Latch 和 Barrier 的用途和区别

Latch 的类说明里是这么说：

A synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads completes.

Barrier 的类说明是

A synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point. CyclicBarriers are useful in programs involving a fixed sized party of threads that must occasionally wait for each other.

可以简单的理解为，Latch 用于协调两个线程组，Barrier 用于线程组内部协调，两个的意图都是可以等待一组线程在指定代码点执行完毕。

Latch 的实现是 CountDownLatch，用于等待 N 个线程都运行过指定代码点，然后 Latch.await()的方法会从阻塞中返回。

Barrier 的实现是 CyclicBarrier，用于 N 个线程等待对方运行到指定代码点，N 个线程全部到达指定代码点后（这里可附加一个 Runnable 对象用于触发到达屏障后触发的动作），Barrier.await()会从阻塞中返回，然后执行后续的流程。

Latch 的适用的场景可以是 MR，Reducer 等待 N 个 Mapper 线程执行完毕，然后进行 Reduce 操作。

Barrier 的可以是多线程分阶段协调。

下面的例子是用 Latch 和 Barrier 实现了两个部分，每个部分 4 个阶段，每个阶段 4 个线程，有序的完成。

```
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.CyclicBarrier;
/**
 * User: xuhuiqing
 * Date: 14-1-15
 * Time: 下午5:07
 */
public class TestLatchAndBarrier {
    public static void main(String[] args) throws Exception {
        final CyclicBarrier barrier = new CyclicBarrier(4, new Runnable() {
            @Override
            public void run() {
            }
        });
        final CountDownLatch latchB = new CountDownLatch(4);
        for (int i = 0; i < 4; i++) {
            new Thread(new Runnable() {
                @Override
                public void run() {
                    try {
                        System.out.println("stage-1 ... " + Thread.currentThread().getId());
                        barrier.await();
                        System.out.println("stage-2 ... " + Thread.currentThread().getId());
                        barrier.await();
                        System.out.println("stage-3 ... " + Thread.currentThread().getId());
                        barrier.await();
                        System.out.println("stage-4 ... " + Thread.currentThread().getId());
                        latchB.countDown();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }).start();
        }
        latchB.await();
        System.out.println("==========================================");
        final CountDownLatch latch = new CountDownLatch(4);
        final CountDownLatch latch1 = new CountDownLatch(4);
        final CountDownLatch latch2 = new CountDownLatch(4);
        final CountDownLatch latch3 = new CountDownLatch(4);
        for (int i = 0; i < 4; i++) {
            new Thread(new Runnable() {
                @Override
                public void run() {
                    try {
```

```
                    System.out.println("stage-1 ... " + Thread.currentThread().getId());
                    latch.countDown();
                    latch.await();
                    System.out.println("stage-2 ... " + Thread.currentThread().getId());
                    latch1.countDown();
                    latch1.await();
                    System.out.println("stage-3 ... " + Thread.currentThread().getId());
                    latch2.countDown();
                    latch2.await();
                    System.out.println("stage-4 ... " + Thread.currentThread().getId());
                    latch3.countDown();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }
    latch3.await();
  }
}
```

结果是：

```
stage-1 ... 10
stage-1 ... 9
stage-1 ... 11
stage-1 ... 12
stage-2 ... 12
stage-2 ... 10
stage-2 ... 11
stage-2 ... 9
stage-3 ... 9
stage-3 ... 11
stage-3 ... 10
stage-3 ... 12
stage-4 ... 12
stage-4 ... 11
stage-4 ... 9
stage-4 ... 10
=========================================
stage-1 ... 13
stage-1 ... 14
stage-1 ... 15
stage-1 ... 16
stage-2 ... 16
stage-2 ... 14
stage-2 ... 13
stage-2 ... 15
stage-3 ... 15
stage-3 ... 13
stage-3 ... 14
stage-3 ... 16
stage-4 ... 16
stage-4 ... 13
stage-4 ... 14
stage-4 ... 15
```