一 、synchronized锁与lock锁的区别：

1、synchronized锁无法判断是否获取锁的状态，lock可以判断是否获取到锁。

2、synchronized锁会自动释放锁（a.线程执行完同步代码之后会释放锁；b.线程执行过程发生异常会释放锁），lock锁需要在finally中手动释放锁（unlock()方法释放锁），否则容易造成线程死锁。

3、用synchronized锁的两个线程1和线程2，如果当前线程1获得锁，线程2线程等待。如果线程1阻塞，线程2会一直等待下去；lock锁不会，如果尝试获取不到锁，线程可以不用一直等待就结束。

4、synchronized锁可重入，不可中断，非公平；lock锁可重入，可判断，可中断，可公平（两者皆可）

5、lock锁适合量大同步的代码的问题，synchronized锁适合代码少量的同步问题。

二、synchronized锁与lock锁的优缺点：

1、synchronized是性能低效的。因为这是一个重量级操作，它对性能最大的影响是阻塞的是实现，挂起线程和恢复线程的操作都需要转入内核态中完成，这些操作给系统的并发性带来了很大的压力。相比之下使用Java提供的Lock对象，性能更高一些。多线程环境下，synchronized的吞吐量下降的非常严重，而ReentrankLock则能基本保持在同一个比较稳定的水平上。

2、基本语法上，ReentrantLock与synchronized很相似，它们都具备一样的线程重入特性，只是代码写法上有点区别而已，一个表现为API层面的互斥锁（Lock），一个表现为原生语法层面的互斥锁（synchronized）。ReentrantLock相对synchronized而言还是增加了一些高级功能，主要有以下三项：

**1、等待可中断：当持有锁的线程长期不释放锁时，正在等待的线程可以选择放弃等待，改**

**为处理其他事情，它对处理执行时间非常上的同步块很有帮助。而在等待由synchronized**

**产生的互斥锁时，会一直阻塞，是不能被中断的。**

**2、可实现公平锁：多个线程在等待同一个锁时，必须按照申请锁的时间顺序排队等待，而**

**非公平锁则不保证这点，在锁释放时，任何一个等待锁的线程都有机会获得锁。**

**synchronized中的锁时非公平锁，ReentrantLock默认情况下也是非公平锁，但可以通过**

**构造方法ReentrantLock（ture）来要求使用公平锁。**

**3、锁可以绑定多个条件：ReentrantLock对象可以同时绑定多个Condition对象（名曰：**

**条件变量或条件队列），而在synchronized中，锁对象的wait（）和notify（）或**

**notifyAll（）方法可以实现一个隐含条件，但如果要和多于一个的条件关联的时候，就不得**

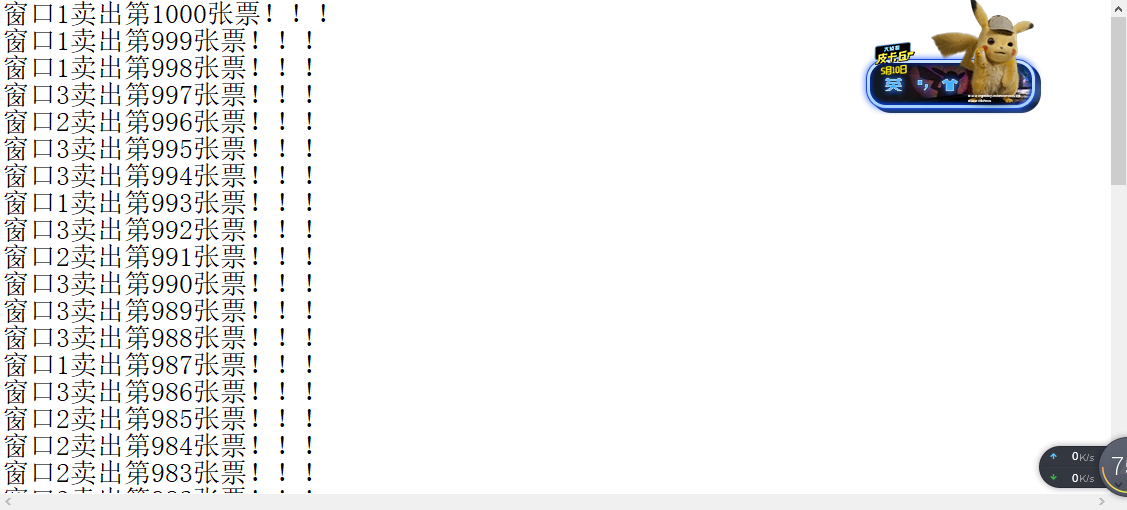
**不额外地添加一个锁，而ReentrantLock则无需这么做，只需要多次调用**

**newCondition（）方法即可。而且我们还可以通过绑定Condition对象来判断当前线程通**

**知的是哪些线程（即与Condition对象绑定在一起的其他线程）。**

**三、**synchronized锁与lock锁的的特点

1、synchronized锁是全部锁住，在没有释放锁是，其他需要使用到锁的线程无法运行，需要等待锁的放开。且只有运行成之后才会释放锁，如果出现卡顿，死机的话，其他线程会一直处于等待休眠状态。



2、lock锁一个线程获取锁，其他线程就阻塞(等待锁的释放)，一个线程获取锁，其他线程获取不到锁，就会放弃，手动释放锁必须在finally中取释放锁，否则锁永远得不到锁死锁。

