SpringBoot avec Kotlin, Kofu et les Coroutines

Speakers : Sébastien Deleuze (Pivotal)
Format : Conférence
Date : 19 avril 2019

L'objectif de Sébastien est de nous montrer comment convertir une application Spring Boot Java en Kotlin avec Kofu et les coroutines.
Kotlin est supporté par Spring depuis Janvier 2017, peu de temps avec que Google le choisisse comme langage phare pour Android.

4 étapes de live-coding sont nécessaires pour transformer complètement l'application. Néanmoins, on peut s'arrêter là où on le souhaite.

Etape 1 : Java to Kotlin

Sébastien part du guide Building web applications with Spring Boot and Kotlin.
Utilisation volontaire de classes sur @Entity et non de data class car JPA nécessite des objets mutables.
L'ID reste nullable var id : Long?

Sur Spring Boot 2.2, possibilité d'utiliser des ConfigurationProperties immutable (data class).

Kotlin permet d'ajouter des méthodes à une classe existante.
Exemple de l'ajout de la méthode toSlug() à la String.

Etape 2 : WebMVC et JPA vers WebFlux et R2DBC

WebFlux est la stack réactive qu'il faut utiliser avec une stack réactive. JDBC est remplacé par R2DBC. R2DBC est une API que peuvent implémenter les éditeurs de base de données. Il existe d'ores et déjà des drivers pour PostreSQL, MySQL, H2 et MS SQL Server
R2DBC n'a aucune dépendance sur Spring : il est utilisable en Java, Kotlin, Scala …
Utilisation de Spring Data R2DBC qui se rapproche de Spring Data JDBC.
Usage de l'annotation @Query. Pas encore de direct queries (génération de requête à partir du nom).
Usage de DatabaseClient implémenté par les drivers réactifs.

Les contraintes JPA sont levées : on peut utiliser les data class.
Les méthodes du repository et du contrôleur renvoient des Mono et des Flux.

Sébastien ajoute un CommentController qui renvoie des contrôleurs avec de la latence.

Etape 3 : ReactiveX to Coroutines

Il y'a 2 semaines, Kotlin a ajouté les Flow. Ils permettent de gèrer la backpressure à l'aide de suspending functions
Les coroutines sont des threads léger. On n'est pas limité en nombre.

L’appel à exchange() est remplacé par un awaitExchange()

Etape 4 : @nnotations remplacées par Kofu DSL

Kofu est un projet d’expérimentation qui ne doit pas être utilisé en production.
On va passer d’un mode auto-configuration à une configuration explicite.
Spring Fu est un incubateur : le support des co-routines dans Spring vient de Spring Fu.
Jafu est le pendant Java de Kofu qui est en Kotlin. De l’avis de Sébastien, Jafu présente moins d’intérêt que Kofu.

Le contrôleur est remplacé en handler. Les routes n’y sont pas définies.
Création d’un BlogRouter décrivant par programmatique les routes : chemin + reeérence method.
Il ne reste alors plus qu’à supprimer les annotations @RequestMapping.