Berner Fachhochschule, Technik und Informatik

Advanced Web Technology 5) Java Server Faces in Action

Dr. E. Benoist

Fall Semester 2010/2011

Table of Contents

Motivations

- **Sun Reference Implementation**
  - Used in the examples last week
  - Does contain only standard components
- **Apache MyFaces JSF implementation**
  - Does contain the same set of components
  - Contains also custom components
  - Is the most used JSF implementation
  - Is linked with other libraries: Tomahawk, Tobago, Trinidad
- **Other implementations**
  - Any vendor can develop his own implementation of JSF
  - Anybody can also develop his own set of new reusable components

Install Apache MyFaces

- **Download the MyFaces libraries**
  - `http://myfaces.apache.org/download.html`
- **Copy all the jar files into your `lib` directory**
  - `commons-beanutils.jar` Provides services for collections of beans
  - `commons-codec.jar` Provides implementation of common encoders and decoders such as Base64, Hexadecimal, Phonetic, and URL
  - `commons-collections.jar` The standard for collection handling in Java (`org.apache.commons.collections` packages)
  - `commons-digester.jar` The Rules-based processing of arbitrary XML documents
  - `commons-logging.jar` Providing a wrap-around for common Logging API's

Install Apache MyFaces (Cont.)

- `myfaces-api.jar` and `myfaces-impl.jar` the core MyFaces library
- `tomahawk.jar` contains other new components
- `commons-discovery.jar` used for locating classes that implement a given interface
- `commons-el.jar` the interpreter for the Expression Language defined in JSP 2.0
- `jstl.jar` JSP Standard Template Library

Update web.xml

- **Define Faces as the default servlet**
  - Create a jsf servlet
  - Map all *.jsf requests to this servlet
  - The servlet will use the corresponding jsp file.

```
<listener>
  <listener-class>
    org.apache.myfaces.webapp.StartupServletContextListener
  </listener-class>
</listener>
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup> 1 </load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```


Use JSF tags in a JSP file

- **Need to declare the JSF tag-libraries at the beginning of the file**
- **Encapsulate all the `<f:..>` and `<h:...>` tags inside a `<f:view>`**

```
<html>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<head><title>Hello World</title></head>
<body>

  <f:view>
    <h1><h:outputText value="Hello World" /></h1>
  </f:view>
</body>
</html>
```


Encapsulate texts of your application

- **No text should be written inside the application**
  - Text can be changed from outside
  - Text can be easily internationalized
- **Define a text file called :**
  `src/ch/bfh/ti/awt/exampleMyFaces/Messages.properties`

```
hello=hello
title=Hello Me
back=Back
send=Send
```

Encapsulate texts of your application (Cont.)

- **Declare the bundle in the JSP file**

```
<f:loadBundle basename="ch.bfh.ti.awt.exampleMyFaces.↘
→Messages"
var="messages" />
```

- **Use the value to display the content (inside the view)**

```
<f:view>
<body>
<h1><h:outputText value="#{messages.hello}" /></h1>
```

- **Content of the bundle is accessible using JSF EL (expression language) in any JSF tag (as `value` for instance).**

Backing Beans

- **Objects can be accessed in the JSF Expression Language**
  - They are called backing beans
  - Need to be registered in the faces-config.xml
  - "Properties" are automatically matched to values
- **Declaration in the faces-config.xml**

```
<managed-bean>
  <description>
    The "backing_file" bean that backs up the name
  </description>
  <managed-bean-name>myuser</managed-bean-name>
  <managed-bean-class>
        ch.bfh.ti.awt.exampleMyFaces.User
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```


Baking Bean (Cont.)

- **The previous declaration corresponds to the following class**
  - It is a bean containing one `String` property called `name`

```
package ch.bfh.ti.awt.exampleMyFaces;
public class User{
    private String name;
    public void setName(String n){
        name = n;
    }
    public String getName(){
        return name;
    }
}
```


Backing Bean (Cont.)

- **The property can be linked inside any JSP file using JSF Expression Language**
- Can be used inside a output text (like for "bundles")

```
<f:loadBundle basename="ch.bfh.ti.awt.exampleMyFaces.↘
→Messages"
var="messages" />
...
<h1><h:outputText value="#{messages.hello}" />
<h:outputText value="#{myuser.name}" /></h1>
```

- The property can be used inside a form (both geter and seter are used)

```
<h:form>
Type your name: <h:inputText value="#{myuser.name}" />
<h:commandButton id="submit" action="sendname" value="Send"↘
→/>
<h:commandLink value="Back" action="back" />
</h:form>
```

Components corresponding to html tags

- **Password (i.e. a field that one can not see)**

```
<h:form>
 <h:inputSecret value="#{user.pwd}" />
</h:form>
```

- **Image (corresponds to an img tag in HTML)**

```
<h:form><br>
 <h:graphicImage id="gi" alt="The image could not be found.↘
 →"
                 value="/image/rose.gif" width="250" height="↘
                 →250"
                 title="This is demo for 'graphicImage' tag" >
 </h:graphicImage>
</h:form>
```


The messages

- **A message can be associated with an input tag.**
  - Messages will not appear normally
  - During validation/conversion process, the component may receive some "messages".
  - They are displayed in the `<h:message>` tag when existing
  - Provide a way to display error messages not only by the object.

```
<h:inputText id="input_text"
                    value="#{MessageBean.a}"
                    required="true"/>
<h:message for="input_text" />
```


Navigation in a JSF application

- **HTML navigation is not JSF conform**
  - `<a href="hello.jsf">` a link
  - `<form action="myprog.jsf">` a form
  - Both remove the consistency of a JSF session.
  - Component tree, status, session, are lost using those links
- **Use JSF navigation instead**
  - `Command link`
  - Is rendered like a link, but must be put inside a form
  - Value is the text displayed inside the link (can be any EL expression)
  - Action corresponds to JSF navigation (can also be the return value of an EL expression)

```
<h:form>
<h:commandLink value="Hello World" action="hello">
</h:commandLink>
</h:form>
```


JSF Navigation

- **Actions can also be generated by Buttons**
  - `Command Button`

```
 <h:form>
Type your name: <h:inputText value="#{myuser.name}" ↘
→/>
<h:commandButton id="submit" action="hello" value="↘
→Send"/>
<h:commandLink value="Back" action="back" />
</h:form>
```

  - Must be included in a form
  - Is rendered with a Button
  - Text in value is displayed in the button
  - Action is sent to the navigation of JSF

Action returned by a method

- **The action of a command link or button can be an EL expression**
  - Can be a string
  - Or the value returned by a method
- **The JSP file:**

```
<h:form>
  <h:outputLabel for="userid">
     <h:outputText value="User ID" />
   </h:outputLabel>
   <h:inputText id="userId" value="#{login.userId}" /><br/>
   <h:outputLabel for="password">
        <h:outputText value="password" />
    </h:outputLabel>
    <h:inputSecret id="password" value="#{login.password}" />
    <br/>
    <h:commandButton action="#{login.login}" value="Login" />
</h:form>
```


Action returned by a method (Cont.)

- **The corresponding Java Bean**

```
public String login(){
    if((userId==null) || (userId.length()<1))
        return "failure";
    if ((password == null) || (password.length()<1))
        return "failure";
    User user = null;
    UserSecurityService service = new
    →UserSecurityService();
    user = service.login(userId,password);
    //user = new User("","");
    if(user == null)
        return "failure";
    return "success";
}
```


Navigation Rules

- **Navigation has to be declared in the `faces-config.xml`**
  - Define a navigation link:
  - A page to come from
  - An action (the value written in the action attributes)
  - A page to go to.

```
<navigation-rule>
  <description>
      The main page, dispatching requests over the different JSF
      →examples
  </description>
   <from-view-id>/menu.jsp</from-view-id>
   <navigation-case>
       <description>
         The Hello World application (simple Hello world in JSF)
       </description>
       <from-outcome>hello</from-outcome>
       <to-view-id>/hello.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
```


Navigation Rules (Cont.)

- **Per default, the navigation does display the name of origin file**
  - If form is `menu.php`
  - we click on a link
  - The URL remains `menu.jsf`
- **We can change it by defining the navigation case to be a "redirect"**

```
<navigation-case>
  <description>
    The Hello World application (simple Hello world in JSF)
  </description>
  <from-outcome>hello</from-outcome>
  <to-view-id>/hello.jsp</to-view-id>
  <redirect />
</navigation-case>
```

The component for displaying tables

- **The `DataTable` component**
  - Can display the content of any table
  - Takes a collection as input, and displays each of the elements
- **A table is composed of columns**
  - Elements in a component are displayed how they are placed (in the same order)
- **A column can contain a "header"**
  - This should not be placed anywhere
  - We use a facet
- **A facet represents a specific role in a component**
  - Facet is recognized using its name ("header" in our case)

A table (dataTable component)

```
<h:dataTable id="dataTable" styleClass="scrollerTable"
             var="module" value="#{modules.data}" rows="3"↘
             →>
<h:column>
        <f:facet name="header">
        <h:outputText value="#{messages.tableHeaderNb}" ↘
        →/>
        </f:facet>
        <h:outputText value="#{module.number}" />
</h:column>
<h:column>
        <f:facet name="header">
        <h:outputText value="#{messages.tableHeaderDesc}" ↘
        →/>
        </f:facet>
        <h:outputText value="#{module.name}" />
</h:column>
```


The Java Bean

```
package ch.bfh.ti.awt.exampleMyFaces;
import java.util.*;
public class DataBean{
    List data;
    public DataBean(){
        data = new LinkedList();
        // The following Data should be hold out of a Database
        data.add(new Module("2405","Marketing"));
        data.add(new Module("7083","Ergonomie, Psychologie"));
        ...
    }
    public List getData(){
        return data;
    }
    public void setData(List l){
        return;
    }
}
```


Event Handling

- **Navigation and Event Handling**
  - Navigation moves between pages
  - Even Handling is fired even if the page remains the same
- **Two types of events**
  - Value Changed Event
    When the value contained in a component is changed
    Typical for Input fields
  - Action Event
    A link or a button have been clicked
    There is no need to use an action and a navigation

Event Handling

- **For instance define links without action**

```
<h:form>
<h:commandLink id="fr" value="#{bundle.french}">
  <f:actionListener type="mypackage.LanguageChangedActionEvent" ↘
  →/>
</h:commandLink>
<h:commandLink id="de" value="#{bundle.german}">
  <f:actionListener type="mypackage.LanguageChangedActionEvent" ↘
  →/>
</h:commandLink>
```

- **Define an event handling**

```
public class LanguageChangedActionEvent implements ActionListener {
    public void processAction(ActionEvent e)
        throws AbortProcessingException{
        String language = e.getComponent().getId();
        FacesContext context = FacesContext.getCurrentInstance();
        context.getViewRoot().setLocale(new Locale(language));
    }
```


The Tomahawk Library

- **Supported by the Apache Fundation**
  - Part of MyFaces project (but not in the core API)
  - Contains useful components
- **Examples of MyFaces Tomahawk Components**
  - Calendar (inline or popup)
  - Data Scroller (for paging of tables)
  - jscookmenu (for creating a dynamic menu)
  - newspaperTable (for displaying Data on many columns, like a newspaper)
  - tabbedPane
  - tree2
  - ...

Install Tomahawk

- **Download Tomahawk**
  - From the apache web site.
- **Install `tomahawk.jar` in your library path**
  - Be sure it will be accordingly deployed
- **Insert the Tomahawk taglib in your JSP files**

```
<%@ taglib uri="http://myfaces.apache.org/tomahawk"
           prefix="t" %>
```

- **Install an Extension Filter in the `web.xml` file**

Install Tomahawk (Cont.)

Filters in the `web.xml`

```
<filter>
  <filter-name>extensionsFilter</filter-name>
  <filter-class>org.apache.myfaces.webapp.filter.↘
  →ExtensionsFilter
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>extensionsFilter</filter-name>
  <url-pattern>*.jsf</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>extensionsFilter</filter-name>
  <url-pattern>/faces/myFacesExtensionResource/*
  </url-pattern>
</filter-mapping>
```

Example: The DataScroller

- **Suppose we have a List of modules**
  - It is stored in a Java Bean
  - The list is displayed in a dataTable
- **We want to add paging functionality**
  - We need the DataScroller Tomahawk Component (`<t:dataScroller>`)
  - But we need to use tomahawk `dataTable` also

A Paged table

```
<h:form>
<t:dataTable id="dataTable"
styleClass="scrollerTable" var="module" value="#{modules.data}" rows="3">
<h:column>
        <f:facet name="header">
        <h:outputText value="#{messages.tableHeaderRowNumber}" />
        </f:facet>
        <h:outputText value="#{module.number}" />
</h:column>
<h:column>
        <f:facet name="header">
        <h:outputText value="#{messages.tableHeaderRowDescription}" />
        </f:facet>
        <h:outputText value="#{module.name}" />
</h:column>
</t:dataTable>
  <h:panelGrid columns="1" styleClass="scrollerTable2"
            columnClasses="standardTable_ColumnCentered">
    <t:dataScroller id="pagination" for="dataTable" fastStep="20" pageCountVar="pageCount"
                    pageIndexVar="pageIndex" styleClass="scroller" paginator="true" paginatorMaxPages="↘
                    →14"
                    paginatorTableClass="paginator" paginatorActiveColumnStyle="font-weight:bold;">
          <f:facet name="previous">
                    <t:outputText value="<" />
          </f:facet>
          <f:facet name="next">
                    <t:outputText value=">" />
          </f:facet>
          <f:facet name="fastforward">
                    <t:outputText value=">>" />
          </f:facet>
Scroller>
```


Conclusion

- **Building a JSF Application**
  - Create the Business Logic
  - Create the Data Beans
  - Define the pages and the connection to beans (using EL)
  - Define Navigation between the pages
- **JSF is powerful**
  - You can implement your own new components
  - You can import new libraries of components (like Tomahawk)

References

- **Book**
  - Wadia et al, The Definitive Guide to Apache MyFaces and Facelets, Apress 2008