5

Creating JAX-WS Clients

Objectives

After completing this lesson, you should be able to do the following:

- Use tools to generate JAX-WS client artifacts
- Call SOAP web services by using JAX-WS in a Java SE environment
- Call SOAP web services by using JAX-WS in a Java EE environment
- Use JAXB binding customization with a SOAP web service
- Create a JAX-WS dispatch client
- Create a client that consumes WS-Policy–enhanced services (WS-MakeConnection)

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 1: Introduction to Web Services
- Lesson 2: Creating XML Documents
- Lesson 3: Processing XML with JAXB
- Lesson 4: SOAP Web Services Overview
- **Lesson 5: Creating JAX-WS Clients** — You are here!

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 6: Exploring REST Services
- Lesson 7: Creating REST Clients
- Lesson 8: Bottom Up JAX Web Services
- Lesson 9: Top Down JAX Web Services
- Lesson 10: Implementing JAX RS Web Services

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 11: Web Service Error Handling
- Lesson 12: Java EE Security and Securing JAX WS

JAX-WS Client Model

There are two method of calling SOAP web services when using JAX-WS clients.

- Pre-generated static clients: This is the most convenient. Concrete service implementations provide access to web services using generated Java classes.
  - Tools read WSDL and produce Java artifacts.
  - There is little to no direct interaction with SOAP or XML.
- Dispatch clients: A generic service instance is used to dispatch hand-crafted SOAP messages to a web service.
  - SOAP message must be constructed by developer-supplied code.
  - Can generate non-standard messages if needed
  - No compile-time dependencies

Role of JAXB in JAX-WS

- WSDL type section contains XML Schemas.
- A tool similar to xjc processes the XML Schema elements and produces JAXB annotated classes.
- Local copies of the XML Schemas can be modified with JAXB custom binding annotations to modify the generated Java artifacts.
  - As long as the XML transmitted is compatible with what the service is expecting
- Client-side generated classes may have the same name as server-side classes. They are completely different classes.
  - If server-side JAXB classes have business logic, it will not be available on the client side.

Creating a JAX-WS Client

- The JAX-WS Reference Implementation (RI) is included in Java SE.
- Basic SOAP web services can be used by a Java SE client.
- Developers use `wsimport` (similar to `xjc`) to generate client artifacts.
  - All you need is the web service's WSDL.

- Adding a new Web Service Client to NetBeans will download a local copy of the service's WSDL and configure a wsimport task to run against the local WSDL as a standard part of the build process for the project.

- If the server-side WSDL changes, the client will NOT see those changes unless you take action. In the Web Service References branch of your NetBeans project, right-click a web service and choose Refresh.

- Make sure to select the check box that says "Also replace local wsdl file with original wsdl located at."

`wsimport` Produced Classes

Running `wsimport` on a "hello world" WSDL (one port with one operation) produces six `.java` files.

- `GetHello.java`: JAXB annotated class used to marshall/unmarshal the SOAP request body
- `GetHelloResponse.java`: JAXB annotated class used to marshall/unmarshal the SOAP response body
- `Hello.java`: JAX-WS port class
- `HelloService.java`: JAX-WS service class
- `ObjectFactory.java`: JAXB factory
- `package-info.java`: JAXB package-level annotations

Local WSDL Files

When configuring a Web Service Client reference in NetBeans, two local copies of the WSDL file are made.

- `MyProject/xml-resources/`
  - You must switch to the Files tab to inspect this directory.
  - This copy is used when running wsimport every time you build your project.
- `MyProject/src/META-INF/wsdl/`
  - Packaged with the application so that a copy of the WSDL does not need to be downloaded when initializing the client
  - Clients can improve performance by using this copy but, by default, it is not used.

Calling a Web Service

After the JAX-WS artifacts are generated, calling a web service is fairly easy.

The `Service` subclass that loads the WSDL. A port factory.

```
HelloService service = new HelloService();
Hello port = service.getHelloPort();
String s = port.getHello("Matt");
System.out.println("Result: " + s);
```

The port with methods that match the operations

Call the web service.

Almost all the steps are the same when creating an EE client. If your client is a managed component that supports injection of resources, you can obtain the service or port via injection.

```
@WebServiceRef(wsdlLocation = "WEB-
  INF/wsdl/localhost_7001/HelloWS/HelloService
  .wsdl")
private HelloService service;
```

Optional performance-enhancing attribute

```
@WebServiceRef(wsdlLocation = "WEB-
  INF/wsdl/localhost_7001/HelloWS/HelloService
  .wsdl")
private Hello port;
```

Thread Safety

What potential threading issues exist for web service clients?

- What is the threading model of the EE component calling the web service?
  - Servlets are multi-threaded. Session EJBs are normally not.
- Can the service and/or port be shared by multiple threads?
  - It depends.
  - The JAX-WS specification says they are not guaranteed to be thread-safe.
  - A JAX-WS implementation may make one or both thread-safe.
  - The code that NetBeans inserts may be thread-safe when deployed to one type of application server but not another.

Creating a Dispatch Client

- A dispatch client does not require `wsimport` or any generated artifacts.
- Dispatch clients manually create SOAP messages and have greater flexibility to create custom SOAP requests.
  - Low-level but powerful

Practice 5-6 shows you how to create a dispatch client.

```
MessageFactory mf =
   MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);
SOAPMessage request = mf.createMessage();
SOAPPart part = request.getSOAPPart();
SOAPEnvelope envelope = part.getEnvelope();
SOAPBody body = envelope.getBody();
SOAPElement operation = body.addChildElement("createDeck", "ns1",
   "http://ejbs/");
```

clientgen

`clientgen` is a WebLogic-specific alternative to `wsimport`.

- When using WS-* extensions, all WSDL to JAX-WS compilers may not understand WS-Policy statements.
- `wsimport`, `clientgen`, and most WSDL compilers provide an Ant task that can be added to your project's build process.
- Do NOT use the Web Service Client option in NetBeans. It configures only `wsimport`.

WS-MakeConnection

WS-MakeConnection is a fairly simple WS-* extension that is supported by WebLogic Server.

- WS-MakeConnection enables polling for SOAP responses instead of requiring a persistent connection.
- WS-MakeConnection is a simple example of WS-Policy.
  - Server and client code can be the same as without the policy.
  - SOAP messages will be different even though the server and client code remain the same.

Adding a VM option to WebLogic Server to dump all server-side SOAP messages to the console can help you to see the difference.

```
-Dcom.sun.xml.ws.transport.http.HttpAdapter.dump=true
```

The methods available in a JAX-WS–generated web service client are determined by:

a. The imported WSDL
b. The service endpoint
c. JAXB binding customizations
d. @WebServiceRef annotation attributes

To call a web service with JAX-WS, you must first use wsimport.

a. True
b. False

Resources

Topic	Website
JAX-WS Homepage	http://jax-ws.java.net/
JSR 224: JavaTM API for XML-Based Web Services (JAX-WS) 2.0	http://jcp.org/en/jsr/detail?id=224
The Java EE 6 Tutorial - Building Web Services with JAX-WS	http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html
Developing WebLogic Web Service Clients	http://docs.oracle.com/cd/E24329_01/web.1211/e24964/client.htm
Oracle WebLogic Server Ant Task Reference	http://docs.oracle.com/cd/E24329_01/web.1211/e24981/anttasks.htm

Summary

In this lesson, you should have learned how to:

- Use tools to generate JAX-WS client artifacts
- Call SOAP web services by using JAX-WS in a Java SE environment
- Call SOAP web services by using JAX-WS in a Java EE environment
- Use JAXB binding customization with a SOAP web service
- Create a JAX-WS dispatch client
- Create a client that consumes WS-Policy–enhanced services (WS-MakeConnection)

Practice 5: Overview

This practice covers the following topics:

- Selecting the JAXB Data Binding and JAXB Providers
- Creating a Card Deck Web Service
- Creating a Java SE Web Service Client
- Creating a Java EE Web Service Client
- Binding Customization
- Creating a JAX-WS Dispatch Web Service Client
- Using WS-MakeConnection with a JAX-WS Client