Java.lang.Process class in Java

- Difficulty Level : Medium
- Last Updated : 19 Jun, 2017

The abstract **Process** class a process- that is, an executing program. Methods provided by the **Process** is used to perform input, output, waiting for the process o complete, checking exit status of the process and destroying process.

- It extends class **Object**.
- It is used primarily as a superclass for the type of object created by exec() in the Runtime class.
- **ProcessBuilder.start()** and **Runtime.getRuntime.exec()** methods creates a native process and return an instance of a subclass of **Process** that can be used to control the process and obtain information about it.
- ProcessBuilder.start() is the most preferred way to create process.

ProcessBuilder.start() vs Runtime.getRuntime.exec(): ProcessBuilder allows us to redirect the standard error of the child process into its standard output. Now we don't need two separate threads one reading from **stdout** and one reading from **stderr**.

Constructor

- **Process():** This is the only constructor.

Methods:

1. **void destroy():** Kills the subprocess.

   ```
   Syntax: public abstract void destroy().
   Returns: NA.
   Exception: NA.
   ```

   ```
   // Java code illustrating destroy()
   // method for windows operating system
   public class ProcessDemo
       public static void main(String[] args)
       {
           try
           {

               // create a new process
               System.out.println("Creating Process");

               ProcessBuilder builder = new
   ProcessBuilder("notepad.exe");
               Process pro = builder.start();

               // wait 10 seconds
               System.out.println("Waiting");
               Thread.sleep(10000);
   ```

```
            // kill the process
            pro.destroy();
            System.out.println("Process destroyed");

        }
            catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
}
```

Output:

```
Creating Process
Waiting
Process destroyed
```

```
// Java code illustrating destroy()
// method for Mac Operating System
import java.lang.*;
import java.io.*;
class ProcessDemo
{
    public static void main(String arg[]) throws IOException,
Exception
    {
        System.out.println("Creating process");

        //creating process
        ProcessBuilder p = new ProcessBuilder(new String[]
                        {"open",
"/Applications/Facetime.app"});
        Process pro = p.start();

        //waiting for 10 second
        Thread.sleep(10000);

        System.out.println("destroying process");

        //destroying process
        pro.destroy();
    }
}
```

Output:

```
Creating process
destroying process
```

2. **int exitValue():** This method returns the exit value for the subprocess.

```
Syntax: public abstract int exitValue().
Returns: This method returns the exit value of
the subprocess represented by this Process object.
By convention, the value 0 indicates normal termination.
Exception: IllegalThreadStateException ,
if the subprocess represented by this Process object has not yet
terminated.
```

```
// Java code illustrating exitValue() method
public class ProcessDemo
{
    public static void main(String[] args)
    {
        try
        {
            // create a new process
            System.out.println("Creating Process");

            ProcessBuilder builder = new
ProcessBuilder("notepad.exe");
            Process pro = builder.start();

            // kill the process
            pro.destroy();

            // checking the exit value of subprocess
            System.out.println("exit value: " +
pro.exitValue());

        }
            catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
}
```

Output:

```
Creating Process
1
```

3. **abstract InputStream getErrorStream():** This method gets the input stream of the subprocess.

```
Syntax: public abstract InputStream getInputStream().
Returns: input stream that reads input from the process out output stream.
Exception: NA.
```

```
// Java code illustrating
// getInputStream() method
import java.lang.*;
import java.io.*;
class ProcessDemo
{
    public static void main(String arg[]) throws IOException,
Exception
    {
        // creating the process
        Runtime r = Runtime.getRuntime();

        // shell script for loop from 1 to 3
        String[] nargs = {"sh", "-c", "for i in
                1 2 3; do echo $i; done"};
        Process p = r.exec(nargs);

        BufferedReader is =
            new BufferedReader(new
InputStreamReader(p.getInputStream()));
        String line;

        // reading the output
        while ((line = is.readLine()) != null)

        System.out.println(line);
    }
}
```

Output:

```
1
2
3
```

4. **abstract OutputStream getOutputStream():** This method gets the output stream of the subprocess. Output to the stream is piped into the standard input stream of the process represented by this Process object.

Syntax: public abstract OutputStream getOutputStream()
Returns: the output stream connected to the normal input of the subprocess.
Exception: NA.

```
// Java code illustrating
// getOutputStream() method
import java.io.BufferedOutputStream;
import java.io.OutputStream;

public class ProcessDemo
{
    public static void main(String[] args)
    {
        try
        {
            // create a new process
            System.out.println("Creating Process");
            Process p =
Runtime.getRuntime().exec("notepad.exe");

            // get the output stream
            OutputStream out = p.getOutputStream();

            // close the output stream
            System.out.println("Closing the output stream");
            out.close();
        }
            catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
}
```

Output:

```
Creating Process...
Closing the output stream...
```

5. **abstract InputStream getErrorStream():** It returns an input stream that reads input from the process **err** output stream.

Syntax: public abstract InputStream getErrorStream().
Returns: the input stream connected to the error stream of the subprocess.
Exception: NA.

```
// Java code illustrating
// getErrorStream() method
import java.io.InputStream;

public class ProcessDemo
{
    public static void main(String[] args)
    {
        try
        {
            // create a new process
            System.out.println("Creating Process");

            Process p =
Runtime.getRuntime().exec("notepad.exe");

            // get the error stream of the process and print
it
            InputStream error = p.getErrorStream();
            for (int i = 0; i < error.available(); i++)
            {
                System.out.println("" + error.read());
            }

            // wait for 10 seconds and then destroy the
process
            Thread.sleep(10000);
            p.destroy();

        }

        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
}
```

Output:

```
Creating Process
```

6. **int waitFor():** Returns the exit code returned by the process. This method does not return until the process on which it is called terminates.

```
Syntax: public int waitFor().
Returns: the exit value of the process. By convention, 0 indicates normal
termination.
Exception: throws InterruptedException.
```

```
// Java code illustrating
// waitFor() method

public class ProcessDemo
{
    public static void main(String[] args)
    {
        try
        {
            // create a new process
            System.out.println("Creating Process");
            Process p =
Runtime.getRuntime().exec("notepad.exe");

            // cause this process to stop
                // until process p is terminated
            p.waitFor();

            // when you manually close notepad.exe
                // program will continue here
            System.out.println("Waiting over");
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }
}
```