How LinkedHashSet Works Internally In Java

I

have already discussed how HashSet works internally in Java . In this post we will understand how HashSet subclass i.e LinkedHashSet works internally in java. Just like HashSet internally uses HashMap to add element to its object similarly LinkedHashSet internally uses LinkedHashMap to add element to its object . Internal working of LinkedHashSet includes two basic questions ,first, How LinkedHashSet maintains Unique Elements ?, second , How LinkedHashSet maintains Insertion Order ? . We will find out the answers of the above questions in this post.

What is LinkedHashSet ?

WHATS HOT

- Difference between Arraylist and Vector : Core Java Interview Collection Question
- Best Books for Learning Java

According to Oracle docs ,

LinkedHashSet is the Hashtable and linked list implementation of the Set interface with predictable iteration order. The linked list defines the **iteration ordering, which is the order in which elements were inserted into the set.** Insertion order is not affected if an element is re-inserted into the set.

Read Also : How TreeMap works internally in java

Why we need LinkedHashSet when we already have the HashSet and TreeSet ?

HashSet and TreeSet classes were added in jdk 1.2 while LinkedHashSet was added to the jdk in java version 1.4

HashSet provides constant time performance for basic operations like (add, remove and contains) method but **elements are in chaotic ordering i.e unordered.**

In TreeSet elements are naturally sorted but **there is increased cost associated with it .**

So , LinkedHashSet is added in jdk 1.4 to maintain ordering of the elements without incurring increased cost.

How LinkedHashSet Works Internally in Java ?

- Amazon Interview Question : First Non repeated character in String
- Count total number of times each alphabet appears in the string java program code with example
- Java 8 new features : Lambda expressions , optional class , Defender methods with examples

Before understanding how LinkedHashSet works internally in java in detail, we need to understand two terms *initial capacity* and *load factor* .

What is Initial capacity and load factor?

The ***capacity*** is the number of buckets(used to store key and value) in the Hash table , and the ***initial capacity*** is simply the capacity at the time Hash table is created.

The ***load factor*** is a measure of how full the Hash table is allowed to get before its capacity is automatically increased.

Constructor of LinkedHashSet depends on above two parameters *initial capacity* and *load factor* .

There are **four constructors present in the LinkedHashSet class .**

All constructors have the same below pattern :

```
// Constructor 1
 public LinkedHashSet (int initialCa
{
   super(initialCapacity , loadFactor ,
}
```

Note : **If initialCapacity or loadFactor parameter value is missing during LinkedHashSet object creation , then default value of initialCapacity or loadFactor is used .**
Default value for initialCapacity : 16 ,
Default value for loadFactor : 0.75f

POPULAR POSTS

- Java Interview Questions
- Top 50 Java Collections Interview Questions and Answers
- What Makes You Stand out from the Crowd
- Threads , Lifecycle Explained with Example
-

For example,

check the **below overloaded constructor , loadFactor is missing** in the LinkedHashSet constructor argument. So during super() call , we use the default value of the loadFactor(0.75f).

```
// Constructor 2
 public LinkedHashSet (int initialCa
{
   super(initialCapacity , 0.75f , true);
}
```

check the **below overloaded constructor , initialCapacity and loadFactor both are missing** in the LinkedHashSet constructor argument. So during super() call , we use the default value of both initialCapacity(16) and loadFactor(0.75f).

```
// Constructor 3
 public LinkedHashSet ()
{
   super(16 , 0.75f , true);
}
```

below is the last overloaded constructor which uses Collection in the LinkedHashSet constructor argument. So during super() call , we use the default value of loadFactor(0.75f).

```
// Constructor 4
 public LinkedHashSet (Collection c
{
   super(Math.max(2*c.size() ,11) , 0.
}
```

Count number of words in the String with Example : Java Program Code

Note : Since LinkedHashSet extends HashSet class.
Above all the 4 constructors are calling the super class (i.e HashSet) constructor , given below

```
        public HashSet (int
initialCapacity , float loadFactor
, boolean dummy)
   {
 1.       map = new LinkedHashMap<>(ir
   }
```

In the above HashSet constructor , there are two main points to notice :

a. We are using extra boolean parameter *dummy* . It is used to distinguish other int, float constructors present in the HashSet class.

b. Internally it is creating a LinkedHashMap object passing the initialCapacity and loadFactor as parameters.

How LinkedHashSet Maintains Unique Elements ?

```
public class HashSet<E>
extends AbstractSet<E>
implements Set<E>, Cloneable, java.ic

{
   private transient HashMap<E,Objec
```

```
    // Dummy value to associate with an
    private static final Object PRESENT

    public HashSet(int initialCapacity ,
        map = new LinkedHashMap<>(in
    }
    // SOME CODE ,i.e Other methods in

    public boolean add(E e) {
        return map.put(e, PRESENT)==n
    }

    // SOME CODE ,i.e Other methods in
}
```

So , we are achieving uniqueness in LinkedHashSet,internally in java through LinkedHashMap . Whenever you create an object of LinkedHashSet it will indirectly create an object of LinkedHashMap as you can see in the italic lines of HashSet constructor.

Read Also : How LinkedHashMap works Internally in Java

As we know in LinkedHashMap each key is unique . So what we do in the LinkedHashSet is that we pass the argument in the add(Elemene E) that is E as a key in the LinkedHashMap . Now we need to associate some value to the key , so what Java apis developer did is to pass the Dummy value that is (new Object ()) which is referred by Object reference PRESENT .

So , actually when you are adding a line in LinkedHashSet like
linkedhashset.add(5) what java does internally is that it will put that element E here 5 as a key in the LinkedHashMap(created during LinkedHashSet object creation) and some dummy value that is Object's object is passed as a value to the key .

Since LinkedHashMap put(Key k , Value v) method does not have its own implementation . LinkedHashMap put(Key k , Value v) method uses HashMap put(Key k , Value v) method.

Now if you see the code of the HashMap put(Key k,Value v) method , you will find something like this

```
 public V put(K key, V value) {
//Some code
}
```

The main point to notice in above code is that put (key,value) will return

1. null , if key is unique and added to the map
2. Old Value of the key , if key is duplicate

So , in LinkedHashSet add() method , we check the return value of map.put(key,value) method with null value
i.e.

```
   public boolean add(E e) {
          return map.put(e,
PRESENT)==null;
      }
```

So , if map.put(key,value) returns null ,then
map.put(e, PRESENT)==null will return true and element is added to the LinkedHashSet.

So , if map.put(key,value) returns old value of the key ,then
map.put(e, PRESENT)==null will return false and element is not added to the LinkedHashSet .

How LinkedHashSet Maintains Insertion Order ?

LinkedHashSet differs from HashSet because it maintains the insertion order . According to LinkedHashSet Oracle docs ,

> *LinkedHashSet implementation differs from HashSet in that it maintains a doubly-linked list running through all of its entries*

LinkedHashSet internally uses LinkedHashMap to add elements to its object.

What is Entry object?

LinkedHashMap consists of a static inner class named as Entry . **Each object of Entry represents a key,value pair**. The key K in the Entry object is the value which needs to be added to the LinkedHashSet object. The value V in the Entry object is any dummy object

called PRESENT.

Insertion Order of the LinkedHashMap is maintained by two Entry fields head and tail , which stores the head and tail of the doubly linked list.

```
 transient LinkedHashMap.Entry
head;
 transient LinkedHashMap.Entry  tail;
```

For double linked list we need to maintain the previous and next Entry objects for each Entry object .
Entry fields ***before*** and ***after*** are used **to store the references to the previous and next Entry objects** .

```
static class Entry extends HashMap.

    Entry before, after ;
    Entry( int hash , K key , V value , N
     super(hash,key,value,next);
    }
}
```

Please mention in the comments in case if you have any questions regarding how LinkedHashSet works internally in Java

About The Author