Stream API

Пакет java.util.stream

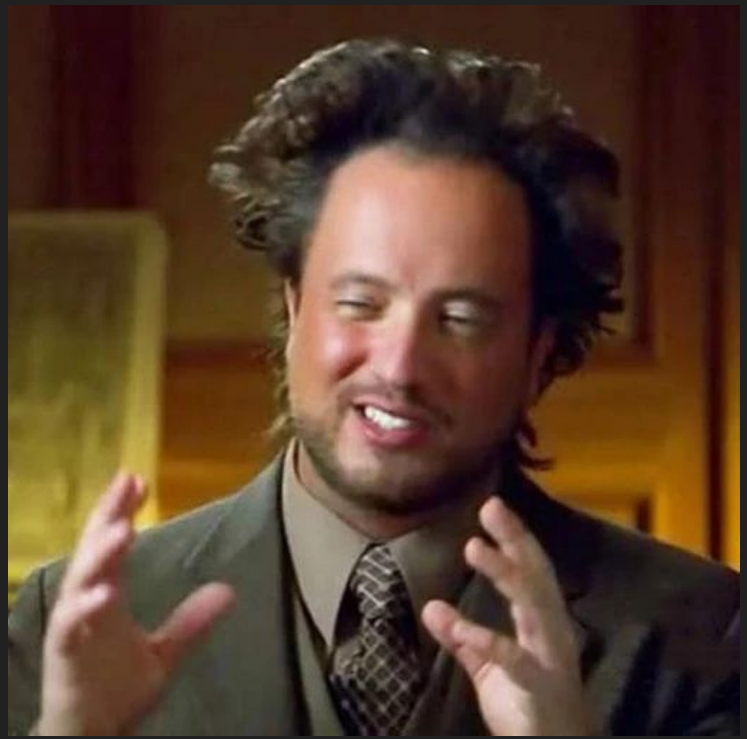
Stream API в Java представляет собой набор классов и интерфейсов, который позволяет функционально обрабатывать коллекции данных.

Он предлагает более чистый и краткий способ итерации, фильтрации, преобразования и выполнения других операций над данными.

- Конвейерная обработка данных
- Поток — последовательность элементов
- Поток может быть последовательным или параллельным
- Конвейер — последовательность операций

статья с картинками

Stream vs Collection



Отличия конвейера от коллекции:

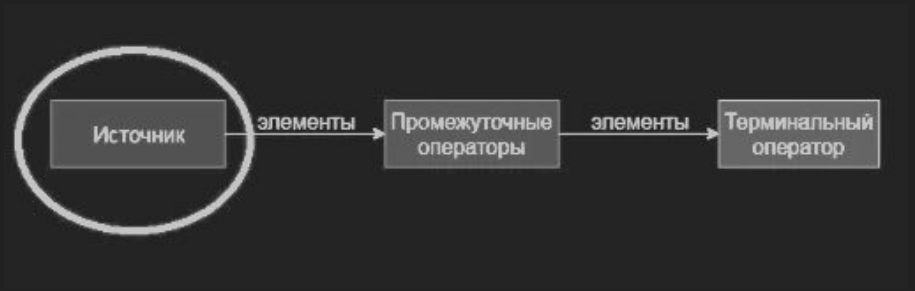
- Элементы не хранятся
- Неявная итерация
- Функциональный стиль — операции не меняют источник
- Большинство операций работают с λ-выражениями
- Ленивое выполнение
- Возможность неограниченного числа элементов

Состав конвейера

```java
package l41.slides.ex1;

import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<String> words = new ArrayList<>();
        words.add("Qwerty");
        words.add("QWE");
        words.add("QWEQWE");
        long count = words.stream()
                .filter(s -> s.length() >5)
                .count();

        // without streams:
        // long count = 0;
        // for (String s : words) {
        //     if (s.length() > 5) {
        //         count++;
        //     }
        // }
    }
}
```

Конвейер =

- Источник
- Промежуточные операции (0 или больше)
- Завершающая операция (одна)



Источники конвейера

- Collection.stream()
- Arrays.stream(Object[])
- Stream.of(Object[])
- IntStream.range(int, int)
- Files.lines(Path), BufferedReader.lines()
- Random.ints()
- Stream.empty()
- Stream.generate(Supplier<T> s)
- Stream.iterate(T seed,
  UnaryOperator<T> f)

Функциональные интерфейсы

```
Stream map(Function mapper)

Stream sorted(Comparator comparator)

Stream filter(Predicate predicate)

void forEach(Consumer action)

// и так далее
```

Function, Comparator, Predicate, Consumer и т.д. – функциональные интерфейсы, в качестве которых можно использовать лямбда выражения

Виды операции

Промежуточные операции:

- Возвращают поток
- Выполняются "лениво"
  - выполнение операции происходит, когда вызывается завершающая операция
- Делятся на:
  - Не хранящие состояние (stateless): выполняются вне зависимости от других элементов
  - Хранящие состояние (stateful): выполнение зависит от других элементов (сортировка)

Терминальные операции:

- Возвращают результат
- Либо имеют побочное действие
- Поток прекращает существование

Промежуточные перации

Методы для промежуточных операций (stateless):

- Stream<T> filter (Predicate<T> p)
  возвращает поток из элементов, соответствующих условию
- Stream<R> map(Function<T,R> mapper)
  преобразует поток элементов T в поток элементов R
- Stream<R> flatMap(Function <T, Stream<R>> mapper)
  преобразует каждый элемент потока T в поток элементов R
- Stream<T> peek(Consumer<T> action)
  выполняет действие для каждого элемента потока T

Методы для промежуточных операций (stateful)

- Stream<T> distinct()

  возвращает поток неповторяющихся элементов
- Stream<T> sorted(Comparator<T> comp)

  возвращает отсортированный поток
- Stream<T> limit(long size)

  возвращает усеченный поток из size элементов
- Stream<T> skip(long n)

  возвращает поток, пропустив n элементов

Терминальные операции

Методы для завершающих операций:

- void forEach(Consumer<T> action)
- void forEachOrdered(Consumer<T> action)
  выполняет действие для каждого элемента потока
  второй вариант гарантирует сохранение порядка элементов

- Optional<T> min(), Optional<T> max()
  возвращают минимальный и максимальный элементы,

- long count(), int (long, double) sum()
  возвращают количество и сумму элементов

- OptionalInt, OptionalLong, OptionalDouble
  - int getAsInt(), long getAsLong(), double getAsDouble()

Контрольная точка

- Понятно ли?

Если все ясно, ставим плюсы, иначе - задаем вопросы.

Домашнее задание

- Ознакомиться с документацией и статьей по ссылкам со второго слайда

The end