Migrating from JAX-RPC to JAX-WS

{scrollbar}

This tutorial will take you through the steps that are most commonly involved in porting a Web Service from JAX-RPC to JAX-WS. This tutorial does not go into details on why these changes are required and the concepts behind it, it is rather a quick note that helps you to deal with migrating your application from the older web service stack to new stack.

Even though JAX-RPC is still supported in Java EE 5, it lacks many advanced features that JAX-WS has like **Annotations**, **JAXB Binding**, **SOAP 1.2**, **RE STful Services** etc. So, it is strongly recommended that you use JAX-WS instead of JAX-RPC for implementing Web Services in Java.

2listpipe

JAXB Migration

The most notable change in JAX-WS 2.0 is the use of **JAXB 2.0** for data-binding between Java and XML. JAX-RPC 1.0 specified a limited mapping between XML and Java.
This effectively eliminates the need of using JAX-RPC mapping file where we define the mapping between Java and WSDL. But also imposes a condition that the return and request values can be able to bind to JAXB. Again, considering the wide variety of data types supported by JAXB it shouldn't be a problem.

JAX-RPC & JAXB
JAX-RPC didn't used JAXB because the first version of JAX-RPC is completed much before JAXB. So, instead of waiting for JAXB to complete JAX-RPC writers developed their own custom mapping.

Annotations

JAX-WS 2.0 relies heavily on the use of **Annotations**. These annotations are used to customize the mapping from JAVA to XML schema/WSDL and are used at runtime to create the necessary files.
Geronimo uses the **Sun** provided **wsgen** utility to create the WSDL and stubs on the fly at the deploy time by processing the annotations specified.

SOAP Standards

JAX-RPC and JAX-WS both support SOAP 1.1. The default binding supported by JAX-WS is SOAP 1.1 over HTTP. But it can also support **SOAP 1.2 binding over HTTP**. As a Java programmer you might not encounter any difference between SOAP 1.1 and SOAP 1.2

Service Endpoint Interface Requirements

As per JAX-RPC a Service Endpoint Interface must extend Remote. JAX-WS removes this condition and you can pretty much make a **POJO** class a Web Service by just adding the **@WebService** annotation at the top of the class.

JAX-RPC Converter SEIsolid package org.apache.geronimo.samples.jaxrpc; import java.math.BigDecimal; import java.rmi.Remote; import java.rmi. RemoteException; public interface Converter extends Remote { public BigDecimal dollarToRupees(BigDecimal dollars) throws RemoteException; public BigDecimal rupeesToEuro(BigDecimal rupees) throws RemoteException; } JAX-WS Converter SEIsolid package org.apache.geronimo.samples.jaxrpc; import java.math.BigDecimal; import javax.jws.WebService @WebService(name = "Converter", targetNamespace = "http://org.apache.geronimo.samples. jaxws") public interface Converter { public BigDecimal dollarToRupees(BigDecimal dollars); public BigDecimal rupeesToEuro(BigDecimal rupees); }

The main differences that you can find here are:

- **Annotations** - JAX-WS requires that all SEIs include the @WebService annotation
- **java.rmi.Remote** - The JAX-RPC SEI extends the java.rmi.Remote interface. JAX-WS no longer requires this.

Also, a JAX-WS SEI makes extensive use of annotations to specify the Java to WSDL mapping whereas this information does not exist in any form of JAX-RPC SEI (which requires JAX-RPC mapping file to map Java to WSDL)

Deployment Descriptor

The deployment descriptor (web.xml) hasn't changed from JAX-RPC to JAX-WS where we just need to expose a POJO class as a servlet and the server creates the artifacts on the fly whereas for JAX-RPC we need to specify the WSDL and JAX-RPC mapping file locations.
With JAX-WS mapping to Java EE 5 and taking the advantage of annotations, the need for Web Service descriptor document **webservices.xml** is eliminated as a Web Service can be effectively described using the annotations

Client Port Lookup

The following code samples demonstrate the difference between JAX-RPC and JAX-WS in client port lookup.

JAX-RPC Converter Clientsolid package org.apache.geronimo.samples.jaxrpc; import java.net.URL; import javax.xml.namespace.QName; import javax.xml.rpc.Service; import javax.xml.rpc.ServiceFactory; import javax.xml.rpc.ServiceException; //a part of client URL url = new URL("http://localhost:8080/jaxrpc-converter/converter?wsdl"); QName qname = new QName("http://org.apache.geronimo.samples.jaxrpc/","ConverterService"); ServiceFactory factory = ServiceFactory.newInstance(); Service service = factory.createService(url, qname); Converter conv = (Converter) service.getPort(Converter.class); JAX-WS Converter Clientsolid package org.apache.geronimo.samples.jaxrpc; import java.net.URL; import javax.xml.namespace.QName; import javax.xml.ws.Service; //a part of client URL url = new URL("http://localhost:8080/jaxrpc-converter/converter?wsdl"); QName qname = new QName("http://org.apache.geronimo.samples.jaxrpc/","ConverterService"); Service service = Service.create(url, qname); Converter conv = (Converter) service.getPort(Converter.class);

The main differences we can observe here are

- The creation of **ServiceFactory** instance is no longer required for creating the Service
- Service maps to **javax.xml.rpc.Service** in JAX-RPC and to **javax.xml.ws.Service** in JAX-WS

RESTful Services

JAX-WS introduced **RESTful Web Services** as successor for SOAP based Web Service. RESTful services already got quite support from many vendors like **Google AdSense**, **Yahoo API's**, **Amazon** etc.
The important things that are introduced in JAX-WS to support RESTful services are:

Provider

Web service endpoints may choose to work at the XML message level by implementing the Provider interface. Here the endpoints access messages or message payloads using this low level, generic API.

Dispatch

The Dispatch API is intended for advanced XML developers who prefer to use XML constructs at the java.lang.transform.Source or javax.xml.soap.SOAPMessage level. For added convenience use of the Dispatch API with JAXB data-bound objects is supported.

Asynchronous Operations

A major difference in operation mapping for JAX-WS over JAX-RPC is the introduction of asynchronous operations. Any WSDL operation with a two-way message flow, or one where the client expects to receive a response, can be mapped to an asynchronous Java representation.
For further reference about asynchronous operations in JAX-WS refer to the references section.

MTOM and SAAJ

JAXWS 2.0 brings in support for optimized transmission of binary data as specified by MTOM (SOAP Message Transmission Optimization Mechanism) and SAAJ (SOAP with Attachments API for Java).
MTOM allows optimized transmission of binary data - any xs:base64Binary or xs:hexBinary schema type can be send as attachment following rules defined by MTOM specification.

References

- Developing a JAX-WS POJO Web Service & Developing JAX-RPC Web Services
- http://www.ibm.com/developerworks/webservices/library/ws-tip-jaxwsrpc.html
- https://jax-rpc.dev.java.net/jaxws20-ea2/docs/UsersGuide.html
- http://blogs.sun.com/trajesh/entry/migrating_from_jax_rpc_to