像进行缩放操作，确保输入图像符合模型的预期输入尺寸（768x1024）。人物图像经过 OpenPose 模型处理后，生成关键点数据，并且通过 Parsing 模型获取分割信息。经过上述操作，生成可以与衣物图像合成的准确人像轮廓，从而为虚拟试衣过程提供更精细的控制。get_mask_location() 函数用于生成遮罩图，用于定义人物区域和背景区域，方便后续的合成工作。最终，这些预处理结果会输入到虚拟试衣模型（如 OOTDiffusionHD 或 OOTDiffusionDC）中进行生成。

模型调用部分，根据用户选择、上传的试衣类别，选择对应的虚拟试衣模型进行推理。在运行时，代码会根据传入的参数配置（如类别、步数等），自动选择适当的模型类型（hd 或 dc），并调用相应的模型。模型根据输入的衣物图像、人物图像和遮罩等信息，生成合成后的试衣效果图。模型执行过程中，图像被分批次处理，并且生成的图像保存在指定的输出目录中。

后处理逻辑方面，生成的图像会经过简单的文件保存操作，并且可以进一步优化图像的显示效果。图片会以 PNG 格式保存在 images_output 目录下，之后返还给前端。

3.4 运行效果

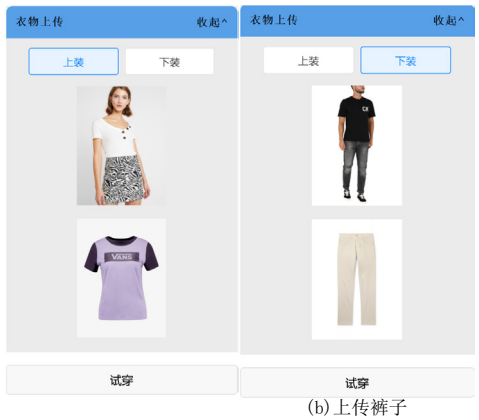
本系统所有的测试工作均在普通安卓手机环境下进行。测试设备为一款中低端安卓智能手机，系统版本为 Android 11，以确保系统的普遍适用性和稳定性。测试主要围绕功能性测试和性能测试两个方面展开。

功能测试主要是测试系统的上传图片、结果生成、下载结果及分享功能。软件界面如图 1 所示。人物及衣服图片的上传功能如图 1（a）（b）所示，当用户上传人物图片、T 恤衫图片和裤子图片后后，后端服务器成功接收到图片。上传图片并点击生成按钮后，后端服务器调用 OOTDiffusion 模型生成多个虚拟试衣结果，前端以轮播图的形式展示出来，如图 1（c）（d）所示。用户点击下载按钮，即可下载对应的试衣结果；点击分享按钮，即可将结果分享至社交平台。

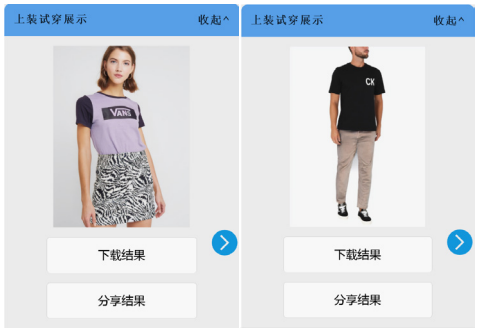
性能测试则集中在端到端延迟和内存占用两个重要指标上。对于端到端延迟，经过多次测试，生成过程的平均时间控制在 30 秒以内，符合要求。这意味着从用户上传图片到最终生成试衣效果图的全过程能够在 30 秒以内完成，保证了系统的高效性。在内存占用方面，测试结果表明，Android 端的系统内存占用大约为 200MB 左右，属于正常范围内，不会导致手机出现卡顿或过度的资源消耗。

4 结语

本文所设计系统能够根据用户上传的衣物图像和人物图像，利用不同的虚拟试衣模型（如 OOTDiffusionHD 和 OOTDiffusionDC）进行图像合成，生成与真实试穿效果相近的试衣效果图。然而，系统也存在一定的局限性。首先，当前的生成效果主要依赖于现有的模型架构，在处理某些复杂服装类型或特定体型时，仍可能出现不自然的配合或细节丢失。例如，针对不同体型的服装合成效果，模型的适应性和准确性仍然有所限制。其次，尽管本系统在大多数情况下能够生成较为真实的效果图，但面对实际的多样化试穿场景（如光照变化、服装纹理复杂度等），仍可能产生一些不理想的合成图像。



(b) 上传裤子



(d) 裤子试穿

图 1 软件界面

参考文献

[1] 李照璇.新零售场景营销对消费者行为的影响研究[D].湖南大学,2023.

[2] 仝子言,和羽萱,王晶,等.虚拟试衣在中国服装行业的应用及商业模式创新研究[J].现代商业,2023,(18):15-18.

[3] 曲欣,刘春艳,齐贝宁,等.AR虚拟试衣技术发展现状及推广前景展望[J].内江科技,2023,44(01):87-88.

[4] 林凤秒,勇金华,陈艳.基于VR技术的虚拟试衣间设计[J].电视技术,2022,46(09):219-221+228.

[5] 亢少军.Flutter技术入门与实战[M].机械工业出版社:202006.1257.

[6] 向治洪.React Native移动开发实战[M].人民邮电出版社:202311.487.

[7] 滕毅,马焯文.基于Uniapp的校园拼车多端小程序设计与实现[J].电子制作,2022,30(16):43-46.

[8] 杨洪涛.Flask中ORM模型的应用及研究[J].电脑编程技巧与维护,2023,(10):49-51+75.

[9] Xu Yuhao, Gu Tao, Chen Weifeng, and Chen Chengcai. OOTDiffusion: Outfitting Fusion based Latent Diffusion for Controllable Virtual Try-on[J]. arXiv preprint arXiv:2403.01779v2, 2024.

 DOI: https://doi.org/10.12345/bdai.v6i6.34071

Research on Buffer Congestion When JAVA Programs Invoke External Commands

Huiyong Yu

China Electronic Technology Group Corporation twenty-seventh Research Institute, ZhengZhou, Henan, 450047, China

Abstract

During the process of writing programs using the JAVA programming language, there are specific business requirements where JAVA programs need to invoke external commands through the operating system's SHELL to accomplish certain functions, such as launching JAVA programs to call the operating system's PING command or ORACLE's EXPDP command. This functionality is a commonly used basic feature, but if the command content is excessively long during the invocation, it may lead to JAVA buffer congestion, resulting in abnormal JAVA program execution. This article provides a detailed description of the invocation process and abnormal phenomena, conducts in-depth analysis of the issues through practical testing, and offers specific solutions.

Keyword

JAVA; Call; External command; Buffer; Congestion

JAVA 程序调用外部命令时缓冲区拥塞的研究

于慧勇

中国电子科技集团公司第二十七研究所，中国·河南 郑州 450047

摘　要

在使用JAVA编程语言编写程序的过程中，存在一些特定的业务需求，JAVA程序需要通过操作系统的SHELL调用外部命令来完成特定的功能，例如启动JAVA程序调用操作系统的PING命令、ORACLE的EXPDP命令等。该功能是一项常用的基础功能，在调用过程中如果命令内容过长，则可能会产生JAVA缓冲区拥塞的问题，造成JAVA程序运行异常。本文通过对调用过程、异常现象进行详细描述，通过实际测试的方式对问题进行深入分析，并给出具体的解决办法。

关键词

JAVA；调用；外部命令；缓冲区；拥塞

1 引言

JAVA 编程语言由于其纯粹的面向对象、语法结构简单易学、平台独立性、语言级支持并发、安全等特性，在 IT 领域中的应用越来越广泛。JAVA 面向对象程序设计研究集中在对其语言规则的进一步简化和对其相关技术的继续优化方面，为计算机软件开发提供更加优质的编程语言 [1]。

JAVA 编程语言其基础类库丰富，与外部系统接口全面，是程序设计过程中经常用到的功能，外部命令或程序的调用丰富了 JAVA 程序的功能，使 JAVA 程序有更好的扩展性。JAVA 程序运行依赖 JAVA 虚拟机，其负责管理功能之一是负责 JAVA 程序的输入、输出流，JAVA 虚拟机默认的输出流的缓冲区容量有限，当外部程序输出信息过多时，就会造成输出流的缓冲区的拥塞，出现程序异常，造成线程不能正常终止，继而造成 JAVA 主程序的逻辑不能顺序执行。在工程应用的过程中，发生此类问题后外部命令的异常信息无法正常传递回 JAVA 程序，不便异常的处理。

Java 的缓冲区拥塞通常出现在如下场景：当生产者线程向缓冲区写入数据的速度持续高于消费者线程的读取速度，缓冲区就会逐渐被填满，最终导致后续写操作因无空间而抛出 BufferOverflowException 。这种拥塞本质上是生产者 - 消费者速率不匹配造成的资源瓶颈。

为了避免拥塞，充分利用缓冲区空间，保证数据平稳流动，通常可以采取以下措施：

合理设置缓冲区容量：根据业务峰值流量预留足够空间，减少频繁扩容。

流量控制：利用滑动窗口协议或信号量机制，动态限制生产者发送速率，确保不超过消费者处理能力。

及时翻转与清理：写完一批数据后调用 flip() 切换到读模式，读完后调用 clear() 或 compact() 释放空间，避免缓冲区“假满”。

线程同步：在多线程环境下，使用锁或 wait()/notify() 机制协调生产者与消费者，防止并发写入导致的溢出。

【作者简介】于慧勇（1980–），男，中国吉林长岭人，本科，高级工程师，从事计算机软件开发、网络安全研究。

本文将采用多线结合读出缓冲区内容的方式进行拥塞处理，在进行缓存清理的同时也能收集到缓冲区信息，更好的获取调用外部命令时的处理结果。

2 调用过程分析

JAVA 程序执行外部命令主要依赖 ProcessBulider.start() 和 Runtime.exec() 方法来实现，并返回 Process 子类的是个实例，该实例可用来控制进程状态并获取相关信息，Process 类提供了执行从进程输入、执行输出到进程、等待进程完成、检查进程的退出状态已经销毁进程的方法。

它的所有标准 IO（既 stdin、stdout 和 stderr）操作都将通过三个流：getOutputStream()、getInputStream() 和 getErrorStream() 方法重定向到父进程，父进程使用这些流来提供到子进程的输入和获得从子进程的输出。在使用过程中请注意，ProcessBulider.start() 和 Runtime.exec() 方法传递的参数有所不同，Runtime. exec() 可接受一个单独的字符串，这个字符串是通过空格来分割可执行命令和参数的，而 ProcessBulider.start() 的构造函数是一个字符串列表或者数组，列表中第一个参数是可执行名，其他的是命令执行时需要的参数，以调用 Windows 系统命令“ipconfig -all”为例，使用过程区别如下：

Process p = Runtime.getRuntime().exec(“ipconfig -all”);

Process p = new ProcessBuilder(“ipcnofig”,”/all”).start();

通过 JDK 开发文档可知，Runtime.exec() 在执行过程中最终还是通过调用 ProcessBulider.start() 方法来执行最终的操作的，本文为便于 JAVA 程序编写、演示，所有代码均使用 Runtime.exec() 运行 Oracle 的 EXPDP（Export Data Pump）命令为例进行测试、分析和处理，expdp 是 Oracle 10g 及以后版本提供的高性能数据导出工具。它采用并行多线程架构，可把整个数据库、指定表空间、用户模式或单张表导出成二进制文件，同时支持压缩、加密和网络传输。相比传统 exp 工具，expdp 速度更快、容错更强，还能通过 QUERY、INCLUDE/EXCLUDE 等参数灵活筛选对象与数据，典型用途包括系统升级、数据迁移、灾备备份和测试环境搭建。[2] 基本 JAVA 程序代码如下：

```
1  package test;
2
3* import java.io.BufferedReader;
8
9  public class JavaExeC2 {
10     public static void main(String[] args) throws InterruptedException {
11         String FileName = "001";
12         Process p = null;
13         try {
14             // 执行命令
15             p = Runtime.getRuntime().exec(
16             "expdp TEST1212/TEST1212@orcle TABLES=EDITSTYLE, directory=BF "
17             + "dumpfile="+FileName+".dmp logfile="+FileName+".log");
18             //如果进程结束，由控制台输出备份的日志文件
19             if (p.waitFor() == 0) {
20                 BufferedReader bufRead = new BufferedReader(
21                         new InputStreamReader(new FileInputStream(
22                                 "E:\\oracle\\BF\\"+FileName+".log")));
23                 String str;
24                 while ((str = bufRead.readLine()) != null) {
25                     System.out.println(str);
26                 }
27                 bufRead.close();
28             }
29         } catch (IOException e) {
30             e.printStackTrace();
31         }
32     }
33 }
```

图 1 Runtime.exec() 程序代码

以上 JAVA 程序代码可正常运行，成功完成了外部命令“EXPDP”的全部操作，运行结果如下：

图 2 Runtime.exec() 程序运行结果

当将上面 JAVA 程序代码第 16 行中增加要备份的数据表，使 EXPDP 命令后字符较长时，JAVA 运行程序代码出现异常，该 JAVA 程序无响应，说明 JAVA 程序在调用外部命令是已经发生缓冲区拥塞的情况，程序不能完成外部命令“EXPDP”的操作。

3 程序异常的分析

在问题分析过程中，IED 工具的控制台并没有报错信息的输出，通过 DEBUG 工具也仅能确定出现 BUG 的代码行位置，仅可以初步判断程序异常发生在调用外部命令部分，不能直接判断程序出错的根本原因。出现这种情况，通过使用常规的工具的手段进行问题排查变得很困难，那么我们通过综合分析法进行分析：对正常运行的程序和出错的程序进行对比，如下表 1：

表 1 程序比较表

序号	比较项目	正常程序	出错程序
1	运行环境	相同	相同
2	调用方式	相同	相同
3	执行命令类型	相同	相同
4	执行命令内容	字符串较短	字符串较长
5	命令在外部执行情况	正常执行	正常执行
6	其它差异	无	无

通过综合比较发现，两个 JAVA 程序仅在执行外部命令的字符串长短存在差异，我们知道 JAVA 程序在调用外部程序时的与外部程序的数据交互是通过字节流缓冲区完成的，缓冲区可以简单的理解为一段内存区域，一段特殊的内存区域，在 JAVA 程序中除了对文件或数据库的操作会使用到缓冲区外，在字符流的操作中，所有的字符都是在内存中形成的，在输出前会将所有的内容暂时保存在内存中，也就是使用了缓冲区暂存数据，如果想在不关闭程序时也可以将字符流的内容全部输出，则可以使用 Write 类中的 flush() 方法完成。

所以我们从字节流缓冲区方面着手，下面对字节流缓冲区进行分析和研究，达到解决调用外部程序时发生异常的问题。通过分析输出流的缓冲区（大小）和字节流缓冲区，JAVA 程序代码中使用 InputStreamOutputStream(顶层父类) 调用外部命令时，由于没有对缓冲区进行处理，形成缓冲区