Rapport de TP : RMI et Services Web SOAP

1 Introduction

Ce TP a pour but de reproduire un serveur RMI et un service Web SOAP en Java. Nous avons mis en place un service Web simple, l'avons publié avec JAX-WS, puis testé avec SOAPUI. Bien que le TP soit réalisé en local, le principe reste celui d'une communication entre deux programmes, potentiellement distants. Pour bien comprendre les explications dans ce qui suit, on va prendre une analogie d'un restaurant avec un client et un serveur.

2 Principe général

Un service Web SOAP permet à un **client** d'appeler une méthode située sur un **serveur**. Le client ne connaît pas le code Java du serveur : il échange via des **messages SOAP** (XML). Le fonctionnement :

1. le client envoie une requête SOAP contenant le nom de la méthode et ses paramètres,
2. le serveur exécute la méthode,
3. il renvoie une réponse SOAP avec le résultat.

SOAP est comme une enveloppe standardisée permettant aux programmes de communiquer, même s'ils n'utilisent pas le même langage.

Analogie: Au lieu de servir la commande du client directement dans un plat, on mets la commande dans une boîte et le client à ses propres méthodes pour ouvrir la boîte et la transformer au plat qu'il a commandé.

3 Namespace et WSDL

Le service utilise un **namespace** :

```
@WebService(targetNamespace="http://www.polytech.fr")
```

Il agit comme un identifiant unique du service et évite les conflits de noms.
Une fois le service publié, le fichier WSDL est disponible à :

`http://localhost:8888/?wsdl`

Le WSDL décrit :

- les méthodes disponibles,
- leurs paramètres,
- les types XML utilisés,
- l'adresse du service.

Il sert de **manuel d'utilisation** pour n'importe quel client(le menu du restaurant).

4 Structure du TP

Le TP contient essentiellement trois fichiers.

4.1 1. MonServiceWeb.java

C'est la classe qui contient les méthodes appelables par le client. Extrait :

```
@WebService
public class MonServiceWeb {
    @WebMethod
    public double conversion(double mt) {
        return mt * 0.9;
    }
}
```

Cette méthode devient un service accessible via SOAP. **Analogie**: c'est la préparation de la commande

4.2 2. Application.java

Ce fichier publie le service à une URL.

```
Endpoint.publish("http://localhost:8888/",
                 new MonServiceWeb());
```

À l'exécution, le service Web devient disponible et prêt à recevoir des requêtes SOAP. **Analogie**: C'est la personne qui que le restaurant est ouvert et prêt.

4.3 3. Etudiant.java

Cette classe illustre l'envoi d'un objet via SOAP (on envoie une commande plus compliquée):

```
@XmlRootElement
public class Etudiant {
    private int id;
    private String nom;
}
```

Grâce à `@XmlRootElement`, l'objet peut être converti automatiquement en XML.

5 Utilisation de SOAPUI

SOAPUI nous permet de tester le service sans écrire de code. Les étapes effectuées :

1. créer un projet SOAP,
2. entrer l'URL du WSDL,

3. choisir une opération (ex. : `convertir`),
4. saisir une valeur (ex. : 100),
5. envoyer la requête.

SOAPUI affiche alors :

- la requête SOAP envoyée (XML),
- la réponse SOAP reçue.

Exemple de requête simplifiée :

```
<convertir><mt>100</mt></convertir>
```

Exemple de réponse :

```
<convertirResponse><return>90</return></convertirResponse>
```

6 Fonctionnement global récapitulé

Une vision globale du déroulement du TP :

1. Création d'une classe Java annotée `@WebService`.
2. Publication du service avec `Endpoint.publish`.
3. Accès automatique au fichier WSDL.
4. Test via SOAPUI en envoyant des requêtes SOAP.
5. Réception des réponses XML envoyées par le serveur.

7 Conclusion

Ce TP m'a permis de comprendre les bases des services Web SOAP en Java :

- le rôle de SOAP comme enveloppe XML de communication,
- l'importance du WSDL et du namespace,
- la publication d'un service avec JAX-WS,
- et l'utilisation de SOAPUI pour tester un service.

Les codes correspondants à chaque partie se trouve dans ce même dossier.