Key Factory Method Operators in the Flowable 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 operators defined in—or used with—Flowable
- Factory method operators
  - These operators create Flowable streams in various ways
    - e.g., `create()` & `generate()`

See [en.wikipedia.org/wiki/Factory_method_pattern](en.wikipedia.org/wiki/Factory_method_pattern)
Key Factory Method
Operators in the Flowable Class
### The `generate()` operator

```java
public static <T> @NonNull Flowable<? extends T> generate(@NonNull Consumer<Emitter<T>> generator)
```

Returns a cold, synchronous, stateless and backpressure-aware generator of values.

Note that the `Emitter.onNext(T)`, `Emitter.onError(java.lang.Throwable)` and `Emitter.onComplete()` methods provided to the function via the `Emitter` instance should be called synchronously, never concurrently and only while the function body is executing. Calling them from multiple threads or outside the function call is not supported and leads to an undefined behavior.

**Backpressure:**
The operator honors downstream backpressure.

**Scheduler:**
generate does not operate by default on a particular `Scheduler`.

**Type Parameters:**
- `T` - the generated value type

**Parameters:**
generator - the `Consumer` called whenever a particular downstream `Subscriber` has requested a value. The callback then should call `onNext`, `onError` or `onComplete` to signal a value or a terminal event. Signaling multiple `onNext` in a call will make the operator signal `IllegalStateException`.

**Returns:**
the new `Flowable` instance

**Throws:**
- `NullPointerException` if `generator` is null

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

- The `generate()` operator
  - Returns a synchronous, stateless, & backpressure-aware generator of values

```java
static <T> Flowable<T> generate
(Consumer<Emitter<T>> generator)
```

*Flowable.generate() is quite different from Observable.generate()*
• The \texttt{generate()} operator
• Returns a synchronous, stateless, & backpressure-aware generator of values
  • The param is called when a downstream Subscriber requests a value

\textbf{Key Factory Method Operators in the Flowable Class}

\begin{verbatim}
static <T> Flowable<T> generate
  (Consumer<Emitter<T>> generator)
\end{verbatim}

\texttt{Consumer}<\texttt{T}>

\textbf{Method Summary}

\begin{tabular}{|c|c|}
  \hline
  Modifier and Type & Method and Description \tabularnewline
  \hline
  void & \texttt{accept(T t)} \tabularnewline
  & Consume the given value. \tabularnewline
  \hline
\end{tabular}

See \url{reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/functions/Consumer.html}
Key Factory Method Operators in the Flowable Class

- The generate() operator
  - Returns a synchronous, stateless, & backpressure-aware generator of values
    - The param is called when a downstream Subscriber requests a value
    - The Emitter should call onNext(), onError(), or onComplete() to signal a value or a terminal event

```java
static <T> Flowable<T> generate
(Consumer<Emitter<T>> generator)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Emitter.html
Key Factory Method Operators in the Flowable Class

- The generate() operator
  - Returns a synchronous, stateless, & backpressure-aware generator of values
    - The param is called when a downstream Subscriber requests a value
    - The Emitter should call onNext(), onError(), or onComplete() to signal a value or a terminal event
    - Only call these methods synchronously, never concurrently, & only while the method body is executing

static <T> Flowable<T> generate(Consumer<Emitter<T>> generator)
The generate() operator

- Returns a synchronous, stateless, & backpressure-aware generator of values
  - The param is called when a downstream Subscriber requests a value
  - Returns a new Flowable instance

```
static <T> Flowable<T> generate(Consumer<Emitter<T>> generator)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Flowable.html
Key Factory Method Operators in the Flowable Class

The `generate()` operator
- Returns a synchronous, stateless, & backpressure-aware generator of values
  - The param is called when a downstream Subscriber requests a value
- Returns a new Flowable instance
  - This Flowable is “cold,” which only emits items upon subscription

```java
static <T> Flowable<T> generate (Consumer<Emitter<T>> generator)
```

The generate() operator

- Returns a synchronous, stateless, & backpressure-aware generator of values
  - The param is called when a downstream Subscriber requests a value
- Returns a new Flowable instance
  - This Flowable is “cold,” which only emits items upon subscription
  - Each Flowable has its own set of items emitted to it

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

- The `generate()` operator
  - Returns a synchronous, stateless, & backpressure-aware generator of values
  - The param is invoked to emit the given # of events when a Subscriber calls `request()` on a Subscription

```java
Flowable
  .generate(emitter -> {
    if (sIntegersEmitted++ < count)
      emitter.onNext(random.nextInt(maxValue));
    else
      emitter.onComplete();
  })
```

```java
void onNext() {
  ... 
  mSubscription
    .request(mRequestSize);
  ...
}
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Flowable/ex2](github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Flowable/ex2)
Key Factory Method Operators in the Flowable Class

- The `generate()` operator
  - Returns a synchronous, stateless, & backpressure-aware generator of values
  - The param is invoked to emit the given # of events when a Subscriber calls request() on a Subscription

```java
Flowable
    .generate(emitter -> {
        if (sIntegersEmitted++ < count)
            emitter.onNext(random
                .nextInt(maxValue));
        else
            emitter.onComplete();
    })
```

```java
void onNext() {
    ...
    mSubscription
        .request(mRequestSize);
    ...
}
```

A Subscriber requests the next tranche of events
The `generate()` operator

- Returns a synchronous, stateless, & backpressure-aware generator of values
- The param is invoked to emit the given # of events when a Subscriber calls request() on a Subscription

```java
void onNext() {
    ... 
    mSubscription
        .request(mRequestSize);
    ... 
}
```

```
Flowable
  .generate(emitter -> {
    if (sIntegersEmitted++ < count))
      emitter.onNext(random
          .nextInt(maxValue));
    else
      emitter.onComplete();
  })
  ... 
```

_A Publisher’s Consumer Emitter is called back mRequestSize times or until it emits onComplete()_
Key Factory Method Operators in the Flowable Class

• The `generate()` operator
  • Returns a synchronous, stateless, & backpressure-aware generator of values
  • The param is invoked to emit the given # of events when a Subscriber calls `request()` on a Subscription
• Project Reactor’s `Flux.create()` operator works in a similar way

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 Flowable Class

• The `generate()` operator
  • Returns a synchronous, stateless, & backpressure-aware generator of values
  • The param is invoked to emit the given # of events when a Subscriber calls `request()` on a Subscription

• Project Reactor’s `Flux.create()` operator works in a similar way
  • However, this operator supports both backpressure-aware publishers & backpressure strategies

---

See [jstobigdata.com/java/backpressure-in-project-reactor](jstobigdata.com/java/backpressure-in-project-reactor)
Key Factory Method Operators in the Flowable Class

- **The `generate()` operator**
  - Returns a synchronous, stateless, & backpressure-aware generator of values
  - The `param` is invoked to emit the given # of events when a Subscriber calls `request()` on a Subscription

- **Project Reactor’s `Flux.create()` operator** works in a similar way
  - However, this operator supports both backpressure-aware publishers & backpressure strategies
  - `Flux.generator()` is like `Observable.generate()` & lacks backpressure support

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

- **The generate() operator**
  - Returns a synchronous, stateless, & backpressure-aware generator of values
  - The param is invoked to emit the given # of events when a Subscriber calls request() on a Subscription
  - Project Reactor’s Flux.create() operator works in a similar way
  - Java Streams generate() method doesn’t support backpressure

```java
generate
static <T> Stream<T> generate(Supplier<T> s)
Returns an infinite sequential unordered stream where each element is generated by the provided Supplier. This is suitable for generating constant streams, streams of random elements, etc.

Type Parameters:
T - the type of stream elements

Parameters:
s - the Supplier of generated elements

Returns:
a new infinite sequential unordered Stream
```
The **generate()** operator

- Returns a synchronous, stateless, & backpressure-aware generator of values
- The param is invoked to emit the given # of events when a Subscriber calls `request()` on a Subscription
- Project Reactor’s `Flux.create()` operator works in a similar way

Java Streams `generate()` method doesn’t support backpressure

- However, it is a “pull-based” model rather than “push-based” pub/sub model, so backpressure support is not necessary
End of Key Factory Method Operators in the Flowable Class (Part 2)