16 juin 2023

SCALA - Labo 04 Rapport

Bot-Tender : Future

Étudiants : Yanik Lange, Ivan Vecerina
Professeurs : Nastaran Fatemi
Assistant : Christopher Meier

Introduction

L'objectif de ce laboratoire était de compléter le Laboratoire 03 Bot Tender afin de de gérer l'asynchronisme et la concurrence lorsque les commandes avaient une durée de production avant d'être livrées. Ainsi tout produit de type différent pouvait être effectué en parallèle et tout produit du même type en séquentiel. Une commande pouvait ainsi être :

- totalement réussite.
- partiellement réussite.
- pas du tout réussite.

Une fonction nous a été fournie afin de simuler la durée et la probabilité de réussite d'une commande. De plus, l'usage de Future.onComplete ainsi que Await n'était pas autorisé.

Choix d'implémentations

Pour que `AccountService` permet l'accès concurrent nous avons modifié la map des accounts en `TrieMap`, ainsi il est possible mettre à jour les différents comptes de manière atomique pour supporter la concurrence.
Pour la gestion des produits nous avons 2 méthodes qui retourne des `Future[ExprTree]`:

`prepareProducts()`: S'occupe de parcourir l'arbre des expressions jusqu'à ce qu'il arrive sur une feuille produit. Une fois arrivé sur un produit il va préparer un arborescence de produit et retourne un future de cette arborescence contenant les produits qui ont réussi. Pour cela il appelle la 2ème méthode qui suit.

`prepareProductSequentially()`: Cette méthode s'occupe de préparer un plat par incrément à la suite en lui donnant un temps de préparation et un chance de succès fourni par la méthode `randomSchedule()` pour chaque unité de produit. Retourne le future d'un produit qui a réussi.

Nous avons également ajouté le `MessageService` en paramètre à `AnalyzerService` pour pouvoir ajouter les messages depuis l'intérieur de celui-ci. Cela nous paraissait logique car il prenait déjà comme paramètre `ProductService` et `AccountService`.

Le méthode `reply()` a été modifiée afin de prendre en 2ème paramètre un callback pour passer la méthode de notification de changement afin de rafraîchir l'écran après l'ajout d'un message. Ainsi on lui fait passer `notifyConnection()` qui est le callback utilisé pour mettre à jour les messages.

A la fin de la méthode `reply()`, nous avons la String `"Votre commande est en cours de préparation"` qui est retourné en 1er car les futures sont seulement retournées une fois completée.