

Specialized Streams

IntStream	for int elements
DoubleStream	for double elements
LongStream	for long elements

It has better performance to use these specialized streams when using numeric data types, because there is no boxing/unboxing

Suppress elements

limit	<code>.limit(5)</code> will limit the result to the first 5 elements
skip	<code>.skip(5)</code> will skip the first 5 elements
filter	<code>.filter(e -> e.getSalary() > 200000)</code> will keep the elements that satisfy the given predicate. In this case, all elements that have salary above 200000

Comparing elements

distinct	<code>.distinct()</code> will compare the elements in the stream using equals() and eliminate duplicates
sorted	<code>.sorted((e1, e2) -> e1.getName().compareTo(e2.getName()))</code> will sort the elements with the given comparator. Elements must be Comparable.
min	Similar to sorted, but it will find the min element according to the given comparator
max	Similar to sorted, but it will find the max element according to the given comparator

Apply a function to each element

map	<code>.map(employeeRepository::findById)</code> will apply the given function and substitute the elements in the stream for new elements. In this case, it received a stream of employee IDs and returned a stream of Employee objects
mapToDouble	similar to <i>map</i> , but the function converts the element to the specified primitive type, resulting in a specialized stream IntStream, DoubleStream or LongStream
mapToInt	
mapToLong	

Apply a function to each element (cont)

flatMap	similar to map, but the number of elements resulting may be different. It's normally used to convert a List of List into a single list with all the elements
peek	<code>.peek(e -> e.salaryIncrement(10.0))</code> will apply the given function to all elements in the list, but doesn't substitute the elements in the list

Reduce elements to single value

reduce	<code>.reduce(0.0, Double::sum)</code> will return a single value. It starts with the identity value (0.0) and applies the given function to each element in the array. In this case it's summing all elements, one by one
allMatch	<code>.allMatch(i -> i % 2 == 0);</code> will check if all elements match the given condition. If so, returns true, else returns false
anyMatch	<code>.anyMatch(i -> i % 2 == 0);</code> will check if one of the elements match the given condition. If so, returns true, else returns false
noneMatch	<code>.noneMatch(i -> i % 2 == 0);</code> will check if no elements match the given condition. If so, returns true, else returns false
findFirst	<code>.findFirst()</code> will return an Optional with the first element in the stream
forEach	<code>forEach(e -> e.salaryIncrement(10.0))</code> will apply the given function to each element in the stream, but it's a terminal operation and returns void
count	<code>.count()</code> outputs the number of elements in the stream



Collect elements

toList `collect(Collectors.toList())`
gather all elements in the stream into a List

toSet `collect(Collectors.toSet())`
gather all elements in the stream into a Set

toCollection `collect(Collectors.toCollection(Validator::new))`
gather all elements in the list in an arbitrary Collection

joining `collect(Collectors.joining(", ")).toString()`
will join String elements with the given separator and return the aggregated String

summarizingDouble

summaryStatistics

partitioningBy `.collect(Collectors.partitioningBy(s -> s.getGrade() >= PASS_THRESHOLD, SHOULD))`
will partition the data into 2 categories based on the given condition. The result will be a Map<Boolean, List<Student>>

groupingBy `.collect(Collectors.groupingBy(Employee::getDepartment))`
will group the elements into categories based on the function. The result will be a Map<Department, List<Employee>>

mapping `mapping(Person::getLastName, toSet())`
it receives a function to be applied to all elements and way of collecting downstream the elements. In this case, it will get the last name of all persons and add them to a set

reducing



By **carlmig**

cheatography.com/carlmig/

Published 2nd September, 2020.

Last updated 30th August, 2018.

Page 2 of 2.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>