Architectures Distribué - Java RMI

GARCIA PENA Loris

14 octobre 2023

Résumé

Ce rapport présente notre travail réalisé dans le cadre du TP1 portant sur Java RMI (Remote Method Invocation). Le TP1 avait pour principaux objectifs :

— Comprendre la notion d'objet distribué.
— Utiliser le passage de stub ou d'objet sérialisé.
— Explorer le téléchargement de code via l'utilisation de la codebase.

Table des matières

1 Conception

1.1 Introduction

La technologie Java Remote Method Invocation (RMI) est un système puissant qui permet à un objet s'exécutant dans une machine virtuelle Java d'invoquer des méthodes sur un objet s'exécutant dans une autre machine virtuelle Java distante. Cette capacité de communication à distance entre des programmes repose sur l'invocation de méthodes sur des objets distribués appellés stub. En d'autres termes, elle offre la possibilité d'interagir avec un objet distant comme s'il était local. Cela favorise la construction d'applications réparties en utilisant des appels de méthode au lieu d'appels de procédure, simplifiant ainsi le développement d'applications distribuées.

Dans le cadre de ce projet, nous nous plaçons dans le contexte d'un cabinet vétérinaire. Chaque patient du cabinet, c'est-à-dire chaque animal, possède une fiche individuelle avec un dossier de suivi médical. L'objectif est de créer un système où chaque vétérinaire du cabinet peut accéder aux fiches des patients à distance. Nous mettrons en place un serveur et développerons un client pour les vétérinaires.

1.2 Architecture globale

L'architecture globale de notre système repose sur trois composants principaux : le serveur, le client et le module commun (*common*). Cette architecture permet la création d'une application distribuée utilisant Java RMI pour la communication à distance. Chaque composant joue un rôle essentiel dans le fonctionnement harmonieux de notre application vétérinaire.

1.2.1 Serveur

Le composant serveur est responsable de la gestion des fiches médicales des patients (animaux) au sein du cabinet vétérinaire. Il est implémenté en tant que projet distinct, ce qui lui permet de fonctionner indépendamment des clients et de garantir une gestion centralisée des données. Le serveur expose des objets distribués via Java RMI, permettant ainsi aux clients d'accéder aux fiches médicales à distance.

Le serveur utilise les fonctionnalités de Java RMI pour fournir des services aux clients, tels que l'ajout, la recherche et la suppression d'animaux dans les fiches médicales. Il communique avec les clients via des objets distribués et assure la cohérence des données médicales dans la base de données centrale du cabinet vétérinaire.

1.2.2 Client

Le composant client est destiné aux vétérinaires du cabinet. Chaque vétérinaire utilise un client pour accéder aux fiches médicales des patients. Les clients sont des projets distincts, ce qui signifie qu'ils peuvent être déployés sur

les postes de travail individuels des vétérinaires. Les clients utilisent Java RMI pour établir des connexions avec le serveur et interagir avec les fiches médicales de manière transparente.

Les clients offrent une interface conviviale permettant aux vétérinaires de consulter, mettre à jour et gérer les fiches médicales des patients. Ils utilisent des objets distribués pour communiquer avec le serveur, garantissant ainsi l'accès aux informations à distance.

1.2.3 Module commun (*common*)

Le module commun, également appelé *common*, joue un rôle crucial en tant que composant partagé entre le serveur et les clients. Il contient les classes, interfaces et données partagées nécessaires à la communication entre le serveur et les clients. En utilisant ce module, nous garantissons la cohérence des données partagées et simplifions le développement en évitant la duplication de code. Il est essentiel pour assurer la compatibilité et la communication efficace entre les parties du système.

Le module commun contient des définitions d'objets partagés, telles que les classes d'espèces animales, les interfaces de communication et d'autres éléments nécessaires à la synchronisation entre le serveur et les clients. En utilisant le module commun, nous évitons les incohérences et favorisons une intégration fluide des composants.

Cette architecture globale permet une collaboration efficace entre les vétérinaires tout en maintenant une gestion centralisée des fiches médicales des patients. Les clients peuvent accéder aux données de manière sécurisée via le serveur, tandis que le module commun facilite la cohérence des informations partagées entre les composants. Le recours à Java RMI assure une communication à distance transparente, renforçant ainsi notre solution pour le cabinet vétérinaire.

1.3 Composants clés

Pour comprendre en détail les composants clés de notre application Java RMI, examinons les principales classes et interfaces qui les composent.

1.3.1 Composants communs (Côté Common)

Classe Espece

La classe `Espece` est un composant commun qui définit une espèce d'animal. Elle implémente l'interface `Serializable` pour permettre la sérialisation des objets lors de la communication à distance. Cette classe contient des propriétés telles que le nom de l'espèce et la durée de vie moyenne. Elle joue un rôle crucial pour définir les caractéristiques des animaux au sein du système.

Interface IAnimal

L'interface `IAnimal` est une interface distante qui définit les méthodes que tout animal doit mettre en œuvre. Elle étend l'interface `Remote`, ce qui permet aux objets qui la mettent en œuvre d'être distribués via Java RMI. Cette interface définit des méthodes pour obtenir le nom de l'animal, afficher des informations sur l'animal, émettre un cri, consulter le dossier de suivi, obtenir des informations sur l'espèce de l'animal, etc.

Interface ICabinetMedical

L'interface `ICabinetMedical` est le pilier central de notre système. Elle définit les services que le cabinet médical offre aux vétérinaires. Cette interface distante expose des méthodes pour ajouter des animaux, chercher des animaux, supprimer des animaux, obtenir la liste des patients, et bien plus encore. Elle permet également d'enregistrer des callbacks pour les alertes.

1.3.2 Composants serveur

Classe Animal

La classe `Animal` est une implémentation de l'interface `IAnimal`. Elle représente un patient du cabinet vétérinaire et stocke des informations telles que le nom, le maître, l'espèce, la race, le cri et le dossier de suivi de l'animal. Cette classe est utilisée côté serveur pour créer et gérer les fiches médicales des animaux.

Classe CabinetMedical

La classe `CabinetMedical` est le cœur du côté serveur. Elle gère une liste d'animaux (patients) au sein du cabinet médical. Cette classe expose les méthodes définies dans l'interface `ICabinetMedical` pour gérer les patients, rechercher des animaux, supprimer des patients, et plus encore.

1.3.3 Composants client

Classes Ours et Client

Côté client, nous trouvons la classe `Ours` qui hérite de la classe `Espece`. Elle représente une espèce spécifique (l'ours) et est utilisée pour créer des animaux de cette espèce.

La classe `Client` est responsable de l'interaction des vétérinaires avec le cabinet médical. Elle propose plusieurs fonctions, notamment l'affichage des animaux, la recherche, l'ajout et la suppression d'animaux. Les vétérinaires utilisent cette classe pour gérer les fiches médicales des patients du cabinet vétérinaire.

1.3.4 Résumé

Les composants clés de notre application Java RMI comprennent des classes telles que `Espece`, `Animal`, `CabinetMedical`, des interfaces comme `IAnimal` et `ICabinetMedical`, ainsi que des classes spécifiques aux espèces d'animaux

comme `Ours`. Ces éléments interagissent pour permettre la gestion des fiches médicales des patients et la communication à distance entre les vétérinaires et le cabinet médical. Ils jouent un rôle essentiel dans la réalisation des objectifs de notre projet Java RMI.

1.4 Communication

La communication dans notre application Java RMI repose sur le mécanisme de communication à distance offert par RMI. Elle permet aux vétérinaires (clients) d'interagir avec le cabinet médical (serveur) de manière transparente, comme s'ils travaillaient en local.

1.4.1 Invocation de Méthodes à Distance

L'invocation de méthodes à distance est le pilier de la communication dans notre système. Grâce à Java RMI, les vétérinaires peuvent invoquer des méthodes définies dans l'interface `ICabinetMedical` sur des objets distants situés côté serveur. Ces méthodes permettent d'ajouter, chercher, supprimer des animaux, et bien plus encore.

Lorsqu'un vétérinaire appelle l'une de ces méthodes, RMI gère automatiquement la communication à distance. Les paramètres sont sérialisés, les appels sont acheminés vers le serveur, et les résultats sont renvoyés au client de manière transparente.

1.4.2 Interface ICabinetMedical

L'interface `ICabinetMedical` joue un rôle central dans la communication. Elle définit les services offerts par le cabinet médical et permet aux clients d'interagir avec ces services. Cette interface distante est partagée entre le client et le serveur, ce qui assure la cohérence des méthodes disponibles des deux côtés.

De plus, l'interface `ICabinetMedical` expose une méthode permettant d'enregistrer des callbacks pour les alertes. Cette fonctionnalité ajoute une couche de communication asynchrone, permettant au serveur de notifier les clients lorsque certains seuils sont atteints.

1.4.3 Transparence de la Communication

L'un des avantages clés de Java RMI est la transparence de la communication. Les vétérinaires interagissent avec le cabinet médical de la même manière qu'ils le feraient en local. Cette transparence permet de simplifier considérablement le développement d'applications distribuées, car elle masque la complexité liée à la communication à distance.

1.4.4 Résumé

La communication au sein de notre application Java RMI est transparente et simplifiée grâce au mécanisme d'invocation de méthodes à distance. Les vétéri-

naires peuvent accéder aux services du cabinet médical de manière intuitive, tout en bénéficiant de la puissance de la communication à distance. Le rôle central de l'interface `ICabinetMedical` facilite la cohérence des opérations disponibles des deux côtés, et la sécurité peut être renforcée pour garantir la fiabilité des échanges.

1.5 Sécurité

La sécurité est un aspect crucial de toute application distribuée, et Java RMI offre des mécanismes intégrés pour la gestion de la sécurité. Dans notre projet, nous avons commencé à mettre en place des mesures de sécurité de base.

1.5.1 Politique de Sécurité

Nous avons configuré une politique de sécurité en utilisant le fichier `security.policy`. Cette politique définit les autorisations et les restrictions qui seront appliquées aux composants de notre application. La politique de sécurité est chargée via la ligne de code suivante :

```
System.setProperty("java.security.policy", "security/security.policy");
```

Dans le fichier de politique de sécurité, nous avons actuellement octroyé toutes les autorisations à notre application en utilisant la règle suivante :

```
grant {
    permission java.security.AllPermission;
};
```

Cela signifie que notre application a la permission d'accéder à toutes les ressources et d'effectuer toutes les actions, ce qui est une configuration très permissive et ne devrait être utilisée que pour des besoins de développement et de test.

Dans un environnement de production, il est fortement recommandé de définir des politiques de sécurité plus restrictives pour protéger les ressources sensibles et garantir l'intégrité de l'application.

1.5.2 Gestionnaire de Sécurité

Nous avons également mis en place un gestionnaire de sécurité avec la ligne de code suivante :

```
System.setSecurityManager(new SecurityManager());
```

Le gestionnaire de sécurité est responsable de l'application des politiques de sécurité définies. Il veille à ce que les actions de l'application soient conformes aux autorisations spécifiées dans la politique de sécurité.

Bien que notre configuration actuelle soit permissive pour faciliter le développement, il est essentiel de noter que dans un environnement de production, des politiques de sécurité appropriées doivent être définies pour protéger l'application contre les menaces potentielles.

1.5.3 Résumé

La sécurité est une considération importante dans toute application Java RMI. Nous avons configuré une politique de sécurité pour définir les autorisations de notre application et un gestionnaire de sécurité pour les appliquer. La politique actuelle est très permissive, mais il est impératif d'adopter des politiques plus restrictives en environnement de production pour garantir la sécurité de l'application et de ses ressources.

1.6 Diagrammes de conception

[Si nécessaire, incluez des diagrammes pour illustrer la conception.]

1.7 RMI Callback

Dans notre application Java RMI, nous utilisons le mécanisme de callback pour informer les clients des seuils atteints en matière de nombre de patients dans le cabinet médical. Cette fonctionnalité est essentielle pour alerter les vétérinaires lorsqu'un certain nombre de patients est atteint.

Côté client :
Dans le package `com.cabinet.common.rmi`, nous avons défini l'interface `IClientCallback`, qui étend l'interface `Remote`. Cette interface contient une méthode, `notifierSeuilAtteint`, qui est appelée par le serveur pour notifier un client de l'atteinte d'un seuil.

Côté serveur (dans CabinetMedical) :
Nous avons une liste, `listeClients`, qui stocke les clients enregistrés pour les notifications de seuil. Le serveur offre une méthode, `enregistrerAlertCallback`, qui permet d'enregistrer un client pour les alertes. Si un client n'est pas déjà enregistré, il est ajouté à la liste.

Côté client :
Dans le package `com.cabinet.client.rmi`, nous avons implémenté la classe `ClientCallbackImpl` qui étend `UnicastRemoteObject` et implémente l'interface `IClientCallback`. Cette classe fournit l'implémentation de la méthode `notifierSeuilAtteint`, qui affiche une alerte sur la console client indiquant le nombre de patients atteint.

Dans `Client.java`, nous créons une instance de `ClientCallbackImpl` et l'enregistrons auprès du serveur en appelant la méthode `enregistrerAlertCallback` du cabinet médical.

Ce mécanisme de callback permet aux clients d'être informés en temps réel des seuils atteints et d'agir en conséquence.

2 Réalisation