第十六章 原子变量与非阻塞同步机制

#### 一．什么是内存模型，为什么需要它

1. 重排序

在编译器中生成的指令顺序，可以与源代码中的顺序不同；处理器可以采用乱序或并行等方式来执行指令。

2. 编译器会把变量保存在寄存器中而不是内存中；缓存可能会改变将写入变量提交到主内存的次序。

3. 保存在处理器缓存中的值，对于其他处理器是不可见的。

#### 二．平台的内存模型

1. 平台的内存模型

每个处理器都拥有自己的缓存，并且定期地与主内存进行协调，在不同的处理器架构中提供了不同级别的缓存一致性，即允许不同的处理器在任意时刻从同一个存储位置上看到不同的值。JVM通过在适当的位置上插入内存栅栏来屏蔽在JMM与底层平台内存模型之间的差异。

Java程序不需要指定内存栅栏的位置，而只需通过正确地使用同步来找出何时将访问共享状态。

2. 重排序

各种使操作延迟或者看似乱序执行的不同原因，都可以归为重排序，内存级的重排序会使程序的行为变得不可预测。

3. Java内存模式简介

　　Java内存模型是通过各种操作来定义的，包括变量的读/写操作，监视器的加锁和释放操作，以及线程的启动和合并操作

　　JMM为程序中所有的操作定义了一个偏序关系，称为Happens-Before，使在正确同步的程序中不存在数据竞争（缺乏Happens-Before关系，那么JVM可以对它们任意地重排序）

* 程序顺序规则。如果程序中操作A在操作B之前，那么在线程中A操作将在B操作之前执行
* 监视器锁规则。在监视器锁上的解锁操作必须在同一个监视器锁上的加锁操作之前执行。（显式锁和内置锁在加锁和解锁等操作上有着相同的内存语义）
* volatile变量规则。对volatile变量的写入操作必须在对该变量的读操作之前执行。（原子变量与volatile变量在读操作和写操作上有着相同的语义）
* 线程启动规则。在线程上对Thread.start的调用必须在该线程中执行任何操作之前执行
* 线程结束规则。线程中的任何操作都必须在其他线程检测到该线程已经结束之前执行，或者从Thread.join中成功返回，或者在调用Thread.isAlive时返回false中断规则。当一个线程在另一个线程上调用interrupt时，必须在被中断线程检测到interrupt调用之前执行（通过抛出InterruptException，或者调用isInterrupted和interrupted）
* 终结器规则。对象的构造函数必须在启动该对象的终结器之前执行完成传递性。如果操作A在操作B之前执行，并且操作B在操作C之前执行，那么操作A必须在操作C之前执行。

#### 三．发布

1. 造成不正确发布的真正原因："发布一个共享对象"与"另一个线程访问该对象"之间缺少一种Happens-Before的关系

2. 不安全的发布

除了不可变对象以外，使用被另一个线程初始化的对象通常都是不安全的，除非对象的发布操作是在使用该对象的线程开始使用之前执行

3. 安全发布

例：BlockingQueue的同步机制保证put在take后执行，A线程放入对象能保证B线程取出时是安全的

　　借助于类库中现在的同步容器、使用锁保护共享变量、或都使用共享的volatile类型变量，都可以保证对该变量的读取和写入是按照happens-before排序的。

happens-before事实上可以比安全发布承诺更强的可见性与排序性。

4. 安全初始化模式

（1）方式一：加锁保证可见性与排序性，存在性能问题。

（2）方式二：提前初始化（定义属性的时候初始化），可能造成浪费资源。

（3）方式三：延迟初始化，建议。

（4）方式四：双重加锁机制（之前是糟糕的模式），注意保证volatile类型，否则出现一致性问题。

#### 四．初始化过程中的安全性

1. 如果能确保初始化过程的安全性，被正确构造的不可变对象在没有同步的情况下也能安全地在多个线程之间共享

2. 如果不能确保初始化的安全性，一些本应为不可变对象的值将会发生改变

3. 初始化安全性只能保证通过final域可达的值从构造过程完成时可见性。对于通过非final域可达的值，或者在构成过程完成后可能改变的值，必须采用同步来确保可见性