Java Parallel Streams Internals: Non-Concurrent & Concurrent Collectors (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

- Understand parallel stream internals, e.g.
  - Know what can change & what can’t
  - Partition a data source into “chunks”
  - Process chunks in parallel via the common fork-join pool
  - Configure the Java parallel stream common fork-join pool
  - Perform a reduction to combine partial results into a single result
  - Recognize key behaviors & differences of non-concurrent & concurrent collectors
- Be aware of non-concurrent & concurrent collector APIs
Non-Concurrent & Concurrent Collector APIs
The Collector interface defines three generic types:

- supplier(): Supplier<A>
- accumulator(): BiConsumer<A, T>
- combiner(): BinaryOperator<A>
- finisher(): Function<A, R>
- characteristics(): Set<Characteristics>

See [www.baeldung.com/java-8-collectors](http://www.baeldung.com/java-8-collectors)
The Collector interface defines three generic types:

- **T** – The type of objects available in the stream
  - e.g., Integer, String, Double, SearchResults, etc.

```java
<<Java Interface>>

Collector<T, A, R>

- supplier(): Supplier<A>
- accumulator(): BiConsumer<A, T>
- combiner(): BinaryOperator<A>
- finisher(): Function<A, R>
- characteristics(): Set<Characteristics>
```
The Collector interface defines three generic types:

- **T**
- **A** – The type of a mutable result container for accumulation
- e.g., List of T, Set of T, ConcurrentHashMap.KeySetView, etc.

<<Java Interface>>

```
Collector<T, A, R>

supplier(): Supplier<A>
accumulator(): BiConsumer<A, T>
combiner(): BinaryOperator<A>
finisher(): Function<A, R>
characteristics(): Set<Characteristics>
```
Non- Concurren t & Concurrent Collector APIs

• The Collector interface defines three generic types
  • T
  • A – The type of a mutable result container for accumulation
    • e.g., List of T, Set of T, ConcurrentHashMap, KeySetView, etc.
    • Lists can be implemented by ArrayList, LinkedList, etc.

See docs.oracle.com/javase/tutorial/collections/implementations/list.html
Non-Concurrent & Concurrent Collector APIs

- The Collector interface defines three generic types
  - `T`
  - `A`
  - `R` – The type of a final result
    - e.g., List of T, CompletableFuture to List of T, ConcurrentHashMap, KeyViewSet, etc.

See [www.baeldung.com/java-8-collectors](http://www.baeldung.com/java-8-collectors)
Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface

```java
<<Java Interface>>

Collector<T,A,R>

- supplier(): Supplier<A>
- accumulator(): BiConsumer<A,T>
- combiner(): BinaryOperator<A>
- finisher(): Function<A,R>
- characteristics(): Set<Characteristics>
```
Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics() – provides a stream with additional information used for internal optimizations, e.g.
    - UNORDERED
      - The collector need not preserve the encounter order

A concurrent collector *should* be UNORDERED, but a non-concurrent collector *can* be ORDERED.
Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - `characteristics()` – provides a stream with additional information used for internal optimizations, e.g.
    - UNORDERED
    - IDENTITY_FINISH
      - The finisher() is the identity function so it can be a no-op
        - e.g. finisher() just returns null

<<Java Interface>>

```
Collector<T,A,R>
```

- `supplier()`: Supplier<A>
- `accumulator()`: BiConsumer<A,T>
- `combiner()`: BinaryOperator<A>
- `finisher()`: Function<A,R>
- `characteristics()`: Set<Characteristics>
Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - `characteristics()` – provides a stream with additional information used for internal optimizations, e.g.
    - UNORDERED
    - IDENTITY_FINISH
    - CONCURRENT
      - `accumulator()` is called concurrently on result container

The mutable result container must be synchronized!!

A concurrent collector *should* be CONCURRENT, but a non-concurrent collector should *not* be!
Non-Concurrent & Concurrent Collector APIs

• Five methods are defined in the Collector interface
  • **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.
    • UNORDERED
    • IDENTITY_FINISH
  • **CONCURRENT**
    • accumulator() is called concurrently on result container
    • The combiner() method is a no-op

<<Java Interface>>

```
Collector<T,A,R>
```

- supplier(): Supplier<A>
- accumulator(): BiConsumer<A,T>
- **combiner()**: BinaryOperator<A>
- finisher(): Function<A,R>
- characteristics(): Set<Characteristics>
Five methods are defined in the Collector interface:

- `characteristics()` – provides a stream with additional information used for internal optimizations, e.g.
  - UNORDERED
  - IDENTITY_FINISH

- CONCURRENT
  - `accumulator()` is called concurrently on result container
  - The `combiner()` method is a no-op
  - A non-concurrent collector can be used with either sequential or parallel streams

Internally, the streams framework decides how to ensure correct behavior.
Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
- **characteristics()** – provides a stream with additional information used for internal optimizations, e.g.

```java
Set<Characteristics> characteristics() {
    return Collections.unmodifiableSet(
        EnumSet.of(
            Collector.Characteristics.CONCURRENT,
            Collector.Characteristics.UNORDERED));
}
```

Any/all characteristics can be set using `EnumSet.of()`

See [docs.oracle.com/javase/8/docs/api/java/util/EnumSet.html](http://docs.oracle.com/javase/8/docs/api/java/util/EnumSet.html)
Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - `characteristics()`
  - `supplier()` – returns a supplier that acts as a factory to generate an empty result container

### Collector<T,A,R> <<Java Interface>>

- `supplier()`: Supplier<A>
- `accumulator()`: BiConsumer<A,T>
- `combiner()`: BinaryOperator<A>
- `finisher()`: Function<A,R>
- `characteristics()`: Set<Characteristics>
Non- Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - supplier() – returns a supplier that acts as a factory to generate an empty result container, e.g.
    - return ArrayList::new

A non-concurrent collector provides a result container for each thread in a parallel stream
Five methods are defined in the Collector interface:

- **characteristics()**
- **supplier()** – returns a supplier that acts as a factory to generate an empty result container, e.g.
  - `return ArrayList::new`
  - `return ConcurrentHashMap::newKeySet`

A concurrent collector has one result container shared by all threads in a parallel stream.
Five methods are defined in the Collector interface:

- characteristics()
- supplier()
- **accumulator()** – returns a bi-consumer that adds a new element to an existing result container.

```java
<<Java Interface>>

Collector<T,A,R>

- supplier(): Supplier<A>
- **accumulator(): BiConsumer<A,T>**
- combiner(): BinaryOperator<A>
- finisher(): Function<A,R>
- characteristics(): Set<Characteristics>
```
Five methods are defined in the Collector interface:

- `characteristics()`
- `supplier()`
- `accumulator()` – returns a bi-consumer that adds a new element to an existing result container, e.g.
  - `return List::add`

A non-concurrent collector needs no synchronization.

See [docs.oracle.com/javase/8/docs/api/java/util/List.html#add](https://docs.oracle.com/javase/8/docs/api/java/util/List.html#add)
Five methods are defined in the Collector interface:

- `characteristics()`
- `supplier()`
- `accumulator()` – returns a bi-consumer that adds a new element to an existing result container, e.g.
  - `return List::add`
  - `return ConcurrentHashMap.KeySetView::add`

A concurrent collector’s result container must be synchronized.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.KeySetView.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.KeySetView.html)
Five methods are defined in the Collector interface:

- `characteristics()`
- `supplier()`
- `accumulator()`
- `combiner()` – returns a binary operator that merges two result containers together

```
<<Java Interface>>

Collector<T,A,R>

supplier(): Supplier<A>
accumulator(): BiConsumer<A,T>
combiner(): BinaryOperator<A>
finisher(): Function<A,R>
characteristics(): Set<Characteristics>
```
Non-.Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - supplier()
  - accumulator()
  - combiner() – returns a binary operator that merges two result containers together, e.g.
    - return (one, another) -> {
      one.addAll(another); return one;
    }

A combiner() is only used for a non-concurrent collector
Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - `characteristics()`
  - `supplier()`
  - `accumulator()`
  - `combiner()` – returns a binary operator that merges two result containers together, e.g.
    
    ```java
    return (one, another) -> {
      one.addAll(another); return one;
    }
    ```

- The `combiner()` method is not called when CONCURRENT is set
Five methods are defined in the Collector interface:

- `characteristics()`
- `supplier()`
- `accumulator()`
- `combiner()`
- `finisher()` — returns a function that converts the result container to final result type.

<<Java Interface>>

```java
Collector<T,A,R>
```

- `supplier()`: Supplier<A>
- `accumulator()`: BiConsumer<A,T>
- `combiner()`: BinaryOperator<A>
- `finisher()`: Function<A,R>
- `characteristics()`: Set<Characteristics>
Five methods are defined in the Collector interface:

- characteristics()
- supplier()
- accumulator()
- combiner()
- finisher() – returns a function that converts the result container to final result type, e.g.
  - `Function.identity()`
Non-Concurrent & Concurrent Collector APIs

- Five methods are defined in the Collector interface
  - characteristics()
  - supplier()
  - accumulator()
  - combiner()
- finisher() – returns a function that converts the result container to final result type, e.g.
  - Function.identity()
  - return null

Should be a no-op if IDENTITY_FINISH characteristic is set
Five methods are defined in the Collector interface:
- `characteristics()`
- `supplier()`
- `accumulator()`
- `combiner()`
- `finisher()` – returns a function that converts the result container to final result type, e.g.
  - `Function.identity()`
  - `return null`

`finisher()` can also be more interesting!
End of Java Parallel Streams Internals: Non-Concurrent & Concurrent Collectors (Part 2)