How ConcurrentHashMap works in Java?

and difference between a synchronized map and concurrentHashMap

JAVINPAUL
MAR 07, 2024 · PAID

4 2 Share

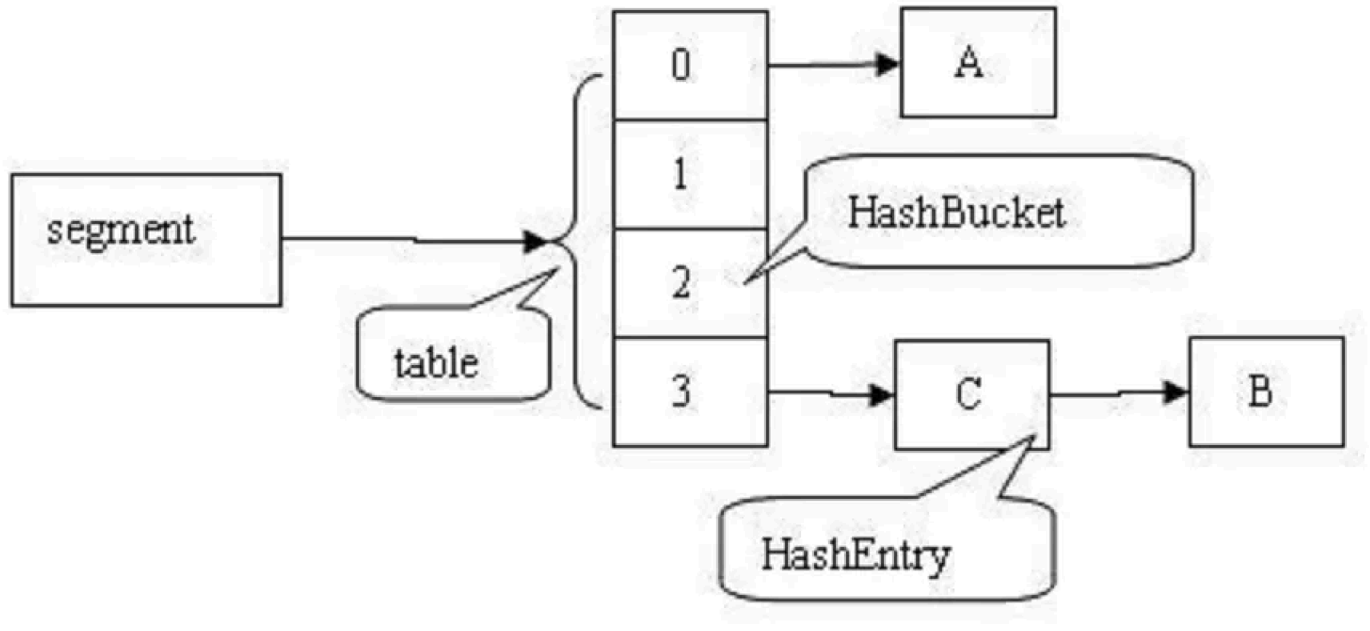
Hello guys, how `HashMap` work in Java, and How `ConcurrentHashMap` works in Java are two of the popular questions in Java interviews. I have seen it many times and even asked about it a couple of times.

Since a lot of you requested me to share core Java interview questions, I have decided to write at least one post about them every week and in this post, I will talk about How `ConcurrentHashMap` work and what is difference between a synchronized Map and `ConcurrentHashMap` in Java.

`ConcurrentHashMap` is a class in Java that provides a thread-safe implementation of the `Map` interface. It was introduced in Java 5 to address the need for a concurrent and scalable map data structure. The primary goal `ConcurrentHashMap` is to allow multiple threads to read and write to the map concurrently without the need for external synchronization.

Here are some key features and aspects of how `ConcurrentHashMap` works:

1. **Segmentation:** `ConcurrentHashMap` is divided into segments, and each segment acts as an independent hash table. The number of segments is determined by the `concurrencyLevel` specified during the creation of the `ConcurrentHashMap`. Each segment is essentially a separate hash table, and different segments can be operated on concurrently by different threads.
   Here is how a segment looks like:



2. **Bucketing:** Each segment is further divided into buckets, and each bucket holds a key-value pair. The number of buckets is typically a power of two. The hashing algorithm is used to determine the segment and bucket for each key, allowing multiple threads to operate on different segments simultaneously.
3. **Read Operations:** Read operations (e.g., `get`) can be performed without any locks, allowing for high concurrency. Each segment maintains its own internal consistency, so reading from one segment doesn't interfere with operations on other segments.
4. **Write Operations:** Write operations (e.g., `put`, `remove`) are performed with the help of a fine-grained lock mechanism. Instead of locking the entire map, only the relevant segment is locked during write operations. This reduces contention among threads and improves performance.
5. **Updates and Retrievals:** The `ConcurrentHashMap` provides atomic operations for updates and retrievals, such as `putIfAbsent`, `replace`, and `compute`. These operations are thread-safe and ensure that modifications are done atomically.
6. **Null Values:** Unlike some other map implementations, `ConcurrentHashMap` allows `null` values, both for keys and values.

Here's a simple example of using `ConcurrentHashMap`:

```
import java.util.concurrent.ConcurrentHashMap;

public class ConcurrentHashMapExample {
    public static void main(String[] args) {
        ConcurrentHashMap<String, Integer> concurrentMap = new
ConcurrentHashMap<>();
```

```
        concurrentMap.put("One", 1);
        concurrentMap.put("Two", 2);
        concurrentMap.put("Three", 3);

        // Thread-safe operations
        concurrentMap.putIfAbsent("Four", 4);
        concurrentMap.compute("Three", (key, value) -> value * 10);

        System.out.println(concurrentMap);
    }
}
```

One of the follow-up questions after this is what is the difference between a synchronized and a concurrent hashmap in Java as both provide thread safety

Difference between a synchronized Map and a ConcurrentHashMap in Java.

oth synchronized maps and `ConcurrentHashMap` in Java provide a level of thread safety for map operations, but they differ in their implementation and performance characteristics.

1. **Synchronization Mechanism:**
    - **Synchronized Map:** A synchronized map is typically created using the `Collections.synchronizedMap` method, which wraps an existing map with synchronization. All map operations are synchronized using a single lock. This means that only one thread can perform any operation on the map at a time, which can lead to contention and reduced concurrency.

   Example:

```
javaCopy code
```

1. - **ConcurrentHashMap:** `ConcurrentHashMap` uses a different approach. It is divided into segments, and each segment has its own lock. This allows multiple threads to operate on different segments concurrently, improving overall concurrency compared to a synchronized map.
2. **The granularity of Locking:**
    - **Synchronized Map:** All operations on a synchronized map are locked using a single lock, which means that only one thread can execute any operation on the map at a time.

- **ConcurrentHashMap:** Uses a finer-grained locking mechanism. Instead of a single lock for the entire map, it divides the map into segments, and each segment has its own lock. This allows multiple threads to operate on different segments simultaneously, reducing contention.

3. **Performance:**
   - **Synchronized Map:** While a synchronized map provides thread safety, it may lead to performance bottlenecks in a highly concurrent environment. The contention for the single lock can limit scalability.
   - **ConcurrentHashMap:** This is designed for improved concurrency and performance in scenarios with a high level of contention. By allowing multiple threads to operate on different segments concurrently, `ConcurrentHashMap` can provide better performance than a synchronized map in certain situations.
4. **Null Values:**
   - **Synchronized Map:** Generally allows `null` values for keys and values.
   - **ConcurrentHashMap:** Also allows `null` values for keys and values.

Here's a simple example illustrating the use of a synchronized map and `ConcurrentHashMap`:

```
// Synchronized Map Map<String, Integer> synchronizedMap =
Collections.synchronizedMap(new HashMap<>()); // ConcurrentHashMap
ConcurrentHashMap<String, Integer> concurrentHashMap = new
ConcurrentHashMap<>();
```

In summary, `ConcurrentHashMap` is often preferred in situations where high concurrency is required, and it can outperform a synchronized map in scenarios with frequent updates from multiple threads. However, the choice between them depends on the specific requirements and characteristics of your application.

That's all guys, happy learning and preparing for Java interviews. Let me know if you like this kind of post more in the comments.

And, if you are preparing for Java interviews you can also check my Java + Spring Interview + SQL Bundle on GUmroad, use code **friends20** to get a 20% discount also

4 Likes · 2 Restacks

Comments

Write a comment...