How Parallel Programs are Developed in Java (Part 3)

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 the parallelism frameworks supported by Java, e.g.
  - Fork-join pools
  - Parallel streams
  - Completable futures
- **Reactive streams**
  - An async programming paradigm concerned with processing data streams & propagating changes between publishers & subscribers

See [en.wikipedia.org/wiki/Reactive_Streams](en.wikipedia.org/wiki/Reactive_Streams)
Overview of Java Reactive Parallelism Frameworks
Overview of Java Reactive Parallelism Frameworks

- Reactive programming is based on four key principles

See [www.reactivemanifesto.org](http://www.reactivemanifesto.org)
Overview of Java Reactive Parallelism Frameworks

- Reactive programming is based on four key principles

1. **Responsive** – provide rapid & consistent response times

See [www.reactivemanifesto.org](http://www.reactivemanifesto.org)
Overview of Java Reactive Parallelism Frameworks

- Reactive programming is based on four key principles

1. **Responsive** – provide rapid & consistent response times

2. **Resilient** – the system remains responsive, even in the face of failures

See [www.reactivemanifesto.org](http://www.reactivemanifesto.org)
Overview of Java Reactive Parallelism Frameworks

• Reactive programming is based on four key principles

1. **Responsive** – provide rapid & consistent response times

2. **Resilient** – the system remains responsive, even in the face of failures

3. **Elastic** – a system should remain responsive, even under varying workload

See [www.reactivemanifesto.org](http://www.reactivemanifesto.org)
Overview of Java Reactive Parallelism Frameworks

- Reactive programming is based on four key principles

1. **Responsive** – provide rapid & consistent response times

2. **Resilient** – the system remains responsive, even in the face of failures

3. **Elastic** – a system should remain responsive, even under varying workload

4. **Message-driven** – asynchronous message-passing to ensure loose coupling, isolation, & location transparency between components

See [www.reactivemanifesto.org](http://www.reactivemanifesto.org)
Overview of Java Reactive Parallelism Frameworks

- Java supports reactive parallelism via the “Flow” API

See [www.reactive-streams.org](http://www.reactive-streams.org)

The Flow API was added in Java 9
Overview of Java Reactive Parallelism Frameworks

- Java supports reactive parallelism via the “Flow” API
- Implements a reactive streams pub/sub framework via two patterns
Overview of Java Reactive Parallelism Frameworks

- Java supports reactive parallelism via the “Flow” API
- Implements a reactive streams pub/sub framework via two patterns

- **Iterator**
  - Applies a “pull model” where app subscribe(s) pull items from a publisher source

See [en.wikipedia.org/wiki/Iterator_pattern](en.wikipedia.org/wiki/Iterator_pattern)
Overview of Java Reactive Parallelism Frameworks

- Java supports reactive parallelism via the “Flow” API
- Implements a reactive streams pub/sub framework via two patterns
  - **Iterator**
  - **Observer**
    - Applies a “push model” that reacts when a publisher source pushes items to subscriber sink(s)

Overview of Java Reactive Parallelism Frameworks

- RxJava & Project Reactor are popular Java reactive streams implementations

See www.baeldung.com/rx-java & projectreactor.io
Overview of Java Reactive Parallelism Frameworks

- RxJava & Project Reactor are popular Java reactive streams implementations
- The `subscribeOn()`, `publishOn()`, & `observeOn()` operators map events to threads & thread pools

See zoltanaltfatter.com/2018/08/26/subscribeOn-publishOn-in-Reactor
Overview of Java Reactive Parallelism Frameworks

- RxJava & Project Reactor are popular Java reactive streams implementations
  - The `subscribeOn()`, `publishOn()`, & `observeOn()` operators map events to threads & thread pools
  - Threads & thread pools are managed by Schedulers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedulers.computation()</td>
<td>Schedules computation bound work (ScheduledExecutorService with pool size = NCPU, LRU worker select strategy)</td>
</tr>
<tr>
<td>Schedulers.immediate()</td>
<td>Schedules work on current thread</td>
</tr>
<tr>
<td>Schedulers.io()</td>
<td>I/O bound work (ScheduledExecutorService with growing thread pool)</td>
</tr>
<tr>
<td>Schedulers.trampoline()</td>
<td>Queues work on the current thread</td>
</tr>
<tr>
<td>Schedulers.newThread()</td>
<td>Creates new thread for every unit of work</td>
</tr>
<tr>
<td>Schedulers.test()</td>
<td>Schedules work on scheduler supporting virtual time</td>
</tr>
<tr>
<td>Schedulers.from(Executor e)</td>
<td>Schedules work to be executed on provided executor</td>
</tr>
</tbody>
</table>

Overview of Java Reactive Parallelism Frameworks

- RxJava & Project Reactor are popular Java reactive streams implementations
  - The subscribeOn(), publishOn(), & observeOn() operators map events to threads & thread pools
  - Threads & thread pools are managed by Schedulers
- There are also specialized parallel processing classes

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/parallel/ParallelFlowable.html
Evaluating Pros & Cons of Reactive Streams Programming Frameworks
Evaluating Pros & Cons of Reactive Streams Programming Frameworks

• Pros of the reactive streams programming frameworks
Evaluating Pros & Cons of Reactive Streams Programming Frameworks

- Pros of the reactive streams programming frameworks
- Support parallelism with a minimal number of threads via a range of thread pools

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedulers.computation()</td>
<td>Schedules computation bound work (ScheduledExecutorService with pool size = NCPU, LRU worker select strategy)</td>
</tr>
<tr>
<td>Schedulers.immediate()</td>
<td>Schedules work on current thread</td>
</tr>
<tr>
<td>Schedulers.io()</td>
<td>I/O bound work (ScheduledExecutorService with growing thread pool)</td>
</tr>
<tr>
<td>Schedulers.trampoline()</td>
<td>Queues work on the current thread</td>
</tr>
<tr>
<td>Schedulers.newThread()</td>
<td>Creates new thread for every unit of work</td>
</tr>
<tr>
<td>Schedulers.test()</td>
<td>Schedules work on scheduler supporting virtual time</td>
</tr>
<tr>
<td>Schedulers.from(Executor e)</td>
<td>Schedules work to be executed on provided executor</td>
</tr>
</tbody>
</table>

See [www.baeldung.com/rxjava-schedulers](http://www.baeldung.com/rxjava-schedulers)
Evaluating Pros & Cons of Reactive Streams Programming Frameworks

- Pros of the reactive streams programming frameworks
  - Support parallelism with a minimal number of threads via a range of thread pools
  - Scale up performance with relatively few resources

See dzone.com/articles/spring-boot-20-webflux-reactive-performance-test
Evaluating Pros & Cons of Reactive Streams Programming Frameworks

- **Pros of the reactive streams programming frameworks**
  - Support parallelism with a minimal number of threads via a range of thread pools
  - Explicit synchronization and/or threading is rarely needed when applying these frameworks

Alleviates many accidental & inherent complexities of concurrency/parallelism
• Pros of the reactive streams programming frameworks
  • Support parallelism with a minimal number of threads via a range of thread pools
  • Explicit synchronization and/or threading is rarely needed when applying these frameworks
  • Integrates streams, asynchrony, & pub/sub paradigms cleanly
Evaluating Pros & Cons of Reactive Streams Programming Frameworks

- Cons of the reactive streams programming frameworks
Evaluating Pros & Cons of Reactive Streams Programming Frameworks

- Cons of the reactive streams programming frameworks
  - It isn’t appropriate in all situations

See www.youtube.com/watch?v=z0a0N9OgaAA
Evaluating Pros & Cons of Reactive Streams Programming Frameworks

• Cons of the reactive streams programming frameworks
  • It isn’t appropriate in all situations

Reactive programming can have a fairly steep learning curve to the uninitiated.

See [www.freecodecamp.org/news/a-complete-roadmap-for-learning-rxjava-9316ee6aeda7](http://www.freecodecamp.org/news/a-complete-roadmap-for-learning-rxjava-9316ee6aeda7)
End of How Parallel Programs Are Developed in Java (Part 3)