HowToDoInJava

Java Stream anyMatch()

Last Updated: March 30, 2022 By: Lokesh Gupta Java 8 Java Stream Basics, Java Stream Methods

Java **Stream *anyMatch(predicate)*** is a **terminal short-circuit operation**. It is used to check if the Stream contains at least one element that satisfies the given predicate.

Table Of Contents

1. Stream anyMatch() Method

1.1. Syntax

Here `predicate` a non-interfering, stateless Predicate to apply to elements of the stream.

The `anyMatch()` method returns `true` if at least one element satisfies the condition provided by `predicate`, else `false`.

Syntax

```
boolean anyMatch(Predicate<? super T> predicate)
```

1.2. Description

- It is a short-circuiting terminal operation.
- It returns whether any elements of this stream match the provided predicate.
- May not evaluate the predicate on all elements if not necessary for determining the result. Method returns `true` as soon as first matching element is encountered.

- If the stream is empty then `false` is returned and the predicate is not evaluated.
- The **difference between allMatch() and anyMatch()** is that `anyMatch()` returns `true` if any of the elements in a stream matches the given predicate. When using `allMatch()`, all the elements must match the given predicate.

2. Stream anyMatch() Examples

Example 1: Checking if Stream contains a Specific Element

In this Java example, we are using the `anyMatch()` method to check if the stream contains the string `"four"`.

As we see that the stream contains the string, so the output of the example is `true`.

Checking if Stream contains an element

```
Stream<String> stream = Stream.of("one", "two", "three", "four");

boolean match = stream.anyMatch(s -> s.contains("four"));

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

Program output.

Output

```
true
```

Example 2: `Stream anyMatch()` with Multiple Predicates

To satisfy multiple conditions, **create a composed predicate with two or more simple predicates**.

In the given example, we have a list of `Employee`. We want to check if there is an employee who is above the age of 50 – is earning above 40,000.

In the list, employees `"D"` and `"F"` are earning above 40k and their age is above 50, so the result is `true`.

Stream anyMatch() with multiple conditions

```
import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Stream;
import lombok.AllArgsConstructor;https://howtodoinjava.com/wp-admin/tools.php
import lombok.Data;

public class Main
```

```
{
  public static void main(String[] args)
  {
    Predicate<Employee> olderThan50 = e -> e.getAge() > 50;
    Predicate<Employee> earningMoreThan40K = e -> e.getSalary() > 40_000;
    Predicate<Employee> combinedCondition = olderThan50.and(earningMoreThan40K);

    boolean result = getEmployeeStream().anyMatch(combinedCondition);
    System.out.println(result);
  }

  private static Stream<Employee> getEmployeeStream()
  {
    List<Employee> empList = new ArrayList<>();
    empList.add(new Employee(1, "A", 46, 30000));
    empList.add(new Employee(2, "B", 56, 30000));
    empList.add(new Employee(3, "C", 42, 50000));
    empList.add(new Employee(4, "D", 52, 50000));
    empList.add(new Employee(5, "E", 32, 80000));
    empList.add(new Employee(6, "F", 72, 80000));

    return empList.stream();
  }
}

@Data
@AllArgsConstructor
class Employee
{
  private long id;
  private String name;
  private int age;
  private double salary;
}
```

Program output.

Output

```
false
```

3. Difference between *anyMatch()* vs *contains()*

Theoretically, there is no difference between `anyMatch()` and contains() when we want to check if an element exists in a `List`.

In some cases, parallelism **feature of Streams may bring an advantage for really large lists**, but we should not casually use the `Stream.parallel()` every time assuming that it may make things faster.

In fact, invoking `parallel()` may bring down the performance for small streams.

4. Conclusion

The `anyMatch()` method can be useful in certain cases where we need to check if there is at least one element in the stream.

The shorter version `list.contains()` also does the same thing and can be used instead.

Happy Learning !!

Sourcecode on Github

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 peek()
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 …

Promoted by usertesting.com
Sponsored

A message from our sponsor

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