Reactive Spring

Speakers : Sébastien Deleuze (Pivotal), Brian Clozel (Pivotal)
Format : Conférence
Date : 7 avril 2017

Travaillent tous les 2 sur Spring Framework, et plus particulièrement depuis 2 ans sur le projet Reactive.

Problématique récurrente :

1. Services en attente d'autres services lents
2. Clients mobiles avec beaucoup de latence

Modèle un thread par requête http : tous les I/O sont bloquants. Pendant ce temps, le thread attend. Les threads passent leur temps à attendre. La capacité des serveurs n'est pas utilisé.

Autre modèle : nombre réduit de threads.
Un I/O Selector thread va gérer les écritures/lecture
Plusieurs Worker Threads qui ne vont jamais attendre et toujours travailler.

Passe d'un modèle IO bloquant à du event-based. La latence devient neutre.

Pour aller vers ce modèle, on peut utiliser l'approche réactive. On pourra traiter plus de demandes. Cela augmente la stabilité.

Pierre angulaire de Spring Framework 5 : Reactive Streams. Brique créée avec Netflix, Redhat, TypeSafe ...
Le Subscriber s'abonne au Publisher. C'est lui qui va demander des événement au Publisher. C'est ce que l'on appelle le « back pressure ».
Reactive Streams est simplement 4 interfaces.
RxJava, Reactor et Akka Streams implémentent ces interfaces.

Chez Pivotal, ils travaillent sur **Reactor**.
Interface Flux : publisher de 0 à n éléments.
Interface Mono : publisher de 0 à 1 éléments

Ces interfaces vont être retrouvées dans les API des Repository, Services et Contrôleurs
Exemple du fetchWeather avec un service de backup si le service principal ne répond pas au bout de 2 secondes.
Pour exécuter le pipeline réactif, on appelle la méthode subscribe().

Comparaison API bloquante vs reactive. Exemple d'API UserRepository
La méthode findAll() ne rend la main que lorsque la liste des utilisateurs a été chargée de la base.

En réactif, la méthode findAll() rend la main immédiatement et renvoie un Flux<User>. Ce Flux peut renvoyer les User au fur et à mesure de leur disponibilité (pas besoin de tous les avoir chargé avant de pouvoir utiliser le 1er).

Flux<User> pourrait être un tableau JSON ou un Server Event Stream ou un JSON Event Stream.

Spring Reactive

Spring Framework 5 utilise Reactor pour ses fonctionnalités internes Reactive. Dans nos applications, on a le choix d'utiliser la librairie Reactive de notre choix grâce à l'API Reactive : RxJava, Reactor et Akka Streams.
Les autres projets Spring sont en cours d'évolution pour prendre en compte l'aspect Reactive : Spring Data, Spring Security, Spring Boot, Spring Cloud.

La stack bloquante Spring MVC + Servlet API + Servlet Container reste supportée dans Spring Framework 5.
En complément, on a un nouveau framework web Spring WebFlux qui s'appuie sur Reactive Streams et Servlet 3.1 / Undertow / Netty. Netty est le conteneur par défaut dans Spring Boot.
Le modèle d'annotations de Spring MVC est conservé, mais il est possible d'utiliser des Lambda avec Router functions.
WebFlux gère à la fois du client et du serveur. Partage de l'event loop Netty.
WebClient est un client HTTP réactif qui peut être une alternative au RestTemplate, même depuis Spring MVC. Permet de commencer à introduire du Reactif.

Démo

Utilisation de MongoDB avec un drivers Reactif.
Possibilité de mettre un Flux<User> dans un modèle Spring MVC

Programmation du router par approche fonctionnelle (cf. photo) en Java ou en Kotlin pour gagner en lisibilité.
L'intérêt des routes permet de les centaliser à un unique endroit. On peut néanmoins créer plusieurs classes de routes : admin, api rest, extranet …
Possibilité de « nested » les routes et de définir dynamiquement des routes.

Spring.start.io : choisir Spring Boot 2.0.0-SNAPSHOT + Reactive Web
Exemple de vraies applications sur le GitHub de la conférence Mix-it.