Key Transforming Operators in the Flux 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 Flux operators
  - Factory method operators
  - Transforming operators
    - Transform the values and/or types emitted by a Flux
      - e.g., flatMap()


```
return Flux
  .fromIterable(bigFractions)

  .flatMap(bf -> Mono
        .fromCallable(() -> bf
           .multiply(sBigFrac))

        .subscribeOn
           (Schedulers
            .parallel()))

  .reduce(BigFraction::add)
  ...
```

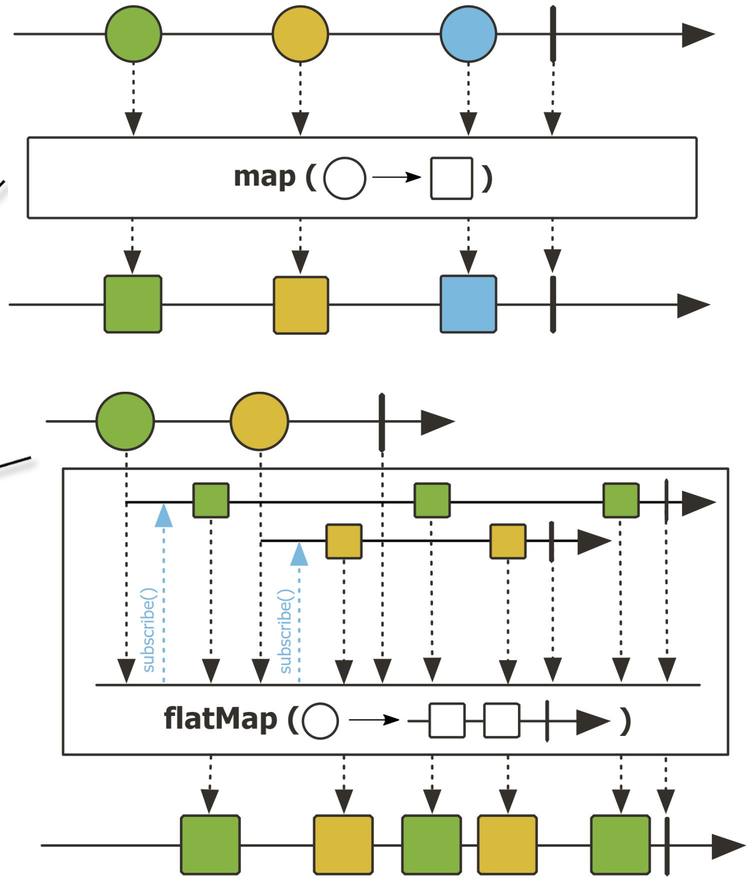
We explain the Project Reactor "flatMap() concurrency idiom"

This idiom is particularly useful for "embarrassing parallel" programs

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()

We also compare & contrast the Project Reactor map() & flatMap() operators



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


```
return Flux
   .fromIterable(bigFractions)

   .flatMap(bf -> Mono
            .fromCallable(() -> bf
               .multiply(sBigFrac))

            .subscribeOn
               (Schedulers
                .parallel()))

   .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
                .multiply(sBigFrac))

             .subscribeOn
                (Schedulers
                 .parallel()))

   .reduce(BigFraction::add)
   ...
```

See ebaytech.berlin/declarative-concurrency-with-reactor-70507e04054a

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
                    .multiply(sBigFrac))

                 .subscribeOn
                    (Schedulers
                     .parallel()))

    .reduce(BigFraction::add)
    ...
```

Create a Flux BigFraction stream from a BigFraction list

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"

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

```
return Flux
   .fromIterable(bigFractions)

   .flatMap(bf -> Mono
            .fromCallable(() -> bf
               .multiply(sBigFrac))

            .subscribeOn
               (Schedulers
                .parallel()))

   .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
                   .multiply(sBigFrac))

                .subscribeOn
                   (Schedulers
                    .parallel()))

   .reduce(BigFraction::add)
   ...
```

Each BigFraction in the stream is processed concurrently 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"

```
return Flux
  .fromIterable(bigFractions)

  .flatMap(bf -> Mono
             .fromCallable(() -> bf
                .multiply(sBigFrac))

             .subscribeOn
                (Schedulers
                 .parallel()))

  .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"

```
return Flux
  .fromIterable(bigFractions)

  .flatMap(bf -> Mono
           .fromCallable(() -> bf
              .multiply(sBigFrac))

           .subscribeOn
             (Schedulers
              .parallel()))

  .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"

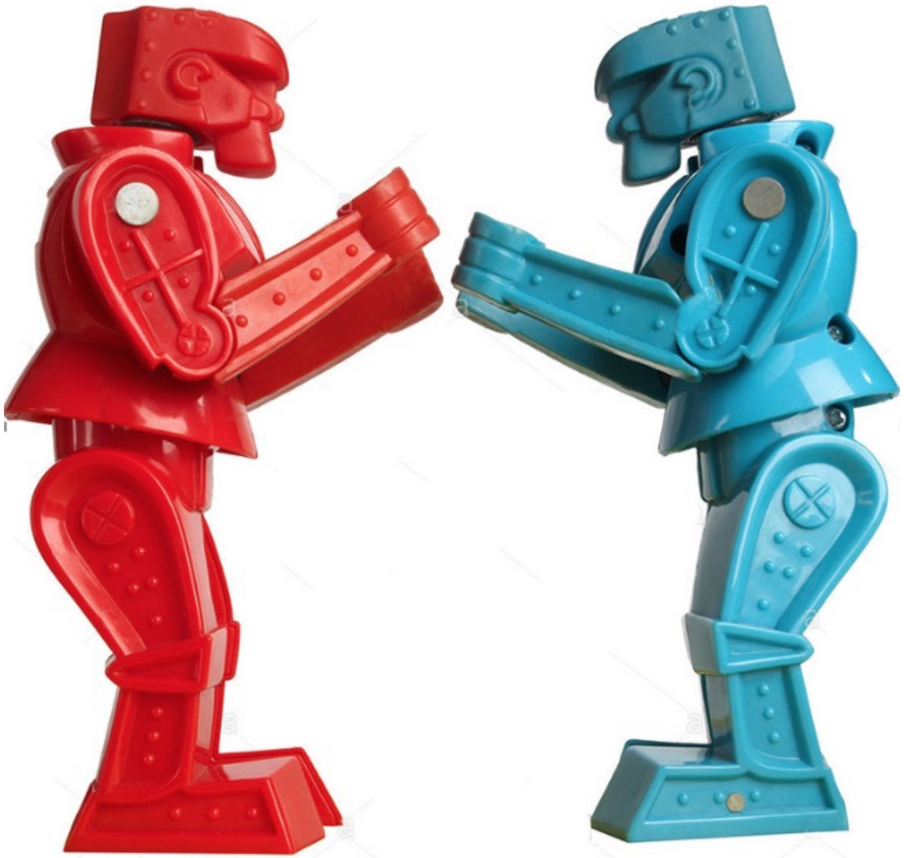
```
return Flux
   .fromIterable(bigFractions)

   .flatMap(bf -> Mono
            .fromCallable(() -> bf
               .multiply(sBigFrac))

            .subscribeOn
               (Schedulers
                .parallel()))

   .reduce(BigFraction::add)
   ...
```

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

Comparing map & flatMap()

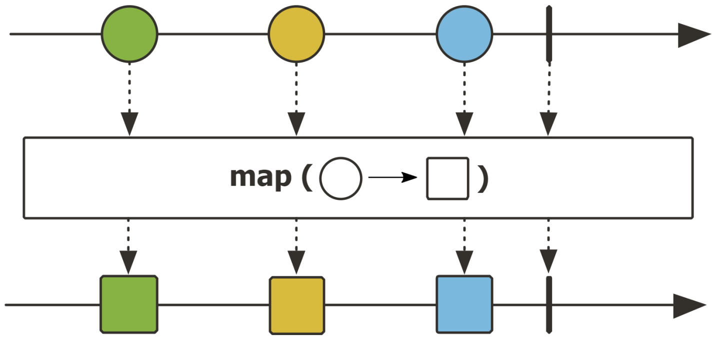
Comparing map() & flatMap()

- The map() vs. flatMap() operators



Comparing map() & flatMap()

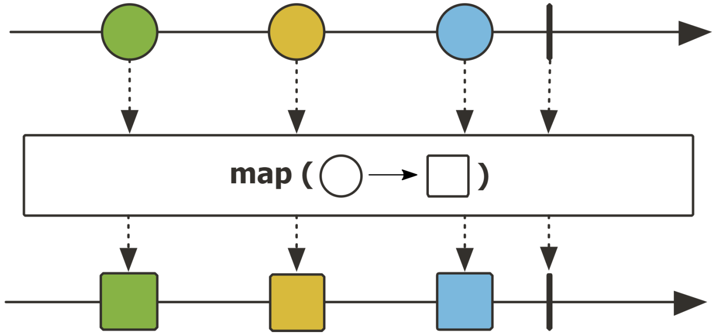
- The map() vs. flatMap() operators
  - The map() operator transforms each value in a Flux stream into a single value



See stackoverflow.com/questions/49115135/map-vs-flatmap-in-reactor

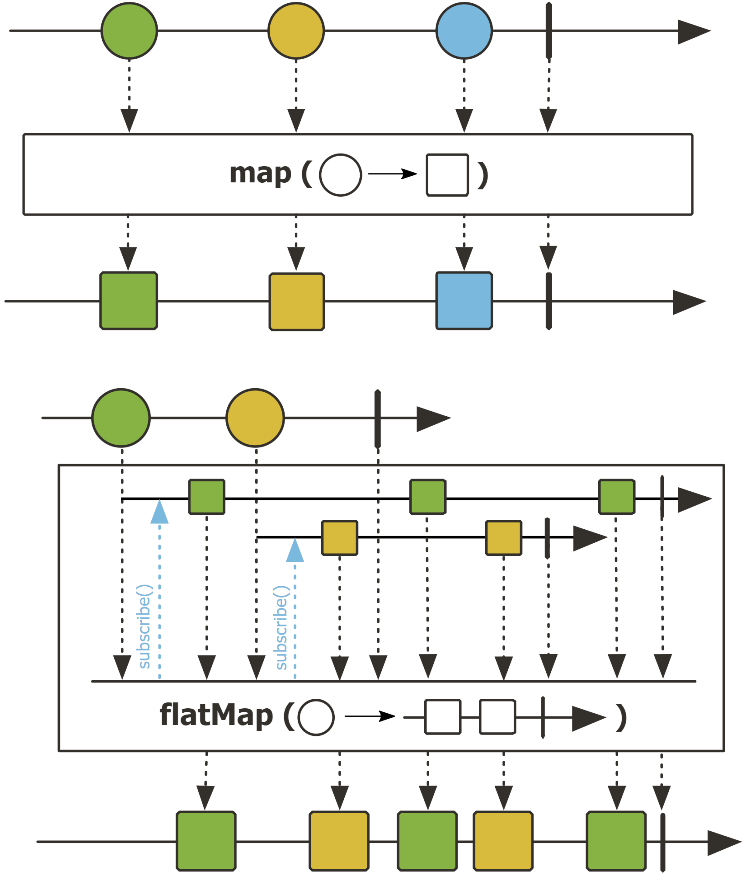
Comparing map() & flatMap()

- 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



Comparing map() & flatMap()

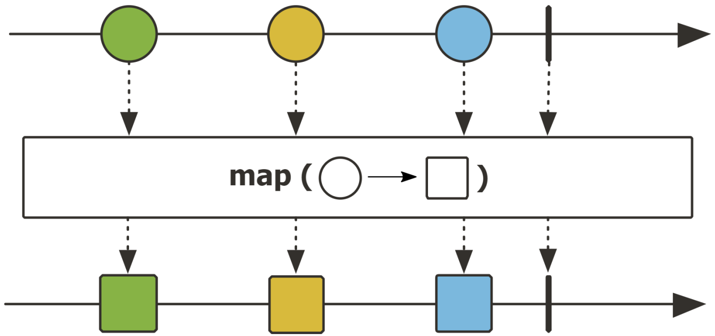
- The map() vs. flatMap() operators
  - The map() operator transforms each value in a Flux stream into a single value
  - The flatMap() operator transforms each value in a Flux stream into an arbitrary number (zero or more) values

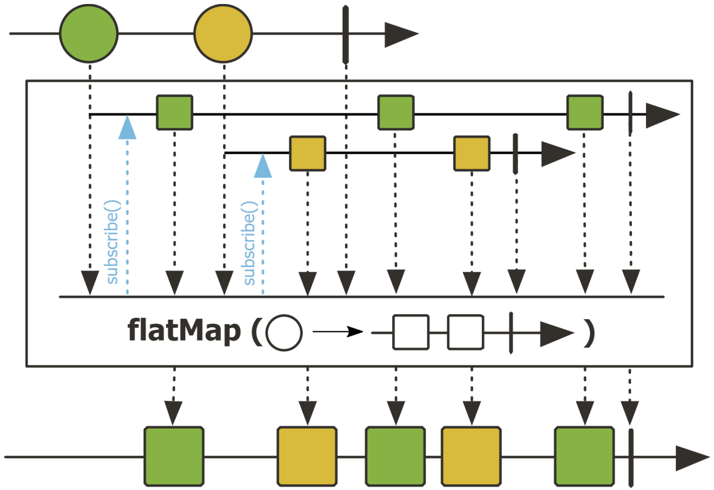


See stackoverflow.com/questions/49115135/map-vs-flatmap-in-reactor

Comparing map() & flatMap()

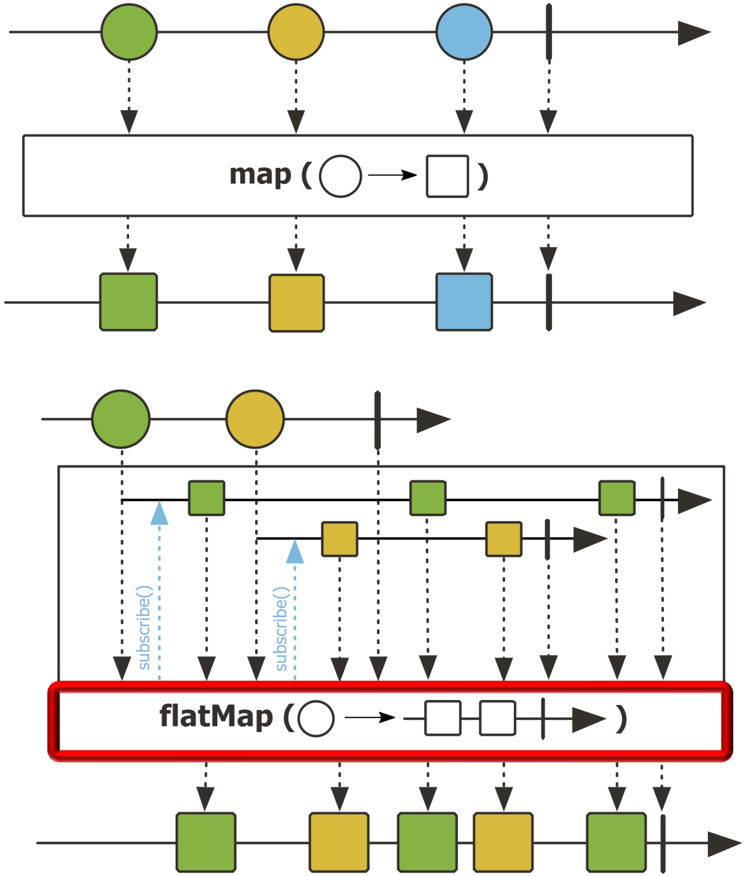
- The map() vs. flatMap() operators
  - The map() operator transforms each value in a Flux stream into a single value
  - The flatMap() operator transforms each value in a Flux stream into an arbitrary number (zero or more) values
    - i.e., intended for asynchronous (often non-blocking) 1-to-N transformations





Comparing map() & flatMap()

- 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
  - flatMap() is used extensively in Project Reactor



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