

08 | 管程：并发编程的万能钥匙

王宝令 2019-03-16



00:00

12:01

讲述：王宝令

大小：11.02M

并发编程这个技术领域已经发展了半个世纪了，相关的理论和技术纷繁复杂。那有没有一种核心技术可以很方便地解决我们的并发问题呢？这个问题如果让我选择，我一定会选择**管程**技术。Java 语言在 1.5 之前，提供的唯一的并发原语就是管程，而且 1.5 之后提供的 SDK 并发包，也是以管程技术为基础的。除此之外，C/C++、C# 等高级语言也都支持管程。

可以这么说，管程就是一把解决并发问题的万能钥匙。

什么是管程

不知道你是否曾思考过这个问题：为什么 Java 在 1.5 之前仅仅提供了 synchronized 关键字及 wait()、notify()、notifyAll() 这三个看似从天而降的方法？在刚接触 Java 的时候，我以为它会提供信号量这种编程原语，因为操作系统原理课程告诉我，用信号量能解决所有并发问题，结果我发现不是。后来我找到了原因：Java 采用的是管程技术，synchronized 关键字及 wait()、notify()、notifyAll() 这三个方法都是管程的组成部分。而管程和信号量是等价的，所谓等价指的是用管程能够实现信号量，也能用信号量实现管程。但是管程更容易使用，所以 Java 选择了管程。

管程，对应的英文是 **Monitor**，很多 Java 领域的同学都喜欢将其翻译成“监视器”，这是直译。操作系统领域一般都翻译成“管程”，这个是意译，而我自己也更倾向于使用“管程”。

所谓**管程**，指的是**管理共享变量以及对共享变量的操作过程，让他们支持并发**。翻译为 Java 领域的语言，就是**管理类的成员变量和成员方法，让这个类是线程安全的**。那管程是怎么管的呢？

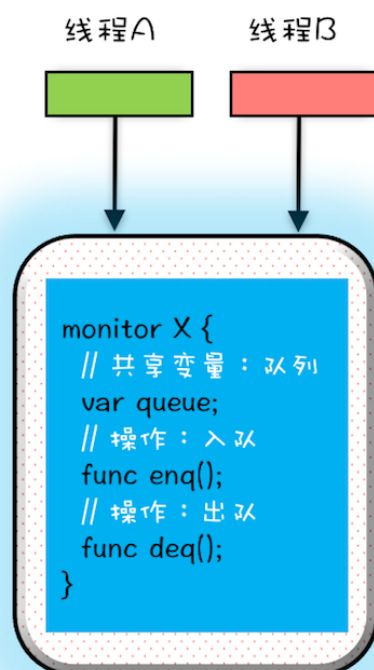
MESA 模型

在管程的发展史上，先后出现过三种不同的管程模型，分别是：Hasen 模型、Hoare 模型和 MESA 模型。其中，现在广泛应用的是 MESA 模型，并且 Java 管程的实现参考的也是 MESA 模型。所以今天我们重点介绍一下 MESA 模型。

在并发编程领域，有两大核心问题：**一个是互斥**，即同一时刻只允许一个线程访问共享资源；另一个是**同步**，即线程之间如何通信、协作。这两大问题，管程都是能够解决的。

我们先来看看管程是如何解决**互斥**问题的。

管程解决互斥问题的思路很简单，就是将共享变量及其对共享变量的操作统一封装起来。在下图中，管程 X 将共享变量 queue 这个队列和相关的操作入队 enq()、出队 deq() 都封装起来了；线程 A 和线程 B 如果想访问共享变量 queue，只能通过调用管程提供的 enq()、deq() 方法来实现；enq()、deq() 保证互斥性，只允许一个线程进入管程。不知你有没有发现，管程模型和面向对象高度契合的。估计这也是 Java 选择管程的原因吧。而我在前面章节介绍的互斥锁用法，其背后的模型其实它就是它。



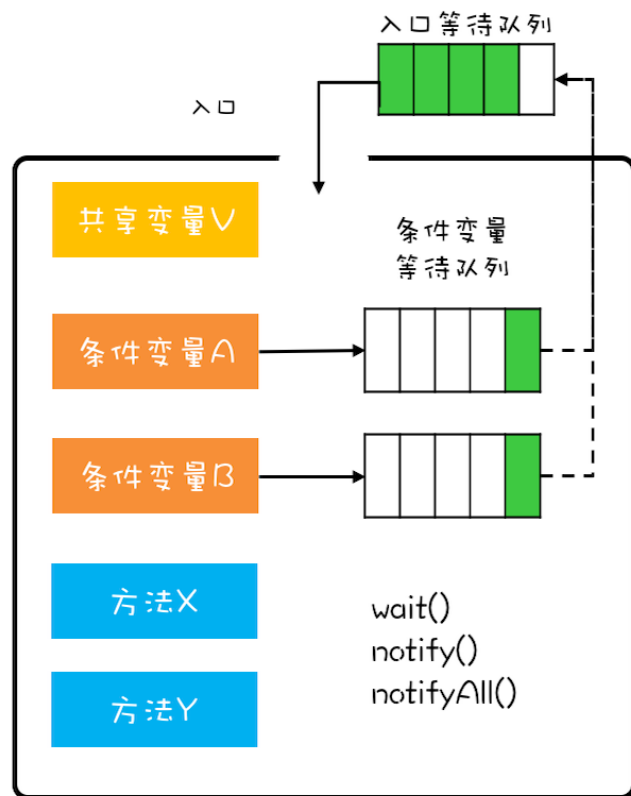
管程模型的代码化语义

那管程如何解决线程间的**同步**问题呢？

这个就比较复杂了，不过你可以借鉴一下我们曾经提到过的就医流程，它可以帮助你快速地理解这个问题。为进一步便于你理解，在下面，我展示了一幅 MESA 管程模型示意图，它详细描述了 MESA 模型的主要组成部分。

在管程模型里，共享变量和对共享变量的操作是被封装起来的，图中最外层的框就代表封装的意思。框的上面只有一个入口，并且在入口旁边还有一个入口等待队列。当多个线程同时试图进入管程内部时，只允许一个线程进入，其他线程则在入口等待队列中等待。这个过程类似就医流程的分诊，只允许一个患者就诊，其他患者都在门口等待。

管程里还引入了条件变量的概念，而且**每个条件变量都对应有一个等待队列**，如下图，条件变量 A 和条件变量 B 分别都有自己的等待队列。



MESA 管程模型

那条件变量和等待队列的作用是什么呢？其实就是解决线程同步问题。你也可以结合上面提到的入队出队例子加深一下理解。


假设有个线程 T1 执行出队操作，不过需要注意的是执行出队操作，有个前提条件，就是队列不能是空的，而队列不空这个前提条件就是管程里的条件变量。如果线程 T1 进入管程后恰好发现队列是空的，那怎么办呢？等待啊，去哪里等呢？就去条件变量对应的等待队列里面等。此时线程 T1 就去“队列不空”这个条件变量的等待队列中等待。这个过程类似于大夫发现你要去验个血，于是给你开了个验血的单子，你呢就去验血的队伍里排队。线程 T1 进入条件变量的等待队列后，是允许其他线程进入管程的。这和你去验血的时候，医生可以给其他患者诊治，道理都是一样的。

再假设之后另外一个线程 T2 执行入队操作，入队操作执行成功之后，“队列不空”这个条件对于线程 T1 来说已经满足了，此时线程 T2 要通知 T1，告诉它需要的条件已经满足了。当线程 T1 得到通知后，会从等待队列里面出来，但是出来之后不是马上执行，而是重新进入到入口等待队列里面。这个过程类似你验血完，回来找大夫，需要重新分诊。

条件变量及其等待队列我们讲清楚了，下面再说说 wait()、notify()、notifyAll() 这三个操作。前面提到线程 T1 发现“队列不空”这个条件不满足，需要进到对应的等待队列里等待。这个过程就是通过调用 wait() 来实现的。如果我们用对象 A 代表“队列不空”这个条件，那么线程 T1 需要调用 A.wait()。同理当“队列不空”这个条件满足时，线程 T2 需要调用 A.notify() 来通知 A 等待队列中的一个线程，此时这个队列里面只有线程 T1。至于 notifyAll() 这个方法，它可以通知等待队列中的所有线程。

这里我还是来一段代码再次说明一下吧。下面的代码实现的是一个阻塞队列，阻塞队列有两个操作分别是入队和出队，这两个方法都是先获取互斥锁，类比管程模型中的入口。

1. 对于入队操作，如果队列已满，就需要等待直到队列不满，所以这里用了 `notFull.await()`。
2. 对于出队操作，如果队列为空，就需要等待直到队列不空，所以就用了 `notEmpty.await()`。
3. 如果入队成功，那么队列就不空了，就需要通知条件变量：队列不空 `notEmpty` 对应的等待队列。
4. 如果出队成功，那就队列就不满了，就需要通知条件变量：队列不满 `notFull` 对应的等待队列。

 复制代码

```
1 public class BlockedQueue<T>{
2     final Lock lock =
3         new ReentrantLock();
4     // 条件变量：队列不满
5     final Condition notFull =
6         lock.newCondition();
7     // 条件变量：队列不空
8     final Condition notEmpty =
9         lock.newCondition();
10
11     // 入队
12     void enq(T x) {
13         lock.lock();
14         try {
15             while (队列已满)
16                 // 等待队列不满
17                 notFull.await();
18             // 省略入队操作...
19             // 入队后，通知可出队
20             notEmpty.signal();
21         }finally {
22             lock.unlock();
23         }
24     }
25 }
```


```
24     }
25     // 出队

26     void deq(){
27         lock.lock();
28         try {
29             while (队列已空)
30                 // 等待队列不空
31                 notEmpty.await();
32             // 省略出队操作...
33             // 出队后, 通知可入队
34             notFull.signal();
35         }finally {
36             lock.unlock();
37         }
38     }
39 }
40 }
41
```

在这段示例代码中, 我们用了 Java 并发包里面的 Lock 和 Condition, 如果你看着吃力, 也没关系, 后面我们还会详细介绍, 这个例子只是先让你明白条件变量及其等待队列是怎么回事。需要注意的是: **await() 和前面我们提到的 wait() 语义是一样的; signal() 和前面我们提到的 notify() 语义是一样的。**

wait() 的正确姿势

但是有一点, 需要再次提醒, 对于 MESA 管程来说, 有一个编程范式, 就是需要在一个 while 循环里面调用 wait()。这个是 MESA 管程特有的。

 复制代码

```
1 while(条件不满足) {
2     wait();
3 }
4
```

Hasen 模型、Hoare 模型和 MESA 模型的一个核心区别就是当条件满足后, 如何通知相关线程。管程要求同一时刻只允许一个线程执行, 那当线程 T2 的操作使线程 T1 等待的条件满足时, T1 和 T2 究竟谁可以执行呢?


1. Hasen 模型里面, 要求 notify() 放在代码的最后, 这样 T2 通知完 T1 后, T2 就结束了, 然后 T1 再执行, 这样就能保证同一时刻只有一个线程执行。
2. Hoare 模型里面, T2 通知完 T1 后, T2 阻塞, T1 马上执行; 等 T1 执行完, 再唤醒 T2, 也能保证同一时刻只有一个线程执行。但是相比 Hasen 模型, T2 多了一次阻塞唤醒操作。
3. MESA 管程里面, T2 通知完 T1 后, T2 还是会接着执行, T1 并不立即执行, 仅仅是从条件变量的等待队列进到入口等待队列里面。这样做的好处是 notify() 不用放到代码的最后, T2 也没有多余的阻塞唤醒操作。但是也有个副作用, 就是当 T1 再次执行的时候, 可能曾经满足的条件, 现在已经不满足了, 所以需要以循环方式检验条件变量。

notify() 何时可以使用

还有一个需要注意的地方，就是 notify() 和 notifyAll() 的使用，前面章节，我曾经介绍过，**除非经过深思熟虑，否则尽量使用 notifyAll()**。那什么时候可以使用 notify() 呢？需要满足以下三个条件：


1. 所有等待线程拥有相同的等待条件；
2. 所有等待线程被唤醒后，执行相同的操作；
3. 只需要唤醒一个线程。

比如上面阻塞队列的例子中，对于“队列不满”这个条件变量，其阻塞队列里的线程都是在等待“队列不满”这个条件，反映在代码里就是下面这 3 行代码。对所有等待线程来说，都是执行这 3 行代码，**重点是 while 里面的等待条件是完全相同的。**

 复制代码

```
1 while (队列已满)
2     // 等待队列不满
3     notFull.await()
4
```

所有等待线程被唤醒后执行的操作也是相同的，都是下面这几行：

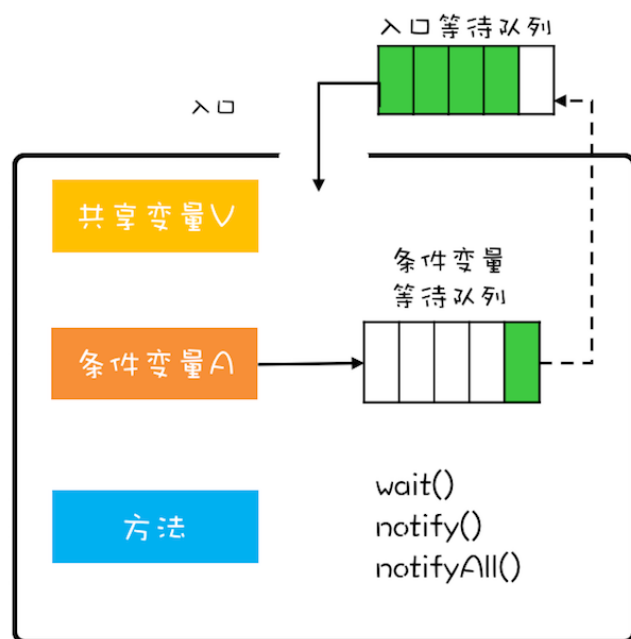
 复制代码

```
1 // 省略入队操作...
2 // 入队后，通知可出队
3 notEmpty.signal();
4
```

同时也满足第 3 条，只需要唤醒一个线程。所以上面阻塞队列的代码，使用 signal() 是可以的。

总结

Java 参考了 MESA 模型，语言内置的管程 (synchronized) 对 MESA 模型进行了精简。MESA 模型中，条件变量可以有多个，**Java 语言内置的管程里只有一个条件变量**。具体如下图所示。



Java 中的管程示意图

Java 内置的管程方案（`synchronized`）使用简单，`synchronized` 关键字修饰的代码块，在编译期会自动生成相关加锁和解锁的代码，但是仅支持一个条件变量；而 Java SDK 并发包实现的管程支持多个条件变量，不过并发包里的锁，需要开发人员自己进行加锁和解锁操作。

并发编程里两大核心问题——互斥和同步，都可以由管程来帮你解决。学好管程，理论上所有的并发问题你都可以解决，并且很多并发工具类底层都是管程实现的，所以学好管程，就是相当于掌握了一把并发编程的万能钥匙。

课后思考

`wait()` 方法，在 Hasen 模型和 Hoare 模型里面，都是没有参数的，而在 MESA 模型里面，增加了超时参数，你觉得这个参数有必要吗？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

猜你喜欢





由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

Ctrl + Enter 发表

0/2000字

提交留言

精选留言(10)



木木匠

思考题：首先，这里的notify()带参数，这应该说的是指操作系统的管程，在java中我没有找到。那这种带超时机制的notify在mesa模型中有必要，原因有两点：

- 1.由于是while 循环，所以就算超时自动唤醒也会去重新检查条件，所以不存在逻辑错误问题
- 2.假设另外一个线程在唤醒之前因为某些原因退出了，那带参数的notify可以超时而进去就绪状态。

参考了操作系统原理课程，如有错误还请老师同学们指正。



2019-03-16



小黄

因为MESA notify是将线程从条件队列转移到入口队列，如果入口队列是阻塞队列，则会循环等待而死锁。不知理解对吗？



2019-03-16



虎虎♡

思考题 我觉得是因为 Hasen和Hoare 模型中T2线程notify T1线程之后中止或阻塞。而在MESA模型中，T2会继续执行，那么T2不能无限期的占有锁，从性能和活跃性上都不合适，所以需要超时参数。

问题：管程方案 Synchronized 怎么实现condition呢？



2019-03-16



我行我素

感觉是必要的，因为在MESA模型中，使用的是循环方式校验条件变量，如果不做超时参数的设置就可能会导致曾经满足的条件此后都不满足而一直循环



2019-03-16



ack

老师好，我想问一下，if 会有参数吗？是在哪里能找到呢，好像我只看到，if 没有参数

老师好，我想问一下notify会有参数吗？是在哪里能找到呢，好像我只看到wait才有参数



2019-03-16

作者回复: 感谢感谢，是wait



花蛋壳

对这一课的内容理解起来有些费劲！“假设有个线程 T1 执行出队操作，不过需要注意的是执行出队操作，有个前提条件，就是队列不能是空的，而队列不空这个前提条件就是管程里的条件变量。”老师所说的这个队列不为空的队列指的只是一个判断条件吧？应该不是指管程里所指的等待队列吧？



2019-03-16

作者回复: 不是，这个指的是BlockedQueue



银时空de梦

用while循环那儿还是不理解，能举例说明，如果不用会出什么问题吗



2019-03-16

作者回复: 等待的条件不成立，出什么问题要看应用场景。例如转账就可能转错。



高源

有必要合理的控制线程，避免阻塞和长时间资源占用



2019-03-16



密码123456

有hasen 是执行完，再去唤醒另外一个线程。能够保证线程的执行。hoare，是中断当前线程，唤醒另外一个线程，执行完再去唤醒，也能够保证完成。而mesa是进入等待队列，不一定有机会能够执行。



2019-03-16

作者回复: 我觉得你真的理解了！！！！



Monica

紧跟上学习步伐



2019-03-16