HowToDoInJava

Java Stream distinct()

Last Updated: March 14, 2022 By: Lokesh Gupta Java 8 Java Stream Basics, Java Stream Methods

Learn to use `Stream.distinct()` method for *finding the distinct elements by field from a Stream*. To find the items that are distinct by multiple fields, we can use the information on the linked post.

Quick Reference for Using distinct() Method

```
List<String> distinctElements = list.stream()
                                 .distinct()
                                 .collect(Collectors.toList())
```

Table Of Contents

1. Stream `distinct()` Method

The `distict()` method is one such **stateful intermediate operation** that uses the state from previously seen elements from the Stream while processing the new items.

Method Syntax

```
Stream<T> distinct()
```

- The `distinct()` method **returns a new stream consisting of the distinct elements** from the given stream. For checking the equality of the stream elements, the `equals()` method is used.
- The `distinct()` method **guarantees the ordering for the streams backed by an ordered collection.** For ordered streams, the element appearing first in the encounter order is preserved.
  For *unordered streams*, no stability guarantees are made.

2. Finding Distinct Strings or Primitives

It is easy finding distinct items from a list of simple types such as `String` and wrapper classes. These classes implement the required `equals()` method, which compares the value stored in it.

In the given example, we have `List` of strings and we want to **find all distinct strings** from the `List`.

We will use Stream to iterate over all the `String` elements and collect the distinct `String` elements into another `List` using `Stream.collect()` terminal operation.

Find all distinct strings

```
Collection<String> list = Arrays.asList("A", "B", "C", "D", "A", "B",

// Get collection without duplicate i.e. distinct only
List<String> distinctElements = list.stream()
                        .distinct()
```

```
                                            .collect(Collectors.toList());

//Let's verify distinct elements
System.out.println(distinctElements);
```

Program output:

```
[A, B, C, D]
```

3. Find Distinct Objects By Field

In real-world applications, we will be dealing with a stream of custom classes or complex types (representing some system entity).

By default, all Java objects inherit the `equals()` method from `Object` class. *The default equals() method compares the references for checking the equality of two instances.* So, it is **highly recommended to override the *equals()* method and define custom logic for object equality**.

If we do not override the `equals()` method in our custom type, then we may see strange behavior while finding the distinct elements from a Stream.

3.1. Overidding `equals()` Method

Let's create a *Person* class for our example. It has three fields: `id`, `fname` and `lname`. Two persons are equal if their `ids` are the same.

Do not forget to override the `equals()` method otherwise, the object equality will not work as expected.

Two Person are equal if their id is same

```
public record Person(Integer id, String fname, String lname) {

  @Override
  public boolean equals(final Object obj) {
    if (this == obj) {
      return true;
    }
    if (obj == null) {
      return false;
    }
    if (getClass() != obj.getClass()) {
      return false;
    }
    Person other = (Person) obj;
    return Objects.equals(id, other.id);
  }
}
```

3.2. Demo

Let's test the code. We will add a few duplicate person records in the `List`. Then we will use the `Stream.distinct()` method to find all instances of Person class with unique `id`.

Java program to find distinct persons by id

```
Person lokeshOne = new Person(1, "Lokesh", "Gupta");
Person lokeshTwo = new Person(1, "Lokesh", "Gupta");
Person lokeshThree = new Person(1, "Lokesh", "Gupta");
Person brianOne = new Person(2, "Brian", "Clooney");
Person brianTwo = new Person(2, "Brian", "Clooney");
Person alex = new Person(3, "Alex", "Kolen");

//Add some random persons
Collection<Person> list = Arrays.asList(alex,
                                        brianOne,
                                        brianTwo,
                                        lokeshOne,
                                        lokeshTwo,
                                        lokeshThree);

// Get distinct people by id
List<Person> distinctElements = list.stream()
            .distinct()
            .collect( Collectors.toList() );

// Let's verify distinct people
System.out.println( distinctElements );
```

Output

```
[
Person [id=1, fname=Lokesh, lname=Gupta],
Person [id=2, fname=Brian, lname=Clooney],
Person [id=3, fname=Alex, lname=Kolen]
]
```

4. Find Distinct Objects by Complex Keys

We may not be getting the distinct items always based on the natural equality rules. Sometimes, business wants to find distinct items based on custom logic.

For example, we may need to find all people who may have any `id` but their full name is the same. In this case, we must check the equality based on `Person` class's `fname` and `lname` fields.

Java does not have any native API for finding the distinct objects while comparing the objects using a provided user function. So we will create our own utility function and then use it.

4.1. distinctByKey()

The `distinctByKey()` function uses a `ConcurrentHashMap` instance to find out if there is an existing key with the same value – where the key is obtained from a function reference.

The parameter to this function is a lambda expression that is used to generate the map key for making the comparison.

Utility function to find distinct by class field

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

We can pass any field getter method as a method argument which will cause the field value to act as the key to the map.

4.2. Demo

Check how we are using `distinctByKey(p -> p.getFname() + " " + p.getLname())` in the `filter()` method.

Creating Complex Key by Appending first name and last name

```
Person lokeshOne = new Person(1, "Lokesh", "Gupta");
Person lokeshTwo = new Person(2, "Lokesh", "Gupta");
Person lokeshThree = new Person(3, "Lokesh", "Gupta");
Person brianOne = new Person(4, "Brian", "Clooney");
Person brianTwo = new Person(5, "Brian", "Clooney");
Person alex = new Person(6, "Alex", "Kolen");

//Add some random persons
Collection<Person> list = Arrays.asList(alex,
                                        brianOne,
                                          brianTwo,
                                          lokeshOne,
                                          lokeshTwo,
                                          lokeshThree);

// Get distinct objects by key
List<Person> distinctElements = list.stream()
            .filter( distinctByKey(p -> p.getFname() + " " + p.getLna
            .collect( Collectors.toList() );

// Again verify distinct people
System.out.println( distinctElements );
```

Program Output:

Output

```
[
Person [id=1, fname=Lokesh, lname=Gupta],
Person [id=4, fname=Brian, lname=Clooney],
Person [id=6, fname=Alex, lname=Kolen]
]
```

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. Getting Distinct Stream Items by Comparing Multiple Fields
2. Java Stream reuse – traverse stream multiple times?
3. Java Stream count() Matches with filter()
4. Java Stream filter()
5. Java Stream map()
6. Java Stream flatMap()
7. Java Stream peek()
8. Java Stream anyMatch()
9. Java Stream allMatch()
0. Java Stream toArray()

3 thoughts on "Java Stream distinct()"

Bala

September 27, 2019 at 3:24 pm

thanks for the article, but is is any way for distinct by all fields of an on object/model/pojo.

Let say my method should work very generic, have person, employee model classes and each model class has 20 fields and need to find the distinct values of all fields.

Reply

PA

September 21, 2018 at 8:38 am

How to filter based on distinct fname and lname from the above example?

Reply

Neel Armstrong

August 17, 2018 at 12:09 pm

Hi Lokesh,

Can you please explain this line

return t -> map.putIfAbsent(keyExtractor.apply(t), Boolean.TRUE) == null;

from

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

Does this mean that the above static method will return an object of Predicate and the prdicate.test will have the body as return t -> map.putIfAbsent(keyExtractor.apply(t), Boolean.TRUE) == null;

If yes , how we can pass the map object here, might be i am not getting it , please explain , how the process will work and how many times the predicate will get a hit.

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