目录

·背景

·创建Stream

·Stream api分类

·Stream api使用

**背景**

Stream于Java8推出，Java 8 中的 Stream 是对集合对象功能的增强，它专注于对集合对象进行各种非常便利、高效的聚合操作，或者大批量数据操作 。

比如要找出 dept\_id 为 1 的所有学生，然后返回以学号升序排序好的学生name集合，使用传统的方式需要多次遍历，并且代码看起来也比较臃肿，使用Stream可以高效地解决此类问题。

**创建Stream**

1.java8 collection接口下新增的 stream() 和 parallelStream() 方法：

List<Student> list = new ArrayList<>();

Stream<Student> s1 = list.stream();

Stream<Student> ps1 = list.parallelStream();

2. Arrays的stream方法：

Student[] stuArr = new Student[] { };

Stream<Student> s2 = Arrays.stream(stuArr);

3.Stream(以及IntStream,DoubleStream,LongStream)的静态方法：

Stream<Student> s3 = Stream.of(stuArr);

Stream<Student> s4 = Stream.iterate(stu1, s -> {

s.setAge(s.getAge() + 1);

return s;

}).limit(3);

Stream<Integer> s5 = Stream.generate(() -> (new Random()).nextInt(100))

.limit(3);

4.使用 BufferedReader.lines() 方法，将每行内容转成流：

Stream<String> s6 = new BufferedReader(new FileReader("filePath")).lines();

5.其它：

Pattern.splitAsStream()

Random.ints()

JarFile.stream()

java.nio.file.Files.walk()

BitSet.stream()

**Stream api分类**

1.中间操作(Intermediate)

一个流可以后面跟随零个或多个 intermediate 操作。中间操作会记录流中的元素所执行的操作，并封装成一个新的流返回，交给下一个操作使用(原来的流执行完中间操作后无法再使用，只能使用返回的新流)。这类操作都是惰性化的（lazy），调用到这类方法时，没有真正开始流的遍历。

Intermediate operations return a new stream. They are always lazy; executing an intermediate operation such as filter() does not actually perform any filtering, but instead creates a new stream that, when traversed, contains the elements of the initial stream that match the given predicate. Traversal of the pipeline source does not begin until the terminal operation of the pipeline is executed.

以下方法为中间操作：

有状态操作：必须拿到流中的所有元素才可继续进行。

无状态操作：unordered()，filter()，map()，flatmap()，peek()

有状态操作：distinct()，sorted()，limit()，skip()

2.结束操作(Terminal)

执行terminal 操作才会真正开始流的遍历。一个流只能有一个 terminal 操作，当这个操作执行后，流就被视为已消耗，无法再被操作。

After the terminal operation is performed, the stream pipeline is considered consumed, and can no longer be used; if you need to traverse the same data source again, you must return to the data source to get a new stream.

结束操作又被分为短路操作与非短路操作：

非短路操作：必须处理完流中的所有元素才可返回。

短路操作：遇到某些符合条件的元素即可返回结果。

如果接收的是一个无限大的stream，又希望在有限时间内完成操作，则需要选择短路操作作为结束操作。

短路操作：anyMatch()，allMatch()，noneMatch()，findFirst()，findAny()

非短路操作：forEach()，toArray()，reduce()，collect()，max()，min()，count()，

forEachOrdered()

**Stream api使用（例子见附件）**

（在流中对对象的修改会导致源数据的变化）

1. filter：过滤，类似于sql中的where。

2. map：将流中的元素映射成另一个元素。

3. flatmap：与map相同，不过flatmap方法的参数为stream。

4. peek：可用于跟踪流的状态。

5. distinct：去重。

6. sorted：排序。不带参数的需要T实现comparable接口。

7. limit：取前n个元素。

8. skip：从n开始取所有元素。

9. anyMatch：任意元素匹配条件则返回true。

10. allMatch：所有元素匹配条件才返回true。

11. noneMatch：所有元素不匹配条件才返回true。

12. findFirst：找到第一个元素。

13. forEach：遍历流，for；

14. max：找到最大的元素。

15. min：找到最小的元素。

16. count：计算元素个数。

17. reduce：规约。

18. collect：

list.stream().collect(Collectors.xxx());

Collectors提供了很多定义好的collect操作，也可以参考Collectors中的实现自定义collect操作。

主要还是来看一下已定义的collect操作：

a. toCollection、toList、toSet、toMap等：转换成集合。

b. counting：计算个数。

c. summarizingInt等：求和。

d. averagingInt等：计算平均值。

e. joining：返回连接流中的元素的toString生成的字符串。

f. maxBy、minBy：计算最大/最小值。

g. reducing：同reduce。

h. collectingAndThen：包裹了一个collect操作，collect操作执行完成后，执行第二个函数。

i. groupingBy：根据元素的属性分组，并将属性作为key返回map。

j. partitioningBy：根据返回的比较结果boolean，进行分组，返回map。

并行流parallelStream相关：

流是否有序取决于数据源和中间操作：比如List/数组本质是有序的，而HashSet不是。比如sorted()操作会使流变得有序，而 unordered()会使流乱序。比如forEach()是忽略流的顺序的。

若流有序，但你并不在意流的顺序时，可以使用 unordered()提升某些有状态操作或结束操作的并行性能。

Whether or not a stream has an encounter order depends on the source and the intermediate operations. Certain stream sources (such as List or arrays) are intrinsically ordered, whereas others (such as HashSet) are not. Some intermediate operations, such as sorted(), may impose an encounter order on an otherwise unordered stream, and others may render an ordered stream unordered, such as [BaseStream.unordered()](https://docs.oracle.com/javase/8/docs/api/java/util/stream/BaseStream.html" \l "unordered--). Further, some terminal operations may ignore encounter order, such as forEach().

In cases where the stream has an encounter order, but the user does not particularly care about that encounter order, explicitly de-ordering the stream with [unordered()](https://docs.oracle.com/javase/8/docs/api/java/util/stream/BaseStream.html" \l "unordered--) may improve parallel performance for some stateful or terminal operations.

1. unordered：乱序。

2. forEach，forEachOrdered：前者忽略流的顺序，后者强制有序进行。

3. findAny：若为非并行流总是取到流中的第一个元素。