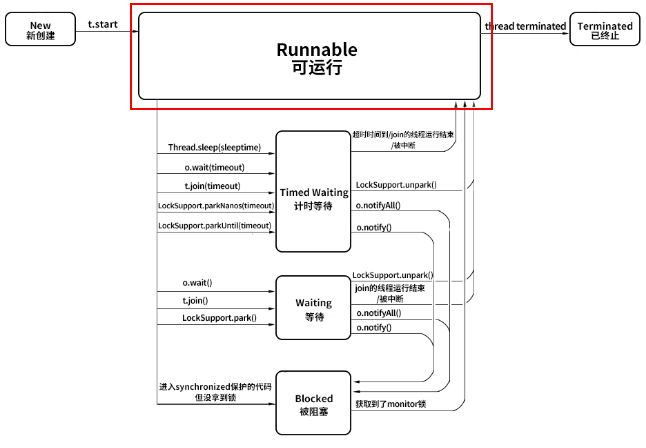
1. **实现线程实质只有一种方式：thread**



1. **实现 Runnable 接口比继承 Thread 类实现线程要好**
2. 实现了 Runnable 与 Thread 类的解耦
3. 在某些情况下可以提高性能，使用实现 Runnable 接口的方式，就可以把任务直接传入线程池，使用一些固定的线程来完成任务，不需要每次新建销毁线程，大大降低了性能开销
4. java 语言不支持双继承，使用runnable易于扩展
5. **如何正确停止线程？为什么 volatile 标记位的停止方法是错误的？**
6. 最正确的停止线程的方式是使用 interrupt。并不是立即停止
7. 最佳实现方式有两种：A：方法签名抛异常 B：run() 强制 try/catch
8. InterruptedException异常需要再次调用interrupt，否则变成“屏蔽了中断请求”
9. 错误的停止方法：stop()，suspend() 和 resume()等，stop会立即中断，导致数据完整性问题，suspend() 和 resume()不会释放锁
10. volatile标记位停止线程不完美（public volatile boolean canceled = false;），如使用ArrayBlockingQueue作生产者消费者模式，ArrayBlockingQueue阻塞了当前线程，导致线程挂起，无法中断
11. **线程的 6 种状态之间的切换**
12. 线程的状态是需要按照箭头方向来走的，比如线程从 New 状态是不可以直接进入 Blocked 状态的，它需要先经历 Runnable 状态
13. 线程生命周期不可逆：一旦进入 Runnable 状态就不能回到 New 状态；一旦被终止就不可能再有任何状态的变化。所以一个线程只能有一次 New 和 Terminated 状态，只有处于中间状态才可以相互转换。
14. 如图：



1. **wait/notify/notifyAll 方法的使用注意事项**
2. wait 必须在 synchronized 保护的同步代码中使用，因为需要持有对象的 monitor 锁
3. 用while条件判断，避免产生“虚假唤醒”（spurious wakeup）的问题
4. wait/notify 和 sleep 方法的异同

相同点：

* 它们都可以让线程阻塞。
* 它们都可以响应 interrupt 中断：在等待的过程中如果收到中断信号，都可以进行响应，并抛出 InterruptedException 异常。

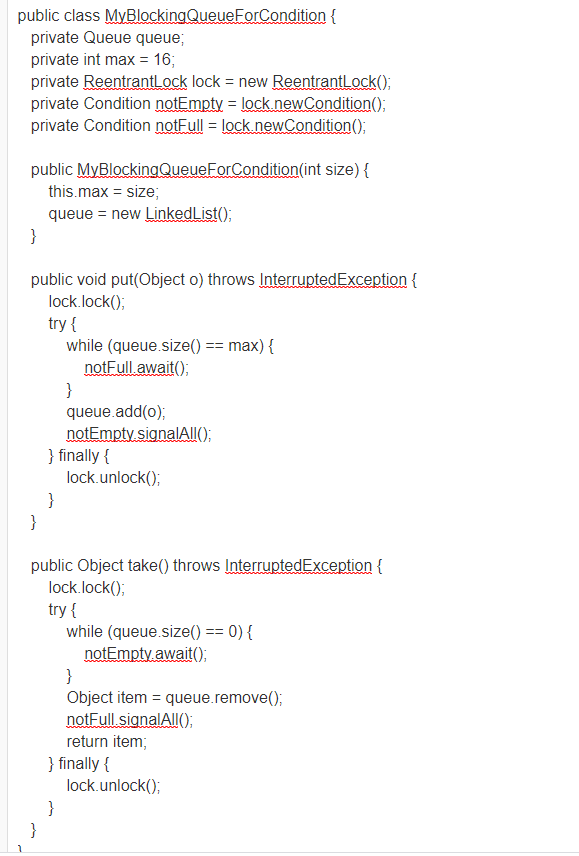
不同点：

* wait 方法必须在 synchronized 保护的代码中使用，而 sleep 方法并没有这个要求。
* 在同步代码中执行 sleep 方法时，并不会释放 monitor 锁，但执行 wait 方法时会主动释放 monitor 锁。
* sleep 方法中会要求必须定义一个时间，时间到期后会主动恢复，而对于没有参数的 wait 方法而言，意味着永久等待，直到被中断或被唤醒才能恢复，它并不会主动恢复。
* wait/notify 是 Object 类的方法，而 sleep 是 Thread 类的方法。

1. **有哪几种实现生产者消费者模式的方法？**
2. BlockingQueue 实现生产者消费者模式



1. Condition 实现生产者消费者模式



1. wait/notify 实现生产者消费者模式

