**今天课程安排：**

Atomic体系介绍

CAS源码分析

CAS的ABA问题

ConcurrenthHashMap

HashMap

HashTable

CopyOnWriteArrayList

ArrayList

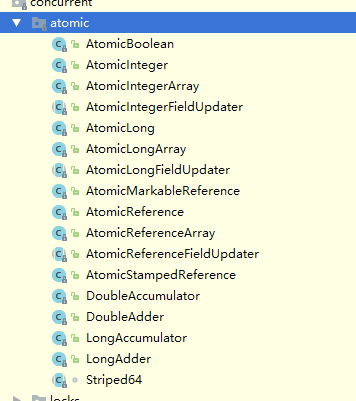
上节课我们学习了锁，通过我们对程序加锁，我们在高并发下也可以达到我们想要的预期

效果，同时我们也知道加锁的话，会造成很多线程需要被等待拿锁效率不是很高（其实很高了）。

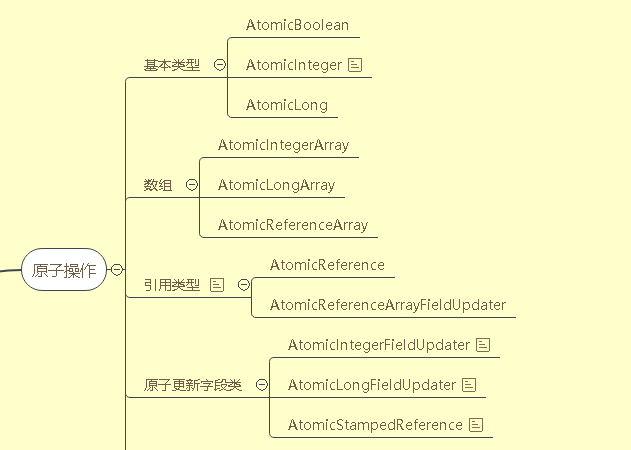
有没有更好的办法了？

今天我们就来学习下，java著名的原子操作包java.util.concurrent.atomic,简称Atomic

# Atomic体系



归纳为:



新增jdk8：LongAdder

首先我们来说下AtomicInteger：

AtomicInteger是对int类型的一个封装，提供原子性的访问和更新操作。常见的api有

java.util.concurrent.atomic.AtomicInteger#incrementAndGet 原子自增1

java.util.concurrent.atomic.AtomicInteger#decrementAndGet原子自减1

java.util.concurrent.atomic.AtomicInteger#addAndGet原子在当前值加上给定的值。

java.util.concurrent.atomic.AtomicInteger#get返回当前最新的值

**场景**：并发统计、订单号

**原理：**

CAS有3个操作数，内存值V，旧的预期值A，要修改的新值B。当且仅当预期值A和内存值V相同时，将内存值V修改为B，否则什么都不做。

CAS算法：

CAS的全称是Compare And Swap 即比较交换，其算法核心思想如下

函数：CAS(V,E,N) 参数：V表示要更新的变量 E预期值 N新值

如果V值等于E值，则将V的值设为N。若V值和E值不同，则说明已经有其他线程做了更新，则当前线程什么都不做。通俗的理解就是CAS操作需要我们提供一个期望值，当期望值与当前线程的变量值相同时，说明还没线程修改该值，当前线程可以进行修改，也就是执行CAS操作，但如果期望值与当前线程不符，则说明该值已被其他线程修改，此时不执行更新操作，但可以选择重新读取该变量再尝试再次修改该变量，也可以放弃操作

Java的CAS操作通过Unsafe类来完成里面基本都是native，即通过JNI调用c/c++等代码

底层核心：cmpxchg

cmpxchg是汇编指令

作用：比较并交换操作数.

如：CMPXCHG r/m,r 将累加器AL/AX/EAX/RAX中的值与首操作数（目的操作数）比较，如果相等，第2操作数（源操作数）的值装载到首操作数，置为1。如果不等， 首操作数的值装载到AL/AX/EAX/RAX并将置为清0

该指令只能用于486及其后继机型。第2操作数（源操作数）只能用8位、16位或32位寄存器。第1操作数（目地操作数）则可用寄存器或任一种存储器寻址方式。

### AtomicBoolean：

Atomic实现都是cas，1 0

Boolean true false

场景： 可以替换volatile。

### CAS原理解析：

利用CPU的CAS指令，同时借助JNI来完成Java的非阻塞算法，其它原子操作都是利用类似的特性完成的。 在 java.util.concurrent 下面的源码中，Atomic, ReentrantLock 都使用了Unsafe类中的方法来保证并发的安全性。

CAS操作是原子性的，所以多线程并发使用CAS更新数据时，可以不使用锁，JDK中大量使用了CAS来更新数据而防止加锁来保持原子更新。

CAS 操作包含三个操作数 ：内存偏移量位置（V）、预期原值（A）和新值(B)。 如果内存位置的值与预期原值相匹配，那么处理器会自动将该位置值更新为新值 。否则，处理器不做任何操作。

### （CAS）Atomic\*源码分析

课上细说，这块会涉及到c、c++代码。

## CAS的ABA问题

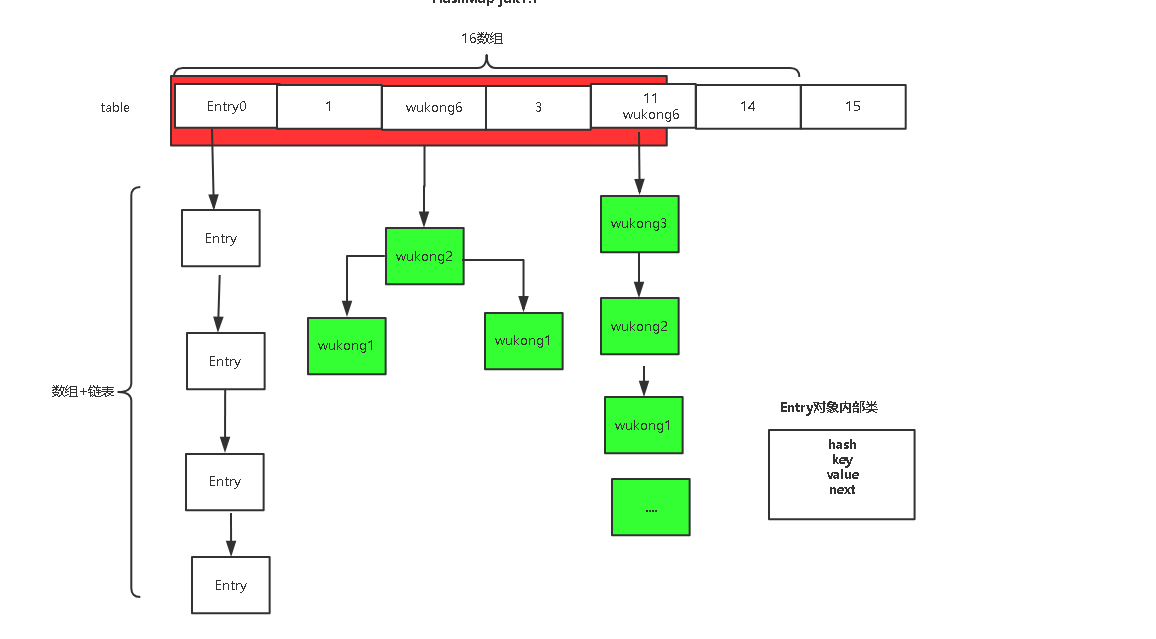
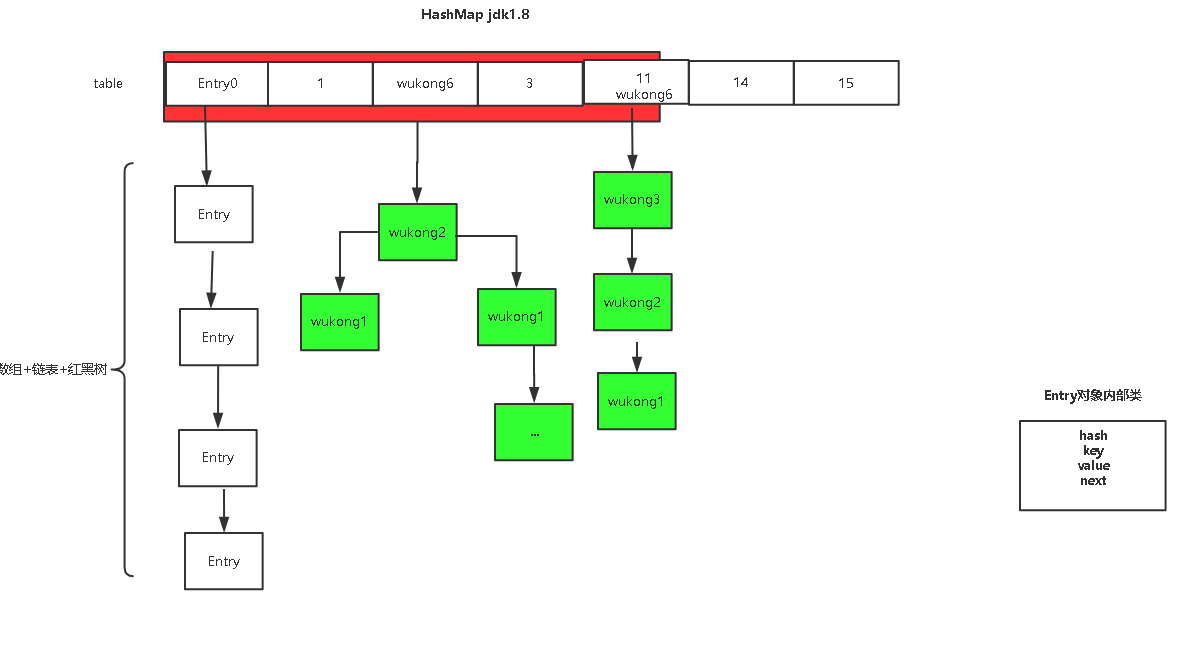
在进行CAS操作的时候，因为在更改V之前，CAS主要询问“V的值是否仍然为A”，所以在第一次读取V之后以及对V执行CAS操作之前，如果将值从A改为B，然后再改回A，会使基于CAS的算法混乱。在这种情况下，CAS操作会成功。这类问题称为ABA问题。

# Collections

# Map

## HashMap&HashTable&ConcurrentHashMap

HashMap源码解析：



HashMap数据结构是什么？

底层数据结构：数组+链表+红黑树

分析（JDK17.和1.8）区别 遍历

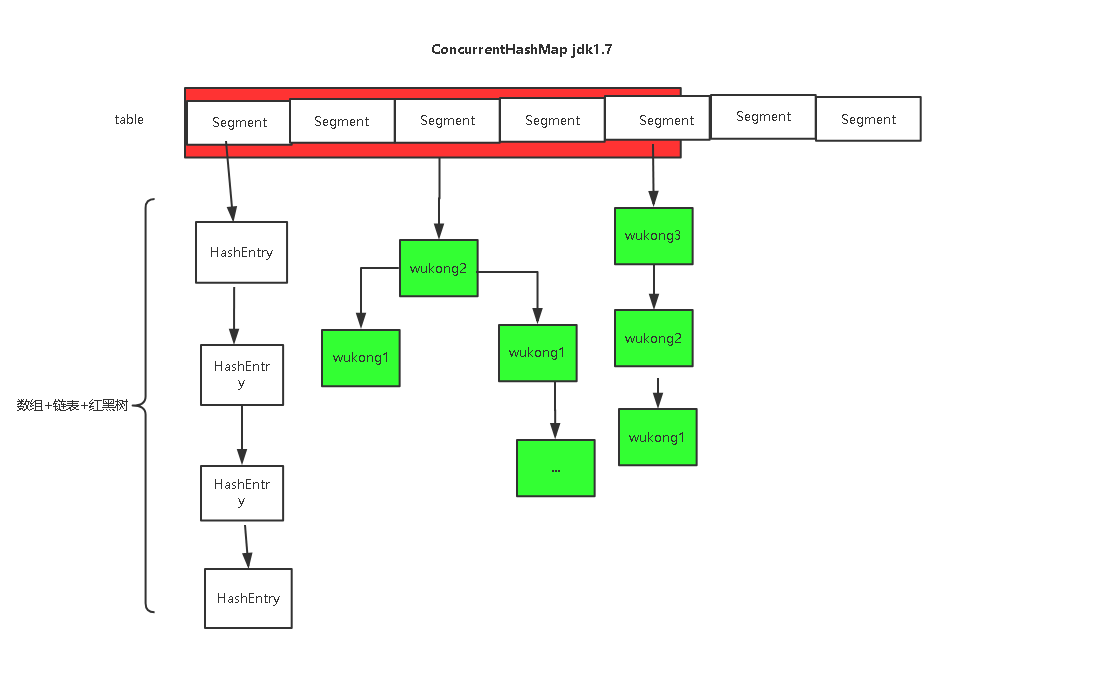
HashMap的扩容和如何优化？

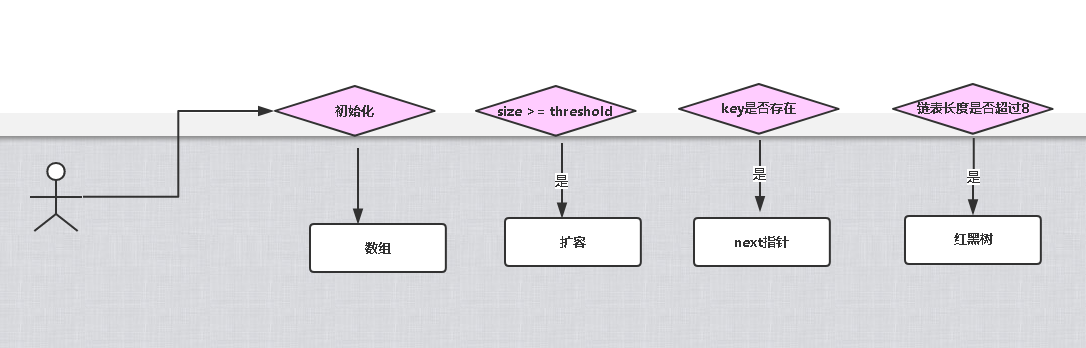
初始化

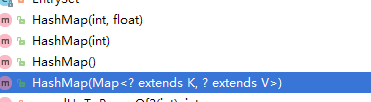
HashTable&ConcurrentHashMap

锁的方式：hashtable jvm锁

分段锁：Segement(1.7)







put发生，初始化指定

ConcurrentHashMap和Hashtable的相同点和不同点

## List

## CopyOnWriteArrayList&ArrayList

ArrayList：数组，默认容量：10 LinkedList:数据结构

CopyOnWriteArrayList原理解析：

读写分离、读多写少 黑白名单

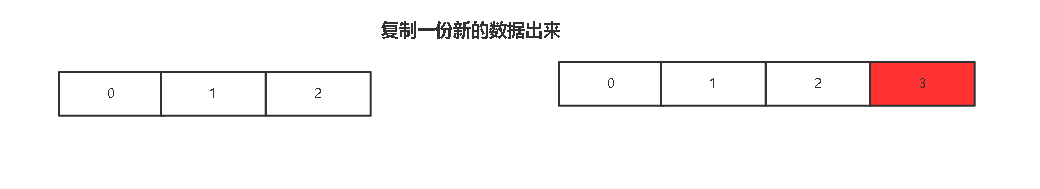
**步骤：**

1、如果写操作未完成，那么直接读取原数组的数据；   
2、如果写操作完成，但是引用还未指向新数组，那么也是读取原数组数据；   
3、如果写操作完成，并且引用已经指向了新的数组，那么直接从新数组中读取

CopyOnWriteArrayList&ArrayList相同点和不同点

数组

Get 有没有锁 ？为什么没有



作业：

新增jdk8：LongAdder

ConcurrentSkipListMap 数据结构+算法

ConcurrentHashMap