SIEMENS

Übungen zu

Reactor

Reactive-Streams

Marwan Abu-Khalil
2022

SIEMENS

Übungen zu Kapitel 1: Reactive-Streams Standard

- **Rc 1.1 Minimal Reactor-Stream**
  Einen ganz einfachen Reacor Stream bauen

- **Rc 1.2 Subscriber API und Lifecycle**
  Subscriber Interface implementieren

- **Rc 1.3 Publisher, Subscription, Subscriber bauen**
  Komplette Pipeline programmieren, Regeln sicherstellen

SIEMENS

Übungen zu Kapitel 2: API

- **Rc 2.1 Einfache Streams:** range(), map(), subscribe() — Einfach
- **Rc 2.2 Subscriber API in Reactor**: LambdaSubscriber und BaseSubscriber — Mittel
- **Rc 2.3 Mehrere Subscriber**: Wann laufen sie parallel? — Mittel
- **Rc 2.4 Subscriber Decoupeled**: Subscriber läuft parallel zur Source — Schwer
- **Rc 2.5 flatMap()-groupBy():** Anwenden und parallelisieren — Schwer

Übungen zu Kapitel 3: Data

SIEMENS

- **Rc 3.1 Backpressure API**: Durch Backpressure Exceptions beheben — Mittel
- **Rc 3.2 Generator API:** create() und generate() — Einfach

SIEMENS

Übungen zu Kapitel 4: Parallelität

- **Rc 4.1 subscribeOn(), publishOn() Funktionsweise:** Parallelelisiere einen Stream — Einfach
- **Rc 4.2 subscribeOn(), publishOn() Performance**: Beschleunigung durch Parallelität — Mittel
- **Rc 4.3 Scheduler Gutfall vs. Schlechtfall**: Laufzeitveränderung durch Scheduler — Schwer
- **Rc 4.4 subscribeOn() vs. publishOn():** Performance Verhaltensunterschied — Mittel
- **Rc 4.5 parallel()-sequential() Speedup:** Performance Automatische Parallelität — Mittel

Rc 1.1: Minimal Rx-Stream
Einen ganz einfachen Reactor Stream bauen

Einfach

SIEMENS

Bauen Sie einen einfachen Stream mit RxJava.

Ein Flux versendet die Zahlen 1-100, ein Subscriber empfängt sie, und schreibt sie auf die Shell.

- **Hinweis:**
  - Flux.range(1, 100) gibt ein Flux zurück, das die Daten emittiert
  - Flux.subscribe(System.out::printl) realisiert einen Subscriber, der die Daten auf die Shell schreibt**.**

- **Lernziel:** Stream Programmierung kennenlernen

- **Musterlösung:** seminar_reactor.exercises.rc_1_standard.rc_1_1_minimal_stream

Rc 1.2 Subscriber API und Lifecycle
org.reactivestreams.Subscriber implementieren

Mittel

SIEMENS

Programmieren Sie einen Subscriber.
Implementieren Sie dafür das Interface org.reactivestreams.Subscriber.

a) Subcribieren Sie Ihren Subscriber an ein Reacor Flux.range(1,100)

b) Zeigen Sie durch Shell-Ausgaben, wann welche Subscriber Methoden aufgerufen werden.

HINWEIS:
- Sie müssen sich die Subscription „merken“, die im onSubscribe() Aufruf übergeben wird
- Sie müssen Subscription.request() aufrufen, damit Daten ausgesendet werden.

- **Lernziel:** Subscriber API des Reactive-Streams Standards und deren Lifecycle kennenlernen

- **Musterlösung:** seminar_reactor.exercises.rc_1_standard.rc_1_2_subscriber_api

Rc 1.3 Publisher, Subscription, Subscriber bauen
Komplette Pipeline programmieren

SIEMENS

Schwer

Implementieren Sie die drei Interfaces aus der Klasse concurrent.Flow (Publisher, Subscription, Subscriber) und bauen Sie aus diesen eine Pipeline, die einen unendlichen Datenfluss verarbeitet.

In der Ausgangsbasis zur Übung ist die Pipeline vorhanden. Beheben Sie folgende Fehler:

a) In der Methode MySimpleSubscription.request() werden onNext() Aufrufe synchron abgesetzt, das führt zu einem Stack Overflow. Dieses Verhalten widerspricht der Regel 3.3: „Subscription.request MUST place an upper bound on ... recursion between Publisher and Subscriber."
Beheben sie dies durch Nutzung eines Executors (das ist ein Thread-Pool)

b) In der Methode MySipmleSubscriber.onNext(Integer item) wird bei jedem Aufruf nur ein einziges Element über den Aufruf request(1) angefordert.
Dies widerpricht der Empfehlung aus Regel 1.1, da es ein sogenanntes Stop-And-Wait Protokoll implementiert, was nicht effizient ist.
Beheben Sie dies, indem Sie einen Batch neuer Items bestellen (request(BUFFER_SIZE)).

- **Lernziel:** Leistung der Pipeline-Implementierung hinsichtlich Nebenläufigkeit erkennen
- **Musterlösung:** seminar_reactor.exercises.rc_1_standard.rc_1_3_pipeline_impl

Siehe auch: https://docs.oracle.com/javase/9/docs/api/java/util/concurrent/Flow.html

Rc 2.1 Übung: Einfache Streams in Reactor

Einfach

SIEMENS

- **a) Stream mit wenigen Stufen:** Baue einen einfachen Stream aus: range(), map(), subscribe(). Zeige druch Shell Ausgaben, was die einzelnen Stufen tun.
  - Hinweise

```
map( i -> {
    System.out.println("mapping i " + 1);
     return i+1;
  });
doOnNext()
```

- **b) In welchen Threads** laufen die Stream-Stufen? Laufen sie parallel oder nacheinander?
  - Hinweis: .map(value -> {System.***out***.println(Thread.*currentThread*(); **return** value;});

- **c) Summe:** Baue einen Stream aus den Stufen range(), map(), filter(), subscribe(), wobei der Subscriber die geraden Zahlen von 0-100 aufsummiert.

Lernziel: Stream API benutzen können und mit Shell-Ausgaben debuggen.

Musterlösung: seminar_reactor.exercises.rc_2_api.rc_2_1_basic_streams

Rc 2.2 Übung
Subscriber API in Reactor

Mittel

SIEMENS

a) Registriere einen Subscriber mittels eines Lambdas. Dieses schreibt das Item sowie den Thread in dem es ausgeführt wird auf die Shell.

b) Programmiere einen einfachen Subscriber mithilfe der Klasse BaseSubscriber. Überschreibe die hookOnNext(), und hookOnSubscribe() Methode. Mache es Dir leicht, indem Du in jedem onNext() Call ein neues Element anforderst (Stop-And-Wait Protokoll)

c) (**Schwer**) Konstruiere einen realistischen Subscriber mithilfe der Klasse Base-Subscriber. Für jedes Item, das in onNext(T item) übergeben wird, wird eine (ggf. länger laufende) Methode aufgerufen. Lagere diese langlaufenden Aufgaben in andere Threads aus (z.B. Executor-Service verwenden). Fordere neue Daten in einem Batch an, das heißt, rufe Subscribe.request(BATCH_SIZE) mit einer BATCH_SIZE z.B. von 32 auf. Führe Buch, wann diese 32 Items verarbeitet sind, und fordere erst dann einen Batch neuer Items an (siehe rc_2_data.subscriber_backpressure)

Lernziel: Die Subscriber API mit Lambda-Subscriber und Base-Subscriber kennenlernen

Lösung: seminar_reactor.exercises.rc_2_api.rc_2_2_subscriber_api

Rc 2.3 Übung Mehrere Subscriber

Mittel

SIEMENS

Baue einen Stream mit mehreren Subscribern und untersuche unter welchen Bedingungen diese gleichzeitig oder nacheinander mit Daten versorgt werden

Hinweis:

- scheduleOn(Schedulers.*parallel*()) parallelisiert die Ausfuehrung eines Streams.
- Falls das Programm sich beendet, ohne dass etwas geschieht, nutze sleep(3000) um kurz zu warten, und überlege, warum das notwendig ist.

- **Lernziel**: Verstehen, wie man Subscriber zueinander parallelisiert
- **Muserlösung**: seminar_reactor.exercises.rc_2_api.rc_2_3_multiple_subscribers

Rc 2.4 Übung Subscriber Decoupled

SIEMENS

Schwer

Konsturiere einen Stream, bei dem die Aufgaben, die der Subscriber in seiner onNext() erledigt, in einem anderen Thread ausgefuehrt werden als die Stream-Source. Dabei soll folgendes Verhalten realisiert werden

- Subscriber läuft in einem anderen Thread als die Source
- Items werden in Batches bei der Source bestellt

a) Realisere dies, ohne dass Du einen eigenen Subscriber implementierst.

- **Hinweise**:
  - publishOn() entkoppelt Publisher und Subscriber
  - Zeige das Verhalten Deines Streams mit den Log-Methoden doOnNext() und doOnRequest()

b) Baue einen Subscriber mithilfe der Klasse Base-Subscriber der folgendes leistet:

- Wenn ein Item in onNext(T item) eintrifft, lagere die Verarbeitung dieses Items in einen anderen Thread aus. Verwende dafür z.B. einen Thread Pool (ExecutorService).
- Fordere neue Daten in einem Batch an, das heißt, rufe Subscribe.requst(BATCH_SIZE) mit einer BATCH_SIZE z.B. von 32 auf.
- Führe Buch, wann diese 32 Items verarbeitet sind, und fordere erst dann neue Items an.

Lernziel: publishOn() und Subscirber-Implementierung kennenlernen

Musterlösung: seminar_reactor.exercises.rc_2_api.rc_2_4_subscriber_decoupled

Rc 2.5 Übung flatMap()- groupBy()

Schwer

SIEMENS

a) Zerlege die Zahlen 1..100 in vier Gruppen, so dass jede Gruppe die Zahlen enthält, die denselben Wert modulo 4 haben. Zeige durch Shell-Output welche Zahl in welcher Gruppe ist.

Hinweis: groupBy() und x -> x%4

b) Füge die Gruppen aus a) so zusammen, dass alle Zahlen wieder in einen gemeinsamen Stream gemerged werden.

Hinweis: flatMap()

c) Parallelisiere die Lösung aus b) so, dass die Elemente verschiedener Gruppen parallel verarbeitet werden. Zeige die Parallelität durch Shell Ausgaben

Hinweis: publishOn(Schedulers.parallel())

Lernziel: flatMap() und groupBy() Anwenden können und deren Relevanz für Parallelität verstehen

Musterlösung: seminar_reactor.exercises.rc_2_api.rc_2_5_flatmap_groupby

Übung Rc 3.1: Reactor Backpressure API

SIEMENS

Mittel

Der Stream in dieser Aufgabe versendet große Items aus einer Source, die nicht Backpressure aware ist. Das kann zu Problemen führen:

a) Verändere den Stream durch Backpressure so, dass es zu einem Absturz kommt (OutOfMemory) indem Du im Subscriber Backpressure erzeugst.
Hinweis: Subscription.request() in den Subscriber einbauen

b) Löse das Problem indem Du eine Backpressure-Stage in den Stream einbaust
Hinweis: onBackpressureBuffer(100)

c) Löse das Problem, indem Du der Source eine OverflowStrategie mitgibst.
Hinweis: create() mit OverflowStrategy.DROP

Lernziel: Notwendigkeit von Backpressure verstehen und API dafür kennenlernen
Musterlösung:seminar_reactor.exercises.rc_3_data.rc_3_1_backpressure_api

Übung Rc 3.2 Reactor Generator API

Einfach

SIEMENS

a) Baue einen Genarator mit generate() der Zufallszahlen emittiert

b) Baue einen Generator mit generate() der ein State Objekt benutzt und die Folge 1..10 emittiert

c) Baue einen Generator mit create() der die Folge 1..10 emittiert

d) Baue einen Generator mit create() der asynchone Events (Button-Clicks) in einen Stream einspeist

Lernziel: Stream Sources mit create() und generate() bauen können

Musterlösung: seminar_reactor.exercises.rc_3_data.rc_3_2_generator_api

Rc 4.1 Übung
subscribeOn(), publishOn() Funktionsweise

SIEMENS

Einfach

Zeige, dass subscribeOn() und publsihOn() dazu führen, dass Stream-Stufen in unterschiedlichen Threads laufen.

a) Concurrency mit publishOn(): Funktionsweise

- Baue einen Stream mit zwei Stufen
- Entkoppele diese Stufen durch publishOn()
- Zeige durch Shell-Ausgaben, dass die Stufen nebenläufig in unterschiedlichen Threads laufen
- **Musterlösung**: seminar_reactor.exercises.rc_4_parallel.rc_4_1_a_publishon_behavior

b) Parellelität mit subscribeOn(): Funktionsweise

- Baue einen Stream mit zwei Subscribern
- Erkenne, dass erst der eine Subscriber bis zu Ende läuft und danach der andere
- Entkopple die Subscriber durch subscribeOn()
- Zeige durch Shell-Ausgaben, dass nun beide Subscriber gleichzeitig laufen, in unterschiedlichen Threads
- **Musterlösung**: seminar_reactor.exercises.rc_4_parallel.rc_4_1_b_subscribeon_behavior

Rc 4.2 Übung
subscribeOn(), publishOn() Performance

SIEMENS

Mittel

Zeige, wie mit die Perfomance eines Streams mit publishOn() oder subscribeOn() verbessert werden kann.

a) Performance durch Concurrency mit publishOn()

- Baue einen Stream mit mit zwei Stufen die langsam sind (CPU intensiv oder blokierend)
- Entkoppele die Stufen durch **publishOn()**
- Zeige, dass der Stream nun schneller läuft
- **Musterlösung**: seminar_reactor.exercises.rc_4_parallel.rc_4_2_a_publishon_speedup

b) Performance duch Parallelität mit subscribeOn()

- Baue einen Stream mit zwei langsamen Subscribern
- Entkopple die Subscriber durch **subscribeOn()**
- Zeige, dass der Stream nun schneller läuft
- **Musterlösung**: seminar_reactor.exercises.rc_4_parallel.rc_4_2_b_subscribeon_speedup

Rc 4.3 Übung
Scheduler Gutfall vs. Schlechtfall

Schwer

SIEMENS

1. Verbessere die Laufzeit der einzelnen Jobs in den Clients eines Streams durch die Wahl eines geeigneten Schedulers

2. Verbessere die Gesamtlaufzeit eines Streams durch die Auswahl eines geeigneten Schedulers

Lernziel: Verhaltensunterschiede der Scheduler kennenlernen

Musterlösung: seminar_reactor.exercises.rc_4_parallel.rc_4_3_schedulers_goodcase_badcase

Übung Rc 4.4
subscribeOn() vs. publishOn()

Mittel

SIEMENS

Der Stream in diesem Beispiel ist bereits parallelisiert, aber er ist dennoch langsam. Finde eine Möglichkeit, ihn auf andere Weise zu parallelisieren, so dass er schneller wird. Ziel: die Laufzeit soll um 50% sinken.

Frage: Warum ist Deine Parallelisierung schneller als die ursprüngliche?

Hinweis: subscribeOn()

Lernziel: Semantik und Verhalten von subscribeOn() in Abgrenzung zu publishOn() verstehen.

Musterlösung: seminar_reactor.exercises.rc_4_parallel.rc_4_4_subscribeon_vs_publishon

Übung Rc 4.5
parallel()-sequential() Speedup

Mittel

SIEMENS

a) Beschleunige den Stream in dieser Aufgabe indem Du die ihn deklarativ parallelisierst, unter Verwendung der Operatoren parallel() sequential() und runOn().

b) Zeige durch Shell Ausgaben, dass die Map-Aufrufe parallel ablaufen.

Lernziel: Nutzen und Anwendung von parallel()-sequential() verstehen

Musterlösung: seminar_reactor.exercises.rc_4_parallel.rc_4_5_para_seq_speedup