JAX-WS SOAP Web Services Example Tutorial

www.journaldev.com/9123/jax-ws-soap-web-services-example-tutorial

Pankaj

Web Services work on client-server model where they communicate over the network. Server side component provides the endpoint URL where service is located and client application can invoke different methods.

There are two types of web services:

1. SOAP Web Services
2. Restful Web Services

Today we will use JAX-WS to create SOAP based web services, but first we will go through some of the jargon words used in SOAP web services.

SOAP

SOAP stands for Simple Object Access Protocol. SOAP is an XML based industry standard protocol for designing and developing web services. Since it's XML based, it's platform and language independent. So our server can be based on JAVA and client can be on .NET, PHP etc. and vice versa.

WSDL

WSDL stands for Web Service Description Language. WSDL is an XML based document that provides technical details about the web service. Some of the useful information in WSDL document are: method name, port types, service end point, binding, method parameters etc.

UDDI

UDDI is acronym for Universal Description, Discovery and Integration. UDDI is a directory of web services where client applications can lookup for web services. Web Services can register to the UDDI server and make them available to client applications.

Advantages of Web Services

Some of the advantages of web services are:

- Interoperability: Because web services work over network and use XML technology to communicate, it can be developed in any programming language supporting web services development.
- Reusability: One web service can be used by many client applications at the same time. For example, we can expose a web service for technical analysis of a stock and it can be used by all the banks and financial institutions.
- Loose Coupling: Web services client code is totally independent with server code, so we have achieved loose coupling in our application. This leads to easy maintenance and easy to extend.
- Easy to deploy and integrate
- Multiple service versions can be running at same time.

JAX-WS

JAX-WS stands for Java API for XML Web Services. JAX-WS is XML based Java API to build web services server and client application. It's part of standard Java API, so we don't need to include anything else which working with it.

Now that we have gone through the web services terminologies, let's go ahead and create a JAX-WS web service. We will create a web service that will expose methods to add, delete and get person objects. So first of all we will create a model bean for our data.

```
package com.journaldev.jaxws.beans;

import java.io.Serializable;

public class Person implements Serializable{

 private static final long serialVersionUID = -
5577579081118070434L;

 private String name;
 private int age;
 private int id;

 public String getName() {
  return name;
 }

 public void setName(String name) {
  this.name = name;
 }

 public int getAge() {
  return age;
 }

 public void setAge(int age) {
  this.age = age;
 }

 public int getId() {
  return id;
 }

 public void setId(int id) {
  this.id = id;
 }

 @Override
 public String toString(){
  return id+"::"+name+"::"+age;
 }

}
```

Now we will have to create an interface where we will declare the methods we will expose in our web services.

```
package com.journaldev.jaxws.service;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

import com.journaldev.jaxws.beans.Person;

@WebService
@SOAPBinding(style =
SOAPBinding.Style.RPC)
public interface PersonService {

 @WebMethod
 public boolean addPerson(Person p);

 @WebMethod
 public boolean deletePerson(int id);

 @WebMethod
 public Person getPerson(int id);

 @WebMethod
 public Person[] getAllPersons();
}
```

Notice the use of `@WebService` and `@SOAPBinding` annotations from JAX-WS API. We can create SOAP web services in **RPC style** or **Document style**. We can use any of these styles to create web services, the different is seen in the way WSDL file is generated.

Now we will write the implementation class as shown below.

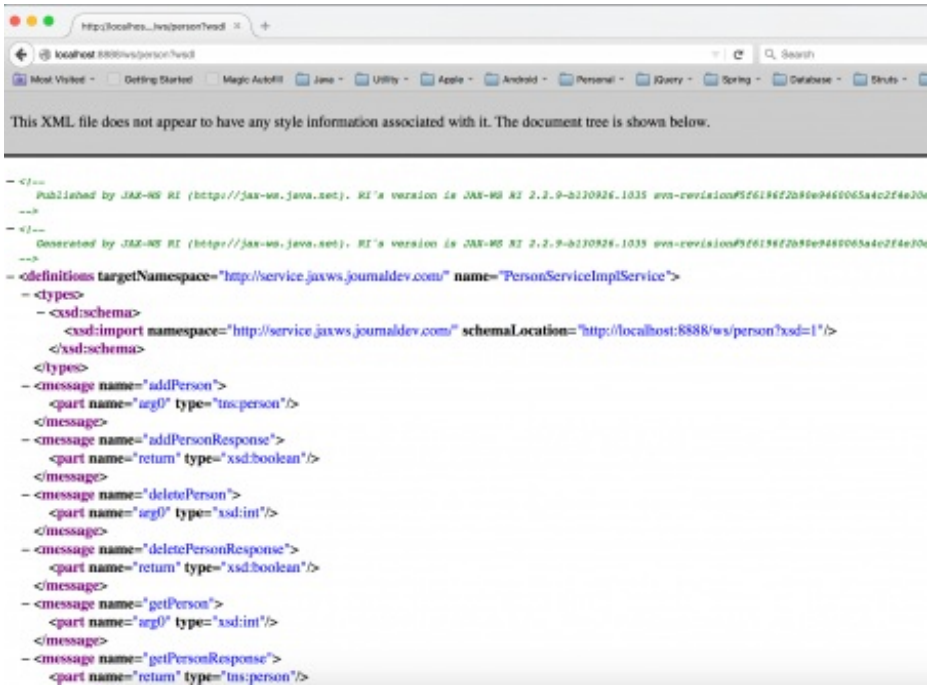
```
package com.journaldev.jaxws.service;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

import javax.jws.WebService;

import com.journaldev.jaxws.beans.Person;

@WebService(endpointInterface = "com.journaldev.jaxws.service.PersonService")
public class PersonServiceImpl implements PersonService {

 private static Map<Integer,Person> persons = new HashMap<Integer,Person>();

 @Override
 public boolean addPerson(Person p) {
  if(persons.get(p.getId()) != null) return false;
  persons.put(p.getId(), p);
  return true;
 }

 @Override
 public boolean deletePerson(int id) {
  if(persons.get(id) == null) return false;
  persons.remove(id);
  return true;
 }

 @Override
 public Person getPerson(int id) {
  return persons.get(id);
 }

 @Override
 public Person[] getAllPersons() {
  Set<Integer> ids = persons.keySet();
  Person[] p = new Person[ids.size()];
  int i=0;
  for(Integer id : ids){
   p[i] = persons.get(id);
   i++;
  }
  return p;
 }

}
```

Most important part is the `@WebService` annotation where we are providing *endpointInterface* value as the interface we have for our web service. This way JAX-WS know the class to use for implementation when web service methods are invoked.

Our web service business logic is ready, let's go ahead and publish it using JAX-WS Endpoint class.

```
package com.journaldev.jaxws.service;

import javax.xml.ws.Endpoint;

public class SOAPPublisher {

 public static void main(String[] args) {
   Endpoint.publish("http://localhost:8888/ws/person", new PersonServiceImpl());
 }

}
```

Just run the above program and your web service will be published at the given endpoint in the program. We can access it's WSDL document by adding **?wsdl** to the endpoint url as shown in below image.



Here is the WSDL code, we will use some of the values from these while writing the client code.

```
<?xml version="1.0" encoding="UTF-8"?><!-- Published by JAX-WS RI (http://jax-
ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn-
revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e. --><!-- Generated by JAX-WS RI
(http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn-
revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e. --><definitions
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://service.jaxws.journaldev.com/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://service.jaxws.journaldev.com/"
name="PersonServiceImplService">
<types>
<xsd:schema>
<xsd:import namespace="http://service.jaxws.journaldev.com/"
```

```
<xsd:import namespace="http://service.jaxws.journaldev.com/
schemaLocation="http://localhost:8888/ws/person?xsd=1"></xsd:import>
</xsd:schema>
</types>
<message name="addPerson">
<part name="arg0" type="tns:person"></part>
</message>
<message name="addPersonResponse">
<part name="return" type="xsd:boolean"></part>
</message>
<message name="deletePerson">
<part name="arg0" type="xsd:int"></part>
</message>
<message name="deletePersonResponse">
<part name="return" type="xsd:boolean"></part>
</message>
<message name="getPerson">
<part name="arg0" type="xsd:int"></part>
</message>
<message name="getPersonResponse">
<part name="return" type="tns:person"></part>
</message>
<message name="getAllPersons"></message>
<message name="getAllPersonsResponse">
<part name="return" type="tns:personArray"></part>
</message>
<portType name="PersonService">
<operation name="addPerson">
<input
wsam:Action="http://service.jaxws.journaldev.com/PersonService/addPersonRequest"
message="tns:addPerson"></input>
<output
wsam:Action="http://service.jaxws.journaldev.com/PersonService/addPersonResponse"
message="tns:addPersonResponse"></output>
</operation>
<operation name="deletePerson">
<input
wsam:Action="http://service.jaxws.journaldev.com/PersonService/deletePersonRequest"
message="tns:deletePerson"></input>
<output
wsam:Action="http://service.jaxws.journaldev.com/PersonService/deletePersonResponse"
message="tns:deletePersonResponse"></output>
</operation>
<operation name="getPerson">
<input
wsam:Action="http://service.jaxws.journaldev.com/PersonService/getPersonRequest"
message="tns:getPerson"></input>
<output
wsam:Action="http://service.jaxws.journaldev.com/PersonService/getPersonResponse"
message="tns:getPersonResponse"></output>
</operation>
<operation name="getAllPersons">
<input
wsam:Action="http://service.jaxws.journaldev.com/PersonService/getAllPersonsRequest"
message="tns:getAllPersons"></input>
<output
wsam:Action="http://service.jaxws.journaldev.com/PersonService/getAllPersonsResponse"
```

```
message="tns:getAllPersonsResponse"></output>
</operation>
</portType>
<binding name="PersonServiceImplPortBinding" type="tns:PersonService">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc">
</soap:binding>
<operation name="addPerson">
<soap:operation soapAction=""></soap:operation>
<input>
<soap:body use="literal" namespace="http://service.jaxws.journaldev.com/">
</soap:body>
</input>
<output>
<soap:body use="literal" namespace="http://service.jaxws.journaldev.com/">
</soap:body>
</output>
</operation>
<operation name="deletePerson">
<soap:operation soapAction=""></soap:operation>
<input>
<soap:body use="literal" namespace="http://service.jaxws.journaldev.com/">
</soap:body>
</input>
<output>
<soap:body use="literal" namespace="http://service.jaxws.journaldev.com/">
</soap:body>
</output>
</operation>
<operation name="getPerson">
<soap:operation soapAction=""></soap:operation>
<input>
<soap:body use="literal" namespace="http://service.jaxws.journaldev.com/">
</soap:body>
</input>
<output>
<soap:body use="literal" namespace="http://service.jaxws.journaldev.com/">
</soap:body>
</output>
</operation>
<operation name="getAllPersons">
<soap:operation soapAction=""></soap:operation>
<input>
<soap:body use="literal" namespace="http://service.jaxws.journaldev.com/">
</soap:body>
</input>
<output>
<soap:body use="literal" namespace="http://service.jaxws.journaldev.com/">
</soap:body>
</output>
</operation>
</binding>
<service name="PersonServiceImplService">
<port name="PersonServiceImplPort" binding="tns:PersonServiceImplPortBinding">
<soap:address location="http://localhost:8888/ws/person"></soap:address>
</port>
</service>
```

```
</definitions>
```

Here is a client program where we are invoking our web service.

```
package com.journaldev.jaxws.service;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.Arrays;

import javax.xml.namespace.QName;
import javax.xml.ws.Service;

import com.journaldev.jaxws.beans.Person;

public class SOAPPublisherClient {

 public static void main(String[] args) throws MalformedURLException {
  URL wsdlURL = new URL("http://localhost:8888/ws/person?wsdl");
  //check above URL in browser, you should see WSDL file

  //creating QName using targetNamespace and name
  QName qname = new QName("http://service.jaxws.journaldev.com/",
"PersonServiceImplService");

  Service service = Service.create(wsdlURL, qname);

  //We need to pass interface and model beans to client
  PersonService ps = service.getPort(PersonService.class);

  Person p1 = new Person(); p1.setName("Pankaj"); p1.setId(1); p1.setAge(30);
  Person p2 = new Person(); p2.setName("Meghna"); p2.setId(2); p2.setAge(25);

  //add person
  System.out.println("Add Person Status="+ps.addPerson(p1));
  System.out.println("Add Person Status="+ps.addPerson(p2));

  //get person
  System.out.println(ps.getPerson(1));

  //get all persons
  System.out.println(Arrays.asList(ps.getAllPersons()));

  //delete person
  System.out.println("Delete Person Status="+ps.deletePerson(2));

  //get all persons
  System.out.println(Arrays.asList(ps.getAllPersons()));

 }

}
```

When we execute this program, we get this output.

```
Add Person Status=true
Add Person Status=true
1::Pankaj::30
[1::Pankaj::30,
2::Meghna::25]
Delete Person Status=true
[1::Pankaj::30]
```

When I run the program again, we get this output.

```
Add Person Status=false
Add Person Status=true
1::Pankaj::30
[1::Pankaj::30,
2::Meghna::25]
Delete Person Status=true
[1::Pankaj::30]
```

Notice that in the second run, add person status is false because it was already added in the first run.

JAX-WS Client Stubs and Test Program

If you look at the above program, we are using the server code itself. However web services just expose WSDL and third party applications don't have access to these classes. So in that case, we can use `wsimport` utility to generate the client stubs. This utility comes with standard installation of JDK. Below image shows what all java classes we get when we run this utility.

```
pankaj:temp pankaj$
pankaj:temp pankaj$ wsimport -s . http://localhost:8888/ws/person?wsdl
parsing WSDL...


Generating code...


Compiling code...

pankaj:temp pankaj$ ls -ltr com/journaldev/jaxws/service/
total 96
-rw-r--r--  1 pankaj  staff   127 Oct  3 01:38 package-info.java
-rw-r--r--  1 pankaj  staff   270 Oct  3 01:38 package-info.class
-rw-r--r--  1 pankaj  staff  3255 Oct  3 01:38 PersonServiceImplService.java
-rw-r--r--  1 pankaj  staff  2433 Oct  3 01:38 PersonServiceImplService.class
-rw-r--r--  1 pankaj  staff  2224 Oct  3 01:38 PersonService.java
-rw-r--r--  1 pankaj  staff  1783 Oct  3 01:38 PersonService.class
-rw-r--r--  1 pankaj  staff  1807 Oct  3 01:38 PersonArray.java
-rw-r--r--  1 pankaj  staff   885 Oct  3 01:38 PersonArray.class
-rw-r--r--  1 pankaj  staff  2031 Oct  3 01:38 Person.java
-rw-r--r--  1 pankaj  staff   934 Oct  3 01:38 Person.class
-rw-r--r--  1 pankaj  staff  1208 Oct  3 01:38 ObjectFactory.java
-rw-r--r--  1 pankaj  staff   628 Oct  3 01:38 ObjectFactory.class
pankaj:temp pankaj$
```

Just copy these classes into your client project, the only change would be the way we get `PersonService` instance. Below is the program for this, the output will be same as above client program. Notice the use of `PersonServiceImplService` class and it's method `getPersonServiceImplPort` to get the PersonService instance.

```
package com.journaldev.jaxws.service.test;

import java.util.Arrays;

import com.journaldev.jaxws.service.Person;
import com.journaldev.jaxws.service.PersonService;
import com.journaldev.jaxws.service.PersonServiceImplService;

public class TestPersonService {

 public static void main(String[] args) {

  PersonServiceImplService serviceImpl = new PersonServiceImplService();

  PersonService service = serviceImpl.getPersonServiceImplPort();

  Person p1 = new Person(); p1.setName("Pankaj"); p1.setId(1); p1.setAge(30);
        Person p2 = new Person(); p2.setName("Meghna"); p2.setId(2);
p2.setAge(25);

        System.out.println("Add Person Status="+service.addPerson(p1));
        System.out.println("Add Person Status="+service.addPerson(p2));

      //get person
        System.out.println(service.getPerson(1));

        //get all persons
        System.out.println(Arrays.asList(service.getAllPersons()));

        //delete person
        System.out.println("Delete Person Status="+service.deletePerson(2));

        //get all persons
        System.out.println(Arrays.asList(service.getAllPersons()));

 }

}
```

That’s all for a quick tutorial on JAX-WS web services, we will look more into java web services in coming posts.

Related Posts: