Key Concurrency & Scheduler

Operators in the Flux Class (Part 1)

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
  - These operators arrange to run other operators in designated threads & thread pools
    - e.g., `subscribeOn()`, `publishOn()`, & `Schedulers.newParallel()`
Key Concurrency Operators in the Flux Class
Key Concurrency Operators in the Flux Class

- The `subscribeOn()` operator
  - Run `subscribe()`, `onSubscribe()`, & `request()` on the specified Scheduler param

```java
Flux<T> subscribeOn
    (Scheduler scheduler)
```

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

• The subscribeOn() operator
  • Run subscribe(), onSubscribe(), & request() on the specified Scheduler param
  • The scheduler param indicates what thread to perform the operation on

Flux<T> subscribeOn
(Scheduler scheduler)

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Scheduler.html
Key Concurrency Operators in the Flux Class

- The `subscribeOn()` operator
  - Run `subscribe()`, `onSubscribe()`, & `request()` on the specified `Scheduler` param
    - The `scheduler` param indicates what thread to perform the operation on
  - Returns the Flux requesting async processing

```java
Flux<T> subscribeOn
    (Scheduler scheduler)
```
Key Concurrency Operators in the Flux Class

- The `subscribeOn()` operator
  - Run `subscribe()`, `onSubscribe()`, & `request()` on the specified Scheduler param
- The `subscribeOn()` semantics are a bit unusual
Key Concurrency Operators in the Flux Class

- The `subscribeOn()` operator
- Run `subscribe()`, `onSubscribe()`, and `request()` on the specified Scheduler param
- The `subscribeOn()` semantics are a bit unusual
- Placing this operator in a chain impacts the execution context of `onNext()`, `onError()`, and `onComplete()` signals

```java
Scheduler publisher = Schedulers.newParallel("publisher", 1));

Flux
    .range(1, sMAX_ITERATIONS)
    .subscribeOn(publisher)
    .map(__ -> BigInteger.valueOf(lowerBound + rand
        .nextInt(sMAX_ITERATIONS)))
    ...
    .doFinally(() -> publisher
dispose())
    .subscribe(sink::next,
        err -> sink
            .complete(),
        sink::complete);
```

See Reactive/flux/ex2/src/main/java/FluxEx.java
### Key Concurrency Operators in the Flux Class

- **The subscribeOn() operator**
- Run subscribe(), onSubscribe(), & request() on the specified Scheduler param
- **The subscribeOn() semantics** are a bit unusual
- Placing this operator in a chain impacts the execution context of onNext(), onError(), & onComplete() signals

```
Scheduler publisher = Schedulers
    .newParallel("publisher", 1));

Flux
    .range(1, sMAX_ITERATIONS)
    .map(__ -> BigInteger
        .valueOf(lowerBound + rand
            .nextInt(sMAX_ITERATIONS)))
    ... 
    .doFinally(() -> publisher
        .dispose())
    .subscribeOn(publisher)
    .subscribe(sink::next,
        err -> sink
            .complete(),
        sink::complete);
```

*subscribeOn() can appear later in the chain & have the same effect*
Key Concurrency Operators in the Flux Class

- The subscribeOn() operator
  - Run subscribe(), onSubscribe(), & request() on the specified Scheduler param
- The subscribeOn() semantics are a bit unusual
  - Placing this operator in a chain impacts the execution context of onNext(), onError(), & onComplete() signals
  - However, if a publishOn() operator appears later in the chain that can change the threading context where the rest of the operators in the chain below it execute (publishOn() can appear multiple times)

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#publishOn
Key Concurrency Operators in the Flux Class

- The `subscribeOn()` operator
  - Run `subscribe()`, `onSubscribe()`, & `request()` on the specified Scheduler param
- The `subscribeOn()` semantics are a bit unusual
- RxJava’s Observable. `subscribeOn()` works the same way

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#subscribeOn](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#subscribeOn)
Key Concurrency Operators in the Flux Class

- The `publishOn()` operator
- Run `onNext()`, `onComplete()`, & `onError()` on a supplied Scheduler param

```java
Flux<T> publishOn(Scheduler scheduler)
```
Key Concurrency Operators in the Flux Class

- The `publishOn()` operator
- Run `onNext()`, `onComplete()`, & `onError()` on a supplied Scheduler param
- The scheduler param indicates what thread to perform the operation on

```java
Flux<T> publishOn(Scheduler scheduler)
```

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Scheduler.html
Key Concurrency Operators in the Flux Class

- The `publishOn()` operator
- Run `onNext()`, `onComplete()`, & `onError()` on a supplied Scheduler param
  - The scheduler param indicates what thread to perform the operation on
- Returns the Flux requesting async processing
Key Concurrency Operators in the Flux Class

- The publishOn() operator
  - Run onNext(), onComplete(), & onError() on a supplied Scheduler param
- The publishOn() semantics are fairly straightforward
The `publishOn()` operator

- Run `onNext()`, `onComplete()`, & `onError()` on a supplied Scheduler param

- The `publishOn()` semantics are fairly straightforward
- It influences the threading context where the rest of the operators in the chain below it execute

```java
Scheduler subscriber = Schedulers
    .newParallel("subscriber", 1));
return Flux
    .create(makeAsyncFluxSink(sb))
    .publishOn(subscriber)
    .map(bigInteger -> FluxEx
        .checkIfPrime(bigInteger, sb))
    .doOnNext(bigInteger -> FluxEx
        .processResult(bigInteger, sb))
    .doFinally(__ ->
        subscriber.dispose())
...
Key Concurrency Operators in the Flux Class

- The `publishOn()` operator
  - Run `onNext()`, `onComplete()`, & `onError()` on a supplied Scheduler param

- The `publishOn()` semantics are fairly straightforward
  - It influences the threading context where the rest of the operators in the chain below it execute
  - Up to any new occurrence of `publishOn()` (if any)

Beware of publishing on too many different threads!

```java
Scheduler subscriber = Schedulers.newParallel("subscriber", 2));
return Flux.create(makeAsyncFluxSink(sb)).
publishOn(subscriber).
.map(bigInteger -> FluxEx.checkIfPrime(bigInteger, sb))
.publishOn(subscriber).
doOnNext(bigInteger -> FluxEx.processResult(bigInteger, sb))
...```
Key Concurrency Operators in the Flux Class

- The publishOn() operator
  - Run onNext(), onComplete(), & onError() on a supplied Scheduler param
- The publishOn() semantics are fairly straightforward
  - It influences the threading context where the rest of the operators in the chain below it execute
- Interactions between publishOn() & subscribeOn() are convoluted..

See www.woolha.com/tutorials/project-reactor-publishon-vs-subscribeon-difference
Key Concurrency Operators in the Flux Class

- The `publishOn()` operator
  - Run `onNext()`, `onComplete()`, & `onError()` on a supplied Scheduler param
- The `publishOn()` semantics are fairly straightforward
- RxJava’s `Observable.observeOn()` operator works the same

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#observeOn](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#observeOn)
Key Concurrency Operators in the Flux Class

• The publishOn() operator
  • Run onNext(), onComplete(), & onError() on a supplied Scheduler param
• The publishOn() semantics are fairly straightforward
  • RxJava’s Observable.observeOn() operator works the same
    • Why RxJava & Project Reactor chose different names for this operator is a mystery..

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#observeOn
Key Scheduler Operators Used By the Flux Class
Key Scheduler Operators Used By the Flux Class

• The Schedulers.newParallel() operator
• Hosts a fixed-sized pool of single-threaded ExecutorService-based workers

static Scheduler newParallel
(String name,
   int parallelism)

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#newParallel
Key Scheduler Operators Used By the Flux Class

- The Schedulers.newParallel() operator
  - Hosts a fixed-sized pool of single-threaded ExecutorService-based workers
  - The params (1) give a name for the scheduler & (2) indicate the # of pooled worker threads

```java
static Scheduler newParallel(String name, int parallelism)
```
The Schedulers.newParallel() operator

- Hosts a fixed-sized pool of single-threaded ExecutorService-based workers
  - The params (1) give a name for the scheduler & (2) indicate the # of pooled worker threads
  - Returns a new Scheduler suitable for parallel computations

static Scheduler newParallel
    (String name,
    int parallelism)
Key Scheduler Operators Used By the Flux Class

- The Schedulers.newParallel() operator
  - Hosts a fixed-sized pool of single-threaded ExecutorService-based workers
    - The params (1) give a name for the scheduler & (2) indicate the # of pooled worker threads
  - Returns a new Scheduler suitable for parallel computations
    - However, it detects & rejects use of blocking Reactor APIs

See [projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html](projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html)
The Schedulers.newParallel() operator

- Hosts a fixed-sized pool of single-threaded ExecutorService-based workers
- Can be used to create a custom parallel scheduler

```java
Scheduler publisher = Schedulers.newParallel("publisher", 1);
Flux
    .range(1, sMAX_ITERATIONS)
    .map(Integer::toUnsignedLong)
    .subscribeOn(publisher)
    .map(sGenerateRandomBigInt)
    .filter(sOnlyOdd)
    .doFinally(() -> publisher.dispose())
    .subscribe(sink::next,
        error ->
            sink.complete(),
        sink::complete);
```

Arrange to emit the random big integers in the "publisher" thread

See Reactive/flux/ex2/src/main/java/FluxEx.java
Key Scheduler Operators Used By the Flux Class

- The `Schedulers.newParallel()` operator
  - Hosts a fixed-sized pool of single-threaded `ExecutorService`-based workers
  - Can be used to create a custom parallel scheduler
    - Not implemented via a “daemon thread”

See [www.baeldung.com/java-daemon-thread](http://www.baeldung.com/java-daemon-thread)
Key Scheduler Operators Used By the Flux Class

- The Schedulers.newParallel() operator
  - Hosts a fixed-sized pool of single-threaded ExecutorService-based workers
  - Can be used to create a custom parallel scheduler
  - Not implemented via a “daemon thread”
    - i.e., the app will not exit until this pool is disposed of properly & explicitly

```java
Scheduler publisher = Schedulers.newParallel("publisher", 1));
Flux
  .range(1, sMAX_ITERATIONS)
  .map(Integer::toUnsignedLong)
  .subscribeOn(publisher)
  .map(sGenerateRandomBigInt)
  .filter(sOnlyOdd)
  .doFinally(() -> publisher.dispose())
  .subscribe(sink::next,
    error ->
      sink.complete(),
    sink::complete);
```

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Scheduler.html#dispose
• The Schedulers.newParallel() operator
  • Hosts a fixed-sized pool of single-threaded ExecutorService-based workers
  • Can be used to create a custom parallel scheduler
• RxJava’s Schedulers doesn’t have an equivalent method
TheSchedulers.newParallel() operator

- Hosts a fixed-sized pool of single-threaded ExecutorService-based workers
- Can be used to create a custom parallel scheduler

RxJava’s Schedulers doesn’t have an equivalent method

- However, the from() method can be used in conjunction with Java’s Executor framework

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/schedulers/Schedulers.html#from
Key Scheduler Operators Used By the Flux Class

- The Schedulers.newParallel() operator
  - Hosts a fixed-sized pool of single-threaded ExecutorService-based workers
  - Can be used to create a custom parallel scheduler
- RxJava’s Schedulers doesn’t have an equivalent method
  - However, the from() method can be used in conjunction with Java’s Executor framework, e.g.

See docs.oracle.com/javase/tutorial/essential/concurrency/pools.html
End of Key Concurrency & Scheduler Operators in the Flux Class (Part 1)