自定义线程池

ThreadPool

```
import java.util.LinkedList;
import java.util.List;

/**
 * 线程池类，线程管理器：创建线程，执行任务，销毁线程，获取线程基本信息
 */
public final class ThreadPool {
    // 线程池中默认线程的个数为5
    private static int worker_num = 5;
    // 工作线程
    private WorkThread[] workThrads;
    // 未处理的任务
    private static volatile int finished_task = 0;
    // 任务队列，作为一个缓冲,List线程不安全
    private List<Runnable> taskQueue = new LinkedList<Runnable>();
    private static ThreadPool threadPool;

    // 创建具有默认线程个数的线程池
    private ThreadPool() {
        this(5);
    }

    // 创建线程池,worker_num为线程池中工作线程的个数
    private ThreadPool(int worker_num) {
        ThreadPool.worker_num = worker_num;
        workThrads = new WorkThread[worker_num];
        for (int i = 0; i < worker_num; i++) {
            workThrads[i] = new WorkThread();
            workThrads[i].start();// 开启线程池中的线程
        }
    }

    // 单态模式，获得一个默认线程个数的线程池
    public static ThreadPool getThreadPool() {
        return getThreadPool(ThreadPool.worker_num);
    }

    // 单态模式，获得一个指定线程个数的线程池,worker_num(>0)为线程池中工作线程的个数
    // worker_num<=0创建默认的工作线程个数
    public static ThreadPool getThreadPool(int worker_num1) {
        if (worker_num1 <= 0)
            worker_num1 = ThreadPool.worker_num;
        if (threadPool == null)
            threadPool = new ThreadPool(worker_num1);
        return threadPool;
    }

    // 执行任务,其实只是把任务加入任务队列，什么时候执行有线程池管理器觉定
    public void execute(Runnable task) {
```

```
        synchronized (taskQueue) {
            taskQueue.add(task);
            taskQueue.notify();
        }
    }

    // 批量执行任务,其实只是把任务加入任务队列，什么时候执行有线程池管理器觉定
    public void execute(Runnable[] task) {
        synchronized (taskQueue) {
            for (Runnable t : task)
                taskQueue.add(t);
            taskQueue.notify();
        }
    }

    // 批量执行任务,其实只是把任务加入任务队列，什么时候执行有线程池管理器觉定
    public void execute(List<Runnable> task) {
        synchronized (taskQueue) {
            for (Runnable t : task)
                taskQueue.add(t);
            taskQueue.notify();
        }
    }

    // 销毁线程池,该方法保证在所有任务都完成的情况下才销毁所有线程，否则等待任务完成才销毁
    public void destroy() {
        while (!taskQueue.isEmpty()) {// 如果还有任务没执行完成，就先睡会吧
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        // 工作线程停止工作，且置为null
        for (int i = 0; i < worker_num; i++) {
            workThrads[i].stopWorker();
            workThrads[i] = null;
        }
        threadPool=null;
        taskQueue.clear();// 清空任务队列
    }

    // 返回工作线程的个数
    public int getWorkThreadNumber() {
        return worker_num;
    }

    // 返回已完成任务的个数,这里的已完成是只出了任务队列的任务个数，可能该任务并没有实际执行完
成
    public int getFinishedTasknumber() {
        return finished_task;
    }

    // 返回任务队列的长度，即还没处理的任务个数
    public int getWaitTasknumber() {
        return taskQueue.size();
    }
```

```
    // 覆盖toString方法，返回线程池信息：工作线程个数和已完成任务个数
    @Override
    public String toString() {
        return "WorkThread number:" + worker_num + "  finished task number:"
                + finished_task + "  wait task number:" + getWaitTasknumber();
    }

    /**
     * 内部类，工作线程
     */
    private class WorkThread extends Thread {
        // 该工作线程是否有效，用于结束该工作线程
        private boolean isRunning = true;

        /*
         * 关键所在啊，如果任务队列不空，则取出任务执行，若任务队列空，则等待
         */
        @Override
        public void run() {
            Runnable r = null;
            while (isRunning) {// 注意，若线程无效则自然结束run方法，该线程就没用了
                synchronized (taskQueue) {
                    while (isRunning && taskQueue.isEmpty()) {// 队列为空
                        try {
                            taskQueue.wait(20);
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                    if (!taskQueue.isEmpty())
                        r = taskQueue.remove(0);// 取出任务
                }
                if (r != null) {
                    r.run();// 执行任务
                }
                finished_task++;
                r = null;
            }
        }

        // 停止工作，让该线程自然执行完run方法，自然结束
        public void stopWorker() {
            isRunning = false;
        }
    }
}
```

TestThreadPool

```
public class TestThreadPool {
    public static void main(String[] args) {
        // 创建3个线程的线程池
        ThreadPool t = ThreadPool.getThreadPool(3);
        t.execute(new Runnable[] { new Task(), new Task(), new Task() });
        t.execute(new Runnable[] { new Task(), new Task(), new Task() });
        System.out.println(t);
        t.destroy();// 所有线程都执行完成才destory
```

```
            System.out.println(t);
        }

        // 任务类
        static class Task implements Runnable {
            private static volatile int i = 1;

            @Override
            public void run() {// 执行任务
                System.out.println("任务 " + (i++) + " 完成");
            }
        }
}
```