[https://mp.weixin.qq.com/s?\_\_biz=MzA5MTkxMDQ4MQ==&mid=2648933134&idx=1&sn=65c2b9982bb6935c54ff33082f9c111f&chksm=88621b30bf159226d41607292a1dc83186f8928744dbc44acfda381266fa2cdc006177b44095&token=773938509&lang=zh\_CN&scene=21#wechat\_redirect](https://mp.weixin.qq.com/s?__biz=MzA5MTkxMDQ4MQ==&mid=2648933134&idx=1&sn=65c2b9982bb6935c54ff33082f9c111f&chksm=88621b30bf159226d41607292a1dc83186f8928744dbc44acfda381266fa2cdc006177b44095&token=773938509&lang=zh_CN&scene=21" \l "wechat_redirect)

# JUC中等待多线程完成的工具类CountDownLatch

## 本篇内容

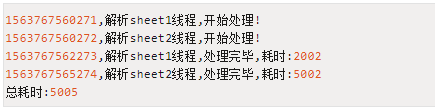
* **介绍CountDownLatch及使用场景**
* **提供几个示例介绍CountDownLatch的使用**
* **手写一个并行处理任务的工具类**

假如有这样一个需求，当我们需要解析一个Excel里多个sheet的数据时，可以考虑使用多线程，每个线程解析一个sheet里的数据，等到所有的sheet都解析完之后，程序需要统计解析总耗时。分析一下：解析每个sheet耗时可能不一样，总耗时就是最长耗时的那个操作。

我们能够想到的最简单的做法是使用join，代码如下：



输出：



代码中启动了2个解析sheet的线程，第一个耗时2秒，第二个耗时5秒，最终结果中总耗时：5秒。上面的关键技术点是线程的 join()方法，此方法会让当前线程等待被调用的线程完成之后才能继续。可以看一下join的源码，内部其实是在synchronized方法中调用了线程的wait方法，最后被调用的线程执行完毕之后，由jvm自动调用其notifyAll()方法，唤醒所有等待中的线程。这个notifyAll()方法是由jvm内部自动调用的，jdk源码中是看不到的，需要看jvm源码，有兴趣的同学可以去查一下。所以JDK不推荐在线程上调用wait、notify、notifyAll方法。

而在JDK1.5之后的并发包中提供的CountDownLatch也可以实现join的这个功能。

## CountDownLatch介绍

**CountDownLatch称之为闭锁，它可以使一个或一批线程在闭锁上等待，等到其他线程执行完相应操作后，闭锁打开，这些等待的线程才可以继续执行**。确切的说，闭锁在内部维护了一个倒计数器。通过该计数器的值来决定闭锁的状态，从而决定是否允许等待的线程继续执行。

### 常用方法

* public CountDownLatch(int count)：构造方法，count表示计数器的值，不能小于0，否者会报异常。
* public void await() throws InterruptedException：调用await()会让当前线程等待，直到计数器为0的时候，方法才会返回，此方法会响应线程中断操作。
* public boolean await(long timeout, TimeUnit unit) throws InterruptedException：限时等待，在超时之前，计数器变为了0，方法返回true，否者直到超时，返回false，此方法会响应线程中断操作。
* public void countDown()：让计数器减1

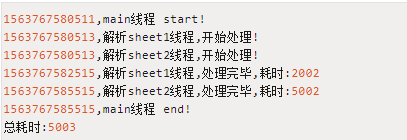
### 使用步骤

1. 创建CountDownLatch对象
2. 调用其实例方法 await()，让当前线程等待
3. 调用 countDown()方法，让计数器减1
4. 当计数器变为0的时候， await()方法会返回

### 示例1：一个简单的示例



输出：



从结果中看出，效果和join实现的效果一样，代码中创建了计数器为2的 CountDownLatch，主线程中调用 countDownLatch.await();会让主线程等待，t1、t2线程中模拟执行耗时操作，最终在finally中调用了 countDownLatch.countDown();,此方法每调用一次，CountDownLatch内部计数器会减1，当计数器变为0的时候，主线程中的await()会返回，然后继续执行。注意：上面的 countDown()这个是必须要执行的方法，所以放在finally中执行。

## 示例2：等待指定的时间

还是上面的示例，2个线程解析2个sheet，主线程等待2个sheet解析完成。主线程说，我等待2秒，你们还是无法处理完成，就不等待了，直接返回。如下代码：



输出：



从输出结果中可以看出，线程2耗时了5秒，主线程耗时了2秒，主线程中调用 countDownLatch.await(2,TimeUnit.SECONDS);，表示最多等2秒，不管计数器是否为0，await方法都会返回，若等待时间内，计数器变为0了，立即返回true，否则超时后返回false。

## 示例3：2个CountDown结合使用的示例

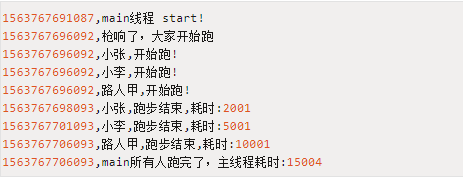
有3个人参加跑步比赛，需要先等指令员发指令枪后才能开跑，所有人都跑完之后，指令员喊一声，大家跑完了。

示例代码：



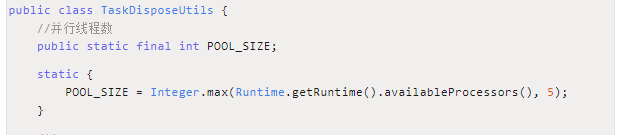


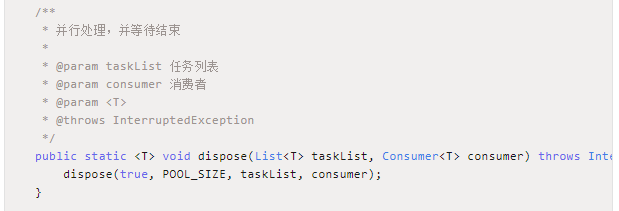
输出：



代码中，t1、t2、t3启动之后，都阻塞在 commanderCd.await();，主线程模拟发枪准备操作耗时5秒，然后调用 commanderCd.countDown();模拟发枪操作，此方法被调用以后，阻塞在 commanderCd.await();的3个线程会向下执行。主线程调用 countDownLatch.await();之后进行等待，每个人跑完之后，调用 countDown.countDown();通知一下 countDownLatch让计数器减1，最后3个人都跑完了，主线程从 countDownLatch.await();返回继续向下执行。

## 手写一个并行处理任务的工具类

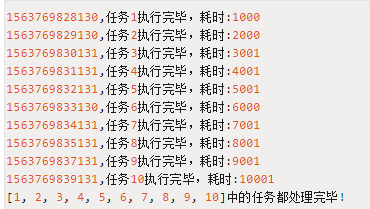








运行代码输出：



**TaskDisposeUtils是一个并行处理的工具类，可以传入n个任务内部使用线程池进行处理，等待所有任务都处理完成之后，方法才会返回。比如我们发送短信，系统中有1万条短信，我们使用上面的工具，每次取100条并行发送，待100个都处理完毕之后，再取一批按照同样的逻辑发送。**