Key Concurrency & Scheduler Operators for the Observable 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 Observable operators
• Concurrency & scheduler operators
  • These operators arrange to run other operators in designated threads & thread pools
  • e.g., subscribeOn(),observeOn(), & Schedulers.newThread()
Key Concurrency Operators in the Observable Class
Key Concurrency Operators in the Observable Class

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

Observable<T> subscribeOn(Scheduler scheduler)

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

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

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

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Scheduler.html
Key Concurrency Operators in the Observable Class

- The `subscribeOn()` operator
- Run the `subscribe()`, `request()`, & `onSubscribe()` methods on the specified Scheduler worker
  - The scheduler param indicates what thread to perform the operation on
  - Scheduler is parameterized so that these mechanisms can also be reused in the Single class

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

The subscribeOn() operator

- Run the subscribe(), request(), & onSubscribe() methods on the specified Scheduler worker
  - The scheduler param indicates what thread to perform the operation on
- Returns the Observable requesting async processing
Key Concurrency Operators in the Observable Class

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

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

```java
Observable
  .range(1, sMAX_ITERATIONS)
  .subscribeOn(Schedulers.newThread())
  .map(__ -> BigInteger.valueOf(lowerBound + rand.nextInt(sMAX_ITERATIONS)))
  .doOnNext(s ->
    ObservableEx.print(s, sb))
  .subscribe(emitter::next,
    error ->
    emitter.complete(),
    emitter::complete);
```

See Reactive/Observable/ex2/src/main/java/ObservableEx.java
Key Concurrency Operators in the Observable Class

- **The subscribeOn() operator**
  - Run the subscribe(), request(), & onSubscribe() methods on the specified Scheduler worker.

- **The subscribeOn() semantics** are a bit unusual.
  - Placing this operator in a chain impacts the execution context of the onNext(), onError(), & onComplete() signals.

```
Observable
  .range(1, sMAX_ITERATIONS)
  .map(__ -> BigInteger
       .valueOf(lowerBound + rand
           .nextInt(sMAX_ITERATIONS)))
  .doOnNext(s ->
            ObservableEx.print(s, sb))
  .subscribeOn(Schedulers
           .newThread())
  .subscribe(emitter::next,
                  error ->
                       emitter.complete(),
                  emitter::complete);
```

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

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

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 Observable Class

• The subscribeOn() operator
  • Run the subscribe(), request(), & onSubscribe() methods on the specified Scheduler worker
  • The subscribeOn() semantics are a bit unusual
• Project Reactor’s operator Flux. subscribeOn() works the same

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

- The observeOn() operator
- Run the onNext(), onComplete(), & onError() methods on a supplied Scheduler worker

```java
Observable<T> observeOn(Scheduler scheduler)
```

Key Concurrency Operators in the Observable Class

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

```
Observable<T>
observeOn(Scheduler scheduler)
```

---

Class Scheduler

```java
java.lang.Object
io.reactivex.rxjava3.core.Scheduler
```

**Direct Known Subclasses:**
- TestScheduler

```
public abstract class Scheduler
extends Object

A Scheduler is an object that specifies an API for scheduling units of work provided in the form of Runnables to be executed without delay (effectively as soon as possible), after a specified time delay or periodically and represents an abstraction over an asynchronous boundary that ensures these units of work get executed by some underlying task-execution scheme (such as custom Threads, event loop, Executor or Actor system) with some uniform properties and guarantees regardless of the particular underlying scheme.
```

Key Concurrency Operators in the Observable Class

- The `observeOn()` operator
- Run the `onNext()`, `onComplete()`, & `onError()` methods on a supplied Scheduler worker
  - The scheduler param indicates what thread to perform the operation on
- Returns the Observable requesting async processing

```java
Observable<T> observeOn(Scheduler scheduler)
```
Key Concurrency Operators in the Observable Class

- The `observeOn()` operator
  - Run the `onNext()`, `onComplete()`, & `onError()` methods on a supplied Scheduler worker
- The `observeOn()` semantics are fairly straightforward
Key Concurrency Operators in the Observable Class

- The `observeOn()` operator
  - Run the `onNext()`, `onComplete()`, & `onError()` methods on a supplied Scheduler worker
- The `observeOn()` semantics are fairly straightforward
  - It influences the threading context where the rest of the operators in the chain below it execute
  - i.e., up to a new occurrence of `observeOn()` in a chain (if any)

```java
defollect Observable
    .create(ObservableEx::emitAsync)
    .observeOn(Schedulers.newThread())
    .map(bi -> ObservableEx
        .checkIfPrime(bi, sb))
    .doOnNext(bi -> ObservableEx
        .processResult(bi, sb))
    .doOnComplete(() ->
        BigFractionUtils.display(sb.toString())
    ).count()
    .ignoreElement();
```

See Reactive/Observable/ex2/src/main/java/ObservableEx.java
Key Concurrency Operators in the Observable Class

- The `observeOn()` operator
  - Run the `onNext()`, `onComplete()`, & `onError()` methods on a supplied Scheduler worker
- The `observeOn()` semantics are fairly straightforward
- Project Reactor’s operator Flux. `publishOn()` works the same

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

- The observeOn() operator
  - Run the onNext(), onComplete(), & onError() methods on a supplied Scheduler worker
  - The observeOn() semantics are fairly straightforward
- Project Reactor’s operator Flux. publishOn() works the same
  - It’s unclear why this operator is named differently from RxJava’s observeOn() operator
Key Scheduler Operators for the Observable Class
Key Scheduler Operators for the Observable Class

- The Schedulers.newThread() operator creates a new, single-threaded ScheduledExecutorService on each invocation.

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

- The Schedulers.newThread() operator
  - Creates a new, single-threaded ScheduledExecutorService on each invocation
  - Returns a shared Scheduler instance that creates a new Thread for each unit of work

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

• The Schedulers.newThread() operator
  • Creates a new, single-threaded ScheduledExecutorService on each invocation
  • This thread is not reused, so it’s only intended for limited situations

See www.baeldung.com/rxjava-schedulers
Key Scheduler Operators for the Observable Class

- The Schedulers.newThread() operator
  - Creates a new, single-threaded ScheduledExecutorService on each invocation
  - This thread is not reused, so it’s only intended for limited situations

```java
... Observable
  .rangeLong(1, sMAX_ITERATIONS)
  .subscribeOn(Schedulers.newThread())
  .map(sGenRandomBigInteger)
... Arrange to emit the random big integers in a new "publisher" thread
.map(sGenRandomBigInteger)
... See Reactive/Observable/ex2/src/main/java/ObservableEx.java
```
Key Scheduler Operators for the Observable Class

• The Schedulers.newThread() operator
  • Creates a new, single-threaded ScheduledExecutorService on each invocation
  • This thread is not reused, so it’s only intended for limited situations
  • When the work is done, the thread is terminated
Key Scheduler Operators for the Observable Class

- The Schedulers.newThread() operator
  - Creates a new, single-threaded ScheduledExecutorService on each invocation
  - This thread is not reused, so it’s only intended for limited situations
  - When the work is done, the thread is terminated
- Project Reactor’s Schedulers. newSingle() operator is similar

See [projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#newSingle](projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#newSingle)
Key Scheduler Operators for the Observable Class

- The `Schedulers.newThread()` operator
  - Creates a new, single-threaded `ScheduledExecutorService` on each invocation
  - This thread is not reused, so it’s only intended for limited situations
  - When the work is done, the thread is terminated
- Project Reactor’s `Schedulers.newSingle()` operator is similar
  - However, its return value must be disposed of properly

See [projectreactor.io/docs/core/snapshot/api/reactor/core/Disposable.html](projectreactor.io/docs/core/snapshot/api/reactor/core/Disposable.html)
End of Key Concurrency & Scheduler Operators for the Observable Class (Part 1)