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

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

See [projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#parallel](http://projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#parallel)
Key Scheduler Operators in the Flux Class

- The Schedulers.parallel() operator
- Hosts a fixed pool of single-threaded ExecutorService-based workers
- Returns a new Scheduler that is suited for parallel work
Key Scheduler Operators in the Flux Class

- The Schedulers.parallel() operator
  - Hosts a fixed pool of single-threaded ExecutorService-based workers
  - Returns a new Scheduler that is suited for parallel work
  - Size obtained by system property reactor.schedulers.defaultPoolSize

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

- The Schedulers.parallel() operator
  - Hosts a fixed pool of single-threaded ExecutorService-based workers
  - Returns a new Scheduler that is suited for parallel work
  - Size obtained by system property reactor.schedulers.defaultPoolSize
  - Falls back to # of processors available to the runtime on init

See docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html
The Schedulers.parallel() operator
- Hosts a fixed pool of single-threaded ExecutorService-based workers
- Returns a new Scheduler that is suited for parallel work
  - Size obtained by system property reactor.schedulers.defaultPoolSize
- Optimized for computation-intensive non-blocking tasks due to its fixed-size

Key Scheduler Operators in the Flux Class

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html
The Schedulers.parallel() operator
- Hosts a fixed pool of single-threaded ExecutorService-based workers
- Returns a new Scheduler that is suited for parallel work
  - Size obtained by system property reactor.schedulers.defaultPoolSize
- Optimized for computation-intensive non-blocking tasks due to its fixed-size
  - i.e., compute-/CPU-bound tasks, not I/O-bound tasks!

Class Schedulers

```java
java.lang.Object
reactor.core.scheduler.Schedulers
```

public abstract class Schedulers
extends Object

Schedulers provides various Scheduler flavors usable by publishOn or subscribeOn:
- `parallel()`: Optimized for fast Runnable non-blocking executions
- `single()`: Optimized for low-latency Runnable one-off executions
- `elastic()`: Optimized for longer executions, an alternative for blocking tasks where the number of active tasks (and threads) can grow indefinitely
- `boundedElastic()`: Optimized for longer executions, an alternative for blocking tasks where the number of active tasks (and threads) is capped
- `immediate()`: to immediately run submitted Runnable instead of scheduling them (somewhat of a no-op or "null object" Scheduler)
- `fromExecutorService(ExecutorService)`: to create new instances around Executors
The Schedulers.parallel() operator

- Hosts a fixed pool of single-threaded ExecutorService-based workers
- Used for event-loops, callbacks, & other computational work

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

```java
Flux
  .fromIterable(bigFractions)
  .flatMap(bf -> Mono
    .fromCallable(() -> bf
      .multiply(bf)))
  .subscribeOn
    (Schedulers.parallel())
  .reduce(BigFraction::add)
```

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

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

Flux
  .fromIterable(bigFractions)
  .flatMap
    (bf -> Mono
     .fromCallable(() -> bf
       multiply(bf)))
  .subscribeOn
    (Schedulers.parallel())
  .reduce(BigFraction::add)

Each BigFraction emitted via from Callable() is multiplied in parallel within the computation thread pool

See Reactivefluxex3srcmainjavafxFluxEx.java
Key Scheduler Operators in the Flux Class

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

Flux
.fromIterable(bigFractions)
.flatMap
  (bf -> Mono
    .fromCallable(() -> bf
      multiply(bf)))
.subscribeOn
  (Schedulers.parallel()))
.reduce(BigFraction::add)

fromCallable() is a “lazy” factory method so multiply() runs in the computation thread pool even though subscribeOn() comes after
The Schedulers.parallel() operator
- Hosts a fixed pool of single-threaded ExecutorService-based workers
- Used for event-loops, callbacks, & other computational work

```
Flux
    .fromIterable(bigFractions)

    .flatMap
        (bf -> Mono
            .fromCallable(() -> bf
                multiply(bf)))

    .subscribeOn
        (Schedulers.parallel())

    .reduce(BigFraction::add)
```

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

• The Schedulers.parallel() operator
  • Hosts a fixed pool of single-threaded ExecutorService-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
Key Scheduler Operators in the Flux Class

- The Schedulers.parallel() operator
  - Hosts a fixed pool of single-threaded ExecutorService-based workers
  - Used for event-loops, callbacks, & other computational work
  - Implemented via “daemon threads”
- RxJava’s Schedulers.computation() works in a similar way
  - i.e., it’s fixed-size & intended for compute-intensive & non-blocking tasks

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

- The Schedulers.parallel() operator
  - Hosts a fixed pool of single-threaded ExecutorService-based workers
  - Used for event-loops, callbacks, & other computational work
  - Implemented via “daemon threads”
  - RxJava’s Schedulers.computation() works in a similar way
- The Java common fork-join pool is also similar wrt CPU-bound tasks

### commonPool

```java
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](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html#commonPool)
End of Key Concurrency & Scheduler Operators in the Flux Class (Part 2)