# JDK1.8 ConcurrentHashMap笔记

ConcurrentHashMap在JDK1.8进行了大幅重写，舍弃了之前segment的思路，总体上多用Unsafe类，CAS算法乐观锁以及Node加锁的方式，提升了性能。本文就几个代表性的知识点说说个人的理解。

先说一下相关的演进：

HashMap，线程不安全。

HashTable，线程安全，但是锁整个桶。

ConcurrentHashMap（JDK1,8之前的版本），线程安全，锁segment。

ConcurrentHashMap（JDK1.8），线程安全，锁Node。

## sizeCtl

了解过HashMap源码的同学都知道，里面有一个扩容用的阈值，即容量的0.75。

在ConcurrentHashMap里，sizeCtl承担这个任务，但也有一定的扩展。

当sizeCtl=-1，代表正在初始化。

当sizeCtl=-N，代表有N-1个线程在扩容。

当sizeCtl=0，代表需要初始化。

当sizeCtl>0，即扩容的阈值，也即总容量\*0.75。

## Table、nextTable

了解HashMap的同学就知道，table就是我们常说的“桶”。nextTable是用来扩容用的临时变量。

## initTable()

当我们构造一个ConcurrentHashMap的时候，只是对一些属性进行设定，并没有初始化。只有当我们对Map进行操作——比如put()的时候才执行初始化，懒汉模式。

当table为空或者table的长度为0时，就代表还未初始化。

对ConcurrentHashMap进行初始化，只能由单线程进行操作。

线程先校验sizeCtl是否<0。

是：这说明有其他线程正在负责进行初始化，挂起。

否：通过Unsafe类的方法，通过CAS，将sizeCtl修改为-1，表示已经有线程负责初始化。

初始容量table是16，sizeCtl是12，容量的0.75倍。

## transfer()

着重说说扩容这个方法，主要是实现方式真好。

基本思路与HashMap相像。

第一步是用单线程构建一个2倍容量的nextTable。

第二步就是用多线程去遍历table里面所有的Node。

这里有一个重要的类：ForwardingNode，它其实就是一个标识类，放在table的桶里，告诉线程本节点为空或已处理完毕，请向后遍历。

当线程遍历到有值的Node时，加锁后进行处理，待处理完毕后，在table原位置插入ForwardingNode，表示处理完毕。

当对节点的数据进行处理时，将链表或树区分为两类，分别放在nextTable的i，i+n位置处。

第三步，当所有数据都复制到nextTable之后，将其复制给table，然后将nextTable置空，sizeCtl设为新容量的0.75倍。