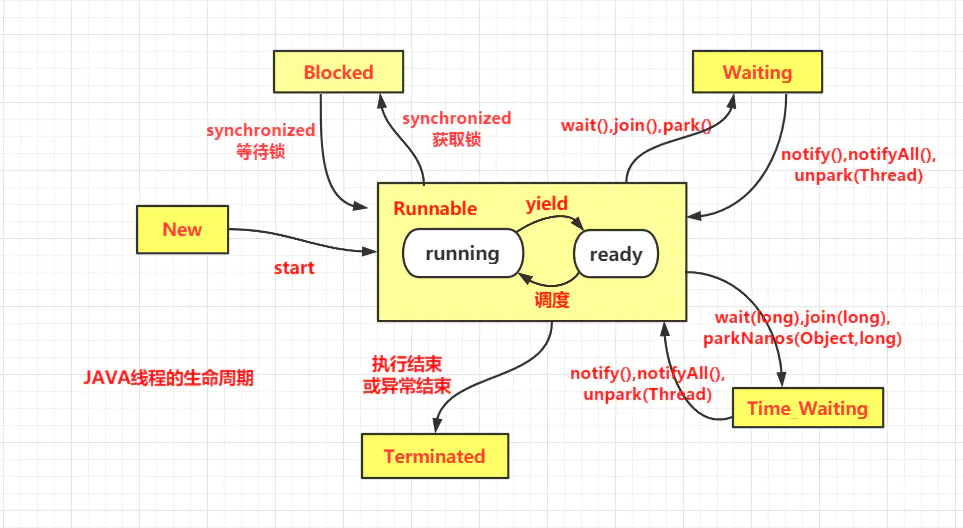
1，Java线程与系统线程

* java的线程是基于操作系统原生的线程模型(非用户态)，通过系统调用，将程序的线程交给系统调度执行
* java线程拥有属于自己的虚拟机栈，当JVM将栈、程序计数器、工作内存等准备好后，会分配一个系统原生线程来执行。Java线程结束，原生线程随之被回收
* 原生线程初始化完毕，会调Java线程的run方法。当JAVA线程结束时，则释放原生线程和Java线程的所有资源
* java方法的执行对应虚拟机栈的一个栈帧，用于存储局部变量、操作数栈、动态链接、方法出口等

1. 生命周期



用stop会强行终止线程，导致线程所持有的全部锁突然释放(不可控制)，而被锁突同步的逻辑遭到破坏。不建议使用

interrupt函数中断线程，但它不一定会让线程退出的。它比stop函数优雅，可控制

当线程处于调用sleep、wait的阻塞状态时，会抛出InterruptedException，代码内部捕获，然后结束线程

线程处于非阻塞状态，则需要程序自己调用interrupted()判断，再决定是否退出

start是Thread类的方法，从线程的生命周期来看，start的执行并不意味着新线程的执行，而是让JVM分配虚拟机栈，进入Runnable状态，start的执行还是在旧线程上

run则是新线程被系统调度，获取CPU时，执行的方法，必须是继承Thread或者是实现Runnable接口

Thread.sleep与Object.wait区别

Thread.sleep需要指定休眠时间，时间一到继续运行;和锁机制无关，不能加锁也不用释放锁

Object.wait需要在synchronized中调用，否则报IllegalMonitorStateException错误。wait方法会释放锁，需要调用相同锁对象Object.notify来唤醒线程

1. 线程池

优点：

线程的每次使用时创建，结束再销毁，是非常巨大的开销。若用缓存的策略(线程池)，暂存曾经创建的线程，复用这些线程，可以减少程序的消耗，提高线程的利用率

降低资源消耗：重复利用线程可降低线程创建和销毁造成的消耗

提高响应速度：当任务到达时，不需要等待线程创建就能立即执行

提高线程的可管理性：使用线程池可以进行统一的分配，监控和调优

ThreadPoolExecutor

1 corePoolSize：核心线程数，线程池维持的线程数量

2 maximumPoolSize：最大的线程数，当阻塞队列不可再接受任务时且maximumPoolSize大于corePoolSize则会创建非核心线程来执行。无任务执行时，会被销毁

3 keepAliveTime：非核心线程在闲暇间的存活时间

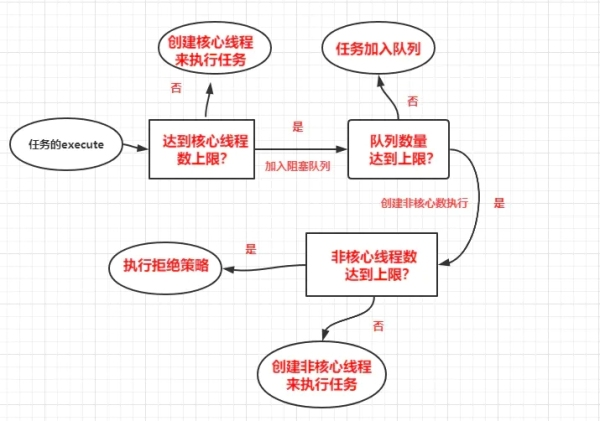
4 TimeUnit：和keepAliveTime配合使用，表示keepAliveTime参数的时间单位

5 workQueue：正在执行的任务数超过corePoolSize时，任务的等待阻塞队列

6 threadFactory：线程的创建工厂

7 handler：拒绝策略，线程数达到了maximumPoolSize，还有任务提交则使用拒绝策略处理。

线程池执行流程：



Executors的四种线程池：

newFixedThreadPool：指定核心线程数，队列是LinkedBlockingQueue无界阻塞队列，永远不可能拒绝任务;适合用在稳定且固定的并发场景，建议线程设置为CPU核数

newCachedThreadPool：核心池大小为0，线程池最大线程数为最大整型，任务提交先加入到阻塞队列中，非核心线程60s没任务执行则销毁，阻塞队列为SynchronousQueue。newCachedThreadPool会不断的创建新线程来执行任务，不建议用

newScheduledThreadPool：ScheduledThreadPoolExecutor(STPE)其实是ThreadPoolExecutor的子类，可指定核心线程数，队列是STPE的内部类DelayedWorkQueue。「STPE的好处是 A 延时可执行任务，B 可执行带有返回值的任务」

newSingleThreadExecutor：和newFixedThreadPool构造方法一致，不过线程数被设置为1了。SingleThreadExecutor比new个线程的好处是;「线程运行时抛出异常的时候会有新的线程加入线程池完成接下来的任务;阻塞队列可以保证任务按FIFO执行」

线程池关闭：

shutdownNow：线程池拒接收新任务，同时立马关闭线程池(执行中的会继续执行完)，队列的任务不再执行，返回未执行任务List

shuwdown：线程池拒接收新任务，同时等待线程池里的任务执行完毕后关闭线程池

1. ThreadLocal

适用于每个线程需要自己独立的实例且该实例需要在多个方法中被使用，也即变量在线程间隔离而在方法或类间共享的场景。

ThreadLocal 提供了线程本地的实例。它与普通变量的区别在于，每个使用该变量的线程都会初始化一个完全独立的实例副本。ThreadLocal 变量通常被private static修饰。当一个线程结束时，它所使用的所有 ThreadLocal 相对的实例副本都可被回收。

ThreadLocal 并不解决线程间共享数据的问题

ThreadLocal 通过隐式的在不同线程内创建独立实例副本避免了实例线程安全的问题

每个线程持有一个 Map 并维护了 ThreadLocal 对象与具体实例的映射，该 Map 由于只被持有它的线程访问，故不存在线程安全以及锁的问题

ThreadLocalMap 的 Entry 对 ThreadLocal 的引用为弱引用，避免了 ThreadLocal 对象无法被回收的问题

ThreadLocalMap 的 set 方法通过调用 replaceStaleEntry 方法回收键为 null 的 Entry 对象的值（即为具体实例）以及 Entry 对象本身从而防止内存泄漏

ThreadLocal 适用于变量在线程间隔离且在方法间共享的场景。