Key Scheduler Operators for Project Reactor Reactive Types (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
  - Scheduler operators
    - These operators arrange to run other operators in designated threads & thread pools
      - e.g., Schedulers.parallel()

Key Scheduler Operators for Project Reactor Reactive Types

Key Scheduler Operators for Project Reactor Reactive Types

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

```
static Scheduler
        parallel()
```

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#parallel

Key Scheduler Operators for Project Reactor Reactive Types

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

```
static Scheduler
       parallel()
```

Key Scheduler Operators for Project Reactor Reactive Types

- 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

DEFAULT_POOL_SIZE

```
public static final int DEFAULT_POOL_SIZE
```

Default pool size, initialized by system property `reactor.schedulers.defaultPoolSize` and falls back to the number of processors available to the runtime on init.

See Also:

`Runtime.availableProcessors()`

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#DEFAULT_POOL_SIZE

Key Scheduler Operators for Project Reactor Reactive Types

- 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

availableProcessors

```
public int availableProcessors()
```

Returns the number of processors available to the Java virtual machine.

This value may change during a particular invocation of the virtual machine. Applications that are sensitive to the number of available processors should therefore occasionally poll this property and adjust their resource usage appropriately.

Returns:

`the maximum number of processors available to the virtual machine; never smaller than one`

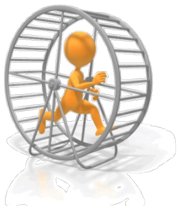
Since:

1.4

See docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html

Key Scheduler Operators for Project Reactor Reactive Types

- 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



Class Schedulers

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`

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html

Key Scheduler Operators for Project Reactor Reactive Types

- 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.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

Key Scheduler Operators for Project Reactor Reactive Types

- 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(sBigFrac))

       .subscribeOn
          (Schedulers.parallel()))

  .reduce(BigFraction::add)
```

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

See Reactive/flux/ex3/src/main/java/FluxEx.java

Key Scheduler Operators for Project Reactor Reactive Types

- 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(sBigFrac))

       .subscribeOn
         (Schedulers.parallel()))

  .reduce(BigFraction::add)
```

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

See Reactive/flux/ex3/src/main/java/FluxEx.java

Key Scheduler Operators for Project Reactor Reactive Types

- 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(sBigFrac))

       .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

Key Scheduler Operators for Project Reactor Reactive Types

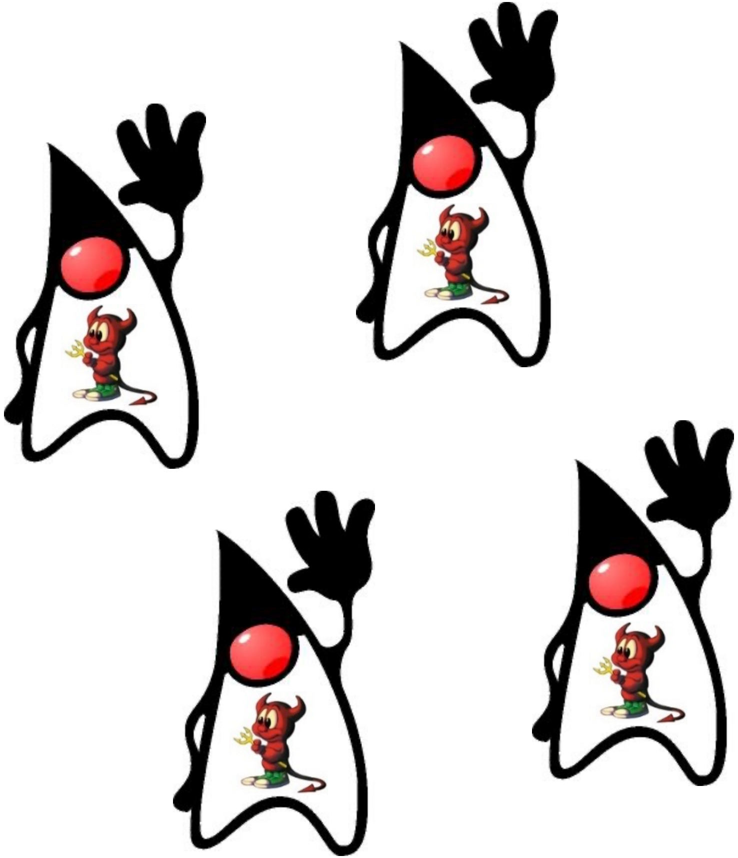
- The Schedulers.parallel() operator
  - Hosts a fixed pool of single-threaded ExecutorService-based workers
  - Used for event-loops, callbacks, & other computational work

```
Flux
  .fromIterable(bigFractions)

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

      .subscribeOn
         (Schedulers.parallel()))

  .reduce(BigFraction::add)
```

Only one thread runs reduce() after all other computations are done

Key Scheduler Operators for Project Reactor Reactive Types

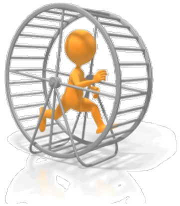
- The Schedulers.parallel() operator
  - Hosts a fixed pool of single-threaded ExecutorService-based workers
  - Used for event-loops, callbacks, & other computational work
  - Implemented via "daemon threads"
    - i.e., won't prevent the app from exiting even if its work isn't done



See www.baeldung.com/java-daemon-thread

Key Scheduler Operators for Project Reactor Reactive Types

- 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



computation

```
@NonNull
public static @NonNull Scheduler computation()
```

Returns a default, shared `Scheduler` instance intended for computational work.

This can be used for event-loops, processing callbacks and other computational work.

It is not recommended to perform blocking, IO-bound work on this scheduler. Use `io()` instead.

The default instance has a backing pool of single-threaded `ScheduledExecutorService` instances equal to the number of available processors (`Runtime.availableProcessors()`) to the Java VM.

Unhandled errors will be delivered to the scheduler Thread's `Thread.UncaughtExceptionHandler`.

This type of scheduler is less sensitive to leaking `Scheduler.Worker` instances, although not disposing a worker that has timed/delayed tasks not cancelled by other means may leak resources and/or execute those tasks "unexpectedly".

If the `RxJavaPlugins.setFailOnNonBlockingScheduler(boolean)` is set to true, attempting to execute operators that block while running on this scheduler will throw an `IllegalStateException`.

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/schedulers/Schedulers.html#computation

Key Scheduler Operators for Project Reactor Reactive Types

- 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

```
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 javase/8/docs/api/java/util/concurrent/ForkJoinPool.html#commonPool

Key Scheduler Operators for Project Reactor Reactive Types

- 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
    - The ManagedBlocker mechanism supports I/O-bound tasks

`public static interface` **`ForkJoinPool.ManagedBlocker`**

Interface for extending managed parallelism for tasks running in `ForkJoinPool`s.

A `ManagedBlocker` provides two methods. Method `isReleasable()` must return `true` if blocking is not necessary. Method `block()` blocks the current thread if necessary (perhaps internally invoking `isReleasable` before actually blocking). These actions are performed by any thread invoking `ForkJoinPool.managedBlock(ManagedBlocker)`. The unusual methods in this API accommodate synchronizers that may, but don't usually, block for long periods. Similarly, they allow more efficient internal handling of cases in which additional workers may be, but usually are not, needed to ensure sufficient parallelism. Toward this end, implementations of method `isReleasable` must be amenable to repeated invocation.

See javase/8/docs/api/java/util/concurrent/ForkJoinPool.ManagedBlocker.html

End of Key Scheduler Operators for Project Reactor Reactive Types (Part 2)