# [Java并发编程：CountDownLatch、CyclicBarrier和Semaphore](http://www.cnblogs.com/dolphin0520/p/3920397.html)

[**并发编程 04——闭锁CountDownLatch 与 栅栏CyclicBarrier**](http://www.cnblogs.com/xingele0917/p/3962288.html)

用给定的计数 初始化 CountDownLatch。由于调用了 countDown() 方法，所以在当前计数到达零之前，await 方法会一直受阻塞。之后，会释放所有等待的线程，await 的所有后续调用都将立即返回。这种现象只出现一次——计数无法被重置。如果需要重置计数，请考虑使用 CyclicBarrier。

**用它可以实现类似计数器的功能。比如有一个任务A，它要等待其他4个任务执行完毕之后才能执行，此时就可以利用CountDownLatch来实现这种功能了。**

两个看上去有点像的类，都在java.util.concurrent下，都可以用来表示代码运行到某个点上，二者的区别在于：

（1）**CyclicBarrier的某个线程运行到某个点上之后，该线程即停止运行，直到所有的线程都到达了这个点，所有线程才重新运行；CountDownLatch则不是，某线程运行到某个点上之后，只是给某个数值-1而已，该线程继续运行**

（2）CyclicBarrier只能唤起一个任务，CountDownLatch可以唤起多个任务

（3）CyclicBarrier可重用，CountDownLatch不可重用，计数值为0该CountDownLatch就不可再用了

1）CountDownLatch和CyclicBarrier都能够实现线程之间的等待，只不过它们侧重点不同：

CountDownLatch一般用于某个线程A等待若干个其他线程执行完任务之后，它才执行；而CyclicBarrier一般用于一组线程互相等待至某个状态，然后这一组线程再同时执行；

另外，CountDownLatch是不能够重用的，而CyclicBarrier是可以重用的。

2）Semaphore其实和锁有点类似，它一般用于控制对某组资源的访问权限。

CountDownLatch : 一个线程(或者多个)， 等待另外N个线程完成某个事情之后才能执行。

CyclicBarrier        : N个线程相互等待，任何一个线程没有到达或完成时，所有的线程都必须互相等待。

**在多线程程序设计中，经常会遇到一个线程等待一个或多个线程的场景，遇到这样的场景应该如何解决？**

如果是一个线程等待一个线程，则可以通过await()和notify()来实现；

如果是一个线程等待多个线程，则就可以使用CountDownLatch和CyclicBarrier来实现比较好的控制。

**下面详细描述下CyclicBarrier的应用场景：**

**假设有只有的一个场景：每个线程代表一个跑步运动员，当运动员都准备好后，才一起出发，只要有一个人没有准备好，大家都等待.**

有四个游戏玩家玩游戏，游戏有三个关卡，每个关卡必须要所有玩家都到达后才能允许通关。

其 实这个场景里的玩家中如果有玩家A先到了关卡1，他必须等待其他所有玩家都到达关卡1时才能通过，也就是说线程之间需要互相等待，这和 CountDownLatch的应用场景有区别，CountDownLatch里的线程是到了运行的目标后继续干自己的其他事情，而这里的线程需要等待其 他线程后才能继续完成下面的工作。