Ejecución de comandos de shell con Java

Introducción

En este apartado veremos cómo podemos aprovechar la `Runtime` y `ProcessBuilder` clases para ejecutar comandos de shell y scripts con Java.

Usamos computadoras para automatizar muchas cosas en nuestro trabajo diario. Los administradores del sistema ejecutan muchos comandos todo el tiempo, algunos de los cuales son muy repetitivos y requieren cambios mínimos entre ejecuciones.

Este proceso también está listo para la automatización. No es necesario ejecutar todo manualmente. Usando Java, podemos ejecutar comandos de shell únicos o múltiples, ejecutar scripts de shell, ejecutar la terminal / símbolo del sistema, establecer directorios de trabajo y manipular variables de entorno a través de clases centrales.

Runtime.exec ()

los `Runtime` class en Java es una clase de alto nivel, presente en cada una de las aplicaciones Java. A través de él, la propia aplicación se comunica con el entorno en el que se encuentra.

Extrayendo el tiempo de ejecución asociado con nuestra aplicación a través del `getRuntime()` método, podemos utilizar el `exec()` método para ejecutar comandos directamente o ejecutar `.bat`/`.sh` archivos.

los `exec()` El método ofrece algunas variaciones sobrecargadas:

- `public Process exec(String command)` – Ejecuta el comando contenido en `command` en un proceso separado.
- `public Process exec(String command, String[] envp)` – Ejecuta el `command`, con una serie de variables de entorno. Se proporcionan como una matriz de cadenas, siguiendo el `name=value` formato.
- `public Process exec(String command, String[] envp, File dir)` – Ejecuta el `command`, con las variables de entorno especificadas, desde dentro del `dir` directorio.
- `public Process exec(String cmdArray[])` – Ejecuta un comando en forma de una matriz de cadenas.
- `public Process exec(String cmdArray[], String[] envp)` – Ejecuta un comando con las variables de entorno especificadas.
- `public Process exec(String cmdarray[], String[] envp, File dir)` – Ejecuta un comando, con las variables de entorno especificadas, desde dentro del `dir` directorio.

Vale la pena señalar que estos procesos se ejecutan externamente desde el intérprete y dependerán del sistema.

Lo que también vale la pena señalar es la diferencia entre `String command` y `String cmdArray[]`. Logran lo mismo. UN `command` se divide en una matriz de todos modos, por lo que el uso de cualquiera de estos dos debería producir los mismos resultados.

Depende de usted decidir si `exec("dir /folder")` o `exec(new String[]{"dir", "/folder"}` es lo que le gustaría usar.

Escribamos algunos ejemplos para ver cómo estos métodos sobrecargados se diferencian entre sí.

Ejecutando un comando desde una cadena

Comencemos con el enfoque más simple de estos tres:

```
Process process = Runtime.getRuntime().exec("ping www.Pharos.sh.com");
```

Al ejecutar este código, se ejecutará el comando que hemos proporcionado en formato String. Sin embargo, no vemos nada cuando ejecutamos esto.

Para validar si esto se ejecutó correctamente, querremos obtener el `process` objeto. Usemos un `BufferedReader` para ver lo que está pasando:

```
public static void printResults(Process process) throws IOException {
    BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
    String line = "";
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
}
```

Ahora, cuando ejecutamos este método después de `exec()` método, debería producir algo similar a:

```
Pinging www.Pharos.sh.com [104.18.57.23] with 32 bytes of data:
Reply from 104.18.57.23: bytes=32 time=21ms TTL=56
Reply from 104.18.57.23: bytes=32 time=21ms TTL=56
Reply from 104.18.57.23: bytes=32 time=21ms TTL=56
Reply from 104.18.57.23: bytes=32 time=21ms TTL=56

Ping statistics for 104.18.57.23:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 21ms, Maximum = 21ms, Average = 21ms
```

Tenga en cuenta que tendremos que extraer la información del proceso del `Process` instancias a medida que avanzamos en otros ejemplos.

Especificar el directorio de trabajo

Si desea ejecutar un comando desde, digamos, una carpeta determinada, haríamos algo como:

```
Process process = Runtime.getRuntime()
        .exec("cmd /c dir", null, new File("C:\Users\"));
      //.exec("sh -c ls", null, new File("Pathname")); for non-Windows users
printResults(process);
```

Aquí, hemos proporcionado el `exec()` método con un `command`, un `null` para nuevas variables de entorno y un `new File()` que se establece como nuestro directorio de trabajo.

La suma de `cmd /c` antes de un comando como `dir` es digno de mención.

Como estoy trabajando en Windows, esto abre el `cmd` y `/c` realiza el comando siguiente. En este caso, es `dir`.

La razón por la que esto no era obligatorio para el `ping` ejemplo, pero es obligatorio para este ejemplo es agradable contestada por un usuario SO.

Ejecutar el fragmento de código anterior resultará en:

```
Volume in drive C has no label.
 Volume Serial Number is XXXX-XXXX

 Directory of C:Users

08/29/2019  05:01 PM    <DIR>          .
08/29/2019  05:01 PM    <DIR>          ..
08/18/2016  09:11 PM    <DIR>          Default.migrated
08/29/2019  05:01 PM    <DIR>          Public
05/15/2020  11:08 AM    <DIR>          User
               0 File(s)              0 bytes
               5 Dir(s)  212,555,214,848 bytes free
```

Echemos un vistazo a cómo podríamos suministrar el comando anterior en varias partes individuales, en lugar de una sola cadena:

```
Process process = Runtime.getRuntime().exec(
        new String[]{"cmd", "/c", "dir"},
        null,
        new File("C:\Users\"));

printResults(process);
```

Ejecutar este fragmento de código también resultará en:

```
Volume in drive C has no label.
 Volume Serial Number is XXXX-XXXX

 Directory of C:Users

08/29/2019  05:01 PM    <DIR>          .
08/29/2019  05:01 PM    <DIR>          ..
08/18/2016  09:11 PM    <DIR>          Default.migrated
08/29/2019  05:01 PM    <DIR>          Public
05/15/2020  11:08 AM    <DIR>          User
               0 File(s)              0 bytes
               5 Dir(s)  212,542,808,064 bytes free
```

En última instancia, independientemente del enfoque, utilizando una sola cadena o una matriz de cadenas, el comando que ingrese siempre se dividirá en una matriz antes de ser procesado por la lógica subyacente.

Cuál le gustaría usar se reduce a cuál le resulta más legible.

Usar variables de entorno

Echemos un vistazo a cómo podemos usar las variables de entorno:

```
Process process = Runtime.getRuntime().exec(
        "cmd /c echo %var1%",
```

```
        new String[]{"var1=value1"});

printResults(process);
```

Podemos proporcionar tantas variables de entorno como queramos dentro de la matriz de cadenas. Aquí, acabamos de imprimir el valor de `var1` utilizando `echo`.

Ejecutar este código devolverá:

```
value1
```

Ejecución de archivos .bat y .sh

A veces, es mucho más fácil descargar todo en un archivo y ejecutar ese archivo en lugar de agregarlo todo mediante programación.

Dependiendo de su sistema operativo, usaría `.bat` o `.sh` archivos. Creemos uno con los contenidos:

```
echo Hello World
```

Entonces, usemos el mismo enfoque que antes:

```
Process process = Runtime.getRuntime().exec(
        "cmd /c start file.bat",
        null,
        new File("C:\Users\User\Desktop\"));
```

Esto abrirá el símbolo del sistema y ejecutará el `.bat` archivo en el directorio de trabajo que hemos establecido.

Ejecutar este código seguramente da como resultado:

Con todos los sobrecargados `exec()` firmas cuidadas, echemos un vistazo a las `ProcessBuilder` class y cómo podemos ejecutar comandos usándola.

ProcessBuilder

`ProcessBuilder` es el mecanismo subyacente que ejecuta los comandos cuando usamos el `Runtime.getRuntime().exec()` método:

```
/**
 * Executes the specified command and arguments in a separate process with
 * the specified environment and working directory.
 *...
*/
public Process exec(String[] cmdarray, String[] envp, File dir) throws
IOException {
    return new ProcessBuilder(cmdarray)
        .environment(envp)
        .directory(dir)
        .start();
}
```

JavaDocs para `Runtime` clase

Echando un vistazo a cómo `ProcessBuilder` toma nuestra opinión del `exec()` y ejecuta el comando, también nos da una buena idea de cómo usarlo.

Acepta un `String[] cmdarray`, y eso es suficiente para que funcione. Alternativamente, podemos proporcionarle argumentos opcionales como el `String[] envp` y `File dir`.

Exploremos estas opciones.

ProcessBuilder: Ejecutando comando desde cadenas

En lugar de poder proporcionar una sola cadena, como `cmd /c dir`, tendremos que dividirlo en este caso. Por ejemplo, si quisiéramos enumerar los archivos en el `C:/Users` directorio como antes, haríamos:

```
ProcessBuilder processBuilder = new ProcessBuilder();
processBuilder.command("cmd", "/c", "dir C:\Users");

Process process = processBuilder.start();
printResults(process);
```

Para ejecutar realmente un `Process`, ejecutamos el `start()` comando y asigne el valor devuelto a un `Process` ejemplo.

Ejecutar este código producirá:

```
 Volume in drive C has no label.
 Volume Serial Number is XXXX-XXXX

 Directory of C:Users

08/29/2019  05:01 PM    <DIR>          .
08/29/2019  05:01 PM    <DIR>          ..
08/18/2016  09:11 PM    <DIR>          Default.migrated
08/29/2019  05:01 PM    <DIR>          Public
05/15/2020  11:08 AM    <DIR>          User
               0 File(s)              0 bytes
               5 Dir(s)  212,517,294,080 bytes free
```

Sin embargo, este enfoque no es mejor que el anterior. ¿Qué es útil con el `ProcessBuilder` clase es que es personalizable. Podemos configurar las cosas mediante programación, no solo a través de comandos.

ProcessBuilder: especifique el directorio de trabajo

En lugar de proporcionar el directorio de trabajo a través del comando, configurémoslo programáticamente:

```
processBuilder.command("cmd", "/c", "dir").directory(new File("C:\Users\"));
```

Aquí, hemos configurado el directorio de trabajo para que sea el mismo que antes, pero hemos sacado esa definición del comando en sí. La ejecución de este código proporcionará el mismo resultado que el último ejemplo.

ProcessBuilder: Variables de entorno

Utilizando `ProcessBuilder`s, es fácil recuperar una lista de variables de entorno en forma de `Map`. También es fácil establecer variables de entorno para que su programa pueda usarlas.

Consigamos las variables de entorno disponibles actualmente y luego agreguemos algunas para su uso posterior:

```
ProcessBuilder processBuilder = new ProcessBuilder();

Map<String, String> environmentVariables  = processBuilder.environment();
environmentVariables.forEach((key, value) -> System.out.println(key + value));
```

Aquí, hemos empaquetado las variables de entorno devueltas en un `Map` y corrí un `forEach()` en él para imprimir los valores en nuestra consola.

La ejecución de este código producirá una lista de las variables de entorno que tiene en su máquina:

```
DriverDataC:WindowsSystem32DriversDriverData
HerokuPathE:Heroku
ProgramDataC:ProgramData
...
```

Ahora, agreguemos una variable de entorno a esa lista y usémosla:

```
environmentVariables.put("var1", "value1");

processBuilder.command("cmd", "/c", "echo", "%var1%");
Process process = processBuilder.start();
printResults(process);
```

Ejecutar este código producirá:

```
value1
```

Por supuesto, una vez que el programa haya terminado de ejecutarse, esta variable no permanecerá en la lista.

ProcessBuilder: Ejecución de archivos .bat y .sh

Si desea ejecutar un archivo, nuevamente, solo proporcionaremos el `ProcessBuilder` instancia con la información requerida:

```
processBuilder
        .command("cmd", "/c", "start", "file.bat")
        .directory(new File("C:\Users\User\Desktop"));
Process process = processBuilder.start();
```

Al ejecutar este código, el símbolo del sistema se abre y ejecuta el `.bat` archivo: