HowToDoInJava

Java Stream filter()

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

Learn to use **Stream.filter(Predicate)** method to traverse all the elements and filter all items which match a given condition through `Predicate` argument.

Table Of Contents

1. Stream filter() Method

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

Syntax

```
Stream<T> filter(Predicate<? super T> condition)
```

`Predicate` is a functional interface and represents the condition to filter out the non-matching elements from the Stream.

- `filter()` is a **intermediate** `Stream` operation.
- It returns a `Stream` consisting of the elements of the given stream that match the given predicate.
- The `filter()` argument should be **stateless predicate** which is applied to each element in the stream to determine if it should be included or not.
- `Predicate` is a functional interface. So, we can also pass lambda expression also.
- It returns a new `Stream` so we can use other operations applicable to any stream.

2. Java Stream filter() Examples

Recommended Reading

The given examples use the predicates to write filter conditions. Read Java Predicates to learn how to write predicates for the different requirements.

2.1. Filtering a Stream using Lambda Expression

In this example, we are iterating over a stream of numbers. We will *find all even numbers from the Stream* and print them into Console.

The inline predicate '`n -> n % 2 == 0`' is a lambda expression.

Find even numbers in stream

```
import java.util.Arrays;
import java.util.List;
```

```
public class Main
{
    public static void main(String[] args)
    {
        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9,

        list.stream()
            .filter(n -> n % 2 == 0)
            .forEach(System.out::println);
    }
}
```

Program output.

Output

```
2
4
6
8
10
```

2.2. Filtering a Stream using Custom Predicate

This example is a rewrite of the first example. It uses `Predicate` class in place of the lambda expression, though both are the same things.

Note that we can write any condition inside the predicate, to match the business requirements.

Find even numbers from Stream using Predicateream

```
import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;

public class Main
{
    public static void main(String[] args)
    {
        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9,

        Predicate<Integer> condition = new Predicate<Integer>()
        {
            @Override
            public boolean test(Integer n) {
                if (n % 2 == 0) {
                    return true;
                }
                return false;
            }
        };

        list.stream().filter(condition).forEach(System.out::println);
    }
}
```

Program output.

Output

```
246810
```

2.3. Filtering and Collecting into a List

We can use the **collect(Collectors.toList())** method to collect the Stream of filtered elements into a `List`.

This example is again a rewrite of the first example. Here, we are collecting the even numbers into a `List` rather than printing them to the Console.

Collecting filtered items into a List

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class Main
{
    public static void main(String[] args)
    {
        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9,

        List<Integer> evenNumbers = list.stream()
                    .filter(n -> n % 2 == 0)
                    .collect(Collectors.toList());

        System.out.println(evenNumbers);
    }
}
```

Program output.

Output

```
[2, 4, 6, 8, 10]
```

2.4. Stream filter() and map() Example

We can use the map() method to collect the stream elements and then convert each number to its square before collecting it to the `List`.

Find even numbers in stream and collect the squares

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class Main
{
    public static void main(String[] args)
    {
        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9,

        List<Integer> evenNumbers = list.stream()
                    .filter(n -> n % 2 == 0)
                    .map(n -> n * n)
                    .collect(Collectors.toList());

        System.out.println(evenNumbers);
    }
}
```

Program output.

Output

```
[4, 16, 36, 64, 100]
```

2.5. Stream filter() with a method throwing Exception

The methods used in predicates return a `boolean` value. These methods generally do simple value comparisons and do not throw any `Exception`.

But, sometimes, we may need to deal with such methods which throw an exception and this method has to be used in the Predicate.

To solve this problem, we must use try-catch statement to catch the checked exception. Then we have a choice to either handle the exception or rethrow as an *unchecked exception*.

Given below is a code example to handle *checked exceptions* thrown from a method that has been used in a Predicate.

```
List<Integer> evenNumbers = list.stream()
    .filter(a -> {
            try {
                return a.someMethodThrowingCheckedException();
            } catch (IOException e) {
                throw new UncheckedException(e);
            }
        })
    .collect(Collectors.toList());
```

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 count() Matches with filter()
2. Java CORS Filter Example
3. Securing REST APIs with RESTEasy Filter
4. Jersey Logging Request and Response Entities using Filter

5. JavaScript Array filter()

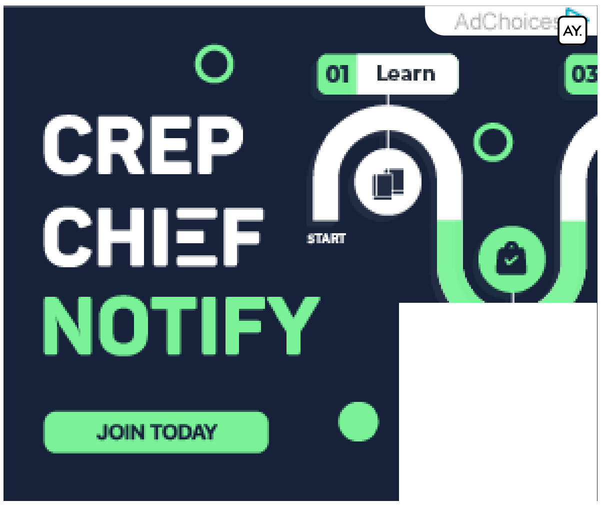
6. Java Stream reuse – traverse stream multiple times?

7. Java Stream min()

8. Java Stream skip()

9. Java Stream findFirst()

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 !!*

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

Copyright © 2022 · Hosted on Cloudways · Sitemap