

超线程多核心下Java多线程编程技术分析 PDF转换可能丢失图片或格式，建议阅读原文

https://www.100test.com/kao_ti2020/144/2021_2022__E8_B6_85_E7_BA_BF_E7_A8_8B_E5_c104_144793.htm

一、Java环境下的多线程技术 构建线程化的应用程序往往会对程序带来重要的性能影响。例如，请考虑这样一个程序，它从磁盘读取大量数据并且在把它们写到屏幕之前处理这些数据(例如一个DVD播放器)。在一个传统的单线程程序(今天所使用的大多数客户端程序)上，一次只有一个任务执行，每一个这些活动分别作为一个序列的不同阶段发生。只有在一块已定义大小的数据读取完成时才能进行数据处理。因此，能处理数据的程序逻辑直到磁盘读操作完成后才得到执行。这将导致非常差的性能问题。在一个多线程程序中，可以分配一个线程来读取数据，让另一个线程来处理数据，而让第三个线程把数据输送到图形卡上去。这三个线程可以并行运行；这样以来，在磁盘读取数据的同时仍然可以处理数据，从而提高了整体程序的性能。许多大量的示例程序都可以被设计来同时做两件事情以进一步提高性能。Java虚拟机(JVM)本身就是基于此原因广泛使用了多线程技术。本文将讨论创建多线程Java代码以及一些进行并行程序设计的最好练习；另外还介绍了对开发者极为有用的一些工具和资源。篇幅所限，不可能全面论述这些问题，所以我想只是重点提一下极重要的地方并提供给你相应的参考信息。

二、线程化Java代码 所有的程序都至少使用一个线程。在C/C和Java中，这是指用对main()的调用而启动的那个线程。另外线程的创建需要若干步骤：创建一个新线程，然后指定给它某种工作。一旦工作做完，该线程将

自动被JVM所杀死。Java提供两个方法来创建线程并且指定给它们工作。第一种方法是子类化Java的Thread类(在java.lang包中)，然后用该线程的工作函数重载run()方法。下面是这种方法的一个示例：

```
public class SimpleThread extends Thread {
public SimpleThread(String str) { super(str). } public void run() { for
(int i = 0. i System.out.println(i " " getName()). try {
sleep((long)(Math.random() * 1000)). } catch
(InterruptedException e) {} } System.out.println("DONE! "
getName()). }}
```

这个类子类化Thread并且提供它自己的run()方法。上面代码中的函数运行一个循环来打印传送过来的字符串到屏幕上，然后等待一个随机的时间数目。在循环十次后，该函数打印"DONE!"，然后退出-并由它杀死这个线程。下面是创建线程的主函数：

```
public class TwoThreadsDemo { public
static void main (String[] args) { new SimpleThread("Do it!").start().
new SimpleThread("Definitely not!").start(). }}
```

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