Java 多线程编程之九：使用 Executors 和 ThreadPoolExecutor 实现的 Java 线程池的例子

2013年08月02日 12:39:39

线程池用来管理工作线程的数量，它持有一个等待被执行的线程的队列。

java.util.concurrent.Executors 提供了 java.util.concurrent.Executor 接口实现来创建 Java 里的线程池。我们写一个简单的程序来解释一下它的工作机制。

首先我们需要有一个 Runnable 类。

WorkerThread.java

```
1. package com.journaldev.threadpool;
2. 
3. public class WorkerThread implements Runnable {
4. 
5.     private String command;
6. 
7.     public WorkerThread(String s){
8.         this.command=s;
9.     }
10. 
11.     @Override
12.     public void run() {
13.         System.out.println(Thread.currentThread().getName()+" Start. 
Command = "+command);
14.         processCommand();
15.         System.out.println(Thread.currentThread().getName()+" End.");
16.     }
17. 
18.     private void processCommand() {
19.         try {
20.             Thread.sleep(5000);
21.         } catch (InterruptedException e) {
22.             e.printStackTrace();
```

```
        }
    }

    @Override
    public String toString(){
        return this.command;
    }
}
```

这里是我们使用 Executors 框架创建了一个固定的线程池的测试程序。

SimpleThreadPool.java

```
package com.journaldev.threadpool;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class SimpleThreadPool {

    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(5);
        for (int i = 0; i < 10; i++) {
            Runnable worker = new WorkerThread("" + i);
            executor.execute(worker);
        }
        executor.shutdown(); // This will make the executor accept no new threads and finish all existing threads in the queue
        while (!executor.isTerminated()) { // Wait until all threads are finish,and also you can use "executor.awaitTermination();" to wait
        }
        System.out.println("Finished all threads");
    }

}
```

程序中我们创建了固定大小为五个工作线程的线程池。然后分配给线程池十个工作，因为线程池大小为五，它将启动五个工作线程先处理五个工作，其他的工作则处于等待状态，一旦有工作完成，空闲下来工作线程就会捡取等待队列里的其他工作进行执行。

这里是以上程序的输出。

```
pool-1-thread-2 Start. Command = 1
pool-1-thread-4 Start. Command = 3
pool-1-thread-1 Start. Command = 0
pool-1-thread-3 Start. Command = 2
pool-1-thread-5 Start. Command = 4
pool-1-thread-4 End.
pool-1-thread-5 End.
pool-1-thread-1 End.
pool-1-thread-3 End.
pool-1-thread-3 Start. Command = 8
pool-1-thread-2 End.
pool-1-thread-2 Start. Command = 9
pool-1-thread-1 Start. Command = 7
pool-1-thread-5 Start. Command = 6
pool-1-thread-4 Start. Command = 5
pool-1-thread-2 End.
pool-1-thread-4 End.
pool-1-thread-3 End.
pool-1-thread-5 End.
pool-1-thread-1 End.
Finished all threads
```

输出表明线程池中至始至终只有五个名为 "pool-1-thread-1" 到 "pool-1-thread-5" 的五个线程，这五个线程不随着工作的完成而消亡，会一直存在，并负责执行分配给线程池的任务，直到线程池消亡。

Executors 类提供了使用了 ThreadPoolExecutor 的简单的 ExecutorService 实现，但是 ThreadPoolExecutor 提供的功能远不止于此。我们可以在创建 ThreadPoolExecutor 实例时指定活动线程的数量，我们也可以限制线程池的大小并且创建我们自己的 RejectedExecutionHandler 实现来处理不能适应工作队列的工作。

这里是我们自定义的 RejectedExecutionHandler 接口的实现。

RejectedExecutionHandlerImpl.java

```
1. package com.journaldev.threadpool;
2. 
3. import java.util.concurrent.RejectedExecutionHandler;
4. import java.util.concurrent.ThreadPoolExecutor;
5. 
6. public class RejectedExecutionHandlerImpl implements 
RejectedExecutionHandler {
7. 
8.     @Override
9.     public void rejectedExecution(Runnable r, ThreadPoolExecutor executor)
{
10.         System.out.println(r.toString() + " is rejected");
11.     }
12. 
13. }
```

ThreadPoolExecutor 提供了一些方法，我们可以使用这些方法来查询 executor 的当前状态，线程池大小，活动线程数量以及任务数量。因此我是用来一个监控线程在特定的时间间隔内打印 executor 信息。

MyMonitorThread.java

```
1. package com.journaldev.threadpool;
2. 
3. import java.util.concurrent.ThreadPoolExecutor;
4. 
5. public class MyMonitorThread implements Runnable
6. {
7.     private ThreadPoolExecutor executor;
8. 
9.     private int seconds;
10. 
11.     private boolean run=true;
12. 
13.     public MyMonitorThread(ThreadPoolExecutor executor, int delay)
```

```
    {
        this.executor = executor;
        this.seconds=delay;
    }

    public void shutdown(){
        this.run=false;
    }

    @Override
    public void run()
    {
        while(run){
                System.out.println(
                    String.format("[monitor] [%d/%d] Active: %d, Completed: %d, Task: %d, isShutdown: %s, isTerminated: %s",
                        this.executor.getPoolSize(),
                        this.executor.getCorePoolSize(),
                        this.executor.getActiveCount(),
                        this.executor.getCompletedTaskCount(),
                        this.executor.getTaskCount(),
                        this.executor.isShutdown(),
                        this.executor.isTerminated()));
                try {
                    Thread.sleep(seconds*1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
        }

    }
}
```

这里是使用 ThreadPoolExecutor 的线程池实现例子。

WorkerPool.java

```
1. package com.journaldev.threadpool;
2. 
3. import java.util.concurrent.ArrayBlockingQueue;
4. import java.util.concurrent.Executors;
5. import java.util.concurrent.ThreadFactory;
6. import java.util.concurrent.ThreadPoolExecutor;
7. import java.util.concurrent.TimeUnit;
8. 
9. public class WorkerPool {
10. 
11.     public static void main(String args[]) throws InterruptedException{
12.         //RejectedExecutionHandler implementation
13.         RejectedExecutionHandlerImpl rejectionHandler = new
RejectedExecutionHandlerImpl();
14.         //Get the ThreadFactory implementation to use
15.         ThreadFactory threadFactory = Executors.defaultThreadFactory();
16.         //creating the ThreadPoolExecutor
17.         ThreadPoolExecutor executorPool = new ThreadPoolExecutor(2, 4,
10, TimeUnit.SECONDS, new ArrayBlockingQueue<Runnable>(2),
threadFactory, rejectionHandler);
18.         //start the monitoring thread
19.         MyMonitorThread monitor = new MyMonitorThread(executorPool,
3);
20.         Thread monitorThread = new Thread(monitor);
21.         monitorThread.start();
22.         //submit work to the thread pool
23.         for(int i=0; i<10; i++){
24.             executorPool.execute(new WorkerThread("cmd"+i));
25.         }
26.         
27.         Thread.sleep(30000);
28.         //shut down the pool
29.         executorPool.shutdown();
```

```
30.         //shut down the monitor thread
31.         Thread.sleep(5000);
32.         monitor.shutdown();
33.         
34.     }
35. }
```

注意在初始化 ThreadPoolExecutor 时，我们保持初始池大小为 2，最大池大小为 4 而工作队列大小为 2。因此如果已经有四个正在执行的任务而此时分配来更多任务的话，工作队列将仅仅保留他们(新任务)中的两个，其他的将会被 RejectedExecutionHandlerImpl 处理。

上面程序的输出可以证实以上观点。

```
pool-1-thread-1 Start. Command = cmd0
pool-1-thread-4 Start. Command = cmd5
cmd6 is rejected
pool-1-thread-3 Start. Command = cmd4
pool-1-thread-2 Start. Command = cmd1
cmd7 is rejected
cmd8 is rejected
cmd9 is rejected
[monitor] [0/2] Active: 4, Completed: 0, Task: 6, isShutdown:  false,
isTerminated:  false
[monitor] [4/2] Active: 4, Completed: 0, Task: 6, isShutdown:  false,
isTerminated:  false
pool-1-thread-4 End.
pool-1-thread-1 End.
pool-1-thread-2 End.
pool-1-thread-3 End.
pool-1-thread-1 Start. Command = cmd3
pool-1-thread-4 Start. Command = cmd2
[monitor] [4/2] Active: 2, Completed: 4, Task: 6, isShutdown:  false,
isTerminated:  false
[monitor] [4/2] Active: 2, Completed: 4, Task: 6, isShutdown:  false,
isTerminated:  false
```

```
pool-1-thread-1 End.
pool-1-thread-4 End.
[monitor] [4/2] Active: 0, Completed: 6, Task: 6, isShutdown:  false,
isTerminated:  false
[monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown:  false,
isTerminated:  false
[monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown:  false,
isTerminated:  false
[monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown:  false,
isTerminated:  false
[monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown:  false,
isTerminated:  false
[monitor] [2/2] Active: 0, Completed: 6, Task: 6, isShutdown:  false,
isTerminated:  false
[monitor] [0/2] Active: 0, Completed: 6, Task: 6, isShutdown:  true,
isTerminated:  true
[monitor] [0/2] Active: 0, Completed: 6, Task: 6, isShutdown:  true,
isTerminated:  true
```

注意 executor 的活动任务、完成任务以及所有完成任务，这些数量上的变化。我们可以调用 shutdown() 方法来结束所有提交的任务并终止线程池。

原文链接：http://www.journaldev.com/1069/java-thread-pool-example-using-executors-and-threadpoolexecutor