链滴

CyclicBarrier 学习

作者：dreamertn9527

原文链接：https://ld246.com/article/1514043779191

来源网站：链滴

CyclicBarrier学习

CyclicBarrier 的字面意思是可循环（Cyclic）使用的屏障（Barrier）。它要做的事情是，让一组线程达一个屏障（也可以叫同步点）时被阻塞，直到最后一个线程到达屏障时，屏障才会开门，所有被屏拦截的线程才会继续干活。线程进入屏障通过CyclicBarrier的await()方法。

CyclicBarrier默认的构造方法是CyclicBarrier(int parties)，其参数表示屏障拦截的线程数量，每个线调用await方法告诉CyclicBarrier我已经到达了屏障，然后当前线程被阻塞。

CyclicBarrier还提供一个更高级的构造函数CyclicBarrier(int parties, Runnable barrierAction)，用在线程到达屏障时，优先执行barrierAction这个Runnable对象，方便处理更复杂的业务场景。

源码分析

构造函数

```
public CyclicBarrier(int parties) {
    this(parties, null);
}
public int getParties() {
    return parties;
}
```

实现原理：在CyclicBarrier的内部定义了一个Lock对象，每当一个线程调用CyclicBarrier的await方时，将剩余拦截的线程数减1，然后判断剩余拦截数是否为0，如果不是，进入Lock对象的条件队列待。如果是，执行barrierAction对象的Runnable方法，然后将锁的条件队列中的所有线程放入锁等队列中，这些线程会依次的获取锁、释放锁，接着先从await方法返回，再从CyclicBarrier的await方中返回。

await源码

```
public int await() throws InterruptedException, BrokenBarrierException {
    try {
        return dowait(false, 0L);
    } catch (TimeoutException toe) {
        throw new Error(toe); // cannot happen
    }
}
```

dowait源码

```
private int dowait(boolean timed, long nanos)
    throws InterruptedException, BrokenBarrierException,
           TimeoutException {
    final ReentrantLock lock = this.lock;
    lock.lock();
    try {
        final Generation g = generation;

        if (g.broken)
            throw new BrokenBarrierException();

        if (Thread.interrupted()) {
```

```
                breakBarrier();
                throw new InterruptedException();
            }

            int index = --count;
            if (index == 0) {  // tripped
                boolean ranAction = false;
                try {
                    final Runnable command = barrierCommand;
                    if (command != null)
                        command.run();
                    ranAction = true;
                    nextGeneration();
                    return 0;
                } finally {
                    if (!ranAction)
                        breakBarrier();
                }
            }

            // loop until tripped, broken, interrupted, or timed out
            for (;;) {
                try {
                    if (!timed)
                        trip.await();
                    else if (nanos > 0L)
                        nanos = trip.awaitNanos(nanos);
                } catch (InterruptedException ie) {
                    if (g == generation && ! g.broken) {
                        breakBarrier();
                        throw ie;
                    } else {
                        // We're about to finish waiting even if we had not
                        // been interrupted, so this interrupt is deemed to
                        // "belong" to subsequent execution.
                        Thread.currentThread().interrupt();
                    }
                }

                if (g.broken)
                    throw new BrokenBarrierException();

                if (g != generation)
                    return index;

                if (timed && nanos <= 0L) {
                    breakBarrier();
                    throw new TimeoutException();
                }
            }
        } finally {
            lock.unlock();
        }
    }
```

当最后一个线程到达屏障点，也就是执行dowait方法时，会在return 0 返回之前调用finally块中的brekBarrier方法。

breakBarrier源代码

```
private void breakBarrier() {
    generation.broken = true;
    count = parties;
    trip.signalAll();
}
```

CyclicBarrier主要用于一组线程之间的相互等待，而CountDownLatch一般用于一组线程等待另一组线程。实际上可以通过CountDownLatch的countDown()和await()来实现CyclicBarrier的功能。即 CuntDownLatch中的countDown()+await() = CyclicBarrier中的await()。注意：在一个线程中先调用ountDown()，然后调用await()。

其它方法：CycliBarrier对象可以重复使用，重用之前应当调用CyclicBarrier对象的reset方法。

reset源码

```
public void reset() {
    final ReentrantLock lock = this.lock;
    lock.lock();
    try {
        breakBarrier();   // break the current generation
        nextGeneration(); // start a new generation
    } finally {
        lock.unlock();
    }
}
```

例：

```
package javalearning;

import java.util.Random;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class CyclicBarrierDemo {
    private CyclicBarrier cb = new CyclicBarrier(4);
    private Random rnd = new Random();

    class TaskDemo implements Runnable{
        private String id;
        TaskDemo(String id){
            this.id = id;
        }
        @Override
        public void run(){
            try {
                Thread.sleep(rnd.nextInt(1000));
                System.out.println("Thread " + id + " will wait");
```

```
                cb.await();
                System.out.println("-------Thread " + id + " is over");
            } catch (InterruptedException e) {
            } catch (BrokenBarrierException e) {
            }
        }
    }

    public static void main(String[] args){
        CyclicBarrierDemo cbd = new CyclicBarrierDemo();
        ExecutorService es = Executors.newCachedThreadPool();
        es.submit(cbd.new TaskDemo("a"));
        es.submit(cbd.new TaskDemo("b"));
        es.submit(cbd.new TaskDemo("c"));
        es.submit(cbd.new TaskDemo("d"));
        es.shutdown();
    }
}
```

使用场景

需要所有的子任务都完成时，才执行主任务，这个时候就可以选择使用CyclicBarrier

常用方法

**public int await() throws InterruptedException,BrokenBarrierException **

在所有参与者都已经在此 barrier 上调用 await方法之前，将一直等待。如果当前线程不是将到达的后一个线程，出于调度目的，将禁用它，且在发生以下情况之一前，该线程将一直处于休眠状态：

- 最后一个线程到达；或者
- 其他某个线程中断当前线程；或者
- 其他某个线程中断另一个等待线程；或者
- 其他某个线程在等待 barrier 时超时；或者
- 其他某个线程在此 barrier 上调用 reset()。

如果当前线程：

- 在进入此方法时已经设置了该线程的中断状态；或者
- 在等待时被中断

则抛出 InterruptedException，并且清除当前线程的已中断状态。如果在线程处于等待状态时 barrier 被 reset()，或者在调用 await 时 barrier 被损坏，抑或任意一个线程正处于等待状态，则抛出 Broke BarrierException 异常。

如果任何线程在等待时被 中断，则其他所有等待线程都将抛出 BrokenBarrierException 异常，并将 arrier 置于损坏状态。

如果当前线程是最后一个将要到达的线程，并且构造方法中提供了一个非空的屏障操作，则在允许其线程继续运行之前，当前线程将运行该操作。如果在执行屏障操作过程中发生异常，则该异常将传播当前线程中，并将 barrier 置于损坏状态。

返回：

到达的当前线程的索引，其中，索引 getParties() - 1 指示将到达的第一个线程，零指示最后一个到的线程

抛出：

InterruptedException - 如果当前线程在等待时被中断

BrokenBarrierException - 如果另一个 线程在当前线程等待时被中断或超时，或者重置了 barrier，者在调用 await 时 barrier 被损坏，抑或由于异常而导致屏障操作（如果存在）失败。

相关实例

```
public class CyclicBarrierTest {

    public static void main(String[] args) throws IOException, InterruptedException {
        //如果将参数改为4，但是下面只加入了3个选手，这永远等待下去
        //Waits until all parties have invoked await on this barrier.
        CyclicBarrier barrier = new CyclicBarrier(3);

        ExecutorService executor = Executors.newFixedThreadPool(3);
        executor.submit(new Thread(new Runner(barrier, "1号选手")));
        executor.submit(new Thread(new Runner(barrier, "2号选手")));
        executor.submit(new Thread(new Runner(barrier, "3号选手")));

        executor.shutdown();
    }
}

class Runner implements Runnable {
    // 一个同步辅助类，它允许一组线程互相等待，直到到达某个公共屏障点 (common barrier point)
    private CyclicBarrier barrier;

    private String name;

    public Runner(CyclicBarrier barrier, String name) {
        super();
        this.barrier = barrier;
        this.name = name;
    }

    @Override
    public void run() {
        try {
            Thread.sleep(1000 * (new Random()).nextInt(8));
            System.out.println(name + " 准备好了...");
            // barrier的await方法，在所有参与者都已经在此 barrier 上调用 await 方法之前，将一直等
。
            barrier.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (BrokenBarrierException e) {
            e.printStackTrace();
        }
        System.out.println(name + " 起跑！ ");
```

```
    }
}
```

原文链接：CyclicBarrier 学习