深入理解Java内存模型（二）——重排序

作者 程晓明 程晓明 关注 82 他的粉丝 发布于 2013年1月27日. 估计阅读时间: 9 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 41 讨论

数据依赖性

如果两个操作访问同一个变量，且这两个操作中有一个为写操作，此时这两个操作之间就存在数据依赖性。数据依赖分下列三种类型：

名称	代码示例	说明
写后读	a = 1;b = a;	写一个变量之后，再读这个位置。
写后写	a = 1;a = 2;	写一个变量之后，再写这个变量。
读后写	a = b;b = 1;	读一个变量之后，再写这个变量。

上面三种情况，只要重排序两个操作的执行顺序，程序的执行结果将会被改变。

前面提到过，编译器和处理器可能会对操作做重排序。编译器和处理器在重排序时，会遵守数据依赖性，编译器和处理器不会改变存在数据依赖关系的两个操作的执行顺序。

注意，这里所说的数据依赖性仅针对单个处理器中执行的指令序列和单个线程中执行的操作，不同处理器之间和不同线程之间的数据依赖性不被编译器和处理器考虑。

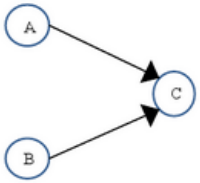
as-if-serial语义

as-if-serial语义的意思指：不管怎么重排序（编译器和处理器为了提高并行度），（单线程）程序的执行结果不能被改变。编译器，runtime 和处理器都必须遵守as-if-serial语义。

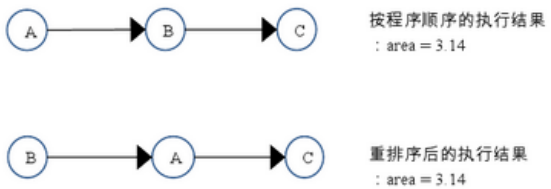
为了遵守as-if-serial语义，编译器和处理器不会对存在数据依赖关系的操作做重排序，因为这种重排序会改变执行结果。但是，如果操作之间不存在数据依赖关系，这些操作可能被编译器和处理器重排序。为了具体说明，请看下面计算圆面积的代码示例：

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

上面三个操作的数据依赖关系如下图所示：



如上图所示，A和C之间存在数据依赖关系，同时B和C之间也存在数据依赖关系。因此在最终执行的指令序列中，C不能被重排序到A和B的前面（C排到A和B的前面，程序的结果将会被改变）。但A和B之间没有数据依赖关系，编译器和处理器可以重排序A和B之间的执行顺序。下图是该程序的两种执行顺序：



as-if-serial语义把单线程程序保护了起来，遵守as-if-serial语义的编译器，runtime 和处理器共同为编写单线程程序的程序员创建了一个幻觉：单线程程序是按程序的顺序来执行的。as-if-serial语义使单线程程序员无需担心重排序会干扰他们，也无需担心内存可见性问题。

程序顺序规则

根据happens- before的程序顺序规则，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；

这里的第3个happens- before关系，是根据happens- before的传递性推导出来的。

这里A happens- before B，但实际执行时B却可以排在A之前执行（看上面的重排序后的执行顺序）。在第一章提到过，如果A happens- before B，JMM并不要求A一定要在B之前执行。JMM仅仅要求前一个操作（执行的结果）对后一个操作可见，且前一个操作按顺序排在第二个操作之前。这里操作A的执行结果不需要对操作B可见；而且重排序操作A和操作B后的执行结果，与操作A和操作B按happens- before顺序执行的结果一致。在这种情况下，JMM会认为这种重排序并不非法（not illegal），JMM允许这种重排序。

在计算机中，软件技术和硬件技术有一个共同的目标：在不改变程序执行结果的前提下，尽可能的开发并行度。编译器和处理器遵从这一目标，从happens- before的定义我们可以看出，JMM同样遵从这一目标。

重排序对多线程的影响

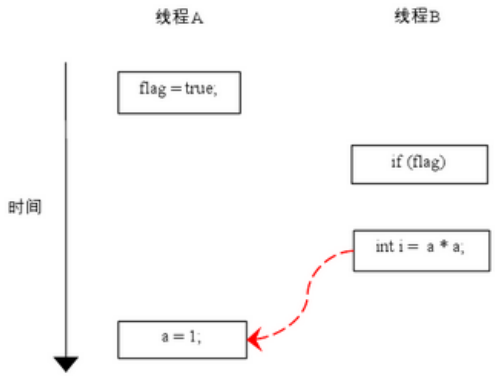
现在让我们来看看，重排序是否会改变多线程程序的执行结果。请看下面的示例代码：

```
class ReorderExample {
int a = 0;
boolean flag = false;

public void writer() {
    a = 1;                    //1
    flag = true;              //2
}

Public void reader() {
    if (flag) {                 //3
        int i =  a * a;         //4
        ……
    }
}
}
```

flag变量是个标记，用来标识变量a是否已被写入。这里假设有两个线程A和B，A首先执行writer()方法，随后B线程接着执行reader()方法。线程B在执行操作4时，能否看到线程A在操作1对共享变量a的写入？

答案是：不一定能看到。

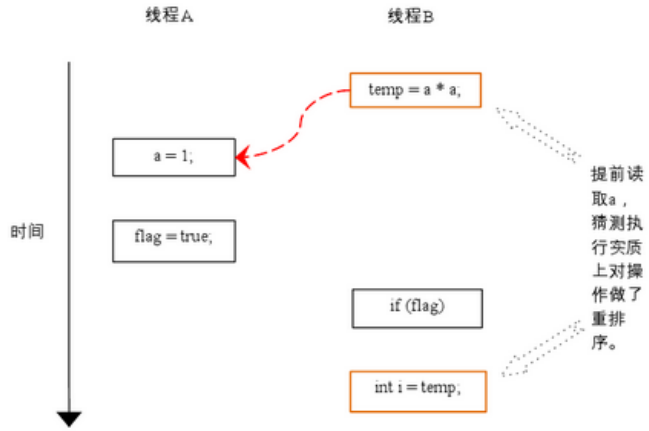
由于操作1和操作2没有数据依赖关系，编译器和处理器可以对这两个操作重排序；同样，操作3和操作4没有数据依赖关系，编译器和处理器也可以对这两个操作重排序。让我们先来看看，当操作1和操作2重排序时，可能会产生什么效果？请看下面的程序执行时序图：



如上图所示，操作1和操作2做了重排序。程序执行时，线程A首先写标记变量flag，随后线程B读这个变量。由于条件判断为真，线程B将读取变量a。此时，变量a还根本没有被线程A写入，在这里多线程程序的语义被重排序破坏了！

※注：本文统一用红色的虚箭线表示错误的读操作，用绿色的虚箭线表示正确的读操作。

下面再让我们看看，当操作3和操作4重排序时会产生什么效果（借助这个重排序，可以顺便说明控制依赖性）。下面是操作3和操作4重排序后，程序的执行时序图：



在程序中，操作3和操作4存在控制依赖关系。当代码中存在控制依赖性时，会影响指令序列执行的并行度。为此，编译器和处理器会采用猜测（Speculation）执行来克服控制相关性对并行度的影响。以处理器的猜测执行为例，执行线程B的处理器可以提前读取并计算a*a，然后把计算结果临时保存到一个名为重排序缓冲（reorder buffer ROB）的硬件缓存中。当接下来操作3的条件判断为真时，就把该计算结果写入变量i中。

从图中我们可以看出，猜测执行实质上对操作3和4做了重排序。重排序在这里破坏了多线程程序的语义！

在单线程程序中，对存在控制依赖的操作重排序，不会改变执行结果（这也是as-if-serial语义允许对存在控制依赖的操作做重排序的原因）；但在多线程程序中，对存在控制依赖的操作重排序，可能会改变程序的执行结果。

参考文献

1. Computer Architecture: A Quantitative Approach, 4th Edition
2. Concurrent Programming on Windows
3. Concurrent Programming in Java™: Design Principles and Pattern
4. JSR-133: Java Memory Model and Thread Specification

5. JSR 133 (Java Memory Model) FAQ

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。

感谢张龙对本文的审校。

给InfoQ中文站投稿或者参与内容翻译工作，请邮件至editors@cn.infoq.com。也欢迎大家通过新浪微博（@InfoQ）或者腾讯微博（@InfoQ）关注我们，并与我们的编辑和其他读者朋友交流。