Как выполнить команду оболочки в Java

В Java мы можем использовать ProcessBuilder легко вызывать внешние команды:

ProcessBuilder processBuilder = new ProcessBuilder();

// - Linux -

// Запустить команду оболочки

processBuilder.command("bash", "-c", "ls /home/mkyong/");

// Запускаем скрипт оболочки

processBuilder.command("path/to/hello.sh");

// - Windows -

// Запустить команду оболочки

processBuilder.command("cmd.exe", "/c", "dir C:\Users\mkyong");

// Запускаем файл bat

processBuilder.command("C:\Users\mkyong\hello.bat");

Process process = processBuilder.start();

## Или например

## Пинг

1.1. Выполните команду внешнего пинга, чтобы пинговать веб-сайт 3 раза, и отобразить вывод.

ProcessBuilderExample1.java

package com.csharpcoderr.process;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class ProcessBuilderExample1 {

public static void main(String[] args) {

ProcessBuilder processBuilder = new ProcessBuilder();

// Запускаем это в Windows, cmd, / c = terminate после этого запуска

processBuilder.command("cmd.exe", "/c", "ping -n 3 google.com");

try {

Process process = processBuilder.start();

// заблокирован :(

BufferedReader reader = new BufferedReader(new

InputStreamReader(process.getInputStream()));

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

int exitCode = process.waitFor();

System.out.println("nExited with error code : " + exitCode);

} catch (IOException e) {

e.printStackTrace();

} catch (InterruptedException e) {

e.printStackTrace();

}}}

Вывод

Pinging google.com [172.217.166.142] with 32 bytes of data:

Reply from 172.217.166.142: bytes=32 time=10ms TTL=55

Reply from 172.217.166.142: bytes=32 time=10ms TTL=55

Reply from 172.217.166.142: bytes=32 time=10ms TTL=55

Ping statistics for 172.217.166.142:

Packets: Sent = 3, Received = 3, Lost = 0 (0% loss),

Approximate round trip times in milli-seconds:

Minimum = 10ms, Maximum = 10ms, Average = 10ms

Exited with error code : 0

Еще одна статья

**1. Обзор**

В этом уроке мы проиллюстрируем два способа **выполнения команды оболочки из кода *Java*** .

Первый - использовать класс *Runtime* и вызвать его метод *exec* .

Вторым и более настраиваемым способом будет создание и использование экземпляра *ProcessBuilder* .

**2. Зависимость операционной системы**

Прежде чем мы собираемся создать новый *Процесс,* выполняющий нашу команду оболочки, мы должны сначала определить операционную систему, на которой работает наша *JVM* .

Это связано с тем, что в *Windows* нам нужно запустить нашу команду в качестве аргумента оболочки *cmd.exe,* а во всех других операционных системах мы можем запустить стандартную оболочку с именем *sh:*

|  |  |
| --- | --- |
| 1  2 | boolean isWindows = System.getProperty("os.name")    .toLowerCase().startsWith("windows"); |

**3. Вход и выход**

Кроме того, нам нужен способ подключиться к входным и выходным потокам нашего процесса.

По крайней мере, **вывод должен быть использован** - иначе наш процесс не вернется успешно, вместо этого он зависнет.

Давайте реализуем обычно используемый класс *StreamGobbler,* который использует *InputStream*:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | private static class StreamGobbler implements Runnable {      private InputStream inputStream;      private Consumer<String> consumer;        public StreamGobbler(InputStream inputStream, Consumer<String> consumer) {          this.inputStream = inputStream;          this.consumer = consumer;      }        @Override      public void run() {          new BufferedReader(new InputStreamReader(inputStream)).lines()            .forEach(consumer);      }  } |

**ПРИМЕЧАНИЕ.** Этот класс реализует интерфейс *Runnable* , что означает, что он может быть выполнен любым [*исполнителем*](https://www.baeldung.com/java-executor-service-tutorial) .

**4. Runtime.exec ()**

Вызов метода *Runtime.exec ()* - это простой, но еще не настраиваемый способ создания нового *подпроцесса* .

В следующем примере мы запросим каталог-каталог домашнего каталога пользователей и распечатаем его в консоли:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | String homeDirectory = System.getProperty("user.home");  Process process;  if (isWindows) {      process = Runtime.getRuntime()        .exec(String.format("cmd.exe /c dir %s", homeDirectory));  } else {      process = Runtime.getRuntime()        .exec(String.format("sh -c ls %s", homeDirectory));  }  StreamGobbler streamGobbler =    new StreamGobbler(process.getInputStream(), System.out::println);  Executors.newSingleThreadExecutor().submit(streamGobbler);  int exitCode = process.waitFor();  assert exitCode == 0; |

**5. *ProcessBuilder***

Для второй реализации нашей вычислительной задачи мы будем использовать *ProcessBuilder* . Это предпочтительнее подхода *Runtime,* потому что мы можем настроить некоторые детали.

Например, мы можем:

* измените рабочий каталог, в котором работает наша команда оболочки, используя *builder.directory ()*
* настроить пользовательскую карту ключ-значение в качестве среды с помощью *builder.environment ()*
* перенаправить потоки ввода и вывода на пользовательские замены
* наследовать их обоих в потоках текущего процесса *JVM,* используя *builder.inheritIO ()*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | ProcessBuilder builder = new ProcessBuilder();  if (isWindows) {      builder.command("cmd.exe", "/c", "dir");  } else {      builder.command("sh", "-c", "ls");  }  builder.directory(new File(System.getProperty("user.home")));  Process process = builder.start();  StreamGobbler streamGobbler =    new StreamGobbler(process.getInputStream(), System.out::println);  Executors.newSingleThreadExecutor().submit(streamGobbler);  int exitCode = process.waitFor();  assert exitCode == 0; |

**6. Заключение**

Как мы видели в этом кратком руководстве, мы можем выполнить команду оболочки в *Java* двумя различными способами.

Как правило, если вы планируете настроить выполнение порожденного процесса, например, изменить его рабочий каталог, вам следует рассмотреть возможность использования *ProcessBuilder* .

Руководство по java.lang.Process API

**1. Вступление**

В этом уроке мы собираемся **подробно взглянуть на *Process*API**.

Для более детального изучения того, как использовать *Process*для выполнения команды оболочки, мы можем обратиться к нашему предыдущему учебнику [here](https://www.baeldung.com/run-shell-command-in-java).

Процесс, на который он ссылается, является исполняемым приложением. Класс *Process*предоставляет методы для взаимодействия с этими процессами, включая извлечение вывода, выполнение ввода, мониторинг жизненного цикла, проверку состояния выхода и уничтожение (уничтожение) его.

**2. Использование класса *Process*для компиляции и запуска Java-программы**

Давайте рассмотрим пример для компиляции и запуска другой Java-программы с помощью API*Process*:

@Test

public void whenExecutedFromAnotherProgram\_\_thenSourceProgramOutput3() throws IOException {

Process process = Runtime.getRuntime()

.exec("javac -cp src src\\main\\java\\com\\baeldung\\java9\\process\\OutputStreamExample.java");

process = Runtime.getRuntime()

.exec("java -cp src/main/java com.baeldung.java9.process.OutputStreamExample");

BufferedReader output = new BufferedReader(new InputStreamReader(process.getInputStream()));

int value = Integer.parseInt(output.readLine());

assertEquals(3, value);

}

Таким образом, приложения для выполнения кода Java в существующем коде Java практически безграничны.

**3. Создание процесса**

Наше Java-приложение может вызывать любое приложение, работающее в нашей компьютерной системе, с учетом ограничений операционной системы.

Поэтому мы можем выполнять приложения. Давайте посмотрим, каковы различные варианты использования, которые мы можем использовать с помощью Process API.

ласс Process Builder позволяет нам создавать подпроцессы в вашем приложении.

Давайте посмотрим, как открыть приложение Notepad для Windows:

ProcessBuilder builder = new ProcessBuilder("notepad.exe");

Process process = builder.start();

**4. Уничтожающий процесс**

*Process*также предоставляет нам методы для уничтожения подпроцессов или процессов. **Хотя, как приложение убито, зависит от платформы**.

Давайте посмотрим, какие варианты использования возможны.

**4.1. Уничтожение процесса по ссылке**

Допустим, мы используем ОС Windows и хотим запустить приложение Notepad и уничтожить его.

Как и раньше, мы можем создать экземпляр приложения Notepad с помощью класса*ProcessBuilder*и метода *start ()*.

Затем мы можем вызвать метод *destroy ()*для нашего объекта *Process*.

\*\* 4.2. Уничтожение процесса по ID

\*\*

Мы также можем уничтожать процессы, которые выполняются в нашей операционной системе, которые не могут быть созданы нашим приложением.

При этом следует соблюдать осторожность, так как мы можем неосознанно уничтожить критический процесс, который может сделать операционную систему нестабильной \*\* .

Сначала нам нужно узнать идентификатор текущего запущенного процесса, проверив диспетчер задач и выяснив pid.

Давайте посмотрим на пример:

Optional<ProcessHandle> optionalProcessHandle = ProcessHandle.of(5232);

optionalProcessHandle.ifPresent(processHandle -> processHandle.destroy());

**4.3. Уничтожение процесса силой**

При выполнении метода *destroy ()*подпроцесс будет уничтожен, как мы видели ранее в этой статье.

* В случае, когда *destroy ()*не работает, у нас есть опция *destroyForcbly ()*. \*\*

Мы всегда должны начинать с метода *destroy ()*. После этого мы можем выполнить быструю проверку подпроцесса, выполнив *isAlive ()*.

Если он возвращает true, тогда выполните *destroyForcbly ()*:

ProcessBuilder builder = new ProcessBuilder("notepad.exe");

Process process = builder.start();

process.destroy();

if (process.isAlive()) {

process.destroyForcibly();

}

\*\* 5. Ожидание завершения процесса

\*\*

У нас также есть два перегруженных метода, с помощью которых мы можем обеспечить ожидание завершения процесса.

\*\* 5.1. *waitfor ()*

\*\*

Когда этот метод выполняется, он переводит **текущий поток процесса выполнения в состояние ожидания блокировки, если только подпроцесс не завершится**.

Давайте посмотрим на пример:

ProcessBuilder builder = new ProcessBuilder("notepad.exe");

Process process = builder.start();

assertThat(process.waitFor() >= 0);

Из приведенного выше примера видно, что текущий поток продолжает выполнение, он будет ожидать завершения потока подпроцесса. Как только подпроцесс завершится, текущий поток продолжит свое выполнение.

\*\* 5.2. *waitfor (long timeOut, TimeUnit time)*

\*\*

Когда этот метод выполняется, он переводит **текущий поток процесса выполнения в состояние ожидания блокировки, если подпроцесс не завершится или не закончится время**.

Давайте посмотрим на пример:

ProcessBuilder builder = new ProcessBuilder("notepad.exe");

Process process = builder.start();

assertFalse(process.waitFor(1, TimeUnit.SECONDS));

Из приведенного выше примера видно, что текущий поток продолжает выполнение, он будет ожидать завершения потока подпроцесса или истечения указанного интервала времени.

Когда этот метод выполняется, он вернет логическое значение true, если подпроцесс завершился, или логическое значение false, если время ожидания истекло до выхода из подпроцесса.

\*\* 6. *exitValue ()*

\*\*

Когда этот метод запущен, текущий поток не будет ждать завершения или уничтожения подпроцесса, однако он выдаст *IllegalThreadStateException*, если подпроцесс не завершен.

* Иначе, если подпроцесс был успешно завершен, это приведет к выходному значению процесса \*\* .

Это может быть любое возможное положительное целое число.

Давайте рассмотрим пример, когда метод *exitValue ()*возвращает положительное целое число, когда подпроцесс был успешно завершен:

@Test

public void

givenSubProcess\_\_whenCurrentThreadWillNotWaitIndefinitelyforSubProcessToEnd\_\_thenProcessExitValueReturnsGrt0()

throws IOException {

ProcessBuilder builder = new ProcessBuilder("notepad.exe");

Process process = builder.start();

assertThat(process.exitValue() >= 0);

}

\*\* 7. *является живым()*

\*\*

Когда мы хотим выполнить бизнес-обработку, которая является субъективной, является ли процесс живым или нет.

Мы можем выполнить быструю проверку, чтобы определить, активен ли процесс или нет, который возвращает логическое значение.

Давайте посмотрим на быстрый пример этого:

ProcessBuilder builder = new ProcessBuilder("notepad.exe");

Process process = builder.start();

Thread.sleep(10000);

process.destroy();

assertTrue(process.isAlive());

**8. Обработка технологических потоков**

По умолчанию созданный подпроцесс не имеет своего терминала или консоли. Все его стандартные операции ввода-вывода (то есть stdin, stdout, stderr) будут отправлены родительскому процессу. Таким образом, родительский процесс может использовать эти потоки для подачи входных данных и получения выходных данных из подпроцесса.

Следовательно, это дает нам огромную гибкость, поскольку дает нам контроль над вводом/выводом нашего подпроцесса.

\*\* 8.1. *getErrorStream ()*

\*\*

Интересно, что мы можем извлечь ошибки, сгенерированные из подпроцесса, и выполнить бизнес-обработку.

После этого мы можем выполнить определенные проверки бизнес-обработки на основе наших требований.

Давайте посмотрим на пример:

@Test

public void givenSubProcess\_\_whenEncounterError\_\_thenErrorStreamNotNull() throws IOException {

Process process = Runtime.getRuntime().exec(

"javac -cp src src\\main\\java\\com\\baeldung\\java9\\process\\ProcessCompilationError.java");

BufferedReader error = new BufferedReader(new InputStreamReader(process.getErrorStream()));

String errorString = error.readLine();

assertNotNull(errorString);

}

\*\* 8.2. *getInputStream ()*

\*\*

Мы также можем получить выходные данные, сгенерированные подпроцессом, и использовать их в родительском процессе, что позволяет обмениваться информацией между процессами:

@Test

public void givenSourceProgram\_\_whenReadingInputStream\_\_thenFirstLineEquals3() throws IOException {

Process process = Runtime.getRuntime().exec(

"javac -cp src src\\main\\java\\com\\baeldung\\java9\\process\\OutputStreamExample.java");

process = Runtime.getRuntime()

.exec("java -cp src/main/java com.baeldung.java9.process.OutputStreamExample");

BufferedReader output = new BufferedReader(new InputStreamReader(process.getInputStream()));

int value = Integer.parseInt(output.readLine());

assertEquals(3, value);

}

\*\* 8.3. *\_ge*\_ tOutputStream () *\_\_*

\*\*

Мы можем отправить входные данные для подпроцесса из родительского процесса:

Writer w = new OutputStreamWriter(process.getOutputStream(), "UTF-8");

w.write("send to child\n");

**8.4. Фильтровать технологические потоки**

Это идеальный вариант использования для взаимодействия с выборочными запущенными процессами.

*Process*предоставляет нам возможность выборочно фильтровать запущенные процессы на основе определенного предиката.

После этого мы можем выполнить бизнес-операции на этом выборочном наборе процессов:

@Test

public void givenRunningProcesses\_\_whenFilterOnProcessIdRange\_\_thenGetSelectedProcessPid() {

assertThat(((int) ProcessHandle.allProcesses()

.filter(ph -> (ph.pid() > 10000 && ph.pid() < 50000))

.count()) > 0);

}

**9. Заключение**

*Process*- мощный класс для взаимодействия на уровне операционной системы.

Запуск команд терминала, а также запуск, мониторинг и уничтожение приложений.