01/12/2023

Création d'un serveur chat

Utilisant les sockets et JAVA NIO

AMESGUINE ABDERRAHMANE
UNIVERSITE HASSAN 2 CASABLANCA

Résumé :

Le projet vise à créer un système de chat simple utilisant les sockets et la technologie Java NIO. Il comprend un serveur de chat et un client de chat, permettant aux utilisateurs de communiquer de manière synchrone via un réseau.

Objectifs du Projet :

- Développer un serveur de chat basé sur Java NIO.
- Créer un client de chat permettant aux utilisateurs de se connecter et d'échanger des messages.
- Établir une communication bidirectionnelle entre le serveur et les clients via des sockets.

Implémentation :

Serveur (classe Server)

Le serveur est responsable de gérer les connexions des clients, de recevoir et de transmettre des messages entre les clients. Il utilise la technologie Java NIO pour assurer une gestion efficace des E/S.

- **Connexion des Clients :** Le serveur accepte les connexions des clients et les configure en mode non bloquant.
- **Communication Bidirectionnelle :** La lecture des messages des clients se fait de manière synchrone, permettant au serveur de traiter efficacement les messages entrants.
- **Fonctionnalités Avancées :** Le serveur offre des fonctionnalités telles que la gestion des déconnexions, l'affichage de la liste des clients connectés, l'envoi de l'historique du chat, etc.

Client (classe Client)

Le client est responsable de se connecter au serveur, d'envoyer des messages et de recevoir les messages des autres clients.

- **Connexion au Serveur :** Le client établit une connexion avec le serveur et configure le canal en mode non bloquant.
- **Communication Asynchrone :** Un thread séparé est utilisé pour lire de manière asynchrone les messages du serveur, permettant au client de rester réactif.
- **Fonctionnalités Utilisateur :** Le client permet à l'utilisateur de choisir un nom, de rejoindre le chat, d'envoyer des messages, et offre une option de déconnexion.

Gestionnaire du Serveur

1. **private static final ArrayList<Clients> clients:**
   - Cet attribut est une liste statique (ArrayList) qui stocke les objets de type Clients. La liste maintient une référence à tous les clients actuellement connectés.

2. **private static final LinkedList<String> ChatHistory:**
   - Cet attribut est une liste chaînée (LinkedList) de chaînes de caractères. Il stocke l'historique du chat, c'est-à-dire les messages précédents échangés entre les clients. La liste est utilisée pour envoyer l'historique aux nouveaux clients rejoignant le serveur.

3. **main(String[] args) :**
   - La fonction principale qui initialise et exécute le serveur de chat.
   - Crée un sélecteur, un **ServerSocketChannel**, et configure le serveur pour écouter sur un port spécifique.

4. **connect(SelectionKey key, Selector selector) :**
   - Méthode appelée lorsqu'un client se connecte au serveur.
   - Accepte la connexion, configure le canal du client en mode non bloquant, et l'enregistre auprès du sélecteur pour les opérations de lecture.

5. **join(SocketChannel clientChannel, String message) :**
   - Gère le processus lorsque le client rejoint le serveur en spécifiant un nom.
   - Affiche la liste des clients connectés, envoie l'historique du chat au nouveau client, et informe les autres clients de la nouvelle connexion.

6. **read(SocketChannel clientChannel, byte[] data) :**
    - Traite la lecture des messages envoyés par les clients.
    - Identifie le type de message (join, exit, message normal) et effectue les actions appropriées.

7. **exit(SocketChannel clientChannel, SelectionKey key) :**
    - Gère la déconnexion d'un client.
    - Informe les autres clients de la déconnexion et supprime le client déconnecté de la liste des clients.

8. **broadcastMessage(SocketChannel sender, String message) :**
    - Envoie un message à tous les clients sauf à l'expéditeur.
    - Utilise un buffer pour éviter de répéter le processus d'écriture.

9. **sendChatHistory(SocketChannel clientChannel)**:
    - Envoie l'historique du chat au client qui vient de se connecter.
    - Utilise un buffer pour envoyer chaque message de l'historique.

10. **addToChatHistory (String message) :**
    - Ajoute un message à l'historique du chat.
    - Limite la taille de l'historique à 10 messages.

11. **Nom_client(SocketChannel clientChannel) :**
    - Renvoie le nom associé à un client donné.
    - Utilisé pour identifier l'expéditeur lors de l'affichage des messages dans la console du serveur.

12. **afficher_liste_clients(SocketChannel clientChannel) :**
    - Affiche la liste des clients connectés à un client spécifique.
    - Utilisé lorsqu'un nouveau client rejoint le serveur.

Gestionnaire du Client

1. **main(String[] args) (dans la classe Client) :**
   - La fonction principale du client qui initialise et exécute le client.
   - Établit une connexion avec le serveur, permet à l'utilisateur de saisir un nom, et commence à lire et à envoyer des messages de manière asynchrone.

2. **LireMessages(SocketChannel socketChannel) :**
   - Lit de manière asynchrone les messages du serveur et les affiche à l'utilisateur.

3. **EnvoiyerMessages(SocketChannel socketChannel, String message) :**
   - Envoie un message au serveur.
   - Utilisé pour permettre à l'utilisateur d'envoyer des messages au chat.

Ces fonctions forment la base du gestionnaire du serveur de chat et du gestionnaire du client, facilitant la communication bidirectionnelle entre eux et offrant une expérience utilisateur interactive.

Résultats Obtenu :

Le système de chat fonctionne de manière efficace, permettant aux utilisateurs de se connecter, de communiquer de manière synchrone, et de profiter de fonctionnalités avancées telles que la gestion des déconnexions et l'historique du chat.

Conclusion :

Le projet a atteint ses objectifs en fournissant un système de chat robuste basé sur la technologie Java NIO. Les fonctionnalités avancées offertes améliorent l'expérience utilisateur, et le code est bien structuré pour permettre une maintenance facile à l'avenir.