HowToDoInJava

Java Stream peek()

Last Updated: March 29, 2022 | By: Lokesh Gupta | Java 8 | Java Stream Basics, Java Stream Methods

Java `Stream` `peek()` method **returns a new Stream consisting of all the elements from the original Stream after applying a given `Consumer` action**.

Note that the `peek()` method is an intermediate Stream operation so, to process the Stream elements through `peek()`, we must use a terminal operation. Using `Stream.peek()` without any terminal operation does nothing.

Table Of Contents

1. Stream peek() Method

1.1. Usage

According to Java docs, the purpose of *peek()* method is to **support debugging when we want to see the elements as they flow through the Stream processing pipeline.**

We can call *peek()* method after every intermediate operation to see the effect of intermediate operation on the Stream elements.

Pseudo Code

```
Stream<T> stream = createStream();

stream.operationOne()
    .peek()
    .operationTwo()
    .peek()
    .terminalOperation();
```

1.2. Method Syntax

The `peek()` returns a new Stream consisting of elements from the original Stream.

Here `action` is a **non-interfering action** to perform on the elements as they are consumed from the Stream. The result elements after performing the `action` are placed into the new Stream.

Syntax

```
Stream<T> peek(Consumer<? super T> action)
```

1.3. Description

- Stream `peek()` method is an *intermediate operation*.
- It *returns a Stream* consisting of the elements of current stream.
- It additionally *perform the provided* `action` *on each element* as elements.

- For parallel stream pipelines, the `action` may be called at whatever time and in whatever thread the element is made available by the upstream operation.
- If the `action` modifies shared state, it is itself responsible for providing the required synchronization.
- `peek()` exists mainly to support **debugging**, where we want to see the elements as they flow past a certain point in a pipeline.

2. Stream peek() Examples

2.1. Using peek() Without Terminal Operation

As mentioned above, `Stream.peek()` without any terminal operation does nothing.

Stream.peek() without terminal operation

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);

list.stream()
  .peek( System.out::println );   //prints nothing
```

2.2. Using peek() with Terminal Operation

Java program to use `peek()` API to debug the Stream operations and *log Stream elements* as they are processed.

Stream.peek() with terminal operation

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);

List<Integer> newList = list.stream()
      .peek(System.out::println)
      .collect(Collectors.toList());

System.out.println(newList);
```

Program output.

Output

```
1
2
3
4
5
[1, 2, 3, 4, 5]
```

3. Conclusion

Stream.peek() method can be useful in visualizing how the stream operations behave and understanding the implications and interactions of complex intermediate stream operations.

Though it is entirely possible to alter the inner state of elements in the Stream, it is never recommended and shall be avoided.

Happy Learning !!

Sourcecode on Github

Recommended Reading:

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

Subscribe

* We do not spam !!

Leave a Comment

Name *

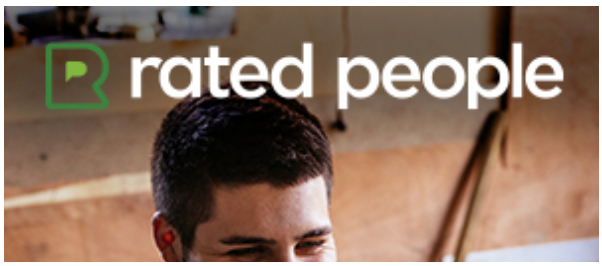
Email *

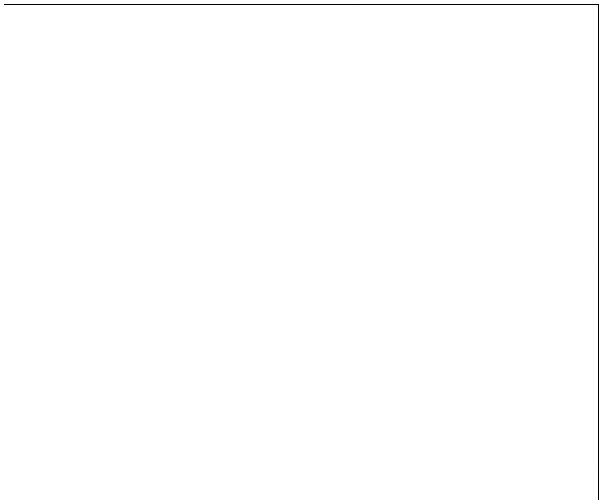
Website

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