



JSF: Controlling Page Navigation

Originals of Slides and Source Code for Examples:

<http://www.coreservlets.com/JSF-Tutorial/>

This somewhat old tutorial covers JSF 1, and is left online for those maintaining existing projects. All new projects should use JSF 2, which is both simpler and more powerful. See <http://www.coreservlets.com/JSF-Tutorial/jsf2/>.

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android. Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



For live training on JSF 1 or 2, please see courses at <http://courses.coreservlets.com/>.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
 - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
 - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
 - Courses developed and taught by coreservlets.com experts (edited by Marty)
 - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services
- Contact hall@coreservlets.com for details**

Topics in This Chapter

- **JSF flow of control**
- **The basic steps in using JSF**
- **Static navigation**
 - One result mapping
- **Dynamic navigation**
 - Multiple result mappings
- **Accessing the request and response objects**
- **Common JSF problems**

5

© 2012 Marty Hall

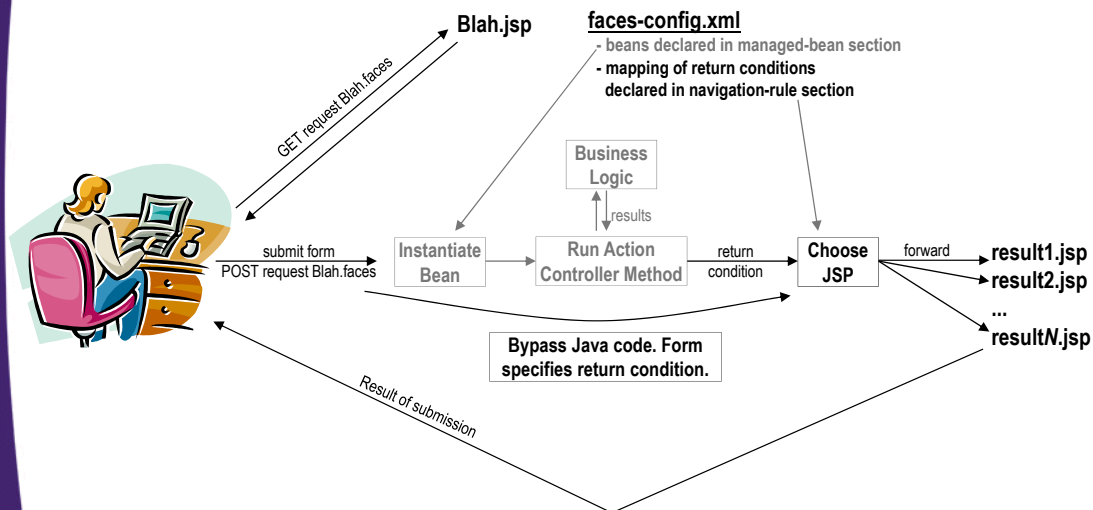


Static Navigation

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

JSF Flow of Control (Highly Simplified)



7

JSF Flow of Control (Simplified)

- **A form is displayed**
 - Form uses `f:view` and `h:form`
- **The form is submitted to itself**
 - Original URL and ACTION URL are `http://.../blah.faces`
- **A bean is instantiated**
 - Listed in the managed-bean section of `faces-config.xml`
- **The action controller method is invoked**
 - Listed in the action attribute of `h:commandButton`
- **The action method returns a condition**
 - A string that matches from-outcome in the navigation rules in `faces-config.xml`
 - In this example, static condition is specified in form
- **A results page is displayed**
 - The page is specified by to-view-id in the navigation rules in `faces-config.xml`

8

Steps in Using JSF

1) Create a bean

- A) Properties for form data
- B) Action controller method
- C) Placeholders for results data

2) Create an input form

- A) Input fields refer to bean properties
- B) Button specifies return condition
(or action controller method that will return condition)

3) Edit faces-config.xml

- A) Declare the bean
- B) Specify navigation rules

4) Create results pages

- Output form data and results data with h:outputText

5) Prevent direct access to JSP pages

- Use a filter that redirects blah.jsp to blah.faces

9

Example: Registration

- **Started by copying jsf-blank-myfaces**
 - Renamed it to jsf-navigation
 - Edited .settings/...component file as in previous lecture
- **Original URL**
 - `http://hostname/jsf-navigation/register.faces`
- **When form submitted**
 - A static page (WEB-INF/results/result.jsp) is displayed
- **Static result**
 - No business logic, beans, or Java code of any sort
- **Main points**
 - Format of original form
 - Use of navigation-rule in faces-config.xml

10

Main Points of This Example

- **Input form has following format:**

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
  HTML markup
  <h:form>
    HTML markup and h:blah tags
  </h:form>
  HTML markup
</f:view>
```

- **faces-config.xml specifies navigation rules:**

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC ...>
<faces-config>
  <navigation-rule>
    <from-view-id>/blah.jsp</from-view-id>
    <navigation-case>
      <from-outcome>some string</from-outcome>
      <to-view-id>/WEB-INF/results/something.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```

11

Step 1: Create a Bean

- **Postponed until next lecture**

- This example ignores form data
- Button in form directly specifies return condition
 - Rather than specifying an action controller method in the bean that will calculate the return condition

12

Step 2: Create Input Form

- **Basic format**

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

- **Invoking page**

- Actual file is *blah.jsp*
- URL is *blah.faces*

13

Step 2: Create Input Form

- **The h:form element**

- ACTION is automatically self (current URL)
- METHOD is automatically POST

- **Elements inside h:form**

- Use special tags to represent input elements
 - **h:inputText** corresponds to `<INPUT TYPE="TEXT">`
 - **h:inputSecret** corresponds to `<INPUT TYPE="PASSWORD">`
 - **h:commandButton** corresponds to `<INPUT TYPE="SUBMIT">`
- In later sections, we will see that input elements will be associated with bean properties
- For static navigation, specify simple string as action of `h:commandButton`
 - String must match navigation rule from `faces-config.xml`

- **More info on h:blah elements**

- <http://java.sun.com/j2ee/javaserverfaces/1.1/docs/tlddocs/>

14

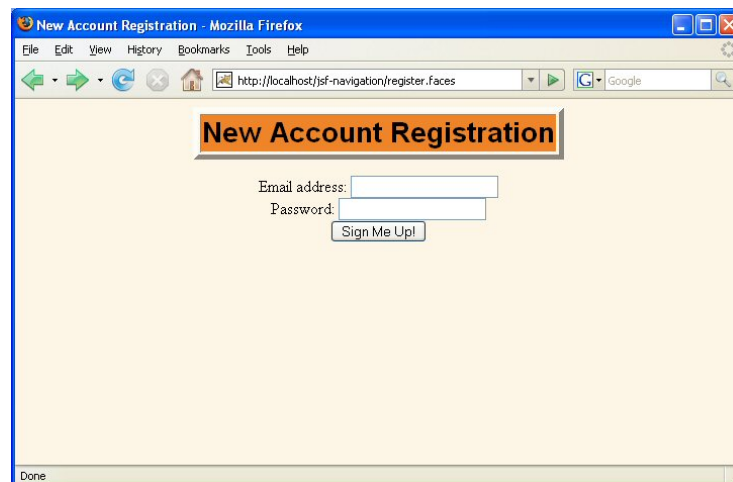
Step 2: Example Code (register.jsp)

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
  <TR><TH CLASS="TITLE">New Account Registration</TH></TR>
</TABLE>
<P>
<h:form>
  Email address: <h:inputText/><BR>
  Password: <h:inputSecret/><BR>
  <h:commandButton value="Sign Me Up!" action="register"/>
</h:form>
</CENTER></BODY></HTML>
</f:view>
```

15

Step 2: Result

- File is .../WebContent/register.jsp
- URL is http://localhost/jsf-navigation/register.faces



16

Step 3: Edit faces-config.xml

- **General format**

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC ...>
<faces-config>
...
</faces-config>
```

- **Specifying the navigation rules**

```
...
<faces-config>
  <navigation-rule>
    <from-view-id>/the-input-form.jsp</from-view-id>
    <navigation-case>
      <from-outcome>string-from-action</from-outcome>
      <to-view-id>/WEB-INF/.../something.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```

17

Step 3: Example Code

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC ...>
<faces-config>
  <navigation-rule>
    <from-view-id>/register.jsp</from-view-id>
    <navigation-case>
      <from-outcome>register</from-outcome>
      <to-view-id>/WEB-INF/results/result.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```

18

Step 4: Create Results Pages

- **RequestDispatcher.forward** used
 - So page can/should be in WEB-INF
- **Example code:**
 - .../WEB-INF/results/result.jsp

```
<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
  <TR><TH CLASS="TITLE">Success</TH></TR>
</TABLE>
<H2>You have registered successfully.</H2>
</CENTER>
</BODY></HTML>
```

19

Step 4: Example Result

- **Note that URL is unchanged**



20

Step 5: Prevent Direct Access to JSP Pages

- **Filename/URL correspondence**
 - Actual files are of the form *blah.jsp*
 - URLs used are of the form *blah.faces*
 - You must prevent clients from directly accessing JSP pages
 - Since they would give erroneous results
- **Strategies**
 - You cannot put input-form JSP pages in WEB-INF
 - Because URL must correspond directly to file location
 - So, use filter in web.xml. But:
 - You have to know the extension (.faces)
 - Assumes no non-JSF .jsp pages
- **This is a major drawback to JSF design**

21

Direct Access to JSP Pages

```
Apache Tomcat/5.5.17 - Error report - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/jsf-test/register.jsp
HTTP Status 500 -
Exception report
message:
description: The server encountered an internal error () that prevented it from fulfilling the request.
exception:
org.apache.jasper.JasperException: Exception in JSP: /register1.jsp:3
1: <@ taglib uri="http://java.sun.com/jsf/core" prefix="f" >
2: <@ taglib uri="http://java.sun.com/jsf/html" prefix="h" >
3: <f:view>
4: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
5: <HTML>
6: <HEAD><TITLE>New Account Registration</TITLE>
Stacktrace:
org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:504)
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:393)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:314)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:264)
javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
root cause:
javax.faces.FacesException: Faces context not found. getResponseWriter will fail. Check if the FacesServlet has been initialized
javax.faces.webapp.UIComponentTag.setupResponseWriter(UIComponentTag.java:926)
javax.faces.webapp.UIComponentTag.doStartTag(UIComponentTag.java:313)
org.apache.myfaces.taglib.core.ViewTag.doStartTag(ViewTag.java:73)
org.apache.jsp.register1_jsp._jsp_meth_f_View_0(register1_jsp.java:88)
org.apache.jsp.register1_jsp._jspService(register1_jsp.java:66)
org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:97)
javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:332)
org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:314)
org.apache.jasper.servlet.JspServlet.service(JspServlet.java:264)
javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
note: The full stack trace of the root cause is available in the Apache Tomcat/5.5.17 logs.
Apache Tomcat/5.5.17
Done
```

22

Preventing Direct Access: FacesRedirectFilter

```
public class FacesRedirectFilter implements Filter {
    private final static String EXTENSION = "faces";

    public void doFilter(ServletRequest req,
                        ServletResponse res,
                        FilterChain chain)
        throws ServletException, IOException {
        HttpServletRequest request = (HttpServletRequest)req;
        HttpServletResponse response = (HttpServletResponse)res;
        String uri = request.getRequestURI();
        if (uri.endsWith(".jsp")) {
            int length = uri.length();
            String newAddress =
                uri.substring(0, length-3) + EXTENSION;
            response.sendRedirect(newAddress);
        } else { // Address ended in "/"
            response.sendRedirect("index.faces");
        }
    }
    ...
}
```

23

Preventing Direct Access: web.xml

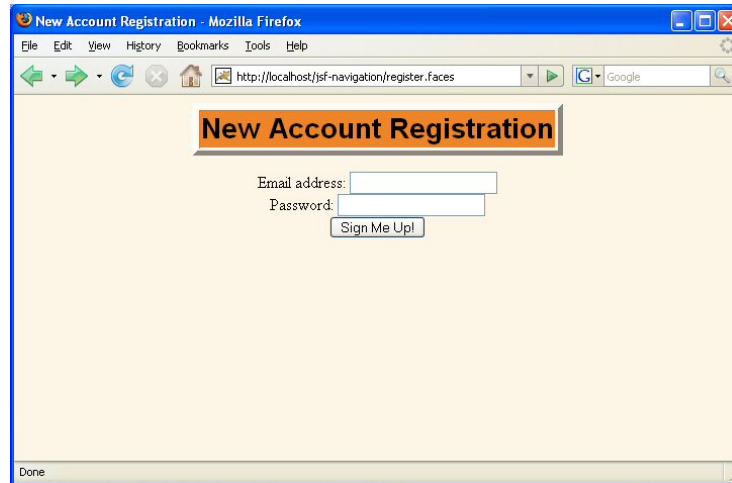
```
...
<filter>
    <filter-name>faces-redirect-filter</filter-name>
    <filter-class>
        coreservlets.FacesRedirectFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>faces-redirect-filter</filter-name>
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

24

Preventing Direct Access: Result

- **Either URL**

- `http://localhost/jsf-navigation/register.faces`
- `http://localhost/jsf-navigation/register.jsp`



25

© 2012 Marty Hall

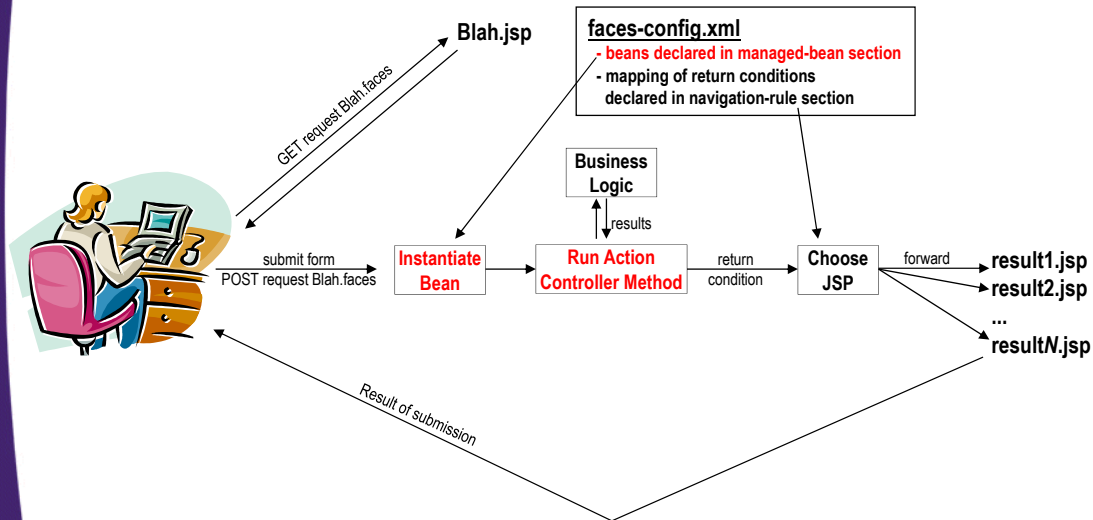


Dynamic Navigation

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

JSF Flow of Control (Simplified)



27

JSF Flow of Control (Simplified)

- **A form is displayed**
 - Form uses f:view and h:form
- **The form is submitted to itself**
 - Original URL and ACTION URL are http://.../blah.faces
- **A bean is instantiated**
 - Listed in the managed-bean section of faces-config.xml
- **The action controller method is invoked**
 - Listed in the action attribute of h:commandButton
- **The action method returns a condition**
 - A string that matches from-outcome in the navigation rules in faces-config.xml
- **A results page is displayed**
 - The page is specified by to-view-id in the navigation rules in faces-config.xml

28

Steps in Using JSF

1) Create a bean

- A) Properties for form data
- B) Action controller method
- C) Placeholders for results data

2) Create an input form

- A) Input fields refer to bean properties
- B) Button specifies action controller method that will return condition

3) Edit faces-config.xml

- A) Declare the bean
- B) Specify navigation rules

4) Create results pages

- Output form data and results data with h:outputText

5) Prevent direct access to JSP pages

- Use a filter that redirects blah.jsp to blah.faces

29

Example: Health Plan Signup

• Original URL

- `http://hostname/jsf-navigation/signup.faces`
 - Collects info to see if user qualifies for health plan

• When form submitted, one of two possible results will be displayed

- User is accepted into health plan
- User is rejected from health plan

• Main points

- Specifying an action controller in the form
- Creating an action controller method in the bean
- Using faces-config.xml to
 - Declare bean
 - Map return conditions to output pages

30

Main Points of This Example

- Specify the controller with `#{beanName.methodName}`

```
<h:commandButton
  value="Sign Me Up!"
  action="#{healthPlanController.signup}"/>
```
- Controller method returns strings corresponding to conditions
 - If null is returned, the form is redisplayed
 - Unlike with Struts, the controller need not extend a special class
- Use `faces-config.xml` to declare the controller as follows

```
<faces-config>
  <managed-bean>
    <managed-bean-name>controller name</managed-bean-name>
    <managed-bean-class>controller class</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
</faces-config>
```

Scope could also be session or application
- Add multiple navigation-rule entries to `faces-config.xml`
 - One for each possible string returned by the controller
 - If no string matches, the form is redisplayed

31

Step 1: Create a Bean

(A) Properties for form data

- Postponed until next lecture

(B) Action controller method

```
public class HealthPlanController {
  public String signup() {
    if (Math.random() < 0.2) {
      return("accepted");
    } else {
      return("rejected");
    }
  }
}
```

(C) Placeholders for results data

- Postponed until next lecture

32

Step 2: Create Input Form

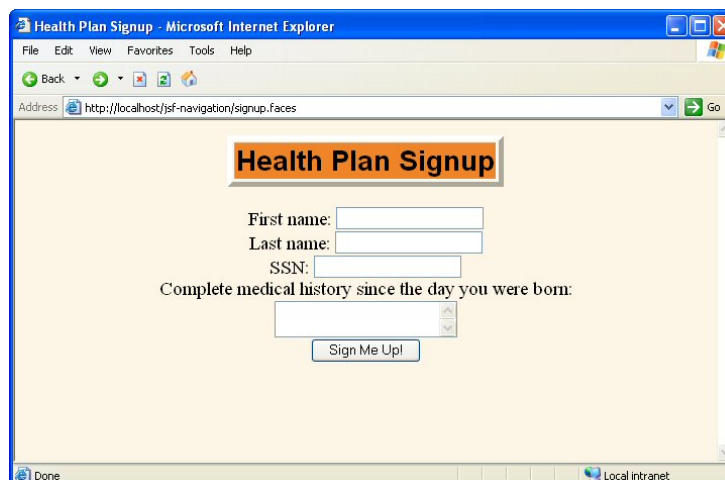
- **Same general syntax as in previous example**
 - Except for action of commandButton

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
...
<h:form>
    First name: <h:inputText/><BR>
    Last name: <h:inputText/><BR>
    ...
    <h:commandButton
        value="Sign Me Up!"
        action="#{healthPlanController.signup}"/>
</h:form>...
</f:view>
```

33

Step 2: Result

- **File is .../WebContent/signup.jsp**
- **URL is http://localhost/jsf-navigation/signup.faces**



34

Step 3: Edit faces-config.xml

(A) Declaring the bean

```
...
<faces-config>
  <managed-bean>
    <managed-bean-name>
      healthPlanController
    </managed-bean-name>
    <managed-bean-class>
      coreservlets.HealthPlanController
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
  ...
</faces-config>
```

Use request scope unless you have a specific reason to use session scope or (rarely) application scope

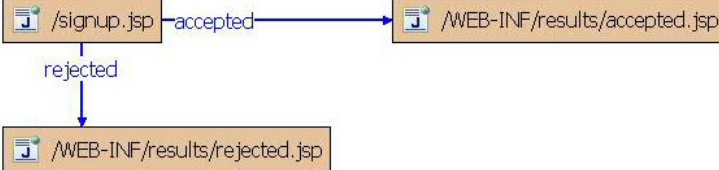
35

Step 3: Edit faces-config.xml

(B) Specifying navigation rules

- Outcomes should match return values of controller

```
<faces-config>
... (bean definitions
   from previous page)
<navigation-rule>
  <from-view-id>/signup.jsp</from-view-id>
  <navigation-case>
    <from-outcome>accepted</from-outcome>
    <to-view-id>/WEB-INF/results/accepted.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>rejected</from-outcome>
    <to-view-id>/WEB-INF/results/rejected.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
</faces-config>
```



The diagram illustrates the navigation rules defined in the XML. It shows a box for the source view `/signup.jsp` with two outgoing arrows. One arrow, labeled `accepted`, points to a box for the target view `/WEB-INF/results/accepted.jsp`. The other arrow, labeled `rejected`, points to a box for the target view `/WEB-INF/results/rejected.jsp`.

36

Step 4: Create Results Pages

- `.../WEB-INF/results/accepted.jsp`

```
<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
  <TR><TH CLASS="TITLE">Accepted!</TH></TR>
</TABLE>
<H2>You are accepted into our health plan.</H2>
Congratulations.
</CENTER>
</BODY></HTML>
```

37

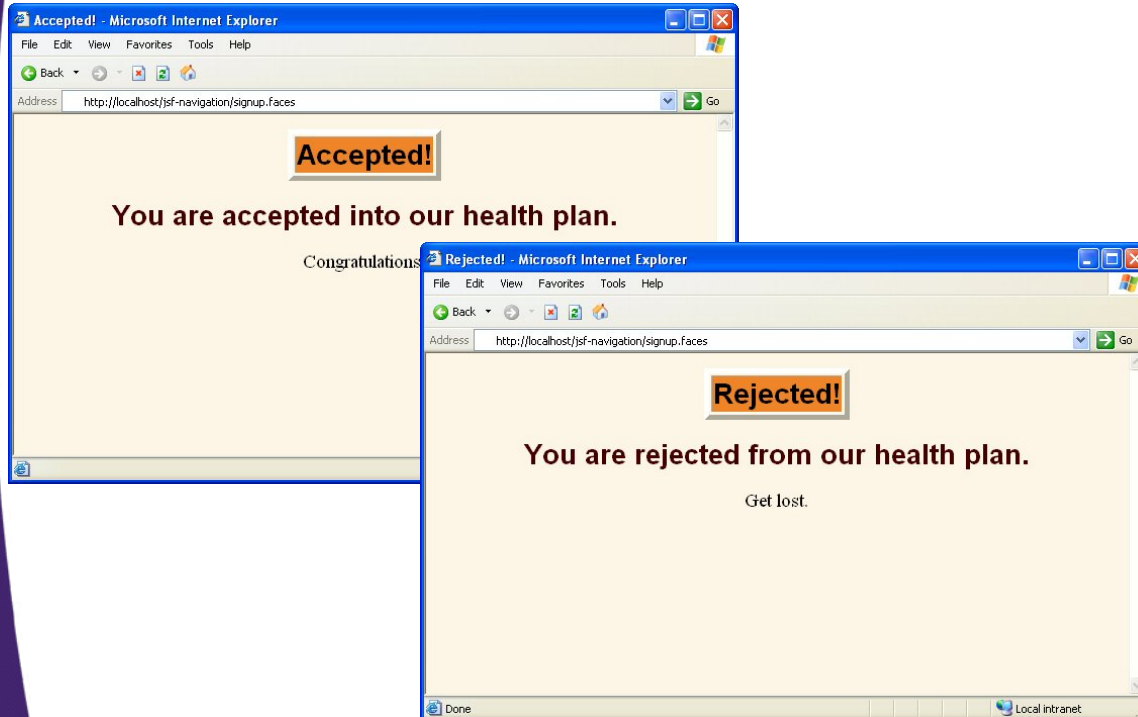
Step 4: Create Results Pages (Continued)

- `.../WEB-INF/results/rejected.jsp`

```
<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=5>
  <TR><TH CLASS="TITLE">Rejected!</TH></TR>
</TABLE>
<H2>You are rejected from our health plan.</H2>
Get lost.
</CENTER>
</BODY></HTML>
```

38

Step 4: Results



39

Step 5: Prevent Direct Access to JSP Pages

- **Use filter that captures url-pattern *.jsp**
 - No changes from previous example

40



Notes and Additional Capabilities

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Summary

- **Wildcards in navigation rule**
 - * for from-view-id matches any starting page
 - Omitting from-outcome results in all values matching
- **from-action in addition to from-outcome**
 - For when different buttons invoke different methods and methods have same values mapped differently. Overused.
- **Getting the request and response objects**

```
ExternalContext context =  
    FacesContext.getCurrentInstance().getExternalContext();  
HttpServletRequest request =  
    (HttpServletRequest) context.getRequest();  
HttpServletResponse response =  
    (HttpServletResponse) context.getResponse();
```
- **Interleaving managed-bean & navigation-rule**
 - It is legal to alternate back and forth

Wildcards in Navigation Rules

- *** for from-view-id matches any starting page**
 - Used when multiple different pages map same return value to same result page

- **Example**

```
<navigation-rule>
  <from-view-id>*</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/WEB-INF/results/success.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

43

Without Wildcards

```
<navigation-rule>
  <from-view-id>/page1.jsp</from-view-id>
  <navigation-case>
    <from-outcome>condition1</from-outcome>
    <to-view-id>/WEB-INF/results/result1.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>unknown-user</from-outcome>
    <to-view-id>/WEB-INF/results/unknown.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/page2.jsp</from-view-id>
  <navigation-case>
    <from-outcome>condition2</from-outcome>
    <to-view-id>/WEB-INF/results/result2.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>unknown-user</from-outcome>
    <to-view-id>/WEB-INF/results/unknown.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

44

With Wildcards

```
<navigation-rule>
  <from-view-id>*</from-view-id>
  <navigation-case>
    <from-outcome>unknown-user</from-outcome>
    <to-view-id>/WEB-INF/results/unknown.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/page1.jsp</from-view-id>
  <navigation-case>
    <from-outcome>condition1</from-outcome>
    <to-view-id>/WEB-INF/results/result1.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/page2.jsp</from-view-id>
  <navigation-case>
    <from-outcome>condition2</from-outcome>
    <to-view-id>/WEB-INF/results/result2.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

45

Wildcard Matching Return Conditions

- **Omitting from-outcome means all other return conditions match**
 - Except for null, which always means redisplay form

- **Example**

```
<navigation-rule>
  <from-view-id>/some-page.jsp</from-view-id>
  <navigation-case>
    <from-outcome>condition1</from-outcome>
    <to-view-id>/WEB-INF/results/result1.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <to-view-id>/WEB-INF/results/default.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

46

Explicit from-outcome

```
<navigation-rule>
  <from-view-id>/page1.jsp</from-view-id>
  <navigation-case>
    <from-outcome>condition1</from-outcome>
    <to-view-id>/WEB-INF/results/result1.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>condition2</from-outcome>
    <to-view-id>/WEB-INF/results/result2.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>condition3</from-outcome>
    <to-view-id>/WEB-INF/results/result2.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>condition4</from-outcome>
    <to-view-id>/WEB-INF/results/result2.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

47

Default (Omitted) from-outcome

```
<navigation-rule>
  <from-view-id>/page1.jsp</from-view-id>
  <navigation-case>
    <from-outcome>condition1</from-outcome>
    <to-view-id>/WEB-INF/results/result1.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <to-view-id>/WEB-INF/results/result2.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

48

from-action

- **Designates the method that you came from**
 - Suppose you had two buttons that invoked two different methods, and both returned "error". But you want the two "error" values to have different results pages.

```
<navigation-rule>
  <from-view-id>/somepage.jsp</from-view-id>
  <navigation-case>
    <from-action>#{beanName.method1}</from-action>
    <from-outcome>error</from-outcome>
    <to-view-id>/WEB-INF/results/err1.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-action>#{beanName.method2}</from-action>
    <from-outcome>error</from-outcome>
    <to-view-id>/WEB-INF/results/err2.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```
 - Easier to avoid using the same names for different things
 - Rarely needed in real life, but some books (*JSF The Complete Reference*) use it needlessly.

49

Getting the Request and Response Objects

- **JSF controller methods do not have direct access to the request and response**
 - Unlike in Struts, where controller method (execute) gets request and response automatically
 - If they are needed, use static method calls to get them

```
ExternalContext context =
  FacesContext.getCurrentInstance().getExternalContext();
HttpServletRequest request =
  (HttpServletRequest)context.getRequest();
HttpServletResponse response =
  (HttpServletResponse)context.getResponse();
```
 - In some environments, you cast results of `getRequest` and `getResponse` to values other than `HttpServletRequest` and `HttpServletResponse`
 - E.g., in a portlet environment, you might cast result to `PortletRequest` and `PortletResponse`

50

Getting the Request and Response Objects

- **Purpose**

- Useful for many request properties
 - Explicit session manipulation (e.g., changing inactive interval or invalidating session)
 - Explicit cookie manipulation (e.g., long-lived cookies)
 - Reading request headers (e.g., User-Agent)
 - Looking up requesting host name
 - Not needed to get request parameters
 - Bean populated automatically as in next lecture
- Useful for a few response properties
 - Setting status codes
 - Setting response headers
 - Setting long-lived cookies

51

Interleaving managed-bean and navigation-rule

- **If you have several different addresses in your app, it is OK to alternate**

```
<managed-bean>
  Stuff for bean1
</managed-bean>
<navigation-rule>
  Rules for address that uses bean1
</navigation-rule>
<managed-bean>
  Stuff for bean2
</managed-bean>
<navigation-rule>
  Rules for address that uses bean2
</navigation-rule>
```

- Of course, it is also OK to put all bean defs at the top, followed by all navigation rules.
 - Whichever organization you find easier to manage

52



Common Problems

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Pressing Button and Nothing Happens

- **In JSF, many error conditions simply result in the system redisplaying the form**
 - No error messages or warnings
 - Very confusing to beginning developers
- **Debugging strategies for these situations**
 - Many of the errors cause the process to abort at certain points. Knowing how far things got is very helpful.
 - Use print statements or IDE breakpoints
 - Put a print statement in the controller method
 - Put a print statement in the empty constructor
 - `public MyBean() { System.out.println("MyBean built"); }`
 - Bean should be instantiated twice for request scope
 - Put print statements in the bean setter methods

Pressing Button and Nothing Happens: Common Cases

1. Return value of controller method does not match from-outcome of navigation-case

- Remember values are case sensitive

2. Using from-action instead of from-outcome

```
<navigation-case>  
  <from-action>accepted</from-action>  
  <to-view-id>/WEB-INF/results/accepted.jsp</to-view-id>  
</navigation-case>
```

Should be from-outcome, not from-action

- This is really a special case of (1), since there is now *no* from-outcome
- This situation occurs frequently with Eclipse users that don't look carefully at the choices Eclipse offers in popup menu for the navigation-case entries.

55

Pressing Button and Nothing Happens: Common Cases

3. Forgetting # in action of h:commandButton

```
<h:commandButton  
  value="Button Label"  
  action="{beanName.methodName}"/>
```

Should have # here

- This is really a special case of (1), since `action="beanName.methodName"` means the literal String "beanName.methodName" is the from-outcome
 - In this situation and several others, it is very helpful to put a print statement in controller method to see if/when it is invoked

4. Typo in from-view-id

- This is a special case of (1), since the from-outcome applies to nonexistent page

56

Pressing Button and Nothing Happens: Common Cases

5. Controller method returns null

- This is often done on purpose to redisplay the form, but can be done accidentally as well.

6. Type conversion error

- You declare field to be of type int, but value is not an integer when you submit.
 - Behavior of redisplaying form is useful here. See validation section.

7. Missing setter method

- You associate textfield with bean property foo, but there is no setFoo method in your bean.
 - Debugging hint: You will see printout for bean being instantiated, but not for controller method

8. Missing h:form

- If you use h:inputText with no surrounding h:form, textfields will still appear but nothing will happen when you press submit button

57

Summary

• Basic steps to using JSF

- Create a bean
 - For now, only contains controller method
- Create an input form
 - The action of h:commandButton refers to controller method
- Edit faces-config.xml
 - Declare bean
 - Define navigation rules
- Create results pages
- Prevent direct access to JSP pages

• Static navigation

- Specify literal outcome as action of button
 - Outcome mapped by faces-config.xml to output page

• Dynamic navigation

- Specify method as action of button
- Method returns outcomes
 - Outcomes mapped by faces-config.xml to output pages

58



Questions?

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.