How ConcurrentHashMap Internally Works In Java With Example

How ConcurrentHashMap works or internal implementation of ConcurrentHashMap is one of the most popular java interview questions under the category concurrency. We have already discussed the other popular java interview questions like internal working of HashSet or How HashMap works in java .

ConcurrentHashMap utilizes the same principles of HashMap, but is designed primarily for a multi-threaded application and hence it does not require explicit

WHATS HOT

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

synchronization. The only thread safe collection objects were Hashtable and synchronized Map prior to JDK 5.

Before learning How ConcurrentHashMap works in Java , we need to look at why ConcurrentHashMap is added to the Java SDK.

Interviewer : Why we need ConcurrentHashMap when we already had Hashtable ?

Hashtable provides concurrent access to the Map.Entries objects by locking the entire map to perform any sort of operation (update,delete,read,create). Suppose we have a web application , the overhead created by Hashtable (locking the entire map) can be ignored under normal load. But under heavy load , the overhead of locking the entire map may prove fatal and may lead to delay response time and overtaxing of the server.

This is where ConcurrentHashMap comes to rescue. According to ConcurrentHashMap Oracle docs, ConcurrentHashMap class is fully interoperable with Hashtable in programs that rely on its thread safety but not on its synchronization details. So the main purpose of this class is to provide the same functionality as of Hashtable but with a performance comparable to HashMap.

ConcurrentHashMap achieves this by a simple tweak. So this leads to our main question

- 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

How ConcurrentHashMap works in Java

According to ConcurrentHashMap Oracle docs,

The constructor of ConcurrentHashMap looks like this :

public ConcurrentHashMap *(int initialCapacity, float loadFactor, int concurrencyLevel)*

So the above line creates a new, empty map with the specified initial capacity, load factor and concurrency level. where,
Important Parameters to consider from ConcurrentHashMap Constructor:

initialCapacity - the initial capacity. The implementation performs *internal sizing to accommodate this many elements.*
concurrencyLevel - the estimated number of concurrently updating threads. The implementation performs *internal sizing to try to accommodate this many threads.*

In the ConcurrentHashMap Api , you will find the following constants.

static final int DEFAULT_INITIAL_CAPACITY = 16;
static final int DEFAULT_CONCURRENCY_LEVEL = 16;

initial capacity parameter and concurrency level parameters of ConcurrentHashMap constructor (or Object) are set to 16 by default.

POPULAR POSTS

Thus, instead of a map wide lock, ConcurrentHashMap maintains a list of 16 locks by default (number of locks equal to the initial capacity , which is by default 16) each of which is used to lock on a single bucket of the Map.This indicates that 16 threads (number of threads equal to the concurrency level , which is by default 16) can modify the collection at the same time , given ,each thread works on different bucket. So unlike hashtable, we perform any sort of operation (update ,delete ,read ,create) without locking on entire map in ConcurrentHashMap.

Retrieval operations (including `get`) generally do not block, so may overlap with update operations (including `put` and `remove`). Retrievals reflect the results of the most recently *completed* update operations holding upon their onset.

The allowed concurrency among update operations is guided by the optional concurrencyLevel constructor argument (default 16), which is used as a hint for internal sizing. The table is internally partitioned to try to permit the indicated number of concurrent updates without contention. Because placement in hash tables is essentially random, the actual concurrency will vary. Ideally, you should choose a value to accommodate as many threads as will ever concurrently modify the table. Using a significantly higher value than you need can waste space and time, and a significantly lower value can lead to thread contention

Interviewer : Can two threads update the ConcurrentHashMap simultaneously ?

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

Yes it is possible that two threads can simultaneously write on the ConcurrentHashMap.
ConcurrentHashMap default implementation allows 16 threads to read and write in parallel.
But in the worst case scenario , when two objects lie in the same segment or same partition, then parallel write would not be possible.

Interviewer : Why ConcurrentHashMap does not allow null keys and null values ?

According to the author of the ConcurrentHashMap (Doug lea himself)

The main reason that nulls aren't allowed in ConcurrentMaps (ConcurrentHashMaps, ConcurrentSkipListMaps) is that ambiguities that may be just barely tolerable in non-concurrent maps can't be accommodated. The main one is that if map.get(key) returns null, you can't detect whether the key explicitly maps to null vs the key isn't mapped. In a non-concurrent map, you can check this via map.contains(key), but in a concurrent one, the map might have changed between
calls.

In simple words,

The code is like this :

```
if (map.containsKey(k)) {
   return map.get(k);
} else {
   throw new KeyNotPresentExceptio
```

```
}
```

It might be possible that key k might be deleted in between the get(k) and containsKey(k) calls. As a result , the code will return null as opposed to KeyNotPresentException (Expected Result if key is not present).

Interviewer : What is the difference between HashMap and ConcurrentHashMap?

The HashMap was not thread safe and therefore could not be utilized in multi-threaded applications. The ConcurrentHashMap was introduced to overcome this shortcoming and also as an alternative to using HashTable and synchronized Maps for greater performance and uses the standard Hashing algorithms to generate hash code for storing the key value pairs.For more difference between HashMap and ConcurrentHashMap check this popular interview question HashMap vs ConcurrentHashMap in java.

Interviewer : Can multiple threads read from the Hashtable concurrently ?

No multiple threads can not read simultaneously from Hashtable. Reason, the get() method of Hashtable is synchronized. As a result , at a time only one thread can access the get() method .
It is possible to achieve full concurrency for reads (all the threads read at the same time) in ConcurrentHashMap by

using volatile keyword.

Interviewer: Does ConcurrentHashMap Iterator behaves like fail fast iterator or fail safe Iterator?

ConcurrentHashMap iterator behaves like fail safe iterator. It will not throw ConcurrentModificationException . We have already discussed Fail Fast Iterator vs Fail Safe Iterator.

Interviewer : Why does Java provide default value of partition count as 16 instead of very high value ?

According to Java docs ,

Ideally, you should choose a value to accommodate as many threads as will ever concurrently modify the table. Using a significantly higher value than you need can waste space and time, and a significantly lower value can lead to thread contention.

Interviewer : Can you write the simple example which proves

ConcurrentHashMap class behaves like fail safe iterator?

ConcurrentHashMap Example :

```
import java.util.concurrent.Concurre
import java.util.Iterator;


public class ConcurrentHashMapExa
{


   public static void main(String[] args
   {
      ConcurrentHashMap<String,String
      premiumPhone.put("Apple", "iPhor
      premiumPhone.put("HTC", "HTC o
      premiumPhone.put("Samsung","S6

      Iterator iterator = premiumPhone.k

      while (iterator.hasNext())
      {
         System.out.println(premiumPho
         premiumPhone.put("Sony", "Xpe
      }

   }

}
```

Output :

```
S6
HTC one
iPhone6
```

Please mention in the comments in case you have any doubts regarding the internal implementation of ConcurrentHashMap.

About The Author

Subham Mittal has worked in Oracle for 3 years .

For more java articles ,Click here to Subscribe JavaHungry

You might also like

- How to Convert Math Number to Equivalent Readable Word in Java : Code with Example | Java Hungry
- SCJP | Java Hungry
- Java 7 features revisited | Java Hungry
- Merge Sort : Java Sorting Program Code along with Example | Java Hungry

Recommended by