二

56 讲一讲什么是 Java 内存模型?

本课时我们主要介绍什么是 Java 内存模型?

从本课时开始，我们会进入到 Java 内存模型的学习。如果你想了解 Java 并发的底层原理，那么 Java 内存模型的知识非常重要，同时也是一个分水岭，可以区分出我们是仅停留在如何使用并发工具，还是能更进一步，知其所以然。

容易混淆：JVM 内存结构 VS Java 内存模型

Java 作为一种面向对象的语言，有很多概念，从名称上看起来比较相似，比如 JVM 内存结构、Java 内存模型，这是两个截然不同的概念，但是很容易混淆。网络上也有不少讲 Java 内存模型的文章，其实写的是 JVM 内存结构。

所以我们就先从整体上概括一下这两者的主要作用：

- JVM 内存结构和 Java 虚拟机的运行时区域有关；
- Java 内存模型和 Java 的并发编程有关。

所以可以看出，这两个概念其实是有很大区别的。下面我们先来简要介绍一下 JVM 内存结构。

JVM 内存结构

我们都知道，Java 代码是要运行在虚拟机上的，而虚拟机在执行 Java 程序的过程中会把所管理的内存划分为若干个不同的数据区域，这些区域都有各自的用途。在《Java 虚拟机规范（Java SE 8）》中描述了 JVM 运行时内存区域结构可分为以下 6 个区。

堆区（Heap**）****：**堆是存储类实例和数组的，通常是内存中最大的一块。实例很好理解，比如 new Object() 就会生成一个实例；而数组也是保存在堆上面的，因为在 Java 中，数组也是对象。

虚拟机栈（Java Virtual Machine Stacks**）****：**它保存局部变量和部分结果，并

在方法调用和返回中起作用。

方法区（Method Area**）****：**它存储每个类的结构，例如运行时的常量池、字段和方法数据，以及方法和构造函数的代码，包括用于类初始化以及接口初始化的特殊方法。

本地方法栈（Native Method Stacks**）****：**与虚拟机栈基本类似，区别在于虚拟机栈为虚拟机执行的 Java 方法服务，而本地方法栈则是为 Native 方法服务。

程序计数器（The PC Register**）****：**是最小的一块内存区域，它的作用通常是保存当前正在执行的 JVM 指令地址。

运行时常量池（Run-Time Constant Pool）：**是方法区的一部分，包含多种常量，范围从编译时已知的数字到必须在运行时解析的方法和字段引用。

注意，以上是 Java 虚拟机规范，不同的虚拟机实现会各有不同，一般会遵守规范。

这里总结一下，JVM 内存结构是由 Java 虚拟机规范定义的，描述的是在 Java 程序执行过程中，由 JVM 管理的不同数据区域，各个区域有其特定的功能。官方的规范地址请点击这里查看。

从 Java 代码到 CPU 指令

看完了 JVM 内存结构，就让我们回到 Java 内存模型上来。我们都知道，编写的 Java 代码，最终还是要转化为 CPU 指令才能执行的。为了理解 Java 内存模型的作用，我们首先就来回顾一下从 Java 代码到最终执行的 CPU 指令的大致流程：

- 最开始，我们编写的 Java 代码，是 *.java 文件；
- 在编译（包含词法分析、语义分析等步骤）后，在刚才的 *.java 文件之外，会多出一个新的 Java 字节码文件（*.class）；
- JVM 会分析刚才生成的字节码文件（*.class），并根据平台等因素，把字节码文件转化为具体平台上的**机器指令；**
- 机器指令则可以直接在 CPU 上运行，也就是最终的程序执行。

为什么需要 JMM（Java Memory Model，Java 内存模型）

在更早期的语言中，其实是不存在内存模型的概念的。

所以程序最终执行的效果会依赖于具体的处理器，而不同的处理器的规则又不一样，不同的

处理器之间可能差异很大，因此同样的一段代码，可能在处理器 A 上运行正常，而在处理器 B 上运行的结果却不一致。同理，在没有 JMM 之前，不同的 JVM 的实现，也会带来不一样的“翻译”结果。

所以 Java 非常需要一个标准，来让 Java 开发者、编译器工程师和 JVM 工程师能够达成一致。达成一致后，我们就可以很清楚的知道什么样的代码最终可以达到什么样的运行效果，让多线程运行结果可以预期，这个标准就是 JMM**，**这就是需要 JMM 的原因。

我们本课时将突破 Java 代码的层次，开始往下钻研，研究从 Java 代码到 CPU 指令的这个转化过程要遵守哪些和并发相关的原则和规范，这就是 JMM 的重点内容。如果不加以规范，那么同样的 Java 代码，完全可能产生不一样的执行效果，那是不可接受的，这也违背了 Java “书写一次、到处运行”的特点。

JMM 是什么

有了上面的铺垫，下面我们就介绍一下究竟什么是 JMM。

JMM 是规范

JMM 是和多线程相关的**一组规范**，需要各个 JVM 的实现来遵守 JMM 规范，以便于开发者可以利用这些规范，更方便地开发多线程程序。这样一来，即便同一个程序在不同的虚拟机上运行，得到的程序结果也是一致的。

如果没有 JMM 内存模型来规范，那么很可能在经过了不同 JVM 的“翻译”之后，导致在不同的虚拟机上运行的结果不一样，那是很大的问题。

因此，JMM 与处理器、缓存、并发、编译器有关。它解决了 CPU 多级缓存、处理器优化、指令重排等导致的结果不可预期的问题。

JMM 是工具类和关键字的原理

之前我们使用了各种同步工具和关键字，包括 volatile、synchronized、Lock 等，其实它们的原理都涉及 JMM。正是 JMM 的参与和帮忙，才让各个同步工具和关键字能够发挥作用，帮我们开发出并发安全的程序。

比如我们写了关键字 synchronized，JVM 就会在 JMM 的规则下，“翻译”出合适的指令，包括限制指令之间的顺序，以便在即使发生了重排序的情况下，也能保证必要的“可见性”，这样一来，不同的 JVM 对于相同的代码的执行结果就变得可预期了，我们 Java 程序员就只需要用同步工具和关键字就可以开发出正确的并发程序了，这都要感谢 JMM。

JMM 里最重要 3 点内容，分别是：**重排序、原子性、内存可见性**。这三个部分的内容，后面我们会详细展开。

总结

以上就是本课时的内容了。本课时，我们先对 JVM 内存结构和 Java 内存模型这两个容易混淆的概念进行了辨析，然后从宏观层面讲解了什么是 Java 内存模型，接下来，我们的脚步从 Java 代码逐渐往下探索，解释了为什么需要 JMM 以及什么是 JMM。

上一页 下一页