HowToDoInJava

Java Spliterator interface

Last Updated: December 26, 2020 By: Lokesh Gupta Java Collections Java Collections, Java Iterator

Java Spliterator interface is an internal iterator that breaks the stream into the smaller parts. These smaller parts can be processed in parallel.

In real life programming, we may never need to use **Spliterator** directly. Under normal operations, it will behave exactly same as Java **Iterator**.

```
Spliterator Syntax

Spliterator<T> spliterator = list.spliterator();
```

The Java collection classes provide ***default* stream()** and **parallelStream()** methods which internally use the Spliterator through the call to the spliterator(). It helps in processing the collection data in parallel.

```
Collection.java

default Stream<E> stream() {
    return StreamSupport.stream(spliterator(), false);
}

default Stream<E> parallelStream() {
    return StreamSupport.stream(spliterator(), true);
}
```

1. Java Spliterator Features

Following is a list of features provided by Spliterator in Java.

1. Spliterator has been introduced in Java 8.
2. It provides support for parallel processing of stream of elements for any collection.
3. It provides **tryAdvance()** method to iterate elements individually in different threads. It helps in parallel processing.
4. To iterate elements sequentially in a single Thread, use **forEachRemaining()** method.
5. The **trySplit()** method is used partition the spliterator, if it is possible.
6. It helps in combining the `hasNext()` and `next()` operations into one method.

2. Java Spliterator Methods

1. **int characteristics()** : returns the list of characteristics of the spliterator. It can be any of ORDERED, DISTINCT, SORTED, SIZED, NONNULL, IMMUTABLE, CONCURRENT, and SUBSIZED.
2. **long estimateSize()** : returns an estimate of the number of elements that would be encountered by a forEachRemaining() traversal, or returns Long.MAX_VALUE if infinite, unknown, or too expensive to compute.
3. **default void forEachRemaining(Consumer action)** : performs the given action for each remaining element, sequentially in the current thread, until all elements have been processed or the action throws an exception.
4. **default Comparator getComparator()** : if the spliterator's source is SORTED by a Comparator, returns that Comparator.
5. **default long getExactSizeIfKnown()** : returns estimateSize() if this Spliterator is SIZED, else -1.
6. **default boolean hasCharacteristics(int characteristics)** : returns true if the dpliterator's characteristics() contain all of the given characteristics.

7. **boolean tryAdvance(Consumer action)** : if a remaining element exists, performs the given action on it, returning `true`; else returns `false`.
8. **Spliterator trySplit()** : if the spliterator can be partitioned, returns a Spliterator covering elements, that will, upon return from this method, not be covered by this Spliterator.

3. Java Spliterator Example

3.1. Spliterator characteristics() example

Java example to verify the characteristics of Spliterator obtained for ArrayList.

```
Spliterator example

ArrayList<String> list = new ArrayList<>();

Spliterator<String> spliterator = list.spliterator();

int expected = Spliterator.ORDERED | Spliterator.SIZED | Spliterator.SUBSIZED;

System.out.println(spliterator.characteristics() == expected);  //true

if (spliterator.hasCharacteristics(Spliterator.ORDERED)) {
    System.out.println("ORDERED");
}

if (spliterator.hasCharacteristics(Spliterator.DISTINCT)) {
    System.out.println("DISTINCT");
}

if (spliterator.hasCharacteristics(Spliterator.SORTED)) {
    System.out.println("SORTED");
}

if (spliterator.hasCharacteristics(Spliterator.SIZED)) {
    System.out.println("SIZED");
}

if (spliterator.hasCharacteristics(Spliterator.CONCURRENT)) {
    System.out.println("CONCURRENT");
}

if (spliterator.hasCharacteristics(Spliterator.IMMUTABLE)) {
    System.out.println("IMMUTABLE");
```

```
}

if (spliterator.hasCharacteristics(Spliterator.NONNULL)) {
    System.out.println("NONNULL");
}

if (spliterator.hasCharacteristics(Spliterator.SUBSIZED)) {
    System.out.println("SUBSIZED");
}
```

Program Output.

```
Console

true

ORDERED
SIZED
SUBSIZED
```

3.2. Spliterator estimateSize() and getExactSizeIfKnown() example

Java example to get the size of backing collection i.e. number of elements to iterate by spliterator.

```
Spliterator example

ArrayList<String> list = new ArrayList<>();

list.add("A");
list.add("B");
list.add("C");
list.add("D");

Spliterator<String> spliterator = list.spliterator();

System.out.println(spliterator.estimateSize());
System.out.println(spliterator.getExactSizeIfKnown());
```

Program Output.

```
Console
```

```
4
4
```

3.3. Spliterator getComparator() example

Java example to find the comparator used by spliterator.

```
Spliterator example

SortedSet<String> set = new TreeSet<>( Collections.reverseOrder() );

set.add("A");
set.add("D");
set.add("C");
set.add("B");

System.out.println(set);

System.out.println(set.spliterator().getComparator());
```

Program Output.

```
Console

[D, C, B, A]
java.util.Collections$ReverseComparator@7852e922
```

3.4. Spliterator trySplit() example

Java example to split the elements to two groups and iterate independently.

```
Spliterator example

ArrayList<String> list = new ArrayList<>();

list.add("A");
list.add("B");
list.add("C");
list.add("D");
list.add("E");
list.add("F");
```

```
Spliterator<String> spliterator1 = list.spliterator();
Spliterator<String> spliterator2 = spliterator1.trySplit();

spliterator1.forEachRemaining(System.out::println);

System.out.println("========");

spliterator2.forEachRemaining(System.out::println);
```

Program Output.

```
Console

D
E
F
========
A
B
C
```

3.5. Spliterator forEachRemaining() example

Java example to perform hasNext() and next() operations in single statement using forEachRemaining() method.

```
Spliterator example

ArrayList<String> list = new ArrayList<>();

list.add("A");
list.add("B");
list.add("C");
list.add("D");

Spliterator<String> spliterator = list.spliterator();

spliterator.forEachRemaining(System.out::println);
```

Program Output.

```
Console
```

```
A
B
C
D
```

4. Conclusion

In this tutorial, we learned the Java Spliterator interface. We learned the Spliterator methods and simple examples to iterate over collections elements and streams apart from other useful methods in Spliterator.

Drop me your questions in comments section.

Happy Learning !!

References:

Spliterator Interface Java Docs

Recommended Reading:

1. ArrayList spliterator() method example
2. Java Iterator interface example
3. Java ListIterator interface

4. Java Comparable Interface
5. Java Comparator Interface
6. Interface vs Abstract Class in Java
7. Java Cloneable interface – Is it broken?
8. Private Methods in Interface – Java 9
9. Spring boot – CommandLineRunner interface example
0. Spring Boot Async Rest Controller with Callable Interface

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

** We do not spam !!*

2 thoughts on “Java Spliterator interface”

Raj G

May 16, 2020 at 9:00 pm

nice understanding with example

Reply

Mafyak

February 10, 2020 at 5:12 am

Why would you use spliterator if you can use stream?

Reply

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