HowToDoInJava

Java 8 Stream concat()

Last Updated: August 30, 2020 By: Lokesh Gupta Java 8 Java Stream Basics

Learn to use **Stream.concat()** method is used to **merge two streams into one stream** which consist of all elements of both merged streams.

1. Stream concat() method

```
Syntax

static <T> Stream<T> concat(Stream<? extends T> firstStream,
                            Stream<? extends T> secondStream)
```

- This method creates a **lazily concatenated stream** whose elements are all the elements of the **firstStream** followed by all the elements of the **secondStream**.
- The resulting stream is **ordered** if both of the input streams are ordered.
- The resulting stream is **parallel** if either of the input streams is parallel.
- When the resulting stream is **closed**, the close handlers for both input streams are invoked.

2. Merge two streams

Java example to merge two streams of numbers – to obtain an stream which contains numbers from both streams.

```
How to merge two java streams
```

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

public class Main
{
  public static void main(String[] args)
  {
    Stream<Integer> firstStream = Stream.of(1, 2, 3);
      Stream<Integer> secondStream = Stream.of(4, 5, 6);

      Stream<Integer> resultingStream = Stream.concat(firstStream, secondStream);

      System.out.println( resultingStream.collect(Collectors.toList()) );
  }
}
```

Program Output.

```
Console

[1, 2, 3, 4, 5, 6]
```

3. Merge multiple streams

Java example to merge four streams of numbers – to obtain an stream which contains numbers from all streams. Notice we have made a static import **to Stream.concat()** function which makes the code readable.

```
How to merge multiple java streams

import java.util.stream.Collectors;
import java.util.stream.Stream;
import static java.util.stream.Stream.*;

public class Main
{
  public static void main(String[] args)
  {
    Stream<Integer> first = Stream.of(1, 2);
      Stream<Integer> second = Stream.of(3,4);
      Stream<Integer> third = Stream.of(5, 6);
      Stream<Integer> fourth = Stream.of(7,8);
```

```
        Stream<Integer> resultingStream = Stream.concat(first, concat(second, concat

        System.out.println( resultingStream.collect(Collectors.toList()) );
    }
}
```

Program Output.

```
Console

[1, 2, 3, 4, 5, 6, 7, 8]
```

4. Java merge streams and retain unique elements

4.1. Numbers and strings

While merging two streams, we can use **distinct()** API and resulting stream will contain only unique elements.

```
Merge streams and retain unique elements

import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Main
{
  public static void main(String[] args)
  {
    Stream<Integer> firstStream = Stream.of(1, 2, 3, 4, 5, 6);
    Stream<Integer> secondStream = Stream.of(4, 5, 6, 7, 8, 9);

    Stream<Integer> resultingStream = Stream.concat(firstStream, secondStream)
                        .distinct();

    System.out.println( resultingStream.collect(Collectors.toList()) );
  }
}
```

Program Output.

```
Console

[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

4.2. Custom objects

In case of merging streams of custom objects, we can drop the duplicate elements during stream iteration. We can use the **distinctByKey()** function created for java stream distinct by object property example.

```
Merge streams and retain unique objects

import java.util.ArrayList;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.function.Function;
import java.util.function.Predicate;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Main
{
  public static void main(String[] args)
  {
    Stream<Employee> stream1 = getEmployeeListOne().stream();
    Stream<Employee> stream2 = getEmployeeListTwo().stream();

    Stream<Employee> resultingStream = Stream.concat(stream1, stream2)
        .filter(distinctByKey(Employee::getFirstName));

    System.out.println( resultingStream.collect(Collectors.toList()) );
  }

  public static <T> Predicate<T> distinctByKey(Function<? super T, Object> keyExtr
  {
      Map<Object, Boolean> map = new ConcurrentHashMap<>();
      return t -> map.putIfAbsent(keyExtractor.apply(t), Boolean.TRUE) == null;
  }

  private static ArrayList<Employee> getEmployeeListOne()
  {
    ArrayList<Employee> list = new ArrayList<>();
    list.add( new Employee(1l, "Lokesh", "Gupta") );
    list.add( new Employee(5l, "Brian", "Piper") );
    list.add( new Employee(7l, "Charles", "Piper") );
    list.add( new Employee(6l, "David", "Beckham") );
        return list;
```

```
    }

    private static ArrayList<Employee> getEmployeeListTwo()
    {
      ArrayList<Employee> list = new ArrayList<>();
      list.add( new Employee(2l, "Lokesh", "Gupta") );
      list.add( new Employee(4l, "Brian", "Piper") );
      list.add( new Employee(3l, "David", "Beckham") );
          return list;
    }
  }
```

Program Output.

```
Console

[
Employee [id=1, firstName=Lokesh, lastName=Gupta],
Employee [id=5, firstName=Brian, lastName=Piper],
Employee [id=7, firstName=Charles, lastName=Piper],
Employee [id=6, firstName=David, lastName=Beckham]]
```

Drop me your questions in comments section related to **merging two or more streams** of objects in Java 8.

Happy Learning !!

Reference:

Stream concat() API Docs

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 String concat() method example
2. Java Stream reuse – traverse stream multiple times?
3. Java Stream forEach()
4. Java Stream max()
5. Java Stream limit()
6. Java Stream skip()
7. Sorting a Stream by Multiple Fields in Java
8. Java 8 – Join or append stream of strings
9. Convert Iterable or Iterator to Stream in Java
0. Java Stream findAny()

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 8 Stream concat()”

Takidoso

August 20, 2019 at 7:22 pm

Hi actually the merging is a concatenating here.
What I am interested in is how to merge streams like figuratively a zipper.

having stream A with numbers (1, 3, 5, 7…) and stream B with even ones (2, 4 ,6 ,8 …)
and a merge result of exactly like this:
(1,2,3,4,5,6,7,8, …)

Reply

Girraj

December 12, 2019 at 11:37 am

```
Stream a = Stream.of(1,3,5,7);
Stream b = Stream.of(2,4,6,8);
Stream c = Stream.concat(a,b).sorted();
```

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