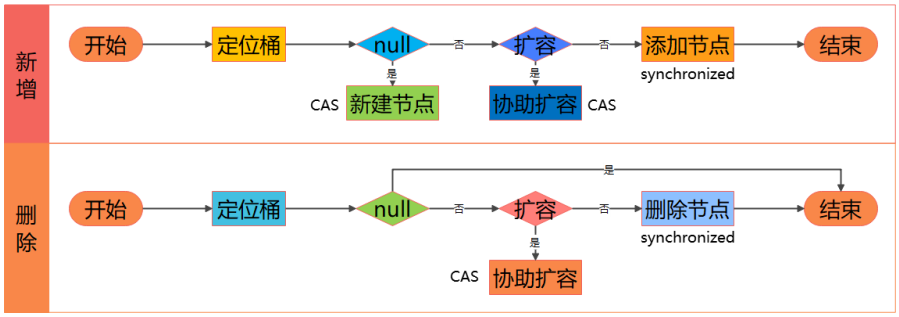
【第03话：自从有了JUC面试官特别喜欢问ConcurrentHashMap是如何保证线程安全的】

ConcurrentHashMap如何保证线程安全(月薪1万的回答示范)

1、底层结构是数组+链表+红黑树
2、使用CAS+synchronized保证线程安全
3、新增和删除过程类似



ConcurrentHashMap底层结构和HashMap比较类似。都是基于数组+链表+红黑树实现的。但是ConcurrentHashmap中多了乐观锁CAS和悲观锁synchronized。而不是全部都是synchronized，这也是ConcurrentHashMap既能保证线程，性能也比较好的原因。

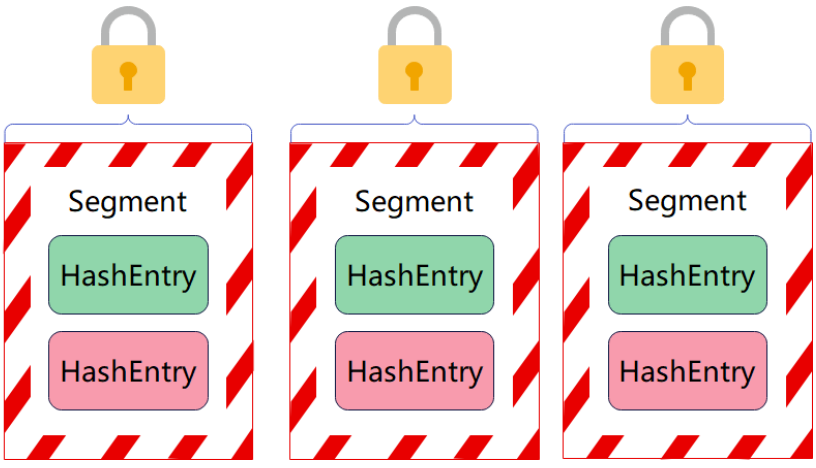
新增数据时，根据Key的Hash先定位桶，数组中每个脚标对应的元素就是一个桶。然后判断桶中是否为null，如果为null说明没有值，使用CAS保证线程安全来新增数据。如果桶中已经有值，会判断数组是否正在扩容，如果正在扩容，使用CAS来协助扩容。如果没有在扩容，使用synchronized保证线程安全，把节点加入到链表或红黑树中。和HashMap相同，如果链表长度大于等于8会转换为红黑树。这样新增就结束了。

删除数据流程时和新增数据类似。首先根据要删除Key定位桶。如果桶中数据为null，说明Key不存在，删除结束。如果如果正在扩容，使用CAS保证线程安全，协助扩容。如果没有在扩容。使用synchronized保证线程安全，删除节点，到此删除结束。

这就是ConcurrentHashMap如何保证线程安全月薪1万的回答示范

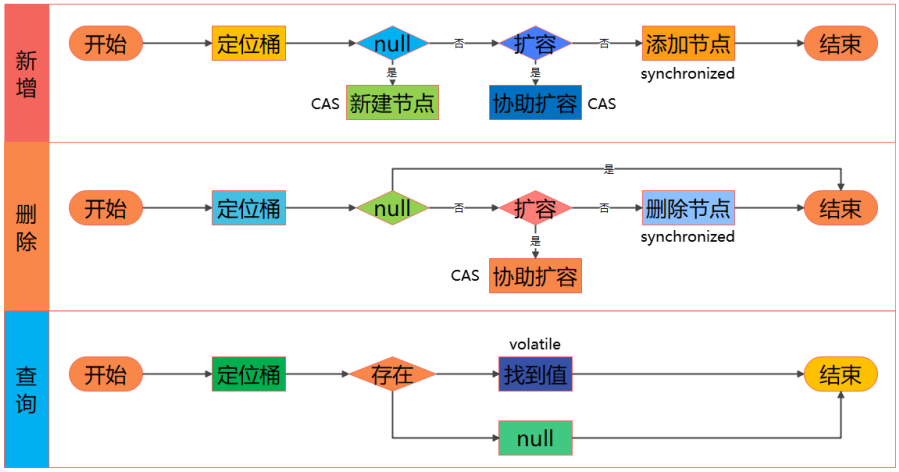
ConcurrentHashMap如何保证线程安全(月薪1.5万的回答示范)

1、Segment维护一个子Hash表
2、Hash表中元素类型HashEntry
3、并发环境下，多个分段可同时进行



ConcurrentHashMap如何保证线程安全(月薪1.5万的回答示范)

1、底层结构是数组+链表+红黑树
2、使用CAS+synchronized保证线程安全
3、新增和删除过程类似
4、查询没有加锁，节点val有volatile



ConcurrentHashMap在1.7版本中是数组+链表+Segment。一个Segment就是一个子哈希表，Segment里维护了一个HashEntry数组。并发环境下如果多个修改发生在不同的分段上，这样线程之间就不存在锁竞争，从而提高了并发效率。

ConcurrentHashMap从1.8版本开始底层数据结构使用数组+链表+红黑树，每个节点类型为Node，一个Entry的子类。使用CAS+synchronized保证线程安全，不再使用Segment。

新增数据时根据Key的Hash定位到数组中桶的位置，然后判断桶中是否为null。如果为null，在桶中新建节点，此时使用CAS保证线程安全。如果不为null，还需要判断数组是否在扩容。如果正在扩容，协助扩容。此时使用CAS保证线程安全。如果没有在扩容，把节点添加到链表或红黑树中。如果链表长度大于8还需要转换为红黑树。整个添加过程都是使用互斥锁synchronized保证线程安全。

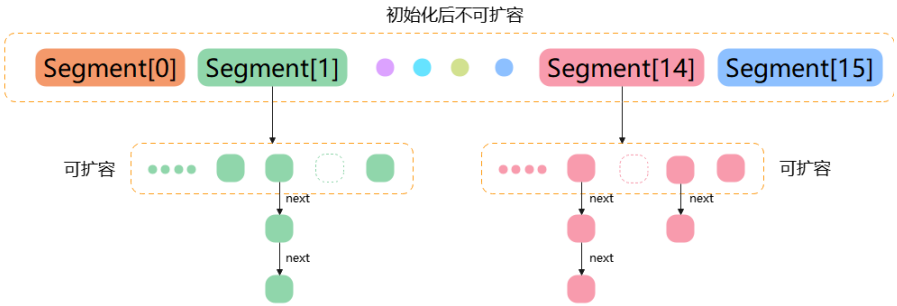
删除数据时和新增数据类似，只是在定位桶时，如果桶中数据为null，直接结束删除。整个过程使用的还是CAS和synchronized保证线程安全。

而查询时使用了无锁设计。根据key的Hash定位到桶。如果桶中存在数据，会遍历寻找，找到后返回节点的值。为了保证值是实时数据，值属性使用volatile进行修饰的，保证值的可见性。

以上就是ConcurrentHashMap如何保证线程安全月薪1万5的回答示范

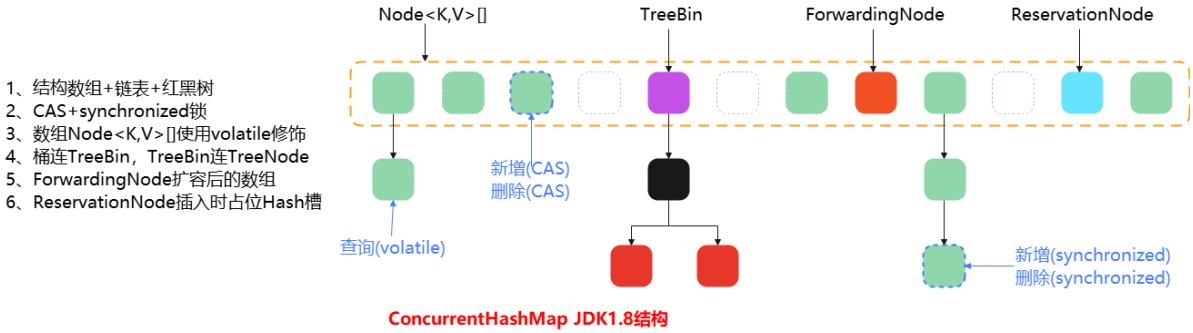
ConcurrentHashMap如何保证线程安全(月薪2万+的回答示范)

1、默认16个Segment
2、分段锁初始化不可扩容
3、每个Segment管理一个Hash表
4、Hash表可扩容



ConcurrentHashMap JDK1.7结构

ConcurrentHashMap如何保证线程安全(月薪2万+的回答示范)



ConcurrentHashMap JDK1.8结构

ConcurrentHashMap在1.7和1.8有明显区别。1.7中是使用Segment保证线程安全。1.8中是使用CAS + synchronized结合Volatile实现线程安全。

先说一下1.7。默认会实例化16个segment分段锁，每个Segment都是可重入锁ReentrentLock的子类，且存储分段锁的数组实例化不可扩容。每个Segment管理一个Hash表，类型为HashEntry。Hash表随着元素个数增加可以扩容。通过这样的方式1.7版本中默认最多并发修改支持到16。虽然使用分段锁相对锁住整个Hash表并发访问下性能有所提升，但是被分为16个Hash表导致访问更加困难而且开销更大。所以在1.8版本中做了一个变化。

在1.8中只是使用了数组+链表+红黑树的方式。结构更加简单。舍弃了1.7中的Segment，变成CAS+synchronized+volatile保证线程安全。

默认对数组使用volatile保证数组的可见性，这也是能让多线程下及时发现数组在扩容。同时扩容时使用ForwardingNode节点存储扩容后的数组，防止多线程下同时出现两个扩容数组。

在新增时如果桶中没有元素，使用CAS保证线程安全。为了在并发场景下保证安全。在桶中没有内容，到添加新的节点之间，先使用ReservationNode占位，并加锁。如果桶中已经有内容了，使用synchronized保证并发安全。把节点插入到链表或红黑树中。为了保证链表转换为红黑树过程中的安全，会先使用TreeBin占位，待转换结果后让红黑树连接TreeBin节点。

删除的过程和新增的过程是类似的，使用CAS+synchronized保证线程安全。

查询时采用无锁设计，效率更高。Node中val使用volatile修饰，这样对于JVM访问CPU缓存中内容时，根据缓存一致性可以拿到最新的值，保证节点值的可见性。这样既保证多线程下读的效率，有保证了读的安全。

以上就是ConcurrentHashMap如何保证线程安全，月薪2万的回答示范