Learn How Pre-defined Non-Concurrent Collectors are Implemented

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 the structure & functionality of non-concurrent collectors for sequential streams
- Know the API for non-concurrent collectors
- Recognize how pre-defined non-concurrent collectors are implemented in the JDK

Class Collectors

```java
java.lang.Object
  java.util.stream.Collectors
```

```java
public final class Collectors
  extends Object

Implementations of Collector that implement various useful reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria, etc.

The following are examples of using the predefined collectors to perform common mutable reduction tasks:
How Pre-defined Non-Concurrent Collectors are Implemented
How Pre-defined Non-Concurrent Collectors are Implemented

- Collectors is a utility class whose factory methods create collectors for common collection types

Class Collectors

```java
java.lang.Object
    java.util.stream.Collectors
```

```java
public final class Collectors
extends Object
```

Implementations of `Collector` that implement various useful reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria, etc.

The following are examples of using the predefined collectors to perform common mutable reduction tasks:

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)
How Pre-defined Non-Concurrent Collectors are Implemented

- Collectors is a utility class whose factory methods create collectors for common collection types
- A utility class is final, has only static methods, no (non-static) state, & a private constructor

How Pre-defined Non-Concurrent Collectors are Implemented

- CollectorImpl defines a simple implementation class for a Collector

See openjdk/8-b132/java/util/stream/Collectors.java#Collectors.CollectorImpl
How Pre-defined Non- Concurrent Collectors are Implemented

• CollectorImpl defines a simple implementation class for a Collector

• However, this class is private to Collectors & is only used internally

```
如何预定义非并发收集器实现

• CollectorImpl定义了一个简单的收集器实现类

• 但是，这个类是私有的，仅用于Collectors内部使用
```
How Pre-defined Non-Concurrent Collectors are Implemented

- Collectors.toList() uses Collector Impl to return a non-concurrent collector that accumulates input elements into a new (Array)List

```java
final class Collectors {
    ...
    public static <T> Collector
            <T, ?, List<T>>
            toList() {
        return new CollectorImpl<>((Supplier<List<T>>)
                ArrayList::new,
                List::add,
                (left, right) -> {
            left.addAll(right);
            return left;
        },
        CH_ID);
    }
}
```

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#toList
How Pre-defined Non-Concurrent Collectors are Implemented

- Collectors.toList() uses CollectorImpl to return a non-concurrent collector that accumulates input elements into a new (Array)List

```java
final class Collectors {
    ...
    public static <T> Collector<T, ?, List<T>> toList() {
        return new CollectorImpl<>(
            ((Supplier<List<T>>) ArrayList::new, List::add,
                (left, right) -> {
                    left.addAll(right);
                    return left;
                },
            CH_ID);
        }
    }
    ...
```

The supplier constructor reference
How Pre-defined Non-Concurrent Collectors are Implemented

- Collectors.toList() uses CollectorImpl to return a non-concurrent collector that accumulates input elements into a new (Array)List

```java
final class Collectors {
    ...
    public static <T> Collector
        <T, ?, List<T>>
    toList() {
        return new CollectorImpl<>
            ((Supplier<List<T>>)
            ArrayList::new,
            List::add,
            (left, right) -> {
                left.addAll(right);
                return left;
            },
            CH_ID);
    } ...
```
How Pre-defined Non-Concurrent Collectors are Implemented

- Collectors.toList() uses CollectorImpl to return a non-concurrent collector that accumulates input elements into a new (Array)List

```java
final class Collectors {
    ...
    public static <T> Collector
            <T, ?, List<T>>
        toList() {
            return new CollectorImpl<>((Supplier<List<T>>) ArrayList::new,
                                     List::add,
                                     (left, right) -> {
                                         left.addAll(right);
                                         return left;
                                     },
                                     CH_ID);
        }
    ...
}
```

*The combiner lambda expression*

This combiner is only used for parallel streams
How Pre-defined Non-Concurrent Collectors are Implemented

- Collectors.toList() uses CollectorImpl to return a non-concurrent collector that accumulates input elements into a new (Array)List

  ```java
  final class Collectors {
      ...
      public static <T> Collector
          <T, ?, List<T>>
          toList() {
              return new CollectorImpl<>((Supplier<List<T>>)()
                  ArrayList::new,
                  List::add,
                  (left, right) -> {
                      left.addAll(right);
                      return left;
                  },
                  CH_ID);
          }
      ...
  }
  ```

  Characteristics set

CH_ID is defined as Collector.Characteristics.IDENTITY_FINISH
How Pre-defined Non-Concurrent Collectors are Implemented

• Collector.of() defines a simple public factory method that implements a Collector

```java
interface Collector<T, A, R> {
    ... public factory method that implements a Collector
    static<T, R> Collector<T, R, R> of
    (Supplier<R> supplier,
     BiConsumer<R, T> accumulator,
     BinaryOperator<R> combiner,
     Characteristics... chars) {
        ...
        return new Collectors
        .CollectorImpl<>
        (supplier,
         accumulator,
         combiner,
         chars);
    }
}
```

*This of() method is passed four params (last param is optional)*

See docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html#of
How Pre-defined Non-Concurrent Collectors are Implemented

- Collector.of() defines a simple public factory method that implements a Collector

```java
interface Collector<T, A, R> {
    ...
    static<T, R> Collector<T, R, R> of(
        Supplier<R> supplier,
        BiConsumer<R, T> accumulator,
        BinaryOperator<R> combiner,
        Function<A, R> finisher,
        Characteristics... chars)
    {
        ...
        return new Collectors
            .CollectorImpl<>(
                supplier,
                accumulator,
                combiner,
                finisher,
                chars);
    }
}
```

This of() method is passed five params (last param is optional)

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html#of](docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html#of)
How Pre-defined Non-.Concurrent Collectors are Implemented

- Collector.of() defines a simple public factory method that implements a Collector
- Both of() versions internally use the private CollectorImpl class

```java
interface Collector<T, A, R> {
    static<T, R> Collector<T, R, R> of(
        Supplier<R> supplier,
        BiConsumer<R, T> accumulator,
        BinaryOperator<R> combiner,
        Function<A, R> finisher,
        Characteristics... chars) {
            ...
            return new Collectors.Collectors.CollectorImpl<>(
                supplier,
                accumulator,
                combiner,
                finisher,
                chars); ...
        }
```

See [openjdk/8-b132/java/util/stream/Collectors.java#Collectors.CollectorImpl](openjdk/8-b132/java/util/stream/Collectors.java#Collectors.CollectorImpl)
End of Learn How Pre-defined Non-Concurrent Collectors are Implemented