Mini Projet : Implémentation de serveurs et clients en utilisant Java RMI, gRPC et les sockets

1. **Sockets :**Service de chat :
    - Serveur de chat (ChatServeur) :

Le serveur écoute en permanence sur un port spécifique (dans cet exemple, le port 12345) en utilisant un objet ServerSocket.

Lorsqu'un client se connecte, le serveur crée un nouveau thread (ClientHandler) pour gérer la communication avec ce client.

Chaque instance de ClientHandler gère les flux de données (InputStream et OutputStream) associés à un client spécifique.

Pour permettre la communication entre les clients, le serveur maintient une liste de PrintWriter associés à chaque client. Chaque client possède un PrintWriter pour envoyer des messages au serveur.

Le serveur diffuse les messages reçus d'un client à tous les autres clients connectés en les envoyant à tous les PrintWriter dans la liste synchronisée clientWriters.

- Client de chat (ChatClient) :

Le client se connecte au serveur en utilisant l'adresse IP et le port du serveur.

Une fois connecté, le client démarre deux threads :

1. Pour la lecture permanente des messages du serveur et l'Affichage sur la console du client.
2. Le deuxième thread permet au client de saisir des messages à envoyer au serveur via la console et les envoie au serveur via un PrintWriter.

Avantages :

- Contrôle total sur la communication entre les applications.
- Prise en charge de nombreux langages de programmation.
- Flexibilité pour implémenter des fonctionnalités personnalisées.

Limitations :

- Nécessite la gestion manuelle de la sérialisation/désérialisation des données.

- Besoin de mettre en œuvre la logique de gestion des connexions et des threads pour prendre en charge les communications concurrentes.

2.**Java RMI :**Gestion d'une liste de tâches :

TaskListService : Définit les méthodes à distance que le serveur fournira, notamment pour ajouter une tâche, supprimer une tâche et récupérer la liste complète des tâches.

TaskListServiceImpl :Cette classe implémente l'interface distante. Elle maintient une liste de tâches et fournit des implémentations synchronisées des méthodes définies dans l'interface distante pour ajouter, supprimer et récupérer les tâches.

TaskListServer : Instancie l'implémentation du serveur, la lie à un nom spécifique dans le registre RMI et démarre le serveur.

TaskListClient : Ce client se connecte au serveur RMI en utilisant l'adresse et le port spécifiés. Il utilise les méthodes distantes fournies par l'interface developpéepour ajouter quelques tâches, récupérer la liste complète des tâches et les afficher.

Avantages :

- Facilité de mise en œuvre pour les développeurs Java, car il utilise des interfaces Java standard.
- Intégration transparente avec les applications Java existantes.
- Gestion transparente de la communication à distance, ce qui permet aux développeurs de se concentrer sur la logique métier.

Limitations :

- Dépendance à Java.
- Configuration complexe du registre RMI.
- Moins performant que d'autres technologies de communication distribuée en termes de vitesse et de scalabilité.

3.gRPC : Service de messagerie :

- Messaging.proto: Ce fichier contient la définition du service de messagerie gRPC et des messages échangés entre le client et le serveur. Il utilise la syntaxe Protobuf pour définir les messages `Message` et `User`, ainsi que les méthodes `SendMessage` et `GetMessagesForUser` du service `MessagingServic` implémenté dans une autre classe.

- MessagingServer.java: Cette classe implémente le serveur gRPC. Elle fournit les implémentations des méthodes définies dans le fichier de protocole. Par exemple, la méthode `sendMessage` est appelée lorsqu'un client envoie un message, et la

méthode `getMessagesForUser` est appelée pour récupérer les messages d'un utilisateur spécifié.

- MessagingClient.java: Cette classe implémente le client gRPC. Elle utilise le stub généré à partir du fichier de protocole pour appeler les méthodes du service de messagerie sur le serveur. Par exemple, elle appelle la méthode `sendMessage` pour envoyer un message et la méthode `getMessagesForUser` pour récupérer les messages d'un utilisateur.

Avantages :

- Performance élevée grâce à l'utilisation du protocole HTTP/2 et de la sérialisation binaire.
- Support multi-langages, permettant l'interopérabilité entre différentes plates-formes.
- Génération automatique de code à partir du fichier de protocole.

Limitations :

- Courbe d'apprentissage plus prononcée pour les développeurs en raison de l'utilisation de protobuf pour la définition des services.
- Configuration plus complexe des serveurs et des clients par rapport à Java RMI.