Hashtable是在JDK1.0中引入的哈希结合，用全互斥方法处理并发情况，性能极差。HashMap是在JDK1.2中引入的，非线程安全的，ConcurrentHashMap在JDK5中引入，在JDK8之前采用的是分段锁的实现，分段锁由内部类Segment实现，继承ReentrantLock，CHM被Segent分了很多段，减小了锁的粒度。在JDK8之后对DHM进行了改造

1. 取消了分段锁机制，进一步降低了冲突概率
2. 引入红黑树结构： 同一个哈希槽上的元素个数超过一定阈值后，单向链表改为红黑树
3. 使用更加优化的方式统计集合内的元素数量

Node 存储单个KV数据节点，内部有key value hash next 指向下一个节点

TreeBin并不存储实际数据，维护对桶内红黑树的读写锁，存储对红黑树的引用

TreeNode 在红黑树结构中实际存储数据的节点

ForwardingNode 扩容转发节点，放置此节点后，外部对原有哈希槽的操作会转发到nextTable上

ReservationNode 执行某些方法时，对其加锁，不如computeIfAbsent等

sizeCtl属性 ： 用来控制table的初始化和扩容操作

最小树化集合容量 64 链表树化阈值 8 从树退化到链表的节点数 6

对size()的优化，在put(),remove()和size()方法中，设计元素总数的更新和计算都彻底避免了了锁的使用，而是使用CAS操作 CounterCell计数单元 ，volatile long baseCount 元素总值

计算size的思路：

当并发量较小时，优先使用CAS的方式直接更新baseCount

当更新baseCount冲突，则会认为进入到比较激烈的竞争状态，通过启用counterCells来减少竞争

如果更新counterCells时多次失败，则通过扩容counterCells数组减少冲突