Как изучить Java Stream API [+5 ресурсов]

Описание

Поток в Java – это последовательность элементов, над которыми могут выполняться последовательные или параллельные операции. Может быть “n” количество промежуточных операций и, наконец, конечная операция, после которой возвращается результат.

Что такое поток?

Потоками можно управлять с помощью Stream API, который был представлен в Java 8. Представьте себе поток как производственный конвейер, в котором некоторые товары должны быть изготовлены, отсортированы, а затем упакованы для отправки. В Java эти товары – объекты или коллекции объектов, операции – производство, сортировка и упаковка, а конвейер – поток.

Компонентами потока являются:

- Начальный вход
- Промежуточные операции
- Конечная операция
- Конечный результат

Давайте рассмотрим некоторые особенности потока в Java:

- Поток – это не структура данных в памяти; скорее это последовательность массивов, объектов или коллекций объектов, с которыми работают

определенные методы.

- Потоки являются декларативными по своей природе, т.е. вы указываете, что делать, но не как делать.
- Они могут быть использованы только один раз, поскольку нигде не хранятся.
- Поток не изменяет исходную структуру данных; он только создает на ее основе новую структуру.
- Он возвращает конечный результат, полученный от конечного метода в конвейере.

Stream API в сравнении с Collections Processing

Коллекция – это структура данных в памяти, которая хранит и обрабатывает данные. Коллекции предоставляют такие структуры данных, как Set, Map, List и т.д., для хранения данных. С другой стороны, поток – это способ эффективной передачи данных после их обработки через конвейер.

Вот пример коллекции ArrayList:

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        list.add(0, 3);
        System.out.println(list);
    }
}

Output:
[3]
```

Как видно из примера выше, вы можете создать коллекцию ArrayList, хранить в ней данные, а затем работать с ними, используя различные методы. Используя поток, вы можете работать с существующей структурой данных и возвращать новое измененное значение. Ниже приведен пример создания коллекции ArrayList и ее фильтрации с помощью потока.

```
import java.util.ArrayList;
import java.util.stream.Stream;

public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList();

        for (int i = 0; i < 20; i++) {
            list.add(i+1);
```

```
        }

        System.out.println(list);

        Stream<Integer> filtered = list.stream().filter(num -> num > 10);
        filtered.forEach(num -> System.out.println(num + " "));
    }
}

#Output

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
11
12
13
14
15
16
17
18
19

20
```

В приведенном выше примере поток создается с использованием существующего списка, и список итерируется для фильтрации значений больше 10. Обратите внимание, что поток ничего не хранит, список просто итерируется, а результат выводится на печать. Если вы попытаетесь распечатать поток, то вместо значений вы получите ссылку на поток.

Работа с Java Stream API

Java Stream API принимает исходную коллекцию элементов или последовательность элементов, а затем выполняет над ними операции для получения конечного результата. Поток – это просто конвейер, через который проходит последовательность элементов и преобразуется определенным образом.

Поток может быть создан из различных источников, включая:

- Коллекция, например, список или набор.
- Массив.
- Из файлов и их путей с помощью буфера.

Существует два типа операций, выполняемых в потоке:-

- Промежуточные операции

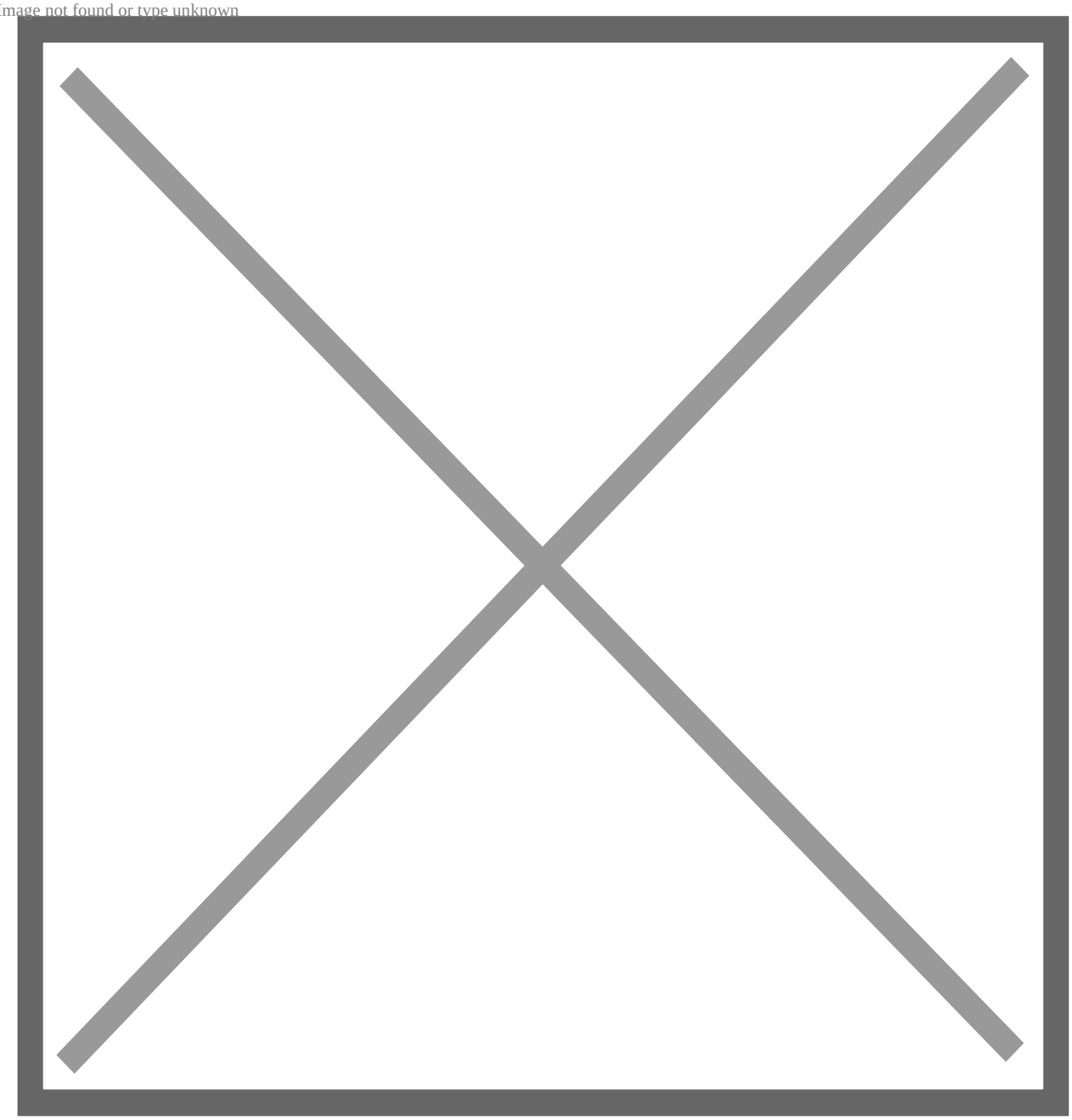
- Терминальные операции

Промежуточные и терминальные операции

Каждая промежуточная операция возвращает новый поток, который преобразует входные данные с помощью указанного метода. На самом деле ничего не происходит; вместо этого поток передается следующему потоку. Только в терминальной операции поток обходится, чтобы получить желаемый результат. Например, у вас есть список из 10 чисел, которые вы хотите отфильтровать и затем сопоставить с чем-то. Не каждый элемент списка будет немедленно пройден, чтобы получить отфильтрованный результат и сопоставить его с чем-то другим. Вместо этого будут проверяться отдельные элементы, и если они удовлетворяют условию, они будут сопоставлены. Новые потоки для каждого элемента.

Операция отображения будет выполняться на отдельные элементы, удовлетворяющие фильтру, а не на весь список. А во время терминальной операции они обходятся и объединяются в один результат. После выполнения терминальной операции поток расходуется и больше не может быть использован. Для повторного выполнения тех же операций необходимо создать новый поток.

Image not found or type unknown



Имея поверхностное понимание того, как работают потоки, давайте перейдем к деталям реализации потоков в Java.

Пустой поток

Создайте пустой поток, используя метод **empty** API Stream.

```
import java.util.stream.Stream;

public class Main {
```

```
    public static void main(String[] args) {
        Stream emptyStream = Stream.empty();
        System.out.println(emptyStream.count());
    }
}

#Output

0
```

Здесь, если вы выведете количество элементов в этом потоке, вы получите на выходе 0, потому что это пустой поток без элементов. Пустые потоки очень помогают избежать исключений нулевого указателя.

Поток из коллекций

Коллекции, такие как Lists и Set, предоставляют метод stream(), который позволяет вам создать поток из коллекции. Затем созданный поток можно просмотреть, чтобы получить конечный результат.

```
ArrayList<Integer> list = new ArrayList();

for (int i = 0; i < 20; i++) {
    list.add(i+1);
}

System.out.println(list);

Stream<Integer> filtered = list.stream().filter(num -> num > 10);
filtered.forEach(num -> System.out.println(num + " "));

#Output

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
11
12
13
14
15
16
17
18
19

20
```

Поток из массивов

Метод **Arrays.stream()** используется для создания потока из массива.

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        String[] stringArray = new String[]{"this", "is", "geekflare"};
        Arrays.stream(stringArray).forEach(item -> System.out.print(item + " "
    }
}

#Output

this is geekflare
```

Вы также можете указать начальный и конечный индекс элементов для создания потока. Начальный индекс является инклюзивным, а конечный – эксклюзивным.

```
String[] stringArray = new String[]{"this", "is", "geekflare"};
Arrays.stream(stringArray, 1, 3).forEach(item -> System.out.print(item + " "))

#Output
this is geekflare
```

Нахождение минимального и максимального числа с помощью потоков

Получить доступ к максимальному и минимальному числу коллекции или массива можно с помощью компараторов в Java. Методы **min()** и **max()** принимают компаратор и возвращают объект Optional. Объект Optional – это объект-контейнер, который может содержать или не содержать ненулевое значение. Если он содержит ненулевое значение, вызов метода **get()** на нем вернет это значение.

```
import java.util.Arrays;
import java.util.Optional;

public class MinMax {
    public static void main(String[] args) {
        Integer[] numbers = new Integer[]{21, 82, 41, 9, 62, 3, 11};

        Optional<Integer> maxValue = Arrays.stream(numbers).max(Integer::compa
        System.out.println(maxValue.get());

        Optional<Integer> minValue = Arrays.stream(numbers).min(Integer::compa
        System.out.println(minValue.get());
    }
}

#Output
```

82

3

Учебные ресурсы

Теперь, когда у вас есть базовое понимание потоков в Java, вот 5 ресурсов, которые помогут вам хорошо освоить Java 8:

Java 8 в действии

Эта книга представляет собой руководство, демонстрирующее новые возможности Java 8, включая потоки, лямбды и функциональный стиль программирования. Викторины и вопросы для проверки знаний также являются частью книги, что поможет вам восстановить пройденный материал. Вы можете приобрести эту книгу в мягкой обложке, а также в формате аудиокниги.

Java 8 Lambdas: Functional Programming For The Masses

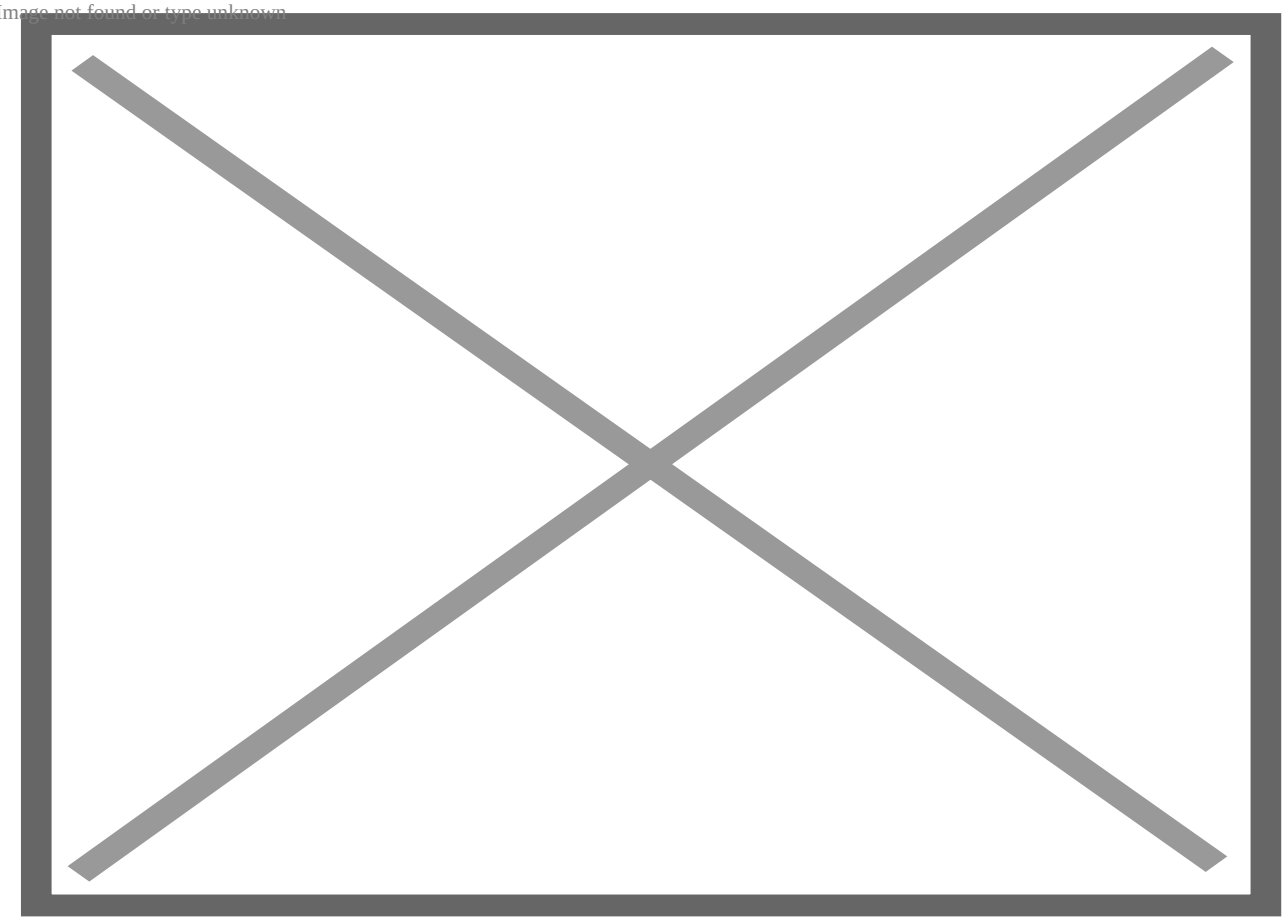
Эта книга создана специально для того, чтобы научить основных разработчиков Java SE тому, как добавление лямбда-выражений влияет на язык Java. Она включает в себя понятные объяснения, упражнения с кодом и примеры, которые помогут вам освоить лямбда-выражения Java 8. Она доступна в мягкой обложке и в издании Kindle.

Java SE 8 для очень нетерпеливых

Если вы опытный разработчик Java SE, эта книга расскажет вам об улучшениях, внесенных в Java SE 8, о потоковом API, добавлении лямбда-выражений, улучшениях в параллельном программировании на Java, а также о некоторых возможностях Java 7, о которых большинство людей не знают. Она доступна только в мягкой обложке.

Изучайте функциональное программирование на Java с помощью лямбд и потоков

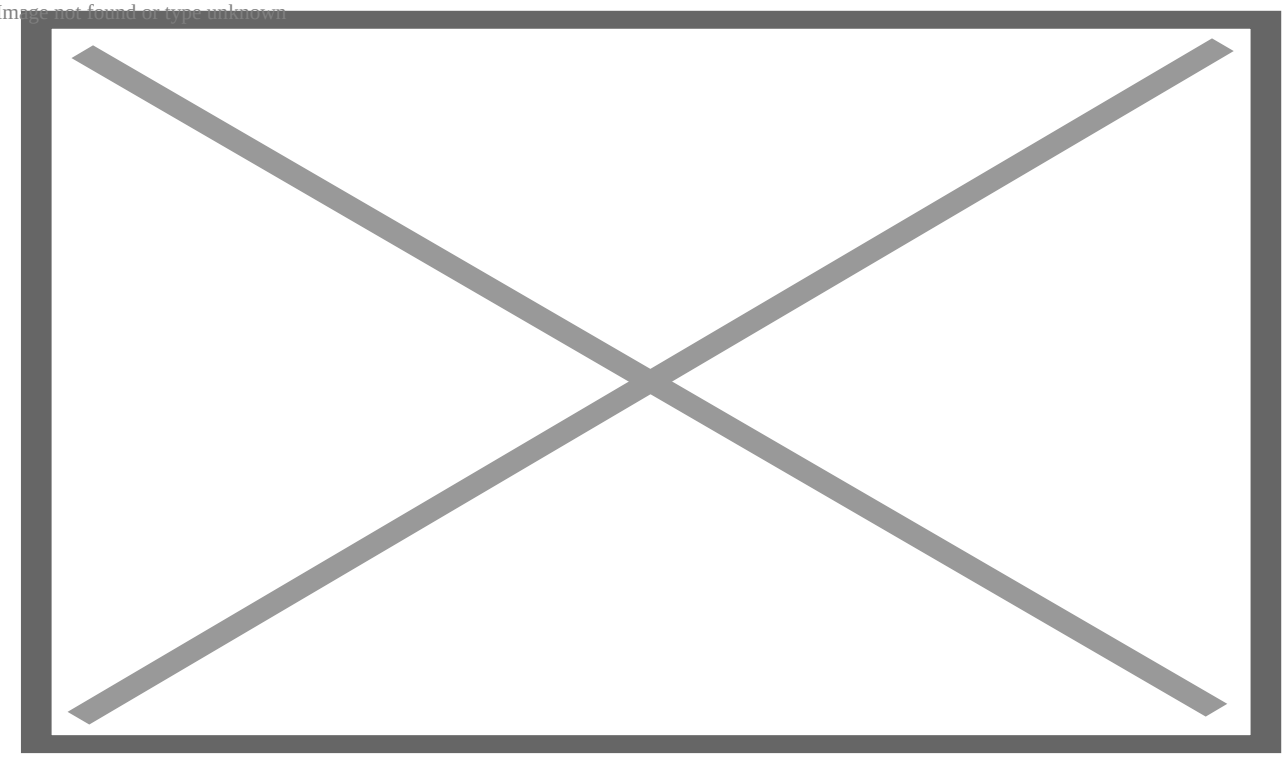
Image not found or type unknown



Этот курс Udemy изучает основы функционального программирования в Java 8 и 9. Лямбда-выражения, ссылки на методы, потоки и функциональные интерфейсы – вот те понятия, на которых сосредоточен этот курс. Он также включает в себя множество Java-загадок и упражнений, связанных с функциональным программированием.

Библиотека классов Java

Image not found or type unknown



Java Class Library – это часть специализации Core Java, предлагаемой Coursera. Он научит вас писать безопасный для типов код, используя Java Generics, понимать библиотеку классов, состоящую из более чем 4000 классов, работать с файлами и обрабатывать ошибки во время выполнения. Однако для прохождения этого курса есть некоторые предварительные условия:

- Введение в Java
- Введение в объектно-ориентированное программирование на Java
- Объектно-ориентированные иерархии в Java

Заключительные слова

Java Stream API и введение лямбда-функций в Java 8 упростили и улучшили многие вещи в Java, такие как параллельная итерация, функциональные интерфейсы, меньший объем кода и т.д.. Однако потоки имеют некоторые ограничения; самое большое ограничение заключается в том, что они могут быть использованы только один раз. Если вы являетесь разработчиком Java, ресурсы, упомянутые выше, помогут вам понять эти темы гораздо более подробно, поэтому обязательно ознакомьтесь с ними.

Дата Создания
27.04.2023