# Java内存模型

1、线程与JVM

2、java内存区域与java内存模型的区别

3、硬件内存架构与java内存模型

4、java内存模型对并发特征的保证

**1基本概念：**

程序：代码，完成某一件任务，代码序列（静态的概念）

进程：程序在某些数据上的一次运行（动态的概念）

线程：一个进程可能包含一个或多个线程（占有资源的独立单元）

**2 JVM与线程**

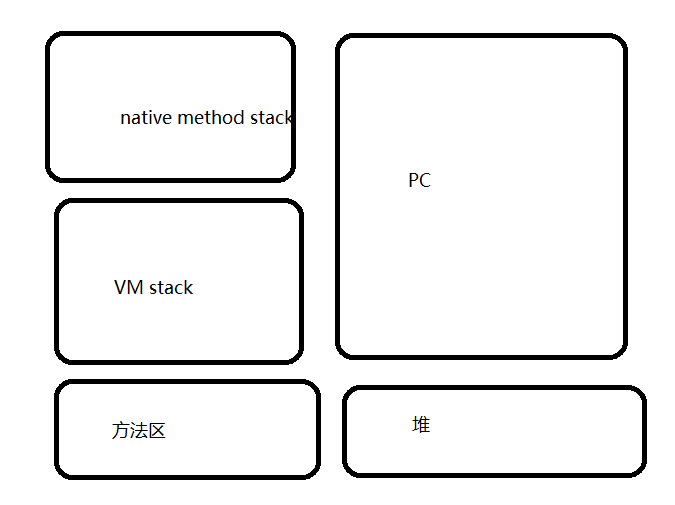
JVM什么时候启动？

Java在虚拟机在类被调用时启动

首先启动JVM线程 –-> 再启动其他的线程（包含main线程）

线程在JVM中运行

**3 JVM内存区域**



**3.1 方法区：**

类信息(类加载器加载的ClassLoader)、常量、static变量 、JIT(即时编译后的代码)

这些数据信息是共享的

数据放满可能出现数据溢出，OOM

**3.2 Java堆区：**

实例对象

GC工作的主要对象

这些数据信息是共享的

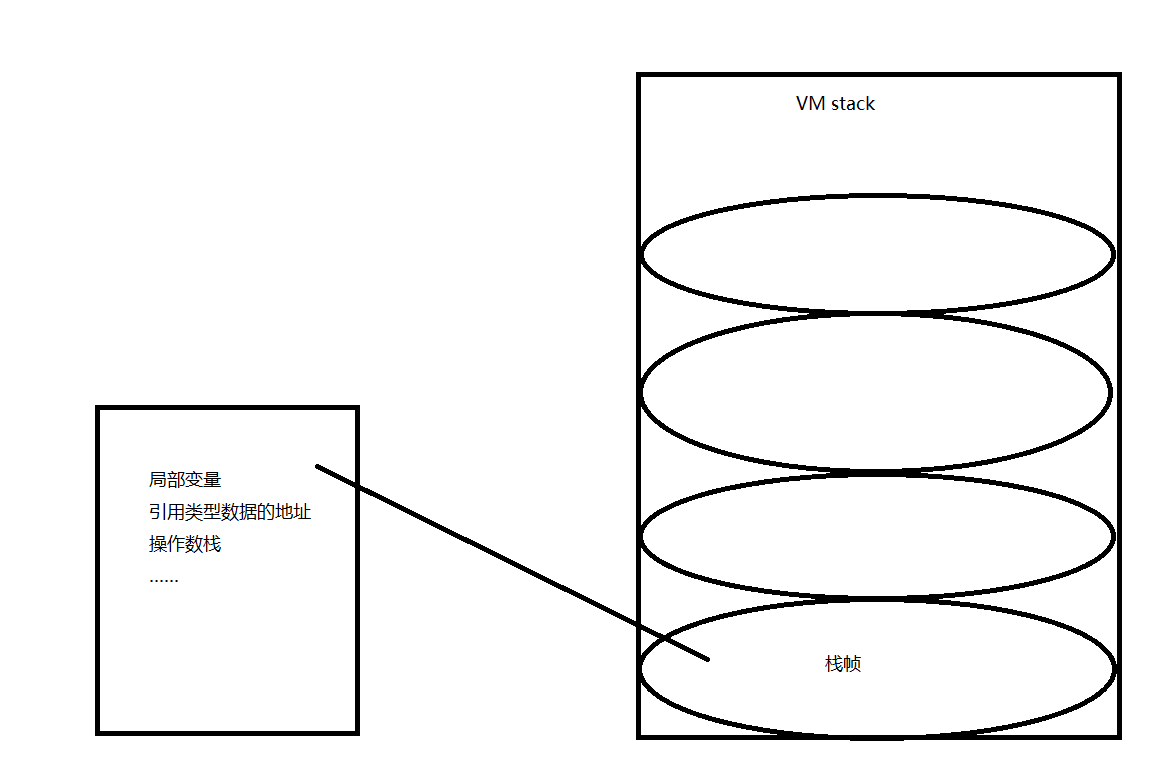
数据放满可能出现数据溢出，OOM

**3.3 VM stack(虚拟机栈)：**

Java方法在运行的内存模型

通常称为栈区

数据放满可能出现数据溢出，OOM



* 每一个方法都有一个栈桢与之对应，栈桢内数据私有
* 栈桢中存放：局部变量、引用类型数据的地址、操作数栈

**3.4 PC：**

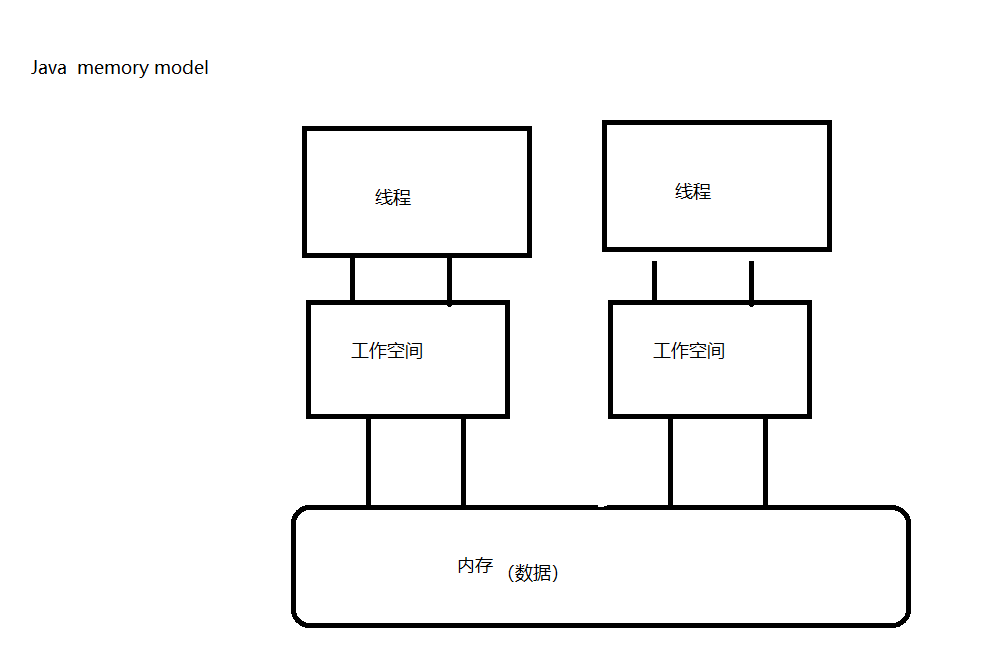
存放java线程的私有数据，这个数据就是执行下一条指令的地址

**3.5 Native method stack(本地方法栈)：**

与JVM的native 方法有关

**4 Java内存模型**

Java memory model，JMM(规范,抽象的模型)



1. 主内存：共享的信息
2. 工作内存：私有信息

* 基本数据类型的数据直接分配到工作内存，
* 引用类型的数据(引用的地址)存放在工作内存
* 引用的对象存放在堆中

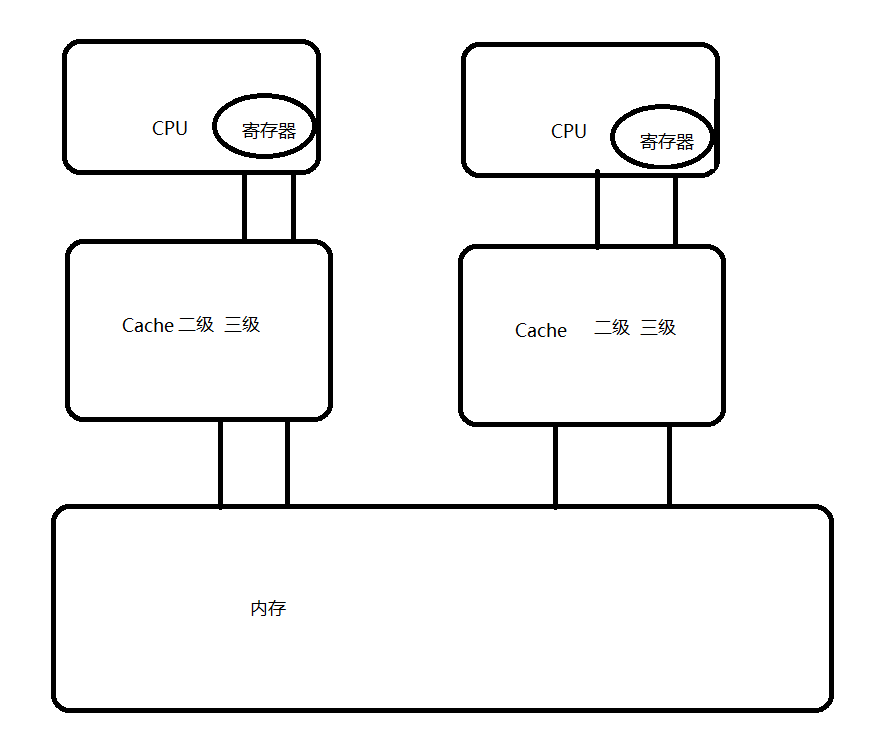
1. 工作方式：

A 线程修改私有数据，直接在工作空间修改

B 线程修改共享数据，把数据复制到工作空间中去，在工作空间中修改，修改完成以后，刷新内存中的数据

**5硬件内存架构与java内存模型**

1. 硬件架构

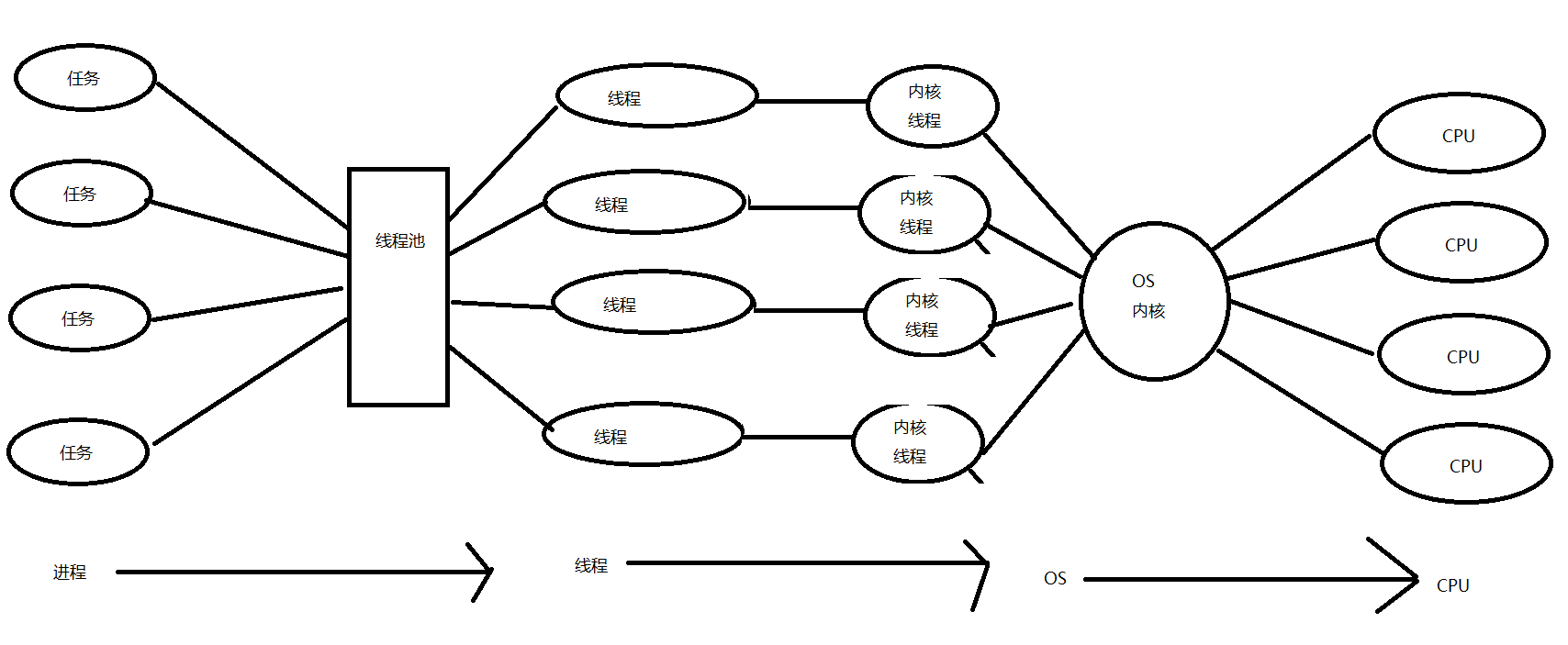


* + 1. CPU缓存的一致性问题：并发处理的不同步
    2. 解决方案：
       1. 总线加锁（） 降低CPU的吞吐量
       2. 缓存上的一致性协议（MESI）

当CPU在Cache中操作数据时，如果该数据是共享变量，该CPU从Cache中读数据到寄存器中，然后进行新修改，最终更新数据到内存中。

如果此时有其他CPU线程也要对该数据进行处理，会发现此时Cache line置无效，其他的CPU就没法从自己的Cache中获取数据处理，必须从内存中读数据。

1. Java线程与硬件处理器

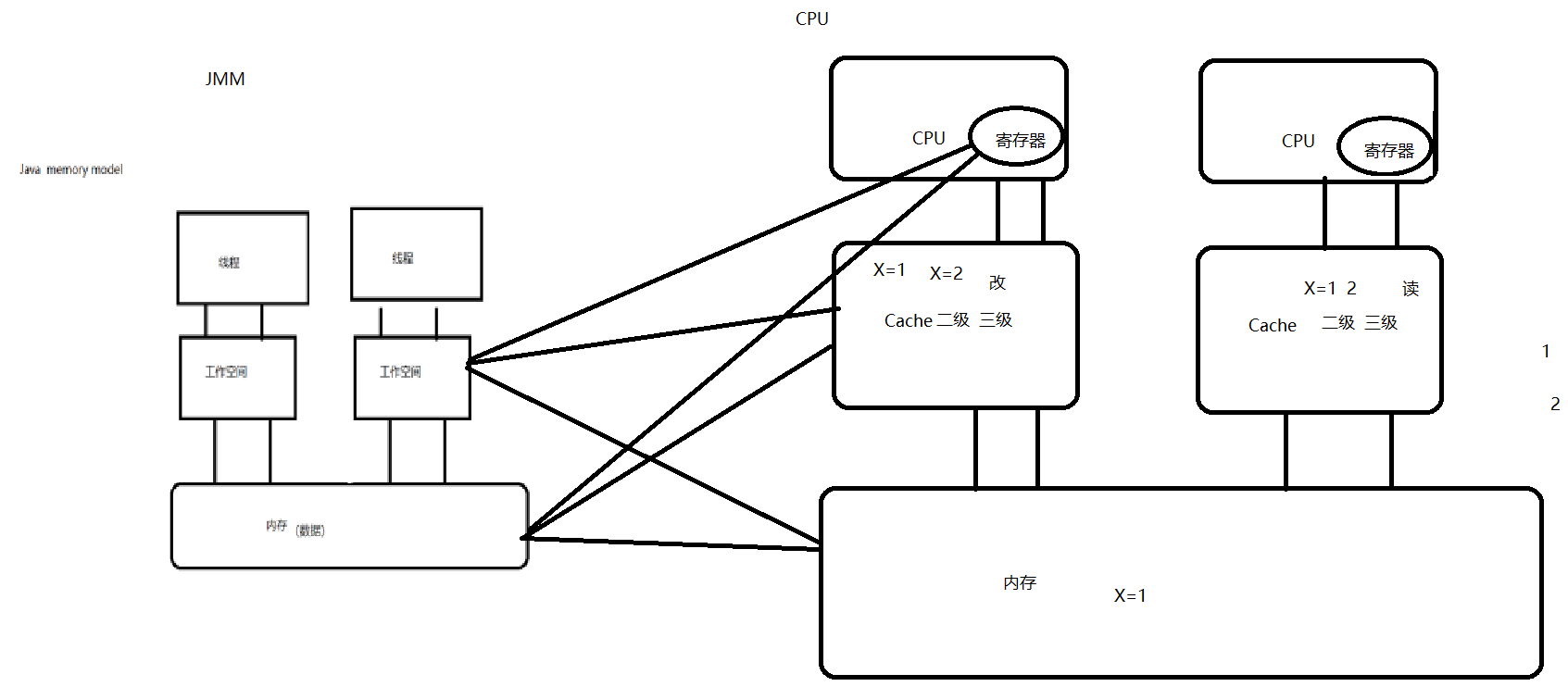
-

- 操作系统的内核：OS内核

- 多个内核线程可以是多个CPU处理，也可以是单个CPU分配时间片处理

- 由 进程 🡪 线程 🡪 OS 🡪 CPU

1. Java内存模型与硬件内存架构的关系

MESI(硬件级别做了处理)

交叉：数据的不一致

1. Java内存模型的必要性

Java内存模型的作用：规范内存数据和工作空间数据的交互

1. 并发编程的三个重要特性

原子性：不可分割 x=1

可见性：线程只能操作自己工作空间中的数据，不能操作其他线程工作空间中的数据

有序性：程序中的顺序不一定就是执行的顺序

编译重排序（代码优化）

指令重排序（指令优化）

重排的目的是提高效率

*重排有两个原则：  
 \* as-if-seria:单线程中重排后不影响执行的结果，多线程。  
 \* happens-before*

1. JMM对三个特征的保证
2. JMM与原子性
3. X=10 写 原子性 如果是私有数据具有原子性，如果是共享数据没原子性（读写）
4. Y=x 没有原子性
   1. 把数据X读到工作空间（原子性）
   2. 把X的值写到Y（原子性）
5. I++ 没有原子性
   1. 读i到工作空间
   2. +1；
   3. 刷新结果到内存
6. Z=z+1 没有原子性
   1. 读z到工作空间
   2. +1；
   3. 刷新结果到内存

多个原子性的操作合并到一起没有原子性

**保证方式：**

Synchronized

JUC Lock的lock

1. JMM与可见性

Volatile:在JMM模型上实现MESI协议

Synchronized:加锁

JUC包 里的Lock的lock

1. JMM与有序性

Volatile：

Synchronized：

Happens-before原则：

1. 程序次序原则：程序次序决定的结果不能变化
2. 锁定原则 ：后一次加锁必须等前一次解锁
3. Volatile原则：变量前加了Volatile，该变量所属位置不能变化(霸道原则)
4. 传递原则：A---B ---C A—C （A在B前执行，B在C前执行，所以A在C前执行）

总结：

JVM内存区域和JMM的关系

JMM和硬件的关系

JMM和并发编程三个重要特征（有序性 as-if-seria happens-before ）