TP sur le modèle client serveur

Yvon Kermarrec

Le but principal de ce TP est de montrer comment on bâtit une session entre client et un serveur et d'introduire la notion de souches (« stubs » en Anglais) utilisées par les RPC, le modèle CORBA et Java RMI.

Ce TP est progressif et il est donc conseillé de suivre les différentes étapes en séquence !

Partie 1 : un service avec UDP

On s'intéresse à un service de base qui fournit l'heure à un client. Pour cela, dans un premier temps, nous allons utiliser les sockets et UDP et vous pouvez récupérer le code complet et les différents programmes Java (DaytimeClient1.java, DaytimeClientHelper1.java et DaytimeServer1.java)

- Consultez le code et représenter les différentes classes avec UML ou un autre formalisme.
- Exécutez le code après l'avoir compilé. Indiquez dans la ligne de commande le numéro du port utilisé par le serveur lors de son lancement. Que se passe t il lorsque vous lancez le client d'abord puis le serveur ? Le serveur puis le client ? pourquoi ?
- Pourquoi avons nous introduit la notion de Helper pour le client ? expliquez en particulier les avantages d'une telle approche en termes de séparation des parties applications et service.
- Pourquoi est il nécessaire pour le serveur d'utiliser la méthode receiveMessageAndSender plutôt que la méthode receiveMessage ?
-

Partie 2 : le même service avec TCP

On s'intéresse cette fois encore au même service que précédemment.

- expliquez les différences avec le code précédent. En particulier expliquez les différences côté serveur pour déterminer qui l'appelle.
- Introduisez artificiellemen,t un délai de service du côté du serveur (avec la commande Thread.sleep (3000) – le thread courant est bloqué pendant 3 secondes). Activez le serveur puis 2 clients. Que se passe t il ? expliquez !

Partie 3 : vers un service complet – une calculette à distance

- compilez dans un premier temps les programmes avec la commande « javac Echo*2.java »
- lancez le serveur puis le client. Le client attend de votre part une série de chaînes de caractères et la dernière est terminée par « . ».
- pourquoi pouvons nous considérer que ce serveur est itératif et en mode connecté ?
- expliquez le rôle du « helper » que l'on définit du côté du client.
- considérez maintenant que la chaîne envoyée vers le serveur est une commande du type « ADD 2 3 » (qui demande la somme de deux nombres). Modifiez le serveur pour qu'il puisse extraire les paramètres et calculer la somme avant de renvoyer le résultat au client. Pour cela vous pourrez vous aider des méthodes Java « split » et « length » pour trouver les différentes parties d'une chaîne séparées par au moins un blanc. Vous compléterez le code en essayant de structurer votre code. Il s'agit de séparer la partie traitement de requêtes de la partie action. Vous pouvez considérer que le serveur fait le traitement de requêtes et que les actions (somme, …) sont des procédures que le serveur appelle.
- Compléter le code de façon à ce qu'il puisse désormais réaliser les multiplications – une commande du type MUL 2 3
- Modifiez le programme côté client pour ne plus lire une chaîne de commande depuis le clavier mais qu'il appelle directement une méthode ADD (2,4) pour demander au serveur l'addition de ces deux nombres (respectivement MUL (2, 3) pour invoquer la multiplication). Utilisez pour cela une approche similaire à celle proposée par la méthode getEcho.
- Que faut il faire si on souhaite enfin considérer une exception déclenchée par le serveur et propagée vers le client ?
- Si vous considérer maintenant d'autres fonctions pour la calculette (Div, sub, pow etc…), pourquoi et comment pourrait on automatiser la génération du code côté client et serveur ?