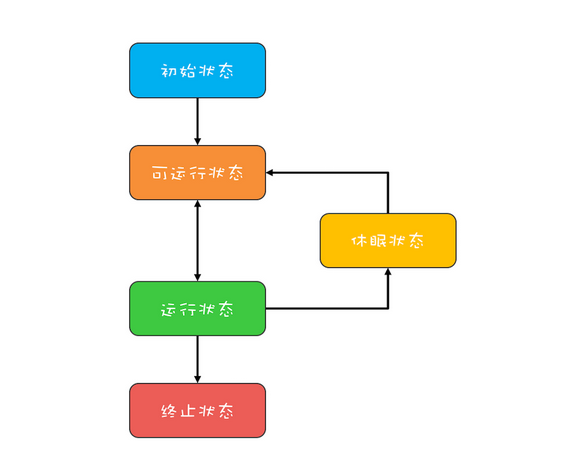
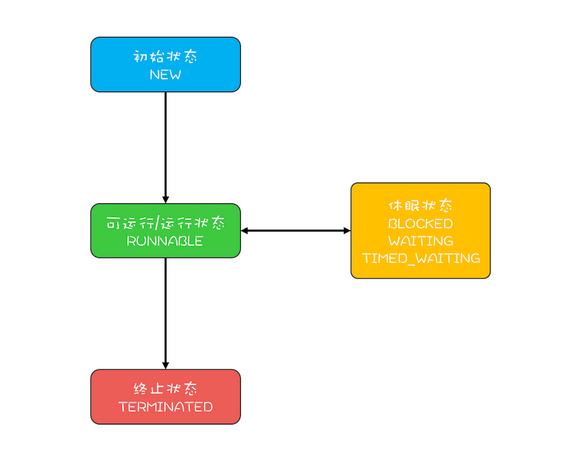
Java线程状态(左)、OS线程状态：



Blocked状态和waiting对OS来说是一种状态，但对java来说是不同状态。

Java调用的阻塞API时(IO流读取)线程状态没有变化还是runnable状态，但是OS层面此时线程处于休眠状态。平时所谓的 Java 在调用阻塞式 API 时，线程会阻塞，指的是操作系统线程的状态，并不是 Java 线程的状态。

状态转换：

1. RUNNABLE 与 BLOCKED 的状态转换

只有syn锁（可重入锁lock时底层调用了LockSupport.park）

2. RUNNABLE 与 WAITING 的状态转换

wait()、join()、LockSupport.park()、

3. RUNNABLE 与 TIMED\_WAITING 的状态转换

Object.wait(long timeout)、Thread.sleep(long millis)、Thread.join(long millis)、

LockSupport.parkNanos(Object blocker, long deadline)、LockSupport.parkUntil(long deadline)

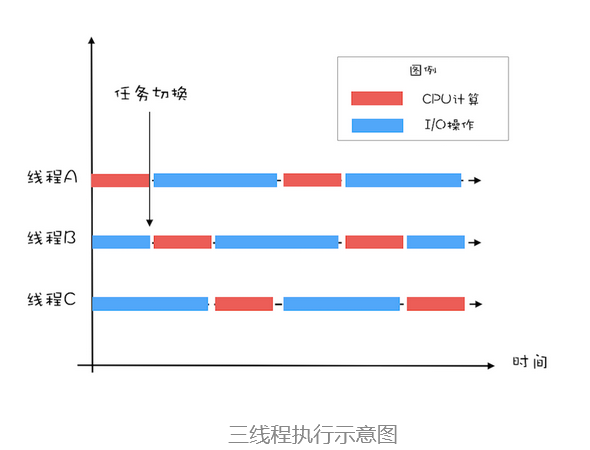
5.从 RUNNABLE 到 TERMINATED 状态

Run执行结束、抛出异常、stop(杀死线程后不会释放可重入锁，不推荐使用)、interrupt、

最佳线程数：

CPU密集型：CPU核数+1

IO密集型：CPU 核数 \* [ 1 +（I/O 耗时 / CPU 耗时）]



为什么要设计可重入锁：

1.能够响应中断。synchronized 的问题是，持有锁 A 后，如果尝试获取锁 B 失败，那么线程就进入阻塞状态，一旦发生死锁，就没有任何机会来唤醒阻塞的线程。但如果阻塞状态的线程能够响应中断信号，也就是说当我们给阻塞的线程发送中断信号的时候，能够唤醒它，那它就有机会释放曾经持有的锁 A。这样就破坏了不可抢占条件了。

2.支持超时。如果线程在一段时间之内没有获取到锁，不是进入阻塞状态，而是返回一个错误，那这个线程也有机会释放曾经持有的锁。这样也能破坏不可抢占条件。

3.非阻塞地获取锁。如果尝试获取锁失败，并不进入阻塞状态，而是直接返回，那这个线程也有机会释放曾经持有的锁。这样也能破坏不可抢占条件。

// 支持中断的API

void lockInterruptibly()

throws InterruptedException;

// 支持超时的API

boolean tryLock(long time, TimeUnit unit)

throws InterruptedException;

// 支持非阻塞获取锁的API

boolean tryLock();

既然有 Java SDK 里面提供了 Lock，为啥还要提供一个 Semaphore ？

Semaphore 还有一个功能是 Lock 不容易实现的，那就是：Semaphore 可以允许多个线程访问一个临界区。比较常见的需求就是我们工作中遇到的各种池化资源，例如连接池、对象池、线程池等等。可以用Semaphore实现限流器。

ReadWriteLock读写锁：

实现类ReentrantReadWriteLock，支持锁降级不支持锁升级。

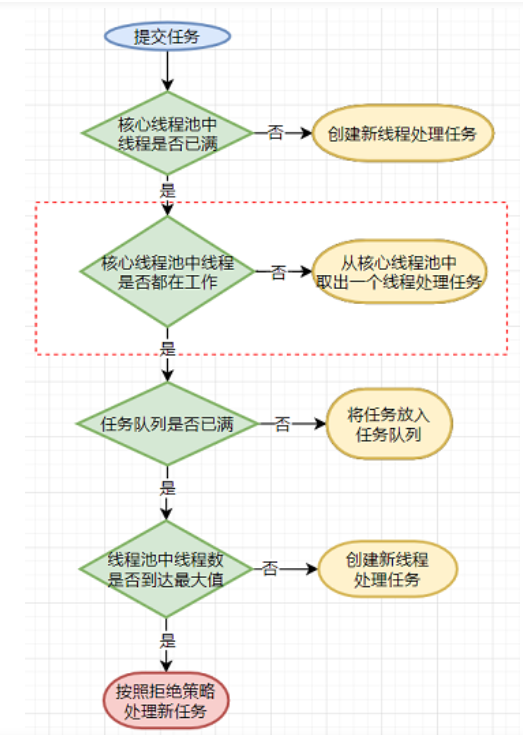
StampedLock 支持的三种锁模式：

写锁、悲观读锁和乐观读。乐观读没有锁。StampedLock 不支持重入。

线程池：

<https://blog.csdn.net/weixin_43461520/article/details/109412916>

<https://javaguide.cn/java/concurrent/java-concurrent-questions-03.html#%E4%BB%80%E4%B9%88%E6%98%AF%E7%BA%BF%E7%A8%8B%E6%B1%A0>



FixedThreadPool和SingleThreadExecutor(一个线程)都是固定线程数(核心线程数和最大线程数一样)，但是阻塞队列是无界的(使用队列的无参构造，最大容量为int最大值)，会堆积大量请求。OOM。

CachedThreadPool核心线程数为0，最大线程数为int最大值。会创建大量请求。OOM。

ScheduledThreadPool 和 SingleThreadScheduledExecutor : 使用的无界的延迟阻塞队列DelayedWorkQueue，任务队列最大长度为 Integer.MAX\_VALUE,可能堆积大量的请求，从而导致 OOM。

