RxJava 2 en Reactive Streams

De belangrijkste veranderingen

RxJava is een library die als doel heeft om asynchrone code beter leesbaar en onderhoudbaar te maken. Inmiddels is de compleet vernieuwde versie 2 uitgebracht, die voldoet aan de Reactive Streams standaard. In dit artikel lees je wat de belangrijkste veranderingen van RxJava zijn ten opzichte van versie 1.

De Observable

Het overgrote deel van de applicaties die wij bouwen, heeft interactie met de buitenwereld. Bij interactie met bronnen als databases, REST services, bestanden of gebruikers moet er altijd rekening worden gehouden dat er vertraging kan optreden in de reactie en dat een antwoord zelfs kan uitblijven. Denk hierbij bijvoorbeeld aan netwerk latency of zelfs errors. In je applicatie moet je er dus rekening mee houden dat deze externe calls altijd asynchroon verlopen. Een aanpak hiervoor, die steeds meer populariteit geniet, is reactive programming.

Voor het schrijven van leesbare asynchrone code kunnen we gebruik maken van ReactiveX (**http://reactivex.io**). ReactiveX is een library die beschikbaar is voor een grote hoeveelheid programmeertalen, waaronder Scala, JavaScript en Java (RxJava). In Java Magazine #1 van 2015 hebben we al een introductie gegeven van RxJava. Inmiddels is RxJava 2 uitgebracht. De grootste verandering is dat deze de Reactive Streams API (**www.reactive- streams.org**) implementeert. In dit artikel kijken we naar de nieuwe functies van RxJava 2. Voor we daar mee aan de slag gaan, geven we in **Listing 1** nog een voorbeeld van hoe je in RxJava met behulp van de Observable data van verschillende asynchrone bronnen kunt combineren om tot een resultaat te komen. Zonder RxJava en de Observable hadden we gebruik moeten maken van callbacks. Dat zou veel onleesbaardere code hebben opgeleverd.

De Observable is het centrale concept binnen RxJava. Een Observable is een potentieel oneindige lijst data, waarop we één of meerdere subscribers kunnen registeren. De subscribers verwerken vervolgens de items in de stroom één voor één. In het voorbeeld in **Listing 1** halen we uit onze repository alle personen op met naam de “Erwin”. We gebruiken vervolgens de filtermethode om alleen de personen met een leeftijd van 65 jaar of ouder te selecteren. Van elke persoon halen we het inkomen op. Dit is een asynchrone operatie. We willen van personen ouder dan 65 hun inkomen ophalen. Hiervoor combineren we het resultaat van de eerste Observable met een tweede Observable. Hiervoor gebruiken we de flatMap operatie.

```
//PersonRepository

public Observable<Person> findByName(String name);
public Observable<BigDecimal> getIncome(Person person);

//Aanroep
repository.findByName(“Erwin”).filter(person -> person.getAge() >= 65)
                .flatMap(person -> repository.getIncome(person))
                .subscribe(income -> totalIncome = totalIncome.add(income));
```

Listing 1

Erwin de Gier is software architect en trainer bij de business line Open Source van Sogeti.