Key Factory Method 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
  • Concurrency & scheduler operators
  • Factory method operators
    • These operators create Flux streams in various ways
      • e.g., create(), range(), & interval()

See en.wikipedia.org/wiki/Factory_method_pattern
Key Factory Method
Operators in the Flux Class
Key Factory Method Operators in the Flux Class

- The `create()` operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously

```java
class Flux {
    static <T> Flux<T> create (Consumer<? super FluxSink<T>> emitter)
}
```

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

• The create() operator
• Create a Flux capable of emitting multiple elements synchronously or asynchronously
• The param emits any # of next() signals followed by zero or one error() or complete() signals

```java
static <T> Flux<T> create
    (Consumer<? super FluxSink<T>> emitter)
```

Interface FluxSink<T>

Type Parameters:
T - the value type

```java
public interface FluxSink<T>
```

Wrapper API around a downstream Subscriber for emitting any number of next signals followed by zero or one onError/onComplete.

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

- The create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously
    - The param emits any # of next() signals followed by zero or one error() or complete() signals
    - Supports more dynamic use cases than the Flux just() & fromIterable() operators

See earlier lesson on “Key Factory Method Operators in the Flux Class (Part 1)”
Key Factory Method Operators in the Flux Class

- The `create()` operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously
    - The param emits any # of `next()` signals followed by zero or one `error()` or `complete()` signals
    - Returns a Flux that emits all the elements generated by the `FluxSink`

```java
static <T> Flux<T> create(Consumer<? super FluxSink<T>> emitter)
```
Key Factory Method Operators in the Flux Class

- The `create()` operator
- Create a Flux capable of emitting multiple elements synchronously or asynchronously

```java
static <T> Flux<T> generate(Supplier<T> supplier, long count) {
    return Flux.create(sink -> {
        for(int i = 0; i < count; ++i)
            sink.next(supplier.get());

        sink.complete();
    });
}
```

Synchronously generate 'count' instances of what’s returned by `supplier.get()`
Key Factory Method Operators in the Flux Class

- The create() operator
- Create a Flux capable of emitting multiple elements synchronously or asynchronously

```java
static <T> Flux<T> generate(
    Supplier<T> supplier,
    long count) {
    return Flux.create(sink -> {
        for(int i = 0; i < count; ++i)
            sink.next(supplier.get());

        sink.complete();
    });
}
```

Generate the next element & emit it
Key Factory Method Operators in the Flux Class

- The `create()` operator
- Create a Flux capable of emitting multiple elements synchronously or asynchronously

```java
static <T> Flux<T> generate(
    Supplier<T> supplier,
    long count) {
    return Flux.create(sink -> {
        for(int i = 0; i < count; ++i)
            sink.next(supplier.get());
        sink.complete();
    });
}
```

Indicate the generator is finished
Key Factory Method Operators in the Flux Class

- The create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously
  - Elements can be emitted from one or more threads
The create() operator

- Create a Flux capable of emitting multiple elements synchronously or asynchronously
- Elements can be emitted from one or more threads
- RxJava’s Flowable.create() works in a similar way
- However, the data types passed to create() differ

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Flowable.html#create](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Flowable.html#create)
Key Factory Method Operators in the Flux Class

- The `create()` operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously
  - Elements can be emitted from one or more threads
  - RxJava’s `Flowable.create()` works in a similar way
- Similar to the `generate()` method in Java Streams

```
Stream.generate(() -> BigFractionUtils.makeBigFraction(new Random(), false))
```

Generate a stream of random, large, & unreduced big fractions

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#generate](docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#generate)
The interval() operator

Create a Flux that emits long values starting with zero (0)

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#interval
Key Factory Method Operators in the Flux Class

- The interval() operator
- Create a Flux that emits long values starting with zero (0)
- The param indicates when to increment a value at the specified time interval

static Flux<Long> interval (Duration period)

<table>
<thead>
<tr>
<th>Class Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.Object</td>
</tr>
<tr>
<td>java.time.Duration</td>
</tr>
</tbody>
</table>

All Implemented Interfaces:
Serializable, Comparable<Duration>, TemporalAmount

public final class Duration extends Object
implements TemporalAmount, Comparable<Duration>, Serializable

A time-based amount of time, such as '34.5 seconds'.

This class models a quantity or amount of time in terms of seconds and nanoseconds. It can be accessed using other duration-based units, such as minutes and hours. In addition, the DAYS unit can be used and is treated as exactly equal to 24 hours, thus ignoring daylight savings effects. See Period for the date-based equivalent to this class.

See docs.oracle.com/javase/8/docs/api/java/time/Duration.html
Key Factory Method Operators in the Flux Class

- The interval() operator
- Create a Flux that emits long values starting with zero (0)
  - The param indicates when to increment a value at the specified time interval
- Returns a new Flux emitting increasing #’s at regular intervals

```
static Flux<Long> interval
(Duration period)
```
Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)
  - Emits values on the Schedulers.parallel() Scheduler

See [projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#parallel](http://projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#parallel)
Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)
  - Emits values on the Schedulers .parallel() Scheduler
  - Other overloaded interval() methods can designate the Scheduler

See [projectreactor.io/docs/core/release/api/reactor/core/scheduler/Scheduler.html](http://projectreactor.io/docs/core/release/api/reactor/core/scheduler/Scheduler.html)
Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)
  - Emits values on theSchedulers .parallel() Scheduler
  - In normal conditions, the Flux will never complete

```
Flux.interval(Duration.ofMillis(500))
```

*Generate a stream of longs every .5 seconds in a background thread*

See Reactive/Flux/ex2/src/main/java/FluxEx.java
Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)
  - Emits values on the Schedulers .parallel() Scheduler
  - In normal conditions, the Flux will never complete

Flux

```
.interval(Duration.ofMillis(500))
```

```
.take(sMAX_ITERATIONS)
```

Use take() to only process sMAX_ITERATIONS # of emitted values from interval()

See upcoming discussion of the Flux.take() method
The interval() operator

- Create a Flux that emits long values starting with zero (0)
- Emits values on the Schedulers .parallel() Scheduler
- In normal conditions, the Flux will never complete
- RxJava’s Observable.interval() works the same

Use take() to only process sMAX_ITERATIONS # of emitted values from interval()
Key Factory Method Operators in the Flux Class

• The range() operator
  
• Build a Flux that will only emit a sequence of ‘count’ incrementing integers, starting from ‘start’

 static Flux<Integer> range (int start, int count)

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#range
Key Factory Method Operators in the Flux Class

• The `range()` operator
  • Build a Flux that will only emit a sequence of `count` incrementing integers, starting from `start`
  • Emits integers between `start` & `start + count` & then completes
Key Factory Method Operators in the Flux Class

• The range() operator
  • Build a Flux that will only emit a sequence of `count’ incrementing integers, starting from `start’
  • Emits integers between `start’ & `start + count’ & then completes

static Flux<Integer> range (int start, int count)
Key Factory Method Operators in the Flux Class

- The range() operator
  - Build a Flux that will only emit a sequence of `count` incrementing integers, starting from `start`
    - Emits integers between `start` & `start + count` & then completes
  - Returns a “ranged” Flux containing count elements

```
static Flux<Integer> range
    (int start, int count)
```
Key Factory Method Operators in the Flux Class

• The range() operator
  • Build a Flux that will only emit a sequence of ‘count’ incrementing integers, starting from ‘start’
  • Works much like a “reactive” for loop

```java
final int sMAX_ITERATIONS = 10;
...
Flux
  .range(1, sMAX_ITERATIONS)
  ...
```

See Reactive/Flux/ex2/src/main/java/FluxEx.java
Key Factory Method Operators in the Flux Class

- The range() operator
  - Build a Flux that will only emit a sequence of ‘count’ incrementing integers, starting from ‘start’
  - Works much like a “reactive” for loop
  - RxJava’s Observable.range() works the same

```java
final int sMAX_ITERATIONS = 10;
...
Observable.range(1, sMAX_ITERATIONS)
...```

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#range](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#range)
The range() operator

- Build a Flux that will only emit a sequence of ‘count’ incrementing integers, starting from ‘start’
- Works much like a “reactive” for loop
- RxJava’s Observable.range() works the same
- Similar to IntStream.rangeClosed() in Java Streams

Emit sMAX_ITERATIONS integers starting at 1

IntStream.rangeClosed

(int startInclusive, int endInclusive)

Returns a sequential ordered IntStream from startInclusive (inclusive) to endInclusive (inclusive) by an incremental step of 1.

API Note:
An equivalent sequence of increasing values can be produced sequentially using a for loop as follows:

```
for (int i = startInclusive; i <= endInclusive ; i++) { ... }
```

Parameters:
- startInclusive - the (inclusive) initial value
- endInclusive - the inclusive upper bound

Returns:
a sequential IntStream for the range of int elements

See docs.oracle.com/javase/8/docs/api/java/util/stream/IntStream.html#rangeClosed
End of Key Factory Method Operators in the Flux Class (Part 2)