深入理解Java内存模型（七）——总结

作者 程晓明 程晓明 关注 82 他的粉丝 发布于 2013年3月16日. 估计阅读时间: 11 分钟 来 ArchSummit北京2018 共同探讨机器学习、信息安全、微服务治理的关键点

处理器内存模型

顺序一致性内存模型是一个理论参考模型，JMM和处理器内存模型在设计时通常会把顺序一致性内存模型作为参照。JMM和处理器内存模型在设计时会对顺序一致性模型做一些放松，因为如果完全按照顺序一致性模型来实现处理器和JMM，那么很多的处理器和编译器优化都要被禁止，这对执行性能将会有很大的影响。

根据对不同类型读/写操作组合的执行顺序的放松，可以把常见处理器的内存模型划分为下面几种类型：

1. 放松程序中写-读操作的顺序，由此产生了total store ordering内存模型（简称为TSO）。
2. 在前面1的基础上，继续放松程序中写-写操作的顺序，由此产生了partial store order 内存模型（简称为PSO）。

3. 在前面1和2的基础上，继续放松程序中读-写和读-读操作的顺序，由此产生了relaxed memory order内存模型（简称为RMO）和PowerPC内存模型。

注意，这里处理器对读/写操作的放松，是以两个操作之间不存在数据依赖性为前提的（因为处理器要遵守as-if-serial语义，处理器不会对存在数据依赖性的两个内存操作做重排序）。

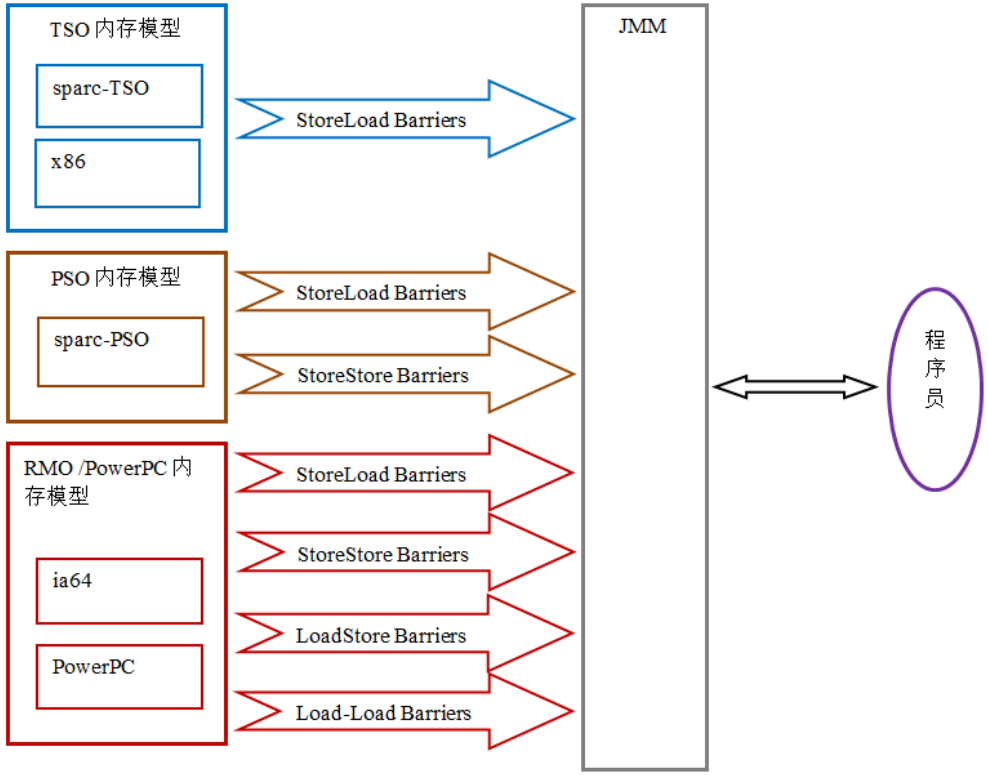
下面的表格展示了常见处理器内存模型的细节特征：

内存模型名称	对应的处理器	Store-Load重排序	Store-Store重排序	Load-Load和Load-Store重排序	可以更早读取到其它处理器的写	可以更早读取到当前处理器的写
TSO	sparc-TSO X64	Y				Y
PSO	sparc-PSO	Y	Y			Y
RMO	ia64	Y	Y	Y		Y
PowerPC	PowerPC	Y	Y	Y	Y	Y

在这个表格中，我们可以看到所有处理器内存模型都允许写-读重排序，原因在第一章以说明过：它们都使用了写缓存区，写缓存区可能导致写-读操作重排序。同时，我们可以看到这些处理器内存模型都允许更早读到当前处理器的写，原因同样是因为写缓存区：由于写缓存区仅对当前处理器可见，这个特性导致当前处理器可以比其他处理器先看到临时保存在自己的写缓存区中的写。

上面表格中的各种处理器内存模型，从上到下，模型由强变弱。越是追求性能的处理器，内存模型设计的会越弱。因为这些处理器希望内存模型对它们的束缚越少越好，这样它们就可以做尽可能多的优化来提高性能。

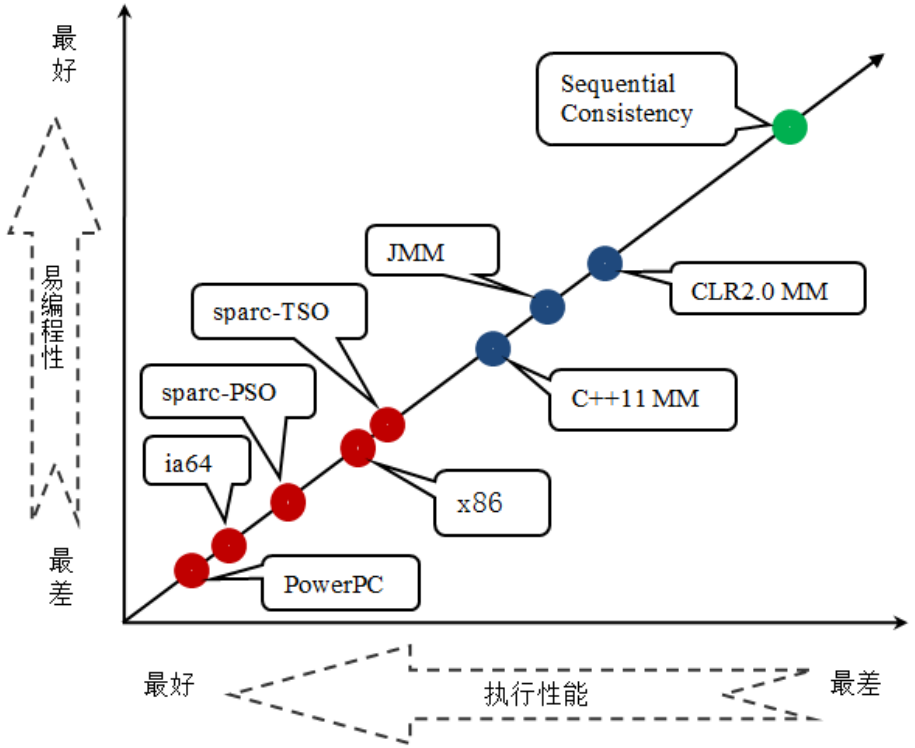
由于常见的处理器内存模型比JMM要弱，java编译器在生成字节码时，会在执行指令序列的适当位置插入内存屏障来限制处理器的重排序。同时，由于各种处理器内存模型的强弱并不相同，为了在不同的处理器平台向程序员展示一个一致的内存模型，JMM在不同的处理器中需要插入的内存屏障的数量和种类也不相同。下图展示了JMM在不同处理器内存模型中需要插入的内存屏障的示意图：



如上图所示，JMM屏蔽了不同处理器内存模型的差异，它在不同的处理器平台之上为java程序员呈现了一个一致的内存模型。

JMM，处理器内存模型与顺序一致性内存模型之间的关系

JMM是一个语言级的内存模型，处理器内存模型是硬件级的内存模型，顺序一致性内存模型是一个理论参考模型。下面是语言内存模型，处理器内存模型和顺序一致性内存模型的强弱对比示意图：



从上图我们可以看出：常见的4种处理器内存模型比常用的3中语言内存模型要弱，处理器内存模型和语言内存模型都比顺序一致性内存模型要弱。同处理器内存模型一样，越是追求执行性能的语言，内存模型设计的会越弱。

JMM的设计

从JMM设计者的角度来说，在设计JMM时，需要考虑两个关键因素：

- 程序员对内存模型的使用。程序员希望内存模型易于理解，易于编程。程序员希望基于一个强内存模型来编写代码。
- 编译器和处理器对内存模型的实现。编译器和处理器希望内存模型对它们的束缚越少越好，这样它们就可以做尽可能多的优化来提高性能。编译器和处理器希望实现一个弱内存模型。

由于这两个因素互相矛盾，所以JSR-133专家组在设计JMM时的核心目标就是找到一个好的平衡点：一方面要为程序员提供足够强的内存可见性保证；另一方面，对编译器和处理器的限制要尽可能的放松。下面让我们看看JSR-133是如何实现这一目标的。

为了具体说明，请看前面提到过的计算圆面积的示例代码：

```
double pi  = 3.14;    //A
double r   = 1.0;     //B
double area = pi * r * r; //C
```

上面计算圆的面积的示例代码存在三个happens- before关系：

1. A happens- before B；
2. B happens- before C；
3. A happens- before C；

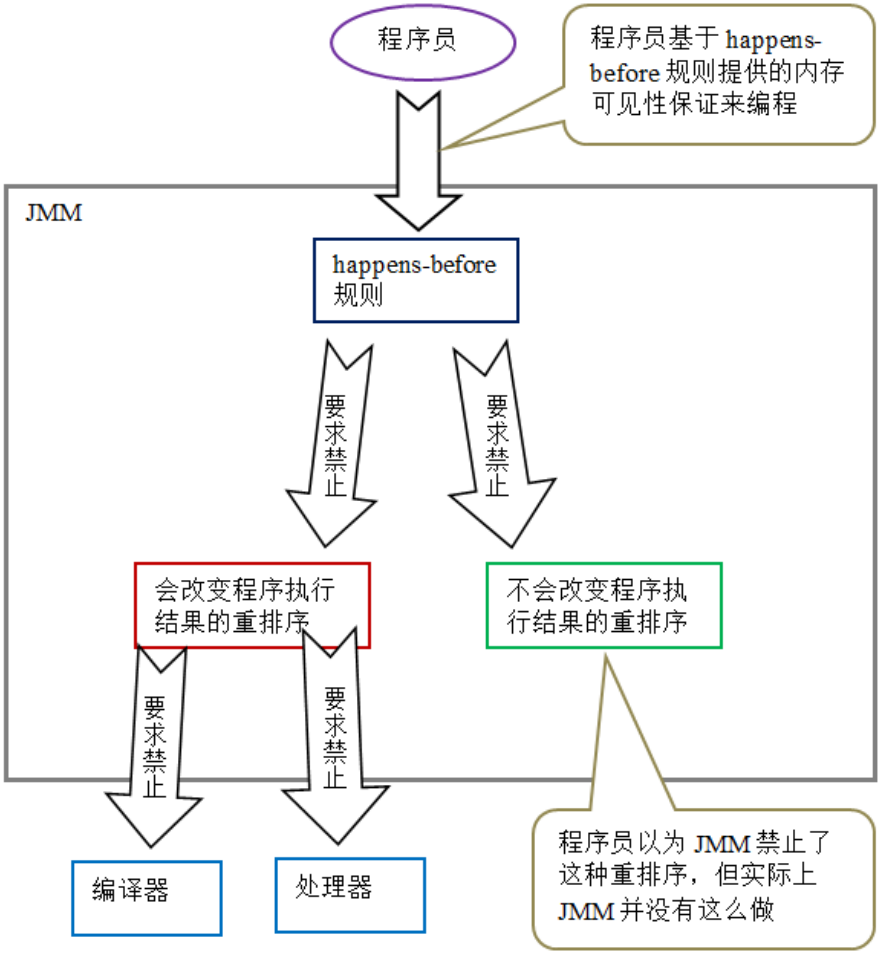
由于A happens- before B，happens- before的定义会要求：A操作执行的结果要对B可见，且A操作的执行顺序排在B操作之前。 但是从程序语义的角度来说，对A和B做重排序即不会改变程序的执行结果，也还能提高程序的执行性能（允许这种重排序减少了对编译器和处理器优化的束缚）。也就是说，上面这3个happens- before关系中，虽然2和3是必需要的，但1是不必要的。因此，JMM把happens- before要求禁止的重排序分为了下面两类：

- 会改变程序执行结果的重排序。
- 不会改变程序执行结果的重排序。

JMM对这两种不同性质的重排序，采取了不同的策略：

- 对于会改变程序执行结果的重排序，JMM要求编译器和处理器必须禁止这种重排序。
- 对于不会改变程序执行结果的重排序，JMM对编译器和处理器不作要求（JMM允许这种重排序）。

下面是JMM的设计示意图：



从上图可以看出两点：

- JMM向程序员提供的happens- before规则能满足程序员的需求。JMM的happens- before规则不但简单易懂，而且也向程序员提供了足够强的内存可见性保证（有些内存可见性保证其实并不一定真实存在，比如上面的A happens- before B）。

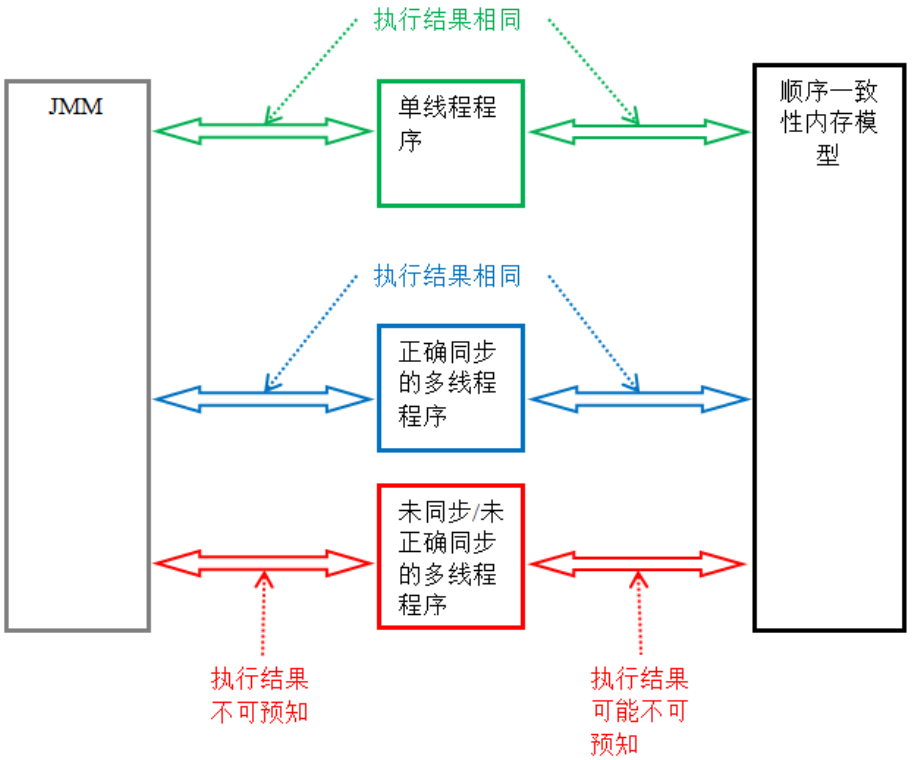
- JMM对编译器和处理器的束缚已经尽可能的少。从上面的分析我们可以看出，JMM其实是在遵循一个基本原则：只要不改变程序的执行结果（指的是单线程程序和正确同步的多线程程序），编译器和处理器怎么优化都行。比如，如果编译器经过细致的分析后，认定一个锁只会被单个线程访问，那么这个锁可以被消除。再比如，如果编译器经过细致的分析后，认定一个volatile变量仅仅只会被单个线程访问，那么编译器可以把这个volatile变量当作一个普通变量来对待。这些优化既不会改变程序的执行结果，又能提高程序的执行效率。

JMM的内存可见性保证

Java程序的内存可见性保证按程序类型可以分为下列三类：

1. 单线程程序。单线程程序不会出现内存可见性问题。编译器，runtime和处理器会共同确保单线程程序的执行结果与该程序在顺序一致性模型中的执行结果相同。
2. 正确同步的多线程程序。正确同步的多线程程序的执行将具有顺序一致性（程序的执行结果与该程序在顺序一致性内存模型中的执行结果相同）。这是JMM关注的重点，JMM通过限制编译器和处理器的重排序来为程序员提供内存可见性保证。
3. 未同步/未正确同步的多线程程序。JMM为它们提供了最小安全性保障：线程执行时读取到的值，要么是之前某个线程写入的值，要么是默认值（0，null，false）。

下图展示了这三类程序在JMM中与在顺序一致性内存模型中的执行结果的异同：



只要多线程程序是正确同步的，JMM保证该程序在任意的处理器平台上的执行结果，与该程序在顺序一致性内存模型中的执行结果一致。

JSR-133对旧内存模型的修补

JSR-133对JDK5之前的旧内存模型的修补主要有两个：

- 增强volatile的内存语义。旧内存模型允许volatile变量与普通变量重排序。JSR-133严格限制volatile变量与普通变量的重排序，使volatile的写-读和锁的释放-获取具有相同的内存语义。
- 增强final的内存语义。在旧内存模型中，多次读取同一个final变量的值可能会不相同。为此，JSR-133为final增加了两个重排序规则。现在，final具有了初始化安全性。

参考文献

1. Computer Architecture: A Quantitative Approach, 4th Edition
2. Shared memory consistency models: A tutorial
3. Intel® Itanium® Architecture Software Developer’s Manual Volume 2: System Architecture
4. Concurrent Programming on Windows
5. JSR 133 (Java Memory Model) FAQ
6. The JSR-133 Cookbook for Compiler Writers
7. Java theory and practice: Fixing the Java Memory Model, Part 2

关于作者

程晓明，Java软件工程师，国家认证的系统分析师、信息项目管理师。专注于并发编程，就职于富士通南大。个人邮箱：asst2003@163.com。

7 讨论