Java Shell命令调用及交互算法封装

 发表于 2018-12-12 |  分类于 [Algorithm](http://www.scassis.cn/blog/categories/Algorithm/)|  309

源网址：<http://www.scassis.cn/blog/2018/12/13/algorithm-shell-executor/#more>

**简介**

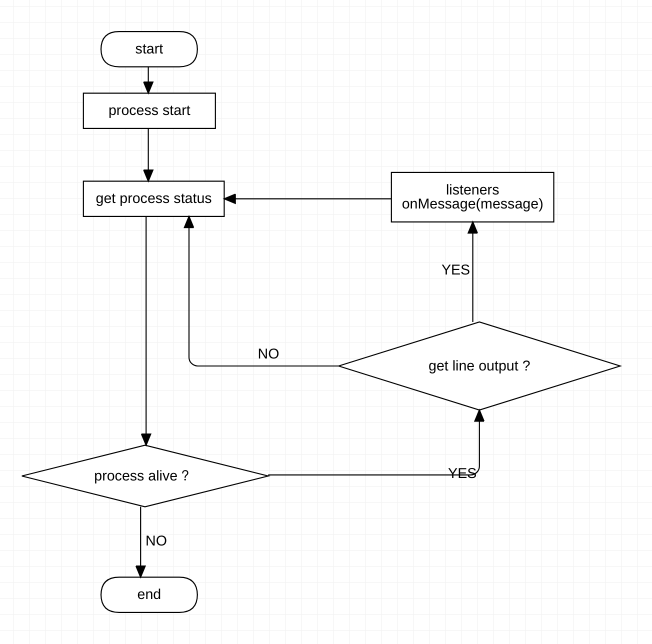
Java调用Shell命令运行子进程的方式如下代码：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | ProcessBuilder processBuilder = new ProcessBuilder("pwd"); Process process = processBuilder.start(); if (process.waitFor(timeout, TimeUnit.MILLISECONDS)) {  status = process.exitValue();  System.out.println(status); } |

现在需要在子进程运行过程中，实时收集子进程输出流的行输出，因此需要设计算法实现该功能。

**设计**

采用**监听模式**，调用子进程执行Shell命令同时注册监听器  
子进程行输出将以实时调用监听器对象传参的方式传递给监听器。

原始程序流程图：  
[](http://www.scassis.cn/blog/2018/12/13/algorithm-shell-executor/shell-executor-flow-chart.png)

**核心代码片段**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 | public static int execute(String command, String directory, Long timeout, final Communicator... communicators) throws CommandTimeoutException {  final ProcessBuilder processBuilder = new ProcessBuilder(command);  if (directory != null) {  File workDir = new File(directory);  if (workDir.exists() && workDir.isDirectory()) {  processBuilder.directory(workDir);  }  }  processBuilder.redirectErrorStream(true);  int status = -1;  try {  final Process process = processBuilder.start();  if (communicators != null && communicators.length > 0) {  communicatorExecutor.submit(() -> {  BufferedReader reader = null;  try {  InputStream inputStream = process.getInputStream();  if (inputStream == null) {  return;  }  reader = new BufferedReader(new InputStreamReader(inputStream));  String line;  while ((line = reader.readLine()) != null) {  for (Communicator communicator : communicators) {  communicator.onMessage(line, process);  }  }  } catch (Exception e) {  logger.warn(e.getMessage(), e);  } finally {  if (reader != null) {  try {  reader.close();  } catch (Exception e) {  logger.warn(e.getMessage(), e);  }  }  }  });  }  if (timeout == null || timeout <= 0) {  status = process.waitFor();  } else {  if (!process.waitFor(timeout, TimeUnit.MILLISECONDS)) {  throw new CommandTimeoutException(String.format("Command execute timeout, timeout: %s, command: %s", timeout, command));  } else {  status = process.exitValue();  }  }  TimeUnit.MILLISECONDS.sleep(100);  } catch (Exception e) {  logger.warn(e.getMessage(), e);  if (e instanceof CommandTimeoutException) {  throw (CommandTimeoutException) e;  }  }  return status; } |

说明：

| **方法参数** | **说明** |
| --- | --- |
| command | Shell命令 |
| directory | 执行命令的目录路径 |
| timeout | 命令执行等待时间，null则一直等待直到命令执行结束 |
| communicators | 交互对象列表，子进程运行输出监听器 |

**用法示例**

用法

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | @Test public void simpleTest() {  try {  int exitValue = ShellExecutor.execute(  "./test.sh",  System.getProperty("user.dir")+"/scripts",  null,  (message, process) -> System.out.println(message)  );  System.out.println("exitValue: " + exitValue);  } catch (ShellExecutor.CommandTimeoutException e) {  System.out.println(e.getMessage());  } } |

Shell脚本 **test.sh** 如下：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | #!/bin/bash  source /etc/profile  i=0 while((${i} < 20)) do  echo "Printout success, index: ${i}"  let "i++"  sleep 2 done  exit 0 |

结果输出

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 | Printout success, index: 0 Printout success, index: 1 Printout success, index: 2 Printout success, index: 3 Printout success, index: 4 Printout success, index: 5 Printout success, index: 6 Printout success, index: 7 Printout success, index: 8 Printout success, index: 9 Printout success, index: 10 Printout success, index: 11 Printout success, index: 12 Printout success, index: 13 Printout success, index: 14 Printout success, index: 15 Printout success, index: 16 Printout success, index: 17 Printout success, index: 18 Printout success, index: 19 exitValue: 0 |