L'api JAX WS 2.0

TP N°4: Utilisation de l'api JAX-WS 2.0

SOMMAIRE

I- Pré-requis :

- Eclipse Mars;
- JDK 1.7;
- Tomcat 7.0 ;
- JAX-WS RI 2.2 (Projet Metro de Glassfish).

I- Objectifs :

- ✓ Utiliser le style RPC ;
- ✓ Utiliser le style Document ;
- ✓ Utiliser la commande wsimport (générer le stub) ;
- ✓ Utiliser la commande wsgen (générer le skeleton) ;
- ✓ Développer un SW en attachant un fichier binaire avec la technique MTOM ;
- ✓ Développer un intercepteur (ou Handler) ;
- ✓ Déployer un SW dans une application WEB.

II- Développement de l'application

Partie 1 : Développer un SW avec JAX-WS en utilisant le style RPC

- Créer un projet java, par exemple (tpjaxwsrpc).
- Créer l'interface ICalculator suivante :

```
package ma.formation.cigma;

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

@WebService
@SOAPBinding(style = Style.RPC)
public interface ICalculator {
	@WebMethod
	public double add(double a, double b);

}
```

- Créer la classe CalculatorImpl suivante :

```
package ma.formation.cigma;

import javax.jws.WebService;

@WebService(endpointInterface = "ma.formation.cigma.ICalculator")
public class CalculatorImpl implements ICalculator {
	@Override
	public double add(double a, double b) {
		return a + b;
	}
}
```

- Créer un "Endpoint Publisher" CalculatorPublisher suivante :

```
package ma.formation.cigma.publisher;

import javax.xml.ws.Endpoint;

import ma.formation.cigma.CalculatorImpl;

public class CalculatorPublisher {
	public static void main(String[] args) {
	Endpoint.publish("http://localhost:9999/cigma/calculator", new CalculatorImpl());
	}
}
```

- Exécuter la méthode main ci-dessus.
- Pour tester le SW, lancer le lien suivant : http://localhost:9999/cigma/calculator?wsdl
  Le fichier WSDL généré par JAW-WS est le suivant :

```
<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
://cigma.formation.ma/"xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoa
p.org/wsdl/" targetNamespace="http://cigma.formation.ma/" name="CalculatorImplService">
<types/>
<message name="add">
<part name="arg0" type="xsd:double"/>
<part name="arg1" type="xsd:double"/>
</message>
<message name="addResponse">
<part name="return" type="xsd:double"/>
</message>
<portType name="ICalculator">
<operation name="add" parameterOrder="arg0 arg1">
<input wsam:Action="http://cigma.formation.ma/ICalculator/addRequest" message="tns:add"/>
<output wsam:Action="http://cigma.formation.ma/ICalculator/addResponse" message="tns:addResponse
"/>
</operation>
</portType>
<binding name="CalculatorImplPortBinding" type="tns:ICalculator">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
<operation name="add">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://cigma.formation.ma/"/>
</input>
<output>
<soap:body use="literal" namespace="http://cigma.formation.ma/"/>
</output>
</operation>
</binding>
<service name="CalculatorImplService">
<port name="CalculatorImplPort" binding="tns:CalculatorImplPortBinding">
<soap:address location="http://localhost:9999/cigma/calculator"/>
</port>
</service>
</definitions>
```

- Créer un SW client manuellement :

Créer la classe CalculatorClient suivante :

```
package ma.formation.cigma.client;

import java.net.MalformedURLException;
import java.net.URL;

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

import ma.formation.cigma.ICalculator;

public class CalculatorClient {

	public static void main(String[] args) {
	   try {
	       URL url = new URL("http://localhost:9999/cigma/calculator?wsdl");
	       // Le premier argument est le service URI
	       // Le deuxième argument est le nom du service
	       QName qname = new QName("http://cigma.formation.ma/", "CalculatorImplService");
	              Service service = Service.create(url, qname);
	              ICalculator calculator = service.getPort(ICalculator.class);
	              System.out.println(calculator.add(10d, 20d));
	       } catch (MalformedURLException e) {
	              e.printStackTrace();
	       }
	     }
	 }
```

- Créer automatiquement le client en utilisant wsimport :

- Créer un projet java, par exemple ***tpjaxwsrpcclientaut.***
- Pour générer les Stubs, lancer la commande DOS suivante :

```
wsimport -keep http://localhost:9999/cigma/calculator?wsdl
```

```
C:\swtps\tpjaxwsrpcclientaut\src>wsimport -keep http://localhost:9999/cigma/calc
ulator?wsdl
```

L'interface suivante ICalculator est créée :

```
package ma.formation.cigma;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.xml.ws.Action;
```

```java
/**
 * This class was generated by the JAX-WS RI.
 * JAX-WS RI 2.2.4-b01
 * Generated source version: 2.2
 *
 */
@WebService(name = "ICalculator", targetNamespace = "http://cigma.formation.ma/")
@SOAPBinding(style = SOAPBinding.Style.RPC)
public interface ICalculator {


    /**
     *
     * @param arg1
     * @param arg0
     * @return
     *     returns double
     */
    @WebMethod
    @WebResult(partName = "return")
    @Action(input = "http://cigma.formation.ma/ICalculator/addRequest", output =
"http://cigma.formation.ma/ICalculator/addResponse")
    public double add(
        @WebParam(name = "arg0", partName = "arg0")
        double arg0,
        @WebParam(name = "arg1", partName = "arg1")
        double arg1);
}
```

La classe suivante est créée :

```java
package ma.formation.cigma;

import java.net.MalformedURLException;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import javax.xml.ws.WebEndpoint;
import javax.xml.ws.WebServiceClient;
import javax.xml.ws.WebServiceException;
import javax.xml.ws.WebServiceFeature;


/**
 * This class was generated by the JAX-WS RI.
 * JAX-WS RI 2.2.4-b01
 * Generated source version: 2.2
 *
 */
@WebServiceClient(name = "CalculatorImplService", targetNamespace =
"http://cigma.formation.ma/", wsdlLocation = "http://localhost:9999/cigma/calculator?wsdl")
public class CalculatorImplService
    extends Service
{
```

```
    private final static URL CALCULATORIMPLSERVICE_WSDL_LOCATION;
    private final static WebServiceException CALCULATORIMPLSERVICE_EXCEPTION;
    private final static QName CALCULATORIMPLSERVICE_QNAME = new
QName("http://cigma.formation.ma/", "CalculatorImplService");

    static {
        URL url = null;
        WebServiceException e = null;
        try {
            url = new URL("http://localhost:9999/cigma/calculator?wsdl");
        } catch (MalformedURLException ex) {
            e = new WebServiceException(ex);
        }
        CALCULATORIMPLSERVICE_WSDL_LOCATION = url;
        CALCULATORIMPLSERVICE_EXCEPTION = e;
    }

    public CalculatorImplService() {
        super(__getWsdlLocation(), CALCULATORIMPLSERVICE_QNAME);
    }

    public CalculatorImplService(WebServiceFeature... features) {
        super(__getWsdlLocation(), CALCULATORIMPLSERVICE_QNAME, features);
    }

    public CalculatorImplService(URL wsdlLocation) {
        super(wsdlLocation, CALCULATORIMPLSERVICE_QNAME);
    }

    public CalculatorImplService(URL wsdlLocation, WebServiceFeature... features) {
        super(wsdlLocation, CALCULATORIMPLSERVICE_QNAME, features);
    }

    public CalculatorImplService(URL wsdlLocation, QName serviceName) {
        super(wsdlLocation, serviceName);
    }

    public CalculatorImplService(URL wsdlLocation, QName serviceName, WebServiceFeature...
features) {
        super(wsdlLocation, serviceName, features);
    }

    /**
     *
     * @return
     *     returns ICalculator
     */
    @WebEndpoint(name = "CalculatorImplPort")
    public ICalculator getCalculatorImplPort() {
        return super.getPort(new QName("http://cigma.formation.ma/", "CalculatorImplPort"),
ICalculator.class);
    }

    /**
     *
     * @param features
     *     A list of {@link javax.xml.ws.WebServiceFeature} to configure on the proxy.
Supported features not in the <code>features</code> parameter will have their default
values.
```

```
     * @return
     *     returns ICalculator
     */
    @WebEndpoint(name = "CalculatorImplPort")
    public ICalculator getCalculatorImplPort(WebServiceFeature... features) {
        return super.getPort(new QName("http://cigma.formation.ma/", "CalculatorImplPort"),
ICalculator.class, features);
    }

    private static URL __getWsdlLocation() {
        if (CALCULATORIMPLSERVICE_EXCEPTION!= null) {
            throw CALCULATORIMPLSERVICE_EXCEPTION;
        }
        return CALCULATORIMPLSERVICE_WSDL_LOCATION;
    }
}
```

- Maintenant, créer la classe Test suivante :

```
package ma.formation.cigma.client;

import ma.formation.cigma.CalculatorImplService;
import ma.formation.cigma.ICalculator;

public class Test {

	public static void main(String[] args) {
		CalculatorImplService calculatorService = new CalculatorImplService();
		ICalculator calculator = calculatorService.getCalculatorImplPort();

		System.out.println(calculator.add(14d, 25d));
	}
}
```

Partie 2 : Développer un SW avec JAX-WS en utilisant le style Document

- Créer un projet JAVA, par exemple tpjaxwsdocument.
- Créer l'interface `ICalculator` suivante :

```
package ma.formation.cigma;

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

@WebService
@SOAPBinding(style = Style.DOCUMENT,use=Use.LITERAL)//Optionnel
public interface ICalculator {
	@WebMethod
	public double add(double a, double b);
}
```

Dans JAX-WS, pour convertir un service web RPC à un service web Document, il suffit de préciser le style DOCUMENT au niveau de l'annotation @SOAPBinding.

- Créer la classe `CalculatorImpl suivante :`

```
package ma.formation.cigma;

import javax.jws.WebService;

@WebService(endpointInterface = "ma.formation.cigma.ICalculator")
public class CalculatorImpl implements ICalculator {

	@Override
	public double add(double a, double b) {
		return a + b;
	}
}
```

- Créer la classe `CalculatorPublisher suivante :`

```
package ma.formation.cigma.publisher;

import javax.xml.ws.Endpoint;

import ma.formation.cigma.CalculatorImpl;

public class CalculatorPublisher {
	public static void main(String[] args) {
		Endpoint.publish("http://localhost:9999/cigma/calculator", new CalculatorImpl());
	}
}
```

- Créer un projet JAVA, par exemple tpjaxwsdocumentclient.
- Lancer la commande suivant :

```
C:\swtps\tpjaxwsdocumentclient\src>wsimport -keep http://localhost:9999/cigma/calculator?wsdl
parsing WSDL...



Generating code...


Compiling code...
```

Vérifier que les classes stubs suivantes ont été créés :

- Add.java
- AddResponse.java
- CalculatorImplService.java
- ICalculator.java
- ObjectFactory.java
- package-info.java

- Créer la classe Test suivante :

```java
package ma.formation.cigma.client;

import ma.formation.cigma.CalculatorImplService;
import ma.formation.cigma.ICalculator;

public class Test {

	public static void main(String[] args) {
		CalculatorImplService calculatorService = new CalculatorImplService();
		ICalculator calculator = calculatorService.getCalculatorImplPort();
		System.out.println(calculator.add(10d, 15d));
	}
}
```

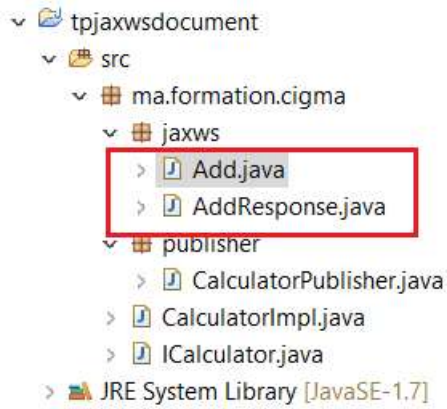
Utilisation de la commande wsgen :

- Lancer la commande wsgen pour générer les skeletons (Les classes de Mapping avec JAX-B, le fichier wsdl et les schémas XSD) :

```
Cd C:\swtps\tpjaxwsdocument\bin
wsgen -keep -verbose -cp . ma.formation.cigma.CalculatorImpl -d C:\swtps\tpjaxwsdocument\src
```

```
C:\swtps\tpjaxwsdocument\bin>wsgen -keep -verbose -cp . ma.formation.cigma.CalculatorImpl -d C:\swtps\tpjaxwsdo
cument\src
```

Vérifier que les deux classes suivantes sont créés :



La classe Add :

```java
package ma.formation.cigma.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
```

```java
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "add", namespace = "http://cigma.formation.ma/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "add", namespace = "http://cigma.formation.ma/", propOrder = {
    "arg0",
    "arg1"
})
public class Add {

    @XmlElement(name = "arg0", namespace = "")
    private double arg0;
    @XmlElement(name = "arg1", namespace = "")
    private double arg1;

    /**
     *
     * @return
     *     returns double
     */
    public double getArg0() {
        return this.arg0;
    }

    /**
     *
     * @param arg0
     *     the value for the arg0 property
     */
    public void setArg0(double arg0) {
        this.arg0 = arg0;
    }

    /**
     *
     * @return
     *     returns double
     */
    public double getArg1() {
        return this.arg1;
    }

    /**
     *
     * @param arg1
     *     the value for the arg1 property
     */
    public void setArg1(double arg1) {
        this.arg1 = arg1;
    }

}
```

La classe AddResponse suivante :

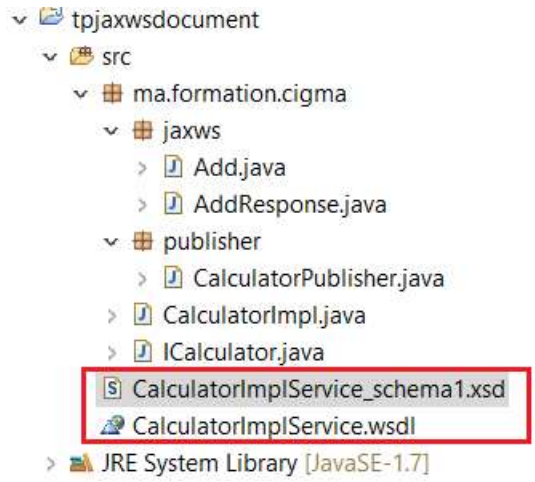
```
package ma.formation.cigma.jaxws;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement(name = "addResponse", namespace = "http://cigma.formation.ma/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addResponse", namespace = "http://cigma.formation.ma/")
public class AddResponse {

    @XmlElement(name = "return", namespace = "")
    private double _return;

    /**
     *
     * @return
     *     returns double
     */
    public double getReturn() {
        return this._return;
    }

    /**
     *
     * @param _return
     *     the value for the _return property
     */
    public void setReturn(double _return) {
        this._return = _return;
    }

}
```

- Pour générer le fichier WSDL lancer la commande suivante :

Cd C:\swtps\tpjaxwsdocument\bin

wsgen -keep -verbose -cp . ma.formation.cigma.CalculatorImpl -d C:\swtps\tpjaxwsdocument\src -wsdl

Le fichier xsd et le fichier WSDL suivants seront crées :



Le contenu du fichier XSD est le suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" targetNamespace="http://cigma.formation.ma/"
xmlns:tns="http://cigma.formation.ma/" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="add" type="tns:add"/>

  <xs:element name="addResponse" type="tns:addResponse"/>

  <xs:complexType name="add">
    <xs:sequence>
      <xs:element name="arg0" type="xs:double"/>
      <xs:element name="arg1" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="addResponse">
    <xs:sequence>
      <xs:element name="return" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Le contenu du fichier WSDL est le suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-
b01. -->
<definitions targetNamespace="http://cigma.formation.ma/" name="CalculatorImplService"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:tns="http://cigma.formation.ma/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://cigma.formation.ma/"
```

```
schemaLocation="CalculatorImplService_schema1.xsd"/>
    </xsd:schema>
  </types>
  <message name="add">
    <part name="parameters" element="tns:add"/>
  </message>
  <message name="addResponse">
    <part name="parameters" element="tns:addResponse"/>
  </message>
  <portType name="ICalculator">
    <operation name="add">
      <input wsam:Action="http://cigma.formation.ma/ICalculator/addRequest"
message="tns:add"/>
      <output wsam:Action="http://cigma.formation.ma/ICalculator/addResponse"
message="tns:addResponse"/>
    </operation>
  </portType>
  <binding name="CalculatorImplPortBinding" type="tns:ICalculator">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="add">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="CalculatorImplService">
    <port name="CalculatorImplPort" binding="tns:CalculatorImplPortBinding">
      <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
    </port>
  </service>
</definitions>
```

Partie 3 : Développer un SW en attachant un fichier binaire avec MTOM

Dans cette partie, nous allons voir comment utiliser les techniques **Message Transmission Optimization Mechanism (MTOM)** et **XML-Binary Optimized Packaging (XOP)** pour transmettre un fichier binaire en pièce attachée du client vers le serveur et du serveur vers le client.

- Créer un projet JAVA, par exemple tpjaxwsrpcwithattachement.
- Développer l'interface ImageServer suivante :

```
package ma.formation.cigma;

import java.awt.Image;

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

@WebService
```

```java
@SOAPBinding(style = Style.RPC)
public interface ImageServer {
	// download a image from server
	@WebMethod
	Image downloadImage(String name);

	// update image to server
	@WebMethod String uploadImage(Image data);

}
```

- Développer la classe ImageServerImpl suivante :

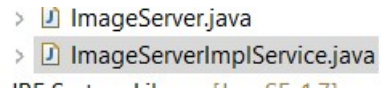
```java
package ma.formation.cigma;

import java.awt.Image;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.jws.WebService;
import javax.xml.ws.WebServiceException;
import javax.xml.ws.soap.MTOM;

@MTOM
@WebService(endpointInterface = "ma.formation.cigma.ImageServer")
public class ImageServerImpl implements ImageServer {
	static final String IMAGE_PATH="C:\\swtps\\tpjaxwsrpcwithattachement\\images\\";


	@Override
	public Image downloadImage(String name) {
		try {
			File image = new File(IMAGE_PATH + name);
			return ImageIO.read(image);
		} catch (IOException e) {
			e.printStackTrace();
			return null;
		}
	}

	@Override
	public String uploadImage(Image data) {
		if (data != null) {
			return "Upload Successful";
		}
		throw new WebServiceException("Upload Failed!");
	}
}
```

Dans la classe ci-dessus, les images existent dans le dossier :
`C:\\swtps\\tpjaxwsrpcwithattachement\\images\\";`
Copier dans ce dossier quelques images pour faire le test.

- Créer la classe ImagePublisher suivante:

```java
package ma.formation.cigma.publisher;

import javax.xml.ws.Endpoint;

import ma.formation.cigma.ImageServerImpl;

public class ImagePublisher {
	public static void main(String[] args) {
		Endpoint.publish("http://localhost:9999/cigma/image", new ImageServerImpl());
	}
}
```

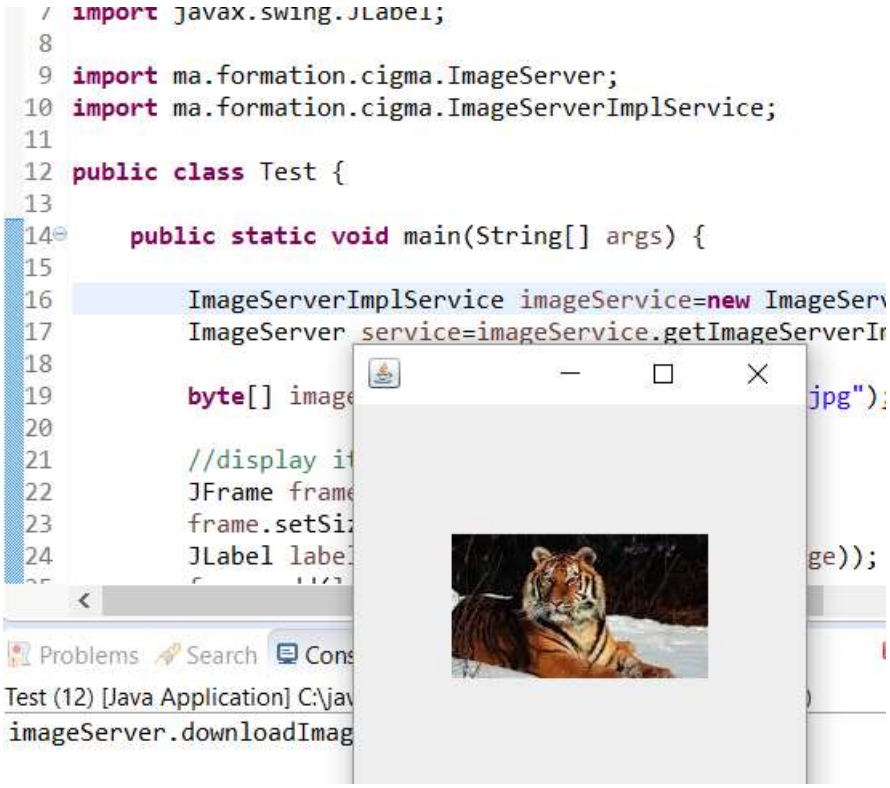
- Créer un projet JAVA, par exemple tpjaxwsrpcattachementclient.
- Lancer la commande suivante : ***wsimport -keep http://localhost:9999/cigma/image?wsdl.***
  Les deux classes suivantes seront créées :



- Créer la classe de test suivante :

```java
package ma.formation.cigma.client;

import java.awt.Image;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

import ma.formation.cigma.ImageServer;
import ma.formation.cigma.ImageServerImplService;

public class Test {

	public static void main(String[] args) {

		ImageServerImplService imageService=new ImageServerImplService();
		ImageServer service=imageService.getImageServerImplPort();

		byte[] image=service.downloadImage("image4.jpg");

		//display it in frame
	  JFrame frame = new JFrame();
	  frame.setSize(300, 300);
	  JLabel label = new JLabel(new ImageIcon(image));
	  frame.add(label);
	  frame.setVisible(true);

	  System.out.println("imageServer.downloadImage() : Download Successful!");
	}
}
```

- Exécuter la méthode main ci-dessus. L'image (image4.jpg) dans cette exemple sera affichée :



- Utiliser TCP/IP Monitor pour visualiser les requêtes/réponses SOAP :

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
	<S:Body>
		<ns2:downloadImage xmlns:ns2="http://cigma.formation.ma/">
			<arg0>image4.jpg</arg0>
		</ns2:downloadImage>
	</S:Body>
</S:Envelope>
```

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
	<S:Body>
		<ns2:downloadImageResponse xmlns:ns2="http://cigma.formation.ma/">
			<return>
				<xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
					href="cid:9de77a89-65f5-4126-8313-
e595a9635bc1@example.jaxws.sun.com"></xop:Include>
			</return>
		</ns2:downloadImageResponse>
	</S:Body>
</S:Envelope>
```

- **Activation de MTOM au niveau du client :**

Créer la classe de test suivante :

```
package ma.formation.cigma.client;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.soap.SOAPBinding;
import ma.formation.cigma.ImageServer;
import ma.formation.cigma.ImageServerImplService;

public class Test2 {

	public static void main(String[] args) {
		String path="C:\\swtps\\tpjaxwsrpcwithattachement\\images\\";
		try {
			ImageServerImplService imageService = new ImageServerImplService();
			ImageServer service = imageService.getImageServerImplPort();
			/************ test upload ***************/
			File imgUpload = new File(path+"newimage.jpg");
			FileInputStream fis=new FileInputStream(imgUpload);
			byte[] data = new byte[2000];
			fis.read(data);
			fis.close();
			// enable MTOM in client
			BindingProvider bp = (BindingProvider) service;
			SOAPBinding binding = (SOAPBinding) bp.getBinding();
			binding.setMTOMEnabled(true);
			String status = service.uploadImage(data);
			System.out.println("imageServer.uploadImage() : " + status);
			System.out.println("imageServer.downloadImage() : Download
Successful!");
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

- Les requêtes/réponses SOAP échangées entre le client et le serveur sont les suivantes :

```
<?xml version="1.0" ?>
GET /cigma/image?wsdl HTTP/1.1
User-Agent: Java/1.7.0_75
Host: localhost:8888
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

POST /cigma/image HTTP/1.1
Accept: text/xml, multipart/related
Content-Type: multipart/related;start="<rootpart*0ef4e8c2-7c48-4b11-8216-
2695aa7ad102@example.jaxws.sun.com>";type="application/xop+xml";boundary="uuid:0ef4e8c2-
7c48-4b11-8216-2695aa7ad102";start-info="text/xml"
SOAPAction: "http://cigma.formation.ma/ImageServer/uploadImageRequest"
User-Agent: JAX-WS RI 2.2.4-b01
Host: localhost:8888
Connection: keep-alive
```

```
Content-Length: 2825

--uuid:0ef4e8c2-7c48-4b11-8216-2695aa7ad102
Content-Id: <rootpart*0ef4e8c2-7c48-4b11-8216-2695aa7ad102@example.jaxws.sun.com>
Content-Type: application/xop+xml;charset=utf-8;type="text/xml"
Content-Transfer-Encoding: binary

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
            <ns2:uploadImage xmlns:ns2="http://cigma.formation.ma/">
                  <arg0>
                        <xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
                              href="cid:c704cbe4-96e1-4022-b072-
22c962912a24@example.jaxws.sun.com"></xop:Include>
                  </arg0>
            </ns2:uploadImage>
      </S:Body>
</S:Envelope>

--uuid:0ef4e8c2-7c48-4b11-8216-2695aa7ad102
Content-Id: <c704cbe4-96e1-4022-b072-22c962912a24@example.jaxws.sun.com>
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary

Les données binaires sont ici.
```

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
            <ns2:uploadImageResponse xmlns:ns2="http://cigma.formation.ma/">
                  <return>Upload Successful</return>
            </ns2:uploadImageResponse>
      </S:Body>
</S:Envelope>
```

Partie 4 : Développer un intercepteur

Les intercepteurs (ou handler) permettent d'intercepter les messages SOAP (requête ou réponse) et offrent la possibilité de modifier leurs contenues. Par exemple, développer un intercepteur du côté client qui attache à l'entête du message SOAP (header) le compte d'un utilisateur (login et password) et au niveau serveur développer un intercepteur qui vérifie ce compte. Si ce compte est valide, le service web sera accessible.

Développer un intercepteur du côté serveur :

- Développer un projet JAVA, par exemple tpjaxwshandler.
- Développer l'interface suivante :

```
package ma.formation.cigma;

import javax.jws.WebMethod;
import javax.jws.WebService;
```

```java
@WebService
public interface ICalculator {
	@WebMethod
	public double add(double a, double b);
}
```

- Créer la classe suivante :

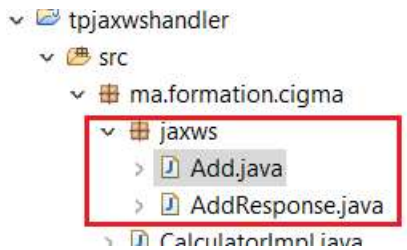
```java
package ma.formation.cigma;

import javax.jws.WebService;

@WebService(endpointInterface = "ma.formation.cigma.ICalculator")
public class CalculatorImpl implements ICalculator {

	@Override
	public double add(double a, double b) {
		return a + b;
	}
}
```

- Lancer la commande wsgen

Cd C:\swtps\tpjaxwshandler\bin

wsgen -keep -verbose -cp . ma.formation.cigma.CalculatorImpl -d C:\swtps\tpjaxwshandler\src

Les deux classes suivante seront crées :



- Développer l'intercepteur suivant :

```java
package ma.formation.cigma.handler;

import java.io.IOException;
import java.util.Iterator;
import java.util.Set;

import javax.xml.namespace.QName;
import javax.xml.soap.Node;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPFault;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
```

```
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;
import javax.xml.ws.soap.SOAPFaultException;

public class UserValidatorHandler implements SOAPHandler<SOAPMessageContext> {
	@Override
	public boolean handleMessage(SOAPMessageContext context) {
		System.out.println("Server : handleMessage()......");
		Boolean isRequest = (Boolean)
context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
		// for response message only, true for outbound messages, false for inbound
		if (!isRequest) {
			try {
				SOAPMessage soapMsg = context.getMessage();
				SOAPEnvelope soapEnv = soapMsg.getSOAPPart().getEnvelope();
				SOAPHeader soapHeader = soapEnv.getHeader();
				// if no header, add one
				if (soapHeader == null) {
					soapHeader = soapEnv.addHeader();
					// throw exception
					generateSOAPErrMessage(soapMsg, "No SOAP header.");
				}
				// Get user a count from SOAP header
				Iterator it =
soapHeader.extractHeaderElements(SOAPConstants.URI_SOAP_ACTOR_NEXT);
				// if no header block for next actor found? throw exception
				if (it == null || !it.hasNext()) {
					generateSOAPErrMessage(soapMsg, "No header block for next
actor.");
				}
				// if no compte found? throw exception
				Node userNode = (Node) it.next();
				String userValue = (userNode == null) ? null :
userNode.getValue();
				if (userValue == null) {
					generateSOAPErrMessage(soapMsg, "No compte in header
block.");
				}
				// if user is not match, throw exception
				if (!userValue.equals("admin001")) {
					generateSOAPErrMessage(soapMsg, "Invalid count, access is
denied.");
				}
				// tracking
				soapMsg.writeTo(System.out);
			} catch (SOAPException e) {
				System.err.println(e);
			} catch (IOException e) {
				System.err.println(e);
			}
		}
		// continue other handler chain
		return true;
	}

	@Override
	public boolean handleFault(SOAPMessageContext context) {
		System.out.println("Server : handleFault()......");
```

```
            return true;
      }

      @Override
      public void close(MessageContext context) {
            System.out.println("Server : close()......");
      }

      @Override
      public Set<QName> getHeaders() {
            System.out.println("Server : getHeaders()......");
            return null;
      }

      private void generateSOAPErrMessage(SOAPMessage msg, String reason) {
            try {
                  SOAPBody soapBody = msg.getSOAPPart().getEnvelope().getBody();
                  SOAPFault soapFault = soapBody.addFault();
                  soapFault.setFaultString(reason);
                  throw new SOAPFaultException(soapFault);
            } catch (SOAPException e) {
            }
      }
}
```

- Développer le fichier xml : handler-chain.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<javaee:handler-chains
     xmlns:javaee="http://java.sun.com/xml/ns/javaee"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <javaee:handler-chain>
    <javaee:handler>
<javaee:handler-class>ma.formation.cigma.handler.UserValidatorHandler</javaee:handler-class>
    </javaee:handler>
  </javaee:handler-chain>
</javaee:handler-chains>
```

- Attacher cet intercepteur au WS :

```
package ma.formation.cigma;

import javax.jws.HandlerChain;
import javax.jws.WebService;

@WebService(endpointInterface = "ma.formation.cigma.ICalculator")
@HandlerChain(file="handler-chain.xml")
public class CalculatorImpl implements ICalculator {

      @Override
      public double add(double a, double b) {
            return a + b;
      }
}
```

- Créer la classe Publisher suivante :

```
package ma.formation.cigma.publisher;

import javax.xml.ws.Endpoint;

import ma.formation.cigma.CalculatorImpl;

public class CalculatorPublisher {
	public static void main(String[] args) {
		Endpoint.publish("http://localhost:9999/cigma/calculator", new CalculatorImpl());
	}
}
```

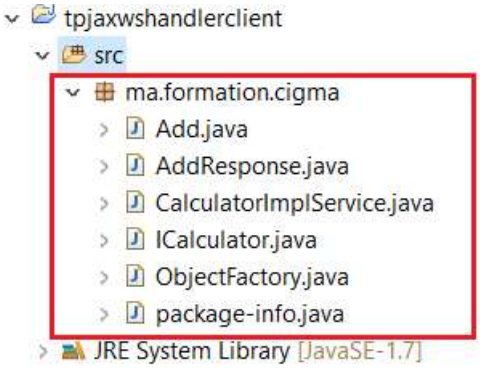
- Lancer la méthode main ci-dessus.

Développement du Handler du côté client :

- Créer un nouveau projet java, par exemple tpjaxwshandlerclient.
- Lancer la commande suivante :

  wsimport -keep -verbose http://localhost:9999/cigma/calculator?wsdl

  Les stubs suivants seront crées :



- Créer la classe Test suivante :

```
package ma.formation.cigma.client;

import ma.formation.cigma.CalculatorImplService;
import ma.formation.cigma.ICalculator;

public class Test {

	public static void main(String[] args) {
		CalculatorImplService calcService = new CalculatorImplService();
		ICalculator c = calcService.getCalculatorImplPort();
		System.out.println(c.add(14, 50));
	}
}
```

- Créer le Handler suivant :

```
package ma.formation.cigma.handler;

import java.io.IOException;
import java.util.Set;

import javax.xml.namespace.QName;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPHeaderElement;
import javax.xml.soap.SOAPMessage;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;

public class UserInjectHandler implements SOAPHandler<SOAPMessageContext> {
	static String userName = "admin001";
	@Override
	public boolean handleMessage(SOAPMessageContext context) {
		System.out.println("Client : handleMessage()......");
		Boolean isRequest = (Boolean)
context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
		// if this is a request, true for outbound messages, false for inbound
		if (isRequest) {
			try {
				SOAPMessage soapMsg = context.getMessage();
				SOAPEnvelope soapEnv = soapMsg.getSOAPPart().getEnvelope();
				SOAPHeader soapHeader = soapEnv.getHeader();
				// if no header, add one
				if (soapHeader == null) {
					soapHeader = soapEnv.addHeader();
				}
				// get mac address
				// add a soap header, name as "userName"
				QName qname = new QName("http://ws.mkyong.com/", "userName");
				SOAPHeaderElement soapHeaderElement =
soapHeader.addHeaderElement(qname);
				soapHeaderElement.setActor(SOAPConstants.URI_SOAP_ACTOR_NEXT);
				soapHeaderElement.addTextNode(userName);
				soapMsg.saveChanges();
				// tracking
				soapMsg.writeTo(System.out);
			} catch (SOAPException e) {
				System.err.println(e);
			} catch (IOException e) {
				System.err.println(e);
			}
		}
		// continue other handler chain
		return true;
	}

	@Override
	public boolean handleFault(SOAPMessageContext context) {
		System.out.println("Client : handleFault()......");
```

```
            return true;
      }

      @Override
      public void close(MessageContext context) {
            System.out.println("Client : close()......");
      }

      @Override
      public Set<QName> getHeaders() {
            System.out.println("Client : getHeaders()......");
            return null;
      }
}
```

- Créer le fichier handler-chain.xml suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<javaee:handler-chains xmlns:javaee="http://java.sun.com/xml/ns/javaee"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <javaee:handler-chain>
            <javaee:handler>
                  <javaee:handler-class>ma.formation.cigma.handler.UserInjectHandler
                  </javaee:handler-class>
            </javaee:handler>
      </javaee:handler-chain>
</javaee:handler-chains>
```

- Attacher ce handler au service web client, comme suit :

```
 *
 */
@WebServiceClient(name = "CalculatorImplService", targetNam
@HandlerChain(file="handler-chain.xml")
public class CalculatorImplService
    extends Service
{

```

- Tester avec un compte valide (userName=admin001). Pour ceci, au niveau du classe handler, attribuer à la variable static userName la valeur admin001 et lancer la classe Test. Le résultat est le suivant :

```
Server : handleMessage()......
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Header/><S:Body><ns2:add
xmlns:ns2="http://cigma.formation.ma/"><arg0>14.0</arg0><arg1>50.0</arg1></ns2:add></S:Body>
</S:Envelope>Server : handleMessage()......
Server : close()......
```

- Tester avec un compte invalide (userName=compteerrone). Pour ceci, au niveau du classe handler, attribuer à la variable static userName la valeur compteerrone et lancer la classe Test. Le résultat est le suivant :

```
Client : getHeaders()......
Client : handleMessage()......
```

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Header><userName
xmlns="http://ws.mkyong.com/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-
ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">compteerrone</userName></S:Heade
r><S:Body><ns2:add
xmlns:ns2="http://cigma.formation.ma/"><arg0>14.0</arg0><arg1>50.0</arg1></ns2:add></S:
Body></S:Envelope>Client : handleFault()......
Client : close()......
Exception in thread "main" javax.xml.ws.soap.SOAPFaultException: Invalid count, access
is denied.
      at com.sun.xml.internal.ws.fault.SOAP11Fault.getProtocolException(Unknown Source)
      at com.sun.xml.internal.ws.fault.SOAPFaultBuilder.createException(Unknown Source)
      at com.sun.xml.internal.ws.client.sei.SyncMethodHandler.invoke(Unknown Source)
      at com.sun.xml.internal.ws.client.sei.SyncMethodHandler.invoke(Unknown Source)
      at com.sun.xml.internal.ws.client.sei.SEIStub.invoke(Unknown Source)
      at com.sun.proxy.$Proxy21.add(Unknown Source)
      at ma.formation.cigma.client.Test.main(Test.java:11)
```

La requête SOAP est la suivante :

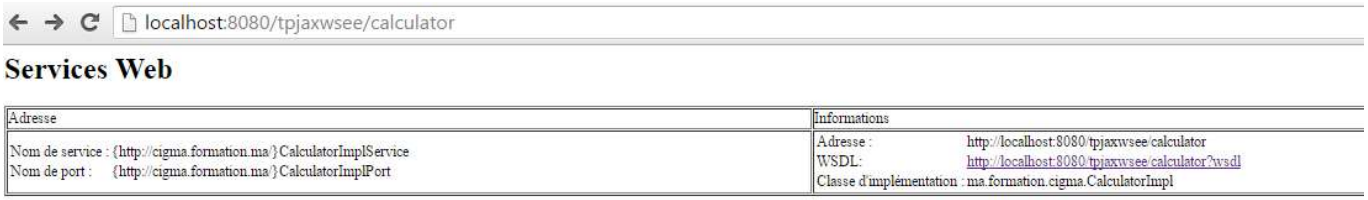
```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Header>
            <userName xmlns="http://ws.mkyong.com/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
      SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">compteerrone</userName>
      </S:Header>
      <S:Body>
            <ns2:add xmlns:ns2="http://cigma.formation.ma/">
                  <arg0>14.0</arg0>
                  <arg1>50.0</arg1>
            </ns2:add>
      </S:Body>
</S:Envelope>
```

Partie 5 : Intégrer un SW avec JAX-WS dans une application WEB

- Développer un projet "Dynamic Web Project", par exemple tpjaxwsee.
- Développer le même service web CalculatorImpl (voir partie 2).
- Créer le fichier *sun-jaxws.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime"
      version="2.0">
      <endpoint name="CalculatorWS" implementation="ma.formation.cigma.CalculatorImpl"
            url-pattern="/calculator" />
</endpoints>
```

- Copier les librairies de l'implémentation de JAX-WS (JAX-WS RI 2.2) dans le dossier WEB-INF/lib et redémarrer le serveur.
- Lancer le lien http://localhost:8080/tpjaxwsee/calculator. La page suivante sera affichée :


localhost:8080/tpjaxwsee/calculator
Services Web
Adresse
Nom de service : {http://cigma.formation.ma/}CalculatorImplService
Nom de port : {http://cigma.formation.ma/}CalculatorImplPort
Informations
Adresse : http://localhost:8080/tpjaxwsee/calculator
WSDL: http://localhost:8080/tpjaxwsee/calculator?wsdl
Classe d'implémentation : ma.formation.cigma.CalculatorImpl