Java Parallel Streams Internals: Implementing a Concurrent Map Collector

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
- Grok performance variance in concurrent & non-concurrent collectors
- Learn how to implement a concurrent Map collector
Rationale for a Concurrent Map Collector
Rationale for a Concurrent Map Collector

- The Java Collectors utility class provides factory methods that make non-concurrent collectors

<table>
<thead>
<tr>
<th>Collectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collectors()</td>
</tr>
<tr>
<td>toCollection(Supplier&lt;C&gt;):Collector&lt;T,?,C&gt;</td>
</tr>
<tr>
<td>toList():Collector&lt;T,?,List&lt;T&gt;&gt;</td>
</tr>
<tr>
<td>toSet():Collector&lt;T,?,Set&lt;T&gt;&gt;</td>
</tr>
</tbody>
</table>

See [www.baeldung.com/java-8-collectors](http://www.baeldung.com/java-8-collectors)
Rationale for a Concurrent Map Collector

- The Java Collectors utility class provides factory methods that make non-concurrent collectors
- It also contains some factory methods that make collectors based on ConcurrentMap
- e.g., ConcurrentHashMap & ConcurrentSkipListMap

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentMap.html](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentMap.html)
The Java Collectors utility class provides factory methods that make non-concurrent collectors. It also contains some factory methods that make collectors based on ConcurrentMap. However, there are no pre-defined concurrent collectors provided by Java that return *sorted* maps. e.g., TreeMap

See [docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html](http://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html)
Rationale for a Concurrent Map Collector

- The ConcurrentMapCollector is designed to overcome this omission with the Java class library

See Java8/ex37/src/main/java/utils/ConcurrentMapCollector.java
Rationale for a Concurrent Map Collector

- The ConcurrentMapCollector is designed to overcome this omission with the Java class library.

The Supplier param can be used to customize the type of Map returned from this collector.

See docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html
Implementing a Generic Concurrent Map Collector
Implementing a Generic Concurrent Map Collector

- ConcurrentMapCollector defines four generic types
Implementing a Generic Concurrent Map Collector

- ConcurrentMapCollector defines four generic types
  - T – The type of objects available from the Stream
    - e.g., String, GCDParam, SimpleImmutableEntry, etc.

See docs.oracle.com/javase/8/docs/api/java/util/AbstractMap.SimpleImmutableEntry.html
Implementing a Generic Concurrent Map Collector

- ConcurrentMapCollector defines four generic types
  - **T**
  - **K** – The type of the key used in the map
    - e.g., Double, String, etc.

```
ConcurrentMapCollector<T, K, V, M> =

finisher()
accumulator()
characteristics()
toMap(Function<T, K>, Function<T, V>, BinaryOperator<V>, Supplier<M>)
supplier()
combiner()

Function<Map<K, V>, M>
BiConsumer<Map<K, V>, T>
Set<Characteristics>
Collector<T, ?, M>
Supplier<Map<K, V>>
BinaryOperator<Map<K, V>>
```
Implementing a Generic Concurrent Map Collector

- ConcurrentMapCollector defines four generic types
  - T
  - A
  - V – The type of the value used in the map
    - e.g., SearchResults, Integer, GCDResult, etc.

<table>
<thead>
<tr>
<th>ConcurrentMapCollector&lt;T, K, V, M&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>finisher()</td>
</tr>
<tr>
<td>accumulator()</td>
</tr>
<tr>
<td>characteristics()</td>
</tr>
<tr>
<td>toMap(Function&lt;T, K&gt;, Function&lt;T, V&gt;, BinaryOperator&lt;V&gt;, Supplier&lt;M&gt;) Collector&lt;T, ?, M&gt;</td>
</tr>
<tr>
<td>supplier()</td>
</tr>
<tr>
<td>combiner()</td>
</tr>
</tbody>
</table>

Function <Map<K, V>, M>
BiConsumer <Map<K, V>, T>
Set<Characteristics>
Collector<T, ?, M>
Supplier <Map<K, V>>
BinaryOperator <Map<K, V>>
Implementing a Generic Concurrent Map Collector

- ConcurrentMapCollector defines four generic types
  - T
  - A
  - V
  - M – The type of Map returned from the collector
    - e.g., ConcurrentHashMap, TreeMap, LinkedHashMap, etc.

ConcurrentMapCollector uses ConcurrentHashMap internally
Implementing a Generic Concurrent Map Collector

- The `toMap()` factory method creates a new instance of `ConcurrentMapCollector` that is parameterized by Java functional interface objects.
• The toMap() factory method creates a new instance of ConcurrentMapCollector that is parameterized by Java functional interface objects
  • e.g., return new ConcurrentMapCollector<>
    (keyMapper,
     valueMapper,
     mergeFunction,
     mapSupplier);
Implementing Concurrent Map Collector Methods
Implementing Concurrent Map Collector Methods

- Five key methods are defined in the ConcurrentMapCollector

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>finisher()</td>
<td>Function &lt;Map&lt;K, V&gt;, M&gt;</td>
</tr>
<tr>
<td>accumulator()</td>
<td>BiConsumer &lt;Map&lt;K, V&gt;, T&gt;</td>
</tr>
<tr>
<td>characteristics()</td>
<td>Set&lt;Characteristics&gt;</td>
</tr>
<tr>
<td>toMap(Function&lt;T, K&gt;, Function&lt;T, V&gt;, BinaryOperator&lt;V&gt;, Supplier&lt;M&gt;)</td>
<td>Collector &lt;T, ?, M&gt;</td>
</tr>
<tr>
<td>supplier()</td>
<td>Supplier &lt;Map&lt;K, V&gt;&gt;</td>
</tr>
<tr>
<td>combiner()</td>
<td>BinaryOperator &lt;Map&lt;K, V&gt;&gt;</td>
</tr>
</tbody>
</table>
Implementing Concurrent Map Collector Methods

• Five key methods are defined in the ConcurrentMapCollector
  • characteristics() – provides additional info to optimize the collector, e.g.
    • UNORDERED
      • The collector need not preserve encounter order
Implementing Concurrent Map Collector Methods

- Five key methods are defined in the ConcurrentMapCollector
  - `characteristics()` – provides additional info to optimize the collector, e.g.
    - UNORDERED
    - CONCURRENT
  - `accumulator()` is called concurrently on the ConcurrentHashMap mutable result container

*ConcurrentHashMap methods are all synchronized!!*

See [www.geeksforgeeks.org/concurrenthashmap-in-java](http://www.geeksforgeeks.org/concurrenthashmap-in-java)
Five key methods are defined in the ConcurrentMapCollector

- **characteristics()** – provides additional info to optimize the collector, 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)
Five key methods are defined in the ConcurrentMapCollector:

- characteristics() – returns a Supplier that acts as a factory method to generate an empty result container
- supplier() – returns a Supplier that acts as a factory method to generate an empty result container
Implementing Concurrent Map Collector Methods

- Five key methods are defined in the ConcurrentMapCollector
  - `characteristics()`
  - **supplier()** – returns a Supplier that acts as a factory method to generate an empty result container, e.g.
  - `return ConcurrentHashMap::new`
Five key methods are defined in the ConcurrentMapCollector:

- `characteristics()`
- `supplier()`
- `accumulator()` – returns a `BiConsumer` that adds a new element to the existing `ConcurrentHashMap`

```
ConcurrentMapCollector<T, K, V, M> (Function<T, K>, Function<T, V>, BinaryOperator<V>, Supplier<M>)

finisher()
accumulator()
characteristics()
toMap(Function<T, K>, Function<T, V>, BinaryOperator<V>, Supplier<M>)
supplier()
combiner()
```

Implementing Concurrent Map Collector Methods
Implementing Concurrent Map Collector Methods

- Five key methods are defined in the ConcurrentMapCollector:
  - characteristics()
  - supplier()
  - **accumulator()** – returns a BiConsumer that adds a new element to the existing ConcurrentHashMap, e.g.
    - `return (Map<K, V> map, T element) -> map`
      .merge(mKeyMapper.apply(element),
      mValueMapper.apply(element),
      mMergeFunction);

ConcurrentHashMap's `merge()` method is efficiently synchronized.

See codepumpkin.com/hashtable-vs-synchronizedmap-vs-concurrenthashmap
Implementing Concurrent Map Collector Methods

• Five key methods are defined in the ConcurrentMapCollector
  • characteristics()
  • supplier()
  • accumulator()
  • combiner() – returns a Binary Operator that merges two result containers together
Implementing Concurrent Map Collector Methods

- Five key methods are defined in the ConcurrentMapCollector
  - `characteristics()`
  - `supplier()`
  - `accumulator()`
  - `combiner()` – returns a Binary Operator that merges two result containers together, e.g.
    - `return (one, another) -> {
        one.putAll(another); return one;
    }

This method is only called for non-concurrent collectors.
Implementing Concurrent Map Collector Methods

- Five key methods are defined in the ConcurrentMapCollector:
  - `characteristics()`
  - `supplier()`
  - `accumulator()`
  - `combiner()`
  - `finisher()` – returns a Function that converts ConcurrentHashMap to the final Map result type
Implementing Concurrent Map Collector Methods

- Five key methods are defined in the ConcurrentMapCollector
  - `characteristics()`
  - `supplier()`
  - `accumulator()`
  - `combiner()`
  - `finisher()` – returns a Function that converts ConcurrentHashMap to the final Map result type, e.g.

```java
return map -> {
    M newMap =
    mMapSupplier.get();

    if (newMap instanceof ConcurrentHashMap)
        return (M) map;
    else {
        newMap.putAll(map);
        return newMap;
    }
};
```

Only copies data if `M` isn’t a ConcurrentHashMap
Implementing Concurrent Map Collector Methods

```java
public class ConcurrentMapCollector<T, K, V, M extends Map<K, V>> implements Collector<T, Map<K, V>, M> {
    private final Function<? super T, ? extends K> mKeyMapper;
    private final Function<? super T, ? extends V> mValueMapper;
    private final Supplier<M> mMapSupplier;
}
```

See Java8/ex37/src/main/java/utils/ConcurrentMapCollector.java
End of Java Parallel Streams Internals: Implementing a Concurrent Map Collector