Key Transforming Operators in the Observable Class (Part 3)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Recognize key Observable operators
- Factory method operators
- Transforming operators
  - Transform the values and/or types emitted by an Observable
- Understand the RxJava flatMap() concurrency idiom

```
return Observable
  .fromIterable(bigFractionList)
  .flatMap(bf -> Observable
    .fromCallable(() -> bf
      .multiply(sBigFraction))
    .subscribeOn
      (Schedulers
        .computation()))
  .reduce(BigFraction::add)
...
Learning Objectives in this Part of the Lesson

• Recognize key Observable operators
  • Factory method operators
• Transforming operators
  • Transform the values and/or types emitted by an Observable
  • Understand the RxJava flatMap() concurrency idiom
• How how to compare & contrast flatMap() & map()
The RxJava flatMap() Concurrency Idiom
The RxJava flatMap() Concurrency Idiom

- flatMap()’s often used when each item emitted by a stream needs to apply its own threading operators

```java
return Observable
  .fromIterable(bigFractions)
  .flatMap(bf -> Observable
          .fromCallable(() -> bf
                        .multiply(sBigFraction)))
  .subscribeOn
  (Schedulers
  .computation())
  .reduce(BigFraction::add)
...
```

See Reactive/Observable/ex3/src/main/java/ObservableEx.java
The RxJava flatMap() Concurrency Idiom

• flatMap()’s often used when each item emitted by a stream needs to apply its own threading operators
• This structure is known as the “flatMap() concurrency idiom”

```java
return Observable
    .fromIterable(bigFractions)
    .flatMap(bf -> Observable
        .fromCallable(() -> bf
            .multiply(sBigFraction)))
    .subscribeOn
    (Schedulers
        .computation())
    .reduce(BigFraction::add)
...
```

See dzone.com/articles/rxjava-idiomatic-concurrency-flatmap-vs-parallel
The RxJava flatMap() Concurrency Idiom

- `flatMap()`'s often used when each item emitted by a stream needs to apply its own threading operators.
- This structure is known as the "flatMap() concurrency idiom".

Create an Observable BigFraction stream from a BigFraction List

```java
return Observable
    .fromIterable(bigFractionList)
    .flatMap(bf -> Observable
        .fromCallable(() -> bf
            .multiply(sBigFraction))
        .subscribeOn(Schedulers.computation()))
    .reduce(BigFraction::add)
...
The RxJava flatMap() Concurrency Idiom

- flatMap()’s often used when each item emitted by a stream needs to apply its own threading operators
- This structure is known as the “flatMap() concurrency idiom”

```
return Observable
  .fromIterable(bigFractionList)
    .flatMap(bf -> Observable
      .fromCallable(() -> bf
        .multiply(sBigFraction))
    
      .subscribeOn
          (Schedulers
            .computation())

    .reduce(BigFraction::add)
...
The RxJava `flatMap()` Concurrency Idiom

- `flatMap()`’s often used when each item emitted by a stream needs to apply its own threading operators
- This structure is known as the “`flatMap()` concurrency idiom”

```java
return Observable
    .fromIterable(bigFractionList)
    .flatMap(bf -> Observable
        .fromCallable(() -> bf
            .multiply(sBigFraction))
        .subscribeOn
            (Schedulers
                .computation()))
    .reduce(BigFraction::add)
...
```

Each BigFraction in the stream is processed concurrently in a pool of worker threads
The RxJava flatMap() Concurrency Idiom

- flatMap()’s often used when each item emitted by a stream needs to apply its own threading operators.
- This structure is known as the “flatMap() concurrency idiom”.

```
return Observable
  .fromIterable(bigFractionList)
  .flatMap(bf -> Observable
    .fromCallable(() -> bf
      .multiply(sBigFraction))
    .subscribeOn
      (Schedulers.computation()))
  .reduce(BigFraction::add)
...
```

"Lazily" emit a Callable that multiplies two BigFraction objects in a nested Observable.
The RxJava `flatMap()` Concurrency Idiom

- `flatMap()`'s often used when each item emitted by a stream needs to apply its own threading operators.
- This structure is known as the "`flatMap()` concurrency idiom".

```java
return Observable
  .fromIterable(bigFractionList)
  .flatMap(bf -> Observable
    .fromCallable(() -> bf
      .multiply(sBigFraction)
    )
  )
  .subscribeOn
  (Schedulers
    .computation()())
  .reduce(BigFraction::add)
..."
The RxJava flatMap() Concurrency Idiom

flatMap()’s often used when each item emitted by a stream needs to apply its own threading operators.

This structure is known as the “flatMap() concurrency idiom”

```java
return Observable
  .fromIterable(bigFractionList)
  .flatMap(bf -> Observable
    .fromCallable(() -> bf
      .multiply(sBigFraction)))
  .subscribeOn
    (Schedulers.computation())
  .reduce(BigFraction::add)
...```

After all the concurrent processing completes then add all the Big Fractions to compute the final sum.

See upcoming lesson on `Key Combining Operations in the Observable Class (Part 2)`
Comparing Observable map() & flatMap()
Comparing Observable map() & flatMap()

• The map() vs. flatMap() operators

See en.wikipedia.org/wiki/Rock 'Em_Sock 'Em_Robots
Comparing Observable map() & flatMap()

- The map() vs. flatMap() operators
- map() transforms each value in an Observable stream into one value

See medium.com/mindorks/rxjava-operator-map-vs-flatmap-427c09678784
Comparing Observable `map()` & `flatMap()`

- The `map()` vs. `flatMap()` operators
- `map()` transforms each value in an Observable stream into one value
  - e.g., used for synchronous 1-to-1 transformations

The # of output elements equal the # of input elements
Comparing Observable map() & flatMap()

- The map() vs. flatMap() operators
  - map() transforms each value in an Observable stream into one value
  - flatMap() transforms each value in an Observable stream into an arbitrary number (0+) values

See medium.com/mindorks/rxjava-operator-map-vs-flatmap-427c09678784
Comparing Observable map() & flatMap()

- The map() vs. flatMap() operators
  - map() transforms each value in an Observable stream into one value
  - flatMap() transforms each value in an Observable stream into an arbitrary number (0+) values
  - e.g., intended for asynchronous 1-to-N transformations

The # of output elements may differ from the # of input elements
Comparing Observable map() & flatMap()

- The map() vs. flatMap() operators
  - map() transforms each value in an Observable stream into one value
  - flatMap() transforms each value in an Observable stream into an arbitrary number (0+) values
  - flatMap() is used extensively in RxJava

POPULAR
End of Key Transforming Operators in the Observable Class (Part 3)