Open Source Java Server Faces

Standardisation of presentation frameworks

Edwin Mol & Dimitry D'Hondt
Java Architects
Real Software
Overall Presentation Goal

Technical introduction to JavaServer Faces.

Learn how to create JSF pages and components using the open source implementation, 'Smile'.

Speakers

● Edwin Mol
  • Java Architect at Real Software.
  • 5+ Years of designing and implementing J2EE solutions.
  • Core developer of 'Smile'.

● Dimitry D'hondt
  • Java Architect at Real Software.
  • 3,5 Years of teaching J2EE + Designing and implementing J2EE applications.
  • Core developer of 'Smile'.
Problem.

Current web development
-> too many technical details.

We need to raise the level of abstraction.
(In standardised way.)
Agenda

- Introducing JSF.
- Basic structure. (Demo 1)
- 'Hello World !' in JSF. (Demo 2)
- Event handling.
- Writing a JSF component. (Demo 3)
- Data binding.
- State Management
- Struts vs. (or with ?) JSF.
- Smile : open source implementation.
Introducing JSF

- Component based presentation layer.
- Standardized (JSR-127)
  - Component market.
  - Reduces vendor lock-in.
- Raises the level of abstraction.
  - Real components. (now: page, scriptlets, tags, ...)
  - Designers (Productivity).
  - Dynamic page construction.
  - Simplifies. (Overall direction of JCP)
  - ...
- Answer to ASP.net
Basic structure (1/2)

● Component tree. (Composite - GoF)

● Example component tree:
  • partsPanel (Grid Layout Manager – 1 column)
    - addComponentPanel (Grid Layout Manager - 4 columns)
      Label 'parentLabel'
      Label 'typeLabel'
      Label 'idLabel'
      Dummy (filler)
      HtmlSelectOneMenu 'parentCombo'
      HtmlSelectOneMenu 'typeCombo'
      HtmlInputText 'identifier'
      HtmlCommandButton 'addButton'
    - ruler
    - resultsPanel
      Dynamically added components...
● JSF Lifecycle:

1. Faces Request
2. Restore Component Tree
3. Apply Request Values
4. Process Events
5. Process Validations
6. Process Events
7. Update Model Values
8. Process Events
9. Process Events
10. Process Events
11. Process Events
12. Render Response
13. Render Response
14. Render Response
15. Response complete
16. Response complete
17. Response complete
18. Conversion Errors/Render Response
JSP vs. Class-based Pages.

- Certain projects are document oriented. (e.g. On-line shop)
  - JSP based model.
    - Greatest control over layout.
    - Mix hand-coded HTML with JSF.
    - Combine with available JSP taglibs.
      Even Struts?

- Other projects are real **applications** in a browser.
  - Class based model.
    - No need to learn servlet/JSP technology
    - Clean model.
    - Simple.
Hello World! (1/2)

```html
<html>
  <head>
    <title>Welcome</title>
  </head>
  <body>
    <f:use_faces>
      <h:input_text value="hello JSP world!"/>
    </f:use_faces>
  </body>
</html>
```
public class HelloWorld implements Page {
    public void init(FacesContext ctx, UIComponent root) {
        Screen screen = new Screen();
        screen.setId("helloScreen");
        screen.setTitle("hello World application...");
        root.getChildren().add(screen);

        HtmlOutputLabel label = new HtmlOutputLabel();
        label.setValue("Hello World");
        screen.getChildren().add(label);
    }
}
Event handling (1/2)

- JSF has similar event handling model as Swing
- UIComponent emits events and broadcasts it to event listeners
- All events subclass javax.faces.FacesEvent
- Phase identifier indicates where in lifecycle event should be processed
- Method binding makes it possible to handle events without registering a listener
Event handling (2/2)

- Registering an Actionlistener
  - JSP based pages:
    ```
    <h:command_button label="Login">
      <f:action_listener
        type="custom.MyActionListener"/>
    </h:command_button>
    ```
  - Class based pages
    ```
    HtmlCommandButton loginButton = new HtmlCommandButton();
    loginButtonaddActionListener(new MyActionListener());
    ```
Example : Simple toolbar component.
- **ToolBar** class represents the toolbar.
- **ToolBarRenderer** class takes care of HTML rendering.
- **ToolBarButton** represents a single button on the toolbar.
- **ToolBarButtonPressedEvent** event that is fired when a toolbar button is pressed. (server-side)
- **ToolBarButtonPressedListener** event that is fired when a toolbar button is pressed. (server-side)

Demo.
- Buttons.
- Toggle buttons.
- Tooltips.
- Left/Right buttons.
- Separators.
● **Component class**

  • `public Object saveState(FacesContext ctx)`
    - Returns a serializeable object that contains the state of the component.
  
  • `public void restoreState(FacesContext ctx, Object state)`
    - Does the opposite.

• Event handler registration (javabeam style).
● Renderer class

- **public void encodeEnd(FacesContext ctx, UIComponent component)** throws IOException
  - Takes care of rendering out the current toolbar state as HTML.
  - Uses overlib to generate tooltips.

- **public void decode(FacesContext ctx, UIComponent component)**
  - Takes care of the apply request values phase.
  - Examines incoming request parameters to determine if and which button was pressed.
● Event handling

```java
public class .. implements ToolBarButtonPressedListener {
    ..
    toolbar.addToolBarButtonPressedListener(this);
    ..
}

..

public void buttonPressed (ToolBarButtonPressedEvent e) {
    log.info("toolbar button " + e.getButton().getAction() + " was pressed.");
}
```
Data binding

- Direct connection to your model beans possible
- JSP 2.0 based
- Supports Read(rvalue) and Write(lvalue)
- Method binding facilitates dynamic method invocation of arbitrary public methods of arbitrary objects.
State management

- Transparent to developer
- Different strategies possible
  - Client side
  - Server side
  - Different compression schemes ...
- Client side state management emulates the behaviour of desktop applications
- Performance penalty for client side state management but scales better.
Struts vs. (or with ?) JSF

- JSF has higher abstraction level
- Tool support
- Struts as foundation, JSF as extension (McClanahan)
  - Transitional
- Foundation vs. User-Interface Framework (Kito Mann)
- Struts -> open-source is key to its success
- JSF -> standard, but open-source implementation(s) remain important.
Smile

- Open source JSF implementation
- Focus on
  - JSP & Class based model
  - Component library
  - Designer application
  - Documentation
  - Support for UI testing/scripting.
- Overall goal: productivity
- Current version 0.3.2
  - Implements Proposed Final Draft of specification without JSP tags.
- 0.4 will include JSP support
If You Only Remember One Thing...

JSF raises the level of abstraction, in a standards based way.
Questions ?