INSA de Toulouse
guermouc@insa-toulouse.fr

Tuto 2 : Client d'un service Web

a. Compétences visées :

En suivant les étapes indiquées dans ce tuto, vous devriez être capable de créer un client pour un service Web existant et de l'invoquer.

b. Développement d'un client pour consommer un service Web

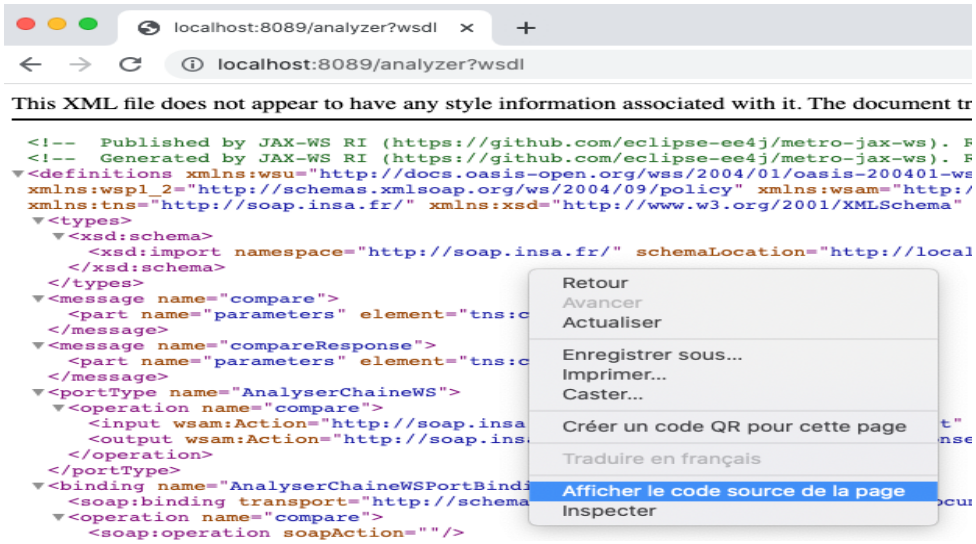
Une fois un service Web est développé et testé, l'étape suivante consiste à l'utiliser, ou autrement dit, l'invoquer via un code client. Pour ce faire, vous allez créer un client. Le client peut être une application développée en Java SE (une classe java…), Java EE avec (JSP, Servlet, …), ou un autre service Web.

Comme vous l'avez vu en cours, pour invoquer un service Web SOAP, vous aurez besoin de savoir où est ce qu'il est, ce qu'il fait, les données attendues, …etc. Ces informations se trouvent où ? Elles se trouvent dans le WSDL. Si vous ne le saviez pas, il faut revoir le cours et les tutos depuis le départ en étant attentif !

Commencez par créer un projet Maven simple. Comme fait précédemment, vous allez ajouter les dépendances dont vous aurez besoin ultérieurement. Ici on a créé un projet *ClientWS*.

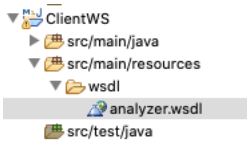
N'oubliez pas de lancer votre service analyzer. Si vous voulez l'appeler, il faut qu'il soit en exécution. Ouvrez le WSDL de votre service.

Puis afficher son code source (clic droit/Afficher le code source de la page).



Copiez le code de votre WSDL. Vous allez l'utiliser dans votre projet.

Allez dans *src/main/resources* du projet que vous venez de créer. Créez un répertoire wsdl (clic droit/new/Folder). Dans ce répertoire, créez un fichier wsdl, et appelez le analyzer.wsdl. Copiez dedans le contenu du WSDL copié.



Pour générer le code à partir du WSDL, vous allez avoir besoin d'un plugin spécifique. Ici vous allez utiliser le plugin *apache* cxf. Ajoutez dans votre fichier *pom.xml* l'élément build comme suit :

```
<build>
        <plugins>
                <plugin>
                    <groupId>org.apache.cxf</groupId>
                    <artifactId>cxf-codegen-plugin</artifactId>
                    <version>${cxf.version}</version>
                    <executions>
                        <execution>
                            <id>generate-sources</id>
                            <phase>generate-sources</phase>
                            <configuration>
                                <sourceRoot>
                                    ${project.build.directory}/generated-sources/cxf
                                </sourceRoot>
                                <wsdlOptions>
                                    <wsdlOption>
                                       <wsdl>
                                           ${basedir}/src/main/resources/wsdl/analyzer.wsdl
                                       </wsdl>
                                        <packagenames>
                                              <packagename>fr.insa.soap.wsdltojava</packagename>
                                           </packagenames>
                                    </wsdlOption>
                                </wsdlOptions>
                            </configuration>
                            <goals>
                                <goal>wsdl2java</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
        </plugins>
  </build>
```

Vous remarquerez l'élément *sourceRoot* (`<sourceRoot>${project.build.directory}/generated-sources/cxf</sourceRoot>`) sous configuration. Dans cet élément, on précise l'emplacement dans lequel les fichiers java seront générés. *Project.build.directory* fait référence à l'emplacement dédié aux fichiers issus des builds (i.e., *target*). Dans ce repertoire, les sources seront mis sous *generated-sources/cxf*.

L'élément *wsdlOption* permet d'indiquer l'emplacement du fichier WSDL (dans le sous élément `<wsdl>${basedir}/src/main/resources/wsdl/analyzer.wsdl</wsdl>`). Le sous élément *packagename* permet d'indiquer le package dans lequel seront générés les codes sources java à partir du fichier WSDL (ici ce sera `fr.insa.soap.wsdltojava`).

Ajoutez l'élément properties également pour préciser les versions de java et de cxf :

INSA de Toulouse
guermouc@insa-toulouse.fr

```
<properties>
  <java.version>11</java.version>
  <cxf.version>3.4.2</cxf.version>
 </properties>
```

Enfin, ajoutez aussi la dépendance *jaxws-rt* que vous avez déjà importé lors du tuto précédent.

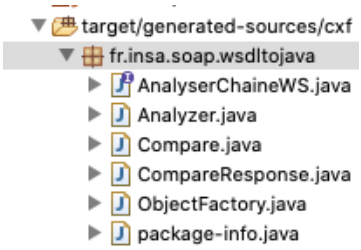
```
<dependencies>
   <!-- https://mvnrepository.com/artifact/com.sun.xml.ws/jaxws-rt -->
   <dependency>
      <groupId>com.sun.xml.ws</groupId>
      <artifactId>jaxws-rt</artifactId>
      <version>2.3.3</version>
   </dependency>
</dependencies>
```

Le fichier pom.xml est comme suit :

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.insa.soap</groupId>
  <artifactId>ClientWS</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
     <java.version>11</java.version>
     <cxf.version>3.4.2</cxf.version>
  </properties>
  <dependencies>
      <!-- https://mvnrepository.com/artifact/com.sun.xml.ws/jaxws-rt -->
      <dependency>
         <groupId>com.sun.xml.ws</groupId>
         <artifactId>jaxws-rt</artifactId>
          <version>2.3.3</version>
      </dependency>
  </dependencies>
  <build>
      <plugins>
         <plugin>
           <groupId>org.apache.cxf</groupId>
           <artifactId>cxf-codegen-plugin</artifactId>
           <version>${cxf.version}</version>
           <executions>
               <execution>
                   <id>generate-sources</id>
                   <phase>generate-sources</phase>
                   <configuration>
                      <sourceRoot>
                          ${project.build.directory}/generated-sources/cxf
                       </sourceRoot>
                       <wsdlOptions>
                          <wsdlOption>
                              <wsdl>${basedir}/src/main/resources/wsdl/analyzer.wsdl</wsdl>
                              <packagenames>
                                 <packagename>fr.insa.soap.wsdltojava</packagename>
                               </packagenames>
                           </wsdlOption>
                       </wsdlOptions>
                   </configuration>
                   <goals>
                      <goal>wsdl2java</goal>
                   </goals>
               </execution>
           </executions>
         </plugin>
       </plugins>
  </build>
</project>
```

Enregistrez et mettez à jour votre projet. Vous allez remarquer que sous *target/generated-sources/cxf*, un package *fr.insa.soap.wsdltojava* a été créé. Dépliez-le. Vous remarquerez un ensemble de classes/interface :



Examinez le fichier WSDL et les classes/interface générées pour comprendre le lien entre les deux.

Par exemple, vous remarquerez que dans le fichier wsdl, vous avez le `<portType name="AnalyserChaineWS">` qui correspond à l'interface générée *AnalyserChaineWS*.

Vous allez invoquer votre service *analyzer*. Créez donc une classe java classique dans laquelle vous allez mettre votre méthode main. Attention, il faut créer la classe à l'emplacement adéquat (i.e., sous src/main/java).

Appelez là comme vous le souhaitez. Ici on l'a appelé *ClientOfAnalyzer*. Insérez le code suivant à votre classe. Les explications sont données sous la forme de commentaires dans le code source.

```
package fr.insa.soap;

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

import fr.insa.soap.wsdltojava.AnalyserChaineWS;
import fr.insa.soap.wsdltojava.Analyzer;

public class ClientOfAnalyzer {

    public static void main(String [] args) throws MalformedURLException {
        //l'adresse du service Web
        final String adresse="http://localhost:8089/analyzer";

        //Création de l'URL
        final URL url=URI.create(adresse).toURL();

        //Instanciation de l'image du service
        final Analyzer service= new Analyzer(url);

        //Création du proxy (en utilisant le portType) pour l'appel du service
        final AnalyserChaineWS port = service.getPort(AnalyserChaineWS.class);

        String chaine ="aaaa";
        //appel de la méthode compare via le port
        System.out.println("La taille de la chaine "+chaine+" est "+port.compare("aaaa"));
    }
}
```

Exécutez votre classe.

c. Exercice :

Si vous souhaitez d'appeler un service distant que vous n'avez pas développé et dont vous ne disposez pas de code, essayez de créer un client pour invoquer un service existant dont le WSDL est consultable sur le lien : https://www.dataaccess.com/webservicesserver/numberconversion.wso?WSDL