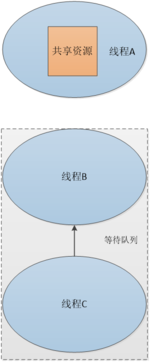
不要总想着用具体的类来编程，优先使用抽象和interface编程

1.ReentrantLock类的其中一个构造器提供了指定****公平策略**** / ****非公平策略****的功能，默认为****非公平策略****。

****公平策略：****在多个线程争用锁的情况下，公平策略倾向于将访问权授予等待时间最长的线程。也就是说，相当于有一个线程等待队列，先进入等待队列的线程后续会先获得锁，这样按照“先来后到”的原则，对于每一个等待线程都是公平的。

****非公平策略：****在多个线程争用锁的情况下，能够最终获得锁的线程是随机的（由底层OS调度）。

注意：一般情况下，使用公平策略的程序在多线程访问时，总体吞吐量（即速度很慢，常常极其慢）比较低，因为此时在线程调度上面的开销比较大。

举个例子：  
假设采用公平策略，线程A首先获取了锁，线程B和线程C等待获取锁，如下图：  


当线程A释放锁时，线程B将经历从 挂起->唤醒 的线程调度过程，线程调度非常耗时。

在线程B的 挂起->唤醒 阶段：

1. 如果采用非公平策略，那么线程C可以立即获取锁，线程C使用完并释放锁后，线程B可能才刚唤醒完成；此时线程B又可以去获取锁，这样线程B和线程C的效率都得到提升，系统吞吐量提升；
2. 如果采用公平策略，线程C即使可用，也要等到线程调度完成，整个系统的吞吐量降低。

因此，当线程持有锁的时间相对较长或者线程请求锁的平均时间间隔较长时，可以考虑使用公平策略。此时线程调度产生的耗时间隔影响会较小。

线程调度是需要花时间的，所以非公平锁的速度反而更快

LockSupport类，是JUC包中的一个工具类，是用来创建锁和其他同步类的基本线程阻塞原语。（[Basic thread blocking primitives for creating locks and other synchronization classes](https://docs.oracle.com/javase/8/docs/api/" \t "https://segmentfault.com/a/_blank)）

LockSupport类的核心方法其实就两个：park()和unpark()，其中park()方法用来阻塞当前调用线程，unpark()方法用于唤醒指定线程。

## **AQS原理简述**

我们在第一节中讲到，AQS框架分离了构建同步器时的一系列关注点，它的所有操作都围绕着资源——同步状态（synchronization state）来展开因此，围绕着资源，衍生出三个基本问题：

1. 同步状态（synchronization state）的管理
2. 阻塞/唤醒线程的操作
3. 线程等待队列的管理

### **3.1 同步状态**

**同步状态的定义**  
同步状态，其实就是资源。AQS使用单个int（32位）来保存同步状态，并暴露出getState、setState以及compareAndSetState操作来读取和更新这个状态。

*/\*\**

*\* 同步状态.*

*\*/***private** **volatile** **int** state;

**protected** **final** **int** **getState**() {

**return** state;

}

**protected** **final** **void** **setState**(**int** newState) {

state = newState;

}*/\*\**

*\* 以原子的方式更新同步状态.*

*\* 利用Unsafe类实现*

*\*/***protected** **final** **boolean** **compareAndSetState**(**int** expect, **int** update) {

**return** unsafe.compareAndSwapInt(**this**, stateOffset, expect, update);

}

### **3.2 线程的阻塞/唤醒**

在JDK1.5之前，除了内置的监视器机制外，没有其它方法可以安全且便捷得阻塞和唤醒当前线程。  
JDK1.5以后，java.util.concurrent.locks包提供了[LockSupport](https://segmentfault.com/a/1190000015562456)类来作为线程阻塞和唤醒的工具。

### **3.3 等待队列**

等待队列，是AQS框架的核心，整个框架的关键其实就是如何在并发状态下管理被阻塞的线程。  
等待队列是严格的FIFO队列，是Craig，Landin和Hagersten锁（CLH锁）的一种变种，采用双向链表实现，因此也叫CLH队列。

个人理解lock的原理，多个线程去竞争获取锁（就是队列的首节点），获取到的直接运行，获取不到的放入一个队列后面面，当本线程执行完成后轮到下一个节点运行，下一个节点就是首节点

然后出现一个condition条件使用，在调用condition.await()时会把当前线程放入到一个类似上述锁的同步队列一样，并且释放同步锁，当调用condition.notifyAll()的时候，会从condition队列里面取出首节点，然后放入同步队列，去竞争，竞争到了就去执行接下来的操作