Java更多的库谜题82：啤酒爆炸 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/138/2021_2022_Java_E6_9B_B4_E5_A4_9A_c97_138717.htm 这一章的许多谜题都涉及到了多线程，而这个谜题涉及到了多进程。如果你用一行命令行带上参数slave去运行这个程序，它会打印什么呢？如果你使用的命令行不带任何参数，它又会打印什么呢？ public class BeerBlast{ static final String COMMAND = "java BeerBlast slave". public static void main(String[] args) throws Exception{ if(args.length == 1 amp. args[0].equals("slave")) { for(int i = 99. i > 0. i--){ System.out.println(i " bottles of beer on the wall"). System.out.println(i " bottles of beer"). System.out.println("You take on down, pass it around,"). System.out.println((i-1) " bottles of beer on the wall"). System.out.println(). } }else{ // Master Process process = Runtime.getRuntime().exec(COMMAND). int exitValue = process.waitFor(). System.out.println("exit value = " exitValue). } }}如果你使用参数slave来运行该程序，它就会打印出那首激动人心的名为 ” 99 Bottles of Beer on the Wall ” 的童谣的歌词，这没有什么神秘的。如果你不使用该参数来运行这个程序，它会启动一个slave进程来打印这首歌谣，但是你看不到slave进程的输出。主进程会等待slave进程结束，然后打印出slave进程的退出值(exit value)。根据惯例，0值表示正常结束，所以0就是你可能期望该程序打印的东西。如果你运行了程序，你可能会发现该程序只会悬挂在那里，不会打印任何东西，看起来slave进程好像永远都在运行着。所以你可能会觉得你应该一直都能听到 ” 99 Bottles of Beer on the Wall ” 这首童谣，即

使是这首歌被唱走调了也是如此，但是这首歌只有99句，而且，电脑是很快的，你假设的情况应该是不存在的，那么这个程序出了什么问题呢？ 这个秘密的线索可以在Process类的文档中找到，它叙述道：“ 由于某些本地平台只提供有限大小的缓冲，所以如果未能迅速地读取子进程(subprocess)的输出流，就有可能会导致子进程的阻塞，甚至是死锁 ” [Java-API]。这恰好就是这里所发生的事情：没有足够的缓冲空间来保存这首冗长的歌谣。为了确保slave进程能够结束，父进程必须排空(drain)它的输出流，而这个输出流从master线程的角度来看是输入流。下面的这个工具方法会在后台线程中完成这项工作： static void drainInBackground(final InputStream is) { new Thread(new Runnable(){ public void run(){ try{ while(is.read() >= 0). } catch(IOException e){ // return on IOException } } }).start(). }如果我们修改原有的程序，在等待slave进程之前调用这个方法，程序就会打印出0： }else{ // Master Process process = Runtime.getRuntime().exec(COMMAND). drainInBackground(process.getInputStream()). int exitValue = process.waitFor(). System.out.println("exit value = " exitValue). }这里的教训是：为了确保子进程能够结束，你必须排空它的输出流；对于错误流（error stream）也是一样，而且它可能会更麻烦，因为你无法预测进程什么时候会倾倒（dump）一些输出到这个流中。在5.0版本中，加入了一个名为ProcessBuilder的类用于排空这些流。它的redirectErrorStream方法将各个流合并起来，所以你只需要排空这一个流。如果你决定不合并输出流和错误流，你必须并行地（concurrently

）排空它们。试图顺序化地（sequentially）排空它们会导致子进程被挂起。 多年以来，很多程序员都被这个缺陷所刺痛。这里对于API设计者们的教训是，Process类应该避免这个错误，也许应该自动地排空输出流和错误流，除非用户表示要读取它们。更一般的讲，API应该设计得更容易做出正确的事，而很难或不可能做出错误的事。