Création d'une Application RMI Simple avec Spring Boot

Dans cette section, nous allons créer une application HelloWorld pour démontrer comment implémenter l'Invocation de Méthodes Distantes (RMI) en utilisant Spring Boot. Cette application est composée de trois projets : un projet commun contenant les interfaces partagées, un projet serveur qui fournit le service RMI, et un projet client qui consomme ce service.

Projet Commun

Objectif et Structure

Le projet commun contient l'interface partagée qui sera utilisée à la fois par le client et le serveur pour communiquer. Le projet commun a pour but de définir l'interface `HelloService`, qui contient la méthode `helloWorld()`. Cette interface représente le contrat du service RMI que le client et le serveur utiliseront pour communiquer.

```
org
    test
        rmi
            common
                interfaces
                    HelloService.java
```

```java
package org.test.rmi.common.interfaces;

// Interface définissant la méthode de service RMI
public interface HelloService {
    String helloWorld(String msg); // Méthode pour retourner un message de
salutation
}
```

Projet Serveur

Objectif et Structure

Le projet serveur implémente l'interface définie dans le projet commun et expose un serveur RMI qui peut être consommé par les clients RMI. Le projet serveur a pour but d'implémenter l'interface `HelloService` à travers `HelloServiceImpl` et d'exposer ce service à l'aide de `RmiServiceExporter`. À l'initialisation, la classe `Application` démarre l'application Spring Boot, tandis que `RMIServerApplicationRunner` initialise le service RMI et l'enregistre sur le registre RMI à l'écoute sur le port 1099.

```
org
    test
        rmi
            server
                config
                    RMIServerApplicationRunner.java
                impl
                    HelloServiceImpl.java
                main
                    Application.java
```

```java
package org.test.rmi.server.impl;

import org.springframework.stereotype.Service;
import org.test.rmi.common.interfaces.HelloService;

// Implémentation de l'interface HelloService
@Service
public class HelloServiceImpl implements HelloService {

    @Override
    public String helloWorld(String msg) {
        return "Hello " + msg; // Retourne un message de salutation
    }
}
```

```java
package org.test.rmi.server.config;

import java.rmi.Naming;
import java.util.logging.Logger;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.remoting.rmi.RmiServiceExporter;
import org.springframework.stereotype.Component;
import org.test.rmi.common.interfaces.HelloService;
import org.test.rmi.server.impl.HelloServiceImpl;

// Composant qui s'exécute lors du démarrage de l'application
@Component
public class RMIServerApplicationRunner implements ApplicationRunner {

    private HelloServiceImpl helloService;
    private Logger logger = Logger.getLogger(this.getClass().getName());

    // Injection de dépendance de HelloServiceImpl
    public RMIServerApplicationRunner(HelloServiceImpl helloService) {
        this.helloService = helloService;
    }

    @Override
```

```
    public void run(ApplicationArguments args) throws Exception {
        // Liste des services RMI enregistrés
        for (String serviceName : Naming.list("rmi://localhost:1099")) {
            System.out.println("Service RMI enregistré : " + serviceName);
        }
    }

    // Exposition du service HelloService en tant que service RMI
    @Bean
    RmiServiceExporter helloServiceExporter() {
        RmiServiceExporter exporter = new RmiServiceExporter();
        exporter.setServiceName(HelloService.class.getSimpleName()); // Nom du
service
        exporter.setServiceInterface(HelloService.class); // Interface du service
        exporter.setService(helloService); // Implémentation du service
        exporter.setRegistryPort(1099); // Port pour le registre RMI
        logger.info("Serveur RMI démarré sur le port 1099"); // Journalisation du
démarrage du serveur
        return exporter; // Retourne l'exportateur de service RMI
    }
}

package org.test.rmi.server.main;

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

// Classe principale de l'application
@SpringBootApplication(scanBasePackages = {
    "org.test.rmi.server.impl", // Package pour l'implémentation du service
    "org.test.rmi.server.config" // Package pour la configuration
})
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args); // Démarre l'application
Spring Boot
    }
}
```

Projet Client

Objectif et Structure

Le projet client consomme le service fourni par le serveur RMI. La classe `Application` démarre l'application Spring Boot, tandis que `RMIClientApplicationRunner` récupère le proxy du service RMI et invoque la méthode `helloWorld`, affichant le résultat.

```
org
    test
        rmi
            client
```

```
                config
                    RMIClientApplicationRunner.java
                    RMIClientConfig.java
                main
                    Application.java
```

```
package org.test.rmi.client.config;

import java.util.logging.Logger;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.remoting.rmi.RmiProxyFactoryBean;
import org.test.rmi.common.interfaces.HelloService;

// Classe de configuration pour le client RMI
@Configuration
public class RMIClientConfig {
    public final Logger logger = Logger.getLogger(this.getClass().getName());

    // Création d'un proxy pour le service RMI
    @Bean
    RmiProxyFactoryBean proxyFactoryBean() {
        String rmiServerURL = String.format("rmi://localhost:1099/%s",
                HelloService.class.getSimpleName()); // URL du serveur RMI
        logger.info("URL du serveur RMI : " + rmiServerURL); // Journalisation de
l'URL du serveur

        RmiProxyFactoryBean proxy = new RmiProxyFactoryBean();
        proxy.setServiceInterface(HelloService.class); // Définir l'interface du
service
        proxy.setServiceUrl(rmiServerURL); // Définir l'URL du service
        proxy.afterPropertiesSet(); // Initialiser le proxy
        return proxy; // Retourner le proxy
    }
}

package org.test.rmi.client.config;

import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.remoting.rmi.RmiProxyFactoryBean;
import org.springframework.stereotype.Component;
import org.test.rmi.common.interfaces.HelloService;

// Composant qui s'exécute lors du démarrage de l'application
@Component
public class RMIClientApplicationRunner implements ApplicationRunner {

    private RmiProxyFactoryBean proxy; // Proxy pour le service RMI

    // Injection de dépendance du proxy RMI
```

```
    public RMIClientApplicationRunner(RmiProxyFactoryBean proxy) {
        this.proxy = proxy;
    }

    @Override
    public void run(ApplicationArguments args) throws Exception {
        // Récupération de l'instance de HelloService via le proxy
        HelloService service = (HelloService) proxy.getObject();
        // Invocation de la méthode helloWorld et affichage du résultat
        System.out.println(service.helloWorld("Dude"));
    }
}

package org.test.rmi.client.main;

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

// Classe principale de l'application
@SpringBootApplication(scanBasePackages = {
    "org.test.rmi.client.config" // Package pour la configuration du client
})
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args); // Démarre l'application
Spring Boot
    }
}
```

Exécution, Configuration, et Workflow

Exécution et Configuration

1. **Démarrer le Serveur** : Exécutez la classe `Application` du projet serveur. Cela démarrera le serveur RMI et exposera le service à l'adresse `rmi://localhost:1099/HelloService`.
2. **Démarrer le Client** : Exécutez la classe `Application` du projet client. Cela établira une connexion au serveur RMI et invoquera la méthode `helloWorld` avec le message "Dude".

Assurez-vous de bien avoir ajouté le projet commun aux build paths des projets serveur et client pour avoir accès à l'interface `HelloService`.

Workflow de la Requête

1. **Le Client** : Le client, à l'exécution, obtient une référence au service `HelloService` via le proxy créé par `RmiProxyFactoryBean`.
2. **Invocation de la Méthode** : Le client appelle la méthode `helloWorld` du service avec un message. Le proxy intercepte cette invocation et la transmet au serveur RMI.
3. **Le Serveur** : Le serveur reçoit la requête, exécute la méthode `helloWorld` et renvoie la réponse au client.
4. **Retour au Client** : Le client reçoit la réponse et l'affiche à l'écran.