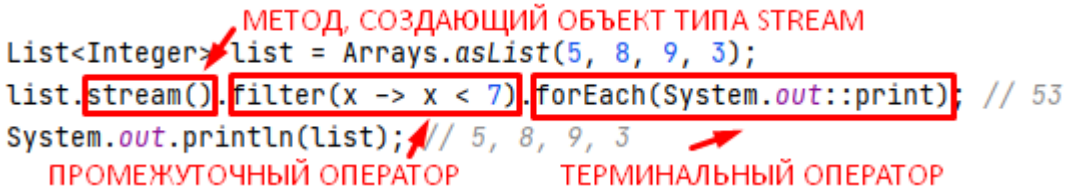
Вопросы по теме Stream API:

1. Что такое Stream?
2. Способы создания Stream?
3. Какие промежуточные операции вы знаете?
4. Какие финальные операции вы знаете?
5. В чем разница между Collection и Stream?
6. Расскажите о параллельной обработке Stream.
7. Критерии, которые могут влиять на производительность в параллельных Stream

Stream API.

API (Application Programming Interface) – это набор инструкций по взаимодействию двух программ между собой (например, пользователь шлет на сервер команду https://www.google.ru/ а сервер шлет ответ в виде html страницы: общение между двумя программами – браузером и сервером посредством команды пользователя).

Stream API – это набор инструкций для работы с данными. Данные получаются из источника, затем обрабатываются промежуточными операторами (их может быть сколько угодно) и завершаются терминальным оператором (он может быть только один). Промежуточный оператор результатом своей работы возвращает объект типа Stream. Источником данных не может быть ассоциативный массив (map).



Stream – интерфейс, содержащий набор методов для работы с объектом типа Stream.

public interface Stream<T> extends BaseStream<T, Stream<T>> { }

Особенности работы Stream:

1. обработка не начнётся до тех пор, пока не будет вызван терминальный оператор;
2. stream после обработки нельзя переиспользовать;
3. работа ведется от терминального оператора.

Терминальный оператор запрашивает элемент у промежуточного оператора, промежуточный у следующего оператора и так по цепочке до источника. Источник отдает элемент, и он проходит через каждый промежуточный оператор, который выполняет какое-то действие над данным элементом или изменяет свое внутреннее поведение. Если при запросе очередного элемента терминальный оператор получает сигнал (от источника или промежуточного оператора), что ничего больше не получит, то он завершает работу stream.

Stream может работать не только с объектами, но и с примитивными типами данных. Эти stream работают с некоторыми отличиями:

– используют спец. лямбды (например, интерфейс IntFunction вместо Function);

– поддерживают дополнительные конечные операции sum(), average(), mapToObj().

int sum = IntStream.of(1, 2, 3).map(x -> x + x).sum(); // 12

Виталий Жуков
unicraft1989@gmail.com
Ссылка на курс Stepik

Создание Stream:

Источник	Код
Значения	**Stream.of(**"знач 1" … "знач n"**)**
Массив	**Arrays.stream(**"массив"**)**
Коллекция	"коллекция"**.stream()**
Строка	"строка"**.chars()**
Файл ()	**Files.lines(**"путь к файлу"**)**
Stream.builder()	**Stream.builder().add(...)...build()**
Stream.iterate()	**Stream.iterate(**"нач.усл.", "выражение для генерации"**)**
Stream.generate()	**Stream.generate(**"выражение для генерации"**)**

При создании stream из файла, каждая строка в файле будет отдельным элементом.

Stream.iterate(1, x -> x + x) – генерация бесконечного stream. Первый элемент – начальное условие, последующие получаются из предыдущего с помощью выражения для генерации.

Stream.generate(() -> Math.random()) - генерация бесконечного stream из константных или случайных значений соответствующих выражению для генерации.

Промежуточные операторы:

Оператор	Назначение
filter(Predicate)	фильтрует по условию
map(Function)	применяет функцию к каждому элементу
flatMap(Function)	преобразует один элемент в массив других
mapMulti(BiConsumer)	улучшенный flatMap
limit(long maxSize)	после первых maxSize элементов завершает стрим
skip(long n)	отбрасывает первые n элементов
sorted(), **sorted**(Comparator)	сортирует
distinct()	убирает повторяющиеся элементы
peek(Consumer)	выполняет действие над каждым элементом возвращает стрим с исходными элементами
takeWhile(Predicate)	возвращает элементы пока они удовлетворяют условию
dropWhile(Predicate)	пропускает элементы пока они удовлетворяют условию затем возвращает оставшуюся часть стрима
boxed()	преобразует примитивный стрим в объектный

Predicate, Function и т.д. – это базовые функциональные интерфейсы.

Виталий Жуков
unicraft1989@gmail.com
Ссылка на курс Stepik

Терминальные операторы:

Оператор	Назначение
void **forEach**(Consumer)	выполняет указанное действие для каждого элемента
void **forEachOrdered**(Consumer)	тоже самое, только добивается правильного порядка вхождения элементов, используется для параллельных стримов
long **count**()	возвращает количество элементов стрима
R **collect**(Collector)	группирует или объединяет элементы по указанному условию собирает элементы в указанную коллекцию
Object[] **toArray**()	возвращает нетипизированный массив с элементами стрима
List<T> **toList**()	возвращает список, который нельзя изменять
T **reduce**(T, BinaryOperator)	преобразует все элементы стрима в один объект
Optional **reduce**(BinaryOperator)	тоже самое, только начальный элемент не задается, а вместо него берется первый элемент стрима
Optional **min**(Comparator)	поиск минимального элемента на основе указанного компаратора
Optional **max**(Comparator)	поиск максимального элемента на основе указанного компаратора
Optional **findAny**()	возвращает любой первый попавшийся элемент стрима (актуально для параллельных стримов)
Optional **findFirst**()	возвращает первый элемент стрима, даже если стрим параллельный
boolean **allMatch**(Predicate)	возвращает true, если все элементы стрима удовлетворяют условию если возвращается false, то прекращается обработка
boolean **anyMatch**(Predicate)	возвращает true, если хотя бы один элемент стрима удовлетворяет условию, если такой элемент встретился, то прекращается обработка
boolean **noneMatch**(Predicate)	возвращает true, если ни один элемент не удовлетворяет условие, если условие = true, оператор прекращает работу и возвращает false
OptionalDouble **average**()	возвращает среднее арифметич. элементов примитивного стрима
sum()	возвращает сумму элементов примитивного стрима
IntSummaryStatistics **summaryStatistics**()	собирает статистику числового примитивного стрима (кол-во элементов, сумму, среднее арифметич., мин. и макс.)

Параллельный stream.

Параллельный стрим использует несколько потоков процессора для обработки своих элементов (многопоточность). Для этого он применяет ForkJoinPool. Данные разбиваются на группы, каждая обрабатывается в отдельном ядре, после обработанные данные соединяются. Если процессор одноядерный, то стрим будет выполнятся как последовательный.

Для создания параллельного стрима нужно вызвать метод *parallelStream()* (у коллекции или массива) или вызвать промежуточный оператор *parallel()*.

При работе с параллельными стримами сохраняется порядок следования элементов, определенный в источнике. Исключение составляет метод *forEach()*, который может выводить элементы в произвольном порядке. Для сохранения порядок следования, нужно вызвать метод *forEachOrdered()*.

Виталий Жуков
unicraft1989@gmail.com
Ссылка на курс Stepik

На производительность параллельного стима влияют следующие факторы:

1. объем данных (разделение / соединение большого объема данных ведет к снижению производительности);
2. количество ядер процессора;
3. структура источника данных (ArrayList быстрее обработается, чем LinkedList, т.к. у LinkedList элементы связаны между собой и их тяжелее разбить на группы);
4. вывод элементов в произвольном порядке работает быстрее, чем в определенном;
5. над примитивными данными обработка происходит быстрее, чем над объектами.

Советы по работе со стримами:

1. не стоит использовать стрим, если задачу можно качественно решить без него;
2. не стоит использовать параллельные стримы без крайней необходимости, обычно накладные расходы нивелируют плюсы от параллельной обработки данных;
3. первыми операциями лучше ставить фильтр или лимит по данным, чтобы остальные операции не тратили время на обработку ненужных данных;
4. один оператор – одна строка, не пишите весь стрим в одну сроку;
5. лучше использовать ссылки на метод, чем лямбда-выражения (меньше накладных расходов и лучше читаемость).

Вопрос: В чем разница между Collection и Stream?

Ответ: Collection – интерфейс, содержащий набор методов для работы с коллекциями. Коллекции позволяют работать со своими элементами по отдельности.

Stream – интерфейс, содержащий набор методов для работы с объектом типа Stream. Stream работает с набором данных как с одним целым.

Виталий Жуков
unicraft1989@gmail.com
Ссылка на курс Stepik