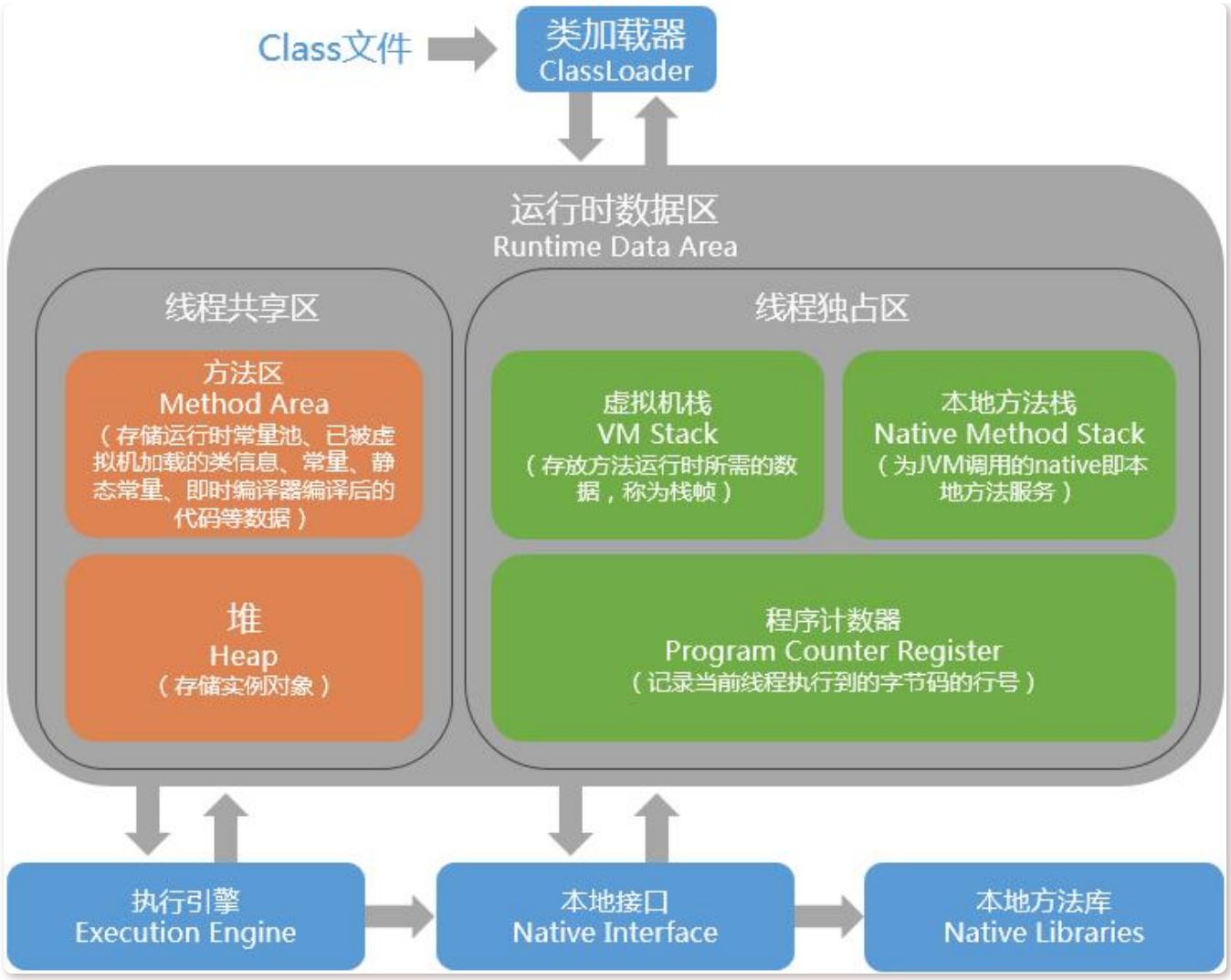
优秀引用

全面理解Java的内存模型（JMM）

终于有人把Java内存模型（JMM）说清楚了

1、引入

对于Java虚拟机的内存模型相信大家都不陌生了，对于每一个线程来说，栈是私有的，而堆是共享的，也就是说在栈中的变量（局部变量、方法定义参数、异常处理器参数）不会在线程之间共享，也就不会有内存可见性的问题，也不受内存模型的影响。

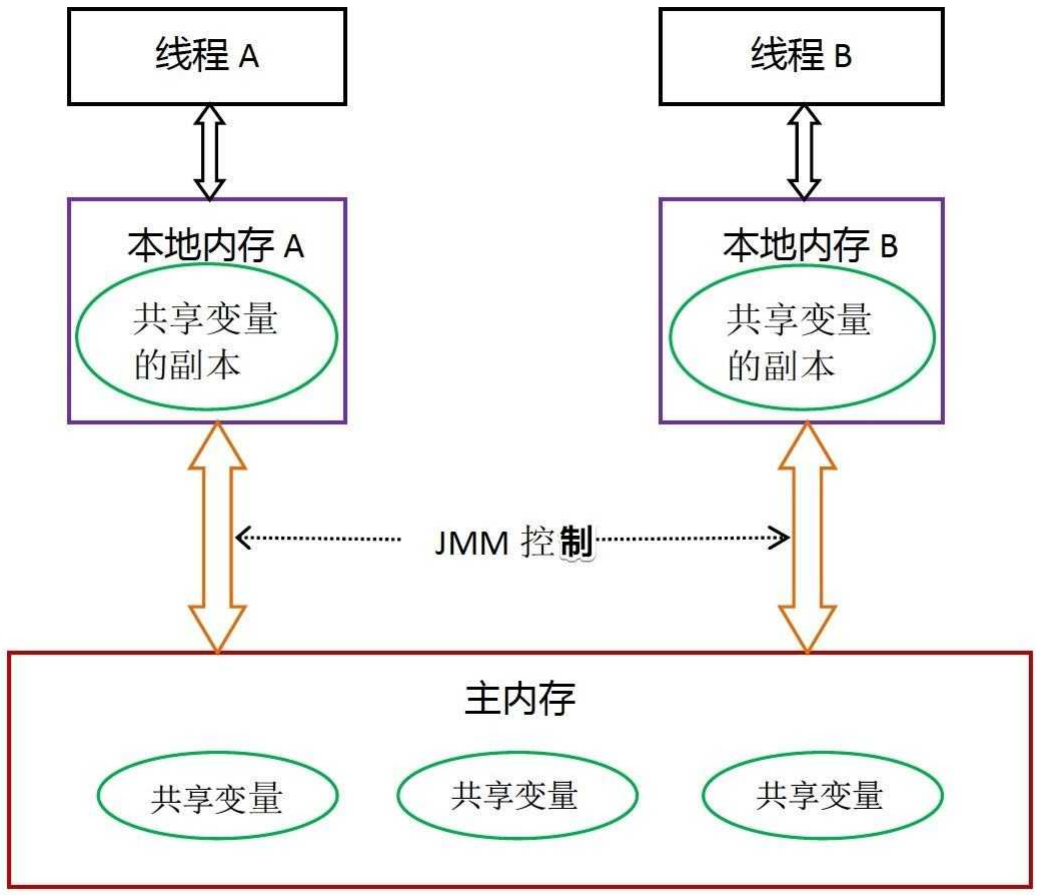


既然堆中的数据是线程共享的，那么一定会存在一个并发问题，但由于每个线程都有自己的本地缓存，导致线程之间读取的数据可能不是最新的，从而出现数据不一致的问题。JMM定义了一种内存模型，用于描述Java程序中多线程之间如何访问共享内存中的变量，从而解决了这个问题。

> 这里扯一嘴，在Java中使用的是共享内存并发模型，用于解决Java中线程间如何通信和数据同步的问题。

2、概述

JMM（Java Memory Model）是Java内存模型的缩写，是一种抽象的概念，定义了Java虚拟机如何在计算机内存中存储和访问Java对象的方法。JMM规范主要用于解决多线程访问共享内存时的**可见性、有序性和原子性问题**。下面是JMM的抽象示意图：



从上图中可以看出：

- 所有的共享变量都存在**主内存**中；
- **每个线程**都保存了一份该线程使用到的**共享变量的副本**。
- 如果线程A与线程B之间要**通信**的话，必须经历下面2个步骤：
    1. 线程A将本地内存A中**更新过**的共享变量**刷新到主内存**中去。
    2. 线程B**到主内存中**去**读取**线程A之前已经更新过的共享变量。

因此，线程A**无法直接访问**线程B的工作内存，那是因为**工作内存是线程独有的**，线程间通信必须借助主内存，这也是JMM中的规定。当主内存中的共享变量被某个线程更新时，**JMM会通过控制主内存与每个线程的本地内存之间的交互，来提供内存可见性保证**。因此通过JMM规范，有效的解决了以下问题：

- **可见性问题**：JMM保证对于一个线程对变量的修改，其他线程能够立即看到这个修改，从而避免了线程之间读取数据不一致的问题；
- **有序性问题**：JMM保证程序的执行顺序是有序的，即按照代码的编写顺序执行，从而避免了出现代码执行顺序混乱的问题；
- **原子性问题**：JMM保证对单个变量的读取和写入操作是原子性的，即不会出现数据竞争问题。

3、JMM内存模型的实现

3.1、简介

在Java中存在着许多并发相关的关键字，如 `valatile、synchronized、final` 等，而这些被大众所熟知的关键字便是Java内存模型封装了底层的实现后提供给开发人员进行使用。因此Java内存模型除了定义了一套规范，还提供了一系列的封装了底层实现的关键字供开发者使用。

3.2、原子性

原子性指的是指一个操作是不可中断的，即多线程环境下，操作不能被其他线程干扰。在Java中，最常用的便是使用关键字 `synchronized` 进行原子性的保证。

3.3、可见性

指当一个线程修改了某一个**共享变量的值**，其他线程**是否能够立即知道该变更**。对于Java中的普通共享变量是不保证可见性的，因为数据修改被写入内存的时机是不确定的，多线程并发下很可能会出现脏读的并发问题，因此便有着上面提及到的**线程私有的工作内存**。

每个线程的工作内存都保存了一份该线程使用到共享变量的副本，即主内存中的拷贝副本，在操作数据时只能在本地的副本中进行操作，不能直接对主内存中的数据进行操作。同时不同线程间也是不能直接进行通讯的，必须借助主内存来实现。

而在Java中提供了 `volatile、synchronized、final` 关键字来保证多线程操作时变量的可见性，其中 `volatile` 关键字提供了这么一个功能，那就是被其修饰的变量在被修改后可以立即同步到主内存中，被其他修饰的变量在每次使用之前都从主内存中刷新获取，从而保证可见性。

3.4、有序性

指程序是有序的按照一定的顺序运行，这一特性主要是针对于操作系统中对程序指令进行重排序造成的并发乱序问题。为了性能和便捷，在JMM中指明，再不改变程序执行结果的前提下，允许编译器和处理器对程序优化进行重排序。

在Java中，可以使用 `synchronized` 和 `volatile` 来保证多线程之间操作的有序性。实现方式有所区别：

- `volatile` 关键字会禁止指令重排；
- `synchronized` 关键字保证同一时刻只允许一条线程操作。

值得注意的是，按照一定的顺序并不一定是代码中的顺序，典型的例子便是线程的创建和执行，下面的示例代码便是先执行第四行后执行第二行：

```
1  Thread t1 = new Thread(() -> {
2    System.out.println("hello thread");
3  }, "t1");
4  t1.start();
```

4、相关面试题

4.1、你知道什么是Java内存模型JMM吗?

JMM（Java内存模型）是Java虚拟机规范中定义的一种内存模型，用于描述Java程序中多线程之间如何访问共享内存中的变量。JMM规定了Java程序中的所有变量都存储在主内存中，每个线程都有自己的本地缓存，线程对变量的读取和写入操作都必须在本地缓存和主内存之间进行同步，以保证数据的可见性、原子性和有序性。

JMM定义了一些规则和约束，如volatile关键字、synchronized关键字等，用于保证程序的正确性和稳定性。使用volatile关键字可以保证变量的可见性，即每个线程都能看到其他线程对该变量的最新修改。使用synchronized关键字可以保证代码块的原子性，即同一时刻只有一个线程能够执行该代码块。

4.2、JMM和volatile他们两个之间的关系是什么?

volatile是对JMM底层实现封装的关键字之一，是JMM的一种实现方式，用于修饰变量，表示该变量是可见性的，即任何对该变量的修改都会立即被其他线程所看到。volatile关键字是JMM的一种实现方式，它可以保证变量的可见性。

在Java中，使用volatile关键字修饰的变量会被存储在主内存中，每个线程访问该变量时都会从主内存中读取最新的值，而不是从线程的本地缓存中读取。因此，使用volatile关键字可以保证变量的可见性，避免出现数据不一致的问题。

同时，volatile关键字还具有禁止指令重排序的作用，即保证代码的有序性。由于JMM规定了线程之间的操作可能会存在重排序的情况，因此使用volatile关键字可以禁止指令重排序，保证程序的正确性和稳定性。

4.3、JMM有哪些特性/能说说JMM的三大特性吗?

JMM（Java内存模型）有三个特性，也被称为JMM的三大特性，分别是原子性、可见性和有序性。

1. **原子性**：原子性是指对于单个变量的读取和写入操作是原子性的，即不会出现数据竞争问题。在JMM中，原子性是通过synchronized关键字和volatile关键字来保证的。
2. **可见性**：可见性是指当一个线程对一个变量进行修改时，其他线程能够立即看到这个修改，从而避免了线程之间读取数据不一致的问题。在JMM中，可见性是通过volatile关键字来保证的。
3. **有序性**：有序性是指程序的执行顺序是有序的，即按照代码的编写顺序执行，从而避免了出现代码执行顺序混乱的问题。在JMM中，有序性是通过happens-before规则来保证的。

> 🌈
>
> happens-before规则定义了程序执行中各个操作之间的偏序关系，主要述说的是，某一段符合该规则的代码，前面代码得到的结果肯定对后方代码是可见的。

4.4、为什么要有JMM，它为什么会出现，作用和功能是什么

在Java多线程编程中，由于多个线程之间共享数据的情况很常见，因此需要一种机制来保证多线程之间的数据访问是正确的。在JVM中，由于每个线程都有自己的本地缓存，因此线程之间读取的数据可能不是最新的，从而出现数据不一致的问题。为了解决这个问题，Java引入了JMM。

JMM的作用就是规定了一些规则，保证多线程访问共享变量的正确性。JMM主要有以下几个功能：

1. **确定共享变量的可见性**：JMM规定，如果一个线程修改了共享变量的值，其他线程必须能够立即看到这个变化。
2. **确定操作的有序性**：JMM规定，如果一个线程对共享变量进行了多次修改，其他线程必须按照这些修改的顺序来看到这些变化。
3. **确定操作的原子性**：JMM规定，如果一个线程在修改共享变量的过程中，其他线程必须等待这个修改完成后再进行操作。

4.5、有了解过happens-before原则吗?

happens-before原则是Java内存模型中一个重要的概念，用于指定多线程程序中操作之间的顺序关系。

happens-before原则规定，如果操作A happens-before操作B，那么操作A的效果在操作B之前对其他线程是可见的。也就是说，如果在一个线程中操作A happens-before操作B，那么在另一个线程中看到操作A的效果一定是在看到操作B的效果之前。