考试指导：java多线程设计模式详解之三 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/180/2021_2022__E8_80_83_E8_AF_95_E6_8C_87_E5_c97_180995.htm 前面谈了多线程应用程序能极大地改善用户相应。例如对于一个Web应用程序，每当一个用户请求服务器连接时，服务器就可以启动一个新线程为用户服务。 然而，创建和销毁线程本身就有一定的开销，如果频繁创建和销毁线程，CPU和内存开销就不可忽略，垃圾收集器还必须负担更多的工作。因此，线程池就是为了避免频繁创建和销毁线程。 每当服务器接受了一个新的请求后，服务器就从线程池中挑选一个等待的线程并执行请求处理。处理完毕后，线程并不结束，而是转为阻塞状态再次被放入线程池中。这样就避免了频繁创建和销毁线程。 Worker Pattern实现了类似线程池的功能。首先定义Task接口：package com.crackj2ee.thread. public interface Task { void execute(). }线程将负责执行execute()方法。注意到任务是由子类通过实现execute()方法实现的，线程本身并不知道自己执行的任务。它只负责运行一个耗时的execute()方法。 具体任务由子类实现，我们定义了一个CalculateTask和一个TimerTask：// CalculateTask.java package com.crackj2ee.thread. public class CalculateTask implements Task { private static int count = 0. private int num = count. public CalculateTask() { count . } public void execute() { System.out.println("[CalculateTask " num "] start..."). try { Thread.sleep(3000). } catch(InterruptedException ie) {} System.out.println("[CalculateTask " num "] done."). } } //

TimerTask.java package com.crackj2ee.thread. public class TimerTask implements Task { private static int count = 0. private int num = count. public TimerTask() { count . } public void execute() { System.out.println("[TimerTask " num "] start..."). try { Thread.sleep(2000). } catch(InterruptedException ie) {} System.out.println("[TimerTask " num "] done."). } } 以上任务均简单的sleep若干秒。 TaskQueue实现了一个队列，客户端可以将请求放入队列，服务器线程可以从队列中取出任务： package com.crackj2ee.thread. import java.util.*. public class TaskQueue { private List queue = new LinkedList(). public synchronized Task getTask() { while(queue.size()==0) { try { this.wait(). } catch(InterruptedException ie) { return null. } } return (Task)queue.remove(0). } public synchronized void putTask(Task task) { queue.add(task). this.notifyAll(). } } 终于到了真正的WorkerThread，这是真正执行任务的服务器线程： package com.crackj2ee.thread. public class WorkerThread extends Thread { private static int count = 0. private boolean busy = false. private boolean stop = false. private TaskQueue queue. public WorkerThread(ThreadGroup group, TaskQueue queue) { super(group, "worker-" count). count . this.queue = queue. } public void shutdown() { stop = true. this.interrupt(). try { this.join(). } catch(InterruptedException ie) {} } public boolean isIdle() { return !busy. } public void run() { System.out.println(getName() " start."). while(!stop) { Task task = queue.getTask(). if(task!=null) { busy = true. task.execute(). busy = false. } } System.out.println(getName() " end."). } } 前面已经讲过，queue.getTask()是一个阻塞方法，服

务器线程可能在此wait()一段时间。此外，WorkerThread还有一个shutdown方法，用于安全结束线程。 最后是ThreadPool，负责管理所有的服务器线程，还可以动态增加和减少线程数： package com.crackj2ee.thread. import java.util.*. public class ThreadPool extends ThreadGroup { private List threads = new LinkedList(). private TaskQueue queue. public ThreadPool(TaskQueue queue) { super("Thread-Pool"). this.queue = queue. } public synchronized void addWorkerThread() { Thread t = new WorkerThread(this, queue). threads.add(t). t.start(). } public synchronized void removeWorkerThread() { if(threads.size()>0) { WorkerThread t = (WorkerThread)threads.remove(0). t.shutdown(). } } public synchronized void currentStatus() { System.out.println("----------------------------------------"). System.out.println("Thread count = " threads.size()). Iterator it = threads.iterator(). 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com