

Web Services And Java Server Faces

Student Guide

Version 1.0

presented by

Hands On Technology Transfer, Inc.
Chelmsford, Massachusetts (USA)

Copyright ©2002-2005 Hands On Technology Transfer, Inc. All Rights Reserved.

This student guide is copyrighted. This document may not, in whole or in part, be copied, duplicated, reproduced, translated, electronically stored, or reduced to machine readable form without prior written consent from Hands On Technology Transfer, Inc. Training programs and additional copies of these course materials are available through Hands On Technology Transfer, Inc. For information, contact HOTT in Chelmsford, Massachusetts, at (800) 413-0939.

Although the material contained herein has been carefully reviewed, HOTT does not warrant it to be free of errors or omissions. HOTT reserves the right to make corrections, updates, revisions or changes to the information contained herein. HOTT does not warrant the material described herein to be free of patent infringement.

Table of Contents

Module 1: Web Services Under Java (w/o EJBs)	1-1
Module Goals and Objectives	1-2
Section 1-1: Overview	1-3
The Basic Approaches	1-4
Section 1-2: The Low-Level Approach	1-5
Client Code	1-6
Examination of Client	1-8
Server Code	1-9
Examination of Server	1-10
Module Summary	1-11
Module 2: Java Server Faces	2-1
Module Goals and Objectives	2-2
Section 2-1: Java Server Faces Overview	2-3
What is JSF?	2-4
Controller-centric Focus of Struts	2-5
Different Focus for JSF	2-6
Whither Struts?	2-7
Section 2-2: Writing a JSF Application	2-9
Components For a Simple Web App	2-10
faces-config.xml	2-11
web.xml	2-13
A JSP Page	2-14
A Form Bean	2-15
Module Summary	2-16

Web Services And Java Server Faces

Course Introduction

Course Goals and Objectives

Major Course Goals

Get more out of your HOTT training

Specific Course Objectives

Upon completion of this course, students will be able to:

Understand JSF and Web Services

Module 1

Web Services Under Java (w/o EJBs)

Module Goals and Objectives

Major Module Goals

Understand how to use web services in the Java framework

Specific Module Objectives

Upon completion of this module, students will be able to:

Write simple web services in Java

Section 1-1

Overview

The Basic Approaches

- In the absence of EJBs, there are two basic effective approaches to writing web service clients and servers in Java:

1. Low-level approach

- Relies on *JAXM* (Java API for XML Messaging)
- A bit more high-level than DOM, but not much

2. Framework approach

- Can wrap web services around Java classes, as well as generate Java stubs from WSDL files
- Typified by the likes of Apache Axis and Systinet Wasp

Section 1-2

The Low-Level Approach

Client Code

- Here is an example SOAP client we'll pick apart in due course

Example 1-1

```
1 /*
2  * JAXM Client using a low-level binding
3  */
4
5 import java.io.*;
6
7 import javax.xml.soap.*;
8 import javax.xml.messaging.*;
9 import java.net.URL;
10
11 import javax.mail.internet.*;
12
13 import javax.xml.transform.*;
14 import javax.xml.transform.stream.*;
15
16 import org.dom4j.*;
17
18 public class SimpleSOAPClient {
19
20     static final String RECEIVER_URI =
21         "http://localhost:8080/jaxm-simple/receiver";
22
23     public static void main(String args[]) {
24
25         try {
26             URLEndpoint endpoint = null;
27             endpoint = new URLEndpoint(RECEIVER_URI);
28
29             SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
30             SOAPConnection connection = scf.createConnection();
31
32             MessageFactory mf = MessageFactory.newInstance();
33
34             // Create a message from the message factory.
35             SOAPMessage msg = mf.createMessage();
36
37             SOAPPart soapPart = msg.getSOAPPart();
38             SOAPEnvelope envelope = soapPart.getEnvelope();
39
40             // create a message body
41             SOAPBody body = envelope.getBody();
42
43             body.addChildElement(
44                 envelope.createName("GetReply", "jaxm",
45                 "http://sun.com/jaxm/someuri/")).addChildElement(
```

Client Code

```
46         "name").addTextNode("test");
47     msg.saveChanges();
48
49     System.err.println("Sending message to URL: " + endpoint.getURL());
50
51     SOAPMessage reply = connection.call(msg, endpoint);
52
53     System.err.println("Message logged in \"sent.msg\"");
54     FileOutputStream sentFile = new FileOutputStream("sent.msg");
55     msg.writeTo(sentFile);
56     sentFile.close();
57
58     System.out.println("Received response from: " + endpoint);
59
60     // Document source, do identity transform to write to
61     // output stream
62     System.out.println("Response:");
63     TransformerFactory tFact = TransformerFactory.newInstance();
64     Transformer transformer = tFact.newTransformer();
65     Source src = reply.getSOAPPart().getContent();
66     StreamResult result = new StreamResult(System.out);
67     transformer.transform(src, result);
68     System.out.println();
69
70     connection.close();
71
72     } catch (Throwable e) {
73         e.printStackTrace();
74     }
75 }
76 }
```

Examination of Client

- Create connection to server

Example 1-1 (again)

```
26     ...     URLEndpoint endpoint = null;
27     endpoint = new URLEndpoint(RECEIVER_URI);
28
29     SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
30     SOAPConnection connection = scf.createConnection();
    ...
```

- Create message in very DOM-like fashion
- Send message, get response
 - Note the return value is the response

Example 1-1 (again)

```
51     ...     SOAPMessage reply = connection.call(msg, endpoint);
    ...
```

- Process the response as necessary
- Simple in theory, complicated in details

Server Code

- Here is the service code that goes along with the preceding client
- We have chosen here to implement as a servlet

Example 1-2

```
1 // Sample servlet that receives messages
2
3 package simple.receiver;
4
5 import javax.xml.messaging.*;
6 import javax.xml.soap.*;
7 import javax.servlet.*;
8 import javax.servlet.http.*;
9
10 // Extend JAXMServlet class
11
12 public class SimpleSOAPServlet
13     extends JAXMServlet
14     implements ReqRespListener
15 {
16     // This is thread safe, so it's okay
17     private MessageFactory fac = null;
18
19     // Handle the message; return value is response.
20     public SOAPMessage onMessage(SOAPMessage message) {
21         System.out.println("in onMessage() - SOAP servlet");
22         try {
23             System.out.println("Here's the message: ");
24             message.writeTo(System.out);
25
26             if (fac == null)
27                 fac = MessageFactory.newInstance();
28             SOAPMessage msg = fac.createMessage();
29             SOAPEnvelope env = msg.getSOAPPart().getEnvelope();
30
31             env.getBody().addChildElement(env.createName("Response"))
32                 .addTextNode("This is a response");
33
34             return msg;
35         } catch (Exception e) {
36             System.err.println("Error handling message" + e);
37             return null;
38         }
39     }
40 }
```

Examination of Server

- **Note here that we are extending JAXMServlet , not HttpServlet**

Example 1-2 (again)

```
12     ...
13     public class SimpleSOAPServlet
14         extends JAXMServlet
15         implements ReqRespListener
16     ...
```

- **We have one method to implement: onMessage**
 - Parameter is inbound message from client
 - Our job is to return the outbound message

Example 1-2 (again)

```
20     ...
21     public SOAPMessage onMessage(SOAPMessage message) {
22     ...
```

- **To make the new message, we use a private MessageFactory**

Example 1-2 (again)

```
26     ...
27         if (fac == null)
28             fac = MessageFactory.newInstance();
29         SOAPMessage msg = fac.createMessage();
30     ...
```

- **Details of XML processing are the same as for the client**

Module Summary

- **Low-level (DOM-like) approach to web services works in Java**
- **There are several frameworks that make web services even easier**

Module 2

Java Server Faces

Module Goals and Objectives

Major Module Goals

Understand what JSF is and how it can be used to the student's advantage

Understand the overall structure of JSF and a JSF application

Specific Module Objectives

Upon completion of this module, students will be able to:

Perform simple maintenance on existing JSF applications

Section 2-1

Java Server Faces Overview

What is JSF?

- **Much like Struts, JSF is a framework for developing Java web applications using the Model-View-Controller paradigm**
- **However, Java Server Faces is much more concerned with the View portion of your application**
 - JSF is designed to integrate easily with Struts, which is much more concerned with the Controller
- **If you're familiar with Struts, initially JSF will look like the same concepts with all the names changed to be different**
 - Like a C# programmer looking at Java for the first time :)

Controller-centric Focus of Struts

- **Struts is focused on locking down the Controller of your application**
- **The UI for a vanilla Struts application is written in the standard HTML paradigm**
 - There are taglibs (`html`) that make writing the UI *much* simpler than writing raw HTML
 - But the programmer is always aware of the basic request/response mechanism
 - * That is, no matter which buttons get clicked, the entire form has to be dealt with
 - Struts has no mechanism for handling actions that only manipulate the UI versus actions that invoke the Controller layer

Different Focus for JSF

- **JSF is different. From the homepage:**

JavaServer Faces technology simplifies building user interfaces for JavaServer applications. Developers of various skill levels can quickly build web applications by: assembling reusable UI components in a page; connecting these components to an application data source; and wiring client-generated events to server-side event handlers.

—<http://java.sun.com/j2ee/javaserverfaces/>

- **JSF is more interested in building *componentized* UIs**
 - JSF UI components are objects on the server
 - Each component is responsible for rendering itself (usually in HTML) on the browser
- **JSF also draws a distinction between *actions* and *events***
 - "Actions" are events from the UI which are to be handled by the Controller layer
 - * Handled by "action controllers"
 - "Events" are events from the UI which cause changes in the UI components themselves
 - * Handled by "event listeners"
- **Rule of thumb: *actions* generally result in page navigation, *events* do not**

Whither Struts?

- **Struts as it currently exists will probably be supplanted**
- **From the Struts site:**

Originally, the Apache Struts software was distributed as one monolithic bundle. Today, the Apache Struts project is comprised of two distinct frameworks and several other subprojects. The two frameworks are Struts Core and Struts Shale. Struts Core is the original action/page framework. Struts Shale is an event/component framework based on JavaServer Faces. Both frameworks are first-class citizens of the Apache Struts project.

—<http://struts.apache.org/>

- ***Struts Core* is the old Struts you know and love**
 - Still under active development
- ***Struts Shale* is the new Struts project build from the ground up around Java Server Faces**
- **A general recommendation:**
 - For new, non-critical/personal practice apps: use *Struts Shale*
 - For new, critical apps: use *Struts Core* or standard JSF
 - For existing Struts "Classic" apps: stay with *Struts Core* for now

Section 2-2

Writing a JSF Application

Components For a Simple Web App

- Many of the concepts you already know from Struts carry over into JSF

Component type	Implemented with	(corresponding Struts implementation)
View components	JSPs	JSPs
Controller components	Action Controller POJOs	Subclasses of Action
View-Controller glue	JavaBean POJOs	Subclasses of ActionForm
Framework configuration	faces-config.xml	struts-config.xml

- Notice that JSF doesn't depend on specialized classes anymore for things like forms and actions
 - Makes for easier integration into O/R mapping frameworks (like Hibernate) &c.

faces-config.xml

1. **navigation-rule** element defines the UI flow, similar to Struts actions/forwards
2. **managed-bean** element defines form beans used by the UI
 - The bean classes themselves can have validation methods

Example 2-1

```
1 <?xml version='1.0' encoding='UTF-8'?>
  ...
41 <!DOCTYPE faces-config PUBLIC
42   "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
43   "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
44
45 <faces-config>
  ...
56 <navigation-rule>
57   <description>
58     The decision rule used by the NavigationHandler to
59     determine which view must be displayed after the
60     current view, greeting.jsp is processed.
61   </description>
62   <from-view-id>/greeting.jsp</from-view-id>
63   <navigation-case>
64     <description>
65       Indicates to the NavigationHandler that the response.jsp
66       view must be displayed if the Action referenced by a
67       UICommand component on the greeting.jsp view returns
68       the outcome "success".
69     </description>
70     <from-outcome>success</from-outcome>
71     <to-view-id>/response.jsp</to-view-id>
72   </navigation-case>
73 </navigation-rule>
  ...
94 <managed-bean>
95   <description>
96     The "backing file" bean that backs up the guessNumber webapp
97   </description>
98   <managed-bean-name>UserNumberBean</managed-bean-name>
99   <managed-bean-class>guessNumber.UserNumberBean</managed-bean-class>
100  <managed-bean-scope>session</managed-bean-scope>
101  <managed-property>
102    <property-name>minimum</property-name>
103    <property-class>int</property-class>
104    <value>0</value>
105  </managed-property>
```

faces-config.xml

```
106     <managed-property>
107         <property-name>maximum</property-name>
108         <property-class>int</property-class>
109         <value>10</value>
110     </managed-property>
111
112 </managed-bean>
    ...
```

web.xml

- **At minimum, a <servlet> and <servlet-mapping> are required to hook in Faces, just like in Struts**

Example 2-2

```
81     ...
82     <!-- Faces Servlet -->
83     <servlet>
84         <servlet-name>Faces Servlet</servlet-name>
85         <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
86         <load-on-startup> 1 </load-on-startup>
87     </servlet>
88
89     <!-- Faces Servlet Mapping -->
90     <servlet-mapping>
91         <servlet-name>Faces Servlet</servlet-name>
92         <url-pattern>/guess/*</url-pattern>
93     </servlet-mapping>
94     ...
```

- **In JSF, the individual JSPs can't be used unless they are accessed through the servlet.**
 - This means we have to restrict access to the JSPs to prevent accidental use

Example 2-2 (again)

```
95     <security-constraint>
96     ...
102     <web-resource-collection>
103         <web-resource-name>
104             Restrict access to JSP pages
105         </web-resource-name>
106         <url-pattern>/greeting.jsp</url-pattern>
107         <url-pattern>/response.jsp</url-pattern>
108     </web-resource-collection>
109     <auth-constraint>
110         <description>
111             With no roles defined, no access granted
112         </description>
113     </auth-constraint>
114 </security-constraint>
115     ...
```

A JSP Page

- This is `greeting.jsp`

Example 2-3

```
...
39 <HTML>
40   <HEAD> <title>Hello</title> </HEAD>
41   <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
42   <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
43   <body bgcolor="white">
44     <f:view>
45     <h:form id="helloForm" >
46       <h2>Hi. My name is Duke. I'm thinking of a number from
47       <h:outputText value="#{UserNumberBean.minimum}"/> to
48       <h:outputText value="#{UserNumberBean.maximum}"/>. Can you guess
49       it?</h2>
50
51       <h:graphicImage id="waveImg" url="/wave.med.gif" />
52     <h:inputText id="userNo" value="#{UserNumberBean.userNumber}"
53       validator="#{UserNumberBean.validate}"/>
54     <h:commandButton id="submit" action="success" value="Submit" />
55     <p>
56     <h:message style="color: red; font-family: 'New Century Schoolbook', serif;
font-style: oblique; text-decoration: overline" id="errors1" for="userNo"/>
57
58     </h:form>
59     </f:view>
60   </body>
61 </HTML>
```

- As you can see, it looks superficially like a Struts page

A Form Bean

- Several read-write properties

Example 2-4

```
...
50 public class UserNumberBean {
    ...
64     public void setUserNumber(Integer user_number) {
65         userNumber = user_number;
66         System.out.println("Set userNumber " + userNumber);
67     }
68
69
70     public Integer getUserNumber() {
71         System.out.println("get userNumber " + userNumber);
72         return userNumber;
73     }
    ...
}
```

- Some read-only properties

Example 2-4 (again)

```
...
76     public String getResponse() {
77         if (userNumber != null && userNumber.compareTo(randomInt) == 0) {
78             return "Yay! You got it!";
79         } else {
80             return "Sorry, " + userNumber + " is incorrect.";
81         }
82     }
    ...
}
```

- And, of course, a validator

Example 2-4 (again)

```
...
128     public void validate(FacesContext context,
129                          UIComponent component,
130                          Object value) throws ValidatorException {
    ...
}
```

Module Summary

- JSF is probably the future of Java web frameworks
- Very useful for RAD (rapid application development)
- JSF is complementary to Struts