08-管程：并发编程的万能钥匙

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？

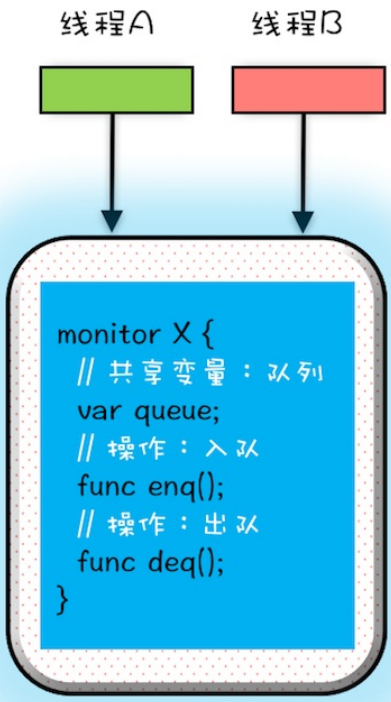
MESA模型

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。



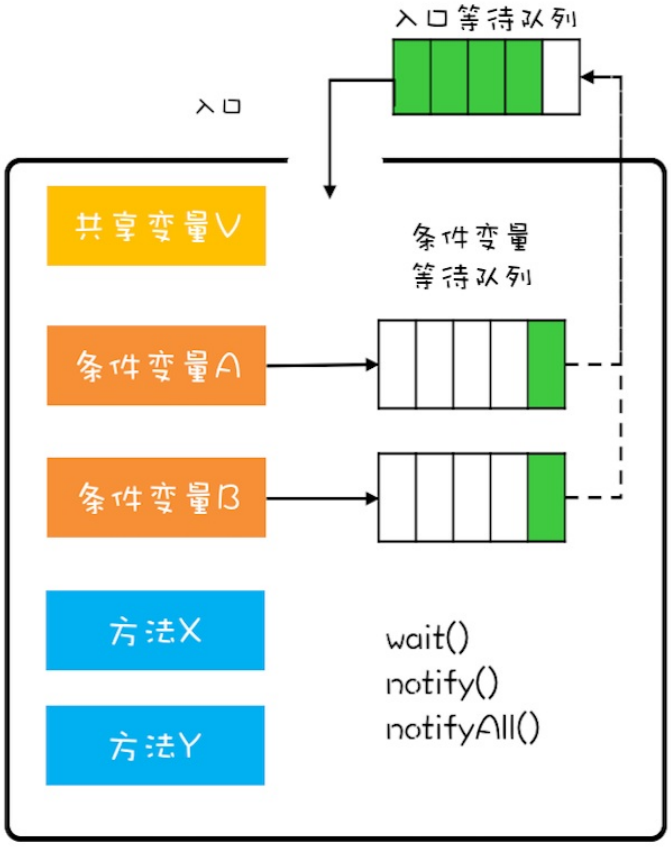
管程模型的代码化语义

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。

```
public class BlockedQueue<T>{
  final Lock lock =
    new ReentrantLock();
  // 条件变量：队列不满
  final Condition notFull =
    lock.newCondition();
  // 条件变量：队列不空
  final Condition notEmpty =
    lock.newCondition();

  // 入队
  void enq(T x) {
    lock.lock();
    try {
      while (队列已满){
        // 等待队列不满
        notFull.await();
      }
      // 省略入队操作...
      //入队后,通知可出队
      notEmpty.signal();
    }finally {
      lock.unlock();
    }
  }
  // 出队
  void deq(){
    lock.lock();
    try {
      while (队列已空){
        // 等待队列不空
        notEmpty.await();
      }
      // 省略出队操作...
      //出队后，通知可入队
      notFull.signal();
    }finally {
      lock.unlock();
    }
  }
}
```

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。**

```
while(条件不满足) {
  wait();
}
```

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**。

```
while (队列已满){
  // 等待队列不满
  notFull.await();
}
```

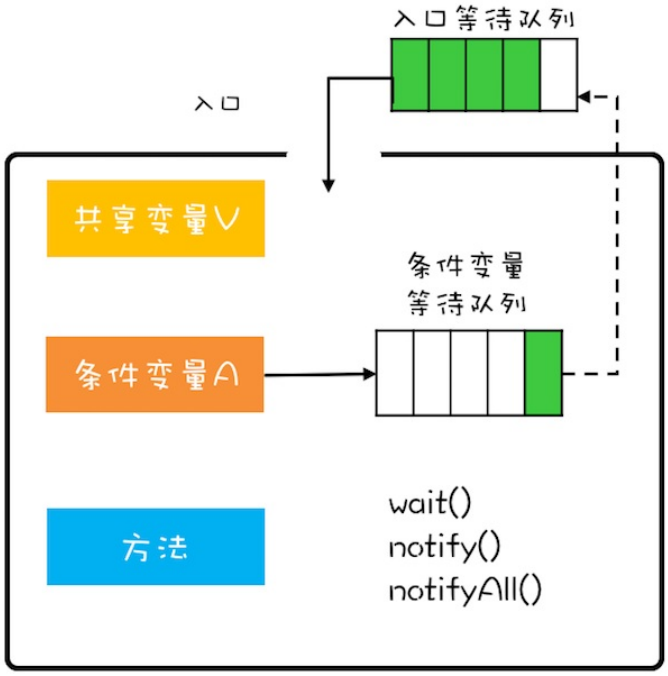
所有等待线程被唤醒后执行的操作也是相同的，都是下面这几行：

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

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

精选留言：

- 三圆 2019-03-16 23:06:15
  看了三遍，终于明白了管程MESA模型了，刚开始一直在想线程T1执行出队是什么意思？到底是哪个队列，是入口等待队列，还是条件等待队列，后来理解了都不是。这个队列应该理解为JDK里面的阻塞队列，里面存在的是共享数据，线程T1,T2分别去操作里面的共享数据，执行数据的入队，出队操作，当然这些操作是阻塞操作。当线程T1对阻塞队列执行数据出队操作时，进入管程，发现阻塞队列为空，此时线程T1进入阻塞队列不为空这个条件的条件等待队列，此时，其他线程还是可以进入管程的，比如T2进来了，对阻塞队列执行数据插入操作，这时就会致使线程T1从条件等待队列出来，进入入口等待队列，准备再一次进入管程……至于wait方法的参数，还是有必要的，因为可能线程需要的条件可能一直无法满足！ [21赞]

  作者回复2019-03-17 10:52:28
  例子选的不好，让你误解了...

- 密码123456 2019-03-16 08:25:34
  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，执行玩再去唤醒，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。 [18赞]

  作者回复2019-03-16 11:06:42
  我觉得你真的理解了！！！！

- Hour 2019-03-22 22:51:25
  今天又用代码验证了下，终于明白为啥用while了！

  当线程被唤醒后，是从wait命令后开始执行的(不是从头开始执行该方法，这点上老师的示意图容易让人产生歧义)，而执行时间点往往跟唤醒时间点不一致，所以条件变量此时不一定满足了，所以通过while循环可以再验证，而if条件却做不到，它只能从wait命令后开始执行，所以要用while

  百看不一练😂😂😂😂
  [6赞]

- 虎虎❤ 2019-03-16 11:45:46
  重新回答思考题，问题变成wait的timeout参数是否必要。

  在MESA模型中，线程T1被唤醒，从条件A的等待队列中（其实是一个set，list的话可能会重复）移除，并加入入口等待队列，重新与其他的线程竞争锁的控制权。那么有这样一种可能，线程T1的优先级比较低，并且经常地有高优先级的线程加入入口等待队列。每次当它获得锁的时候，条件已经不满足了（被高优先级的线程抢先破坏了条件）。即使T1可以得到调度，但是也没办法继续执行下去。

最后T1被饿死了（有点冷。。。）

另外我刚才的问题想通了。不需要实现像lock一样的条件对象，并调用condition.await(). Synchronized用判断条件+wait（）就可以了。
[5赞]

作者回复2019-03-16 17:55:04
我觉得你已经能把管程的运作在大脑里演绎出来了！

- CCC 2019-03-17 14:42:14
MESA模型和其他两种模型相比可以实现更好的公平性，因为唤醒只是把你放到队列里而不保证你一定可以执行，最后能不能执行还是要看你自己可不可以抢得到执行权也就是入口，其他两种模型是显式地唤醒，有点内定的意思了。 [4赞]

作者回复2019-03-17 17:11:54
内定都出来了，真是理论联系生活

- Handongyang 2019-03-16 18:53:24
感谢老师的精彩分享，谈一下个人对信号量和管程的理解。

信号量机制是可以解决同步/互斥的问题的，但是信号量的操作分散在各个进程或线程中，不方便进行管理，因每次需调用PV操作，还可能导致死锁或破坏互斥请求的问题。

管程是定义了一个数据结构和能为并发所执行的一组操作，这组操作能够进行同步和改变管程中的数据。这相当于对临界资源的同步操作都集中进行管理，凡是要访问临界资源的进程或线程，都必须先通过管程，由管程的这套机制来实现多进程或线程对同一个临界资源的互斥访问和使用。管程的同步主要通过condition类型的变量（条件变量），条件变量可执行操作wait()和signal()。管程一般是由语言编译器进行封装，体现出OOP中的封装思想，也如老师所讲的，管程模型和面向对象高度契合的。 [3赞]

作者回复2019-03-17 11:03:23
是的，管程只是一种解决并发问题的模型而已。

- life is short, enjoy more. 2019-03-23 20:47:08
好记性不如烂笔头
管程定义：
管理共享内存和操作共享内存的过程，令其支持并发。

并发编程领域两大难题，互斥和同步。

互斥如何保证：封装共享数据及其对应操作，每个操作任意时刻只允许一个线程执行。

同步如何保证：使用MESA模型。

MESA模型之前没接触过，只是能够按照老师的思路顺下来，还没有深刻的理解，希望老师在接下来的文章中，加深MESA的使用😄

[2赞]

- 江南豆沙包 2019-03-23 08:40:29
老师，有个疑问，文中说到的条件变量，假如 synchronized(instance) { 做一些事情 }，这样一段代码，

程序实际运行过程中条件变量是什么呢 [2赞]

作者回复2019-03-23 11:56:10
没用到条件变量，只有调用wait和notify的时候才会用到

- Hour 2019-03-22 07:28:57
老师，针对条件变量的while循环，还是不太理解，您说是范式，那它一定是为了解决特定的场景而强烈推荐的，也有评论说是为了解决虚假唤醒，但唤醒后，不也是从条件的等待队列进入到入口的等待队列，抢到锁后，重新进行条件变量的判断，用if完全可以啊，为什么必须是while，并且是范式？

望老师赐教！ [2赞]

作者回复2019-03-22 22:15:38
code1；
if (条件不满足)
wait()
code2；

当调用wait()时，阻塞。被唤醒时，就直接执行code2了，没机会重新判断。

- linqw 2019-04-07 11:39:35
管程的组成锁和0或者多个条件变量，java用两种方式实现了管程①synchronized+wait、notify、notifyAll ②lock+内部的condition，第一种只支持一个条件变量，即wait，调用wait时会将其加到等待队列中，被notify时，会随机通知一个线程加到获取锁的等待队列中，第二种相对第一种condition支持中断和增加了时间的等待，lock需要自己进行加锁解锁，更加灵活，两个都是可重入锁，但是lock支持公平和非公平锁，synchronized支持非公平锁，老师，不知道理解的对不对 [1赞]

作者回复2019-04-08 15:41:49
总结很全面！

- 小李子 2019-04-05 21:58:40
wait() 不加超时参数，相当于得一直等着别人叫你去门口排队，加了超时参数，相当于等一段时间，再没人叫的话，我就受不了自己去门口排队了，这样就诊的机会会大一点，是这样理解吧？ [1赞]

作者回复2019-04-06 16:38:17
挺形象，就诊机会不一定大，但是能避免没人叫的时候傻等

- 白马居士 2019-04-05 21:03:34
第一次看到MESA管程模型的时候很膈应，为什么要满足队列不空才能出队，在对照了《Java并发编程艺术》后，现在再看这一章才意识到“队列不空”只是人为的设置的条件A，只是为了说明管程支持多条件控制并发而自己设置控制条件的一个特例，是自己的设置，具体是什么与模型无关；初次看到这段还以为是MESA管程模型规定了“队列不空”这个出队条件，所以特别混乱。 [1赞]

- 红衣闪闪亮晶晶 2019-03-26 15:53:07
老师，我能明白如果t1线程被唤醒后再次进入等待队列，但是可能再次走到条件变量那里再次因为条件不满足随后再次开始等待，所以需要增加超时，所以当我给wait加了超时，时间到了以后t1再次开始while中的判断，如果满足便自己回到入口等待队列？
我这样理解对吗？ [1赞]

作者回复2019-03-30 08:54:31
如果没超时，A线程wait了，由于代码的bug，没有其他线程notify，就会导致A一直wait。增加超时之后

，A线程可以自己来决定是否继续等待。这样代码的健壮性会更好

- 坯子蔡 2019-03-21 15:30:34
线程T1 进行出队操作，发现队列为空，进入不为空条件等待队列，线程T2进行入队操作完后，唤醒T1，T1从新进入入口等待队列，当再次获取互斥锁，进入队列是否为空条件判断，当不满足条件再次进入条件等待队列，为什么要用while进行循环判断， 始终不明白，看了ArrayBlockQueue源码也是用while 循环判断？请老师解惑一下！！！ [1赞]

- 佑儿 2019-03-20 23:32:31
老师，您好，结合第六讲，我的理解是：简单来说，一个锁实际上对应两个队列，一个是就绪队列，对应本节的入口等待队列，一个是阻塞队列，实际对应本节的条件变量等待队列，wait操作是把当前线程放入条件变量的等待队列中，而notifyall是将条件变量等待队列中的所有线程唤醒到就绪队列（入口等待队列）中，实际上哪个线程执行由jvm操作，我这样的理解对吗？ [1赞]

作者回复2019-03-22 23:36:13
对

- QQ怪 2019-03-18 22:13:15
不加超时时间可能会一直不满足条件一直不跳出 [1赞]

- boyxie 2019-03-18 09:23:25
想起来前段时间看的AQS，很像是MESA的具体实现哈，不知道是否理解正确，以前看到监视器模型就没有深究它的理论模型，看来还是要追根溯源，这样才能真正理解，比如看了很多并发包的源码，但是现在加上老师的理论，理解起来会豁然开朗。 [1赞]

作者回复2019-03-18 11:54:55
就是mesa的实现

- 7 2019-03-17 00:53:44
假设当前运行线程为t2 阻塞线程（在条件变量等待队列中的线程）t1
Hasen 条件变量满足 t2唤醒t1 此时t2也即将结束因为notify在t2执行末尾 t1可以立刻执行
Hoare 条件变量满足 t2唤醒t1 t2阻塞 t1执行
MESA 条件变量满足 t2唤醒t1 t2继续执行 t1进到【入口等待队列 】，当t1再次获得互斥锁时可能条件变量已经改变如是这样t1会再次进入阻塞队列 如果如此反复，将一直在while循环里 所以需要wait中添加时间参数很有必要 [1赞]

作者回复2019-03-17 11:09:19
我觉得程序员免不了写出有并发问题的程序，增加一个超时参数，更保险一些

- 墨名次 2019-03-17 00:12:53
老师，我觉得自己理解错然后死循环了，😂，这一段：“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。”。

线程T1出列，正好发现队列为空，那么T1会进入条件变量对应的队列等待，假设一个进程只有一个线程T1，那线程T1不是要一直在条件变量队列里面等待吗，往下看，wait是有超时时间的，超时后，线程T1重新进入入口队列，入队成功后，T1再次出队，这时发现队列为空，又要进入条件变量队列等待……这不是死循环了吗😂

[1赞]

- 小黄 2019-03-16 23:50:27
  不理解，notify会把条件队列里线程移到入口队列。其他线程会因为锁阻塞在入口队列，那入口队列怎么保证线程安全的？ [1赞]

  作者回复2019-03-17 10:54:07
  JVM保证，或者并发包里的AQS也能保证