concurrent 并发

atomin 原子性, 读法aotumeike

volatile的理解

1volatile是Java虚拟机提供的轻量级同步机制(轻量的意思是低配)

1.1保证可见性

1.2不保证原子性

1.3精致指令重排

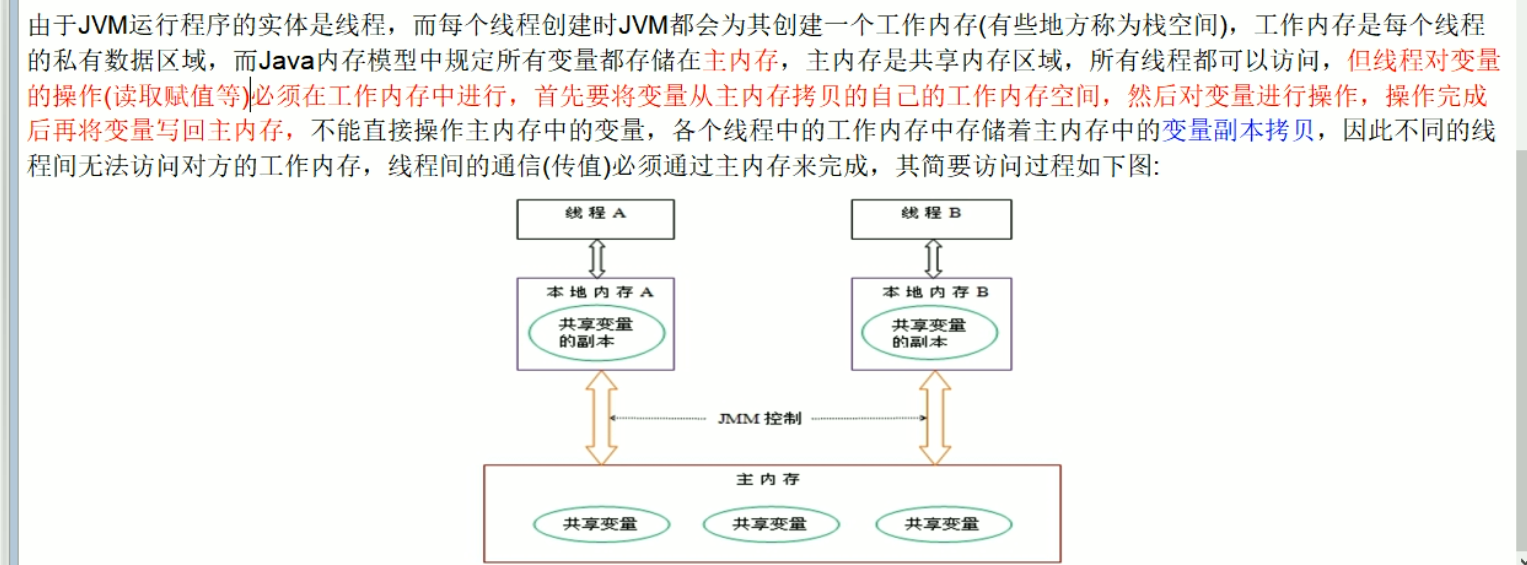
jvm(java虚拟机)

jmm(java内存模型):本身是一种抽象的概念并不真实存在,它描述的是一组规则或规范,

通过这组规范定义了程序中各个变量(包括实例字段,静态字段和 构成数组对象的元素)的访问方式.

主内存:

自己的工作内存:

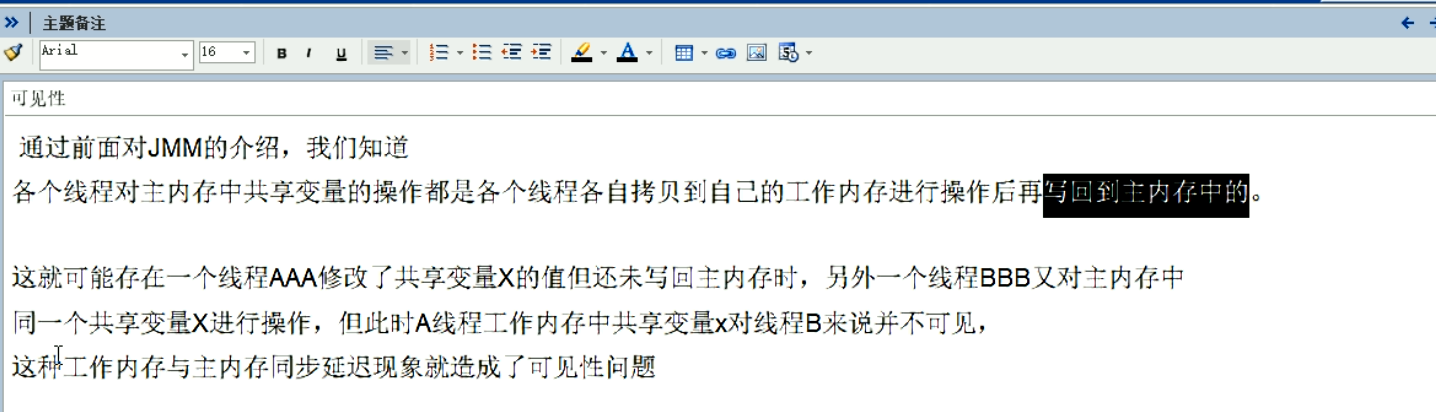


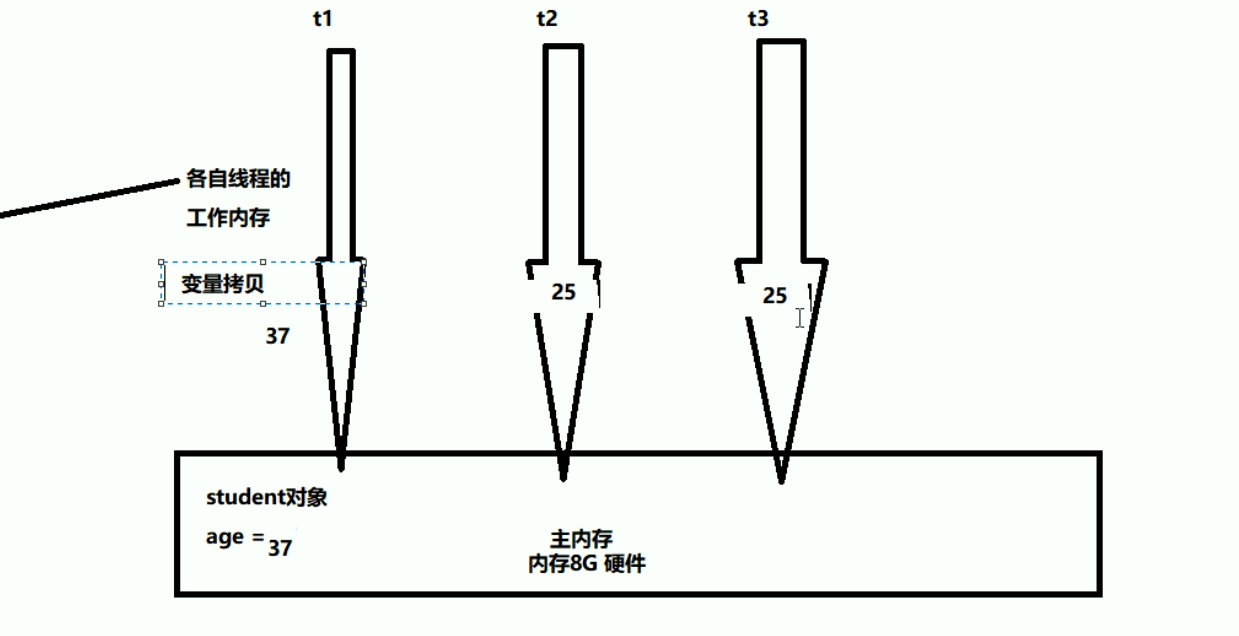
读取速率:硬盘<内存<cpu

Cpu和内存之间还有缓存

内存模型中三大特性

1可见性:只要有一个线程修改完自己的工作空间的值并且写回给主空间以后要及时通知其他线程,这个及时通知的这种情况俗称jmm内存模型中的可见性

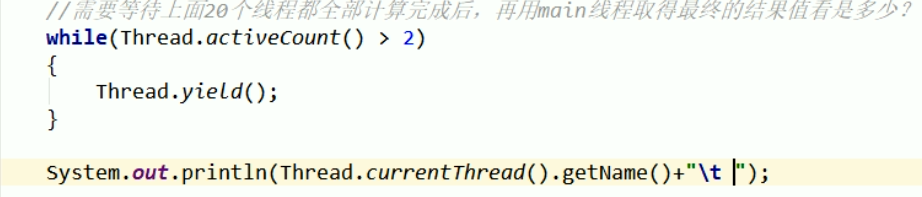




可见性的代码

package com;  
class Data{  
 volatile int i=0;  
 public void setI(){  
 i=60;  
 }  
};  
public class MyVoliteDemo {  
 public static void main(String[] args) {  
 Data data = new Data();  
 Thread thread = new Thread(){  
 @Override  
 public void run() {  
 System.*out*.println(Thread.*currentThread*().getName()+"修改前-----"+data.i);  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 data.setI();  
 System.*out*.println("修改后"+data.i);  
 }  
 };  
 thread.start();  
 while (data.i==0){  
 *// System.out.println("wqeqweqw");* }  
 System.*out*.println("-------");  
 }  
}

线程礼让,让别的线程执行完,在执行这个线程 大于2是因为Java最少有2个main线程



解决volatile不保证原子性问题

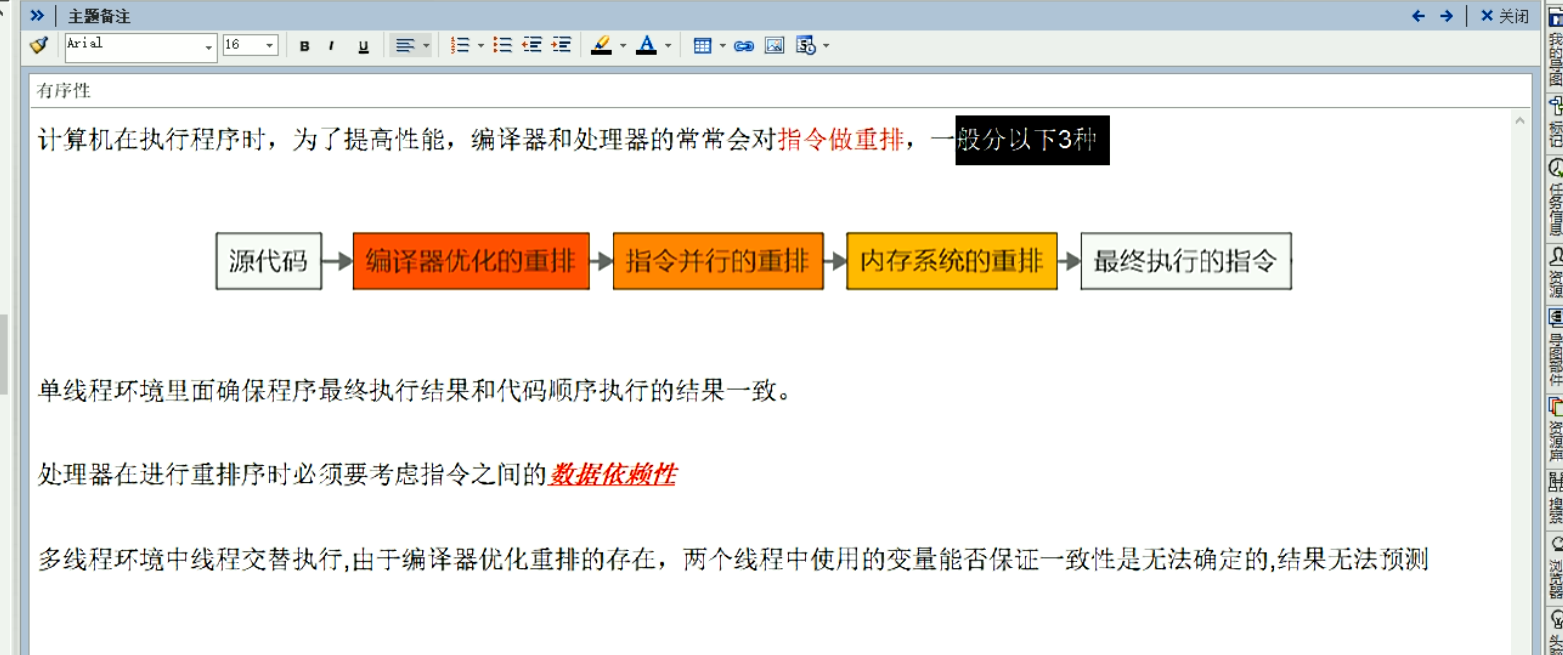
使用AtomicInteger 原子性的Integer 读法 e tao mei ke

AtomicInteger atomicInteger = new AtomicInteger();

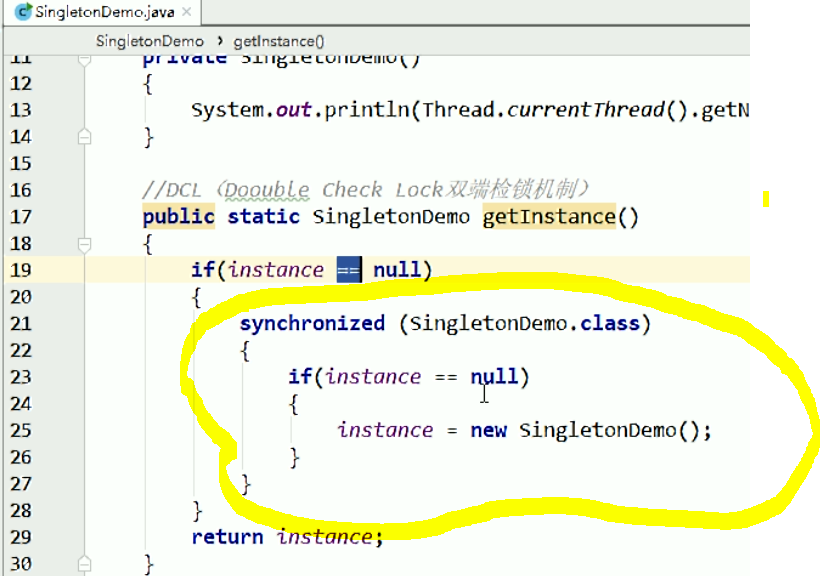
括号不写值默认为0,写多少默认值就为多少

(需要了解juc)

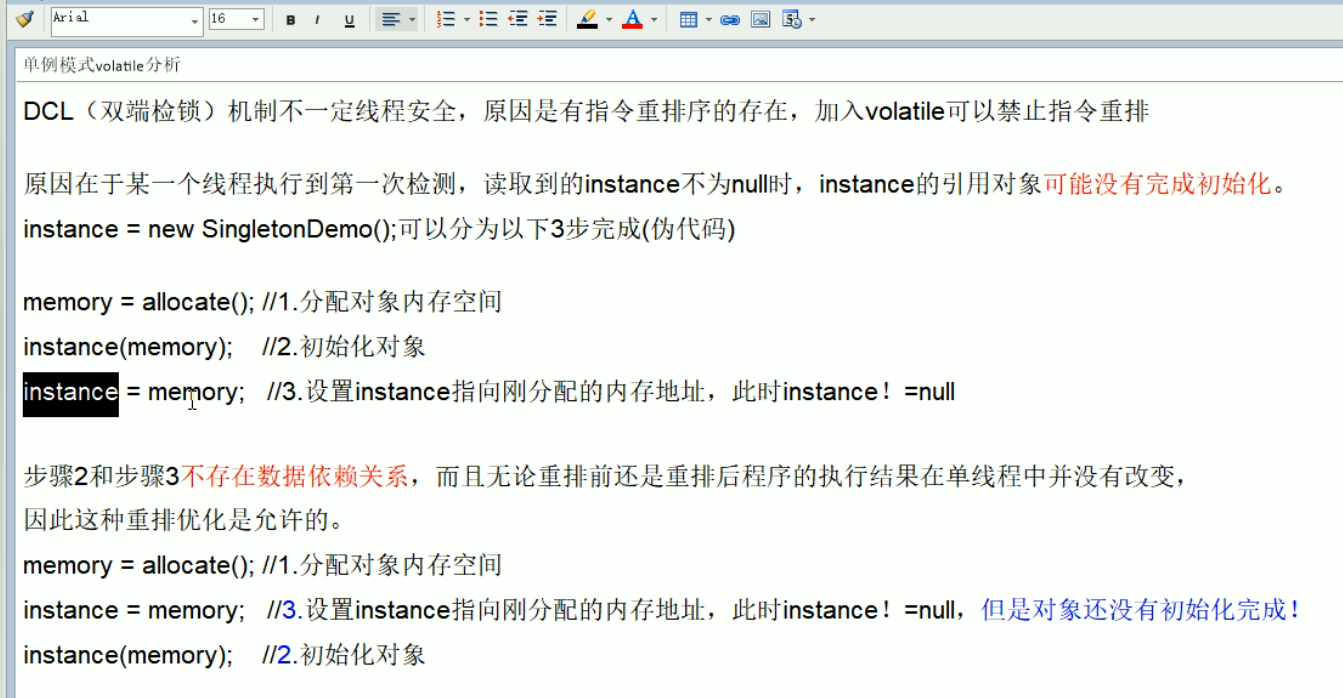
有序性,禁止指令重排

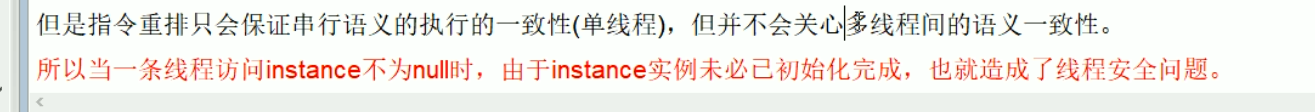


单例模式在高并发情况下需要加锁,对new对象加锁,双端检索机制(正确性99.99%)

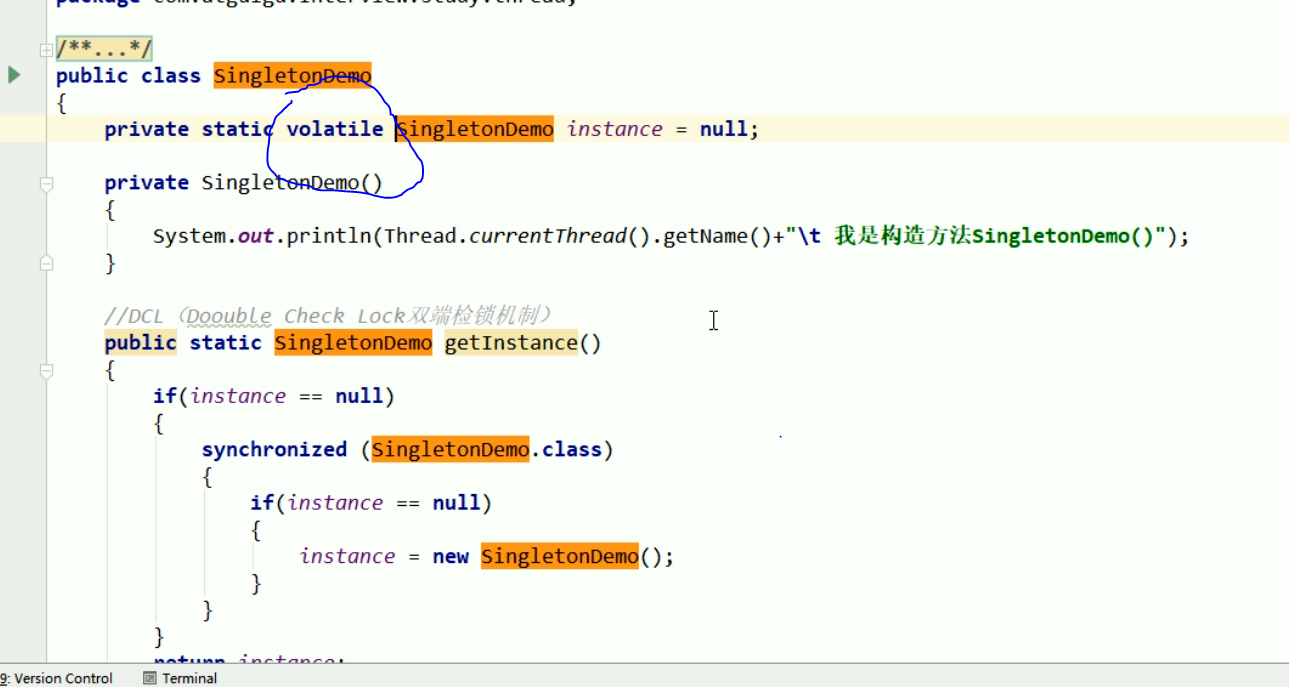


为什么会发生多线程的错误呢?-----和对象的初始化与为对象分配内存地址的顺序有关,指令重排序不要能保证多线程安全,所以会发生多线程问题,





解决办法---------禁止指令重排-----使用volatile



**Volatlie笔记**

