线程池

原文地址：jenkov 作者： Jakob Jenkov 译者：长源 校对：方腾飞

线程池（Thread Pool）对于限制应用程序中同一时刻运行的线程数很有用。因为每启动一个新线程都会有相应的性能开销，每个线程都需要给栈分配一些内存等等。

我们可以把并发执行的任务传递给一个线程池，来替代为每个并发执行的任务都启动一个新的线程。只要池里有空闲的线程，任务就会分配给一个线程执行。在线程池的内部，任务被插入一个阻塞队列（Blocking Queue），线程池里的线程会去取这个队列里的任务。当一个新任务插入队列时，一个空闲线程就会成功的从队列中取出任务并且执行它。

线程池经常应用在多线程服务器上。每个通过网络到达服务器的连接都被包装成一个任务并且传递给线程池。线程池的线程会并发的处理连接上的请求。以后会再深入有关 Java 实现多线程服务器的细节。

Java 5 在 java.util.concurrent 包中自带了内置的线程池，所以你不用非得实现自己的线程池。你可以阅读我写的 java.util.concurrent.ExecutorService 的文章以了解更多有关内置线程池的知识。不过无论如何，知道一点关于线程池实现的知识总是有用的。

这里有一个简单的线程池实现：

```
01 public class ThreadPool {
02
03   private BlockingQueue taskQueue = null;
04   private List<PoolThread> threads = new ArrayList<PoolThread>();
05   private boolean isStopped = false;
06
07   public ThreadPool(int noOfThreads, int maxNoOfTasks) {
08     taskQueue = new BlockingQueue(maxNoOfTasks);
09
10     for (int i=0; i<noOfThreads; i++) {
11       threads.add(new PoolThread(taskQueue));
12     }
13     for (PoolThread thread : threads) {
14       thread.start();
15     }
16   }
17
18   public void synchronized execute(Runnable task) {
19     if(this.isStopped) throw
20       new IllegalStateException("ThreadPool is stopped");
21
22     this.taskQueue.enqueue(task);
23   }
24
25   public synchronized boolean stop() {
26     this.isStopped = true;
27     for (PoolThread thread : threads) {
```

```
      thread.stop();
    }
  }

}
```

(校注：原文有编译错误，我修改了下)

```
public class PoolThread extends Thread {

  private BlockingQueue<Runnable> taskQueue = null;
  private boolean       isStopped = false;

  public PoolThread(BlockingQueue<Runnable> queue) {
    taskQueue = queue;
  }

  public void run() {
    while (!isStopped()) {
      try {
        Runnable runnable =taskQueue.take();
        runnable.run();
      } catch(Exception e) {
        // 写日志或者报告异常,
        // 但保持线程池运行.
      }
    }
  }

  public synchronized void toStop() {
    isStopped = true;
    this.interrupt(); // 打断池中线程的 dequeue() 调用.
  }

  public synchronized boolean isStopped() {
    return isStopped;
  }
}
```

线程池的实现由两部分组成。类 ThreadPool 是线程池的公开接口，而类 PoolThread 用来实现执行任务的子线程。

为了执行一个任务，方法 ThreadPool.execute(Runnable r) 用 Runnable 的实现作为调用参数。在内部，Runnable 对象被放入 阻塞队列 (Blocking Queue)，等待着被子线程取出队列。

一个空闲的 PoolThread 线程会把 Runnable 对象从队列中取出并执行。你可以在 PoolThread.run() 方法里看到这些代码。执行完毕后，PoolThread 进入循环并且尝试从队列中再取出一个任务，直到线程终止。

调用 ThreadPool.stop() 方法可以停止 ThreadPool。在内部，调用 stop 先会标记 isStopped 成员变量（为 true）。然后，线程池的每一个子线程都调用 PoolThread.stop() 方法停止运行。注意，如果线程池的 execute() 在 stop() 之后调用，execute() 方法会抛出 IllegalStateException 异常。

子线程会在完成当前执行的任务后停止。注意 PoolThread.stop() 方法中调用了 this.interrupt()。它确保阻塞在 taskQueue.dequeue() 里的 wait() 调用的线程能够跳出 wait() 调用（校对注：因为执行了中断

interrupt，它能够打断这个调用），并且抛出一个 InterruptedException 异常离开 dequeue() 方法。这个异常在 PoolThread.run() 方法中被截获、报告，然后再检查 isStopped 变量。由于 isStopped 的值是 true，因此 PoolThread.run() 方法退出，子线程终止。

（校对注：看完觉得不过瘾？更详细的线程池文章参见Java线程池的分析和使用）

下一节 : Anatomy of a Synchronizer

宅男小何
2013/03/08 9:38上午

浅显易懂，支持！！

```
public void run(){
11 while(!isStopped()){
12 try{
13 Runnable runnable = (Runnable) taskQueue.dequeue();
14 runnable.run();
15 } catch(Exception e){
16 // 写日志或者报告异常,
17 // 但保持线程池运行.
18 }
19 }
20 }
```

这里面应该控制下频率最好，O(∩_∩)O~

sean
2013/05/06 11:48上午

1. ThreadPool类 的 boolean stop()，方法体里没有return。

2. ThreadPool类里用到的 BlockingQueue，是不是 http://ifeve.com/blocking-queues/ 里构造的那个？

如果是，那么 PoolThread类构造函数用的是 java.util.concurrent.BlockingQueue，报错。

如果不是，就意味着用的是 java.util.concurrent.BlockingQueue，那么ThreadPool类的第8行，new BlockingQueue(maxNoOfTasks); 是不成立的，BlockingQueue是interface。

长源_in355hz

2013/05/07 1:51下午

1. ThreadPool 类的 stop() 方法不需要中间 return 的, 因为必须等待所有子线程终止

2. ThreadPool 类用的 BlockQueue 是在另外一篇文章 http://tutorials.jenkov.com/java-concurrency/blocking-queues.html 里实现的那个，代码可以参考原文 http://tutorials.jenkov.com/java-concurrency/thread-pools.html ，在这篇文章里面的 import 是错误的，谢谢指正

kid
2013/07/02 10:06上午

1.return this.isStopped;

2.可以new ArrayBlockingQueue(maxNoOfTasks);

以上是我的观点

kid
2013/07/02 10:10上午

可以new ArrayBlockingQueue(maxNoOfTasks);

另外在ThreadPool中stop方法中调用的应该是thread.toStop()方法吧？

kid
2013/07/02 10:11上午

可以new ArrayBlockingQueue(maxNoOfTasks);

另外在ThreadPool中stop方法中调用的应该是thread.toStop()方法吧？

zen_me_yi
2014/01/11 5:41下午

taskQueue.take(); 是不是 Object dequeue()这个方法？ 求解释

daima_buyanzheng??
2014/01/11 5:50下午

代码ThreadPool 稍微有点小问题 额

Silence

2014/03/27 10:09下午

两个问题：

1、ThreadPool中的方法stop，其中的thread.stop()是不是应该改成调用PoolThread类中的toStop()方法。

2、“子线程会在完成当前执行的任务后停止。注意 PoolThread.stop() 方法中调用了 this.interrupt()。”一句中PoolThread.stop()的stop方法在PoolThread中是不存在的，应该是toStop()。

忘解答，谢谢！

zhang_lemon
2015/07/10 5:17下午

思路明白了，但是代码有许多问题。

1、BlockingQueue没有指定实现类

2、public void synchronized excute(Runnable task){} 应该将void 放置在synchronized后面

3、public synchronized boolean stop(){}声明了返回类型是boolean，但却没有return语句

这个代码楼主自己执行过吗？

BlackLee
2015/10/23 5:01下午

BlockingQueue在java.util.concurrent里有实现

mataszhang
2015/11/19 4:56下午

当阻塞队列满了，提交任务的方法execute()会阻塞。这也是个问题!

毕竟是示例代码，知道原理就行了

续写xyw
2016/04/19 6:31下午

感谢分享！

zhili
2018/06/12 4:37下午

以上提到的问题, 原文部分已经更改,

http://tutorials.jenkov.com/java-concurrency/thread-pools.html

前往此处查看.