Programación de Servicios y Procesos

PSP UT1

03 Procesos en Java

0. Comprobaciones iniciales

Por favor, arrancad Windows en la máquina virtual y comprobad que tenéis instalado Eclipse

1. Creación de un proceso en Java

Desde java se pueden ejecutar procesos en el sistema como si los ejecutáramos como comandos en la terminal (salvo que el árbol de dependencias de los procesos tendría como padre al proceso de java).

Hay 2 clases principalmente para crear estos procesos: Runtime y ProcessBuilder.

Ambas clases son gestionadas mediante la clase Process.

1.1 ProcessBuilder

ProcessBuilder

Proporciona una manera flexible y controlada de crear y gestionar procesos.

Permite especificar el comando a ejecutar y configurar aspectos como el directorio de trabajo.

1.1 ProcessBuilder

La clase ProcessBuilder maneja los siguientes atributos del proceso:

- comando,
- entorno,
- directorio de trabajo,
- recurso de entrada estándar,
- el destino de la salida estándar,
- redireccionar la salida estándar de error.

1.1 ProcessBuilder

MÉTODOS CLAVE

command(List<String> command): Define el comando a ejecutar como una lista de cadenas

```
ProcessBuilder processBuilder = new ProcessBuilder("cmd.exe", "/c", "dir");
```

1.1 ProcessBuilder

MÉTODOS CLAVE

directory(File directory): Establece el directorio de trabajo en el que se ejecutará el proceso.

```
processBuilder.directory(new File("C:\\"));
```

1.1 ProcessBuilder

MÉTODOS CLAVE

start(): Inicia el proceso que se ha configurado. Devuelve un objeto *Process*.

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

1.1 ProcessBuilder

MÉTODOS CLAVE

redirectErrorStream(boolean redirect): Redirige el flujo de error estándar a la salida estándar

```
processBuilder.redirectErrorStream(true);
```

1.1 ProcessBuilder

Ejemplo 1. Abrir Notepad y la calculadora utilizando ProcessBuilder.

1.1 ProcessBuilder. Ejemplo 1

throws se utiliza en la declaración de un método para indicar que el método puede lanzar excepciones específicas. Esto permite a otros desarrolladores saber que deben manejar esas excepciones al invocar el método

```java
import java.io.IOException;

/*Ejemplo uso de las clases ProcessBuilder y Process con método start()*/
public class ejemplo01 {

    public static void main(String[] args) throws IOException {
        //Ejecuto el proceso de notepad (abrira la app)
        ProcessBuilder pb = new ProcessBuilder("NOTEPAD");
        Process p = pb.start();

        //Ejecuto el proceso de calculadora (abrira la app)
        ProcessBuilder pb2 = new ProcessBuilder("CALC");
        Process p2 = pb2.start();


    }

}
```

Crea una instancia de *ProcessBuilder* con el comando para abrir Notepad

Inicia el proceso

1.1 ProcessBuilder

IMPORTANTE

Por defecto, el proceso (o subproceso) creado no tiene su propia terminal o consola.

1.1 ProcessBuilder

Flujos de los procesos

Cada proceso en Java creado por *ProcessBuilder* tiene tres flujos estándar:

- Entrada estándar (stdin): Utilizado para recibir datos del exterior.
- Salida estándar (stdout): Utilizado para enviar datos al exterior.
- Salida de error estándar (stderr): Utilizado para enviar mensajes de error.

1.1 ProcessBuilder

getOutputStream()

Es parte de la clase *Process* y se utiliza para obtener un flujo de salida (OutputStream) del proceso que has iniciado.

Este flujo de salida permite enviar datos a la entrada estándar (stdin) del proceso, lo que es útil cuando deseas interactuar con el proceso enviándole comandos o datos.

```
OutputStream outputStream = process.getOutputStream();
```

1.1 ProcessBuilder

redirectInput ()

Se utiliza en el contexto de la clase *ProcessBuilder* para redirigir la entrada estándar (stdin) de un proceso a un archivo o a otro flujo de entrada.

Esto es útil cuando deseas que un proceso lea datos de un archivo en lugar de esperar la entrada del usuario desde la consola.

```
// Redirigir la entrada estándar desde un archivo
processBuilder.redirectInput(new File("input.txt"));
```

1.1 ProcessBuilder

getInputStream ()

Es parte de la clase *Process* y se utiliza para obtener un flujo de entrada (*InputStream*) del proceso que se ha iniciado.

Este flujo de entrada permite leer la salida estándar (stdout) del proceso, lo que es útil para capturar y manejar los datos que el proceso genera mientras se está ejecutando

1.1 ProcessBuilder

getInputStream ()

Para leer la salida usamos el método *read*() de *InputStream*, donde nos devolverá carácter a carácter la salida generada por el comando. Ejemplo

```
InputStream is = proc.getInputStream();
int c;
while ((c = is.read()) != -1)
        System.out.print((char) c);
is.close();
```

1.1 ProcessBuilder

redirectOutput ()

Es un método de la clase *ProcessBuilder* que se utiliza para redirigir la salida estándar (stdout) de un proceso a un archivo o a otro flujo de salida.

Esto es útil cuando deseas guardar la salida de un comando o proceso en un archivo en lugar de que se imprima en la consola.

```
// Redirigir la salida estándar a un archivo
processBuilder.redirectOutput(new File("output.txt"));
```

1.1 ProcessBuilder

getErrorStream ()

Es parte de la clase *Process* y se utiliza para obtener un flujo de entrada (*InputStream*) que captura la salida de error (stderr) de un proceso que se ha iniciado.

Esto es útil para manejar cualquier mensaje de error que el proceso pueda generar durante su ejecución.

```
InputStream errorStream = process.getErrorStream();
```

1.1 ProcessBuilder

redirectError ()

Es parte de la clase *ProcessBuilder* y se utiliza para redirigir la salida de error estándar (stderr) de un proceso a un archivo o a otro flujo de salida.

Esto es útil para capturar mensajes de error de un proceso sin que se impriman en la consola, permitiendo su almacenamiento o análisis posterior

```
// Redirigir la salida de error a un archivo
processBuilder.redirectError(new File("error.log"));
```

1.1 ProcessBuilder

redirectErrorStream ()

Es parte de la clase *ProcessBuilder* y se utiliza para combinar la salida de error estándar (stderr) de un proceso con su salida estándar (stdout).

Esto significa que cualquier mensaje de error que normalmente iría a la salida de error se enviará a la misma ubicación que la salida estándar, facilitando la captura y el manejo de ambos tipos de salida.

Inicialmente, esta propiedad es *false*, significa que la salida estándar y la salida de error se envían a dos corrientes separadas.

1.2 Process

Es una clase abstracta que permite controlar los procesos nativos del sistema que devuelven las clases *Runtime* y *ProcessBuilder*.

La clase *Process* proporciona métodos para realizar:

- La entrada desde el proceso
- Obtener la salida del proceso
- Esperar a que el proceso se complete
- Comprobar el estado de salida del proceso
- Destruir el proceso.

1.2 Process

Para ejecutar los comandos de Windows que no tienen ejecutable (por ejemplo el comando *DIR*) es necesario utilizar el comando *CMD.EXE*.

Construir un objeto *ProcessBuilder*

```
Process p = new ProcessBuilder( "CMD", "C/", "DIR").start()
```

1.2 Process

waitFor ()

Es un método de la clase *Process* y se utiliza para hacer que el hilo que lo llama espere hasta que el proceso asociado termine su ejecución.

Es una herramienta útil para controlar la sincronización en programas que ejecutan procesos externos.

```
int exitCode = process.waitFor();
```

1.3 Entorno de trabajo

Cuando se lanza un programa desde Eclipse no ocurre lo mismo que cuando se lanza desde Windows.

Eclipse trabaja con unos directorios predefinidos y puede ser necesario indicar a nuestro programa cual es la ruta donde hay que buscar algo.

Para ello hay que utilizar el método *directory*

1.3 Entorno de trabajo

directory ()

Es un método de la clase *ProcessBuilder* y se utiliza para establecer el directorio de trabajo del proceso que se va a crear.

Esto significa que se puede especificar en qué directorio debe ejecutarse el proceso, lo que es útil cuando el comando que se va a ejecutar depende de ciertos archivos o directorios que deben estar en un contexto específico

```
processBuilder.directory(new File("C:\\"));
```

1.4 Finalización de un proceso lanzado

System.exit ()

Se utiliza para terminar la ejecución de la aplicación Java.

Este método permite salir de manera explícita y se puede proporcionar un código de salida que indica si la finalización fue exitosa o si ocurrió un error.

```
System.exit(int status);
```

1.4 Finalización de un proceso lanzado

System.exit ()

Por defecto, en un programa Java, si no se incluye esta orden el valor devuelto es 0, que normalmente responde a una finalización correcta del proceso.

Podemos devolver el valor entero que consideremos para indicar que ha finalizado de manera correcta o incorrecta.

Lo habitual es devolver 0 para la finalización correcta y cualquier otro valor si es incorrecta.

1.5 Métodos getInputStream y getOutputStream

Vamos a ver un par de ejemplos sobre las sentencias *getInputStream* y *getErrorStream*.

1.5 Método getInputStream

EJEMPLO

Realizar un programa que ejecute el comando DIR en CMD de Windows y que se muestre el resultado por pantalla

1.5 Método getInputStream

```java
public class Ej2_DirPantalla {

    public static void main(String[] args) throws IOException {
        // TODO Auto-generated method stub

        Process p = new ProcessBuilder("CMD","/C","DIR").start();

        try {
            //capturamos el stream de salida
            InputStream is = p.getInputStream();

            //mostramos en pantalla caracter a caracter, leyendo con read()
            int c;

            while ((c = is.read()) !=-1)
                System.out.print((char) c);
            is.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Comprobacion de error
        int exitVal;
        try {
            //Esperamos a que el subproceso p finalice
            //Recoge lo devuelva System.exit()
            exitVal = p.waitFor();
            System.out.println("Valor de Salida" + exitVal);
        }catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

}
```

1.6 Método getErrorStream

Cuando tengamos que con líneas en lugar de carácter a caracter, utilizamos la clase *BufferedReader*.

Esta clase tiene el método *readln()*

InputStream → **InputStreamReader** → **BufferedReader**

```
InputStream in=proc.getErrorStream();
BufferedReader bf=new BufferedReader (new InputStreamReader (in));
String r=bf.readLine();
String linea = null;
while ((linea = bf.readLine()) != null)
        System.out.println("ERROR >" + linea);
```

1.6 Método getErrorStream

EJEMPLO

En el ejemplo anterior, ejecuta el comando *DIR* de manera errónea y muestra por pantalla el error.

1.6 Método getErrorStream

```
        // COMPROBACION DE ERROR - 0 bien - 1 mal
        int exitVal;
        try {
            exitVal = p.waitFor();
            System.out.println("Valor de Salida: " + exitVal);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        try {
            InputStream er = p.getErrorStream();
            BufferedReader brer = new BufferedReader(new InputStreamReader(er));
            String liner = null;
            while ((liner = brer.readLine()) != null)
                System.out.println("ERROR >" + liner);
        } catch (IOException ioe) {
                ioe.printStackTrace();
        }

    }
}// Ejemplo2
```

1.7 Llamar a otro programa Java

Para llamar a otro programa Java (debe estar ya compilado, con extensión .class) se debe realizar de la siguiente forma:

```
ProcessBuilder pb = new ProcessBuilder("java", "paq1.paq2.programa");
```

Normalmente, el programa compilado .class se encuentra en la carpeta bin del proyecto y en un paquete en concreto, por lo tanto, será necesario crear un objeto *File* que referencie a dicho directorio.

1.7 Llamar a otro programa Java

EJEMPLO

Realizar un programa en Java que llame al programa anterior.

1.7 Llamar a otro programa Java

```java
public class Ejemplo03 {
  public static void main(String[] args) throws IOException {

    //creamos objeto File al directorio donde esta Ejemplo2
    File directorio = new File(".\\bin");

    //El proceso a ejecutar es Ejemplo2Error
    ProcessBuilder pb = new ProcessBuilder("java", "com.ies2.Ejemplo2Error");

    //se establece el directorio donde se encuentra el ejecutable
    pb.directory(directorio);

    System.out.printf("Directorio de trabajo: %s%n",pb.directory());

    //se ejecuta el proceso
    Process p = pb.start();


   //obtener la salida devuelta por el proceso
    try {
        InputStream is = p.getInputStream();
        int c;
        while ((c = is.read()) != -1)
            System.out.print((char) c);
        is.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

  }
}// Ejemplo3
```

1.7 Llamar a otro programa Java

Recordatorio paso de parámetros.

```
public class Factorial {
      public static void main(String[] args) {
            int numero = Integer.parseInt(args[0]);
            int contador=1;
            for(int i = numero; i>0; i--) {contador=contador*i;}
            System.out.println(contador);
      }
}
```

Para crear un proceso pasando parámetros:

```
pb = new ProcessBuilder ("java", "Factorial", n1.toString());
```