**多线程之间相互协作的通信**

一、synchronized+notify+wait+flag

这种方案是通过**对两个线程中分别要执行的方法加锁synchronized**，保证每次执行main时不被sub打断，执行sub循环时，不被main打断。

这里采用了对象object的notify和wait来实现线程之间的通信。当main方法执行完成后，让执行main方法的线程等待，等待sub方法执行完成后，通知(notify)main线程然后继续执行。这种方式有一个缺点，由于notify和wait使用的是Object的方法，所以不能单独的让某个特定的线程收到通知或者让他等待，而**在存在多个线程同时等待时，只能通过notifyAll来通知所有的线程。不够灵活**。

二、lock+condition+flag

这种方式是利用了Java5中提供的lock和condition，利用共享变量flag来实现线程之间的相互通信。同时在这个小例子中，相比上一个例子中增加了一个线程的循环。这是为了体现使用condition的优点。

使用condition可以非常灵活的去控制线程与线程之间的通信。因为在一个类中可以创建多个condition的实例，

我们可以通过condition不同的实例的signal和await方法来标识不同的两个线程之间相互通信的标识，而不是统一使用object的notify和wait方法了。

同时利用lock方法可以利用锁的重入机制实现更加灵活的锁的应用。可以在需要的时候加锁或解锁。

这样我们就可以实现多个线程之间的协调通信了。

三、semaphere+flag

这里semaphere代表一个信号量，它可以指示共享资源的个数，也就是同时访问资源的线程个数。这里主要通过semaphere的acquire和release实现锁的功能从而实现线程之间的通信。

利用semaphere不仅可以实现多个线程协调循环通信，在必要时还可以控制同一时间访问资源的个数。更加的灵活和方便。

以上是实习多个线程之间相互协调通信的几种方案。