Key Scheduler Operators in the Observable 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 Observable operators
  - Factory method operators
  - Transforming operators
- Concurrency & scheduler operators
  - These operators arrange to run other operators in designated threads & thread pools
  - e.g., Schedulers.computation()
Key Scheduler Operators for the Observable Class
Key Scheduler Operators for the Observable Class

- The `Schedulers.computation()` operator
  - Hosts a fixed-size pool of single-threaded Executor Service-based workers

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/schedulers/Schedulers.html#computation](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/schedulers/Schedulers.html#computation)
Key Scheduler Operators for the Observable Class

- The Schedulers.computation() operator
  - Hosts a fixed-size pool of single-threaded Executor Service-based workers
  - Returns a new Scheduler that is suited for parallel work
Key Scheduler Operators for the Observable Class

- The Schedulers.computation() operator
  - Hosts a fixed-size pool of single-threaded Executor Service-based workers
  - Returns a new Scheduler that is suited for parallel work
  - Optimized for fast Runnable non-blocking executions

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/schedulers/Schedulers.html
Key Scheduler Operators for the Observable Class

- The Schedulers.computation() operator
  - Hosts a fixed-size pool of single-threaded Executor Service-based workers
  - Returns a new Scheduler that is suited for parallel work
  - Optimized for fast Runnable non-blocking executions
    - i.e., compute-/CPU-bound
    - *not* I/O-bound!

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/schedulers/Schedulers.html
Key Scheduler Operators for the Observable Class

- The Schedulers.computation() operator
  - Hosts a fixed-size pool of single-threaded Executor Service-based workers
  - Used for event-loops, processing callbacks, & other computational work

Arrange to multiply a List of Big Integer objects in a background thread in computation thread pool

```java
return Observable
  .fromIterable(bigFractionList)
  .flatMap(bf -> Observable
    .fromCallable(() -> bf
      .multiply(sBigFraction))
    .subscribeOn
    (Schedulers
     .computation()))
  .reduce(BigFraction::add)
```

See Reactive/Observable/ex3/src/main/java/ObservableEx.java
Key Scheduler Operators for the Observable Class

- The Schedulers.computation() operator
  - Hosts a fixed-size pool of single-threaded Executor Service-based workers
  - Used for event-loops, processing callbacks, & other computational work

```
return Observable
  .fromIterable(bigFractionList)
    .flatMap(bf -> Observable
      .fromCallable(() -> bf
        .multiply(sBigFraction))
      .subscribeOn
        (Schedulers
          .computation()))
    .reduce(BigFraction::add)
```

*Each BigFraction emitted via from Callable() is multiplied in parallel within the computation thread pool*
Key Scheduler Operators for the Observable Class

- The Schedulers.computation() operator

- Hosts a fixed-size pool of single-threaded Executor Service-based workers

- Used for event-loops, callbacks, & other computational work

```java
return Observable
  .fromIterable(bigFractionList)
  .flatMap(bf -> Observable
    .fromCallable(() -> bf
      .multiply(sBigFraction))
    .subscribeOn
      (Schedulers.computation()))
  .reduce(BigFraction::add)
```

*fromCallable() is a “lazy” factory method so multiply() runs in the computation thread pool even though subscribeOn() comes after*
Key Scheduler Operators for the Observable Class

- The Schedulers.computation() operator
  - Hosts a fixed-size pool of single-threaded Executor Service-based workers
  - Used for event-loops, callbacks, & other computational work

```java
return Observable
  .fromIterable(bigFractionList)
    .flatMap(bf -> Observable
                    .fromCallable(() -> bf
                        .multiply(sBigFraction))
          .subscribeOn
            (Schedulers.computation()))
  .reduce(BigFraction::add)
```

Only one thread runs reduce() after all other computations are done
Key Scheduler Operators for the Observable Class

- The Schedulers.computation() operator
  - Hosts a fixed-size pool of single-threaded Executor Service-based workers
  - Used for event-loops, callbacks, & other computational work
  - Implemented via “daemon threads”
    - i.e., won’t prevent the app from exiting even if its work isn’t done

See [www.baeldung.com/java-daemon-thread](http://www.baeldung.com/java-daemon-thread)
The Schedulers.computation() operator
- Hosts a fixed-size pool of single-threaded Executor Service-based workers
- Used for event-loops, callbacks, & other computational work
- Implemented via “daemon threads”
- The Schedulers.parallel() operator in Project Reactor is similar
  - i.e., intended for computer-/CPU-bound tasks, not I/O-bound tasks

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#parallel
Key Schedulers Operators for the Observable Class

- **The Schedulers.computation() operator**
  - Hosts a fixed-size pool of single-threaded Executor Service-based workers
  - Used for event-loops, callbacks, & other computational work
  - Implemented via “daemon threads”
  - The Schedulers.parallel() operator in Project Reactor is similar
  - The Java common fork-join pool is also similar

---

```
public static ForkJoinPool commonPool()

Returns the common pool instance. This pool is statically constructed; its run state is unaffected by attempts to 
shutdown() or shutdownNow(). However this pool and any 
ongoing processing are automatically terminated upon program 
System.exit(int). Any program that relies on asynchronous 
task processing to complete before program termination should 
invoke commonPool().awaitQuiescence, before exit.

Returns:
the common pool instance
```

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html#commonPool](docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html#commonPool)
Key Scheduler Operators for the Observable Class

• The Schedulers.computation() operator
  • Hosts a fixed-size pool of single-threaded Executor Service-based workers
  • Used for event-loops, callbacks, & other computational work
  • Implemented via “daemon threads”
  • The Schedulers.parallel() operator in Project Reactor is similar
  • The Java common fork-join pool is also similar
  • However, ManagedBlocker enables it to also work with I/O-bound tasks

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.ManagedBlocker.html
End of Key Scheduler Operators in the Observable Class (Part 2)