二

54 CyclicBarrier 和 CountdownLatch 有什么异同?

本课时我们主要介绍 CyclicBarrier 和 CountDownLatch 有什么不同。

CyclicBarrier

作用

CyclicBarrier 和 CountDownLatch 确实有一定的相似性，它们都能阻塞一个或者一组线程，直到某种预定的条件达到之后，这些之前在等待的线程才会统一出发，继续向下执行。正因为它们有这个相似点，你可能会认为它们的作用是完全一样的，其实并不是。

CyclicBarrier 可以构造出一个集结点，当某一个线程执行 await() 的时候，它就会到这个集结点开始等待，等待这个栅栏被撤销。直到预定数量的线程都到了这个集结点之后，这个栅栏就会被撤销，之前等待的线程就在此刻统一出发，继续去执行剩下的任务。

举一个生活中的例子。假设我们班级春游去公园里玩，并且会租借三人自行车，每个人都可以骑，但由于这辆自行车是三人的，所以要凑齐三个人才能骑一辆，而且从公园大门走到自行车驿站需要一段时间。那么我们模拟这个场景，写出如下代码：

```
public class CyclicBarrierDemo {

    public static void main(String[] args) {

        CyclicBarrier cyclicBarrier = new CyclicBarrier(3);

        for (int i = 0; i < 6; i++) {

            new Thread(new Task(i + 1, cyclicBarrier)).start();

        }

    }

    static class Task implements Runnable {

        private int id;
```

```
        private CyclicBarrier cyclicBarrier;
        public Task(int id, CyclicBarrier cyclicBarrier) {
            this.id = id;
            this.cyclicBarrier = cyclicBarrier;
        }
        @Override
        public void run() {
            System.out.println("同学" + id + "现在从大门出发，前往自行车驿站");
            try {
                Thread.sleep((long) (Math.random() * 10000));
                System.out.println("同学" + id + "到了自行车驿站，开始等待其他人到达");
                cyclicBarrier.await();
                System.out.println("同学" + id + "开始骑车");
            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (BrokenBarrierException e) {
                e.printStackTrace();
            }
        }
    }
}
```

在这段代码中可以看到，首先建了一个参数为 3 的 CyclicBarrier，参数为 3 的意思是需要等待 3 个线程到达这个集结点才统一放行；然后我们又在 for 循环中去开启了 6 个线程，每个线程中执行的 Runnable 对象就在下方的 Task 类中，直接看到它的 run 方法，它首先会打印出"同学某某现在从大门出发，前往自行车驿站"，然后是一个随机时间的睡眠，这就代表着从大门开始步行走到自行车驿站的时间，由于每个同学的步行速度不一样，所以时间用随机值来模拟。

当同学们都到了驿站之后，比如某一个同学到了驿站，首先会打印出“同学某某到了自行车驿站，开始等待其他人到达”的消息，然后去调用 CyclicBarrier 的 await() 方法。一旦它调

用了这个方法，它就会陷入等待，直到三个人凑齐，才会继续往下执行，一旦开始继续往下执行，就意味着 3 个同学开始一起骑车了，所以打印出“某某开始骑车”这个语句。

接下来我们运行一下这个程序，结果如下所示：

```
同学1现在从大门出发，前往自行车驿站
同学3现在从大门出发，前往自行车驿站
同学2现在从大门出发，前往自行车驿站
同学4现在从大门出发，前往自行车驿站
同学5现在从大门出发，前往自行车驿站
同学6现在从大门出发，前往自行车驿站
同学5到了自行车驿站，开始等待其他人到达
同学2到了自行车驿站，开始等待其他人到达
同学3到了自行车驿站，开始等待其他人到达
同学3开始骑车
同学5开始骑车
同学2开始骑车
同学6到了自行车驿站，开始等待其他人到达
同学4到了自行车驿站，开始等待其他人到达
同学1到了自行车驿站，开始等待其他人到达
同学1开始骑车
同学6开始骑车
同学4开始骑车
```

可以看到 6 个同学纷纷从大门出发走到自行车驿站，因为每个人的速度不一样，所以会有 3 个同学先到自行车驿站，不过在这 3 个先到的同学里面，前面 2 个到的都必须等待第 3 个人到齐之后，才可以开始骑车。后面的同学也一样，由于第一辆车已经被骑走了，第二辆车依然也要等待 3 个人凑齐才能统一发车。

要想实现这件事情，如果你不利用 CyclicBarrier 去做的话，逻辑可能会非常复杂，因为你也不清楚哪个同学先到、哪个后到。而用了 CyclicBarrier 之后，可以非常简洁优雅的实现这个逻辑，这就是它的一个非常典型的应用场景。

执行动作 barrierAction

public CyclicBarrier(int parties, Runnable barrierAction)：当 parties 线程到达集结点时，继续往下执行前，会执行这一次这个动作。

接下来我们再介绍一下它的一个额外功能，就是执行动作 barrierAction 功能。CyclicBarrier 还有一个构造函数是传入两个参数的，第一个参数依然是 parties，代表需要几个线程到齐；第二个参数是一个 Runnable 对象，它就是我们下面所要介绍的 barrierAction。

当预设数量的线程到达了集结点之后，在出发的时候，便会执行这里所传入的 Runnable 对象，那么假设我们把刚才那个代码的构造函数改成如下这个样子：

```
CyclicBarrier cyclicBarrier = new CyclicBarrier(3, new Runnable() {
    @Override
    public void run() {
        System.out.println("凑齐3人了，出发！");
    }
});
```

可以看出，我们传入了第二个参数，它是一个 Runnable 对象，在这里传入了这个 Runnable 之后，这个任务就会在到齐的时候去打印"凑齐3人了，出发！"。上面的代码如果改成这个样子，则执行结果如下所示：

```
同学1现在从大门出发，前往自行车驿站
同学3现在从大门出发，前往自行车驿站
同学2现在从大门出发，前往自行车驿站
同学4现在从大门出发，前往自行车驿站
同学5现在从大门出发，前往自行车驿站
同学6现在从大门出发，前往自行车驿站
同学2到了自行车驿站，开始等待其他人到达
同学4到了自行车驿站，开始等待其他人到达
同学6到了自行车驿站，开始等待其他人到达
凑齐3人了，出发！
同学6开始骑车
```

```
同学2开始骑车

同学4开始骑车

同学1到了自行车驿站，开始等待其他人到达

同学3到了自行车驿站，开始等待其他人到达

同学5到了自行车驿站，开始等待其他人到达

凑齐3人了，出发！

同学5开始骑车

同学1开始骑车

同学3开始骑车
```

可以看出，三个人凑齐了一组之后，就会打印出“凑齐 3 人了，出发！”这样的语句，该语句恰恰是我们在这边传入 Runnable 所执行的结果。

值得注意的是，这个语句每个周期只打印一次，不是说你有几个线程在等待就打印几次，而是说这个任务只在“开闸”的时候执行一次。

CyclicBarrier 和 CountDownLatch 的异同

下面我们来总结一下 CyclicBarrier 和 CountDownLatch 有什么异同。

相同点：都能阻塞一个或一组线程，直到某个预设的条件达成发生，再统一出发。

但是它们也有很多不同点，具体如下。

- **作用对象不同**：CyclicBarrier 要等固定数量的线程都到达了栅栏位置才能继续执行，而 CountDownLatch 只需等待数字倒数到 0，也就是说 CountDownLatch 作用于事件，但 CyclicBarrier 作用于线程；CountDownLatch 是在调用了 countDown 方法之后把数字倒数减 1，而 CyclicBarrier 是在某线程开始等待后把计数减 1。
- **可重用性不同**：CountDownLatch 在倒数到 0 并且触发门闩打开后，就不能再次使用了，除非新建一个新的实例；而 CyclicBarrier 可以重复使用，在刚才的代码中也可以看出，每 3 个同学到了之后都能出发，并不需要重新新建实例。CyclicBarrier 还可以随时调用 reset 方法进行重置，如果重置时有线程已经调用了 await 方法并开始等待，那么这些线程则会抛出 BrokenBarrierException 异常。
- **执行动作不同**：CyclicBarrier 有执行动作 barrierAction，而 CountDownLatch 没这个功能。

总结

以上就是本课时的内容，在本课时中，首先介绍了 CyclicBarrier 的作用、代码示例和执行动作，然后对 CyclicBarrier 和 CountDownLatch 的异同进行了总结。

上一页 下一页