计算多线程环境下执行任务总时间

> 通常在多线程环境下由于任务并发执行，很难计算执行任务所用时间。
> 在jdk1.5之后引入java.util.concurrent.CountDownLatch类之后，则可以简单精确的计算时间了，
>
> CountDownLatch类在实例化时会初始化一个计数值，调用await()方法之后会阻塞当前线程，调用countDown()方法之后会将初始化的计数值自减1，当计数值为0是，执行await()方法阻塞的线程将被唤醒继续执行。
> 代码如下：

```
// 代码摘录自《Effective Java》 第二版 69条 感谢Joshua Bloch 大师
public static long time(ExecutorService exec, final int concurrency,
        final Runnable action) throws InterruptedException {
    final CountDownLatch ready = new CountDownLatch(concurrency);
    final CountDownLatch start = new CountDownLatch(1);
    final CountDownLatch done = new CountDownLatch(concurrency);

    for (int i = 0; i < concurrency; i++)
        exec.execute(new Runnable() {
            @Override
            public void run() {
                ready.countDown();
                try {
                    start.await();
                    action.run();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                } finally {
                    done.countDown();
                }
            }
        });
    ready.await();
    long startNanos = System.nanoTime();
    start.countDown();
    done.await();
    return System.nanoTime() - startNanos;
}
```

> 本例中初始化三个CountDownLatch分别表示任务执行的三个阶段，准备阶段，即将开始执行阶段，完成阶段。
>
> 1. 在主线程中首先执行ready.await();阻塞主线程禁止它向下执行。
> 2. 然后在线程池execute方法中，每个线程都执行一次ready.countDown()方法修改计数值。
> 3. 在线程池中每个线程执行ready.countDown()之后，ready的计数值为零，主线程会被唤醒，然后记录action任务开始时间。
> 4. 线程池内的线程此时执行start.await()，全部进入阻塞状态，此时主线程被唤醒执行start.countDown()（start的计数值为1，只需要执行一次start.countDown()），唤醒线程池内全部线程，这就保证池内所有线程在开始执行任务时在同一起点上。
> 5. 此时主线程会执行done.await()阻塞，池内线程每完成一个任务之后会执行一次done.countDown()将done的计数器自减1，当所有任务都完成之后主线程会被唤醒，然后计算耗时。
>
> 注意：线程池提交的任务数一定要不少与ready和done的计数器，否则池内所有线程即使都执行countDown()，也不能唤醒主线程，造成线程饥饿死锁，建议计数器的值和提交的任务量相等。
>
> 技巧：方法中用到了代理模式，方法中传入的action任务，并没有直接提交到exector线程池中执行，而是有初始化一个Runnable匿名类，在此匿名类中执行action的run方法，虽然执行的不是start方法，但是它是放在一个Runnable中执行，这么写就可以在action执行的前后加入一些业务逻辑，其实在JDK的ThreadPoolExecutor源码中也是这么封装的。

```
// 测试代码
public static void main(String args[]) throws OperationException,
        InterruptedException {
    final int concurrency = 10;
    final ExecutorService exec = Executors.newFixedThreadPool(concurrency);
```

```
        long custTime = time(exec, concurrency, new Runnable() {
            Random r = new Random();
            @Override
            public void run() {
                try {
                    // 模拟执行任务
                    TimeUnit.SECONDS.sleep(r.nextInt(10));
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
        });
        System.out.println("cust time :" + custTime / 1000000000 + " s");
        exec.shutdown();
    }
```