Understand Java Parallel Streams Internals: Splitting, Combining, & Pooling

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

- Understand parallel stream internals, e.g.
- Know what can change & what can’t
- Splitting, combining, & pooling mechanisms

```java
public interface ManagedBlocker {
    boolean block() throws InterruptedException;
    boolean isReleasable();
}

public interface Spliterator<T> {
    boolean tryAdvance (Consumer<? Super T> action);
    Spliterator<T> trySplit();
    long estimateSize();
    int characteristics();
}

final class Collectors {
    ...
    public static <T> Collector<T, ?, List<T>> toList() {
        ...
    }
    public static <T> Collector<T, ?, Set<T>> toSet() {
        ...
    }
    ...
}
```
Parallel Stream Splitting, Combining, & Pooling Mechanisms
A parallel stream's splitting, combining, & pooling mechanisms are often invisible.

Parallel Stream Splitting, Combining, & Pooling Mechanisms

- Input x
- Intermediate operation (behavior f) → Output f(x)
- Intermediate operation (behavior g) → Output g(f(x))
- Terminal operation (behavior h)

Stream factory operation ()
Parallel Stream Splitting, Combining, & Pooling Mechanisms

- A parallel stream’s splitting, combining, & pooling mechanisms are often invisible
- All Java collections have predefined spliterators that create parallel streams

```java
interface Collection<E> {
  ...
  default Spliterator<E> spliterator() { return Spliterators.splitter(this, 0); }

  default Stream<E> parallelStream() { return StreamSupport.stream(spliterator(), true); }...
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/Collection.html](https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html)
Parallel Stream Splitting, Combining, & Pooling Mechanisms

- A parallel stream’s splitting, combining, & pooling mechanisms are often invisible
- All Java collections have predefined spliterators that create parallel streams

```java
interface Collection<E> {
    ...
    default Spliterator<E> spliterator() {
        return Spliterators.spliterator(this, 0);
    }

    default Stream<E> parallelStream() {
        return StreamSupport.stream(spliterator(), true);
    }
    ...
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/Spliterator.html](docs.oracle.com/javase/8/docs/api/java/util/Spliterator.html)
Parallel Stream Splitting, Combining, & Pooling Mechanisms

- A parallel stream’s splitting, combining, & pooling mechanisms are often invisible
- All Java collections have predefined spliterators that create parallel streams

```java
interface Collection<E> {
    ...  
    default Spliterator<E> spliterator() {
        return Spliterators.spliterator(this, 0);
    }

    default Stream<E> parallelStream() {
        return StreamSupport.stream(spliterator(), true);
    }
    ...  
}
```

See [blog.logentries.com/2015/10/java-8-introduction-to-parallelism-and-spliterator](blog.logentries.com/2015/10/java-8-introduction-to-parallelism-and-spliterator)
A parallel stream’s splitting, combining, & pooling mechanisms are often invisible

All Java collections have predefined spliterators that create parallel streams

Java also predefines collector factory methods in the Collectors utility class

```java
final class Collectors {
    ...
    public static <T> Collector<T, ?, List<T>>
        toList() { ... }
    public static <T> Collector<T, ?, Set<T>>
        toSet() { ... }
    ...
}
```

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html](docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html)
A parallel stream’s splitting, combining, & pooling mechanisms are often invisible

- All Java collections have predefined spliterators that create parallel streams
- Java also predefines collector factory methods in the Collectors utility class

```java
final class Collectors {
    ...
    public static <T> Collector<T, ?, List<T>>
        toList() { ... }

    public static <T> Collector<T, ?, Set<T>>
        toSet() { ... }
    ...
}
```

These non-concurrent collectors can work seamlessly with parallel streams
A parallel stream’s splitting, combining, & pooling mechanisms are often invisible

All Java collections have predefined spliterators that create parallel streams

Java also predefined collector factory methods in the Collectors utility class

The common fork-join pool is used to run intermediate operations on chunks of data

See www.baeldung.com/java-fork-join
However, parallel streams programmers can also customize these mechanisms.

See upcoming lessons on "Java Parallel Stream Internals"
However, parallel streams programmers can also customize these mechanisms.

interface Spliterator<T> {
  boolean tryAdvance
    (Consumer<? Super T> action);
  Spliterator<T> trySplit();
  void forEachRemaining
    (Consumer<? Super T> action);
  long estimateSize();
  int characteristics();
}

An interface used to traverse & partition elements of a source.

See docs.oracle.com/javase/8/docs/api/java/util/Spliterator.html
Parallel Stream Splitting, Combining, & Pooling Mechanisms

- However, parallel streams programmers can also customize these mechanisms

```java
interface Spliterator<T> {
    boolean tryAdvance (Consumer<? Super T> action);
    Spliterator<T> trySplit();
    void forEachRemaining (Consumer<? Super T> action);
    long estimateSize();
    int characteristics();
}
```

The streams framework uses this method to process elements in sequential and parallel streams.

See earlier lesson on “Java Streams: Applying Spliterators"
Parallel Stream Splitting, Combining, & Pooling Mechanisms

- However, parallel streams programmers can also customize these mechanisms

```java
interface Spliterator<T> {
    boolean tryAdvance(Consumer<? Super T> action);
    Spliterator<T> trySplit();
    void forEachRemaining(Consumer<? Super T> action);
    long estimateSize();
    int characteristics();
}
```

The streams framework uses this method to partition elements in a parallel stream.

See upcoming lesson on “Java Parallel Streams Internals: Partitioning”
However, parallel streams programmers can also customize these mechanisms.

```java
interface Collector<T, A, R> {
    Supplier<A> supplier();
    BiConsumer<A, T> accumulator();
    BinaryOperator<A> combiner();
    Function<A, R> finisher();
    Set<Collector.Characteristics> characteristics();
    ...
}
```

A framework that accumulates input elements into a concurrent and/or non-concurrent mutable result containers.

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html](docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html)
However, parallel streams programmers can also customize these mechanisms.

```java
public interface ManagedBlocker {
    boolean block() throws InterruptedException;
    boolean isReleasable();
}
```

This interface provides managed parallelism for tasks running in the common fork-join pool.

End of Understand Java Parallel Stream Internals: Splitting, Combining, & Pooling