# Volatile关键字实现原理

## 认识volatile关键字

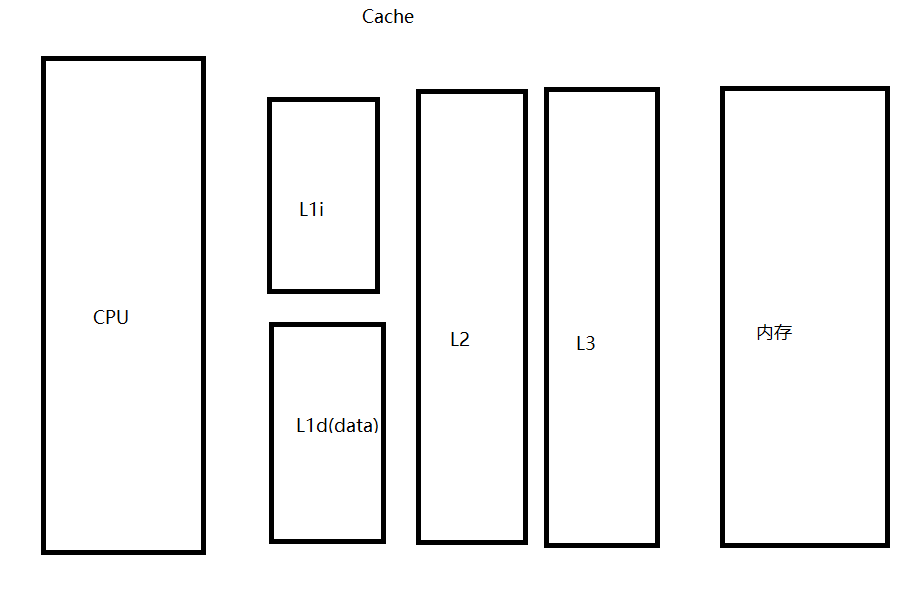
程序举例

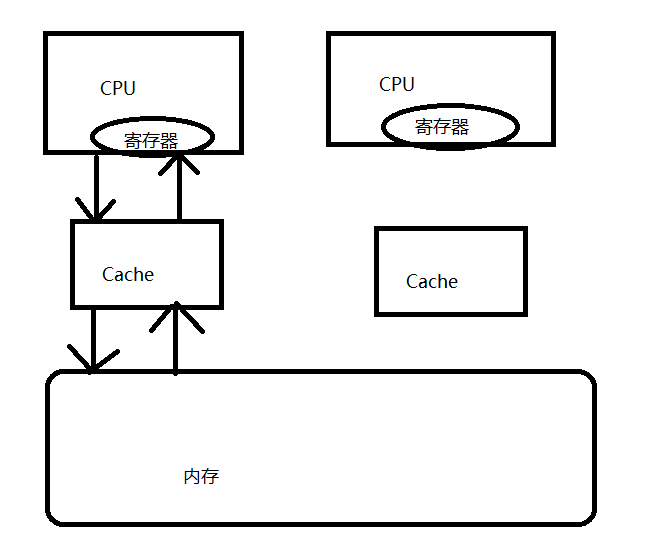
用一个线程读数据，一个线程改数据

存在数据的不一致性

## 机器硬件CPU与JMM

1. CPU Cache模型





程序的局部执行原理

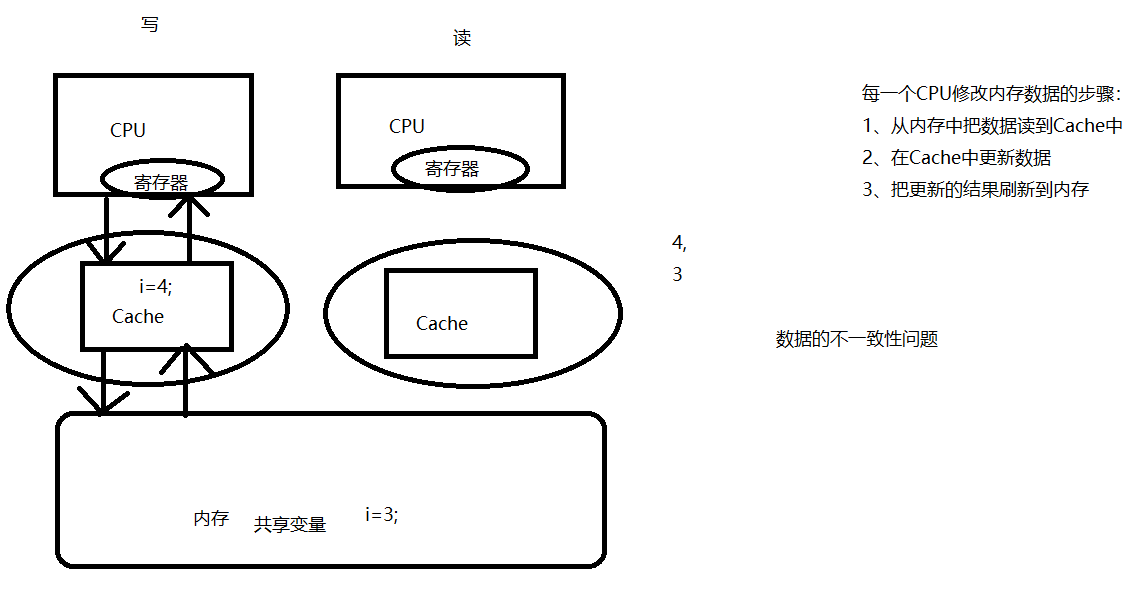
1. L1li：一级缓存

L1d：一级缓存

L2：二级缓存

L3：三级缓存

1. Cache是硬件
2. 增加了CPU的吞吐率
3. CPU缓存的一致性问题



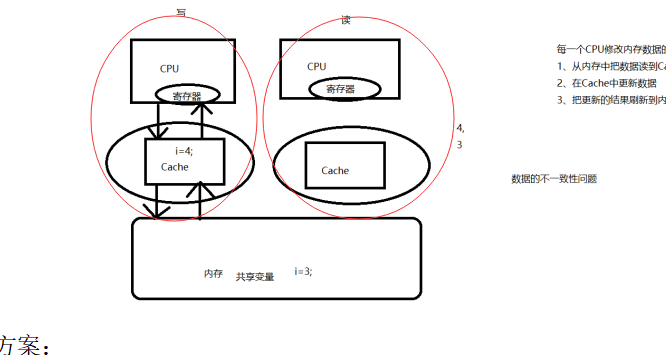
解决方案：

1. 总线加锁（粒度太大）：

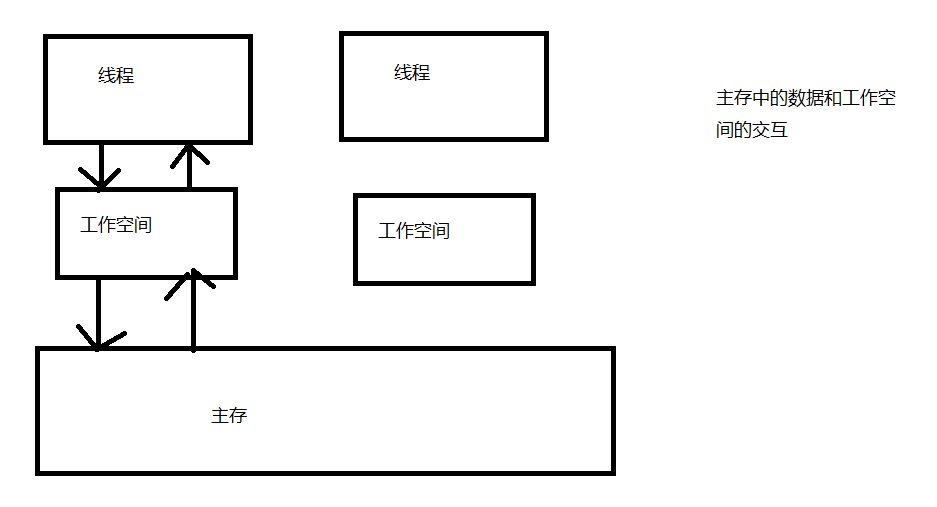
地址总线、数据总线、控制总线，CPU对总线加锁，会导致其余的CPU无法访问这个总线，从而解决问题，但是会导致总的效率下降。

1. MESI（）
   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将该变量的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。

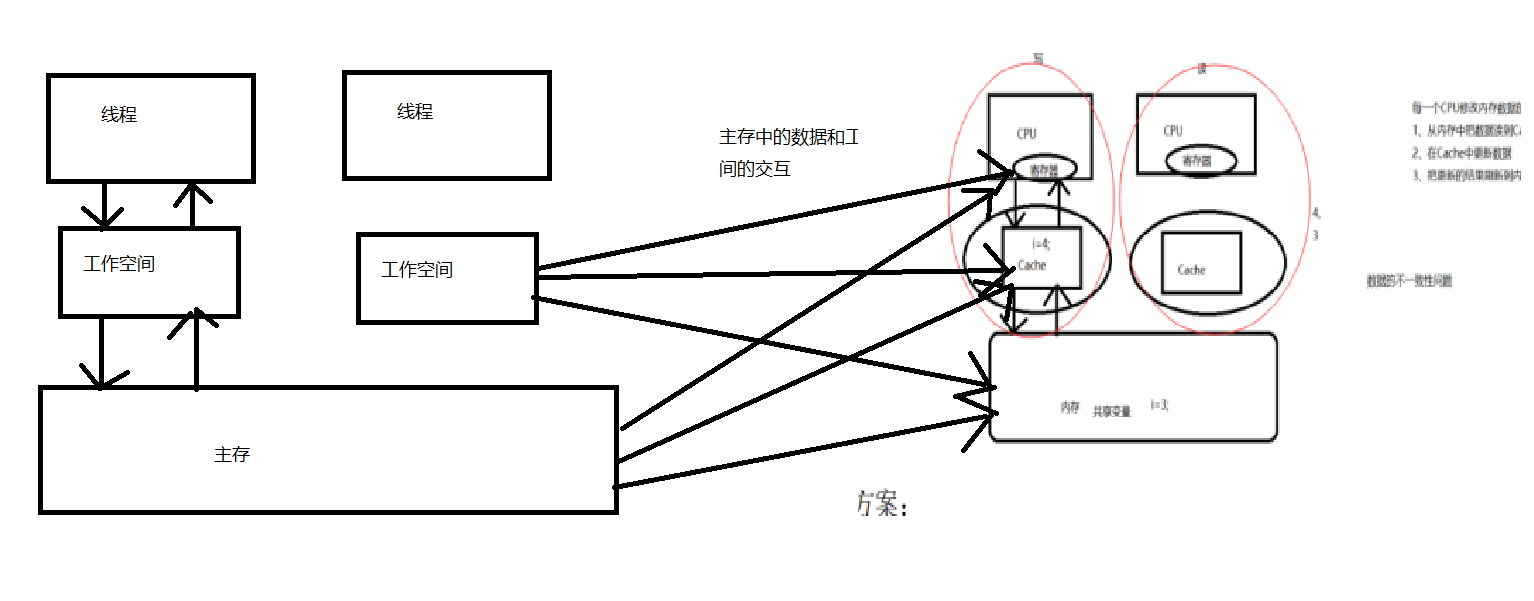
Cache line是CPU的实现机制， CPU的cache中会增加很多的Cache line



1. Java内存模型（JMM）



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(实际是读到寄存器中修改)，修改完成后刷新到内存



* 工作区间里的数据可能在 CPU、cache、内存中
* 主存里的数据可能在CPU、cache、内存中
* 工作区间中存放的是引用对象的地址

## Volatile关键字的语义分析

**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

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

重排序（编译阶段、指令优化阶段）

输入程序的代码顺序并不是实际执行的顺序

重排序后对单线程没有影响，对多线程有影响

Volatile

Happens-before

volatile规则：

对于volatile修饰的变量：

1. volatile之前的代码不能调整到他的后面
2. volatile之后的代码不能调整到他的前面（as if seria）
3. 霸道（位置不变化）

Int i=0;

Int a=3;

Int b=5;

Volatile Int j=3;

Int i=0;

Int a=3;

Int b=5;

Int m=i+j;

I++;

J++;

1. volatile的原理和实现机制(锁、轻量级)

HSDIS --反编译---汇编

Java --class---JVM---》ASM文件

Volatile int a ;

Lock :a

## 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**

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

了解CPU的流水线作业

用HSDIS进行 反编译-反汇编

Java 🡪 class 🡪 JVM 🡪 ASM文件（汇编语言文件）