Overview of the ParallelFlowable Class

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

• Understand the capabilities of the ParallelFlowable class

Class ParallelFlowable<T>

java.lang.Object

io.reactivex.rxjava3.parallel.ParallelFlowable<T>

Type Parameters:

T - the value type

public abstract class ParallelFlowable<T>

extends Object

Abstract base class for parallel publishing of events signaled to an array of Subscribers.

Use from(Publisher) to start processing a regular Publisher in 'rails'. Use runOn(Scheduler) to introduce where each 'rail' should run on thread-wise. Use sequential() to merge the sources back into a single Flowable.

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html
Learning Objectives in this Part of the Lesson

• Understand the capabilities of the ParallelFlowable class
• Simplifies parallel processing cf. the flatMap() concurrency idiom

return Observable
  .fromArray(bigFractionArray)
  .flatMap(bf -> Observable
    .fromCallable(() -> bf
      .multiply(sBigReducedFrac))
    .subscribeOn(Schedulers.computation()))
  .reduce(BigFraction::add)...
Overview of the ParallelFlowable Class
The RxJava flatMap() concurrency idiom performs relatively well, but is also somewhat convoluted.

Return an Observable that emits multiplied BigFraction objects via the RxJava flatMap() concurrency idiom

```
return Observable
  .fromArray(bigFractionArray)
    .flatMap(bf -> Observable
        .fromCallable(() -> bf
          .multiply(sBigFraction))
        .subscribeOn(Schedulers.computation()))
    .reduce(BigFraction::add)
...  
```
Overview of the ParallelFlowable Class

- The RxJava flatMap() concurrency idiom performs relatively well, but is also somewhat convoluted.
- Particularly in comparison with Java parallel streams

```java
return Stream
    .of(bigFractionArray)
    .parallel()
    .map(bf -> bf
        .multiply(sBigFraction))
    .reduce(ZERO, BigFraction::add)
    .flatMap(bf -> Observable
        .fromCallable(() -> bf
            .multiply(sBigFraction))
        .subscribeOn(Schedulers.computation()))
    .reduce(BigFraction::add)
    ...
```

See [docs.oracle.com/javase/tutorial/collections/stream/parallelism.html](docs.oracle.com/javase/tutorial/collections/stream/parallelism.html)
Overview of the ParallelFlowable Class

- ParallelFlowable is a subset of Flowable that provides a more concise means of processing multiple values in parallel.

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html)
Overview of the ParallelFlowable Class

- ParallelFlowable is a subset of Flowable that provides a more concise means of processing multiple values in parallel.
- Similar to Java parallel streams.

Overview of the ParallelFlowable Class

- ParallelFlowable is a subset of Flowable that provides a more concise means of processing multiple values in parallel.
- Similar to Java parallel streams.
  - i.e., intended for “embarrassingly parallel” tasks.

“Embarrassingly parallel” tasks have little/no dependency or need for communication between tasks or for sharing results between them.

See [en.wikipedia.org/wiki/Embarrassingly_parallel](en.wikipedia.org/wiki/Embarrassingly_parallel)
Overview of the ParallelFlowable Class

- ParallelFlowable is a subset of Flowable that provides a more concise means of processing multiple values in parallel
  - Similar to Java parallel streams
  - Avoids the convoluted syntax of the flatMap() concurrency idiom

See dzone.com/articles/rxjava-idiomatic-concurrency-flatmap-vs-parallel
Overview of the ParallelFlowable Class

- ParallelFlowable is a subset of Flowable that provides a more concise means of processing multiple values in parallel
  - Similar to Java parallel streams
  - Avoids the convoluted syntax of the flatMap() concurrency idiom
  - The Flowable.parallel() factory method creates a ParallelFlowable

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Flowable.html#parallel](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Flowable.html#parallel)
ParallelFlowable is a subset of Flowable that provides a more concise means of processing multiple values in parallel:

- Similar to Java parallel streams
- Avoids the convoluted syntax of the flatMap() concurrency idiom
- The Flowable.parallel() factory method creates a ParallelFlowable
  - Elements are processed in parallel via ‘rails’ in round-robin order

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Flowable.html#parallel
Overview of the ParallelFlowable Class

- ParallelFlowable is a subset of Flowable that provides a more concise means of processing multiple values in parallel
  - Similar to Java parallel streams
  - Avoids the convoluted syntax of the flatMap() concurrency idiom
- The Flowable.parallel() factory method creates a ParallelFlowable
  - Elements are processed in parallel via ‘rails’ in round-robin order
  - By default, the # of rails is set to the # of available CPU cores

See docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#availableProcessors
Overview of the ParallelFlowable Class

- ParallelFlowable is a subset of Flowable that provides a more concise means of processing multiple values in parallel
  - Similar to Java parallel streams
  - Avoids the convoluted syntax of the `flatMap()` concurrency idiom
- The `Flowable.parallel()` factory method creates a ParallelFlowable
  - Elements are processed in parallel via ‘rails’ in round-robin order
  - By default, the # of rails is set to the # of available CPU cores
    - This setting can be changed programmatically

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Flowable.html#parallel](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Flowable.html#parallel)
Overview of the ParallelFlowable Class

- ParallelFlowable supports a subset of Flowable operators that process elements in parallel across the rails
  - e.g., map(), filter(), concatMap(), flatMap(), collect(), & reduce()

See [github.com/ReactiveX/RxJava/wiki/Parallel-flows](https://github.com/ReactiveX/RxJava/wiki/Parallel-flows)
Overview of the ParallelFlowable Class

- The `runOn()` operator specifies where each 'rail' will observe its incoming elements

```java
ParallelFlowable<T> runOn(Scheduler scheduler)
```

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html#runOn](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html#runOn)
Overview of the ParallelFlowable Class

- The `runOn()` operator specifies where each 'rail' will observe its incoming elements
- Specified via a Scheduler that performs no work-stealing

```
ParallelFlowable<T> runOn(Scheduler scheduler)
```
Overview of the ParallelFlowable Class

- The `runOn()` operator specifies where each 'rail' will observe its incoming elements
  - Specified via a Scheduler that performs no work-stealing
  - Returns the new Parallel Flowable instance
Overview of the ParallelFlowable Class

- A ParallelFlowable can be converted back into a Flowable via sequential()

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html#sequential](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html#sequential)
Overview of the ParallelFlowable Class

- A ParallelFlowable can be converted back into a Flowable via `sequential()`
- Merge the values from each 'rail' in a round-robin fashion

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html#sequential](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html#sequential)
Overview of the ParallelFlowable Class

- ParallelFlowable.reduce() can also be used to convert back into a Flowable

```java
public final @NonNull Flowable<T> reduce(@NonNull BiFunction<T, T, T> reducer)
```

Reduces all values within a 'rail' and across 'rails' with a reducer function into one `Flowable` sequence.

Note that the same reducer function may be called from multiple threads concurrently.

- **Backpressure:** The operator honors backpressure from the downstream and consumes the upstream rails in an unbounded manner (requesting `Long.MAX_VALUE`).
- **Scheduler:** reduce does not operate by default on a particular Scheduler.
- **Parameters:** reducer - the function to reduce two values into one.
- **Returns:** the new Flowable instance emitting the reduced value or empty if the current ParallelFlowable is empty
- **Throws:** `NullPointerException` - if reducer is null

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html#reduce](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html#reduce)
Overview of the ParallelFlowable Class

- ParallelFlowable.reduce() can also be used to convert back into a Flowable
- Reduces all values within a 'rail' & across 'rails' into a Flowable

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html#reduce
Overview of the ParallelFlowable Class

- ParallelFlowable.reduce() can also be used to convert back into a Flowable
  - Reduces all values within a 'rail' & across 'rails' into a Flowable
  - The BiFunction param reduces two values into one successively

Flowable<T> reduce
(BiFunction<T,T,T> reducer)

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/functions/BiFunction.html
Overview of the ParallelFlowable Class

- ParallelFlowable.reduce() can also be used to convert back into a Flowable
- Reduces all values within a 'rail' & across 'rails' into a Flowable
- The BiFunction param reduces two values into one successively
- Return a regular Flowable that contains just one element
Elements that flow through a ParallelFlowable stream are processed in parallel.

Multiply BigFraction objects in parallel using RxJava’s ParallelFlowable operators:

```java
return Flowable
  .fromArray(bigFractionArray)
  .parallel()
  .runOn
      (Schedulers.computation())
  .map(bf -> bf
      .multiply(sBigReducedFrac))
  .reduce(BigFraction::add)
  .firstElement()
  .doOnSuccess(displayResults)
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Flowable/ex3](github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Flowable/ex3)
Overview of the ParallelFlowable Class

- Elements that flow through a ParallelFlowable stream are processed in parallel

```
return Flowable
  .fromArray(bigFractionArray)
  .parallel()
  .runOn
    (Schedulers.computation())
  .map(bf -> bf
    .multiply(sBigReducedFrac))
  .reduce(BigFraction::add)
  .firstElement()
  .doOnSuccess(displayResults)
```

**Convert an array into a Flowable**
Overview of the ParallelFlowable Class

- Elements that flow through a ParallelFlowable stream are processed in parallel

```java
return Flowable
    .fromArray(bigFractionArray)
    .parallel()
    .runOn
    (Schedulers.computation())
    .map(bf -> bf
        .multiply(sBigReducedFrac))
    .reduce(BigFraction::add)
    .firstElement()
    .doOnSuccess(displayResults)
```

Convert the Flowable into a ParallelFlowable
Overview of the ParallelFlowable Class

- Elements that flow through a ParallelFlowable stream are processed in parallel.

```java
return Flowable
  .fromArray(bigFractionArray)
  .parallel()
  .runOn(
    (Schedulers.computation())
  .map(bf -> bf
    .multiply(sBigReducedFrac))
  .reduce(BigFraction::add)
  .firstElement()
  .doOnSuccess(displayResults)
```

Designate the computation Scheduler that multiplies each image in parallel.
Elements that flow through a ParallelFlowable stream are processed in parallel.

Return Flowable

```
.array(bigFractionArray)
.parallel()
.runOn(Schedulers.computation())
.map(bf -> bf
  .multiply(sBigReducedFrac))
.reduce(BigFraction::add)
.firstElement()
.doOnSuccess(displayResults)
```

Multiply BigFraction objects in parallel using map()
Elements that flow through a ParallelFlowable stream are processed in parallel. Return Flowable

\[
\text{return Flowable} \ \text{fromArray(bigFractionArray)} \\
\text{.parallel()} \\
\text{.runOn} \\
\text{\quad (Schedulers.computation())} \\
\text{.map(bf -> bf} \\
\text{\quad .multiply(sBigReducedFrac))} \\
\text{.reduce(BigFraction::add)} \\
\text{.firstElement()} \\
\text{.doOnSuccess(displayResults)}
\]

Reduce all values within a 'rail' & across 'rails' via BigFraction::add into a Flowable sequence with one element.
Elements that flow through a ParallelFlowable stream are processed in parallel.

Return a Maybe that emits the one & only element in the Flowable.
Elements that flow through a ParallelFlowable stream are processed in parallel

```
return Flowable
  .fromArray(bigFractionArray)
  .parallel()
  .runOn
    (Schedulers.computation())
  .map(bf -> bf
    .multiply(sBigReducedFrac))
  .reduce(BigFraction::add)
  .firstElement()
  .doOnSuccess(displayResults)
```

Display the results if all goes well
End of Overview of the ParallelFlowable Class