Java程序多进程运行模式的实例分析 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/240/2021_2022_Java_E7_A8_8B_E5_BA_8F_c104_240567.htm 一般我们在java中运行其它类中的方法时，无论是静态调用，还是动态调用，都是在当前的进程中执行的，也就是说，只有一个java虚拟机实例在运行。而有的时候，我们需要通过java代码启动多个java子进程。这样做虽然占用了一些系统资源，但会使程序更加稳定，因为新启动的程序是在不同的虚拟机进程中运行的，如果有一个进程发生异常，并不影响其它的子进程。 在Java中我们可以使用两种方法来实现这种要求。最简单的方法就是通过Runtime中的exec方法执行java classname。如果执行成功，这个方法返回一个Process对象，如果执行失败，将抛出一个IOException错误。下面让我们来看一个简单的例子。 // Test1.java文件 import java.io.*. public class Test { public static void main(String[] args) { FileOutputStream fOut = new FileOutputStream("c:\Test1.txt"). fOut.close(). System.out.println("被调用成功!"). } } // Test_Exec.java public class Test_Exec { public static void main(String[] args) { Runtime run = Runtime.getRuntime(). Process p = run.exec("java test1"). } } 通过java Test_Exec运行程序后，发现在C盘多了个Test1.txt文件，但在控制台中并未出现"被调用成功！"的输出信息。因此可以断定，Test已经被执行成功，但因为某种原因，Test的输出信息未在Test_Exec的控制台中输出。这个原因也很简单，因为使用exec建立的是Test_Exec的子进程，这个子进程并没有自己的控制台，因此，它并不会输出任何信息。 如果要输

出子进程的输出信息，可以通过Process中的getInputStream得到子进程的输出流（在子进程中输出，在父进程中就是输入），然后将子进程中的输出流从父进程的控制台输出。具体的实现代码如下如示： // Test_Exec_Out.java import java.io.*. public class Test_Exec_Out { public static void main(String[] args) { Runtime run = Runtime.getRuntime(). Process p = run.exec("java test1"). BufferedInputStream in = new BufferedInputStream(p.getInputStream()). BufferedReader br = new BufferedReader(new InputStreamReader(in)). String s. while ((s = br.readLine()) != null) System.out.println(s). } } 从上面的代码可以看出，在Test_Exec_Out.java中通过按行读取子进程的输出信息，然后在Test_Exec_Out中按每行进行输出。 上面讨论的是如何得到子进程的输出信息。那么，除了输出信息，还有输入信息。既然子进程没有自己的控制台，那么输入信息也得由父进程提供。我们可以通过Process的getOutputStream方法来为子进程提供输入信息（即由父进程向子进程输入信息，而不是由控制台输入信息）。我们可以看看如下的代码： // Test2.java文件 import java.io.*. public class Test { public static void main(String[] args) { BufferedReader br = new BufferedReader(new InputStreamReader(System.in)). System.out.println("由父进程输入的信息：" br.readLine()). } } // Test_Exec_In.java import java.io.*. public class Test_Exec_In { public static void main(String[] args) { Runtime run = Runtime.getRuntime(). Process p = run.exec("java test2"). BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(p.getOutputStream())). bw.write("向子进程输出信息"). bw.flush(). bw.close(). // 必须得关闭流，否则无法

向子进程中输入信息 // System.in.read(). } } 从以上代码可以看出，Test1得到由Test_Exec_In发过来的信息，并将其输出。当你不加bw.flash()和bw.close()时，信息将无法到达子进程，也就是说子进程进入阻塞状态，但由于父进程已经退出了，因此，子进程也跟着退出了。如果要证明这一点，可以在最后加上System.in.read()，然后通过任务管理器（在windows下）查看java进程，你会发现如果加上bw.flush()和bw.close()，只有一个java进程存在，如果去掉它们，就有两个java进程存在。这是因为，如果将信息传给Test2，在得到信息后，Test2就退出了。在这里有一点需要说明一下，exec的执行是异步的，并不会因为执行的某个程序阻塞而停止执行下面的代码。因此，可以在运行test2后，仍可以执行下面的代码。 exec方法经过了多次的重载。上面使用的只是它的一种重载。它还可以将命令和参数分开，如exec("java.test2")可以写成exec("java", "test2")。exec还可以通过指定的环境变量运行不同配置的java虚拟机。 除了使用Runtime的exec方法建立子进程外，还可以通过ProcessBuilder建立子进程。ProcessBuilder的使用方法如下：// Test_Exec_Out.java import java.io.*. public class Test_Exec_Out { public static void main(String[] args) { ProcessBuilder pb = new ProcessBuilder("java", "test1"). Process p = pb.start(). … … } } 在建立子进程上，ProcessBuilder和Runtime类似，不同的ProcessBuilder使用start()方法启动子进程，而Runtime使用exec方法启动子进程。得到Process后，它们的操作就完全一样的。 ProcessBuilder和Runtime一样，也可设置可执行文件的环境信息、工作目录等。 ProcessBuilder pb = new ProcessBuilder("Command", "arg2", "arg2", ’ ’ ’ ). // 设置环

境变量 Map env = pb.environment(). env.put("key1", "value1"). env.remove("key2"). env.put("key2", env.get("key1") "_test"). pb.directory("..\abcd"). // 设置工作目录 Process p = pb.start(). // 建立子进程 