原 【Java并发】 - CountDow

2017年06月19日 09:15:01 阅读数：6970 更多

概述

CountDownLatch是一个用来控制并发的很常见 他的线程执行到某一操作，比如说 去解析一个excel的数据，为了更快的解析则每个sheet都使用一个线程去进行解析，但 heet的解析工作完成之后才能进行 就可以使用CountDownLatch。

使用

例子：
这里有三个线程（main，thread1，thread2），其中main线程将调用countDownLatch的await方法去等待另外两个线程的某个操作的结束（调用countDownLatch的countDown方法）。

```
1  public class CountDownLatchDemo {
2
3          public static void main(String[] args) throws InterruptedException{
4
5                  CountDownLatch countDownLatch = new CountDownLatch(2){
6                          @Override
7                          public void await() throws InterruptedException {
8                                  super.await();
9                                  System.out.println(Thread.currentThread().getName() +  " count down is ok");
10                         }
11                 };
12
13                 Thread thread1 = new Thread(new Runnable() {
14                         @Override
15                         public void run() {
16                                 //do something
17                                 try {
```

```
                        } catch
                                                                    }
                            Syst                                etName() + " is done");
                            coun
                        }
                }, "thread1");

                Thread thread2 = new
                        @Override
                        public void
                            //do
                            try

                            } ca

                            }
                            System.out.println(Thread.currentThread().getName() + " is done");
                            countDownLatch.countDown();
                        }
                }, "thread2");


                thread1.start();
                thread2.start();

                countDownLatch.await();
        }

}
```


输出：

```
test thread1 is done
test thread2 is done
test thread3 is done
```

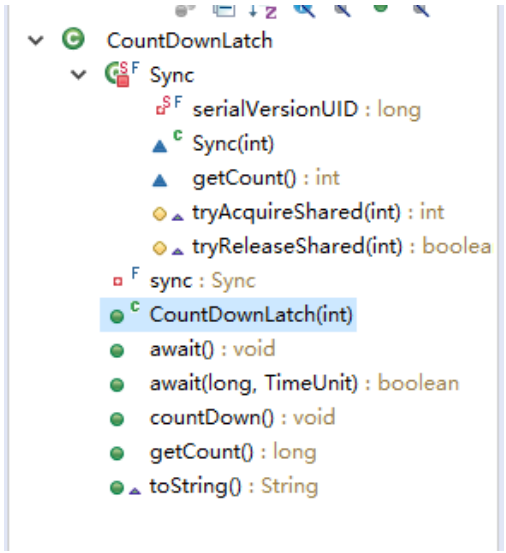
```
4 test thread3 count down is ok 5 mai
```


2

这里的CountDownLatch的构造函数中使用的int型变量 所以需要等到调用了两次countDown() 后主线程的await()方法才会返回。这意味着如果我们错误的估计了需要等待的操作的个数或者在 那么将意味着await()方法将永远的阻塞

实现原理

CountDownLatch类实际上是使用计数器的方式去控制 时候传入了一个int变量这个时候在类的 始化一个int的变量，每当我们调用countDown()方法的时候就使得这个变量的值减1，而对于await 表示所有的操作都已经完成，否则继续 实际上如果了解AQS的话应该很容易想到可以使用AQ 而CountDownLatch实际上也就是这



从结构上来看CountDownLatch的实现还是很简单的，通过很常见的继承AQS的方式来完成自己的同步器。

CountDownLatch的同步器实现：

```
private static final class Sync exte
    private static final long serial
        //初始化state
    Sync(int count) {
        setState(count);
    }

    int getCount() {
        return getState();
    }
        //尝试获取同步状态
        //只有当同步状态为0的时候返回
        //同步状态不为0则返回-1
    protected int tryAcquireShared(i
        return (getState() == 0) ? 1
    }
        //自旋+CAS的方式释放同步状态
    protected boolean tryReleaseShared(int releases) {
        // Decrement count; signal when transition to zero
        for (;;) {
            int c = getState();
            if (c == 0)
                return false;
            int nextc = c-1;
            if (compareAndSetState(c, nextc))
                return nextc == 0;
        }
    }
}
```


比较关键的地方是tryAquireShared()方法的实现，因为在父类的AQS中aquireShared()方法在调用tryAquireShared()方法的时候的判断依据是返回值是否大于零。

```
public final void acquireShared(int arg) {
    if (tryAcquireShared(arg) < 0)
                //失败则进入等待队列
```

```
        doAcquireShared(arg);
    }
```

同步器的实现相对都比较简单，主要思路和上面基本

CountDownLatch的主要方法（本身代码量就很少就

```
public class CountDownLatch {
    private static final class Sync
        private static final long se

        Sync(int count) {
            setState(count);
        }

        int getCount() {
            return getState();
        }

        protected int tryAcquireShared(int acquires) {
            return (getState() == 0) ? 1 : -1;
        }

        protected boolean tryReleaseShared(int releases) {
            // Decrement count; signal when transition to zero
            for (;;) {
                int c = getState();
                if (c == 0)
                    return false;
                int nextc = c-1;
                if (compareAndSetState(c, nextc))
                    return nextc == 0;
            }
```

```
        }
    }

    private final Sync sync;

        //初始化一个同步器
    public CountDownLatch(int count)
        if (count < 0) throw new Ill
        this.sync = new Sync(count);
    }
        //调用同步器的acquireSharedIn
        //并且是响应中断的
    public void await() throws Inter
        sync.acquireSharedInterrupti
    }

        //调用同步器的releaseShared方
    public void countDown() {
        sync.releaseShared(1);
    }
        //获取剩余的count
    public long getCount() {
        return sync.getCount();
    }

    public String toString() {
        return super.toString() + "[Count = " + sync.getCount() + "]";
    }
}
```


2

最后：由于CountDownLatch需要开发人员很明确需要等待的条件，否则很容易造成await()方法一直阻塞的情况。

2

想对作者说点什么

CountDownLatch理解一：与join的区别 1.6万

首先，我们来看一个应用场景1： 假设一条流水线上有 一个任务的完成需要他们三者协…

CountDownLatch的简单理解 1.4万

CountDownLatch的概念CountDownLatch是一个同步工具类，用来协调多个线程之间的同步，或者说起到线程之间的通信（而不是用…

并发编程工具之一：CountDownLatch 用法 3586

CountDownLatch 用法CountDownLatch是java.util.concurrent包中一个类，CountDownLatch只要提供的机制是多个（具体数量等于初…

java并发包里CountDownLatch的使用详解 541

A synchronization aid that allows one or more threads to wait until a set of operations being perfo...

你真的理解CountDownLatch与CyclicBarrier使用场景吗? 5421

相信每个想深入了解多线程开发的Java开发者都会遇到CountDownLatch和CyclicBarrier，大家也在网上看到各种介绍原理，代码的，…

CountDownLatch实现原理 1078

CountDownLatch用过很多次了，突然有点好奇，它是如何实现阻塞线程的，一开始还以为跟LockSupport有关。今天浏览了一下它的…

CountDownLatch使用 6786

分享牛原创，CountDownLatch类的使用，CountDow…时候，子线程还没有结束的时候…

并发技术_3_CountDownLatch 347

CountDownLatch Latch门栓，也就是有"门锁"的功能…也就是N个线程是不能继续往下…

【CountDownLatch】死循环检测模板/多 529

意义： CountDownLatch的一个作用是检测死循环，[…。 有的同学会用脚本循环执行…

CountDownLatch--控制三个线程执行顺 1307

前言线程中run方法调用CountDownLatch。CountDo…) 表示countObj计数减少。 count…

相关热词 javall 与java java的～ java java

2

个人资料

LightOfMiracle 关注

原创	粉丝	喜欢	评论
50	10	9	18

等级： 博客 3 访问： 3万+

积分： 877 排名： 6万+

勋章：

2

最新文章

常用Java开发工具类

Spring Boot 注解方式自定义Endpoint

Java还要再学一遍基础（十五）获取Unsafe

基于canal的Spring-Boot-Starter

Spring Boot整合oauth2.0搭建统一授权服务（密码模式）

归档

2018年6月	2篇
2018年5月	1篇
2018年3月	1篇
2018年1月	2篇
2017年7月	9篇

展开