# Volatile关键字实现原理

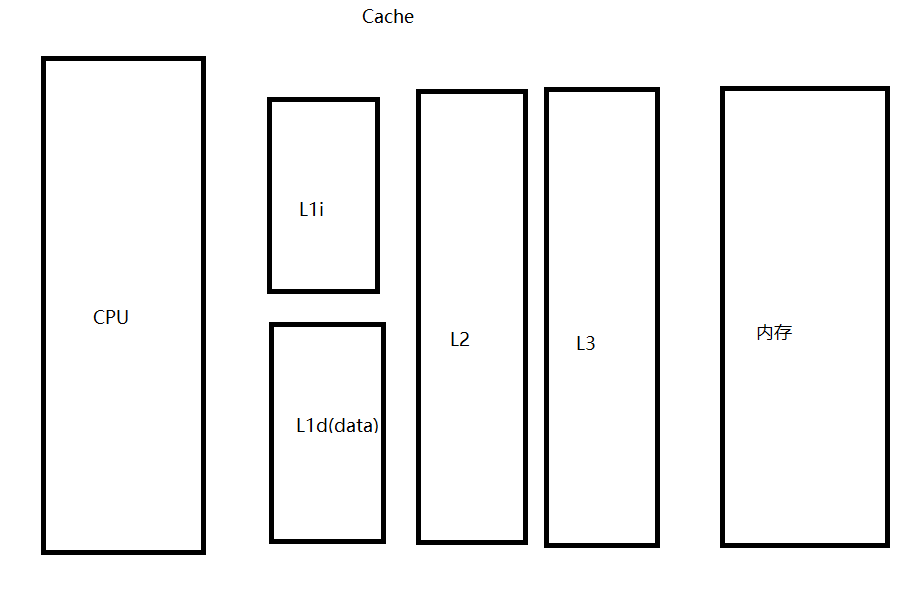
## 认识volatile关键字

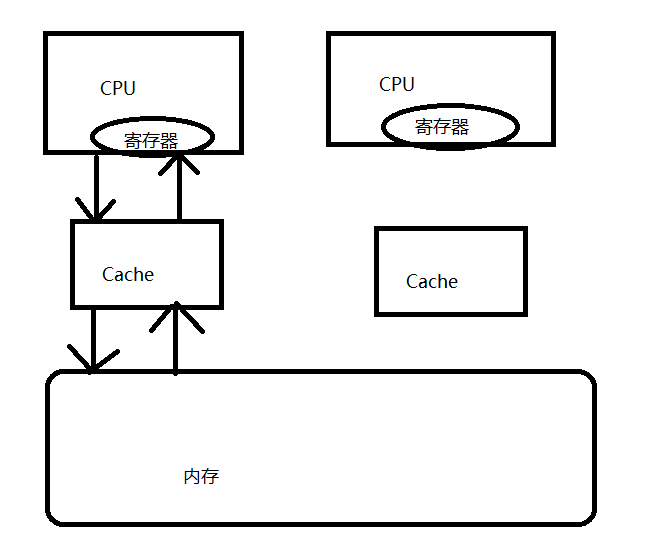
**当一个共享变量被volatile修饰时，它会保证修改的值立即被更新到主存**

## 机器硬件CPU与JMM

由于 cpu 的速度远远高于 内存的速度 ， 因此有 一级缓存、二级缓存、三金缓存

1. CPU Cache模型



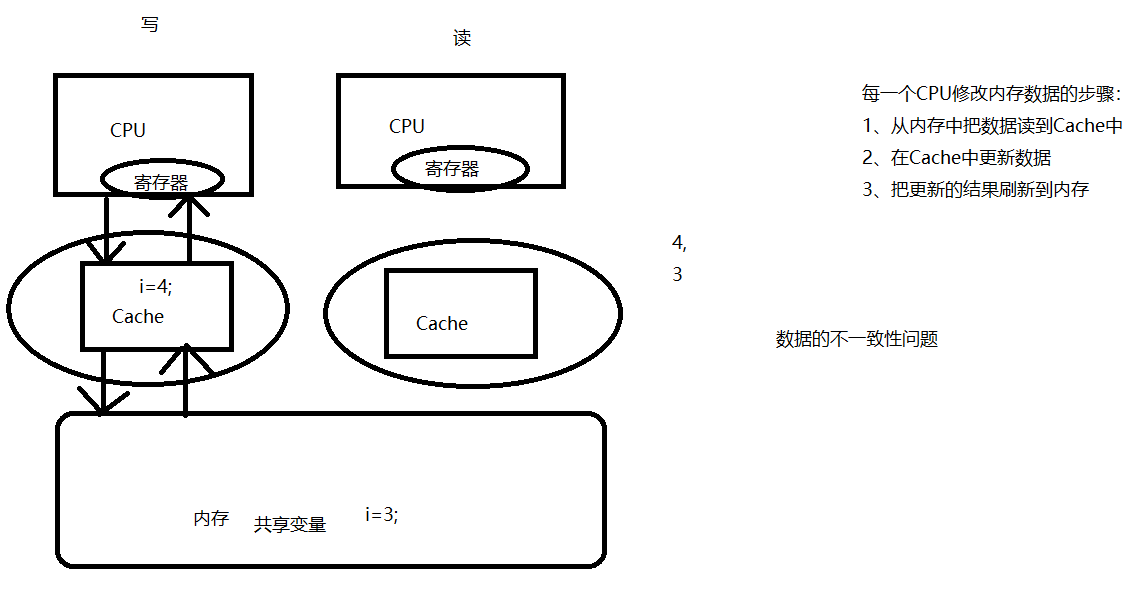


程序的局部执行原理

数据先由内存 写到 缓存 ，再由缓存写到 再到寄存器 ， cpu 直接读取 寄存器中的数据

如果寄存器中找不到数据 ，就再到缓存中找 如果缓存中没有再到 内存中读取 ，如果找到了 则拷贝一份

1. CPU缓存的一致性问题

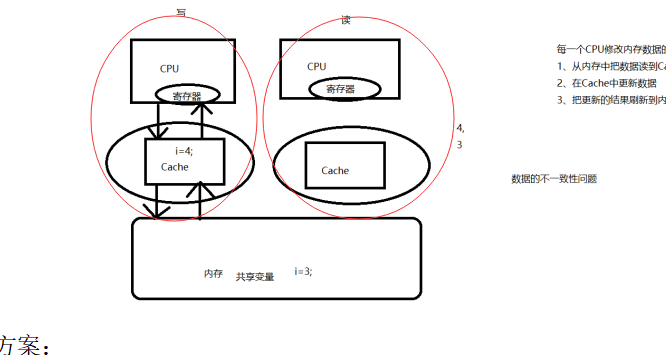


由于操作数据 是先改变 缓存中的数据 并且两个线程中的cache是独立不可见的 ，线程在线程写数据期间 ，还未将数据同步到内存，可能 另个一线程读自己的cache中的数据 可能读到的就是脏数据

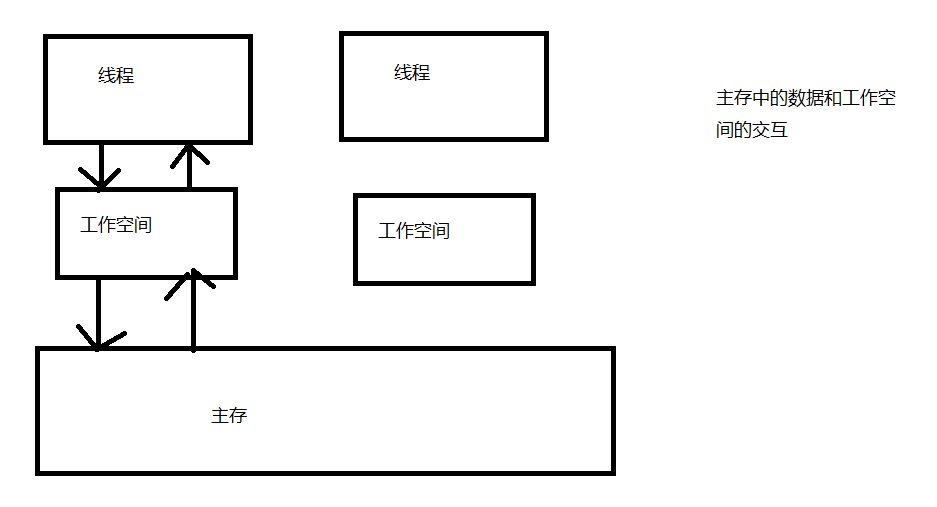
解决方案：

1. 总线加锁（粒度太大）
2. MESI（缓存一致性协议）
   1. 读操作：不做任何事情，把Cache中的数据读到寄存器
   2. 写操作：发出信号通知其他的CPU将改变量的Cache line置为无效，其他的CPU要访问这个变量的时候，只能从内存中获取（等于是cache 不能使用）。

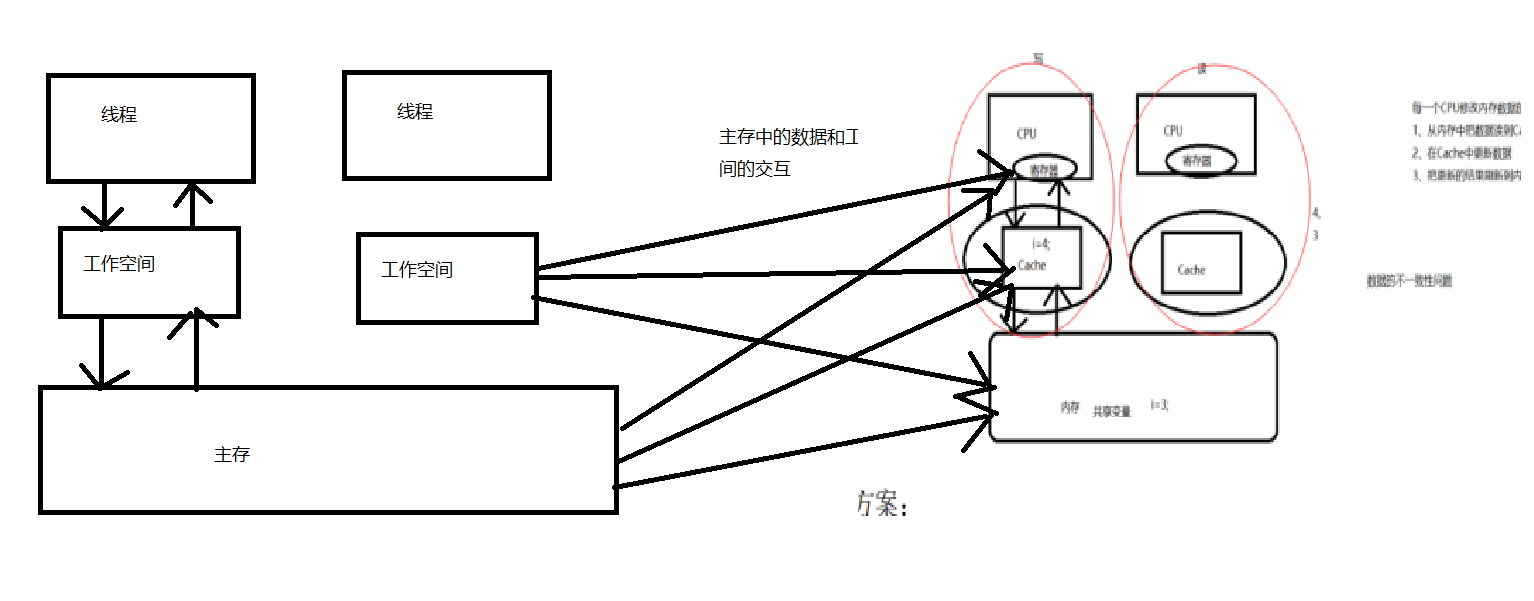
Cache line CPU的cache中会增加很多的Cache line



1. Java内存模型



1. 主存中的数据所有线程都可以访问（共享数据）
2. 每个线程都有自己的工作空间，（本地内存）（私有数据）
3. 工作空间数据：局部变量、内存的副本
4. 线程不能直接修改内存中的数据，只能读到工作空间来修改，修改完成后刷新到内存



总结 ： 我们一般操在多线程操作数据时 , 当某一线程 改变数据时 只是改变当前工作空间中的值 ， 而其他线程并不能感知到 。 也就是说 在A线程修改数据后工作空间数据发生变化，然后刷新内存中的数据。但是线程B工作空间 中的数据还是原来数据的备份 ， 线程B 不知道数据已经发生了 改变 。如果想让B线程 数据发生了改变 需要给当前数据 加 volatile 关键字

（valatile ： 通过控制cpu Cache line链，不让其他线程在其他自己的工作空间读数据 而是到内存中读数据）

## Volatile关键字

volatile作用：让其他线程能够马上感知到某一线程多某个变量的修改

1. 保证可见性 (100%保证，修改数据会让其他线程立即感知到)

对共享变量的修改，其他的线程马上能感知到

不能保证原子性 读、写、（i++）

1. 保证有序性 （但是不是 100% 保证原子性： 在多个多个原子性操纵的时候不具备原子性 如：i++ 能保证其中每一步的原子性 不能保证i++ 的原子性）

**打破 ：有序性（编译阶段、指令优化阶段）**

代码在编译时代码重排， 运行时指令重排

**有序性**一把双刃剑，虽然优化了程序的执行效率，但是在某些情况下，会影响到多线程的执行结果

在一个变量被volatile修饰后，JVM会为我们做两件事：

1.在每个volatile写操作前插入StoreStore屏障，在写操作后插入StoreLoad屏障。

2.在每个volatile读操作前插入LoadLoad屏障，在读操作后插入LoadStore屏障。

因此volatile可以在编译时和运行时的阻止指令重排 ，让代码严格按照编码顺序执行

1. volatile的原理和实现机制(本质就是 ：锁、轻量级)

Volatile int a ; 就是对变量 加 lock 锁 ，线程读取加锁的资源 ，直接从主内存读取数据

## Volatile的使用场景

1. 状态标志（开关模式）

public class ShutDowsnDemmo extends Thread{

private volatile boolean started=false;

@Override

public void run() {

while(started){

dowork();

}

}

public void shutdown(){

started=false;

}

}

1. 双重检查锁定（double-checked-locking）

DCL（7）

public class Singleton {

private volatile static Singleton instance;

public static Singleton getInstance(){

if(instance==null){

synchronized (Singleton.class){

instance=new Singleton();

}

}

return instance;

}

}

1. 需要利用顺序性

## volatile与synchronized的区别

1. 使用上的区别

Volatile只能修饰变量，synchronized只能修饰方法和语句块

1. 对原子性的保证

synchronized可以保证原子性，Volatile不能保证原子性

1. 对可见性的保证

都可以保证可见性，但实现原理不同

Volatile对变量加了lock，synchronized使用monitorEnter和monitorexit monitor JVM

1. 对有序性的保证

Volatile能保证有序，synchronized可以保证有序性，但是代价（重量级）并发退化到串行

1. 其他

synchronized引起阻塞

Volatile不会引起阻塞

#### **volatile缺点**

#### volatile不能保证原子性，也不会阻塞，（能立即感知到值的变化 ，但是有时候不知道具体的值）

#### volatile可以配合synchronized保证原子性。