Key Transforming Operators in the Flux Class (Part 2)

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 Flux operators
  - Factory method operators
- Transforming operators
  - Transform the values and/or types emitted by a Flux
    - e.g., flatMap()
Learning Objectives in this Part of the Lesson

• Recognize key Flux operators
• Factory method operators
• Transforming operators
  • Transform the values and/or types emitted by a Flux
  • e.g., flatMap()

return Flux
  .fromCallable(() -> BigFraction
      .reduce(unreducedFraction))
  .subscribeOn(scheduler)
  .flatMap(reducedFraction ->
    Flux
      .fromCallable(() ->
          reducedFraction
          .multiply(sBigReducedFrac)
      .subscribeOn(scheduler));

This lesson also describes the Project Reactor flatMap() concurrency idiom
Key Transforming Operators in the Flux Class
The flatMap() operator

Transform the elements emitted by this Flux asynchronously

```
<R> Flux<R> flatMap
    (Function<? super T, ? extends Publisher<? extends R>>
        mapper)
```

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#flatMap](projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#flatMap)
Key Transforming Operators in the Flux Class

• The flatMap() operator
  
• Transform the elements emitted by this Flux asynchronously
  
• These elements are emitted into inner Publishers

\[
\text{<R> Flux<R> flatMap (Function<? super T, }
\text{? extends Publisher<? extends R>> mapper)}
\]
The flatMap() operator

Transform the elements emitted by this Flux asynchronously

These elements are emitted into inner Publishers

Each <T> input element is mapped to a Publisher<R>

```java
<R> Flux<R> flatMap
    (Function<? super T,
             ? extends Publisher<? extends R>>
        mapper)
```
Key Transforming Operators in the Flux Class

- The flatMap() operator
- Transform the elements emitted by this Flux asynchronously
  - These elements are emitted into inner Publishers
    - Each <T> input element is mapped to a Publisher<R>
    - That publisher will emit one or more items
Key Transforming Operators in the Flux Class

- The flatMap() operator
  - Transform the elements emitted by this Flux asynchronously
    - These elements are emitted into inner Publishers
    - These inner publishers are then flattened into one Flux by merging
Key Transforming Operators in the Flux Class

- The flatMap() operator
  - Transform the elements emitted by this Flux asynchronously
    - These elements are emitted into inner Publishers
  - These inner publishers are then flattened into one Flux by merging
    - They thus can interleave
      - Especially when used for concurrent processing

See upcoming walkthrough of the “flatMap() concurrency idiom” example
• The flatMap() operator
  • Transform the elements emitted by this Flux asynchronously
    • These elements are emitted into inner Publishers
  • These inner publishers are then flattened into one Flux by merging
    • They thus can interleave

The # of output elements may differ from the # of input elements
Key Transforming Operators in the Flux Class

- The `flatMap()` operator
- Transform the elements emitted by this Flux asynchronously
  - These elements are emitted into inner Publishers
  - These inner publishers are then flattened into one Flux by merging
    - They thus can interleave

`flatMap()` can transform the values and/or type of elements it processes
The `flatMap()` operator

- Transform the elements emitted by this Observable asynchronously

- This method is often used to trigger concurrent processing

```java
return Flux
    .fromCallable(() -> BigFraction
        .reduce(unreducedFraction))
    .subscribeOn(scheduler)
    .flatMap(reducedFraction ->
        Flux
            .fromCallable(() ->
                reducedFraction
                .multiply(sBigReducedFrac))
            .subscribeOn
                (scheduler));
```

See upcoming discussion on the Project Reactor `flatMap()` concurrency idiom
Key Transforming Operators in the Flux Class

- The flatMap() operator
  - Transform the elements emitted by this Observable asynchronously
  - This method is often used to trigger concurrent processing

```
return Flux
  .fromCallable(() -> BigFraction
    .reduce(unreducedFraction))
  .subscribeOn(scheduler)
  .flatMap(reducedFraction ->
    Flux
      .fromCallable(() ->
        reducedFraction
          .multiply(sBigReducedFrac))
      .subscribeOn(scheduler));
```

Return a Flux to a multiplied big fraction using the Project Reactor flatMap() concurrency idiom

See Reactive/flux/ex3/src/main/java/FluxEx.java
Key Transforming Operators in the Flux Class

- The flatMap() operator
  - Transform the elements emitted by this Flux asynchronously
  - This method is often used to trigger concurrent processing
- RxJava’s Observable.flatMap() method works the same way

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#flatMap](http://reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#flatMap)
Key Transforming Operators in the Flux Class

- The flatMap() operator
  - Transform the elements emitted by this Flux asynchronously
  - This method is often used to trigger concurrent processing
  - RxJava’s Observable.flatMap() method works the same way
  - Similar to the Java Streams flatMap() operator

```java
List<String> a = List.of("d", "g");
List<String> b = List.of("a", "c");
Stream.of(a, b)
    .flatMap(List::stream)
    .sorted()
    .forEach(System.out::println);
```

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#flatMap
Key Transforming Operators in the Flux Class

- flatMap() doesn’t guarantee the order of the items in the resulting stream
Key Transforming Operators in the Flux Class

- flatMap() doesn’t guarantee the order of the items in the resulting stream
- use concatMap() if order matters

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#concatMap](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#concatMap)
The Project Reactor
flatMap() Concurrency Idiom
The Project Reactor flatMap() Concurrency Idiom

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

```java
return Flux
    .fromIterable(bigFractions)
    .flatMap(bf -> Mono
        .fromCallable(() -> bf)
        .subscribeOn
            (Schedulers
                .parallel())
        .map(multiplyBigFracs))
    .reduce(BigFraction::add)
...
```
The Project Reactor 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 Flux
    .fromIterable(bigFractions)
    .flatMap(bf -> Mono
        .fromCallable(() -> bf)
        .subscribeOn
            (Schedulers
                .parallel())
        .map(multiplyBigFracs))
    .reduce(BigFraction::add)
...
The Project Reactor 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 Flux
    .fromIterable(bigFractions)
    .flatMap(bf -> Mono
        .fromCallable(() -> bf)
        .subscribeOn
            (Schedulers
                .parallel())
        .map(multiplyBigFracs))
    .reduce(BigFraction::add)
...
The Project Reactor 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”

Iterate thru the Flux stream multiplying big fractions in the parallel thread pool

```
return Flux
  .fromIterable(bigFractions)
  .flatMap(bf -> Mono
    .fromCallable(() -> bf)
    .subscribeOn
      (Schedulers
        .parallel())
    .map(multiplyBigFracs))
  .reduce(BigFraction::add)
...
```

See Reactive/flux/ex3/src/main/java/FluxEx.java
The Project Reactor `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 Flux
    .fromIterable(bigFractions)
    .flatMap(bf -> Mono
        .fromCallable(() -> bf)
        .subscribeOn
            (Schedulers.parallel())
        .map(multiplyBigFracs))
    .reduce(BigFraction::add)
...
The Project Reactor 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 Flux
    .fromIterable(bigFractions)
    .flatMap(bf -> Mono
        .fromCallable(() -> bf)
        .subscribeOn
        (Schedulers.parallel())
        .map(multiplyBigFracs))
    .reduce(BigFraction::add)
...```

*Emit each BigFraction in the “inner publisher”*
The Project Reactor 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 Flux
    .fromIterable(bigFractions)
    .flatMap(bf -> Mono
        .fromCallable(() -> bf)
        .subscribeOn
        (Schedulers.parallel())
        .map(multiplyBigFracs))
    .reduce(BigFraction::add)
...)
```

Arrange to process each emitted BigFraction in the parallel thread pool
The Project Reactor 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 Flux
    .fromIterable(bigFractions)
    .flatMap(bf -> Mono
        .fromCallable(() -> bf)
        .subscribeOn
            (Schedulers.parallel())
        .map(multiplyBigFracs))
    .reduce(BigFraction::add)
...
```

Multiply each BigFraction in a thread from the parallel thread pool
The Project Reactor 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 Flux
    .fromIterable(bigFractions)
    .flatMap(bf -> Mono
        .fromCallable(() -> bf)
        .subscribeOn
            (Schedulers
                .parallel())
        .map(multiplyBigFracs))
    .reduce(BigFraction::add)
...
Comparing map & flatMap()
Comparing map() & flatMap()

- The map() vs. flatMap() operators
Comparing map() & flatMap()

- The map() vs. flatMap() operators
- The map() operator transforms each value in a Flux stream into a single value
  - i.e., intended for synchronous, non-blocking, 1-to-1 transformations

See stackoverflow.com/questions/49115135/map-vs-flatmap-in-reactor
The map() vs. flatMap() operators

- The map() operator transforms each value in a Flux stream into a single value.
- The flatMap() operator transforms each value in a Flux stream into an arbitrary number (zero or more) values.
- i.e., intended for asynchronous (often non-blocking) 1-to-N transformations.

End of Key Transforming Operators in the Flux Class (Part 2)