate to children:

```
<h:form id="logonForm">
  <h:panelGrid columns="2">
    <h:outputText value="Username:"/>
    <h:inputText id="username"
                 value="#{logon.username}"/>
    <h:outputText value="Password:"/>
    <h:inputSecret id="password"
                   value="#{logon.password}"/>
    <h:commandButton value="Logon"
                     action="#{logon.authenticate}"/>
  </h:panelGrid>
</h:form>
```

Components Fire Server Side Events

- Struts does not have an event processing model
  - Except for handling a form submit by calling execute()
- JSF uses standard JavaBeans event model:
  - Event listeners can be registered on component instances
  - Components fire events when “interesting” things happen
- Two standard patterns for event firing:
  - *ActionSource* – Component fires event stating that a particular action component was activated by the user
    - Submit button, hyperlink
  - *EditableValueHolder* – Component fires event stating that the value of an input component was changed
- Event model is extensible

Flexible Rendering Strategy

- Struts rendering strategy has limitations
  - HTML tags render *only* HTML
  - HTML tags work *only* in JSP pages
- In JSF, responsibility for rendering is split from the component, creating a separate *Renderer* class
  - A set of *Renderers* is combined into a *RenderKit*
  - Supports reuse of components in different environments
    - HTML, WML, Xforms, SVG, ...

Flexible View Representation Strategy

- Struts supports *only* JSP for view representation
  - Although add-on alternatives exist (without custom tags)
- JSF supports alternative view representations as a first class principle
  - Support for JSP mandated for standard components
  - Support for JSP in (nearly) all third party components
  - JSF supports extensible *ViewHandler* for alternatives
- Example alternative view implementations:
  - Facelets (https://facelets.dev.java.net)
  - Shale Clay (http://struts.apache.org/struts-shale/)
  - SVG and XForms RenderKits from third parties

Binding To Model Data

- Struts *bean* and *html* tags support limited “pull” model:
  - Executed as page is rendered to retrieve dynamic data
  - <bean:write name=”foo” property=”bar.baz”/>
- JSP Standard Tag Library (JSTL) replaces many of these use cases based on standard syntax
  - <c:out value=”${foo.bar.baz}”/>
- Struts “EL Extension” library lets you do this in Struts

Binding To Model Data

- JSF components extend this binding functionality:
  - Syntax and semantics based on JSP/JSTL EL syntax
    - <h:outputText value="#{foo.bar.baz}"/>
  - Can bind any component property, not just value
    - <h:panelGrid id="logonForm" rendered="#{empty user}">
  - Input components also *push* data back into the model:
    - <h:inputText id="username" value="#{logonBean.username}"/>
  - EL evaluation semantics are extensible
    - This will be discussed more later

Automatic Creation Of Beans

- Struts creates *ActionForm* beans on demand, when processing a form submit
- JavaServer Faces generalizes this concept:
  - Create any bean on demand, stored in any scope
  - As a side effect of evaluating an expression
- JSF Managed Beans:
  - Declared in faces-config.xml file
  - Defines bean name, class, scope
  - Also supports initializing properties
    - To either literal values or via expressions
    - Basic “dependency injection” facility

Automatic Creation Of Beans

- Example managed bean declaration:

```
<managed-bean>
  <managed-bean-name>logon</managed-bean-name>
  <managed-bean-class>...</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <!-- Should we support "remember me" cookies? -->
  <managed-property>
    <property-name>rememberMe</property-name>
    <value>true</value>
  </managed-property>
</managed-bean>
```

Page Navigation

- Struts navigates based on the *ActionForward* returned from calling an *Action.execute()* method:
  - Transitions can be defined locally or globally
  - Returning null means “I have completed this response”
- JSF supports a similar strategy, based on three items:
  - Current view (same as Struts)
  - Output value that was returned (except it is a String)
  - Which action method on this view was invoked
    - Can be simulated with Struts “dispatch actions”
  - Returning null means “please redisplay the current page”

Backing Beans

- Struts applications generally use form beans (instanceof *ActionForm*) and one or more actions (instanceof *Action*) per JSP page:
  - Form bean properties are generally strings
  - Form beans generally stored in request scope
  - Form beans must extend *ActionForm* or be *ActionDynaForm*
  - Action classes are singletons
    - Cannot use instance variables for per-request state
  - Action is responsible for conversion and pushing data to the model tier

Backing Beans

- JavaServer Faces applications generally have a single "backing bean" for each view:
  - No required base class or implemented interface
  - Typically are managed beans (for automatic creation)
  - Typically stored in request scope
  - Can use backing bean properties for request state *or* binding component instances
  - Components (not the application) are responsible for conversion to model data types
  - Application can be designed to push data to the model directly, or to have an action method perform this task

Logon Form Example

- JSP page for logging in to an application

```
<h:form id="logonForm">
  <h:panelGrid columns="2">
    <h:outputText value="Username:"/>
    <h:inputText id="username"
                 value="#{logon.username}"
                required="true"/>
    <h:outputText value="Password:"/>
    <h:inputSecret id="password"
                   value="#{logon.password}"/>
    <h:messages/>
    <h:commandButton value="Logon"
                     action="#{logon.authenticate}"/>
  </h:panelGrid>
</h:form>
```

Logon Form Example

- Managed bean declaration to define backing bean

```
<managed-bean>
  <managed-bean-name>logon</managed-bean-name>
  <managed-bean-class>
    com.mycompany.mypackage.MyLogonBean
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Logon Form Example

- Navigation rule to handle successful authentication

```
<navigation-rule>
  <from-view-id>/logon.jsp</from-view-id>
  <navigation-case>
    <from-action>#{logon.authenticate}</from-action>
    <from-outcome>success</from-outcome>
    <to-view-id>/mainmenu.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Logon Form Example

- Backing bean implementation class (part 1)

```
public class MyLogonBean { // No required base class

  // Property for username
  private String username = null;
  public String getUsername() { return this.username; }
  public void setUsername(String username)
    { this.username = username; }

  // Property for password
  private String password = null;
  public String getPassword() { return this.password; }
  public void setPassword(String password)
    { this.password = password; }
  ...
```

Logon Form Example

- Backing bean implementation class (part 2)

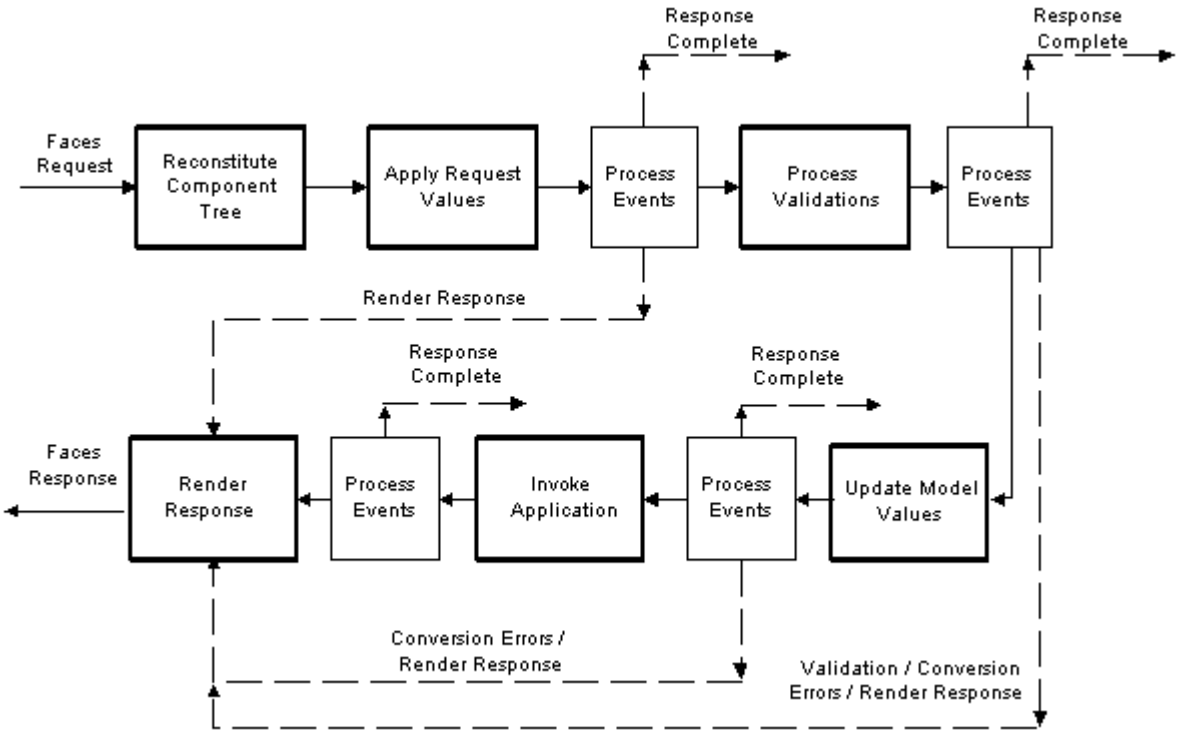
```
// The authentication method - simple signature
public String authenticate() {

  // Perform authentication
  Authenticator authenticator = ...; // Get biz logic
  if (authenticator.authenticate(username,password)) {
    return "success"; // Trigger navigation rule
  }

  // Report failure
  FacesContext.getCurrentInstance().addMessage(...);
  return null; // Redisplay the current page
}
}
```

Request Processing Lifecycle

- Struts runs all requests through an instance of *RequestProcessor*
  - Set locale from incoming request (or saved session flag)
  - Empty hook method for preprocessing
  - Identify the action to be invoked
  - Create and populate corresponding form bean (if any)
  - Perform server side validations (if specified)
  - Invoke action *execute()* method
  - Use returned *ActionForward* to navigate

Request Processing Lifecycle

- JSF runs all requests through lifecycle phases



Extension Points

- Struts allows substantial flexibility in customizing:
  - Custom *RequestProcessor* implementation
  - Custom configuration metadata classes (extra state info)
  - Base *Action* classes to share common functionality
- JSF offers extensive fine grained extension points:
  - Default action listener (analogous to Struts call to *execute*)
  - Navigation Handler (manage dialogs, authentication checks)
  - View Handler (non-JSP view representations)
  - State Manager (state saving and restoring)
  - VariableResolver and PropertyResolver (EL evaluation)
- Can use extension points to build framework on JSF

Configuration Metadata

- Struts is configured by *struts-config.xml* files:
  - Must be explicitly listed in /WEB-INF/web.xml
- JSF is configured by *faces-config.xml* files:
  - First, load META-INF/faces-config.xml files found in JAR files in the web application
  - Next, load explicitly listed *faces-config.xml* files
  - Finally, implicitly load */WEB-INF/faces-config.xml* if present

Configuration Metadata

- Struts is configured by *struts-config.xml* files:
  - Must be explicitly listed in /WEB-INF/web.xml
- JSF is configured by *faces-config.xml* files:
  - First, load META-INF/faces-config.xml files found in JAR files in the web application
  - Next, load explicitly listed *faces-config.xml* files
  - Finally, implicitly load */WEB-INF/faces-config.xml* if present

Comparison Summary So Far

- JavaServer Faces provides a rich UI component model
- But what about application oriented functionality?
  - Answer – build a framework *on top of* JavaServer Faces
  - Leverage the rich set of extension points
  - Do not waste time re-implementing redundant functionality
- We will examine one such framework in more detail:
  - Shale Framework
  - Developed by the Struts community
  - http://struts.apache.org/struts-shale/

Comparison Summary So Far

- JavaServer Faces provides a rich UI component model
- But what about application oriented functionality?
  - Answer – build a framework *on top of* JavaServer Faces
  - Leverage the rich set of extension points
  - Do not waste time re-implementing redundant functionality
- We will examine one such framework in more detail:
  - Shale Framework
  - Developed by the Struts community
  - http://struts.apache.org/struts-shale/

The Concepts Behind Shale

- Provide rich web application framework support:
  - Functionally equivalent to what is provided by Struts
- Build on top of JavaServer Faces:
  - Avoid re-implementing features that already exist
  - Leverage the extension points that are provided
- Provide leading-edge solutions to current problems:
  - Server side support for AJAX based components and apps
  - Optionally leverage features of Java SE 5 (“Tiger”) to reduce the amount of required configuration metadata
- These goals have been achieved

Shale Value Added Features

- *Tiles Framework* – Reuse defined layouts
  - Equivalent to corresponding Struts functionality
  - Implemented with “standalone” version of Tiles
    - Factoring out dependencies on core Struts APIs
- *Validator Framework* – Client and server validation
  - Equivalent to corresponding Struts functionality
  - Implemented using *Jakarta Commons Validator*
    - But exposed as a JavaServer Faces **Validator**
  - Enables client side JavaScript-based validation for JavaServer Faces components

Shale Value Added Features

- *View Controller* – Application oriented callbacks
  - Avoid need for “setup actions” often seen in Struts apps
  - Four event callbacks are provided:
    - Init() -- When corresponding view is created or restored
    - Preprocess() -- We are about to process a form submit
    - Prerender() -- We are about to render the current view
    - Destroy() -- After rendering has been completed

Shale Value Added Features

- *Dialogs* – Structured conversations with users
  - Dialogs are defined in terms of *states* and *transitions*
  - Three types of states:
    - *Action* – modelled as a JSF method binding expression
    - *View* – modelled as display of a view, plus following form submit
    - *Subdialog* – use other dialog definitions with “black box” reusability
  - Transitions between states are based on logical outcomes
    - Like standard JSF navigation rules
    - Transitions defined locally or globally like Struts ActionForwards
  - Can easily be modelled with a UML state diagram

Shale Value Added Features

- *Clay Plug In* – Full HTML views tied to separate component definitions:
  - Attractive when page authors wish to use standard HTML page authoring tools
  - Allows creation of reusable sets of components (more fine grained than Tiles)
  - Particularly attractive to developers familiar with Tapestry

Shale Value Added Features

- *Shale Remoting* – Perfect back end for AJAX apps
  - Serve static resources from webapp or JARs
  - Map incoming URLs to method binding expressions
    - Typically on a managed bean
  - Helper methods for JSF component authors
  - Small (40k), standalone (no dependency on rest of Shale)

Shale Value Added Features

- *Tiger Extensions* – Annotations based declarations reduce or eliminate need for XML configuration
  - Register JSF components, converters, renderers, validators
  - Define managed beans solely with annotations
  - Support view controller functionality without having to implement this interface
  - Totally optional – core of Shale requires only JDK 1.4

The Final Value – Tools Support

- It is possible to build apps with JSF and Shale that are functionally equivalent to Struts based applications
  - JSP pages have roughly the same complexity
  - Configuration files are slightly simpler
  - Java classes are slightly reduced in number, and have simpler calling sequences
- Want an example?
  - Struts MailReader application
  - Available as part of Struts 1.x releases
  - JSF version available in nightly builds of Shale

The Final Value – Tools Support

- In addition, JSF was designed to be “toolable”
  - Components can be self describing
  - Rendering can be performed at design time
  - Configuration files can be manipulated graphically
- Leading to the ability to deliver tools like **Java Studio Creator**
  - http://developers.sun.com/jscreator/
  - Version 2 released on January 26, 2006
  - Also available – technology preview of AJAX enabled components that work in the tool as well as at runtime
- Let's see a demo of components inside a tool ...

Summary

- Struts Action Framework is a robust, mature, framework for building web based applications in Java:
  - Stable release of version 1.3 is imminent:
    - Refactored request processor for easier customization
  - Merger with WebWork 2.x ensures ongoing improvements:
    - Combined action and action form
    - Interceptor based customization of action processing
    - Robust support for AJAX
- But Struts does not have a user interface component model ...

Summary

- If you need:
  - Sophisticated user interface components:
    - Available from multiple parties (because API is standardized)
    - Support simple and complex use cases
    - Best way to package AJAX functionality
  - Functional equivalence to Struts Action Framework from an application perspective
- You can have this today with:
  - JavaServer Faces
  - Shale Framework

Sun Developer Network

- Free resources for all developers
  - Tools (Creator, Java Studio Enterprise Edition)
  - Forums
  - FAQs
  - Early Access
  - Newsletters
- Join Today!
  - http://developers.sun.com

Thank You

Craig R. McClanahan
Craig.McClanahan@sun.com