[Java进程间通信学习](https://www.cnblogs.com/fnlingnzb-learner/p/11664000.html)

转自：<https://www.iteye.com/blog/polim-1278435>

**进程间通信的主要方法有：**  
（1）管道（Pipe）：管道可用于具有亲缘关系进程间的通信，允许一个进程和另一个与它有共同祖先的进程之间进行通信。  
（2）命名管道（named pipe）：命名管道克服了管道没有名字的限制，因此，除具有管道所具有的功能外，它还允许无亲缘关系进程间的通信。命名管道在文件系统中有对应的文件名。命名管道通过命令mkfifo或系统调用mkfifo来创建。  
（3）信号（Signal）：信号是比较复杂的通信方式，用于通知接受进程有某种事件发生，除了用于进程间通信外，进程还可以发送信号给进程本身；linux除了支持Unix早期信号语义函数sigal外，还支持语义符合Posix.1标准的信号函数sigaction（实际上，该函数是基于BSD的，BSD为了实现可靠信号机制，又能够统一对外接口，用sigaction函数重新实现了signal函数）。Linux中可以使用kill -12 进程号，像当前进程发送信号，但前提是发送信号的进程要注册该信号。  
example:  
OperateSignal operateSignalHandler = new OperateSignal();  
Signal sig = new Signal("USR2");  
Signal.handle(sig, operateSignalHandler);  
（4）消息（Message）队列：消息队列是消息的链接表，包括Posix消息队列system V消息队列。有足够权限的进程可以向队列中添加消息，被赋予读权限的进程则可以读走队列中的消息。消息队列克服了信号承载信息量少，管道只能承载无格式字节流以及缓冲区大小受限等缺限。  
（5）共享内存：使得多个进程可以访问同一块内存空间，是最快的可用IPC形式。是针对其他通信机制运行效率较低而设计的。往往与其它通信机制，如信号量结合使用，来达到进程间的同步及互斥。  
（6）内存映射（mapped memory）：内存映射允许任何多个进程间通信，每一个使用该机制的进程通过把一个共享的文件映射到自己的进程地址空间来实现它。  
Java 中有类 MappedByteBuffer实现内存映射  
（7）信号量（semaphore）：主要作为进程间以及同一进程不同线程之间的同步手段。  
（8）套接口（Socket）：更为一般的进程间通信机制，可用于不同机器之间的进程间通信。起初是由Unix系统的BSD分支开发出来的，但现在一般可以移植到其它类Unix系统上：Linux和System V的变种都支持套接字。  
  
**管道方式**  
一、Java 启动子进程方式

[复制代码](javascript:void(0);)

1 1、

2 Runtime rt = Runtime.getRuntime();

3 Process process = rt.exec("java com.test.process.T3");

4 2、

5 ProcessBuilder pb = new ProcessBuilder("java", "com.test.process.T3");

6 Process p = pb.start();

[复制代码](javascript:void(0);)

二、Java父、子进程通信方式(管道方式)  
父进程获取子进程输出流方式

[复制代码](javascript:void(0);)

1 BufferedInputStream in = new BufferedInputStream(p.getInputStream());

2 BufferedReader br = new BufferedReader(new InputStreamReader(in));

3 String s;

4 while ((s = br.readLine()) != null) {

5 System.out.println(s);

6 }

[复制代码](javascript:void(0);)

子进程获取父进程输入流方式

[复制代码](javascript:void(0);)

1 package com.test.process;

2 import java.io.BufferedReader;

3 import java.io.IOException;

4 import java.io.InputStreamReader;

5

6 public class T3 {

7

8 public static void main(String[] args) throws IOException {

9 System.out.println("子进程被调用成功!");

10

11 BufferedReader bfr = new BufferedReader(new InputStreamReader(System.in));

12

13 while (true) {

14 String strLine = bfr.readLine();

15 if (strLine != null) {

16 System.out.println("hi:" + strLine);

17 }

18 }

19 }

20

21 }

[复制代码](javascript:void(0);)

三、详细测试类  
父进程测试类:

[复制代码](javascript:void(0);)

1 package com.test.process.pipe;

2 import java.io.IOException;

3

4 public class ProcessTest {

5

6 public static void main(String[] args) throws IOException, InterruptedException {

7 Process p = Runtime.getRuntime().exec("java com.test.process.pipe.MyTest");

8

9 StringBuilder sbuilder = new StringBuilder();

10 for(int k=0;k<1;k++){

11 sbuilder.append("hello");

12 }

13

14 int outSize = 1;

15 TestOut out[] = new TestOut[outSize];

16 for(int i=0;i<outSize;i++){

17 out[i] = new TestOut(p,sbuilder.toString().getBytes());

18 new Thread(out[i]).start();

19 }

20

21 int inSize = 1;

22 TestIn in[] = new TestIn[inSize];

23 for(int j=0;j<inSize;j++){

24 in[j] = new TestIn(p);

25 new Thread(in[j]).start();

26 }

27 }

28 }

[复制代码](javascript:void(0);)

子进程类

[复制代码](javascript:void(0);)

1 package com.test.process.pipe;

2 import java.io.BufferedReader;

3 import java.io.InputStreamReader;

4

5 public class MyTest {

6 public static void main(String[] args) throws Exception {

7 //读取父进程输入流

8 BufferedReader bfr = new BufferedReader(new InputStreamReader(System.in));

9 while (true) {

10 String strLine = bfr.readLine();

11 if (strLine != null) {

12 System.out.println(strLine);//这个地方的输出在子进程控制台是无法输出的，只可以在父进程获取子进程的输出

13 }else {

14 return;

15 }

16 }

17 }

18 }

[复制代码](javascript:void(0);)

TestIn类

[复制代码](javascript:void(0);)

1 package com.test.process.pipe;

2 import java.io.BufferedReader;

3 import java.io.InputStream;

4 import java.io.InputStreamReader;

5

6 public class TestIn implements Runnable{

7

8 private Process p = null;

9 public TestIn(Process process){

10 p = process;

11 }

12

13 @Override

14 public void run() {

15 try {

16 InputStream in = p.getInputStream();

17 BufferedReader bfr = new BufferedReader(new InputStreamReader(in));

18 String rd = bfr.readLine();

19 if(rd != null){

20 System.out.println(rd);//输出子进程返回信息(即子进程中的System.out.println()内容)

21 }else{

22 return;

23 }

24 //注意这个地方，如果关闭流则子进程的返回信息无法获取，如果不关闭只有当子进程返回字节为8192时才返回，为什么是8192下面说明.

25 //bfr.close();

26 //in.close();

27 } catch (Exception e) {

28 e.printStackTrace();

29 }

30 }

31 }

[复制代码](javascript:void(0);)

TestOut类

[复制代码](javascript:void(0);)

1 package com.test.process.pipe;

2 import java.io.IOException;

3 import java.io.OutputStream;

4

5 public class TestOut implements Runnable {

6

7 private Process p = null;

8 private byte []b = null;

9

10 public TestOut(Process process,byte byt[]){

11 p = process;

12 b = byt;

13 }

14

15 @Override

16 public void run() {

17 try {

18 OutputStream ops = p.getOutputStream();

19 //System.out.println("out--"+b.length);

20 ops.write(b);

21 //注意这个地方如果关闭，则父进程只可以给子进程发送一次信息，如果这个地方开启close()则父进程给子进程不管发送大小多大的数据，子进程都可以返回

22 //如果这个地方close()不开启，则父进程给子进程发送数据累加到8192子进程才返回。

23 //ops.close();

24 } catch (IOException e) {

25 e.printStackTrace();

26 }

27 }

28 }

[复制代码](javascript:void(0);)

备注：  
1、子进程的输出内容是无法在控制台输出的，只能再父类中获取并输出。  
2、父进程往子进程写内容时如果关闭字节流，则子进程的输入流同时关闭。  
3、如果父进程中输入、输出流都不关闭，子进程获取的字节流在达到8129byte时才返回。  
4、关闭子进程一定要在父进程中关闭 p.destroy()  
  
实例1：

[复制代码](javascript:void(0);)

1 /\*\*

2 \*如下另一种情况说明

3 \*如果像如下情况执行会出现说明情况呢

4 \*前提说明：TestOut类中开启ops.close();

5 \*/

6 package com.test.process.pipe;

7 import java.io.IOException;

8

9 public class ProcessTest {

10

11 public static void main(String[] args) throws IOException, InterruptedException {

12 Process p = Runtime.getRuntime().exec("java com.test.process.pipe.MyTest");

13

14 TestOut out = new TestOut(p,"Hello everyone".getBytes());

15 new Thread(out).start();

16

17 TestIn ti = new TestIn(p);

18 new Thread(ti).start();

19

20 Thread.sleep(3000);

21

22 TestOut out2 = new TestOut(p,"-Hello-everyone".getBytes());

23 new Thread(out2).start();

24

25 TestIn ti2 = new TestIn(p);

26 new Thread(ti2).start();

27 }

28 }

[复制代码](javascript:void(0);)

执行后输出结果为:

[复制代码](javascript:void(0);)

Hello everyone

java.io.IOException: Stream closed

at java.io.BufferedInputStream.getBufIfOpen(BufferedInputStream.java:145

)

at java.io.BufferedInputStream.read(BufferedInputStream.java:308)

at sun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:264)

at sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:306)

at sun.nio.cs.StreamDecoder.read(StreamDecoder.java:158)

at java.io.InputStreamReader.read(InputStreamReader.java:167)

at java.io.BufferedReader.fill(BufferedReader.java:136)

at java.io.BufferedReader.readLine(BufferedReader.java:299)

at java.io.BufferedReader.readLine(BufferedReader.java:362)

at com.test.process.pipe.TestIn.run(TestIn.java:20)

at java.lang.Thread.run(Thread.java:662)

[复制代码](javascript:void(0);)

由此可见当创建一个子进程后，p.getOutputStream();p.getInputStream();通过两种方式使父进程与子进程建立管道连接，而当close()连接时管道关闭，在通过调用  
p.getOutputStream();p.getInputStream();时直接出现IOException，结论为当父子进程建立连接后，通过管道长连接的方式进程信息传输，当close时在通过获取子进程的输入输出流  
都会出现IOException  
  
实例2：  
在实例1的基础上进行修改，会出现什么结果呢，如下

[复制代码](javascript:void(0);)

1 package com.test.process.pipe;

2 import java.io.IOException;

3

4 public class ProcessTest {

5

6 public static void main(String[] args) throws IOException, InterruptedException {

7 Process p = Runtime.getRuntime().exec("java com.test.process.pipe.MyTest");

8

9 TestOut out = new TestOut(p,"Hello everyone".getBytes());

10 new Thread(out).start();

11

12 TestIn ti = new TestIn(p);

13 new Thread(ti).start();

14

15 Process p2 = Runtime.getRuntime().exec("java com.test.process.pipe.MyTest");

16 TestOut out2 = new TestOut(p2,"-Hello-everyone".getBytes());

17 new Thread(out2).start();

18

19 TestIn ti2 = new TestIn(p2);

20 new Thread(ti2).start();

21 }

22 }

[复制代码](javascript:void(0);)

输出结果：

Hello everyone

-Hello-everyone

综上可见每个父进程创建一个子进程后，通过p.getOutputStream();p.getInputStream();建立管道连接后，无法关闭流，如果关闭了则需要重新建立进程才可以达到通信的效果。  
如果不关闭流，则传输的字符内容累加到8192byte时才可以返回。  
  
为什么是8192byte呢?

JDK 源码分析

[复制代码](javascript:void(0);)

1 class TestLambda {

2 @FunctionalInterface

3 interface A {

4 int use();

5 }

6

7 public static int getValue(int value) {

8 return value;

9 }

10

11 public void useValue(int value) {

12 A a = () -> { return getValue(value); };

13 }

14 }

15

16 Process p = Runtime.getRuntime().exec("java com.test.process.pipe.MyTest");

17

18 public Process exec(String command) throws IOException {

19 return exec(command, null, null);

20 }

21

22 public Process exec(String command, String[] envp, File dir)

23 throws IOException {

24 if (command.length() == 0)

25 throw new IllegalArgumentException("Empty command");

26

27 StringTokenizer st = new StringTokenizer(command);

28 String[] cmdarray = new String[st.countTokens()];

29 for (int i = 0; st.hasMoreTokens(); i++)

30 cmdarray[i] = st.nextToken();

31 return exec(cmdarray, envp, dir);

32 }

33

34 public Process exec(String[] cmdarray, String[] envp, File dir)

35 throws IOException {

36 return new ProcessBuilder(cmdarray)

37 .environment(envp)

38 .directory(dir)

39 .start();

40 }

[复制代码](javascript:void(0);)

接下来会执行 ProcessBuilder.start

1 return ProcessImpl.start(cmdarray,environment,dir,redirectErrorStream);

执行ProcessImpl.start(final class ProcessImpl extends Process )  
OutputStream  
InputStream 是在这里声明的  
如下：

[复制代码](javascript:void(0);)

1 //关键这个地方 创建的为FileDescriptor 管理的方式底层也是通过文件的方式实现的，原理跟linux的管道相同

2 stdin\_fd = new FileDescriptor();

3 stdout\_fd = new FileDescriptor();

4 stderr\_fd = new FileDescriptor();

5

6 handle = create(cmdstr, envblock, path, redirectErrorStream,

7 stdin\_fd, stdout\_fd, stderr\_fd);

8

9 java.security.AccessController.doPrivileged(

10 new java.security.PrivilegedAction() {

11 public Object run() {

12 stdin\_stream =

13 new BufferedOutputStream(new FileOutputStream(stdin\_fd));

14 stdout\_stream =

15 new BufferedInputStream(new FileInputStream(stdout\_fd));

16 stderr\_stream =

17 new FileInputStream(stderr\_fd);

18 return null;

19 }

20 });

21 }

[复制代码](javascript:void(0);)

Process类中的说明

[复制代码](javascript:void(0);)

1 public abstract class Process

2 {

3 /\*\*

4 \* Gets the output stream of the subprocess.

5 \* Output to the stream is piped into the standard input stream of

6 \* the process represented by this <code>Process</code> object.

7 \* <p> //该处说明OutputStream 是通过管道的方式进行的处理

8 \* Implementation note: It is a good idea for the output stream to

9 \* be buffered.

10 \*

11 \* @return the output stream connected to the normal input of the

12 \* subprocess.

13 \*/

14 abstract public OutputStream getOutputStream()

15 }

[复制代码](javascript:void(0);)

BufferedReader类中

1 private static int defaultCharBufferSize = 8192;//默认字符数组长度

另外Java中还提供了PipedInputStream、PipedOutputStream类，但这2个类用在多进程间交互是无法实现的。  
  
**总结:**  
1、如果Java中要涉及到多进程之间交互，子进程只是简单的做一些功能处理的话建议使用  
Process p = Runtime.getRuntime().exec("java \*\*\*\*类名");  
p.getOutputStream()  
p.getInputStream() 的方式进行输入、输出流的方式进行通信  
如果涉及到大量的数据需要在父子进程之间交互不建议使用该方式，该方式子类中所有的System都会返回到父类中，另该方式不太适合大并发多线程  
2、内存共享(MappedByteBuffer)  
该方法可以使用父子进程之间通信，但在高并发往内存内写数据、读数据时需要对文件内存进行锁机制，不然会出现读写内容混乱和不一致性，Java里面提供了文件锁FileLock，但这个在父/子进程中锁定后另一进程会一直等待，效率确实不够高。  
RandomAccessFile raf = new RandomAccessFile("D:/a.txt", "rw");  
FileChannel fc = raf.getChannel();   
MappedByteBuffer mbb = fc.map(MapMode.READ\_WRITE, 0, 1024);  
FileLock fl = fc.lock();//文件锁  
3、Socket 这个方式可以实现，需要在父子进程间进行socket通信  
4、队列机制 这种方式也可以实现，需要父/子进程往队列里面写数据，子/父进程进行读取。不太好的地方是需要在父子进程之间加一层队列实现，队列实现有ActiveMQ,FQueue等  
5、通过JNI方式，父/子进程通过JNI对共享进程读写  
6、基于信号方式，但该方式只能在执行kill -12或者在cmd下执行ctrl+c 才会触发信息发生。

[复制代码](javascript:void(0);)

1 OperateSignal operateSignalHandler = new OperateSignal();

2 Signal sig = new Signal("SEGV");//SEGV 这个linux和window不同

3 Signal.handle(sig, operateSignalHandler);

4

5 public class OperateSignal implements SignalHandler{

6 @Override

7 public void handle(Signal arg0) {

8 System.out.println("信号接收");

9 }

10 }

[复制代码](javascript:void(0);)

7、要是在线程间也可以使用Semaphore  
8、说明一下Java中没有命名管道