Web Service Tutorial

Creating and Connecting to SOAP Web Service using JAX-WS

Java API for XML Web Services (JAX-WS) is a standardized API for creating and consuming SOAP (Simple Object Access Protocol) web services.

In this Tutorial, we'll create a SOAP web service and connect to it using JAX-WS.

JAX-WS is a framework that simplifies using SOAP.

Tools

To follow this tutorial, you need to prepare the following tools:

1. A Java Development kit(JDK). Please download JDK (jdk 8 Standard Edition) from https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html.
2. The **Inject** java package. The Package can be download from https://repo1.maven.org/maven2/javax/inject/javax.inject/1/javax.inject-1.jar.

Introduction

The web service built on JAX-WS can be divided into two parts: a server and a client. The client is quite easy, because we can use wsimport tool, a web service tool provided by java, to generate client code automatically. Let's focus on the server first.

Server

There are two ways of building SOAP web services. We can go with a top-down approach or a bottom-up approach.

In a top-down (contract-first) approach, a WSDL document is created, and the necessary Java classes are generated from the WSDL. In a bottom-up (contract-last) approach, the Java classes are written, and the WSDL is generated from the Java classes.

Writing a WSDL file can be quite difficult depending on how complex your web service is. This makes the bottom-up approach an easier option. On the other hand, since your WSDL is generated from the Java classes, any change in code might cause a change in the WSDL. This is not the case for the top-down approach.

In this tutorial, we'll take a look at both approaches.

Prerequisite

The service discussed in this tutorial provides employee information. We would like to let our client check the information about employees in our company. Hence, we first have to prepare some classes about our employee. Please copy the following code and build packpages respectively.

1. Employee.java

```java
package com.baeldung.jaxws.server;

public class Employee { //the model of employee
    private int id;
    private String firstName;

    public Employee() {

    }
    public Employee(int id, String firstName) {
        this.id = id;
        this.firstName = firstName;
    }

    public int getId() {
        return id;
    }

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

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}
```

2. EmployeeRepository.java

```java
package com.baeldung.jaxws.server.repository;

import java.util.List;
import com.baeldung.jaxws.server.Employee;


public interface EmployeeRepository { //interface of the function we provide

    List<Employee> getAllEmployees() throws Exception;

    Employee getEmployee(int id) throws Exception;

    Employee updateEmployee(int id, String name) throws Exception;

    boolean deleteEmployee(int id) throws Exception;

    Employee addEmployee(int id, String name) throws Exception;

    int count();
}
```

3. EmployeeRepositoryImpl.java

```
package com.baeldung.jaxws.server.repository;

import  java.util.ArrayList;
import java.util.List;
import com.baeldung.jaxws.server.Employee;
import java.lang.Exception;

// the implementation of functions we want to provide for client
public class EmployeeRepositoryImpl implements EmployeeRepository {
    private List<Employee> employeeList;

    public EmployeeRepositoryImpl() {
        employeeList = new ArrayList<>();
        employeeList.add(new Employee(1, "Jane"));
        employeeList.add(new Employee(2, "Jack"));
        employeeList.add(new Employee(3, "George"));
    }

    public List<Employee> getAllEmployees() {
        return employeeList;
    }

    public Employee getEmployee(int id) throws
        Exception{for (Employee emp : employeeList) {
            if (emp.getId() == id) {
                return emp;
            }
        }
        throw new Exception();
    }

    public Employee updateEmployee(int id, String name) throws
        Exception{for (Employee employee1 : employeeList) {
            if (employee1.getId() == id) {
                employee1.setId(id);
                employee1.setFirstName(name);
                return employee1;
            }
        }
        throw new Exception();
    }

    public boolean deleteEmployee(int id) throws
        Exception{for (Employee emp : employeeList) {
            if (emp.getId() == id) {
                employeeList.remove(emp);
                return true;
            }
        }
        throw new Exception();
    }

    public Employee addEmployee(int id, String name)  throws Exception{
        for (Employee emp : employeeList) {
```

```
            if (emp.getId() == id) {
                throw new  Exception();
            }
        }
        Employee employee = new Employee(id, name);
        employeeList.add(employee);
        return employee;
    }

    public int count() {
        return employeeList.size();
    }
}
```

Top-down Approach

1. In the top-down approach, we define our service in WSDL (Web Service Definition Language) file. The following shows the code in our WSDL file named *employeeservicetopdown.wsdl*. If You want to know more about the WSDL, please recall information you have learned in Module 3; You can also check this blog. Create the below WSDL file under your java project.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://topdown.server.jaxws.baeldung.com/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://topdown.server.jaxws.baeldung.com/"
  qname="EmployeeServiceTopDown">
    <types>
        <xsd:schema
          targetNamespace="http://topdown.server.jaxws.baeldung.com/">
            <xsd:element name="countEmployeesResponse" type="xsd:int"/>
        </xsd:schema>
    </types>

    <message name="countEmployees">
    </message>
    <message name="countEmployeesResponse">
        <part name="parameters" element="tns:countEmployeesResponse"/>
    </message>
    <portType name="EmployeeServiceTopDown">
        <operation name="countEmployees">
            <input message="tns:countEmployees"/>
            <output message="tns:countEmployeesResponse"/>
        </operation>
    </portType>
    <binding name="EmployeeServiceTopDownSOAP"
      type="tns:EmployeeServiceTopDown">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
          style="document"/>
        <operation name="countEmployees">
            <soap:operation
              soapAction="http://topdown.server.jaxws.baeldung.com/
```

```
            EmployeeServiceTopDown/countEmployees"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="EmployeeServiceTopDown">
        <port name="EmployeeServiceTopDownSOAP"
          binding="tns:EmployeeServiceTopDownSOAP">
            <soap:address
              location="http://localhost:8080/employeeservicetopdown"/>
        </port>
    </service>
</definitions>
```

2. Generate server code using wsimport command

   To generate web service source files from a WSDL document, we can use the *wsimport* tool which is part of JDK (at $JAVA_HOME/bin). Open windows cmd and execute:

```
cd \path\to\javaproject
wsimport -s src\ -p com.baeldung.jaxws.server.topdown
   employeeservicetopdown.wsdl
```

   Wsimport is the java tool provided by your jdk. You should find it in your **%JAVA_HOME%\bin** directory. In my case, I find it in *C:\Program Files\Java\jdk1.8.0_241\bin* directory. The parameter **-s** src\means we store generated source codesin topdown directory. The parameter **-p com.baeldung.jaxws.server.topdown** means the generated code are in the package topdown. The *wsimport* tool has generated the web service endpoint interface *EmployeeServiceTopDown*. It declares the web service methods.

   The generated files:

   - *EmployeeServiceTopDown.java* – is the service endpoint interface (SEI) that contains method definitions
   - *ObjectFactory.java* – contains factory methods to create instances of schema derived classes programmatically
   - *EmployeeServiceTopDown_Service.java* – is the service provider class that can be used by a JAX-WS client

   If you get all the files above, then congratulations, let's go to next step. If you have problem at this stage, this link might help. https://www.baeldung.com/jax-ws . You can also ask questions in our wechat group.

3. Implement our service
   In step 2, we already got the service endpoint interface that contains method definitions. Next, we have to implement the methods in the interface.Create EmployeeServiceTopDownImpl.java in topdown package.

```
package com.baeldung.jaxws.server.topdown;

import javax.inject.Inject;
import javax.jws.WebMethod;
import  javax.jws.WebService;

import com.baeldung.jaxws.server.Employee;
```

```
import com.baeldung.jaxws.server.repository.EmployeeRepository;
import  com.baeldung.jaxws.server.repository.EmployeeRepositoryImpl;


@WebService(
        name = "EmployeeServiceTopDown",
        endpointInterface =
"com.baeldung.jaxws.server.topdown.EmployeeServiceTopDown",
        targetNamespace  =  "http://topdown.server.jaxws.baeldung.com/")
public class EmployeeServiceTopDownImpl
        implements  EmployeeServiceTopDown  {
    @Inject
    private EmployeeRepository employeeRepositoryImpl = new
EmployeeRepositoryImpl();
    @WebMethod
    public Employee getEmployee(int id) throws
        Exception{return
        employeeRepositoryImpl.getEmployee(id);
    }

    @WebMethod
    public Employee updateEmployee(int id, String name) throws
        Exception{return employeeRepositoryImpl.updateEmployee(id,
        name);
    }

    @WebMethod
    public boolean deleteEmployee(int id) throws Exception {
        return employeeRepositoryImpl.deleteEmployee(id);
    }

    @WebMethod
    public Employee addEmployee(int id, String name) throws Exception {
        return employeeRepositoryImpl.addEmployee(id, name);
    }

    @WebMethod
    public int countEmployees() {
        return employeeRepositoryImpl.count();
    }
}
```

4. Publish our service

   Once we implement the service endpoint interface, the server is finished successfully. To publish the web services (top-down and bottom-up), we need to pass an address and an instance of the web service implementation to the *publish()* method of the *javax.xml.ws.Endpoint* class.

```
package com.baeldung.jaxws.server;

//import com.baeldung.jaxws.server.bottomup.EmployeeServiceImpl;
import  com.baeldung.jaxws.server.topdown.EmployeeServiceTopDownImpl;

import javax.xml.ws.Endpoint;

public class EmployeeServicePublisher
    { public static void main(String[] args)
    {
        Endpoint.publish("http://localhost:8080/employeeservicetopdown", new
EmployeeServiceTopDownImpl()); //publish topdown service
//        Endpoint.publish("http://localhost:8080/employeeservice",new
EmployeeServiceImpl()); //publish bottomup service
    }
}
```

We can now run *EmployeeServicePublisher* to start the web service. To make use of CDI features, the web services can be deployed as WAR file to application servers like WildFly or GlassFish.

Bottom-up Approach

In a bottom-up approach, we have to create both the endpoint interface and the implementation classes. The WSDL is generated from the classes when the web service is published.

Let's create a web service that will perform simple CRUD(Create, Read, Update, Delete) operations on *Employee* data.

1. EmployeeService.java

   Get the service endpoint interface, corresponds to *EmployeeServiceTopDown.java* in top-down approach.

```
package com.baeldung.jaxws.server.bottomup;

import javax.jws.WebMethod;
import  javax.jws.WebService;
import java.lang.Exception;
import com.baeldung.jaxws.server.Employee;

@WebService
public interface EmployeeService {

    @WebMethod
    Employee getEmployee(int id) throws Exception;

    @WebMethod
    Employee updateEmployee(int id, String name) throws Exception;

    @WebMethod
    boolean deleteEmployee(int id) throws Exception;

    @WebMethod
    Employee addEmployee(int id, String name) throws Exception;

    // ...
}
```

2. EmployeeServiceImpl.java

In step 1, we have created the structure of the web service. we have to create the implementation of the web service. The following is web service implementation it, corresponds to the 3rd step in top-down approach.

```
package com.baeldung.jaxws.server.bottomup;

import javax.inject.Inject;
import javax.jws.WebMethod;
import  javax.jws.WebService;
import java.lang.Exception;
import com.baeldung.jaxws.server.repository.EmployeeRepository;
import com.baeldung.jaxws.server.repository.EmployeeRepositoryImpl;
import com.baeldung.jaxws.server.Employee;

@WebService(endpointInterface =
"com.baeldung.jaxws.server.bottomup.EmployeeService")
public class EmployeeServiceImpl implements EmployeeService {

    @Inject
    private EmployeeRepository employeeRepositoryImpl = new
EmployeeRepositoryImpl();
    @WebMethod
    public Employee getEmployee(int id) throws Exception{
        return employeeRepositoryImpl.getEmployee(id);
    }

    @WebMethod
    public Employee updateEmployee(int id, String name) throws Exception{
        return employeeRepositoryImpl.updateEmployee(id, name);
    }

    @WebMethod
    public boolean deleteEmployee(int id) throws Exception {
        return employeeRepositoryImpl.deleteEmployee(id);
    }

    @WebMethod
    public Employee addEmployee(int id, String name) throws Exception {
         return employeeRepositoryImpl.addEmployee(id, name);
    }

    @WebMethod
    public int countEmployees() {
        return employRepositoryImpl.count();
    }
}
```

3. Publish the service

Publish the bottom-up approach service just like the 4th step in top-down approach.

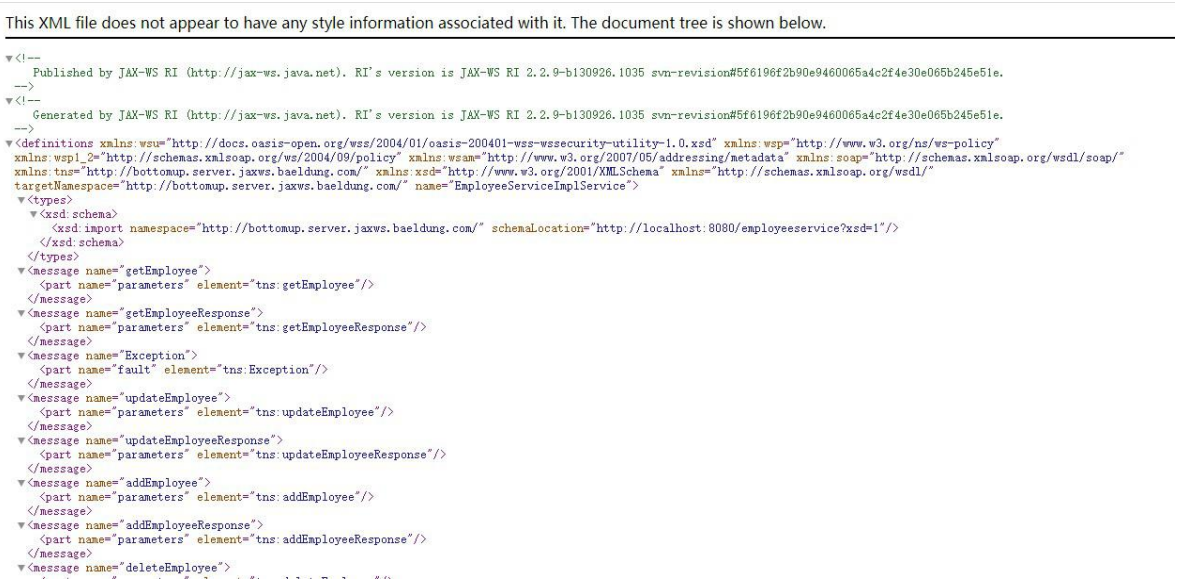
```
package com.baeldung.jaxws.server;

import com.baeldung.jaxws.server.bottomup.EmployeeServiceImpl;
//import  com.baeldung.jaxws.server.topdown.EmployeeServiceTopDownImpl;

import javax.xml.ws.Endpoint;

public class EmployeeServicePublisher {
    public static void main(String[] args) {
//        Endpoint.publish("http://localhost:8080/employeeservicetopdown", new
EmployeeServiceTopDownImpl()); //publish topdown service
        Endpoint.publish("http://localhost:8080/employeeservice",new
EmployeeServiceImpl()); //publish bottomup service
    }
}
```

Check the Published Web Service

To check if the web service has been published successfully., you can open your browser and enter the following link to check the bottom-up service.

```
http://localhost:8080/employeeservice?wsdl
```

You can get a result from the request like this:



If you got the result, congratulations. We can now focus on the client part of web service.

Client

The client can use the methods defined in the service endpoint interface. We can get the code we need in the client by wsimport.

```
wsimport -keep -s src\ -p com.baeldung.jaxws.client
http://localhost:8080/employeeservice?wsdl
```

We use the wsdl file from the web service to generate our client code. **-keep** means to keep the source code. Other parameters were explained in the top-down approach section. We should add a client to try connect the web service.

```
package com.baeldung.jaxws.client;
import java.net.URL;
import  java.lang.Exception;
import java.util.List;

public class EmployeeServiceClient {
    public static void main(String[] args) throws Exception {
        URL url = new URL("http://localhost:8080/employeeservice?wsdl");
        EmployeeServiceImplService service = new
EmployeeServiceImplService(url);
        EmployeeService  proxy  =  service.getEmployeeServiceImplPort();
        proxy.addEmployee(4,"yourname");
        Employee fourth = proxy.getEmployee(4);
        System.out.println(fourth.firstName);
    }
}
```

The program should output "yourname" in the console.

Tutorial Assessment

1. In the top-down service, can the client use **getEmployee**, **updateEmployee** methods? Please explain.
2. If we want the client connected to the bottom-up web service to use **countEmployees** method, what modification should we do in the **EmployeeService.java** and **EmployeeServiceImp.java** ?
3. What is GlassFish?

FAQ

1. The complier shows that the **Inject** class not found.
   You should add an extra package to your project. The Package can be download in https://repo1.maven.org/maven2/javax/inject/javax.inject/1/javax.inject-1.jar.
2. If you have problems in this tutorial, please feel free to contact me in our wechat group.

References

[1] https://www.baeldung.com/jax-ws

[2] https://github.com/eugenp/tutorials/tree/master/jee-7

The tutorial is based on the above projects. Much thanks to the projects author.