https://mp.weixin.qq.com/s/ukvveN3QRhxpXWIo03QK6A

# Java内存模型

## 什么是内存模型

Java内存模型是根据英文Java Memory Model（JMM）翻译过来的。其实JMM并不像JVM内存结构一样是真实存在的。他只是一个抽象的概念。

Java内存模型的相关知识在 JSR-133: Java Memory Model and Thread Specification 中描述的。JMM是和多线程相关的，他描述了一组规则或规范，这个规范定义了一个线程对共享变量的写入时对另一个线程是可见的。

**Java内存模型（Java Memory Model ,JMM）就是一种符合内存模型规范的，屏蔽了各种硬件和操作系统的访问差异的，保证了Java程序在各种平台下对内存的访问都能得到一致效果的机制及规范**。目的是解决由于多线程通过共享内存进行通信时，存在的原子性、可见性（缓存一致性）以及有序性问题。

那么，我们这里就先来说说什么是所谓的内存模型规范、这里提到的原子性、可见性以及有序性又是什么东西？

### 原子性

**线程是CPU调度的基本单位**。CPU有时间片的概念，会根据不同的调度算法进行线程调度。所以在多线程场景下，就会发生原子性问题。因为线程在执行一个读改写操作时，在执行完读改之后，时间片耗完，就会被要求放弃CPU，并等待重新调度。这种情况下，读改写就不是一个原子操作。即存在原子性问题。

### 缓存一致性

在多核CPU，多线程的场景中，每个核都至少有一个L1 缓存。多个线程访问进程中的某个共享内存，且这多个线程分别在不同的核心上执行，则每个核心都会在各自的cache中保留一份共享内存的缓冲。由于多核是可以并行的，可能会出现多个线程同时写各自的缓存的情况，而各自的cache之间的数据就有可能不同。

在CPU和主存之间增加缓存，在多线程场景下就可能存在缓存一致性问题，也就是说，在多核CPU中，每个核的自己的缓存中，关于同一个数据的缓存内容可能不一致。

### 有序性

除了引入了时间片以外，由于处理器优化和指令重排等，CPU还可能对输入代码进行乱序执行，比如load->add->save 有可能被优化成load->save->add 。这就是有序性问题。

多CPU多级缓存导致的一致性问题、CPU时间片机制导致的原子性问题、以及处理器优化和指令重排导致的有序性问题等，都是硬件的不断升级导致的。那么，有没有什么机制可以很好的解决上面的这些问题呢？

最简单直接的做法就是废除处理器和处理器的优化技术、废除CPU缓存，让CPU直接和主存交互。但是，这么做虽然可以保证多线程下的并发问题。但是，这就有点因噎废食了。

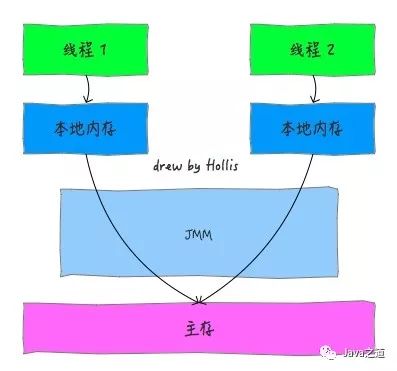
所以，为了保证并发编程中可以满足原子性、可见性及有序性。有一个重要的概念，那就是——内存模型。

为了保证共享内存的正确性（可见性、有序性、原子性），内存模型定义了共享内存系统中多线程程序读写操作行为的规范。通过这些规则来规范对内存的读写操作，从而保证指令执行的正确性。它与处理器有关、与缓存有关、与并发有关、与编译器也有关。他解决了CPU多级缓存、处理器优化、指令重排等导致的内存访问问题，保证了并发场景下的一致性、原子性和有序性。

针对上面的这些问题，不同的操作系统都有不同的解决方案，而Java语言为了屏蔽掉底层的差异，定义了一套属于Java语言的内存模型规范，即Java内存模型。

Java内存模型规定了所有的变量都存储在主内存中，每条线程还有自己的工作内存，线程的工作内存中保存了该线程中是用到的变量的主内存副本拷贝，线程对变量的所有操作都必须在工作内存中进行，而不能直接读写主内存。不同的线程之间也无法直接访问对方工作内存中的变量，线程间变量的传递均需要自己的工作内存和主存之间进行数据同步进行。

而**JMM就作用于工作内存和主存之间数据同步过程。他规定了如何做数据同步以及什么时候做数据同步。**



## Java内存模型的实现

了解Java多线程的朋友都知道，在Java中提供了一系列和并发处理相关的关键字，比如volatile、synchronized、final、concurrent包等。其实这些就是Java内存模型封装了底层的实现后提供给程序员使用的一些关键字。

在开发多线程的代码的时候，我们可以直接使用synchronized等关键字来控制并发，从来就不需要关心底层的编译器优化、缓存一致性等问题。所以，Java内存模型，除了定义了一套规范，还提供了一系列原语，封装了底层实现后，供开发者直接使用。

本文并不准备把所有的关键字逐一介绍其用法，因为关于各个关键字的用法，网上有很多资料。读者可以自行学习。本文还有一个重点要介绍的就是，我们前面提到，并发编程要解决原子性、有序性和一致性的问题，我们就再来看下，在Java中，分别使用什么方式来保证。

### 原子性

在Java中，为了保证原子性，提供了两个高级的字节码指令monitorenter和monitorexit。在synchronized的实现原理文章中，介绍过，这两个字节码，在Java中对应的关键字就是synchronized。

因此，在Java中可以使用synchronized来保证方法和代码块内的操作是原子性的。

### 可见性

Java内存模型是通过在变量修改后将新值同步回主内存，在变量读取前从主内存刷新变量值的这种依赖主内存作为传递媒介的方式来实现的。

Java中的volatile关键字提供了一个功能，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，被其修饰的变量在每次是用之前都从主内存刷新。因此，可以使用volatile来保证多线程操作时变量的可见性。

除了volatile，Java中的synchronized和final两个关键字也可以实现可见性。只不过实现方式不同，这里不再展开了。

### 有序性

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。**

好了，这里简单的介绍完了Java并发编程中解决原子性、可见性以及有序性可以使用的关键字。读者可能发现了，好像synchronized关键字是万能的，他可以同时满足以上三种特性，这其实也是很多人滥用synchronized的原因。

但是synchronized是比较影响性能的，虽然编译器提供了很多锁优化技术，但是也不建议过度使用。

### 面试如何回答

前面我介绍完了一些和Java内存模型有关的基础知识，只是基础，并不是全部，因为随便一个知识点还是都可以展开的，如volatile是如何实现可见性的？synchronized是如何实现有序性的？

但是，当面试官问你：能简单介绍下你理解的内存模型吗？

首先，先和面试官确认一下：您说的内存模型指的是JMM，也就是和并发编程有关的那一个吧？

在得到肯定答复后，再开始介绍（如果不是，那可能就要回答堆、栈、方法区哪些了….囧…）：

Java内存模型，其实是保证了Java程序在各种平台下对内存的访问都能够得到一致效果的机制及规范。目的是解决由于多线程通过共享内存进行通信时，存在的原子性、可见性（缓存一致性）以及有序性问题。

除此之外，Java内存模型还提供了一系列原语，封装了底层实现后，供开发者直接使用。如我们常用的一些关键字：synchronized、volatile以及并发包等。