## CountDownLatch

* Java5.0在java.util.concurrent包中提供了多种并发容器类来改进同步容器的性能。
* CountDownLatch一个同步辅助类，是一个或多个线程等待其他线程完成各自的工作后再执行。到达自己预期状态的线程会调用CountDownLatch的countDown方法，而等待的线程会调用CountDownLatch的await方法。
* 闭锁可以延迟线程的进度直到其到达终止状态，闭锁可以用来确保某些活动直到其他活动都完成才继续执行：
* 确保某个计算在其需要的所有资源都被初始化之后才继续执行；
* 确保某个服务在其依赖的所有其他服务都已经启动之后才启动；
* 等待直到某个操作所有参与者都准备就绪再继续执行。

*/\*\*  
 \* CountDownLatch:闭锁  
 \* 在完成某些运算时，只有其他所有的运算全部完成，当前运算才能继续执行  
 \*/***public class** TestCountDownLatch {  
  
 **public static void** main(String[] args) {  
  
 *//实例化一个倒计数器，5个线程协同工作* **final** CountDownLatch latch = **new** CountDownLatch(5);  
  
 LatchDemo ld = **new** LatchDemo(latch);  
  
 **long** start = System.*currentTimeMillis*();  
  
 **for** (**int** i = 0; i < 5; i++) {  
 **new** Thread(ld).start();  
 }  
 **try** {  
 *//等待，当技术减到0时，所有线程并行执行* latch.await();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 **long** end = System.*currentTimeMillis*();  
  
 System.***out***.println(**"耗费时间为："** + (end - start));  
 }  
}  
  
**class** LatchDemo **implements** Runnable {  
  
 **private** CountDownLatch **latch**;  
  
 **public** LatchDemo(CountDownLatch latch) {  
 **this**.**latch** = latch;  
 }  
  
 @Override  
 **public void** run() {  
 **synchronized** (**this**) {  
 **try** {  
 **for** (**int** i = 0; i < 50000; i++) {  
 **if** (i % 2 == 0) {  
 System.***out***.println(i);  
 }  
 }  
 } **finally** {  
 *//一个线程完成后，计数减1* **latch**.countDown();  
 }  
 }  
 }  
}

## 常见面试题

1. 解释一下CountDownLatch概念？

CountDownLatch是并发包中用来控制一个或者多个线程等待其他线程完成操作。的并发工具类。

1. CountDownLatch和CyclicBarrier的不同之处？
2. 给出一些CountDownLatch使用的例子？
3. CountDownLatch类中主要的方法？