JAX-WS @WebService example

Example simple-webservice can be browsed at https://github.com/apache/tomee/tree/master/examples/simple-webservice

Creating Web Services with JAX-WS is quite easy. Little has to be done aside from annotating a class with `@WebService`. For the purposes of this example we will also annotate our component with `@Stateless` which takes some of the configuration out of the process and gives us some nice options such as transactions and security.

@WebService

The following is all that is required. No external xml files are needed. This class placed in a jar or war and deployed into a compliant Java EE server like TomEE is enough to have the Calculator class discovered and deployed and the webservice online.

```
import javax.ejb.Stateless;
import javax.jws.WebService;

@Stateless
@WebService(
        portName = "CalculatorPort",
        serviceName = "CalculatorService",
        targetNamespace = "http://superbiz.org/wsdl",
        endpointInterface = "org.superbiz.calculator.ws.CalculatorWs")
public class Calculator implements CalculatorWs {

    public int sum(int add1, int add2) {
        return add1 + add2;
    }

    public int multiply(int mul1, int mul2) {
        return mul1 * mul2;
    }
}
```

@WebService Endpoint Interface

Having an endpoint interface is not required, but it can make testing and using the web service from other Java clients far easier.

```
import javax.jws.WebService;

@WebService(targetNamespace = "http://superbiz.org/wsdl")
public interface CalculatorWs {

    public int sum(int add1, int add2);

    public int multiply(int mul1, int mul2);
}
```

Calculator WSDL

The wsdl for our service is autmatically created for us and available at http://127.0.0.1:4204/Calculator?wsdl. In TomEE or Tomcat this would be at http://127.0.0.1:8080/simple-webservice/Calculator?wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" name=
"CalculatorService"
                  targetNamespace="http://superbiz.org/wsdl"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:tns="http://superbiz.org/wsdl" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <xsd:schema attributeFormDefault="unqualified" elementFormDefault="unqualified"
                targetNamespace="http://superbiz.org/wsdl" xmlns:tns=
"http://superbiz.org/wsdl"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="multiply" type="tns:multiply"/>
      <xsd:complexType name="multiply">
        <xsd:sequence>
          <xsd:element name="arg0" type="xsd:int"/>
          <xsd:element name="arg1" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="multiplyResponse" type="tns:multiplyResponse"/>
      <xsd:complexType name="multiplyResponse">
        <xsd:sequence>
          <xsd:element name="return" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:element name="sum" type="tns:sum"/>
      <xsd:complexType name="sum">
        <xsd:sequence>
          <xsd:element name="arg0" type="xsd:int"/>
          <xsd:element name="arg1" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>
```

```
      <xsd:element name="sumResponse" type="tns:sumResponse"/>
      <xsd:complexType name="sumResponse">
        <xsd:sequence>
          <xsd:element name="return" type="xsd:int"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="multiplyResponse">
    <wsdl:part element="tns:multiplyResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="sumResponse">
    <wsdl:part element="tns:sumResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="sum">
    <wsdl:part element="tns:sum" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="multiply">
    <wsdl:part element="tns:multiply" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="CalculatorWs">
    <wsdl:operation name="multiply">
      <wsdl:input message="tns:multiply" name="multiply"/>
      <wsdl:output message="tns:multiplyResponse" name="multiplyResponse"/>
    </wsdl:operation>
    <wsdl:operation name="sum">
      <wsdl:input message="tns:sum" name="sum"/>
      <wsdl:output message="tns:sumResponse" name="sumResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CalculatorServiceSoapBinding" type="tns:CalculatorWs">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="multiply">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="multiply">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="multiplyResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="sum">
      <soap:operation soapAction="" style="document"/>
      <wsdl:input name="sum">
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="sumResponse">
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
```

```
  <wsdl:service name="CalculatorService">
    <wsdl:port binding="tns:CalculatorServiceSoapBinding" name="CalculatorPort">
      <soap:address location="http://127.0.0.1:4204/Calculator?wsdl"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Accessing the @WebService with javax.xml.ws.Service

In our testcase we see how to create a client for our `Calculator` service via the `javax.xml.ws.Service` class and leveraging our `CalculatorWs` endpoint interface.

With this we can get an implementation of the interfacce generated dynamically for us that can be used to send compliant SOAP messages to our service.

```
import org.junit.BeforeClass;
import org.junit.Test;

import javax.ejb.embeddable.EJBContainer;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import java.net.URL;
import java.util.Properties;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;

public class CalculatorTest {

    @BeforeClass
    public static void setUp() throws Exception {
        Properties properties = new Properties();
        properties.setProperty("openejb.embedded.remotable", "true");
        //properties.setProperty("httpejbd.print", "true");
        //properties.setProperty("httpejbd.indent.xml", "true");
        EJBContainer.createEJBContainer(properties);
    }

    @Test
    public void test() throws Exception {
        Service calculatorService = Service.create(
                new URL("http://127.0.0.1:4204/Calculator?wsdl"),
                new QName("http://superbiz.org/wsdl", "CalculatorService"));

        assertNotNull(calculatorService);

        CalculatorWs calculator = calculatorService.getPort(CalculatorWs.class);
        assertEquals(10, calculator.sum(4, 6));
        assertEquals(12, calculator.multiply(3, 4));
    }
}
```

For easy testing we'll use the Embeddable EJBContainer API part of EJB 3.1 to boot CXF in our testcase. This will deploy our application in the embedded container and bring the web service online so we can invoke it.

Running

Running the example can be done from maven with a simple 'mvn clean install' command run from the 'simple-webservice' directory.

When run you should see output similar to the following.

```
```

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running org.superbiz.calculator.ws.CalculatorTest
INFO -
********************************************************************************
INFO - OpenEJB http://tomee.apache.org/
INFO - Startup: Sat Feb 18 19:11:50 PST 2012
INFO - Copyright 1999-2012 (C) Apache OpenEJB Project, All Rights Reserved.
INFO - Version: 4.0.0-beta-3
INFO - Build date: 20120218
INFO - Build time: 03:32
INFO -
********************************************************************************
INFO - openejb.home = /Users/dblevins/work/all/trunk/openejb/examples/simple-
webservice
INFO - openejb.base = /Users/dblevins/work/all/trunk/openejb/examples/simple-
webservice
INFO - Created new singletonService
org.apache.openejb.cdi.ThreadSingletonServiceImpl@16bdb503
INFO - succeeded in installing singleton service
INFO - Using 'javax.ejb.embeddable.EJBContainer=true'
INFO - Cannot find the configuration file [conf/openejb.xml].  Will attempt to create
one for the beans deployed.
INFO - Configuring Service(id=Default Security Service, type=SecurityService,
provider-id=Default Security Service)
INFO - Configuring Service(id=Default Transaction Manager, type=TransactionManager,
provider-id=Default Transaction Manager)
INFO - Creating TransactionManager(id=Default Transaction Manager)
INFO - Creating SecurityService(id=Default Security Service)
INFO - Beginning load: /Users/dblevins/work/all/trunk/openejb/examples/simple-
webservice/target/classes
INFO - Using 'openejb.embedded=true'
INFO - Configuring enterprise application:
/Users/dblevins/work/all/trunk/openejb/examples/simple-webservice
INFO - Auto-deploying ejb Calculator: EjbDeployment(deployment-id=Calculator)
INFO - Configuring Service(id=Default Stateless Container, type=Container, provider-
id=Default Stateless Container)
INFO - Auto-creating a container for bean Calculator: Container(type=STATELESS,
id=Default Stateless Container)
INFO - Creating Container(id=Default Stateless Container)
INFO - Configuring Service(id=Default Managed Container, type=Container, provider-
id=Default Managed Container)
INFO - Auto-creating a container for bean org.superbiz.calculator.ws.CalculatorTest:
Container(type=MANAGED, id=Default Managed Container)
INFO - Creating Container(id=Default Managed Container)
INFO - Using directory /var/folders/bd/f9ntqy1m8xj_fs006s6crtjh0000gn/T for stateful
session passivation
INFO - Enterprise application "/Users/dblevins/work/all/trunk/openejb/examples/simple-
webservice" loaded.
INFO - Assembling app: /Users/dblevins/work/all/trunk/openejb/examples/simple-
```

```
webservice
INFO - ignoreXmlConfiguration == true
INFO - ignoreXmlConfiguration == true
INFO - existing thread singleton service in SystemInstance()
org.apache.openejb.cdi.ThreadSingletonServiceImpl@16bdb503
INFO - OpenWebBeans Container is starting...
INFO - Adding OpenWebBeansPlugin : [CdiPlugin]
INFO - All injection points were validated successfully.
INFO - OpenWebBeans Container has started, it took [62] ms.
INFO - Created Ejb(deployment-id=Calculator, ejb-name=Calculator, container=Default
Stateless Container)
INFO - Started Ejb(deployment-id=Calculator, ejb-name=Calculator, container=Default
Stateless Container)
INFO - Deployed
Application(path=/Users/dblevins/work/all/trunk/openejb/examples/simple-webservice)
INFO - Initializing network services
INFO - can't find log4j MDC class
INFO - Creating ServerService(id=httpejbd)
INFO - Creating ServerService(id=cxf)
INFO - Creating ServerService(id=admin)
INFO - Creating ServerService(id=ejbd)
INFO - Creating ServerService(id=ejbds)
INFO - Initializing network services
INFO -   ** Starting Services **
INFO -   NAME                 IP              PORT
INFO -   httpejbd             127.0.0.1       4204
INFO - Creating Service {http://superbiz.org/wsdl}CalculatorService from class
org.superbiz.calculator.ws.CalculatorWs
INFO - Setting the server's publish address to be http://nopath:80
INFO - Webservice(wsdl=http://127.0.0.1:4204/Calculator,
qname={http://superbiz.org/wsdl}CalculatorService) --> Ejb(id=Calculator)
INFO -   admin thread         127.0.0.1       4200
INFO -   ejbd                 127.0.0.1       4201
INFO -   ejbd                 127.0.0.1       4203
INFO - -------
INFO - Ready!
INFO - Creating Service {http://superbiz.org/wsdl}CalculatorService from WSDL:
http://127.0.0.1:4204/Calculator?wsdl
INFO - Creating Service {http://superbiz.org/wsdl}CalculatorService from WSDL:
http://127.0.0.1:4204/Calculator?wsdl
INFO - Default SAAJ universe not set
INFO - TX NotSupported: Suspended transaction null
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.584 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

Inspecting the messages

The above test case will result in the following SOAP messages being sent between the clien and server.

sum(int, int)

Request SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:sum xmlns:ns1="http://superbiz.org/wsdl">
      <arg0>4</arg0>
      <arg1>6</arg1>
    </ns1:sum>
  </soap:Body>
</soap:Envelope>
```

Response SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:sumResponse xmlns:ns1="http://superbiz.org/wsdl">
      <return>10</return>
    </ns1:sumResponse>
  </soap:Body>
</soap:Envelope>
```

multiply(int, int)

Request SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:multiply xmlns:ns1="http://superbiz.org/wsdl">
      <arg0>3</arg0>
      <arg1>4</arg1>
    </ns1:multiply>
  </soap:Body>
</soap:Envelope>
```

Response SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns1:multiplyResponse xmlns:ns1="http://superbiz.org/wsdl">
      <return>12</return>
    </ns1:multiplyResponse>
  </soap:Body>
</soap:Envelope>
```

Inside the jar

With so much going on it can make things look more complex than they are. It can be hard to believe that so much can happen with such little code. That's the benefit of having an app server.

If we look at the jar built by maven, we'll see the application itself is quite small:

```
$ jar tvf target/simple-webservice-1.1.0-SNAPSHOT.jar
     0 Sat Feb 18 19:17:06 PST 2012 META-INF/
   127 Sat Feb 18 19:17:04 PST 2012 META-INF/MANIFEST.MF
     0 Sat Feb 18 19:17:02 PST 2012 org/
     0 Sat Feb 18 19:17:02 PST 2012 org/superbiz/
     0 Sat Feb 18 19:17:02 PST 2012 org/superbiz/calculator/
     0 Sat Feb 18 19:17:02 PST 2012 org/superbiz/calculator/ws/
   855 Sat Feb 18 19:17:02 PST 2012 org/superbiz/calculator/ws/Calculator.class
   288 Sat Feb 18 19:17:02 PST 2012 org/superbiz/calculator/ws/CalculatorWs.class
```

This single jar could be deployed any any compliant Java EE implementation. In TomEE you'd simply place it in the `tomee.home/webapps/` directory. No war file necessary. If you did want to create a war, you'd simply place the jar in the `WEB-INF/lib/` directory of the war.

The server already contains the right libraries to run the code, such as Apache CXF, so no need to include anything extra beyond your own application code.