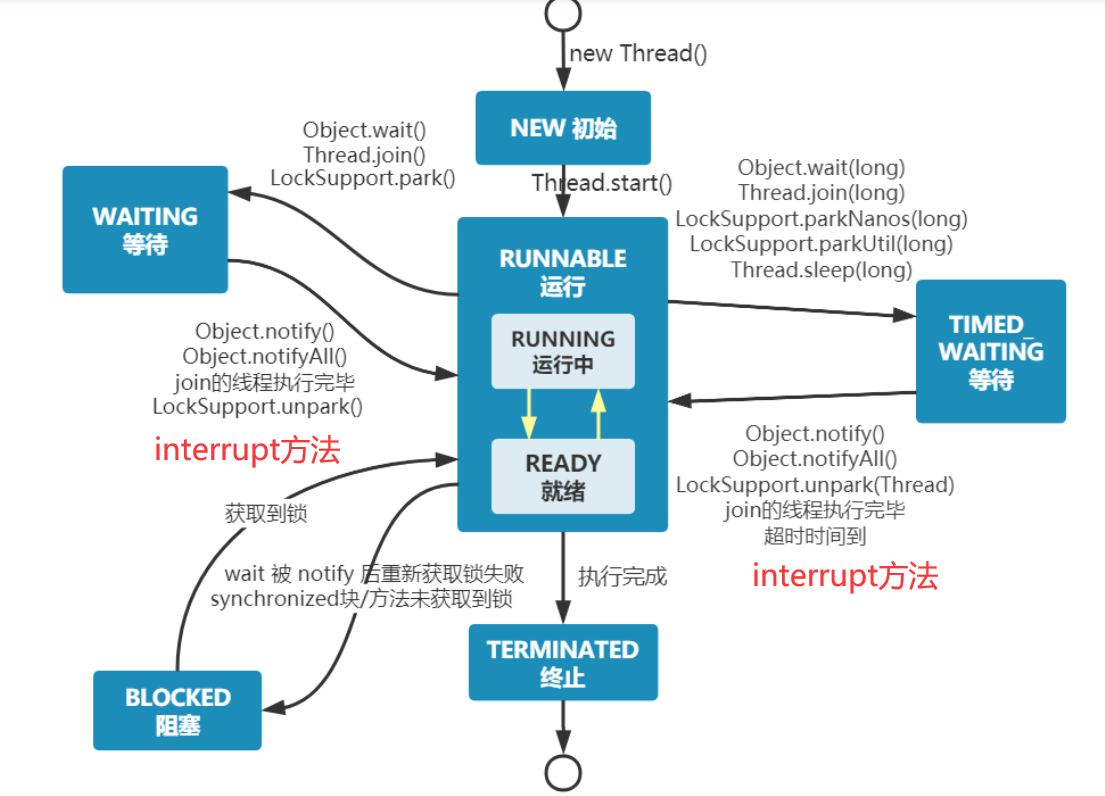


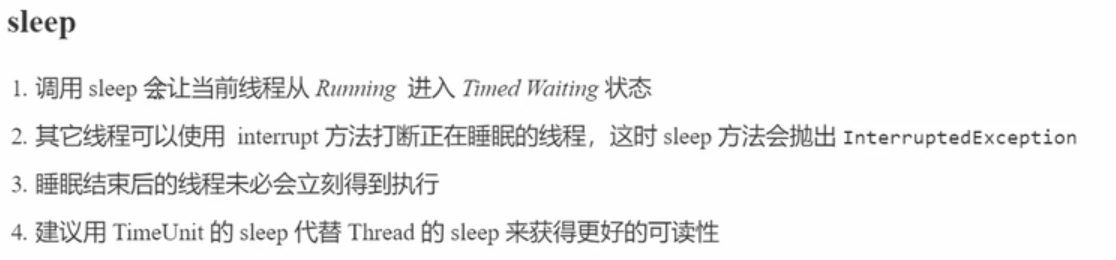
Runnable中的run方法就是线程的执行过程，这样可以将线程的定义和线程的执行过程分开。如果用第一种方法，则线程的执行步骤就是Thread中重写的run方法，而第二种方法，在调用Thread的run方法时内部调用的是传进来的runnable的run方法，仅此而已。Future也是个Runnable(实现了接口)，但它也实现了Future接口。这两个接口，Runnable的功能是执行run方法，Future的功能是获取返回结果get方法，FutureTask实现这两个接口，就增加了这两个功能，具体实现是FutureTask中的属性Callable接口来完成，这个接口的call方法是有返回值的，run方法执行call方法，get方法得到call的返回结果。

线程转换：



Thread常用方法：

Sleep：



静态方法 从Runnable状态到TIMED\_WAITING状态。不会释放锁。

Yield：

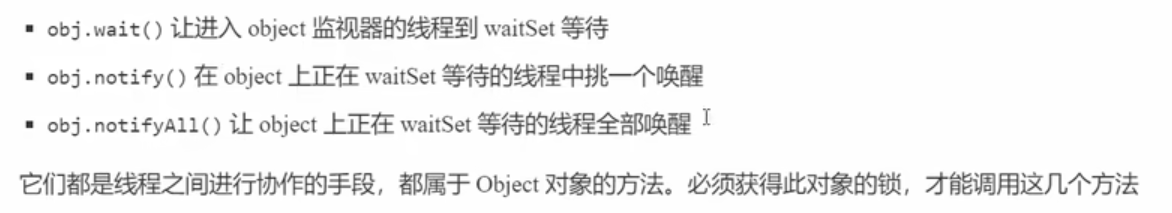


成员方法 让出cup，让优先级高的执行，但具体cup分配给谁要看操作系统。线程一直处于Runnable状态(从java视角看)，实际上线程从running到ready(从OS角度看)。线程默认优先级是5，最大10，最小1.

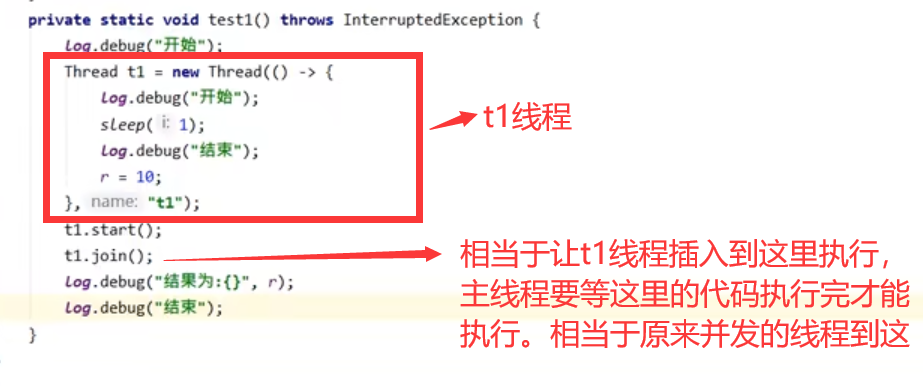
Join：

可以看作是让原来并发的线程顺序执行

Wait、notify:



这些方法必须要在syn代码块中使用，通过锁对象来调用，在哪个线程中调用wait方法，哪个线程就会进入waiting状态。

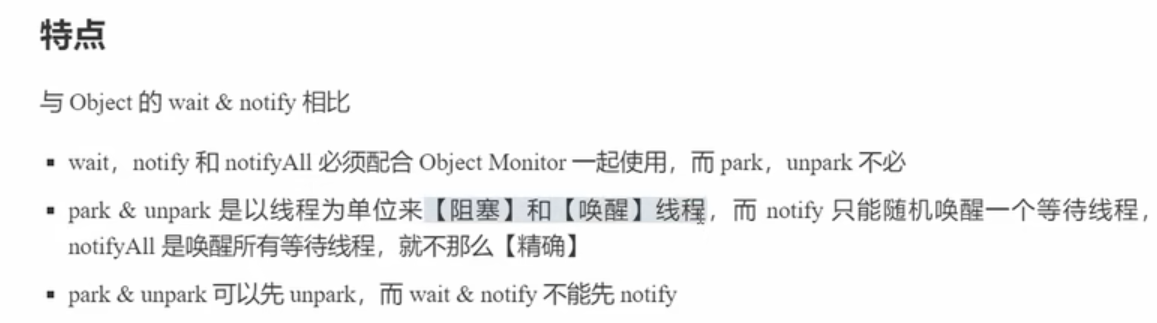


主线程从Runnable状态到Waiting状态。还有一个带参数的join方法，线程从Runnable状态到Timed\_Waiting状态

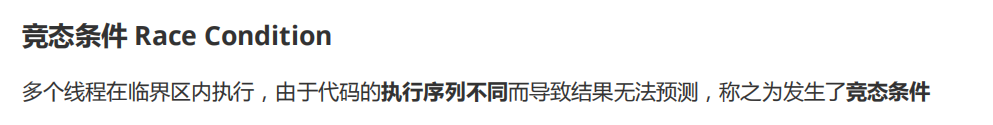
Park&unPark:



先unpark后park，park会失效。

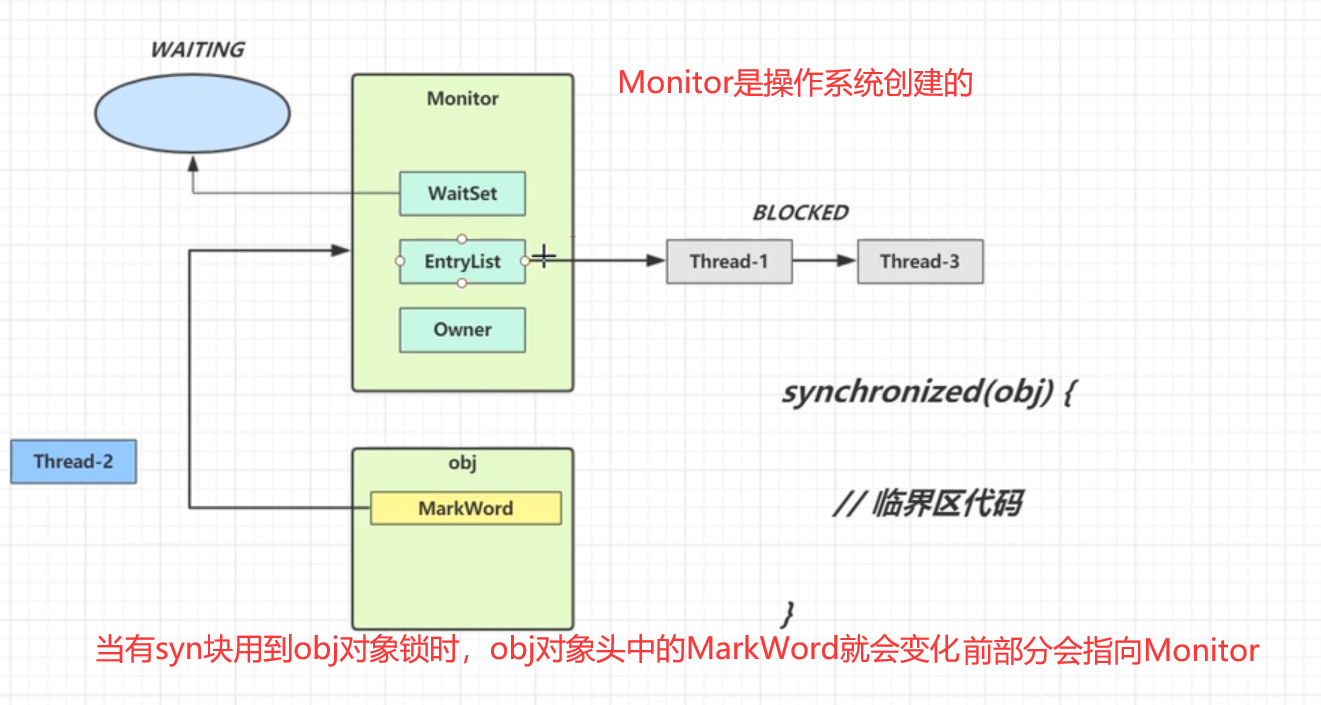


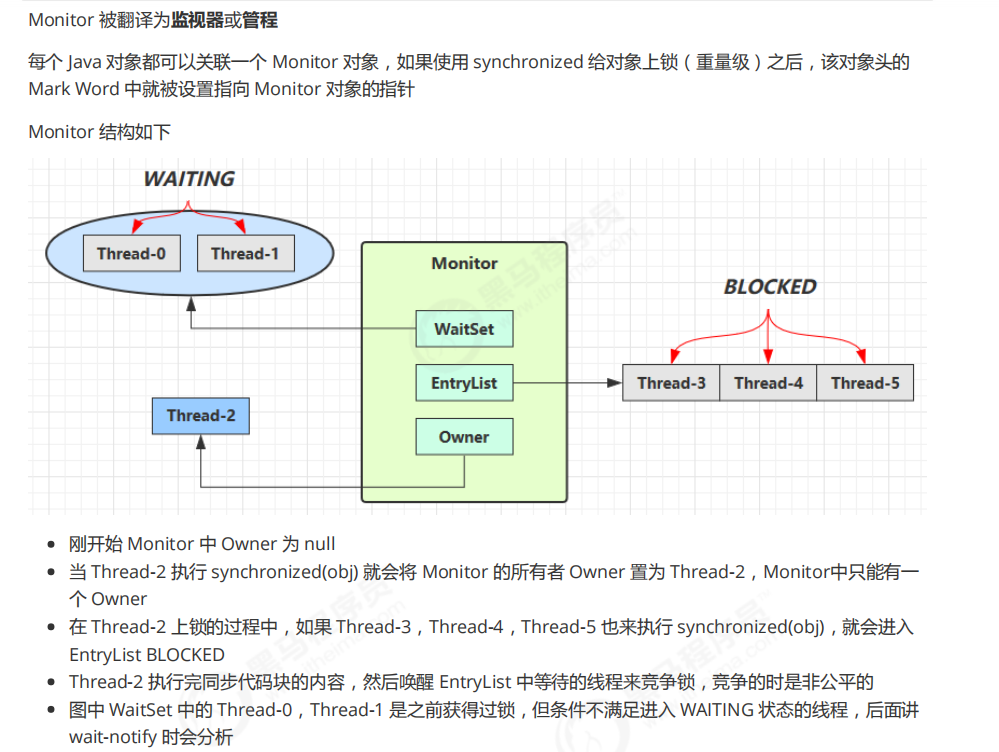
共享模型之管程：



非静态方法上的syn用的是this对象锁，静态方法上的syn用的是本类的Class对象锁

重量级锁：

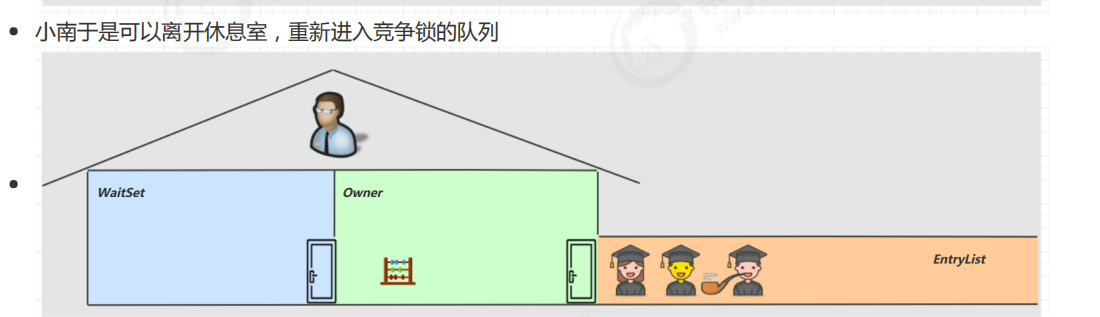
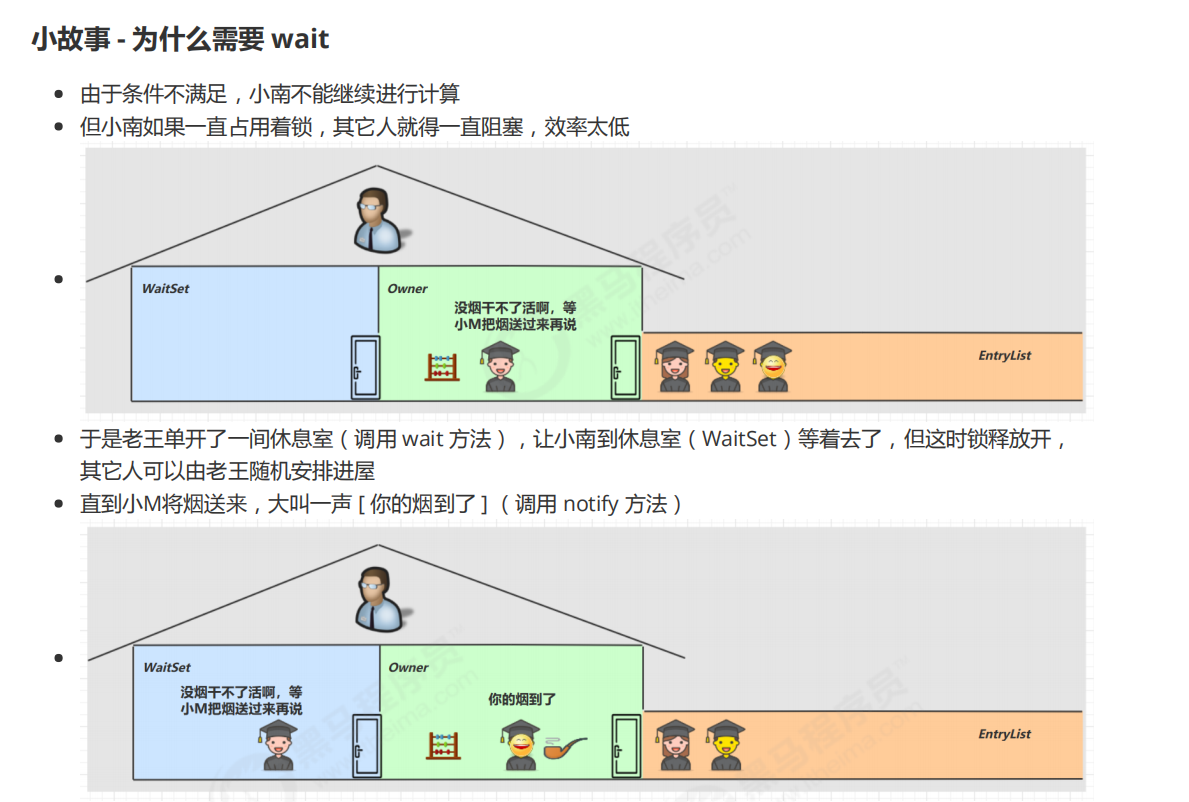




加锁解锁操作都是一条字节码指令。出现异常也会释放锁。

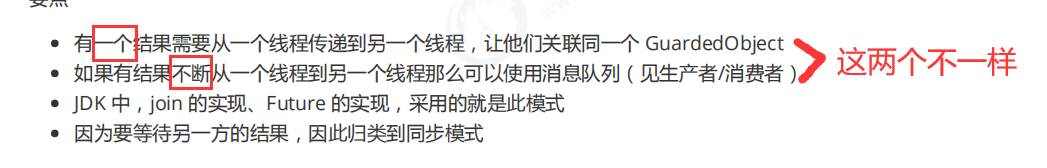
轻量级锁：

Wait、notify：





同步模式之保护性暂停：





Join方法等价于下面：



活锁：

活锁出现在两个线程互相改变对方的结束条件，最后谁也无法结束

ReentrantLock：



可打断：

lock.lock()方法在等待获得锁的过程中是不会被打断的。

lock.lockInterruptibly()方法和上面一样不过可以被打断。

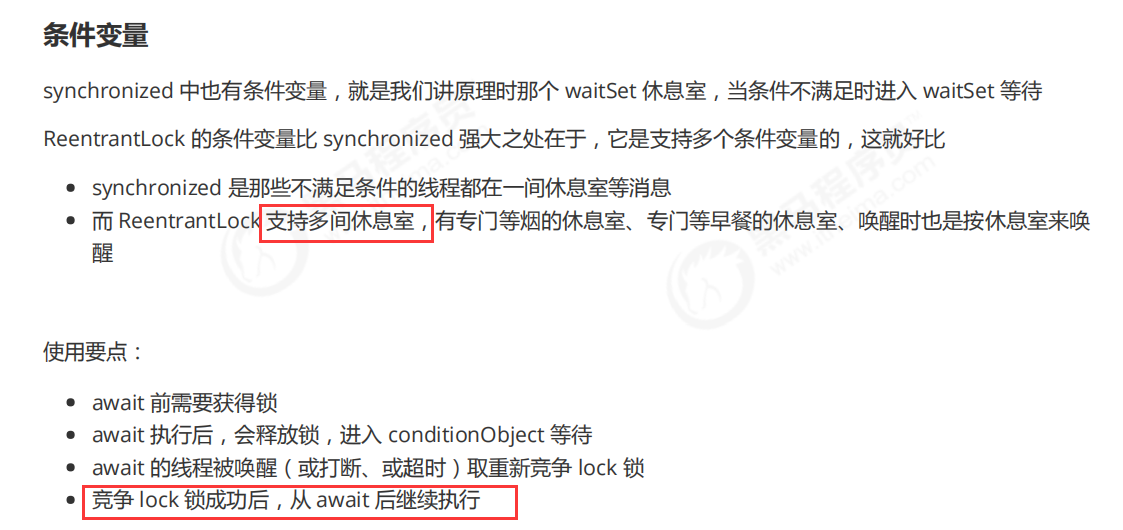
锁超时：

lock.tryLock()方法获取不到锁线程不会被阻塞，而是返回false，继续执行。获取到锁返回true。

tryLock(long timeout, TimeUnit unit)，在规定时间内获取锁返回true否则返回false，无论返回什么，线程都会继续执行，不会阻塞。

公平锁可以解决饥饿问题。但效率低。

条件变量：



Java内存模型：共享变量的存储区域称为主存，私有变量存放的区域称为工作内存。

可见性：

volatile（易变关键字） 只能加在成员变量和静态成员变量上。他可以避免线程从自己的工作缓存中查找变量的值，必须到主存中获取它的值，线程操作 volatile 变量都是直接操作主存。不能解决原子性问题。

Syn也可以解决可见性问题，在操作共享变量时加上锁。

有序性：

指令重排在多线程情况下会出问题，volatile可以防止指令重排。

Volatile原理：

