Java基础—创建Java程序中的线程池 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/145/2021_2022_Java_E5_9F_BA_E7_A1_80_c104_145626.htm 线程是Java的一大特性，它可以是给定的指令序列、给定的方法中定义的变量或者一些共享数据(类一级的变量)。在Java中每个线程有自己的堆栈和程序计数器（PC），其中堆栈是用来跟踪线程的上下文（上下文是当线程执行到某处时，当前的局部变量的值），而程序计数器则用来跟踪当前线程正在执行的指令。 在通常情况下，一个线程不能访问另外一个线程的堆栈变量，而且这个线程必须处于如下状态之一： 1.排队状态（Ready），在用户创建了一个线程以后，这个线程不会立即运行。当线程中的方法start()被调用时，这个线程就会进行排队状态，等待调度程序将它转入运行状态（Running）。当一个进程被执行后它也可以进行排队状态。如果调度程序允许的话，通过调用方法yield()就可以将进程放入排队状态。 2.运行状态(Running)，当调度程序将CPU的运行时间分配给一个线程，这个线程就进入了运行状态开始运行。 3.等待状态（Waiting），很多原因都可以导致线程处于等待状态，例如线程执行过程中被暂停，或者是等待I/O请求的完成而进入等待状态。 在Java中不同的线程具有不同的优先级，高优先级的线程可以安排在低优先级线程之前完成。如果多个线程具有相同的优先级，Java会在不同的线程之间切换运行。一个应用程序可以通过使用线程中的方法setPriority()来设置线程的优先级，使用方法getPriority()来获得一个线程的优先级。 线程的生命周期一个线程的的生命周期可以分成两阶段：生存（Alive）周期和

死亡（Dead）周期，其中生存周期又包括运行状态（Running）和等待状态（Waiting）。当创建一个新线程后，这个线程就进入了排队状态（Ready），当线程中的方法start()被调用时，线程就进入生存周期，这时它的方法isAlive()始终返回真值，直至线程进入死亡状态。线程的实现有两种方法可以实现线程，一种是扩展java.lang.Thread类，另一种是通过java.lang.Runnable接口。Thread类封装了线程的行为。要创建一个线程，必须创建一个从Thread类扩展出的新类。由于在Thread类中方法run()没有提供任何的操作，因此，在创建线程时用户必须覆盖方法run()来完成有用的工作。当线程中的方法start()被调用时，方法run()再被调用。下面的代码就是通过扩展Thread类来实现线程： import java.awt.*.class Sample1{public static void main(String[] args){Mythread test1=new Mythread(1).Mythread test2=new Mythread(2).test1.start().test2.start().}}class Mythread extends Thread {int id.Mythread(int i){ id=i.}public void run() {int i=0.while(id i==1){try {sleep(1000).} catch(InterruptedException e){}}System.out.println(“ The id is ” id).} 通常当用户希望一个类能运行在自己的线程中，同时也扩展其它某些类的特性时，就需要借助运行Runnable接口来实现。Runnable接口只有一个方法run()。不论什么时候创建了一个使用Runnable接口的类，都必须在类中编写run()方法来覆盖接口中的run()方法。例如下面的代码就是通过Runnable接口实现的线程： import java.awt.*.import java.applet.Applet.public class Bounce extends Applet implements Runnable{static int r=30.static int x=100.static int y=30.Thread t. public void init(){t = new

Thread(this).t.start().}public void run() {int y1= 1. int i=1.int sleeptime=10.while(true){y =(i*y). if(y-rgetSize().height) y1*=-1.try{t.sleep(sleeptime).}catch(InterruptedException e){ }}}}

为什么要使用线程池在Java中，如果每当一个请求到达就创建一个新线程，开销是相当大的。在实际使用中，每个请求创建新线程的服务器在创建和销毁线程上花费的时间和消耗的系统资源，甚至可能要比花在处理实际的用户请求的时间和资源要多得多。除了创建和销毁线程的开销之外，活动的线程也需要消耗系统资源。如果在一个JVM里创建太多的线程，可能会导致系统由于过度消耗内存或“ 切换过度 ”而导致系统资源不足。为了防止资源不足，服务器应用程序需要一些办法来限制任何给定时刻处理的请求数目，尽可能减少创建和销毁线程的次数，特别是一些资源耗费比较大的线程的创建和销毁，尽量利用已有对象来进行服务，这就是“ 池化资源 ”技术产生的原因。 线程池主要用来解决线程生命周期开销问题和资源不足问题。通过对多个任务重用线程，线程创建的开销就被分摊到了多个任务上了，而且由于在请求到达时线程已经存在，所以消除了线程创建所带来的延迟。这样，就可以立即为请求服务，使应用程序响应更快。另外，通过适当地调整线程池中的线程数目可以防止出现资源不足的情况。 创建一个线程池一个比较简单的线程池至少应包含线程池管理器、工作线程、任务队列、任务接口等部分。其中线程池管理器（ThreadPool Manager）的作用是创建、销毁并管理线程池，将工作线程放入线程池中；工作线程是一个可以循环执行任务的线程，在没有任务时进行等待；任务队列的作用是提供一种缓冲机制，将没有处理的任务放在任务

队列中；任务接口是每个任务必须实现的接口，主要用来规定任务的入口、任务执行完后的收尾工作、任务的执行状态等，工作线程通过该接口调度任务的执行。下面的代码实现了创建一个线程池，以及从线程池中取出线程的操作：