HowToDoInJava

Java CopyOnWriteArrayList class

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

Java CopyOnWriteArrayList is a thread-safe variant of **ArrayList** in which all mutative operations (add, set, and so on) are implemented by making a fresh copy of the underlying array.

It's **immutable snapshot** style iterator method uses a reference to the state of the array at the point that the iterator was created. This helps in usecases when traversal operations vastly outnumber list update operations and we do not want to synchronize the traversals and still want thread safety while updating the list.

Table of Contents

1. CopyOnWriteArrayList Hierarchy

The `CopyOnWriteArrayList` class implements following interfaces – List, RandomAccess, Cloneable and Serializable.

```
CopyOnWriteArrayList.java

public class CopyOnWriteArrayList<E>
    implements  List<E>,
          RandomAccess,
          Cloneable,
          Serializable

{
  private transient volatile Object[] array;

  //implementation
}
```

2. CopyOnWriteArrayList Features

The important things to learn about **Java CopyOnWriteArrayList** class are:

- CopyOnWriteArrayList class implement `List` and `RandomAccess` interfaces and thus provide all functionalities available in ArrayList class.
- Using CopyOnWriteArrayList is costly for update operations, because each mutation creates a cloned copy of underlying array and add/update element to it.
- It is thread-safe version of ArrayList. Each thread accessing the list sees its own version of snapshot of backing array created while initializing the iterator for this list.
- Because it gets snapshot of underlying array while creating iterator, it **does not throw ConcurrentModificationException**.
- Mutation operations on iterators (remove, set, and add) are not supported. These methods throw `UnsupportedOperationException`.
- CopyOnWriteArrayList is a concurrent replacement for a **synchronized List** and offers better concurrency when iterations outnumber mutations.
- It allows duplicate elements and heterogeneous Objects (use generics to get compile time errors).
- Because it creates a new copy of array everytime iterator is created, **performance is slower** than ArrayList.

3. CopyOnWriteArrayList Example

Java program to show how iterators created at different times sees through snapshot version of list in CopyOnWriteArrayList. In given example, we first created list and **itr1** when list had elements (1,2,3).

Then we added one more element to list and again created an iterator **itr2**.

Finally we verified the elements in both iterators.

```
CopyOnWriteArrayList Example

CopyOnWriteArrayList<Integer> list = new CopyOnWriteArrayList<>(new Integer[] {1,2,3});

System.out.println(list); //[1, 2, 3]

//Get iterator 1
Iterator<Integer> itr1 = list.iterator();

//Add one element and verify list is updated
list.add(4);

System.out.println(list); //[1, 2, 3, 4]

//Get iterator 2
Iterator<Integer> itr2 = list.iterator();

System.out.println("====Verify Iterator 1 content====");

itr1.forEachRemaining(System.out :: println); //1,2,3

System.out.println("====Verify Iterator 2 content====");

itr2.forEachRemaining(System.out :: println); //1,2,3,4
```

Program Output.

```
Console

[1, 2, 3]
[1, 2, 3, 4]
====Verify Iterator 1 content====
1
2
3
====Verify Iterator 2 content====
1
2
3
4
```

4. CopyOnWriteArrayList Constructors

- **CopyOnWriteArrayList()** : Creates an empty list.
- **CopyOnWriteArrayList(Collection c)** : Creates a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
- **CopyOnWriteArrayList(object[] array)** : Creates a list holding a copy of the given array.

5. CopyOnWriteArrayList Methods

CopyOnWriteArrayList class all the methods which are supported in ArrayList class. The behavior is different only in case of iterators (**snapshot iterator**) AND new backing array created during mutations in the list.

Additionally it provides few methods which are additional to this class.

- **boolean addIfAbsent(object o)** : Append the element if not present.
- **int addAllAbsent(Collection c)** : Appends all of the elements in the specified collection that are not already contained in this list, to the end of this list, in the order that they are returned by the specified collection's iterator.

For all other methods supported, visit ArrayList methods section.

6. Java CopyOnWriteArrayList Usecases

We can prefer to use CopyOnWriteArrayList over normal ArrayList in following cases:

1. When list is to be used in concurrent environemnt.
2. Iterations outnumber the mutation operations.
3. Iterators must have snapshot version of list at the time when they were created.
4. We don't want to synchronize the thread access programatically.

7. Java CopyOnWriteArrayList Performance

Due to added step of creating a new backing array everytime the list is updated, it performs worse than ArrayList. There is no performance overhead on read operations and both classes perform same.

8. Conclusion

In this Java Collection tutorial, we learned to use **CopyOnWriteArrayList** class, it's constructors, methods and usecases. We learned the **CopyOnWriteArrayList internal working in java** as well as **CopyOnWriteArrayList vs synchronized arraylist**.

We gone through **Java CopyOnWriteArrayList example program** to demo how snapshot iterators works.

Drop me your questions in comments.

Happy Learning !!

Reference:

CopyOnWriteArrayList Java Docs

Recommended Reading:

1. [Solved]: javax.xml.bind.JAXBException: class java.util.ArrayList nor any of its super class is known to this context
2. Java TransferQueue – Java LinkedTransferQueue class
3. Java TreeMap class
4. Java PriorityBlockingQueue class
5. Java ArrayBlockingQueue class
6. Java CopyOnWriteArraySet class
7. Java LinkedHashMap class
8. Java Hashtable class
9. Java HashSet class
0. Java LinkedList class

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 ...

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