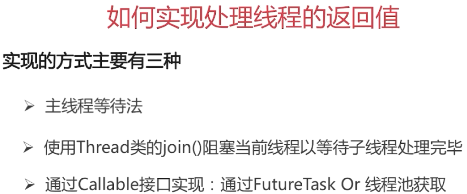
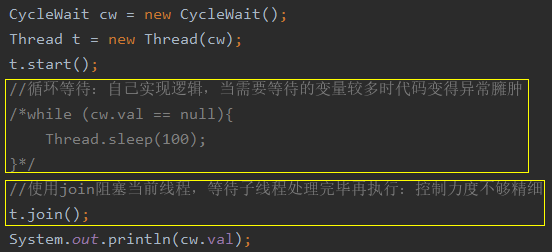
## 多线程

Thread是实现了Runnable接口的类，使得run支持多线程。因类的单一继承原则，推荐多使用Runnable接口。另外Thread类有start方法，Runnable中没有这个方法，所以用Runnable接口实现多线程的方式在启动线程时需要用Thread类来启动，然后将Runnable通过Thread的构造函数传参来启动线程。

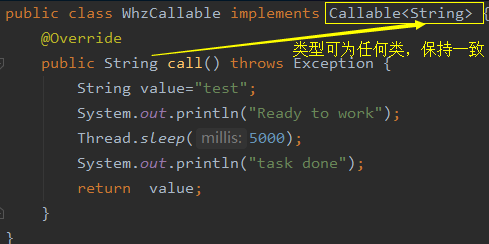


### 一、处理线程的返回值

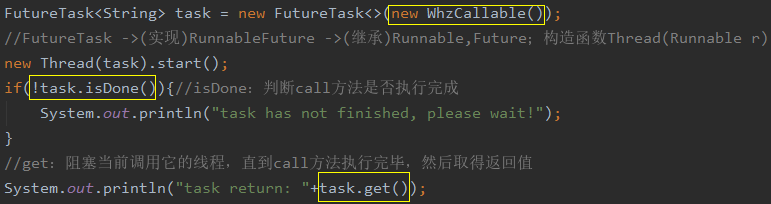
1. 主线程等待法。缺点：需要自己实现循环等待逻辑，当需要等待的变量较多时代码变得异常臃肿。
2. 使用thread类的join()阻塞当前线程以等待子线程处理完毕。缺点：控制力度不够精细。



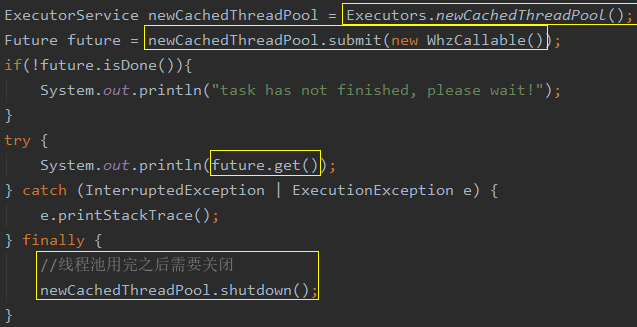
1. 通过callable接口实现，通过FutureTask或者线程池获取。



FutureTask：

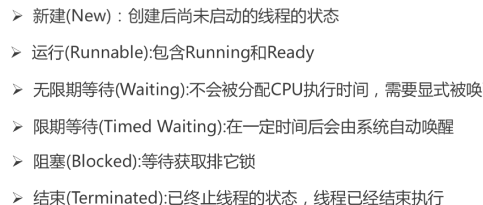


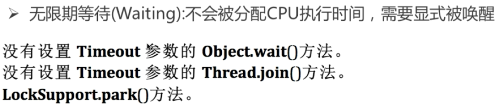
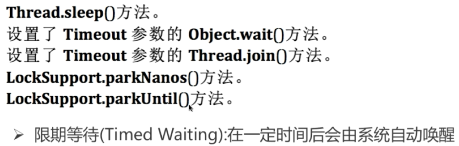
线程池：



### 二、线程状态

新建（NEW）



### 三、sleep和wait区别

**sleep**

1. sleep是Thread的一个静态（static）方法。可在任何地方使用。Runnable实现的线程也可以使用sleep方法。而且避免了线程之前相互调用sleep()方法，引发死锁。
2. sleep()执行时需要赋予一个沉睡时间。在沉睡期间（阻塞线程期间），CPU会放弃这个线程，执行其他任务。当沉睡时间到了之后，该线程会自动苏醒，不过此时线程不会立刻被执行，而是要等CPU分配资源，和其他线程进行竞争。
3. 此外如果这个线程之前获取了一个锁，在沉睡期间，这个锁不会释放。其他等待这个锁的程序，必须等待这个线程醒来，且执行完后才能运行。

**wait**

1.wait()是Object类的一个方法。当调用wait()方法时，该线程会进入和该对象相关的等待池中，并释放它所拥有的机锁。

2.执行wait()后，必须使用notify()方法或notifyAll()方法或设置等待时间(wait(long time))唤醒在等待线程池中的线程。

3.wait()必须放在synchronized 方法或block中，否则会在运行时报“java.lang.IllegalMonitorStateException”异常

### 四、notifyAll和notify

**锁池：**

假设线程A已经拥有了某个对象(不是类)的锁，而其它线程B、C想要调用这个对象的某个synchronized方法(或者块)，由于B、C线程在进入对象的synchronized方法(或者块)之前必须先获得该对象锁的拥有权，而恰好该对象的锁目前正被线程A所占用，此时B、C线程就会被阻塞，进入一个地方去等待锁的释放，这个地方便是该对象的锁池。

**等待池：**

假设线程A调用了某个对象的wait()方法，线程A就会释放该对象的锁，同时线程A就进入到了该对象的等待池中，进入到等待池中的线程不会去竞争该对象的锁。

**notifyAll和notify区别：**

notifyAll()会让所有处于等待池的线程全部进入锁池去竞争获取锁的机会。

notify()只会随机选取一个处于等待池中的线程进入锁池去竞争获取锁的机会。

### 五、yield

yield()让当前正在运行的线程回到可运行状态，以允许具有相同优先级的其他线程获得运行的机会。因此，使用yield()的目的是让具有相同优先级的线程之间能够适当的轮换执行。但是，实际中无法保证yield()达到让步的目的，因为，让步的线程可能被线程调度程序再次选中。

打个比方：现在有很多人在排队上厕所，好不容易轮到这个人上厕所了，突然这个人说：“我要和大家来个竞赛，看谁先抢到厕所！”，然后所有的人在同一起跑线冲向厕所，有可能是别人抢到了，也有可能他自己有抢到了。

应用场景：如果你觉得一个线程不是那么重要，或者优先级非常低，而且又害怕它会占用太多的CPU资源，那么可以在适当的时候调用Thread.yield()，给予其他重要线程更多的工作机会。

### 六、interrupt的应用场景

通常interrupt适用于在线程执行中的循环标记判断。

interrupt机制并不仅仅是一个中断状态位的变化和检测，它还可以进行中断异常的处理。

interrupt方法是用于中断线程的，调用该方法的线程的状态将被置为"中断"状态。注意：线程中断仅仅是设置线程的中断状态位，不会停止线程。所以当一个线程处于中断状态时，如果再由wait、sleep以及join三个方法引起的阻塞，那么JVM会将线程的中断标志重新设置为false，并抛出一个InterruptedException异常，然后开发人员可以中断状态位“的本质作用-----就是程序员根据try-catch功能块捕捉jvm抛出的InterruptedException异常来做各种处理，比如如何退出线程。总之interrupt的作用就是需要用户自己去监视线程的状态位并做处理。”

同时可以做这样的理解：

Thread.currentThread().interrupt(); 这个用于清除中断状态，这样下次调用Thread.interrupted()方法时就会一直返回为true，因为中断标志已经被恢复了。

而调用isInterrupted()只是简单的查询中断状态，不会对状态进行修改。

interrupt（）是用来设置中断状态的。返回true说明中断状态被设置了而不是被清除了。我们调用sleep、wait等此类可中断（throw InterruptedException）方法时，一旦方法抛出InterruptedException，当前调用该方法的线程的中断状态就**会被jvm自动清除**了，就是说我们调用该线程的isInterrupted 方法时是返回false。如果你想保持中断状态，可以再次调用interrupt方法设置中断状态。这样做的原因是，java的中断并不是真正的中断线程，而只设置标志位（中断位）来通知用户。如果你捕获到中断异常，说明当前线程已经被中断，不需要继续保持中断位。

interrupted是静态方法，返回当前线程的中断状态。例如，如果当前线程被中断（没有抛出中断异常，否则中断状态就会被清除），你调用interrupted方法，第一次会返回true。然后，当前线程的中断状态被方法内部清除了。第二次调用时就会返回false。如果你刚开始一直调用isInterrupted，则会一直返回true，除非中间线程的中断状态被其他操作清除了。

------------------------------------------------------------------------------------

一个线程不应该由其他线程来强制中断或停止，而是应该由线程自己自行停止。

所以，Thread.stop, Thread.suspend, Thread.resume 都已经被废弃了。

而 Thread.interrupt 的作用其实也不是中断线程，而是「通知线程应该中断了」，

具体到底中断还是继续运行，应该由被通知的线程自己处理。

具体来说，当对一个线程，调用 interrupt() 时，

① 如果线程处于被阻塞状态（例如处于sleep, wait, join 等状态），那么线程将立即退出被阻塞状态，并抛出一个InterruptedException异常。仅此而已。

② 如果线程处于正常活动状态，那么会将该线程的中断标志设置为 true，仅此而已。被设置中断标志的线程将继续正常运行，不受影响。

interrupt() 并不能真正的中断线程，需要被调用的线程自己进行配合才行。

也就是说，一个线程如果有被中断的需求，那么就可以这样做。

① 在正常运行任务时，经常检查本线程的中断标志位，如果被设置了中断标志就自行停止线程。

② 在调用阻塞方法时正确处理InterruptedException异常。（例如，catch异常后就结束线程。）

## 线程池

newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。

newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。

newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。

newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定。