RxJava les mains dans le code

Date : 8 avril 2015
Format : Hands-on-Lab
Speakers : Simon Baslé et Laurent Doguin de chez Couchbase

Pré-requis :
JDK 8, Maven
git clone https://github.com/simonbasle/practicalRx.git

Contexte

L'objectif de ce Lab consiste à migrer du code legacy vers du RxJava.
L'application legacy (d'un an au tout au plus) utilise Java 8 et Spring Boot.
Elle s'appuie sur des services hebergés sur la machine de Simon
Le repo git contient les branches correspondant aux différentes étapes du lab.
Pas de test unitaire.

Présentation de RxJava

Objectifs :

1. Mettre en pratique RxJava,
2. Appliquer les principaux opérateurs,
3. Migrer une application legacy

Pourquoi RxJava

Besoin d'asynchrone car bloquer c'est mal.
Or, le code asynchrone n'est pas facilement lisible
Difficile avec les API bas niveau de java : Future de Java et Callback, la gestion d'erreur, d'exceptions, annulations, synchro, threading

RxJava vient du monde **.NET**. Ce sont les **Reactive Extension de .NET**
Chez Netflix, ils ont fait son portage en Java et l'ont open-sourcé
Autres portages : RxJS, RxPython …

Principes

Iterable – Iterator et Pull vs Observable / Observer et Push
La donnée deviens un flux, poussé par le consommateur.
Approche plus descriptive.

L'**interface Observable** est un flux connectable à un Observer.
Un Observer a 3 actions : onNext, onError et onCompleted (fin du flux).
Le flux est composable et permet d'être plus descriptif dans ce que l'on veut faire.
Dans un flux, on peut avoir un seul élément (de type scalaire) : onNext suivi d'un onCompleted.
On peut également avoir des séquences finies et des séquences infinies (fondées sur le temporel)

L'application

API Rest pour accéder à la monnaire du Future : Doge Coin.
Couche de contrôleurs REST
Couche de services
Les services sont câblés à une base de données et des services externes.
On souhaite migrer les services vers du Rx afin qu'ils soient observables.

Opérateurs utiles

- Observable.from : prend en entrée une collection en un flux d'éléments
- Observable.just : retourne un flux retournant l'élément passé en paramètre tout le mettant en cache
- Observable.create : permet de créer from scratch un Observable. A nous d'appeler à la main les onNext, onError et onCompleted
- Map : fonction de transformation

Etape 1 : utiliser le Obserbable de RxJava

La signature des services à changer. Les contrôleurs ne compilent plus. Dans un 1er temps, on peut utiliser la méthode toBlocking() et les méthodes suivantes :

- *take*(nombre d'éléments à garder)

- *single* : l'observable garantie qu'un seul élément doit être renvoyé par l'Observable. Si 0 ou plus, une exception est levée.
- *singleOrDefault*(default value)

Transformer

Besoin : somme de la capacité de hashing de chaque utilisateur

Opérateurs utiles :

- flatMap : fonction de transformation renvoyant un Observable. Remet à plat avec mélange possible des résultats.
- reduce : permet de sommer les capacités

Filtrer

Services à migrer :
- UserService
    - findAll sera pour l'instant adapté naïvement
    - composer sur findAll pour findById / findByLogin
- SearchService

Opérateurs utiles :
- Observable.just : permet de renvoyer N éléments
- filter : on passe une fonction prédicat à un flux source
- take
- flatMap
    - pour récupérer en asynchrone des données supplémentaires pour le filtre
- singleOrDefault(null) permet de ne pas casser l'API existante utiliser par les contrôleurs et qui se base sur la valeur null.

Effet de bord

Besoin : logger des étapes intermédiaires
- doOnNext
- doOnError
- doOnCompleted
- doOnEach

Ajouter une ligne de log à chaque fois qu'un utilisateur se connecte dans PoolService

Combinaisons

- concat : prend N flux en entrée et en renvoie qu'un seul. L'ordre est respecté.
- merge : les items sont mergés au fil de l'eau. Mélange possible.
- zipWith : combine les éléments de plusieurs flux. Le nombre d'éléments des flux doit être identique.

Cas d'usage sur le service StatService.getAllStats

Opérateur retry

2 variantes d'Observable : à chaud et à froid.
Un Observable froid induit systématiquement à un appel réseau.
L'Obserable chaud permet de faire du cache

Opérateur OnErrorReturn

Permet de renvoyer une valeur par défaut en cas d'erreur.

Réponse Asynchrone

- Retourner un DeferredResult<T>
- Subscribe sur les flux
    - onNext : injecter le résultat via setResult
    - Utiliser le zip, flatMap, single pour détecter les mauvais utilisateurs

Mise en œuvre sur le AdminController

Etape 8

Mise en place d'une solution de secours : appel d'une API payante si l'API gratuite ne fonctionne pas. Tolérance aux pannes.

ExchangeRateService

- Appelle deux API externes

OnErrorResumeNext

Opérateur RxJava qui sait basculer vers un flux de secours en cas d'erreur.