Key Combining 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 operations
  • Transforming operators
  • Concurrency & scheduler operators
• Error handling operators
• Combining operators
  • These operators create a Flux from multiple sources or iterations
    • e.g., reduce(), collectList(), & collect()
Key Combining Operators in the Flux Class
Key Combining Operators in the Flux Class

- The `reduce()` operator
- Reduce the values from this Flux sequence into a single object of the same type as the emitted items

```java
Mono<U> reduce
    (BiFunction<T, T, T> reducer)
```

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#reduce](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#reduce)
Key Combining Operators in the Flux Class

- The reduce() operator
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Reduction is performed using a BiFunction param

```
Mono<U> reduce
  (BiFunction<T, T, T> reducer)
```

Interface BiFunction<T,U,R>

Type Parameters:
- T - the type of the first argument to the function
- U - the type of the second argument to the function
- R - the type of the result of the function

All Known Subinterfaces:
- BinaryOperator<T>

Functional Interface:
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

See docs.oracle.com/javase/8/docs/api/java/util/function/BiFunction.html
Key Combining Operators in the Flux Class

- The `reduce()` operator
- Reduce the values from this Flux sequence into a single object of the same type as the emitted items
- Reduction is performed using a `BiFunction` param
- This param is passed the intermediate result of the reduction & the current value

```java
Mono<U> reduce
(BiFunction<T, T, T> reducer)
```

Flux of Integers from 1..4

```
1 + 2 + 3 + 4
0 + 1  2  3  4
```

This value is initialized to zero (0) for Integer
Key Combining Operators in the Flux Class

- The reduce() operator
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Reduction is performed using a `BiFunction` param
  - This param is passed the intermediate result of the reduction & the current value
  - It returns the next intermediate value of the reduction

```
Mono<U> reduce
(BiFunction<T, T, T> reducer)
```

Flux of Integers from 1..4

```
1 + 2 + 3 + 4
```

```
0 + 1 + 3 + 6 + 10
```
The reduce() operator
- Reduce the values from this Flux sequence into a single object of the same type as the emitted items
- Reduction is performed using a BiFunction param
  - This param is passed the intermediate result of the reduction & the current value
    - It returns the next intermediate value of the reduction
  - The process repeats for each pair of values

```
Mono<U> reduce
(BiFunction<T, T, T> reducer)
```

Flux of Integers from 1..4

```
0 + 1 + 2 + 3 + 4
= 10
```
Key Combining Operators in the Flux Class

- The reduce() operator
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Reduction is performed using a BiFunction param
  - This param is passed the intermediate result of the reduction & the current value
    - It returns the next intermediate value of the reduction
  - The process repeats for each pair of values

```
Mono<U> reduce
(BiFunction<T, T, T> reducer)
```

Flux of Integers from 1..4

```
0 + 1 = 1
1 + 2 = 3
3 + 3 = 6
6 + 4 = 10
```

Key Combining Operators in the Flux Class

- The `reduce()` operator
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items
    - Reduction is performed using a `BiFunction` param
    - This param is passed the intermediate result of the reduction & the current value
      - It returns the next intermediate value of the reduction
    - The process repeats for each pair of values
Key Combining Operators in the Flux Class

• The `reduce()` operator
  • Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  • Reduction is performed using a `BiFunction` param
  • This param is passed the intermediate result of the reduction & the current value
  • It returns the next intermediate value of the reduction
  • The process repeats for each pair of values

```
Mono<U> reduce
(BiFunction<T, T, T> reducer)
```

Flux of Integers from 1..4

```
0 + 1 = 1
1 + 2 = 3
3 + 3 = 6
6 + 4 = 10
```

11
Key Combining Operators in the Flux Class

- The `reduce()` operator
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Reduction is performed using a `BiFunction` param
    - This param is passed the intermediate result of the reduction & the current value
      - It returns the next intermediate value of the reduction
      - The process repeats for each pair of values

```
Mono<U> reduce
(BiFunction<T, T, T> reducer)
```

Diagram:
- Flux of Integers from 1..4
- Reduction process shown with addition operation:
  - 0
  - 1
  - 3
  - 6
  - 10

Flow of integers: 1 + 2 + 3 + 4
Result: 10

Code snippet:
```
Mono<Integer> result = flux.reduce((acc, item) -> acc + item);
```
Key Combining Operators in the Flux Class

- **The reduce() operator**
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Reduction is performed using a BiFunction param
    - This param is passed the intermediate result of the reduction & the current value
      - It returns the next intermediate value of the reduction
  - The process repeats for each pair of values

```
Mono<U> reduce
(BiFunction<T, T, T> reducer)
```

![Diagram](image)
Key Combining Operators in the Flux Class

- The `reduce()` operator
- Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Reduction is performed using a `BiFunction` param
  - The result of the reduced Flux is emitted from the final call as sole item of a `Mono`

```java
Mono<? extends U> reduce
    (BiFunction<? super T, ? super T, ? extends T> reducer)
```
Key Combining Operators in the Flux Class

- The `reduce()` operator
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Reduction is performed using a BiFunction param
  - The result of the reduced Flux is emitted from the final call as sole item of a Mono
  - If the Flux emits no items Mono will be empty
Key Combining Operators in the Flux Class

- The reduce() operator
- Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Reduction is performed using a BiFunction param
- The result of the reduced Flux is emitted from the final call as sole item of a Mono
  - If the Flux emits no items Mono will be empty
  - The internally accumulated value is discarded upon cancellation or error

```java
Mono<U> reduce
   (BiFunction<T, T, T> reducer)
```
Key Combining Operators in the Flux Class

- The reduce() operator
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Upstream must signal onComplete() before accumulator can be emitted

```java
return Flux
    .fromArray(bigFractions)
    .flatMap(bf ->
        multiplyFractions(bf,
            Schedulers.parallel()))
    .reduce(BigFraction::add)
...
```

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

- The `reduce()` operator
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Upstream must signal `onComplete()` before accumulator can be emitted
  - Sources that are infinite & never complete will never emit anything through this operator
Key Combining Operators in the Flux Class

- The `reduce()` operator
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Upstream must signal `onComplete()` before accumulator can be emitted
  - Sources that are infinite & never complete will never emit anything through this operator
  - An infinite source may lead to a fatal `OutOfMemoryError`
Key Combining Operators in the Flux Class

- The reduce() operator
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items
  - Upstream must signal onComplete() before accumulator can be emitted
- RxJava’s Observable.reduce() operator works the same

```
return Observable
  .fromArray(bigFractions)
  .flatMap(bf ->
    multiplyFrations(bf,Schedulers.computation()))
  .reduce(BigFraction::add) ...
```

See `reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#reduce`
Key Combining Operators in the Flux Class

- **The reduce() operator**
  - Reduce the values from this Flux sequence into a single object of the same type as the emitted items.
  - Upstream must signal onComplete() before accumulator can be emitted.
  - RxJava’s Observable.reduce() operator works the same.
  - Similar to the Stream.reduce() method in Java Streams.

```java
int result = List.of(1, 2, 3, 4, 5, 6).stream()
    .reduce(0, Math::addExact);
```

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#reduce](docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#reduce)
Key Combining Operators in the Flux Class

- The `collectList()` operator
  - Collect all elements emitted by this Flux into a List

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#collectList](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#collectList)
Key Combining Operators in the Flux Class

- The `collectList()` operator
- Collect all elements emitted by this Flux into a List
- Returns a Mono to a List containing all values from this Flux

```java
public abstract class Mono<T> extends Object implements CorePublisher<T>

A Reactive Streams Publisher with basic rx operators that completes successfully by emitting an element, or with an error.
```

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html)
Key Combining Operators in the Flux Class

- The `collectList()` operator
  - Collect all elements emitted by this Flux into a List
- The list is emitted by the Mono when this sequence completes

```java
Flux.fromIterable(bigFractions)
    .flatMap(...)
    .filter(fraction -> fraction.compareTo(0) > 0)
    .collectList()
...
Key Combining Operators in the Flux Class

- The `collectList()` operator
  - Collect all elements emitted by this Flux into a List
  - The list is emitted by the Mono when this sequence completes
- RxJava’s `Observable.collect()` is a generalization of `collectList()`

```java
Observable.fromIterable(bigFractions)
.flatMap(...)
.filter(fraction -> fraction.compareTo(0) > 0)
.collect(toList())
...
```

Collect the filtered `BigFractions` into a list

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

- The `collectList()` operator
  - Collect all elements emitted by this Flux into a List
  - The list is emitted by the Mono when this sequence completes
  - RxJava’s Observable.collect() is a generalization of collectList()
- Similar to the Stream.collect() method in Java Streams

```java
List<Integer> evenNumbers = List.of(1, 2, 2, 3, 4, 5, 6, 6).stream()
    .filter(x -> x % 2 == 0)
    .toList();
```

See [docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Stream.html#toList()](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Stream.html#toList())
Key Combining Operators in the Flux Class

• The collect() operator
  • Collect all elements emitted by this Flux into a container

\[
\text{<R, A> \ Mono<R> collect (Collector<? super T, A, ? extends R> collector)}
\]

See \text{projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#collect}
Key Combining Operators in the Flux Class

- The collect() operator
- Collect all elements emitted by this Flux into a container
  - The param is the Java Stream Collector interface
  - This interface defines the supplier(), accumulator(), combiner(), & finisher() methods

```java
```

### Interface Collector<T,A,R>

**Type Parameters:**
- T - the type of input elements to the reduction operation
- A - the mutable accumulation type of the reduction operation (often hidden as an implementation detail)
- R - the result type of the reduction operation

**Description:**
A mutable reduction operation that accumulates input elements into a mutable result container, optionally transforming the accumulated result into a final representation after all input elements have been processed. Reduction operations can be performed either sequentially or in parallel.

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html](docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html)
Key Combining Operators in the Flux Class

- The collect() operator
- Collect all elements emitted by this Flux into a container
  - The param is the Java Stream Collector interface
- The collected result is emitted via a Mono when this sequence completes

```java
<R, A> Mono<R> collect
(Collector<? super T,
  A,
  ? extends R> collector)
```
Key Combining Operators in the Flux Class

- The collect() operator
  - Collect all elements emitted by this Flux into a container
  - Can be used to seamlessly integrate Project Reactor & Java Streams capabilities
  
  ```java
  return monos -> Mono
    .when(mimos)
    .materialize()
    .flatMap(v -> Flux
      .fromIterable(mimos)
      .map(Mono::block)
      .collect(toList()));
  ```

  Return a Mono to a List of results that were computed asynchronously

See Reactive/flux/ex3/src/main/java/utils/MonosCollector.java
Key Combining Operators in the Flux Class

- **The collect() operator**
  - Collect all elements emitted by this Flux into a container
  - Can be used to seamlessly integrate Project Reactor & Java Streams capabilities
- RxJava’s operator Observable. collect() works the same

```java
Observable.fromIterable(bigFractions)
  .flatMap(...)
  .filter(fraction -> fraction.compareTo(0) > 0)
  .collect(toList())
...```

Collect the filtered BigFractions into a list

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

- The collect() operator
  - Collect all elements emitted by this Flux into a container
  - Can be used to seamlessly integrate Project Reactor & Java Streams capabilities
  - RxJava’s operator Observable. collect() works the same
  - Similar to the Stream.collect() method in Java Streams

```java
Set<Integer> evenNumbers = List.of(1, 2, 2, 3, 4, 4, 5, 6, 6).stream().filter(x -> x % 2 == 0).collect(toSet());
```

Collect even #'d Integers into a Set

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#collect](docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#collect)
End of Key Combining Operators in the Flux Class (Part 2)