

Difference between ConcurrentHashMap, Hashtable and Synchronized Map in Java

ConcurrentHashMap vs Hashtable vs Synchronized Map



Though all three collection classes are thread-safe and can be used in multi-threaded, concurrent Java application, there is a significant difference between them, which arise from the fact that how they achieve their thread-safety. Hashtable is a legacy class from JDK 1.1 itself, which uses synchronized methods to achieve thread safety. All methods of Hashtable are synchronized which makes them quite slow due to contention if a number of thread increases. Synchronized Map is also not very different than Hashtable and provides similar performance in concurrent Java programs. The only difference between Hashtable and Synchronized Map is that later is not a legacy and you can wrap any Map to create it's synchronized version by using `Collections.synchronizedMap()` method.

On the other hand, `ConcurrentHashMap` is specially designed for concurrent use i.e. more than one thread. By default it simultaneously allows 16 threads to read and write from Map without any external synchronization. It is also very scalable because of stripped locking technique used in the [internal implementation of ConcurrentHashMap](#) class.

Unlike `Hashtable` and `Synchronized Map`, it never locks whole Map, instead, it divides the map into segments and locking is done on those. Though it performs better if a number of reader threads are greater than the number of writer threads.

To be frank, Collections classes are the heart of Java API though I feel using them judiciously is an art. It's my personal experience where I have improved the performance of Java application by using `ArrayList` where legacy codes were unnecessarily using `Vector` etc. Prior Java 5, One of the major drawback of Java Collection framework was a lack of scalability.

In multi-threaded Java application synchronized collection classes like `Hashtable` and `Vector` quickly becomes the bottleneck; to address scalability JDK 1.5 introduces some good concurrent collections which are highly efficient for high volume, low latency system electronic trading systems In general those are the backbone for Concurrent fast access to stored data.

In this tutorial, we will look on `ConcurrentHashMap`, `Hashtable`, `HashMap` and `synchronized Map` and see the difference between `ConcurrentHashMap` and `Hashtable` and `synchronized Map` in Java. We have already discussed some key [difference between HashMap and Hashtable in Java](#) in this blog and those will also help you to answer this question during interviews.

Why need ConcurrentHashMap and CopyOnWriteArrayList

The synchronized collections classes, Hashtable, and Vector, and the synchronized wrapper classes, `Collections.synchronizedMap()` and `Collections.synchronizedList()`, provide a basic conditionally thread-safe implementation of Map and List. However, several factors make them unsuitable for use in highly concurrent applications, for example, their single collection-wide lock is an impediment to scalability and it often becomes necessary to lock a collection for a considerable time during iteration to prevent [ConcurrentModificationException](#).

`ConcurrentHashMap` and `CopyOnWriteArrayList` implementations provide much higher concurrency while preserving thread safety, with some minor compromises in their promises to callers. `ConcurrentHashMap` and `CopyOnWriteArrayList` are not necessarily useful everywhere you might use `HashMap` or `ArrayList`, but are designed to optimize specific common situations. Many concurrent applications will benefit from their use.

Difference between ConcurrentHashMap and Hashtable

So what is the difference between Hashtable and ConcurrentHashMap, both can be used in the multithreaded environment but once the size of Hashtable becomes considerable large performance degrade because for iteration it has to be locked for a longer duration.

Since ConcurrentHashMap introduced the concept of segmentation, how large it becomes only certain part of it get locked to provide thread safety so many other readers can still access map without waiting for iteration to complete.

In Summary, ConcurrentHashMap only locked certain portion of Map while Hashtable locks full map while doing iteration. This will be clearer by looking at this diagram which explains the internal working of ConcurrentHashMap in Java.