JavaServer Faces - Navigation

This tutorial explain the navigation handling in JSF and shows the usage with an small example application. We will show multiple and complex examples for navigation rules. For examples navigation outcomes depending on application logic, actions, static navigations and global navigations for the complete application.

Example:

```
<navigation-rule>
      <description>Rule of input.jsp</description>
      <from-view-id>/input.jsp</from-view-id>
      <navigation-case>
            <description>show success.jsp</description>
            <from-action>#{myBean.checkName}</from-action>
            <from-outcome>success</from-outcome>
            <to-view-id>/success.jsp</to-view-id>
      </navigation-case>

</navigation-rule>
```

General

Author:
Sascha Wolski
http://www.laliluna.de/tutorials.html – Tutorials for Struts, EJB, xdoclet and eclipse.

Date:
March, 8th 2005

Software:
Eclipse 3.x
MyEclipse 3.8.x
JavaServer Faces

PDF

http://www.laliluna.de/download/jsf-navigation-en.pdf

Sources

http://www.laliluna.de/download/JSF-navigation-source.zip

The navigation handler

The heart of the navigation system is the navigation handler, which is the actual code that decides which page to load next. The navigation handler operates in response to events, which are usually executed by action sources, like submit buttons or submit hyperlinks.

```
<h:commandButton action="success" />
<h:commandLink action="success" />
```

An action source returns an outcome value to the navigation handler and the navigation handler choose the navigation case that match the outcome value and display the page specified in the navigation case.

An action source can either associated with an action method on a backing bean. This method usually perform application logic and returns a hardcoded or logical outcome value. For example the command button below associates with the action method *checkName* of the backing bean *myBean.*

```
<h:commandLink action="#{myBean.checkName}" />
```

The method checkName can perform application logic, like getting data from a database, and then return a outcome value. Or the method and check something and choose a logical outcome based on the checks.

The navigation handler operates on a set of navigation rules, which defines all of the application's possible navigation paths. The Navigation rules are defined in the application configuration file with XML.

Some IDE's allow you to visually edit the navigation rules and some other plugins like JSF Console allow an easy edit of the navigation rules.

The XML tags of the application configuration file.

Here are an overview of the xml tags you can use to create navigation rules in your application configuration file (faces-config.xml)

```
<navigation-rules>
      <description>
      <display-name>
      <icon>
      <from-view-id>
      <navigation-case>
            <description>
            <display-name>
            <icon>
            <from-action>
            <from-outcome>
            <to-view-id>
            <redirect>
```

Navigation rules are defined with the *<navigation-rules>* element. All nested elements which are bold are required.

Optional you can specify the page to which it applies with the *<from-view-id>*. Each *<navigation-rules>* has one or more *<navigation-case>* elements, which must have a *<to-view-id>* element that specified the next page to load.

You can also set up rules for all pages, or groups of pages (like Struts global ActionForwards). A navigation rule is global if its *<form-view-id>* element is an asterisk (*) and is used for all pages. You have the same effect by omitting the *<form-view-id>* element. If you want to group pages with the same start characters like “confirm” you can add a asterisk to the end of the name like “confirm*”. All pages starts with “confirm” (like confirmOrder.jsp) will be grouped by this navigation case. (Asterisks aren't supported anywhere else in the string.) If there are several rules that starts with “confirm”, JSF will choose the longest one.

```
<navigation-rule>
      <from-view-id>*</from-view-id>
      <navigation-case>
            <from-outcome>login</from-outcome>
            <to-view-id>/login.jsp</to-view-id>
      </navigation-case>
      <navigation-case>
            <from-outcome>logout</from-outcome>
            <to-view-id>/logout.jsp</to-view-id>
      </navigation-case>
</navigation-rule>
```

A more common use is associating a rule with an entry directory:

```
<navigation-rule>
      <from-view-id>/help/*</from-view-id>
      <navigation-case>
            <from-outcome>start</from-outcome>
            <to-view-id>/help/start.jsp</to-view-id>
      </navigation-case>
      <navigation-case>
            <from-outcome>search</from-outcome>
            <to-view-id>/help/search.jsp</to-view-id>
      </navigation-case>
</navigation-rule>
```

This specifies a rule that applies to all pages within the context-relative directory.

Within the *<navigation-case>* element you have four different choices:

- Specify no *<from-action>* and no *<from-outcome>*. This navigation case will be used if the outcome value does not match any outcome value specified in the navigation cases.

```
  <navigation-case>
      <to-view-id>/confirm.jsp</to-view-id>
  </navigation-case>
```

- Specify an outcome with a *<from-outcome>* element. If the outcome value of the page equals one of the values specified by the *<from-outcome>* element, this navigation case will be used. For example if the outcome is success the navigation case below will be used.

```
  <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/confirm.jsp</to-view-id>
   </navigation-case>
```

- Specify a *<from-action>* element. The *<from-action>* element specifies an action method in a backing bean which returns a logical or hardcoded outcome value. For example the navigation case shown below is used if the method checkName of the backing bean myBean specified with the *<from-action>* element is executed.

```
  <navigation-case>
      <from-action>#{myBean.checkName}</from-action>
      <to-view-id>/confirm.jsp</to-view-id>
   </navigation-case>
```

- Specify both elements, *<from-outcome>* and *<from-action>.* The outcome value must be returned by an action method of a backing bean. If no the method do not return the value (if you use a *<h:commandButton action="success" />* the navigation case will be ignored.

```
  <navigation-case>
      <from-action>#{myBean.checkName}</from-action>
      <from-outcome>success</from-outcome>
      <to-view-id>/confirm.jsp</to-view-id>
   </navigation-case>
```

If the *<redirect>* element is specified within a *<navigation-case>* element, JSF send an HTTP redirect instead of forwarding to the view internally. Both the *<navigation-rule>* and the *<navigation-case>* elements can optionally have *<description>*, *<display-name>* and *<icon>* elements which are generally used for tools.

The example application

Let's start with a small application that illustrate the usage of the navigation handler.

Note: If you are new with JavaServer Faces have a look at the First Java Server Faces tutorial.

Create a new Java project *Navigation* and add the JSF capabilities.

Bean Class MyBean

Create a new java class *MyBean* in the package *de.laliluna.tutorial.navigation*.

Add a property *name* and provide a getter and setter method.

Add a method *checkName()* which checks the property name if it has zero length and return a outcome value as String.

The following source code shows the class MyBean:

```
/**
 * Bean Class MyBean
 */
public class MyBean {

      // properties
      private String name;

      // getter and setter methods
      public String getName() {
            return name;
      }
      public void setName(String name) {
            this.name = name;
      }

      // action methods
      public String checkName(){

            // check property name
            // if length is zero return failure
            // else return success
            if(name.length() == 0)
                  return "failure";
            else
                  return "success";

      }
}
```

The navigation rules

The example application contains a welcome page. After submitting the welcome page the user will be redirect to an input page, where he can input a name. If the user submit the name with a zero length a error page will be displayed, otherwise a success page will be displayed. On the error page the user can go back to the input page and correct their input. On the success page the named which is submit by the user will be displayed and he can go back to the welcome page.

Open the application configuration file *faces-config.xml* and add the following navigation rules within the *<faces-config>* element.

The first navigation rule is for the *welcome.jsp* and has one navigation case. If the outcome of the *welcome.jsp* is *next* the navigation handler forwards to the *input.jsp*.

```
<navigation-rule>
      <description>Rule of welcome.jsp</description>
      <from-view-id>/welcome.jsp</from-view-id>
      <navigation-case>
            <description>redirect to input.jsp</description>
            <from-outcome>next</from-outcome>
            <to-view-id>/input.jsp</to-view-id>
            <redirect/>
      </navigation-case>
</navigation-rule>
```

The next navigation rule is for the *input.jsp* and has two navigation cases. Both navigation cases requires that the outcome value of the *input.jsp* is returned by the action method, we have specified in the bean class above.

```
<navigation-rule>
      <description>Rule of input.jsp</description>
      <from-view-id>/input.jsp</from-view-id>
      <navigation-case>
            <description>show success.jsp</description>
            <from-action>#{myBean.checkName}</from-action>
            <from-outcome>success</from-outcome>
            <to-view-id>/success.jsp</to-view-id>
      </navigation-case>
      <navigation-case>
            <description>show failure.jsp</description>
            <from-action>#{myBean.checkName}</from-action>
            <from-outcome>failure</from-outcome>
            <to-view-id>/failure.jsp</to-view-id>
      </navigation-case>
</navigation-rule>
```

The navigation rule below is for the *success.jsp* and has one navigation case, which redirects the user to the *welcome.jsp*.

```
<navigation-rule>
      <description>Rule of success.jsp</description>
      <from-view-id>/success.jsp</from-view-id>
      <navigation-case>
            <description>redirect to welcome.jsp</description>
            <from-outcome>back</from-outcome>
            <to-view-id>/welcome.jsp</to-view-id>
            <redirect/>
      </navigation-case>
</navigation-rule>
```

The last navigation rule is for the *failure.jsp*. If the outcome value of the page is “back”, it redirects the user to the *input.jsp*, where he can retype the name.

```
<navigation-rule>
      <description>Rule of failure.jsp</description>
      <from-view-id>/failure.jsp</from-view-id>
      <navigation-case>
            <description>redirect to input.jsp</description>
            <from-outcome>back</from-outcome>
            <to-view-id>/input.jsp</to-view-id>
            <redirect/>
      </navigation-case>
</navigation-rule>
```

Add a *<managed-bean>* element for the bean class *MyBean*, we have created above. The nested element *<managed-bean-name>* specifies the name, which is used to access the bean properties and methods. The *<managed-bean-class>* element specifies the class which represents the bean. With the *<managed-bean-scope>* element you can specify in which scope the bean will be placed, in our case we use the request scope.

```
<managed-bean>
      <description>
            MyBean
      </description>
      <managed-bean-name>myBean</managed-bean-name>
      <managed-bean-class>de.laliluna.tutorial.navigation.MyBean</managed-bean-
class>
      <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Welcome page

Create a new JSP file *welcome.jsp* in the folder */WebRoot* of your project.

Add a welcome text and a *<h:commandButton>* element. The attribute *action* specifies the outcome for this page, in our case it is *next*.

The following code shows the file welcome.jsp:

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

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
      <title>Welcome Page</title>
</head>

<body>
      <f:view>
            <h:form id="bookList">
                  Welcome Page <br><br>

                  <h:commandButton
                        id="submit"
                        action="next"
                        value="Goto input page" />
            </h:form>
      </f:view>
</body>
</html>
```

Input page

Create a JSP file *input.jsp* in ther folder */WebRoot* of your project.

Add a *<h:inputText>* element which is associated with the property *name* of the bean and a *<h:commandButton>.* The attribute *action* of the *<h:commandButton>* element is linked to the method *checkName()* of the bean. The method checks the length of the property *name* and return *success,* if the length is greaten then zero or *failure,* if the length is zero.

The following code shows the content of the file *input.jsp*:

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

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
      <title>Welcome Page</title>
</head>

<body>
      <f:view>
```

```
            <h:form id="bookList">
                  <h:inputText value="#{myBean.name}" />

                  <h:commandButton
                        id="Add"
                        action="#{myBean.checkName}"
                        value="Check Name" />
            </h:form>
      </f:view>
</body>
</html>
```

Success page

Create a new JSP file *success.jsp* in the folder */WebRoot* of your project.

Add a success message and output the value of the property *name* with a *<h:outputText>* element. Define a *<h:commandButton>* element.The attribute *action* of the *<h:commandButton>* holds the outcome for the *success.jsp*, in our case *back*.

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

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
      <title>Welcome Page</title>
</head>

<body>
      <f:view>
            <h:form id="success">
                  Success page.     <br><br>

                  The Name is '<h:outputText value="#{myBean.name}" />' <br><br>

                  <h:commandButton
                        id="failure"
                        value="Back to welcome page"
                        action="back" />
            </h:form>
      </f:view>
</body>
</html>
```

Failure page

Create a new JSP file *failure.jsp* in the folder */WebRoot* of your project.

Add a failure message and a *<h:commandButton>*. The attribute *action* of the *<h:commandButton>* holds the outcome for the *failure.jsp*, in our case *back*.

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

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
      <title>Welcome Page</title>
</head>

<body>
```

```
      <f:view>
            <h:form id="failure">
                  Failure page.     <br><br>

                  <h:commandButton
                        id="failure"
                        value="Back to input page"
                        action="back" />
            </h:form>
      </f:view>
</body>
</html>
```

Thats all. Now you can test the application. We recommend a Jboss or Tomcat installation.

Call your project with the following link:

http://localhost:8080/Navigation/welcome.faces