MEDNINI
Soufyen

Programmation réseau - TP RMI

Actuellement, mon programme marche pour le chat avec le thread, j'ai eu beaucoup de soucis concernant l'affichage des messages (ça bouclait à l'infini etc.) mais j'ai réussi à déboguer. Je suis parti sur une base d'enregistrer les messages dans une arrayList afin d'avoir une sorte d'historique et de ne pas écraser chaque message à chaque fois. À chaque nouveau message, l'index d'une arrayList est modifié et ajoute son message à sa table. Après discussion avec certains de mes camarades, on a trouvé la méthode optimale. J'ai pris beaucoup de retards ne sachant pas par où commencer. Les utilisateurs peuvent choisir leurs noms, et leurs messages, à chaque fois il s'affichera sur la console des autres clients par l'intermédiaire du thread. Pour quitter (fermer la boucle) ils n'ont qu'à écrire « exit ». J'avais seulement commencé à regarder les Callbacks, voulant vraiment faire quelque chose de propre avec les threads. Je me pose également une question, dans mon programme actuel comment aurais-je pu afficher les messages de connexion sur les consoles clients car le programme actuel renvoie un message seulement chez le client concerné. Je me doute que cela doit se passer au niveau du thread. J'ai eu une idée avec l'arrayList Clients à la toute fin, est-ce que cela aurait fonctionné ?

J'avais déjà commencé à répondre aux questions bien avant je vous les laisse ci-dessous, avec des captures d'écrans de mes méthodes.

Q1) c'est difficile de broadcaster car RMI est approprié pour de l'unicast : chaque client a besoin d'un registre qui communique avec le serveur, ce serait inapproprié d'utiliser alors RMI pour plusieurs utilisateurs. De plus chaque client doit renvoyer des réponses au serveur.

Q2) C'est une méthode qui interroge en continu les clients pour vérifier s'ils ont des données à transférer et si ces données les concernent ou non.

Q3) on peut avoir des problèmes si le tableau (ArrayList) vient à être perdu, on perdrait la totalité de la conversation. De ce fait il est important d'implémenter une méthode « synchronized » afin d'avoir toujours un potentiel backup.

Q4)Interface Chat :

```
public interface Chat extends Remote{

    public String messageBienvenue() throws RemoteException;

    public ArrayList<String> message  = new ArrayList<String>();

    public void envoiMessage(String client, String msg) throws RemoteException;

    public ArrayList<String> recupMessage(int pos) throws RemoteException;

    public String getClient(String nom) throws RemoteException;

    public ArrayList<String> Clients = new ArrayList<String>() ;
```

- **Le tableau *message* stockera les messages clients, avec leurs noms.**
- **Le tableau Clients permettra de stocker les différents clients connectés.**

```java
public void envoiMessage(String client, String msg) throws RemoteException {

    message.add(client + " : " + msg);


}
```

Cette méthode ajoute dans le tableau *message*, le nom du client et son message.

```java
public synchronized String getClient(String nom) throws RemoteException{
    String rep = nom + "Un nouvel utilisateur vient d'arriver dans le chat !\n";
    Clients.add(rep);
    System.out.println(rep);
    return rep;
}
```

La méthode getClient prend en paramètre le nom du client et affiche qu'un utilisateur s'est connecté. De plus elle ajoute une valeur dans le tableau Clients (et retourne également un String à utiliser dans la classe Client).

```java
public synchronized ArrayList<String> recupMessage(int pos) throws RemoteException {
    ArrayList<String> msg = new ArrayList<String>();

        for(int i = pos; i< message.size() ; i++) {

            msg.add(message.get(i));
        }

    return msg;

}
```

Cette méthode prend en paramètre la position (en principe une taille de tableau) et incrémente une variable i qui va de cette valeur position à la taille du tableau *message* défini dans l'interface. Dans cette boucle elle ajoute dans le nouveau tableau *msg* les valeurs de *message*, donc le client et le nom. Enfin elle retourne ce tableau msg pour des utilisations extérieures.

```java
public static void main(String args[]) throws Exception {

    try {
            LocateRegistry.createRegistry(1099);
        } catch (RemoteException e) {
        }
        Serveur chatServeur = new Serveur();
        Naming.rebind("//localhost/ChatRMI", chatServeur);
        System.out.println("Serveur prêt!");


}
```

Enfin on crée le main, où l'on définit un nouveau registre sur le port 1099. On map également une nouvelle instance de serveur sur l'adresse //localhost/ChatRMI et affiche « Serveur prêt » dans la console.

Dans la classe PoleThread, on crée le constructeur et une méthode run.
Cette méthode run définie un ArrayList qui affichera msg reçu en prenant en paramètre la taille du premier tableau.

```java
public class PollThread extends Thread{

    ArrayList<String> message;

    Chat serveur;
    int msgCount =0;

    public PollThread(Chat serveur) throws IOException{
        // TODO Auto-generated constructor stub
        this.serveur = serveur;

    }

    public void run(){

        ArrayList<String> msg = new ArrayList<String>();

        while(true) {

            try {

            msg = serveur.recupMessage(message.size());
```

La boucle parcourra le tableau et affichera les messages contenus dans le tableau msg.

De plus on incrémente un compteur pour afficher le nombre de messages envoyés au total.

```java
public Client() throws MalformedURLException, RemoteException, NotBoundException {
    Serveur = (Chat)Naming.lookup("//localhost/ChatRMI");
    sc = new Scanner(System.in);
    try {
        new PollThread(Serveur).start();
    } catch (IOException e) {

        e.printStackTrace();
        System.out.println(e);
    }

}
```

Dans la classe Client, on regarde l'adresse sur laquelle on a map le serveur et on définit un nouveau thread sur ce serveur.

```java
public static void main(String args[]) throws Exception {
    Client chatClient = new Client();
    System.out.println(chatClient.Serveur.messageBienvenue());

    System.out.println("Veuillez entrer votre nom : ");
    String client = chatClient.sc.nextLine();




    System.out.println("Conversation ('exit' pour se déconnecter):");
    System.out.println("Nouvel utilisateur s'est connecté!");
    while(true) {


        String message = null;
        chatClient.Serveur.getClient(client);

        message = chatClient.sc.nextLine();

        if(!message.equals("exit")) {

            chatClient.Serveur.envoiMessage(client,message);
        }
        else {
            System.out.println("Vous avez quitté le chat.");
            System.out.println(client + " s'est déconnecté");
            break;
        }

    }
    System.out.println("Fermeture de la conversation.");
```

La main si dessus, va demander au client son nom et son message. Tant que le message est différent d'exit, le chat continue.