## ConcurrentHashMap

* Java5.0在java.util.concurrent包中提供了多种并发容器类来改进同步容器的性能。
* ConcurrentHashMap是并发效率更高的Map，用来替换其他线程安全的Map容器，比如Hashtable和Collections.synchronizedMap。实际上，并发执行时，线程安全的容器只能保证自身的数据不被破坏，但无法保证业务的行为是否正确。错误的理解这里的线程安全，不恰当的使用ConcurrentHashMap，往往会导致出现问题。
* ConcurrentHashMap同步容器类是Java5增加的一个线程安全的哈希表。对与多线程的操作，介于HashMap与Hashtable之间。内部采用“锁分段”机制替代Hashtable的独占锁。进而提高性能。
* 此包还提供了设计用于多线程上下文中的Collection实现：

ConcurrentHashMap、ConcurrentSkipListMap、ConcurrentSkipListSet、CopyOnWriteArrayList和CopyOnWriteArraySet。

* 当期望许多线程访问一个给定collection时，ConcurrentHashMap通常优于同步的HashMap，ConcurrentSkipListMap通常优于同步的TreeMap。
* 当期望的读数和遍历远远大于列表的更新数时，CopyOnWriteArrayList优于同步的ArrayList。

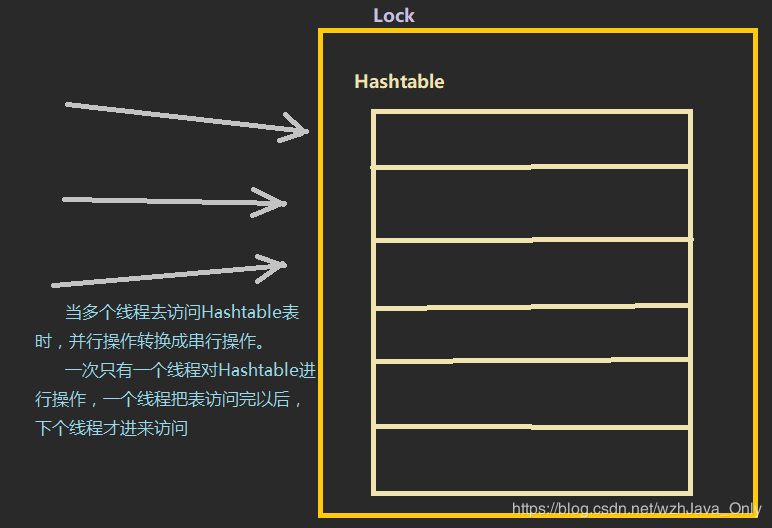
注：JDK1.8以后ConcurrentHashMap由锁的分段机制变为CAS。

*/\*\*  
 \* CopyOnWriteArrayList/CopyOnWriteArraySet：“写入并复制”  
 \*  
 \* 注意：添加操作多时，效率低，因为每次添加时都会进行复制，开销非常大，并发迭代操作多时可以提高效率。  
 \*/*

**public class** TestCopyOnWrite {  
  
 **public static void** main(String[] args) {  
  
 HelloThread ht = **new** HelloThread();  
  
  
 **for** (**int** i = 0; i < 10; i++) {  
 **new** Thread(ht).start();  
 }  
 }  
}  
  
**class** HelloThread **implements** Runnable {  
  
*// private static List<String> list = Collections.synchronizedList(new ArrayList<String>());* **private static** CopyOnWriteArrayList<String> *list* = **new** CopyOnWriteArrayList<>();  
  
 **static** {  
 *list*.add(**"AA"**);  
 *list*.add(**"BB"**);  
 *list*.add(**"CC"**);  
 }  
  
 @Override  
 **public void** run() {  
 Iterator<String> it = *list*.iterator();  
 **while** (it.hasNext()) {  
 System.***out***.println(it.next());  
  
 *list*.add(**"AA"**);  
 }  
 }  
}

HashMap与Hashtable的底层都是哈希表。

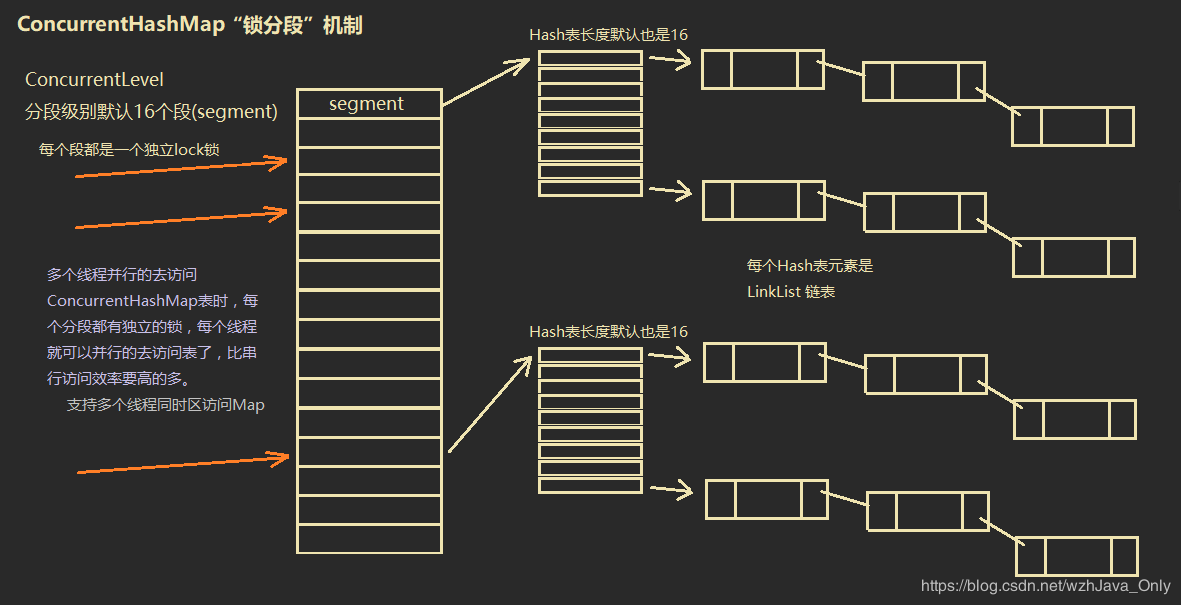
* Hashtable线程安全，效率低。HashTable不允许key和value为null。且存在[复合操作]（如“若存在则删除”，“若不存在则添加”）的安全性问题，原因是hashtable使用时锁的是全表，其他线程只能阻塞；



* HashMap线程不安全，在多线程环境下，使用HashMap进行put操作时，可能会引起死循环，导致CPU利用率接近100%，所以在并发情况下不能使用HashMap；

## 锁分段机制

我们知道Map的结构，是采用数组+链表的结构实现的（JDK1.8改采用数组+链表+红黑树实现），而ConcurrentHashMap采用了一种“分段锁”来实现的。



ConcurrentHashMap使用Segment（分段锁）技术，将数据分成一段一段的存储，然后给每一段数据配一把锁，当一个线程占用锁访问其中一个段数据的时候，其他段的数据也能被其他线程访问，能够实现真正的并发访问。所以说，ConcurrentHashMap在并发情况下，不仅保证了线程安全，而且提高了性能。

ConcurrentLevel分段的级别默认是16个段(segment)，每一个段中有独立的HashMap表，每个HashMap表的元素又是一个LinkList链表。

16个段，每个段形成了独立的哈希表，每个分段都有独立的锁，当多个线程去访问ConcurrentHashMap时，每个线程就可以并行的去访问表。

Collection工具类转化为synchronizedMap转换为线程安全的。

JDK1.8将ConcurrentHashMap升级了，分段锁几乎取消了，ConcurrentHashMap底层大量使用CAS算法，CAS算法理解为无锁算法无涉及到阻塞。