Java Stream Internals: Execution

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 stream internals, e.g.
  • Know what can change & what can’t
  • Recognize how a Java stream is constructed
  • Be aware of how a Java stream is executed
    • e.g., how stateless & stateful intermediate operations & run-to-completion & short-circuiting terminal operations are run
Java Stream Execution
Java Stream Execution

- When terminal operation runs the streams framework picks an execution plan

See developer.ibm.com/technologies/java/articles/j-java-streams-3-brian-goetz/#executing-a-stream-pipeline
Java Stream Execution

- When terminal operation runs the streams framework picks an execution plan
- The plan is based on properties of the source & aggregate operations

Input x

**Stream map(Function<...> mapper)**

Output f(x)

**Stream filter(Predicate<...> pred)**

Output g(f(x))

**Stream sorted()**

Output h(g(f(x)))

**R collect(Collectors<...> collector)**
Java Stream Execution

- When terminal operation runs the streams framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories

```
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collectors<...> collector)
```
Java Stream Execution

• When terminal operation runs the streams framework picks an execution plan
  • The plan is based on properties of the source & aggregate operations
  • Intermediate operations are divided into two categories:
    • Stateless
      • e.g., filter(), map(), flatMap(), etc.

A pipeline with only stateless operations runs in one pass (even if it’s parallel)
When terminal operation runs the streams framework picks an execution plan:

- The plan is based on properties of the source & aggregate operations.

Intermediate operations are divided into two categories:

- Stateless
  - e.g., sorted(), limit(), distinct(), dropWhile(), etc.
- Stateful

A pipeline with stateful operations is divided into sections & runs in multiple passes.
When terminal operation runs the streams framework picks an execution plan. The plan is based on properties of the source & aggregate operations. Intermediate operations are divided into two categories. Terminal operations are also divided into two categories.

Java Stream Execution

Input $x$

- **Stream map** ($\text{Function} \rightarrow \text{mapper}$)
  - Output $f(x)$
  - **Stream filter** ($\text{Predicate} \rightarrow \text{pred}$)
    - Output $g(f(x))$
    - **Stream sorted**()
      - Output $h(g(f(x)))$
        - $R \ \text{collect}(\text{Collector} \rightarrow \text{collector})$
Java Stream Execution

- When terminal operation runs the streams framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories
  - Terminal operations are also divided into two categories
    - Run-to-completion
      - e.g., reduce(), collect(), forEach(), etc.

These terminal operation process data in bulk using Spliterator.forEachRemaining().
When terminal operation runs, the streams framework picks an execution plan.

The plan is based on properties of the source & aggregate operations.

Intermediate operations are divided into two categories.

Terminal operations are also divided into two categories:
- Run-to-completion
- Short-circuiting
  - e.g., anyMatch(), findFirst(), etc.

These terminal operation process data one element at a time using tryAdvance().
End of Java Stream Internals: Execution