Polytech Nice - Sophia Antipolis

Techniques Modernes de Programmation Concurrente Elixir

de

BARIOL ALAOUI Salah-Eddine

Novembre 2018

Architecture

Serveur

La première étape est d'appeler KVServer.accept(4040), où 4040 est le port. La première étape de accept/1 consiste à écouter le port jusqu'à ce que le socket devienne disponible, puis à appeler loop_acceptor/1. On peut dire que accept/1 crée une room pour les utilisateurs. loop_acceptor/1 est une boucle qui va attendre l'arrivé d'un nouveau client, le servir et rebouclé en attendant.

serve/1 est une boucle qui lit les messages du clients. serve/1 utilise read_line/1 pour lire les messages, cette dernière fonction utilise gen_tcp.recv/2 pour recevoir les message, de même write_line/2 utilise gen_tcp.send/2 pour écrire au socket.

Gestion d'un crash du serveur

Si notre serveur crash, étant donné qu'il n'y a pas de supervision, nous ne seront pas capable de gérer plus de requête, car le serveur ne sera pas redémarré. Pour cela nous avons besoin d'un arbre de supervision. On utilise la bibliothèque task. Lors du lancement du programme on crée un tache, a l'aide de bibliothèque task qui ouvrira une connexion sur le port 4040.

Connexions simultanées

Si nous essayons de connecter plusieurs utilisateurs en ouvrant plusieurs terminaux, le serveur ne sera capable que de répondre qu'au premier. Quand un client est connecté, nous ne pouvons accepter d'autres clients.

Pour faire en sorte que notre serveur accepte plusieurs connexions, nous avons besoin d'un superviseur qui accepte les connexions et les met en concurrence. L'acception

d'une concurrence se fait dans loop_acceptor. Pour cela on utilisateur un superviseur de la bibliothèque Task qui va accepter temporairement une connexion et l'insérer dans un arbre de supervision.

On inverse ensuite le contrôleur de la relation de supervision pour que ça soit l'enfant, c'est-à-dire l'utilisateur qui soit en charge de la relation, ainsi si la relation est rompu c'est lui qui perd la connexion et non le serveur, qui entraînerait la perte de la connexion pour tous les clients.

Commandes

À partir de la notre architecture est quasiment terminé, on ajoute un système de commande lorsque l'utilisateur se connecte pour s'authentifier en donnant son nom avec /join, pour quitter avec /leave. Pour tous autre messages le serveur utilise la commande /say, sinon il s'agit d'une erreur.

Structure

On utilise une structure Room pour symboliser le lieu où les utilisateurs peuvent se parler entre eux, cette Room contient une liste d'utilisateur, quand un utilisateur s'authentifie on l'ajoute à la room, quand il quitte on l'enlève de la room et on coupe la connexion. Ensuite les utilisateurs, ont un nom et une socket pour la connexion.

Chat

lorsque la commande say est exécuté, on lance la fonction say qui notifie tous les clients excepté l'émetteur.

Compilation

Pour réaliser ce projet, j'ai utiliser mix pour gérer l'architecture du projet pour lancer le programme il suffit de se placer dans le dossier

kv$_umbrella/apps/kv_serverestdelancerlacommande$

PORT=4040 mix run –no-halt

La précision du port n'est pas obligatoire, étant donné que le programme ouvre directement une connexion sur le port 4040.