

Advanced Web Technology

5) Facelets in JSF

Emmanuel Benoist
Fall Term 2016-17

Using Facelets

- Motivation
 - JSTL, a template language for JSF
- First Example : The Modules Directory
- JSP Standard Template Library: JSTL
 - The "if" Tag
- Reusable Composite Components
- Create a new Component

Motivation

JSTL, a template language for

JSTL, a template Language for JSF

- ▶ **Template for JSF**
 - ▶ Similar to Smarty for PHP
 - ▶ Used for the creation of the JSF Component tree
- ▶ **Include pages in pages**
 - ▶ Part of pages may be similar
 - ▶ Header / footers for instance
- ▶ **Loop inside a page**
 - ▶ For or while loops can visit a structure
- ▶ **Conditional Branching**
 - ▶ If / then /else

First Example : The Modules Directory

First Example : The Modules Directory

- ▶ **We need to create a simple directory of Modules**
- ▶ **We create various xhtml files**
 - ▶ `template.xhtml` for storing the main design for the site
 - ▶ `index.xhtml` the welcome file
 - ▶ `awt.xhtml` file presenting the Advanced Web Technology Module
 - ▶ `webProg.xhtml` file presenting the webProgramming
 - ▶ `menu.xhtml` contains just the menu. It is included in the other files

The template

- ▶ **We include the name spaces corresponding to jsf and facelets tags**
 - ▶ JSF tags are prefixed with `h` or `f`
 - ▶ Facelets specific tags are prefixed with `ui`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD_XHTML_1.0_
→ Transitional//EN"
" http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.
→ dtd" >
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:ui="http://java.sun.com/jsf/facelets" >
<head>
```

The template (Cont.)

- ▶ **We define the basic layout.**

- ▶ We insert CSS and Javascript in such a file

```
<title>Facelets–Test</title>
<style type="text/css" >
<!--
  .box {
    float: right;
    width: 50%;
    border: black dotted 1px;
    padding: 5px
  }
-->
</style>
</head>
<body>
```

The template (Cont.)

- ▶ **Template contains placeholders that can be overwritten**

- ▶ We use `ui:insert` to define areas that can be overwritten
- ▶ each `insert` receives a name and a default value (the content of the tag).

```
<h:form>
<h1>Facelets–Template</h1>
<div class="box" >
  <ui:insert name="navigation" />
</div>
<ui:insert name="content" >
  Default Text for content (only if no content has been defined)
</ui:insert>
</h:form>
</body>
</html>
```

The home page

- ▶ **The index.xhtml file greets a user**

- ▶ Must be a valid XHTML document, so includes all the XHTML "stuff"
- ▶ Everything that is outside the `ui:composition` tag is ignored.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
" http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" >
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:ui="http://java.sun.com/jsf/facelets" >
<body>
This text will never be displayed
(text before composition is ignored)
<ui:composition template="template.xhtml" >
```

The home page (Cont.)

- ▶ **We are using new facelets tags:**

- ▶ `ui:composition` used to reference the template for this page
- ▶ `ui:define` (its name must match the one of the `ui:insert` in the template).
- ▶ `ui:include` to include the content from another document

```
<ui:composition template="template.xhtml" >
  <ui:define name="navigation" >
    <ui:include src="menu.xhtml" />
  </ui:define>
</ui:composition>
This text will neither be used
(text after composition is ignored too)
</body>
</html>
```

A menu page

- ▶ This page is included in all the other pages
- ▶ It must contain the same “xhtml” stuff (like definitions of name spaces).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//>
EN"
...
<ui:composition>
<h3>Content table</h3>
<hr />
<h:panelGrid column="1">
<h:commandLink value="Home" action="index" />
<h:commandLink value="Web_Programming" action="webProg" />
<h:commandLink value="Advanced_Web_Technology" action="awt" />
</h:panelGrid>
</ui:composition>
This text will neither be used
(text after composition is ignored too)
</body>
</html>
```

Facelets TagLibs

Facelets support the following tag libraries

- ▶ **Templating library** The one we have already seen (with the `ui:...` tags.)
- ▶ **JSF libraries** core and html jsf libraries are supported by Facelets.
- ▶ **JSTL** facelets provides partial support for JSTL tags. Function library is fully supported, Core is only partially:
 - ▶ `c:if` for conditional branching
 - ▶ `c:forEach` for looping in a collection
 - ▶ `c:catch` For emulating a try catch
 - ▶ `c:set` For defining manually some attribute variables

JSP Standard Template Library JSTL

JSP Standard Template Library: JSTL

- ▶ **Introduces some programming in the view part**
 - ▶ Not used as a template language
 - ▶ JSTL is not fully supported

JSTL : Loops

▶ Looping with explicit numeric values

```
<c:forEach var="name" begin="x" end="y" step="z" >
  Blah, blah <h:outputText value="#{name}" />
</c:forEach>
```

▶ Looping over data structures

- ▶ Can loop down arrays, strings, collections, maps

```
<c:forEach var="name"
  items="array-or-collection" >
  Blah, blah <h:outputText value="#{name}" />
</c:forEach>
```



The "if" Tag

The "if" Tag

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:c="http://java.sun.com/jsp/jstl/core"
>
...
<ul>
<c:forEach var="i" begin="1" end="10" >
  <li><h:outputText value="#{i}" />
    <c:if test="#{i > 7}" >
      (greater than 7)
    </c:if></li>
</c:forEach>
</ul>
```

The "choose" Tag

```
<ul>
<c:forEach var="i" begin="1" end="10" >
  <li>#{i}
    <c:choose>
      <c:when test="#{i <= 4}" > (small)
      </c:when>
      <c:when test="#{i <= 8}" > (medium)
      </c:when>
      <c:otherwise> (large)
      </c:otherwise>
    </c:choose>
  </li>
</c:forEach>
</ul>
```

Reusable Composite Components

Reusable Composite Component

- ▶ **Facelets allows to create lightweight composition components**
Two basic steps
 - ▶ Create a tag source file, which will contain the XHTML code that defines your component
 - ▶ Register the tag in your custom tag library, so it can be used in your Facelets applications.
- ▶ **You can reuse this component as much as possible**

Example: Custom “inputTextLabeled” Component

- ▶ **We need a component that can be used like this:**

```
<custom:inputTextLabeled  
  label="Name"  
  value="#{module.name}" />
```

- ▶ **What it does:**
 - ▶ Is a combination of a label and an input text

Creating a tag source file

In the directory: `web/resources/components/mylib/` we write the following file:

input.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//\n\n->EN"  
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" >  
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.\n\n->com/jsf/html"  
  xmlns:f="http://java.sun.com/jsf/core" xmlns:composite="http://\n\n->java.sun.com/jsf/composite" >  
  <composite:interface>  
    <composite:attribute name="label" />  
    <composite:attribute name="value" />  
  </composite:interface>  
<composite:implementation>  
  <h:outputLabel value="#{cc.attrs.label}:" />  
  <h:inputText value="#{cc.attrs.value}" />  
  </h:outputLabel>  
</composite:implementation> </html>
```

Let's use the new component / tag

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//>
<EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.
com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core" xmlns:ui="http://java.sun.
com/jsf/facelets"
  xmlns:mylib="http://java.sun.com/jsf/composite/components/
mylib">
<head><title>head</title></head>
<body>
<f:view> <h:form>
  <h:panelGrid columns="1">
    <mylib:input label="Name" value="#{mod.name}"/>
    <mylib:input label="Nb" value="#{mod.number}"/>
    <h:commandButton value="Add module"
      actionListener="#{moduleDictionary.addModule}" />
  </h:panelGrid>
</h:form> </f:view> </body> </html>
```

Icon component

- ▶ We want to design a component icon on which one can click
- ▶ Example of use:

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:mylib="http://java.sun.com/jsf/composite/
components/mylib">

  <mylib:icon actionMethod="#{places.logout}"
    image="#{resource['images:back-
arrow.jpg']}" />

  ...
</html>
```

- ▶ Action method is executed when the icon is clicked on.

The icon component

The file `/web/resources/components/mylib/icon.xhtml`

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//>
<EN"
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:composite="http://java.sun.com/jsf/composite">
<composite:interface>
  <composite:attribute name="image" />
  <composite:attribute name="actionMethod"
    methodSignature="java.lang.String action()" />
</composite:interface>
<composite:implementation>
<h:form>
  <h:commandLink action="#{cc.attrs.actionMethod}" immediate="
true">
  <h:graphicImage value="#{cc.attrs.image}"
    styleClass="icon" />
  </h:commandLink>
</h:form> </composite:implementation></html>
```

Add an optional parameter

- ▶ We want to add an optional style parameter
- ▶ Example of use:

```
<util:icon actionMethod="#{places.logout}"
  image="#{resource['images:back-arrow-
.jpg']}"
  styleClass="customIconClass" />
```

- ▶ We need to add the line in the interface

```
<composite:attribute name="styleClass" default="icon"
  required="false" />
```

- ▶ So we can change the implementation to:

```
<h:graphicImage value="#{cc.attrs.image}"
  styleClass="#{cc.attrs.styleClass}" />
```

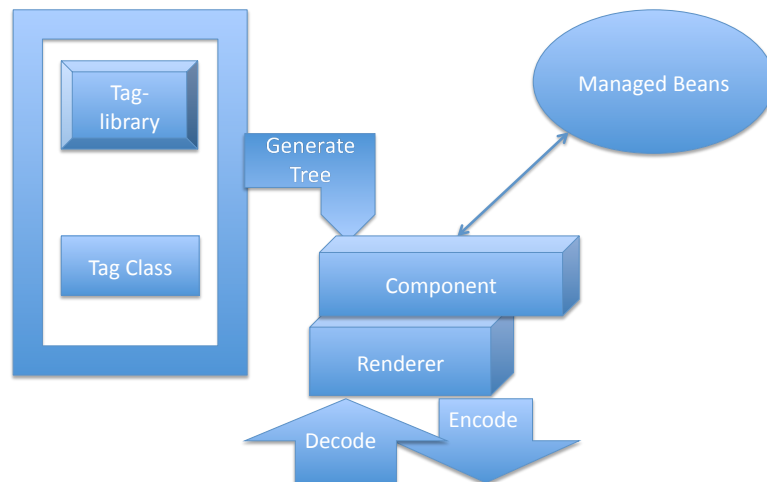
JSF Component Model

- ▶ **Much like Swing's component model**
 - ▶ It has events and properties
 - ▶ also has containers that contain components,
 - ▶ and that also are components that can be contained by other containers.
 - ▶ In theory, the JSF component model is divorced from HTML and JSP.
 - ▶ The standard set of components that ships with JSF has JSP bindings and generates HTML renderings.
- ▶ **Component functionality typically centers around two actions: decoding and encoding data.**
 - ▶ *Decoding* is the process of converting incoming request parameters to the values of the component.
 - ▶ *Encoding* is converting the current values of the component into the corresponding markup, that is, HTML.

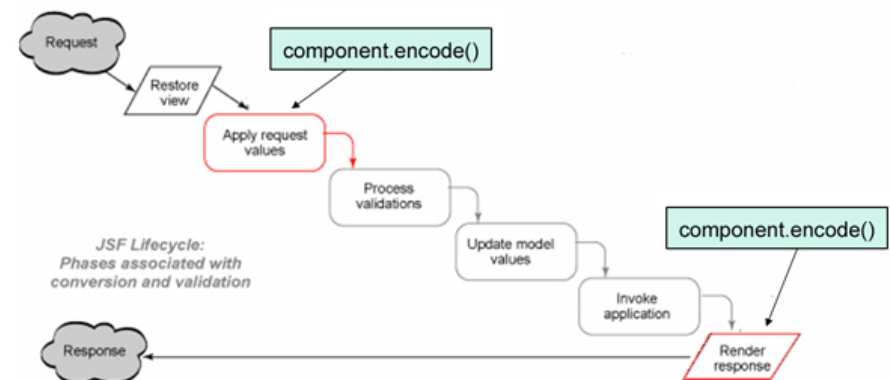
Component and Renderer

- ▶ **Two implementation approach**
 - ▶ *direct implementation*, the component itself implements decoding and encoding.
 - ▶ *delegated implementation*, the component delegates to a renderer that does the encoding and decoding.
 - ▶ delegated implementation: you can associate your component with different renderers that will represent it in different ways on the page; for example a multi-select list box versus a list of check boxes.
- ▶ **JSF components consist of two parts: the component and the renderer.**
 - ▶ The JSF Component class defines the state and behavior of a UI component;
 - ▶ a renderer defines how the component will be read from the request and how it will be displayed – usually through HTML.

Actors



JSF life-cycle and JSF components



Further component concepts

- ▶ **UIComponent is the base class for all JSF components.**
 - ▶ When you develop your own components you will subclass `UIComponentBase`,
 - ▶ it extends `UIComponent`,
 - ▶ default implementations for the abstract methods in `UIComponent`.
- ▶ **Components:**
 - ▶ have parents and identifiers.
 - ▶ They are associated with a component type, (for registering in `faces-config.xml`)
 - ▶ You can use the JSF-EL (expression language) to bind JSF components to managed bean properties.
- ▶ **A component can be a `ValueHolder` or an `EditableValueHolder`.**
- ▶ **Components like form field components have a `ValueExpression` that is evaluated as a EL expression.**

We create a new component

- ▶ **Define what we really want to have**
 - ▶ Design a prototyp of how your component should look like
 - ▶ Design a prototyp of how your component should be used
- ▶ **Define a Component and its Renderer**
 - ▶ Component has in charge the relation with the backing bean
 - ▶ The Renderer is specific for HTML (in our case)
- ▶ **Integrate the Component in a JSP Tag Lib**
 - ▶ Define a new Tag class
 - ▶ Define a new tag lib
- ▶ **Integrate everything in JSF**
 - ▶ Integrate your Component and Renderer in the `faces-config.xml`

Create a new Component

Conclusion

- ▶ **Facelets are the standard**
 - ▶ JSF 2.0 will only support facelets for its view part
 - ▶ Until JSF 1.2 JSP were the standard for writing pages
- ▶ **Facelets allow easy creation of components**
 - ▶ Custom composition components are much easier to write than with custom componets
- ▶ **Facelets allow to enrich view part of JSF**
 - ▶ You can use a part of JSTL (if / foreach / try catch)
 - ▶ You can use the Unified Expression Language

References

- ▶ **The definitive Guide to Apache MyFaces and Facelets, Z. Wadia, M. Marinschek, H. Saleh and D. Byrne, Apress, 2008**
- ▶ <http://www.ibm.com/developerworks/java/library/j-facelets/>
- ▶ <http://www.ibm.com/developerworks/java/library/j-jsf2fu2/index.html>