<https://www.cnblogs.com/yuxiang1/p/9183671.html>

[**Java中线程同步锁和互斥锁有啥区别？看完你还是一脸懵逼？**](https://www.cnblogs.com/yuxiang1/p/9183671.html)

首先不要钻概念牛角尖，这样没意义。

也许java语法层面包装成了sycnchronized或者明确的XXXLock，但是底层都是一样的。无非就是哪种写起来方便而已。

锁就是锁而已，避免多个线程对同一个共享的数据并发修改带来的数据混乱。

锁要解决的大概就只有这4个问题：

* “谁拿到了锁“这个信息存哪里（可以是当前class，当前instance的markword，还可以是某个具体的Lock的实例）
* 谁能抢到锁的规则（只能一个人抢到 - Mutex；能抢有限多个数量 - Semphore；自己可以反复抢 - 重入锁；读可以反复抢到但是写独占 - 读写锁……）
* 抢不到时怎么办（抢不到玩命抢；抢不到暂时睡着，等一段时间再试/等通知再试；或者二者的结合，先玩命抢几次，还没抢到就睡着）
* 如果锁被释放了还有其他等待锁的怎么办（不管，让等的线程通过超时机制自己抢；按照一定规则通知某一个等待的线程；通知所有线程唤醒他们，让他们一起抢……）

有了这些选择，你就可以按照业务需求组装出你需要锁。

关于“互斥”和“同步”的概念

* 答案很清楚了，互斥就是线程A访问了一组数据，线程BCD就不能同时访问这些数据，直到A停止访问了
* 同步就是ABCD这些线程要约定一个执行的协调顺序，比如D要执行，B和C必须都得做完，而B和C要开始，A必须先得昨晚。

这是两种典型的并发问题。恰当的使用锁，可以解决同步或者互斥的问题。

你可以说Mutex是专门被设计来解决互斥的；Barrier，Semphore是专门来解决同步的。但是这些都离不开上述对上述4个问题的处理。同时，如果遇到了其他的具体的并发问题，你也可以定制一个锁来满足需要。

**另外一个解释或许能明白更多**

所谓互斥，就是不同线程通过竞争进入临界区（共享的数据和硬件资源），为了防止访问冲突，在有限的时间内只允许其中之一独占性的使用共享资源。如不允许同时写

同步关系则是多个线程彼此合作，通过一定的逻辑关系来共同完成一个任务。一般来说，同步关系中往往包含互斥，同时对临界区的资源会按照某种逻辑顺序进行访问。如先生产后使用

总的来说，两者的区别就是：

互斥是通过竞争对资源的独占使用，彼此之间不需要知道对方的存在，执行顺序是一个乱序。

同步是协调多个相互关联线程合作完成任务，彼此之间知道对方存在，执行顺序往往是有序的。