Services web SOAP en Java, workflow, astuces, et bonnes pratiques

Approches de Développement de Services Web

Lors de la création d'un service web, il existe deux approches principales :

1. **Contract-First (*Top-Down*)** :
    - Commencer par le contrat WSDL et générer les classes Java nécessaires à partir de celui-ci.
    - Cette approche permet de garder le contrat indépendant de l'implémentation.
2. **Contract-Last (*Bottom-Up*)** :
    - Commencer par le code Java et générer le WSDL à partir des classes.
    - Cela peut être plus simple, car écrire un WSDL peut devenir complexe, en particulier pour des services Web intriqués.
    - Cependant, des modifications dans le code Java peuvent nécessiter des changements correspondants dans le WSDL généré.

Frameworks et Outils

Utiliser des frameworks peut simplifier les complexités associées à SOAP :

1. **JAX-WS (Java API for XML Web Services)** :
    - Une API standardisée pour créer et consommer des services Web SOAP en Java.
    - Supporte à la fois les approches *contract-first* et *contract-last*.
2. **Spring-WS** :
    - Un framework qui ne prend en charge que l'approche *contract-first*.
    - Simplifie la gestion de SOAP et améliore l'intégration avec les applications Spring.

Aperçu de Java API for XML-Based Services (JAX-WS)

Principe

1. **définition** : un modèle de programmation de services Web complétant le modèle de base fourni par **Java API for XML-based RPC (JAX-RPC)**.
2. **utilité** : il simplifie le développement des clients et services Web en utilisant des proxy dynamiques et des annotations Java, permettant une intégration fluide dans les applications Java.

`wsimport`

1. un outil CLI inclus avec la JDK permettant la génération automatique de classes Java pour un client d'un service web à partir d'un document WSDL.
2. Les classes générées incluent des stubs (proxys), des classes de données (POJO) et des interfaces nécessaires pour interagir avec le service web défini dans le WSDL.
3. les artefacts générés sont compatibles jusqu'à la JDK1.8, donc il faut s'assurer que les projets serveur et client utilise cette version de la JDK également.

Créer un service web SOAP avec JAX-WS (*Contract-Last*) et Spring Boot

Création du projet

Pour créer un projet Spring Boot, vous pouvez utiliser Spring Initializr, soit via un plugin IDE (comme Eclipse ou IntelliJ), soit en utilisant la version en ligne (https://start.spring.io/).

Remplissez les informations de votre projet :

1. **Project** : Maven Project
2. **Language** : Java
3. **Spring Boot** : Sélectionnez la version avec Java 17 (minimum).
4. **Group** : (ex : org.examples)
5. **Artifact** : (ex : math-service)
6. **Name** : (ex : MathService)
7. **Description** : (ex : Demo project for SOAP web services)
8. **Package Name** : (ex : org.examples.mathservice)
9. **Packaging** : Jar

Ajoutez les dépendances nécessaires :

1. **Spring Web** : Pour les fonctionnalités Web (*via Spring Initializr*).
2. **Spring Boot** :
   - Modifier la version de Spring Boot à la main dans le `pom.xml` à 2.7.18 pour assurer la compatibilité avec JDK 1.8.
   - Selon l'IDE utilisé, assurez-vous de configurer le build path et les options du compilateur pour utiliser la JDK 1.8.
3. **JAX-WS** : Ajoutez les dépendances à la main dans le `pom.xml` après la création du projet.

```
<!--pom.xml-->
<dependencies>
    ...
    <dependency>
        <groupId>org.jboss.spec</groupId>
        <artifactId>jboss-javaee-6.0</artifactId>
        <version>1.0.0.Final</version>
        <type>pom</type>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>com.sun.xml.ws</groupId>
        <artifactId>rt</artifactId>
        <version>2.3.1</version>
    </dependency>
    ...
</dependencies>
```

Configuration

JDK1.8 et `wsimport`

1. installez la JDK 1.8 selon votre OS: (https://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html).
2. `wsimport` est installé automatiquement avec la JDK 1.8 (*dans le dossier `bin/`*)
3. **N.B.** il faut bien s'assurer que le dossier `bin/` est dans la variable PATH du système pour pouvoir utiliser `wsimport` en ligne de commandes.

Conception

Définir les services web

Lors de la création de services web, il faut adopter une interface simple pour les opérations afin de les rendre facilement consommables par les clients.

En Java, cela implique de créer une paire interface/classe annotées par `@WebService` pour définir l'interface et l'implémentation d'un service web. Les opérations exposées par le service web sont des méthodes déclarées au niveau de l'interface du service web, annotées par `@WebMethod`, et implémentées par sa classe d'implémentation. De plus, l'annotation `@WebService` de la classe d'implémentation aura l'attribut `endpointInterface="com.example.somepackage.ServiceWebInterface"` pour référencer l'interface du service implémenté.

Enfin, pour chaque opération du service web il faut prévoir les cas d'erreurs possibles et les traiter via le mécanisme d'exceptions en Java.

```java
// ../source/soap/java/snippets/web_service_soap_generic_example_jax_ws.java
package com.example.somepackage;

/* Interface générique d'un service web */
@WebService
public interface ServiceWebSoap {
    @WebMethod
    typeRetour1 operation1([params]);

    @WebMethod
    typeRetour2 operation2([params]);

    @WebMethod
    typeRetour3 operation3([params]);
    ...
}

/* Classe d'implémentation d'un service web */
@WebService(endpointInterface="com.example.somepackage.ServiceWebSoap")
public class ServiceWebSoapImpl implements ServiceWebSoap {
    /* attributs */
    /* méthodes */
    // méthodes de l'interface
    @Override
    public typeRetour1 operation1([params]){
        // implémentation
    }
```

```
    @Override
    public typeRetour2 operation2([params]) {
        // implémentation
    }

    @Override
    public typeRetour3 operation3([params]) {
        // implémentation
    }
    ...

    // autres méthodes
}
```

Publier le service web

Publier le service web consiste à l'associer à un endpoint et de générer son WSDL afin qu'il soit exploitable par des applications clientes. Pour ce faire, il suffit de créer une classe serveur qui déclare une méthode `main()` contenant l'instruction `javax.xml.ws.Endpoint.publish("uri_service_web", new ServiceWebImpl())`.

Une fois `main()` exécutée, le service web sera consultable via son URI (*e.g.*, *http://localhost:8080/employeeservice*) et prêt à être consommé par les application clientes.

```
//
../source/soap/java/snippets/web_service_soap_publisher_generic_example_jax_ws.jav
a
package com.example.mypackage.server;

import javax.xml.ws.Endpoint;

public class WebServicePublisher {
    public static void main(String[] args) {
        Endpoint.publish("uri_service_web",
                new ServiceWebImpl());
        System.err.println("Server is ready");
    }
}
```

Définir les données échangées

Dans le cadre des web services, le modèle des données échangées comporte soit des types natifs, soit des types personnalisés. Les types de données sont encodés en XML en WSDL afin d'être échangés au sein de messages SOAP entre le fournisseur du service web et ses clients.

Exemple avec des types natifs

Supposons qu'on souhaite créer un service web pour effectuer des opérations arithmétiques d'une calculatrice. Les opérations du service incluent l'addition, la soustraction, la multiplication, et la division, alors que les données échangées sont de type entier. Une exception sera levée si l'on essaie de diviser par zéro. Voici un exemple avec JAX-WS et Spring Boot :

```
/src/main/java
    org.example.web_services.mathservice.main
        Application.java
    org.example.web_services.mathservice.publisher
        MathServicePublisher.java
    org.example.web_services.mathservice.service
        MathService.java
        MathServiceImpl.java
/src/main/resources
    application.properties
```

```
package org.example.web_services.mathservice.service;

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

// Déclaration de l'interface MathService comme un service web
@WebService
public interface MathService {

    @WebMethod
    int add(int a, int b);

    @WebMethod
    int subtract(int a, int b);

    @WebMethod
    int multiply(int a, int b);

    @WebMethod
    int divide(int a, int b) throws IllegalArgumentException; // Exception levée
si division par zéro
}

package org.example.web_services.mathservice.service;

import javax.jws.WebService;

// Implémentation de l'interface MathService
@WebService(endpointInterface = "org.example.web_services.service.MathService")
public class MathServiceImpl implements MathService {

    // Implémentation de l'addition
    @Override
    public int add(int a, int b) {
```

```
        return a + b;
    }

    // Implémentation de la soustraction
    @Override
    public int subtract(int a, int b) {
        return a - b;
    }

    // Implémentation de la multiplication
    @Override
    public int multiply(int a, int b) {
        return a * b;
    }

    // Implémentation de la division
    @Override
    public int divide(int a, int b) throws IllegalArgumentException {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero"); //
Gestion de l'exception
        }
        return a / b;
    }
}

package org.example.web_services.mathservice.publisher;

import javax.xml.ws.Endpoint;

import org.example.web_services.service.MathServiceImpl;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Configuration;

// Configuration pour publier le service web lors du démarrage de l'application
@Configuration
public class MathServicePublisher implements CommandLineRunner {
    private static final String SERVICE_URI = "http://localhost:8080/mathservice";
// URI du service

    @Override
    public void run(String... args) throws Exception {
        // Publication du service web
        Endpoint.publish(SERVICE_URI, new MathServiceImpl());
        System.err.println("Web Service successfully published at: " +
SERVICE_URI);
        System.err.println("Server ready!"); // Message indiquant que le serveur
est prêt
    }
}

package org.example.web_services.mathservice.main;

import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;

// Classe principale pour démarrer l'application Spring Boot
@SpringBootApplication(scanBasePackages = {
    "org.example.web_services.service", // Scanner le package du service
    "org.example.web_services.publisher" // Scanner le package de publication
})
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args); // Démarrage de
l'application
    }
}
```

Lors de son publication, le service web aura le WSDL suivant :

```
<?xml version='1.0' encoding='UTF-8'?><!-- Published by JAX-WS RI (http://jax-
ws.java.net). RI's version is
JAX-WS RI 2.3.1 svn-revision#6ef5f7eb9a938dbc4562f25f8fa0b67cc4ff2dbb. --><!--
Generated by JAX-WS RI (http://javaee.github.io/metro-jax-ws).
RI's version is JAX-WS RI 2.3.1 svn-
revision#6ef5f7eb9a938dbc4562f25f8fa0b67cc4ff2dbb. -->
<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://service.web_services.mathservice.example.org/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://service.web_services.mathservice.example.org/"
name="MathServiceImplService">
    <types>
        <xsd:schema>
            <xsd:import
namespace="http://service.web_services.mathservice.example.org/"
                schemaLocation="http://localhost:8080/mathservice?xsd=1" />
        </xsd:schema>
    </types>
    <message name="add">
        <part name="parameters" element="tns:add" />
    </message>
    <message name="addResponse">
        <part name="parameters" element="tns:addResponse" />
    </message>
    <message name="divide">
        <part name="parameters" element="tns:divide" />
    </message>
    <message name="divideResponse">
        <part name="parameters" element="tns:divideResponse" />
```

```
    </message>
    <message name="multiply">
        <part name="parameters" element="tns:multiply" />
    </message>
    <message name="multiplyResponse">
        <part name="parameters" element="tns:multiplyResponse" />
    </message>
    <message name="subtract">
        <part name="parameters" element="tns:subtract" />
    </message>
    <message name="subtractResponse">
        <part name="parameters" element="tns:subtractResponse" />
    </message>
    <portType name="MathService">
        <operation name="add">
            <input
wsam:Action="http://service.web_services.mathservice.example.org/MathService/addRe
quest"
                message="tns:add" />
            <output
wsam:Action="http://service.web_services.mathservice.example.org/MathService/addRe
sponse"
                message="tns:addResponse" />
        </operation>
        <operation name="divide">
            <input
wsam:Action="http://service.web_services.mathservice.example.org/MathService/divid
eRequest"
                message="tns:divide" />
            <output
wsam:Action="http://service.web_services.mathservice.example.org/MathService/divid
eResponse"
                message="tns:divideResponse" />
        </operation>
        <operation name="multiply">
            <input
wsam:Action="http://service.web_services.mathservice.example.org/MathService/multi
plyRequest"
                message="tns:multiply" />
            <output

wsam:Action="http://service.web_services.mathservice.example.org/MathService/multi
plyResponse"
                message="tns:multiplyResponse" />
        </operation>
        <operation name="subtract">
            <input
wsam:Action="http://service.web_services.mathservice.example.org/MathService/subtr
actRequest"
                message="tns:subtract" />
            <output

wsam:Action="http://service.web_services.mathservice.example.org/MathService/subtr
actResponse"
```

```
                message="tns:subtractResponse" />
        </operation>
    </portType>
    <binding name="MathServiceImplPortBinding" type="tns:MathService">
        <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>
        <operation name="divide">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
        </operation>
        <operation name="multiply">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
        </operation>
        <operation name="subtract">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
        </operation>
    </binding>
    <service name="MathServiceImplService">
        <port name="MathServiceImplPort" binding="tns:MathServiceImplPortBinding">
            <soap:address location="http://localhost:8080/mathservice" />
        </port>
    </service>
</definitions>
```

Exemple avec des types personnalisés

Prenons un autre exemple plus complexe. Supposons que l'on dispose d'un ensemble d'employés ayant chacun un **identifiant** (*un entier*) et un **nom** (*un String*). Les employés sont des **Plain Old Java Object** (**POJO**), désignant des objets avec des attributs et des getters/setters.

On souhaite définir un service web permettant de manipuler cet ensemble de la manière suivante :

1. lister tous les employés ;
2. savoir le nombre des employés dans l'ensemble ;
3. récupérer un employé par son identifiant ;
4. ajouter un nouveau employé ;
5. supprimer un employé existant ;
6. modifier les informations d'un employé existant.

De plus, une erreur survient si l'on essaye :

1. de récupérer/modifier/supprimer un employé dont l'identifiant n'existe pas comme identifiant valide d'un employé dans l'ensemble.
2. d'ajouter un employé avec un identifiant déjà affecté à un autre employé dans l'ensemble.

Voici un exemple avec JAX-WS et Spring Boot :

```
/src/main/java
    org.example.web_services.employeeservice.exceptions
        EmployeeException.java
        EmployeeNotFoundException.java
        EmployeeAlreadyExistsException.java
    org.example.web_services.employeeservice.main
        Application.java
    org.example.web_services.employeeservice.model
        Employee.java
    org.example.web_services.employeeservice.publisher
        EmployeeServicePublisher.java
    org.example.web_services.employeeservice.service
        EmployeeService.java
        EmployeeServiceImpl.java
/src/main/resources
    application.properties
```

```java
package org.example.web_services.employeeservice.model;

public class Employee {
    /* ATTRIBUTES */
    private int id; // Identifiant de l'employé
    private String name; // Nom de l'employé

    /* CONSTRUCTORS */
    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
```

```
    }

    /* METHODS */
    public int getId() {
        return id; // Récupérer l'identifiant
    }

    public void setId(int id) {
        this.id = id; // Définir l'identifiant
    }

    public String getName() {
        return name; // Récupérer le nom
    }

    public void setName(String name) {
        this.name = name; // Définir le nom
    }
}
```

```
package org.example.web_services.employeeservice.exceptions;

// Superclasse de toutes les classes liées aux employées
public class EmployeeException extends Exception {
    public EmployeeException() {

    }

    public EmployeeException(String message) {
        super(message);
    }
}
```

```
package org.example.web_services.employeeservice.exceptions;

// Exception levée si l'employée n'est pas trouvé
public class EmployeeNotFoundException extends Exception {
    public EmployeeNotFoundException() {
    }

    public EmployeeNotFoundException(String message) {
        super(message);
    }
}
```

```
package org.example.web_services.employeeservice.exceptions;

// Exception levée si un employée existe déjà
public class EmployeeAlreadyExistsException extends Exception {
    public EmployeeAlreadyExistsException() {
    }

    public EmployeeAlreadyExistsException(String message) {
        super(message);
```

```
    }
}

// src/main/java/org/example/web_services/service/EmployeeService.java
package org.example.web_services.employeeservice.service;

import java.util.List;

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

@WebService
public interface EmployeeService {

    /* METHODS */
    @WebMethod
    int count(); // Compter le nombre d'employés

    @WebMethod
    List<Employee> getEmployees(); // Récupérer la liste des employés

    @WebMethod
    Employee addEmployee(int id, String name) throws
EmployeeAlreadyExistsException; // Ajouter un employé

    @WebMethod
    Employee getEmployee(int id) throws EmployeeNotFoundException; // Récupérer un
employé par identifiant

    @WebMethod
    Employee updateEmployee(int id, String name) throws EmployeeNotFoundException;
// Modifier un employé

    @WebMethod
    boolean deleteEmployee(int id) throws EmployeeNotFoundException; // Supprimer
un employé
}

package org.example.web_services.employeeservice.service;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;

import javax.jws.WebService;

import
org.example.web_services.employeeservice.exceptions.EmployeeAlreadyExistsException
;
import
org.example.web_services.employeeservice.exceptions.EmployeeNotFoundException;
import org.example.web_services.employeeservice.model.Employee;
```

```
@WebService(endpointInterface =
"org.example.web_services.employeeservice.service.EmployeeService")
public class EmployeeServiceImpl implements EmployeeService {

    /* ATTRIBUTES */
    private List<Employee> employees; // Liste des employés

    /* CONSTRUCTORS */
    public EmployeeServiceImpl() {
        employees = new ArrayList<>(); // Initialiser la liste des employés
        employees.addAll(Arrays.asList(
                new Employee(1, "Joe"),
                new Employee(2, "Jane"),
                new Employee(3, "Steve"),
                new Employee(4, "Alice"),
                new Employee(5, "Bob")
        ));
    }

    /* METHODS */
    @Override
    public int count() {
        return employees.size(); // Compter le nombre d'employés
    }

    @Override
    public List<Employee> getEmployees() {
        return employees; // Récupérer la liste des employés
    }

    @Override
    public Employee addEmployee(int id, String name) throws
EmployeeAlreadyExistsException {
        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (target.isPresent())
            throw new EmployeeAlreadyExistsException("Error: An employee with ID
" + id + " already exists");

        Employee employee = new Employee(id, name);
        employees.add(employee); // Ajouter l'employé à la liste
        return employee; // Retourner l'employé ajouté
    }

    @Override
    public Employee getEmployee(int id) throws EmployeeNotFoundException {
        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (!target.isPresent())
            throw new EmployeeNotFoundException("Error: No employee with ID " +
```

```
id + " exists");

        return target.get(); // Retourner l'employé trouvé
    }

    @Override
    public Employee updateEmployee(int id, String name) throws
EmployeeNotFoundException {
        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (!target.isPresent())
            throw new EmployeeNotFoundException("Error: No employee with ID " +
id + " exists");

        target.get().setName(name); // Mettre à jour le nom de l'employé
        return target.get(); // Retourner l'employé mis à jour
    }

    @Override
    public boolean deleteEmployee(int id) throws EmployeeNotFoundException {
        Optional<Employee> target = employees.stream()
                .filter(e -> e.getId() == id)
                .findFirst();

        if (!target.isPresent())
            throw new EmployeeNotFoundException("Error: No employee with ID " +
id + " exists");

        return employees.remove(target.get()); // Supprimer l'employé de la liste
    }
}

package org.example.web_services.employeeservice.publisher;

import javax.xml.ws.Endpoint;

import org.example.web_services.employeeservice.service.EmployeeServiceImpl;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Configuration;

//Configuration pour publier le service web lors du démarrage de l'application
@Configuration
public class EmployeeServicePublisher implements CommandLineRunner {
    private static final String SERVICE_URI =
"http://localhost:8080/employeeservice"; // URI du service

    @Override
    public void run(String... args) throws Exception {
        // Publication du service web
        Endpoint.publish(SERVICE_URI, new EmployeeServiceImpl());
        System.err.println("Web Service successfully published at: " +
SERVICE_URI);
```

```
            System.err.println("Server ready!");
        }
    }

    package org.example.web_services.employeeservice.main;

    import org.springframework.boot.SpringApplication;
    import org.springframework.boot.autoconfigure.SpringBootApplication;

    // Classe principale pour démarrer l'application Spring Boot
    @SpringBootApplication(scanBasePackages = {
            "org.example.web_services.employeeservice.exceptions", // Scanner le
    package des exceptions
            "org.example.web_services.employeeservice.model", // Scanner le package du
    modèle des données
            "org.example.web_services.employeeservice.service", // Scanner le package
    du service
            "org.example.web_services.employeeservice.publisher", // Scanner le
    package de publication
    })
    public class Application {
        public static void main(String[] args) {
            SpringApplication.run(Application.class, args);
        }
    }
```

Lors de son publication, le service web aura le WSDL suivant :

```
<?xml version='1.0' encoding='UTF-8'?><!-- Published by JAX-WS RI (http://jax-
ws.java.net). RI's version is
JAX-WS RI 2.3.1 svn-revision#6ef5f7eb9a938dbc4562f25f8fa0b67cc4ff2dbb. --><!--
Generated by JAX-WS RI (http://javaee.github.io/metro-jax-ws).
RI's version is JAX-WS RI 2.3.1 svn-
revision#6ef5f7eb9a938dbc4562f25f8fa0b67cc4ff2dbb. -->
<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://service.employeeservice.web_services.example.org/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://service.employeeservice.web_services.example.org/"
    name="EmployeeServiceImplService">
    <types>
        <xsd:schema>
            <xsd:import
namespace="http://service.employeeservice.web_services.example.org/"
                schemaLocation="http://localhost:8080/employeeservice?xsd=1" />
        </xsd:schema>
```

```
    </types>
    <message name="count">
        <part name="parameters" element="tns:count" />
    </message>
    <message name="countResponse">
        <part name="parameters" element="tns:countResponse" />
    </message>
    <message name="addEmployee">
        <part name="parameters" element="tns:addEmployee" />
    </message>
    <message name="addEmployeeResponse">
        <part name="parameters" element="tns:addEmployeeResponse" />
    </message>
    <message name="EmployeeAlreadyExistsException">
        <part name="fault" element="tns:EmployeeAlreadyExistsException" />
    </message>
    <message name="getEmployee">
        <part name="parameters" element="tns:getEmployee" />
    </message>
    <message name="getEmployeeResponse">
        <part name="parameters" element="tns:getEmployeeResponse" />
    </message>
    <message name="EmployeeNotFoundException">
        <part name="fault" element="tns:EmployeeNotFoundException" />
    </message>
    <message name="getEmployees">
        <part name="parameters" element="tns:getEmployees" />
    </message>
    <message name="getEmployeesResponse">
        <part name="parameters" element="tns:getEmployeesResponse" />
    </message>
    <message name="deleteEmployee">
        <part name="parameters" element="tns:deleteEmployee" />
    </message>
    <message name="deleteEmployeeResponse">
        <part name="parameters" element="tns:deleteEmployeeResponse" />
    </message>
    <message name="updateEmployee">
        <part name="parameters" element="tns:updateEmployee" />
    </message>
    <message name="updateEmployeeResponse">
        <part name="parameters" element="tns:updateEmployeeResponse" />
    </message>
    <portType name="EmployeeService">
        <operation name="count">
            <input

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/countRequest"
                message="tns:count" />
            <output

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/countResponse"
```

```
                message="tns:countResponse" />
        </operation>
        <operation name="addEmployee">
            <input

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/addEmployeeRequest"
                message="tns:addEmployee" />
            <output

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/addEmployeeResponse"
                message="tns:addEmployeeResponse" />
            <fault message="tns:EmployeeAlreadyExistsException"
                name="EmployeeAlreadyExistsException"

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/addEmployee/Fault/EmployeeAlreadyExistsException" />
        </operation>
        <operation name="getEmployee">
            <input

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/getEmployeeRequest"
                message="tns:getEmployee" />
            <output

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/getEmployeeResponse"
                message="tns:getEmployeeResponse" />
            <fault message="tns:EmployeeNotFoundException"
name="EmployeeNotFoundException"

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/getEmployee/Fault/EmployeeNotFoundException" />
        </operation>
        <operation name="getEmployees">
            <input

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/getEmployeesRequest"
                message="tns:getEmployees" />
            <output

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/getEmployeesResponse"
                message="tns:getEmployeesResponse" />
        </operation>
        <operation name="deleteEmployee">
            <input

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/deleteEmployeeRequest"
                message="tns:deleteEmployee" />
```

```
            <output

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/deleteEmployeeResponse"
                message="tns:deleteEmployeeResponse" />
            <fault message="tns:EmployeeNotFoundException"
name="EmployeeNotFoundException"

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/deleteEmployee/Fault/EmployeeNotFoundException" />
        </operation>
        <operation name="updateEmployee">
            <input

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/updateEmployeeRequest"
                message="tns:updateEmployee" />
            <output

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/updateEmployeeResponse"
                message="tns:updateEmployeeResponse" />
            <fault message="tns:EmployeeNotFoundException"
name="EmployeeNotFoundException"

wsam:Action="http://service.employeeservice.web_services.example.org/EmployeeServi
ce/updateEmployee/Fault/EmployeeNotFoundException" />
        </operation>
    </portType>
    <binding name="EmployeeServiceImplPortBinding" type="tns:EmployeeService">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
        <operation name="count">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
        </operation>
        <operation name="addEmployee">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
            <fault name="EmployeeAlreadyExistsException">
                <soap:fault name="EmployeeAlreadyExistsException" use="literal" />
            </fault>
        </operation>
        <operation name="getEmployee">
```

```
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
            <fault name="EmployeeNotFoundException">
                <soap:fault name="EmployeeNotFoundException" use="literal" />
            </fault>
        </operation>
        <operation name="getEmployees">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
        </operation>
        <operation name="deleteEmployee">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
            <fault name="EmployeeNotFoundException">
                <soap:fault name="EmployeeNotFoundException" use="literal" />
            </fault>
        </operation>
        <operation name="updateEmployee">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
            <fault name="EmployeeNotFoundException">
                <soap:fault name="EmployeeNotFoundException" use="literal" />
            </fault>
        </operation>
    </binding>
    <service name="EmployeeServiceImplService">
        <port name="EmployeeServiceImplPort"
binding="tns:EmployeeServiceImplPortBinding">
            <soap:address location="http://localhost:8080/employeeservice" />
        </port>
    </service>
</definitions>
```

Abstraction des détails internes

Parfois une opération exposée peut utiliser une logique métier complexe et/ou de bas niveau pour accomplir une sous-tâche de sa fonctionnalité complète. Dans ce cas, cette logique métier complexe peut être encapsulée au sein d'une autre opération interne et non exposée par le service web. On parle ainsi de l'abstraction des détails internes d'une classe.

```
//
../source/soap/java/snippets/web_service_soap_generic_example_with_operation_abstr
action_jax_ws.java
package com.example.somepackage;

/* Interface générique d'un service web */
@WebService
public interface ServiceWebSoap {
    @WebMethod
    typeRetour1 operation1([params]);

    @WebMethod
    typeRetour2 operation2([params]);

    @WebMethod
    typeRetour3 operation3([params]);
    ...
}

/* Classe d'implémentation d'un service web */
@WebService(endpointInterface="com.example.somepackage.ServiceWebSoap")
public class ServiceWebSoapImpl implements ServiceWebSoap {
    /* attributs */
    /* méthodes */
    // méthodes de l'interface
    @Override
    public typeRetour1 operation1([params]){
        // implémentation
        // éventuellement invocation de méthodes internes
    }

    @Override
    public typeRetour2 operation2([params]) {
        // implémentation
        // éventuellement invocation de méthodes internes
    }

    @Override
    public typeRetour3 operation3([params]) {
        // implémentation
        // éventuellement invocation de méthodes internes
    }

    // méthode d'implémentation internes
    visibilite typeRetour4 operationInterne1([params]) {
```

```
        // implémentation
        // en général visibilite = private ou protected
    }
    ...
}
```

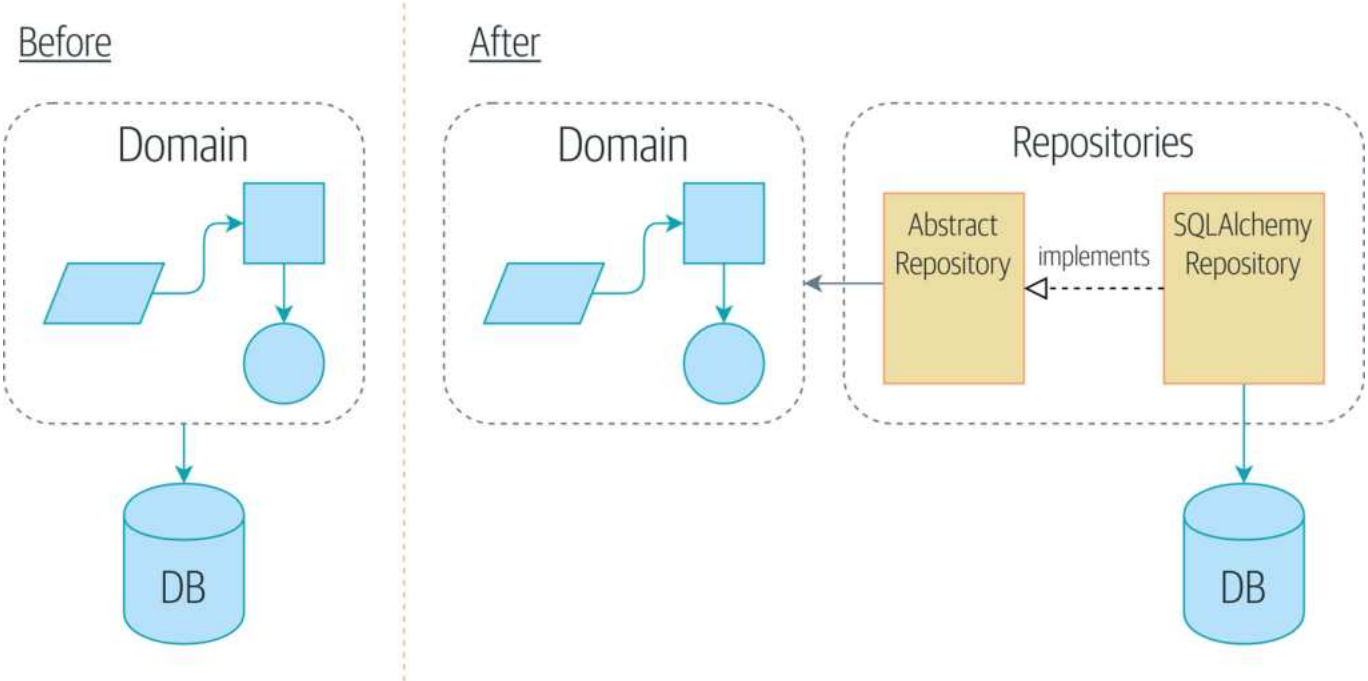
Utilisation du Repository Pattern

Motivation

Dans l'exemple du service web défini sur un ensemble d'employés, si les employés sont stockés dans une base de données, il est essentiel d'abstraire l'accès aux données afin de garantir que l'implémentation du service web demeure indépendante du mécanisme de persistance. Le Repository pattern permet de séparer la logique métier des détails d'accès aux données, facilitant ainsi la maintenance et l'évolution de l'application.

Le design pattern Repository

Le design pattern Repository permet d'encapsuler l'accès aux données et de fournir une interface pour la gestion des entités. Avec Spring Data JPA, le développement devient plus rapide et moins verbeux, car de nombreuses opérations CRUD peuvent être générées automatiquement. L'utilisation de H2 permet également de bénéficier d'une base de données légère en mémoire pour les tests et le développement pendant l'exécution.



... org.springframework.boot spring-boot-starter-data-jpa com.h2database h2 runtime ... ```

Exemple de service web des employés avec Spring Data JPA et H2 database

```
/src/main/java
    org.example.web_services.employeeservice.data
        DatabaseInitializer.java
```

```
    org.example.web_services.employeeservice.exceptions
        EmployeeException.java
        EmployeeNotFoundException.java
        EmployeeAlreadyExistsException.java
    org.example.web_services.employeeservice.main
        Application.java
    org.example.web_services.employeeservice.model
        Employee.java
    org.example.web_services.employeeservice.publisher
        EmployeeServicePublisher.java
    org.example.web_services.employeeservice.repository
        EmployeeRepository.java
    org.example.web_services.employeeservice.service
        EmployeeService.java
        EmployeeServiceImpl.java
/src/main/resources
    application.properties
```

```
spring.application.name=org.example.web_services.employeeservice.server
server.port=8081
spring.h2.console.enabled=true # accessible at <url>:<server.port>/h2-console
spring.datasource.driverClassName=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.datasource.url=jdbc:h2:mem:testdb;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

```
package org.example.web_services.employeeservice.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

// Annotation pour définir l'entité JPA
@Entity
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id; // Identifiant de l'employé

    private String name; // Nom de l'employé

    // Constructeurs
    public Employee() {
```

```
    }

    public Employee(String name) {
        this.name = name;
    }

    // Getters et Setters
    public int getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

package org.example.web_services.employeeservice.repository;

import org.example.web_services.employeeservice.model.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

// Interface de repository pour gérer les employés
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {
}

package org.example.web_services.employeeservice.service;

import java.util.List;

import javax.jws.WebService;

import
org.example.web_services.employeeservice.exceptions.EmployeeAlreadyExistsException
;
import
org.example.web_services.employeeservice.exceptions.EmployeeNotFoundException;
import org.example.web_services.employeeservice.model.Employee;
import org.example.web_services.employeeservice.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;

@WebService(endpointInterface =
"org.example.web_services.employeeservice.service.EmployeeService")
public class EmployeeServiceImpl implements EmployeeService {
```

```
    @Autowired
    private EmployeeRepository repository; // Injection de dépendance du
repository

    @Override
    public int count() {
        return (int) repository.count(); // Compter les employés
    }

    @Override
    public List<Employee> getEmployees() {
        return repository.findAll(); // Récupérer la liste des employés
    }

    @Override
    public Employee addEmployee(String name) throws EmployeeAlreadyExistsException
{
        // Vérification d'existence d'un employé par son nom
        if (repository.findAll().stream().anyMatch(e -> e.getName().equals(name)))
{
            throw new EmployeeAlreadyExistsException("Error: An employee with name
" + name + " already exists");
        }
        return repository.save(new Employee(name)); // Ajouter l'employé
    }

    @Override
    public Employee getEmployee(int id) throws EmployeeNotFoundException {
        return repository.findById(id)
            .orElseThrow(() -> new EmployeeNotFoundException("Error: No employee
with ID " + id + " exists")); // Récupérer un employé par identifiant
    }

    @Override
    public Employee updateEmployee(int id, String name) throws
EmployeeNotFoundException {
        Employee employee = getEmployee(id);
        employee.setName(name); // Mettre à jour le nom
        return repository.save(employee); // Sauvegarder les modifications
    }

    @Override
    public boolean deleteEmployee(int id) throws EmployeeNotFoundException {
        Employee employee = getEmployee(id);
        repository.delete(employee); // Supprimer l'employé
        return true; // Indiquer la suppression réussie
    }
}

package org.example.web_services.employeeservice.data;

import org.example.web_services.employeeservice.model.Employee;
import org.example.web_services.employeeservice.repository.EmployeeRepository;
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

//Configuration class for initializing the database with test data
@Configuration
public class DatabaseInitializer {
    // Logger instance for logging information
    private Logger logger = LoggerFactory.getLogger(DatabaseInitializer.class);

    // Bean that runs on application startup to preload the database
    @Bean
    CommandLineRunner initDataBase(EmployeeRepository repository) {
        return args -> {
            // Preloading the database with test Employee data and logging each
entry
            logger.info("Preloading database with " + repository.save(
                    new Employee("Joe")));
            logger.info("Preloading database with " + repository.save(
                    new Employee("Jane")));
            logger.info("Preloading database with " + repository.save(
                    new Employee("Steve")));
            logger.info("Preloading database with " + repository.save(
                    new Employee("Alice")));
            logger.info("Preloading database with " + repository.save(
                    new Employee("Bob")));
        };
    }

    // Bean that runs on application startup to provide an implementation for the
web service
    @Bean
    EmployeeService initEmployeeService() {
        return new EmployeeServiceImpl();
    }
}

package org.example.web_services.employeeservice.main;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

//EntityScan annotation to specify the package containing JPA entities
@EntityScan(basePackages = {
        "org.example.web_services.employeeservice.model"
})
//EnableJpaRepositories annotation to specify the package for JPA repositories
@EnableJpaRepositories(basePackages = {
        "org.example.web_services.employeeservice.repository"
})
//Main Spring Boot application class with configurations for package scanning
```

```
@SpringBootApplication(scanBasePackages = {
        "org.example.web_services.employeeservice.exceptions",
        "org.example.web_services.employeeservice.service",
        "org.example.web_services.employeeservice.data",
        "org.example.web_services.employeeservice.publisher",
})
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Consommer un service web SOAP à partir d'un client JAX-WS

Principe

Consommer un service web consiste à en créer un proxy qui sera utilisé par une application cliente afin d'invoquer ses opérations. Le proxy se chargera de réaliser les opérations réseaux, d'encodage et de décodage de données nécessaires avant d'invoquer la méthode souhaitée sur le service web.

Il faut donc générer la classe du proxy et les classes dont il dépend à partir de la WSDL du service web. Pour ce faire, il faut utiliser `wsimport` après la publication du service web (*i.e., le serveur est en cours d'exécution*) de la manière suivante :

```
wsimport -keep -p "com.example.package.clients" "uri_service_web?wsdl"
```

Cette commande générera les fichiers `.java` et `.class` nécessaires dans le package `chemin/dossier/courant/com.example.package.clients` à partir du WSDL du service web sur `"uri_service_web?wsdl"`

Ensuite, il faut importer les fichiers `.java` dans un package du projet client qui souhaite consommer le service web.

```
/* dans une méthode désirée du projet client */
// créer une instance de l'URI du service web à consommer
URL url = new URL("uri_service_web?wsdl");

// récupérer une instance de l'implémentation du service web
com.example.mypackage.client.ServiceWebImpl serviceImpl =
    new com.example.mypackage.client.ServiceWebImpl(url);

// récupérer un proxy du service web
com.example.mypackage.client.ServiceWeb proxy =
serviceImpl.getServiceWebImplPort();

// invoquer les méthodes exposées souhaitées du service web sur le proxy
proxy.operation1([params]);
```

```
proxy.operation2([params]);
...
```

Exemple d'une CLI consommant un web service calculatrice

Dans cet exemple, l'idée consiste à consommer un service web exposant des opérations d'une calculatrice. Ces opérations sont les mêmes vues dans l'exemple du service web `MathService`, notamment l'addition, la soustraction, la multiplication, et la division sur des données entières.

Pour ce faire, on crée un projet Spring Boot pour l'application cliente. On commence par générer le code Java nécessaire pour les opérations réseaux, d'encodage et de décodage sousjaçent le proxy en utilisant `wsimport` et on importe le paquetage obtenu au sein du projet client:

```
wsimport -keep -p "org.example.web_services.mathservice.client"
"http://localhost:8080/mathservice?wsdl"
```

Ces opérations seront utilisées par une CLI permettant à un client de les consommer. La CLI aura besoin ainsi de récupérer une instance du proxy du service web sur lequel elle invoquera l'opération choisie par le client.

Pour simplier, la CLI se contente ici d'afficher simplement les résultats de quelques opérations arithmétiques de base sans permettre à l'utilisateur de saisir lui même les valeurs d'une manière interactive, le but étant d'illustrer comment le proxy peut être utilisé par une CLI.

```
package org.example.web_services.client;

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

/**
 * <p>Java class for add complex type.
 *
 * <p>The following schema fragment specifies the expected content contained
within this class.
 *
 * <pre>
 * &lt;complexType name="add">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="arg0" type="{http://www.w3.org/2001/XMLSchema}int"/>
 *         &lt;element name="arg1" type="{http://www.w3.org/2001/XMLSchema}int"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
```

```
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "add", propOrder = {
    "arg0",
    "arg1"
})
public class Add {

    protected int arg0;
    protected int arg1;

    /**
     * Gets the value of the arg0 property.
     *
     */
    public int getArg0() {
        return arg0;
    }

    /**
     * Sets the value of the arg0 property.
     *
     */
    public void setArg0(int value) {
        this.arg0 = value;
    }

    /**
     * Gets the value of the arg1 property.
     *
     */
    public int getArg1() {
        return arg1;
    }

    /**
     * Sets the value of the arg1 property.
     *
     */
    public void setArg1(int value) {
        this.arg1 = value;
    }

}


package org.example.web_services.client;

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

```
/**
 * <p>Java class for addResponse complex type.
 *
 * <p>The following schema fragment specifies the expected content contained
within this class.
 *
 * <pre>
 * &lt;complexType name="addResponse">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="return" type="
{http://www.w3.org/2001/XMLSchema}int"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addResponse", propOrder = {
    "_return"
})
public class AddResponse {

    @XmlElement(name = "return")
    protected int _return;

    /**
     * Gets the value of the return property.
     *
     */
    public int getReturn() {
        return _return;
    }

    /**
     * Sets the value of the return property.
     *
     */
    public void setReturn(int value) {
        this._return = value;
    }

}


package org.example.web_services.client;

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

```
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for divide complex type.
 *
 * <p>The following schema fragment specifies the expected content contained
within this class.
 *
 * <pre>
 * &lt;complexType name="divide">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="arg0" type="{http://www.w3.org/2001/XMLSchema}int"/>
 *         &lt;element name="arg1" type="{http://www.w3.org/2001/XMLSchema}int"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "divide", propOrder = {
    "arg0",
    "arg1"
})
public class Divide {

    protected int arg0;
    protected int arg1;

    /**
     * Gets the value of the arg0 property.
     *
     */
    public int getArg0() {
        return arg0;
    }

    /**
     * Sets the value of the arg0 property.
     *
     */
    public void setArg0(int value) {
        this.arg0 = value;
    }

    /**
     * Gets the value of the arg1 property.
     *
```

```
     */
    public int getArg1() {
        return arg1;
    }

    /**
     * Sets the value of the arg1 property.
     *
     */
    public void setArg1(int value) {
        this.arg1 = value;
    }

}


package org.example.web_services.client;

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


/**
 * <p>Java class for divideResponse complex type.
 *
 * <p>The following schema fragment specifies the expected content contained
within this class.
 *
 * <pre>
 * &lt;complexType name="divideResponse">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="return" type="
{http://www.w3.org/2001/XMLSchema}int"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "divideResponse", propOrder = {
    "_return"
})
public class DivideResponse {

    @XmlElement(name = "return")
    protected int _return;
```

```
    /**
     * Gets the value of the return property.
     *
     */
    public int getReturn() {
        return _return;
    }

    /**
     * Sets the value of the return property.
     *
     */
    public void setReturn(int value) {
        this._return = value;
    }

}


package org.example.web_services.client;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.xml.bind.annotation.XmlSeeAlso;
import javax.xml.ws.Action;
import javax.xml.ws.RequestWrapper;
import javax.xml.ws.ResponseWrapper;


/**
 * This class was generated by the JAX-WS RI.
 * JAX-WS RI 2.2.9-b130926.1035
 * Generated source version: 2.2
 *
 */
@WebService(name = "MathService", targetNamespace =
"http://service.web_services.example.org/")
@XmlSeeAlso({
    ObjectFactory.class
})
public interface MathService {


    /**
     *
     * @param arg1
     * @param arg0
     * @return
     *     returns int
     */
    @WebMethod
    @WebResult(targetNamespace = "")
```

```
    @RequestWrapper(localName = "add", targetNamespace =
"http://service.web_services.example.org/", className =
"org.example.web_services.client.Add")
    @ResponseWrapper(localName = "addResponse", targetNamespace =
"http://service.web_services.example.org/", className =
"org.example.web_services.client.AddResponse")
    @Action(input =
"http://service.web_services.example.org/MathService/addRequest", output =
"http://service.web_services.example.org/MathService/addResponse")
    public int add(
        @WebParam(name = "arg0", targetNamespace = "")
        int arg0,
        @WebParam(name = "arg1", targetNamespace = "")
        int arg1);

    /**
     *
     * @param arg1
     * @param arg0
     * @return
     *     returns int
     */
    @WebMethod
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "divide", targetNamespace =
"http://service.web_services.example.org/", className =
"org.example.web_services.client.Divide")
    @ResponseWrapper(localName = "divideResponse", targetNamespace =
"http://service.web_services.example.org/", className =
"org.example.web_services.client.DivideResponse")
    @Action(input =
"http://service.web_services.example.org/MathService/divideRequest", output =
"http://service.web_services.example.org/MathService/divideResponse")
    public int divide(
        @WebParam(name = "arg0", targetNamespace = "")
        int arg0,
        @WebParam(name = "arg1", targetNamespace = "")
        int arg1);

    /**
     *
     * @param arg1
     * @param arg0
     * @return
     *     returns int
     */
    @WebMethod
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "multiply", targetNamespace =
"http://service.web_services.example.org/", className =
"org.example.web_services.client.Multiply")
    @ResponseWrapper(localName = "multiplyResponse", targetNamespace =
"http://service.web_services.example.org/", className =
"org.example.web_services.client.MultiplyResponse")
```

```
    @Action(input =
"http://service.web_services.example.org/MathService/multiplyRequest", output =
"http://service.web_services.example.org/MathService/multiplyResponse")
    public int multiply(
        @WebParam(name = "arg0", targetNamespace = "")
        int arg0,
        @WebParam(name = "arg1", targetNamespace = "")
        int arg1);

    /**
     *
     * @param arg1
     * @param arg0
     * @return
     *     returns int
     */
    @WebMethod
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "subtract", targetNamespace =
"http://service.web_services.example.org/", className =
"org.example.web_services.client.Subtract")
    @ResponseWrapper(localName = "subtractResponse", targetNamespace =
"http://service.web_services.example.org/", className =
"org.example.web_services.client.SubtractResponse")
    @Action(input =
"http://service.web_services.example.org/MathService/subtractRequest", output =
"http://service.web_services.example.org/MathService/subtractResponse")
    public int subtract(
        @WebParam(name = "arg0", targetNamespace = "")
        int arg0,
        @WebParam(name = "arg1", targetNamespace = "")
        int arg1);

}


package org.example.web_services.client;

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 2.2.9-b130926.1035
 * Generated source version: 2.2
 *
 */
```

```
@WebServiceClient(name = "MathServiceImplService", targetNamespace =
"http://service.web_services.example.org/", wsdlLocation =
"http://localhost:8080/mathservice?wsdl")
public class MathServiceImplService
    extends Service
{

    private final static URL MATHSERVICEIMPLSERVICE_WSDL_LOCATION;
    private final static WebServiceException MATHSERVICEIMPLSERVICE_EXCEPTION;
    private final static QName MATHSERVICEIMPLSERVICE_QNAME = new
QName("http://service.web_services.example.org/", "MathServiceImplService");

    static {
        URL url = null;
        WebServiceException e = null;
        try {
            url = new URL("http://localhost:8080/mathservice?wsdl");
        } catch (MalformedURLException ex) {
            e = new WebServiceException(ex);
        }
        MATHSERVICEIMPLSERVICE_WSDL_LOCATION = url;
        MATHSERVICEIMPLSERVICE_EXCEPTION = e;
    }

    public MathServiceImplService() {
        super(__getWsdlLocation(), MATHSERVICEIMPLSERVICE_QNAME);
    }

    public MathServiceImplService(WebServiceFeature... features) {
        super(__getWsdlLocation(), MATHSERVICEIMPLSERVICE_QNAME, features);
    }

    public MathServiceImplService(URL wsdlLocation) {
        super(wsdlLocation, MATHSERVICEIMPLSERVICE_QNAME);
    }

    public MathServiceImplService(URL wsdlLocation, WebServiceFeature... features)
{
        super(wsdlLocation, MATHSERVICEIMPLSERVICE_QNAME, features);
    }

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

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

    /**
     *
     * @return
     *     returns MathService
```

```
     */
    @WebEndpoint(name = "MathServiceImplPort")
    public MathService getMathServiceImplPort() {
        return super.getPort(new QName("http://service.web_services.example.org/",
"MathServiceImplPort"), MathService.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 MathService
     */
    @WebEndpoint(name = "MathServiceImplPort")
    public MathService getMathServiceImplPort(WebServiceFeature... features) {
        return super.getPort(new QName("http://service.web_services.example.org/",
"MathServiceImplPort"), MathService.class, features);
    }

    private static URL __getWsdlLocation() {
        if (MATHSERVICEIMPLSERVICE_EXCEPTION!= null) {
            throw MATHSERVICEIMPLSERVICE_EXCEPTION;
        }
        return MATHSERVICEIMPLSERVICE_WSDL_LOCATION;
    }

}

// ... Rest of generated code

package org.example.web_services.config;

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

import org.example.web_services.client.MathService;
import org.example.web_services.client.MathServiceImplService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * Configuration class for the MathService.
 * This class provides a Spring bean for the MathService proxy.
 */
@Configuration
public class MathServiceConfig {
    // URL of the MathService WSDL
    private static final String SERVICE_URL = "http://localhost:8080/mathservice?
wsdl";

    /**
```

```
     * Creates a MathService proxy bean.
     *
     * @return an instance of MathService
     * @throws MalformedURLException if the URL is invalid
     */
    @Bean
    MathService mathServiceProxy() throws MalformedURLException {
        return new MathServiceImplService(new URL(SERVICE_URL))
                .getMathServiceImplPort();
    }
}
```

```
package org.example.web_services.ui;

import org.example.web_services.client.MathService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

/**
 * Command Line Interface (CLI) for the MathService.
 * This class executes math operations and prints the results.
 */
@Component
public class MathServiceCLI implements CommandLineRunner {

    // Autowired MathService proxy to consume the web service
    @Autowired
    private MathService mathService;

    @Override
    public void run(String... args) throws Exception {
        // Demonstrating basic math operations using the MathService
        System.out.println("1 + 1 = " + mathService.add(1, 1));
        System.out.println("4 - 2 = " + mathService.subtract(4, 2));
        System.out.println("12 x 3 = " + mathService.multiply(12, 3));
        System.out.println("12 / 4 = " + mathService.divide(12, 4));

        // Handling division by zero
        try {
            System.out.println("12 / 4 = " + mathService.divide(12, 0));
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
        }

    }

}
```

```
package org.example.web_services.main;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication(scanBasePackages = {
        "org.example.web_services.client",
        "org.example.web_services.config",
        "org.example.web_services.ui"
})
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Exemple d'une CLI consommant le service web `EmployeeService`

Dans cet exemple, l'idée consiste à consommer le service web `EmployeeService`. On commence par générer le code Java nécessaire pour les opérations réseaux, d'encodage et de décodage sousjaçent le proxy en utilisant `wsimport` et on importe le paquetage obtenu au sein du projet client. Ensuite, on configure l'application client pour se connecter au service web par le biais du proxy. Enfin, La CLI permet d'interagir avec le service web en affichant des options à l'utilisateur et en traitant ses entrées. La CLI est implémentée de manière similaire à celle du service `MathService`.

```
package org.anonbnr.web_services.employeeservice.client;

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


/**
 * <p>Java class for addEmployee complex type.
 *
 * <p>The following schema fragment specifies the expected content contained
within this class.
 *
 * <pre>
 * &lt;complexType name="addEmployee">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="arg0" type="{http://www.w3.org/2001/XMLSchema}string"
minOccurs="0"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addEmployee", propOrder = {
```

```
    "arg0"
})
public class AddEmployee {

    protected String arg0;

    /**
     * Gets the value of the arg0 property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getArg0() {
        return arg0;
    }

    /**
     * Sets the value of the arg0 property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setArg0(String value) {
        this.arg0 = value;
    }

}


package org.anonbnr.web_services.employeeservice.client;

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


/**
 * <p>Java class for addEmployeeResponse complex type.
 *
 * <p>The following schema fragment specifies the expected content contained
within this class.
 *
 * <pre>
 * &lt;complexType name="addEmployeeResponse">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="return" type="
{http://service.employeeservice.web_services.anonbnr.org/}employee"
```

```
minOccurs="0"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addEmployeeResponse", propOrder = {
    "_return"
})
public class AddEmployeeResponse {

    @XmlElement(name = "return")
    protected Employee _return;

    /**
     * Gets the value of the return property.
     *
     * @return
     *     possible object is
     *     {@link Employee }
     *
     */
    public Employee getReturn() {
        return _return;
    }

    /**
     * Sets the value of the return property.
     *
     * @param value
     *     allowed object is
     *     {@link Employee }
     *
     */
    public void setReturn(Employee value) {
        this._return = value;
    }

}


package org.anonbnr.web_services.employeeservice.client;

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


/**
 * <p>Java class for count complex type.
```

```
 *
 * <p>The following schema fragment specifies the expected content contained
within this class.
 *
 * <pre>
 * &lt;complexType name="count">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "count")
public class Count {


}

package org.anonbnr.web_services.employeeservice.client;

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


/**
 * <p>Java class for countResponse complex type.
 *
 * <p>The following schema fragment specifies the expected content contained
within this class.
 *
 * <pre>
 * &lt;complexType name="countResponse">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="return" type="
{http://www.w3.org/2001/XMLSchema}int"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
```

```
@XmlType(name = "countResponse", propOrder = {
    "_return"
})
public class CountResponse {

    @XmlElement(name = "return")
    protected int _return;

    /**
     * Gets the value of the return property.
     *
     */
    public int getReturn() {
        return _return;
    }

    /**
     * Sets the value of the return property.
     *
     */
    public void setReturn(int value) {
        this._return = value;
    }

}

// ... Remaining generated code

package org.anonbnr.web_services.employeeservice.config;

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

import org.anonbnr.web_services.employeeservice.client.EmployeeService;
import org.anonbnr.web_services.employeeservice.client.EmployeeServiceImplService;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
* Configuration class for the EmployeeService.
*/
@Configuration
public class EmployeeServiceConfig {
     private static final String SERVICE_URL =
"http://localhost:8080/employeeservice?wsdl";

     /**
      * Creates a EmployeeService proxy bean.
      *
      * @return an instance of EmployeeService
      * @throws MalformedURLException if the URL is invalid
      */
     @Bean
     EmployeeService employeeServiceProxy() throws MalformedURLException {
```

```
            return new EmployeeServiceImplService(new
URL(SERVICE_URL)).getEmployeeServiceImplPort();
        }
}

package org.anonbnr.web_services.employeeservice.ui;

import org.anonbnr.web_services.employeeservice.client.Employee;
import
org.anonbnr.web_services.employeeservice.client.EmployeeAlreadyExistsException_Exc
eption;
import
org.anonbnr.web_services.employeeservice.client.EmployeeNotFoundException_Exceptio
n;
import org.anonbnr.web_services.employeeservice.client.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

/**
* Command Line Interface (CLI) for the EmployeeService.
*/
@Component
public class EmployeeServiceCLI implements CommandLineRunner {

    @Autowired
    private EmployeeService employeeService;

    @Override
    public void run(String... args) throws Exception {
        // Example operations
        System.out.println("Number of employees: " + employeeService.count());

        // Add a new employee
        try {
            employeeService.addEmployee("John Doe");
            System.out.println("Added employee: John Doe");
        } catch (EmployeeAlreadyExistsException_Exception e) {
            System.err.println("Error: " + e.getMessage());
        }

        // Get all employees
        employeeService.getEmployees().forEach(emp -> displayEmployee(emp));

        // Get employee by ID
        try {
            Employee employee = employeeService.getEmployee(1);
            displayEmployee(employee);
        } catch (EmployeeNotFoundException_Exception e) {
            System.err.println("Error: " + e.getMessage());
        }

        // Update employee
        employeeService.updateEmployee(1, "John Smith");
```

```
            System.out.println("Updated employee: John Smith");

            // Remove employee
            employeeService.deleteEmployee(1);
            System.out.println("Deleted employee with ID 1");
        }

        private void displayEmployee(Employee employee) {
            System.out.println("ID: " + employee.getId() + ", Name: " +
    employee.getName());
        }
    }
```

```
    package org.anonbnr.web_services.employeeservice.main;

    import org.springframework.boot.SpringApplication;
    import org.springframework.boot.autoconfigure.SpringBootApplication;

    @SpringBootApplication(scanBasePackages = {
            "org.anonbnr.web_services.employeeservice.client",
            "org.anonbnr.web_services.employeeservice.config",
            "org.anonbnr.web_services.employeeservice.ui",
    })
    public class Application {

        public static void main(String[] args) {
            SpringApplication.run(Application.class, args);
        }
    }
```

Une fois que l'application client est configurée et que le serveur est en cours d'exécution, exécutez la CLI. L'utilisateur pourra alors effectuer des opérations sur le service `EmployeeService`, telles que compter le nombre d'employés, récupérer des employés par ID, ajouter, mettre à jour et supprimer des employés. Lors de l'exécution de la CLI, l'utilisateur pourrait voir quelque chose comme ceci :

```
Number of employees: 0
Added employee: John Doe
ID: 1, Name: John Doe
Updated employee: John Smith
Deleted employee with ID 1
```