Key Factory Method 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
  • These operators create Flux streams in various ways
  • e.g., generate()

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 `generate()` operator
• Create a Flux by generating signals 1-by-1 via a callback

```java
static <T> Flux<T> generate
(Consumer<SynchronousSink<T>> generator)
```

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

- The `generate()` operator
- Create a Flux by generating signals 1-by-1 via a callback
- The Consumer param is called in a loop after a downstream Subscriber has subscribed

\[
\text{static } \langle T \rangle \ \text{Flux}\langle T \rangle \ \text{generate} \\
(\text{Consumer}\langle \text{SynchronousSink}\langle T \rangle \rangle \ \text{generator})
\]

### Interface Consumer\langle T \rangle

**Type Parameters:**
- \( T \) - the type of the input to the operation

**All Known Subinterfaces:**
- Stream.Builder\langle T \rangle

**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/Consumer.html](docs.oracle.com/javase/8/docs/api/java/util/function/Consumer.html)
Key Factory Method Operators in the Flux Class

- The `generate()` operator
  - Create a Flux by generating signals 1-by-1 via a callback
    - The Consumer param is called in a loop after a downstream Subscriber has subscribed
    - The callback should call `next()`, `error()`, or `complete()` on a SynchronousSink to signal a value or a terminal event

```java
static <T> Flux<T> generate
    (Consumer<SynchronousSink<T>> generator)
```

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

• The generate() operator

• Create a Flux by generating signals 1-by-1 via a callback

  • The Consumer param is called in a loop after a downstream Subscriber has subscribed

  • The new Flux instance is returned

```java
static <T> Flux<T> generate
    (Consumer<SynchronousSink<T>> generator)
```
Key Factory Method Operators in the Flux Class

- The generate() operator
  - Create a Flux by generating signals 1-by-1 via a callback
    - The Consumer param is called in a loop after a downstream Subscriber has subscribed
    - The new Flux instance is returned
      - This Flux is “cold,” which only emits item upon subscription

```java
static <T> Flux<T> generate
    (Consumer<SynchronousSink<T>> generator)
```

Key Factory Method Operators in the Flux Class

• The generate() operator
  • Create a Flux by generating signals 1-by-1 via a callback
    • The Consumer param is called in a loop after a downstream Subscriber has subscribed
  • The new Flux instance is returned
    • This Flux is “cold,” which only emits item upon subscription
    • Each observer has its own set of items emitted to it

static <T> Flux<T> generate (Consumer<SynchronousSink<T>> generator)
Key Factory Method Operators in the Flux Class

- The generate() operator
  - Create a Flux by generating signals 1-by-1 via a callback
  - It is only allowed to generate one event at a time, which supports backpressure

```java
Flux.generate((SynchronousSink<BigFraction> sink) -> sink
               .next(BigFractionUtils.makeBigFraction(sRANDOM, false)))
```

Key Factory Method Operators in the Flux Class

- The `generate()` operator
  - Create a Flux by generating signals 1-by-1 via a callback
  - It is only allowed to generate one event at a time, which supports backpressure

```java
Flux.generate(((SynchronousSink<BigFraction> sink) -> sink
  .next(BigFractionUtils.makeBigFraction(sRANDOM, false)))
```

... Generate an infinite stream of random unreduced big fractions

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

- **The generate() operator**
  - Create a Flux by generating signals 1-by-1 via a callback
  - It is only allowed to generate one event at a time, which supports backpressure

Flux

```java
generate((SynchronousSink<BigFraction> sink) -> sink
  .next(BigFractionUtils.makeBigFraction(sRANDOM, false)))
  .take(sMAX_FRACTIONS)
...```

Can be used with `take()` to limit the number of elements generated

See earlier lesson on “Key Suppressing Operators in the Flux Class”
Key Factory Method Operators in the Flux Class

- The generate() operator
  - Create a Flux by generating signals 1-by-1 via a callback
  - It is only allowed to generate one event at a time, which supports backpressure
  - In contrast, create() simply produces events whenever it wishes to do so

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 `generate()` operator
  - Create a Flux by generating signals 1-by-1 via a callback
  - It is only allowed to generate one event at a time, which supports backpressure
    - In contrast, `create()` simply produces events whenever it wishes to do so
      - i.e., it ignores backpressure

See [www.wideopeneats.com/i-love-lucy-chocolate-factory](http://www.wideopeneats.com/i-love-lucy-chocolate-factory)
The `generate()` operator

- Create a Flux by generating signals 1-by-1 via a callback
- It is only allowed to generate one event at a time, which supports backpressure
- RxJava’s `Observable.generate()` works in a similar way

```java
Observable.of()
    .generate((Emitter<BigFraction> emit) -> emit.onNext(BigFractionUtils.makeBigFraction(sRANDOM, false))) ...
```

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

- The `generate()` operator
  - Create a Flux by generating signals 1-by-1 via a callback
  - It is only allowed to generate one event at a time, which supports backpressure
  - RxJava’s `Observable.generate()` works the same
  - Similar to `Stream.generate()` in Java Streams

```java
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](http://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#generate)
End of Key Factory Method Operators in the Flux Class (Part 3)