HowToDoInJava

Java Stream forEachOrdered()

Last Updated: March 15, 2022 | By: Lokesh Gupta | Java 8 | Java Stream Basics, Java Stream Methods

The **Stream forEachOrdered()** method is used to iterate over all the elements of the given Stream and to perform a Consumer action on each element of the Stream, **in the encounter order** of the Stream if the Stream has a defined encounter order.

Table Of Contents

1. Stream forEachOrdered() Method

1.1. Method Syntax

The `forEachOrdered()` method syntax is as follows:

Syntax

```
void forEachOrdered(Consumer<? super T> action)
```

Here `Consumer` is a functional interface and action represents a non-interfering action to be performed on each element in the stream.

1.2. Description

- The `forEachOrdered()` method is a **terminal operation**. It means that it does not return an output of type `Stream`.
- After *forEachOrdered()* is performed, the stream pipeline is considered consumed, and can no longer be used.
- If we need to traverse the same data source again, we must return to the data source to get a new stream.
- Performs an `action` for each element of this stream, in the **encounter order** of the stream if the stream has a defined encounter order.
- Performing the action for one element `happens-before` performing the `action` for subsequent elements. But for any given element, the `action` may be performed in whatever `Thread` the library chooses.

2. Stream forEach() vs forEachOrdered()

The behavior of *forEach()* operation is explicitly non-deterministic. For parallel streams, `forEach()` operation does not guarantee to respect the encounter order of the Stream.

While the *forEachOrdered()* operation **respects the** encounter order of the stream if the stream has a defined encounter order. This behavior is also true for parallel streams as well as sequential streams.

> We may lose the benefits of parallelism if we use `forEachOrdered()` with parallel Streams.

Let us understand with a Java program that iterates over a Stream of Integers and verifies encounter order.

```
List<Integer> list = Arrays.asList(2, 4, 6, 8, 10);

list.stream().parallel().forEach( System.out::println );    //1

list.stream().parallel().forEachOrdered( System.out::println ); //2
```

Now, lets compare the output of both statements

```
//forEach()
6
10
8
4
2

//forEachOrdered()
2
4
6
8
10
```

3. Stream forEachOrdered() Examples

Example 1: Java program to iterate over Stream of Integers and to print into the Console

```
List<Integer> list = Arrays.asList(2, 4, 6, 8, 10);

list.stream()
    .forEachOrdered( System.out::println );
```

Program output.

```
2
4
6
8
10
```

Example 2: Java program to iterate over Stream of Integers in reverse order

```
List<Integer> list = Arrays.asList(2, 4, 6, 8, 10);

list.stream()
  .sorted(Comparator.reverseOrder())
  .forEachOrdered(System.out::println);
```

Program output.

```
10
8
6
4
2
```

Drop me your questions related to the **Stream forEachOrdered() method** in Java Stream API.

Happy Learning !!

Sourcecode on Github

Was this post helpful?

Let us know if you liked the post. That's the only way we can improve.

Yes

No

Recommended Reading:

1. Java Stream reuse – traverse stream multiple times?
2. Java Stream count() Matches with filter()
3. Java Stream forEach()
4. Java Stream sorted()
5. Java Stream max()
6. Java Stream min()
7. Java Stream limit()
8. Java Stream skip()
9. Java Stream findFirst()
0. Java Stream findAny()

Leave a Comment

Name *

Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...

HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us

- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial