

java.lang

Class ProcessBuilder

java.lang.Object

java.lang.ProcessBuilder

```
public final class ProcessBuilder
extends Object
```

This class is used to create operating system processes.

Each `ProcessBuilder` instance manages a collection of process attributes. The `start()` method creates a new `Process` instance with those attributes. The `start()` method can be invoked repeatedly from the same instance to create new subprocesses with identical or related attributes.

Each process builder manages these process attributes:

- a *command*, a list of strings which signifies the external program file to be invoked and its arguments, if any. Which string lists represent a valid operating system command is system-dependent. For example, it is common for each conceptual argument to be an element in this list, but there are operating systems where programs are expected to tokenize command line strings themselves - on such a system a Java implementation might require commands to contain exactly two elements.
- an *environment*, which is a system-dependent mapping from *variables* to *values*. The initial value is a copy of the environment of the current process (see `System.getenv()`).
- a *working directory*. The default value is the current working directory of the current process, usually the directory named by the system property `user.dir`.
- a source of *standard input*. By default, the subprocess reads input from a pipe. Java code can access this pipe via the output stream returned by `Process.getOutputStream()`. However, standard input may be redirected to another source using `redirectInput`. In this case, `Process.getOutputStream()` will return a *null output stream*, for which:
 - the `write` methods always throw `IOException`
 - the `close` method does nothing
- a destination for *standard output* and *standard error*. By default, the subprocess writes standard output and standard error to pipes. Java code can access these pipes via the input streams returned by `Process.getInputStream()` and `Process.getErrorStream()`. However, standard output and standard error may be redirected to other destinations using `redirectOutput` and `redirectError`. In this case, `Process.getInputStream()` and/or `Process.getErrorStream()` will return a *null input stream*, for which:
 - the `read` methods always return `-1`
 - the `available` method always returns `0`
 - the `close` method does nothing
- a *redirectErrorStream* property. Initially, this property is `false`, meaning that the standard output and error output of a subprocess are sent to two separate streams, which can be accessed using the `Process.getInputStream()` and `Process.getErrorStream()` methods.

If the value is set to `true`, then:

- standard error is merged with the standard output and always sent to the same destination (this makes it easier to correlate error messages with the corresponding output)
- the common destination of standard error and standard output can be redirected using `redirectOutput`
- any redirection set by the `redirectError` method is ignored when creating a subprocess
- the stream returned from `Process.getErrorStream()` will always be a *null input stream*

Modifying a process builder's attributes will affect processes subsequently started by that object's `start()` method, but will never affect previously started processes or the Java process itself.

Most error checking is performed by the `start()` method. It is possible to modify the state of an object so that `start()` will fail. For example, setting the command attribute to an empty list will not throw an exception unless `start()` is invoked.

Note that this class is not synchronized. If multiple threads access a `ProcessBuilder` instance concurrently, and at least one of the threads modifies one of the attributes structurally, it *must* be synchronized externally.

Starting a new process which uses the default working directory and environment is easy:

```
Process p = new ProcessBuilder("myCommand", "myArg").start();
```

Here is an example that starts a process with a modified working directory and environment, and redirects standard output and error to be appended to a log file:

```
ProcessBuilder pb =
    new ProcessBuilder("myCommand", "myArg1", "myArg2");
Map<String, String> env = pb.environment();
env.put("VAR1", "myValue");
env.remove("OTHERVAR");
env.put("VAR2", env.get("VAR1") + "suffix");
pb.directory(new File("myDir"));
File log = new File("log");
pb.redirectErrorStream(true);
pb.redirectOutput(Redirect.appendTo(log));
Process p = pb.start();
assert pb.redirectInput() == Redirect.PIPE;
assert pb.redirectOutput().file() == log;
assert p.getInputStream().read() == -1;
```

To start a process with an explicit set of environment variables, first call `Map.clear()` before adding environment variables.

Since:

1.5

Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
static class	ProcessBuilder.Redirect Represents a source of subprocess input or a destination of subprocess output.

Constructor Summary

Constructors

Constructor and Description
ProcessBuilder(List<String> command) Constructs a process builder with the specified operating system program and arguments.
ProcessBuilder(String... command) Constructs a process builder with the specified operating system program and arguments.

Method Summary

Methods

Modifier and Type	Method and Description
List<String>	command() Returns this process builder's operating system program and arguments.
ProcessBuilder	command(List<String> command) Sets this process builder's operating system program and arguments.
ProcessBuilder	command(String... command) Sets this process builder's operating system program and arguments.
File	directory()

	Returns this process builder's working directory.
ProcessBuilder	directory (File directory) Sets this process builder's working directory.
Map<String,String>	environment () Returns a string map view of this process builder's environment.
ProcessBuilder	inheritIO () Sets the source and destination for subprocess standard I/O to be the same as those of the current Java process.
ProcessBuilder.Redirect	redirectError () Returns this process builder's standard error destination.
ProcessBuilder	redirectError (File file) Sets this process builder's standard error destination to a file.
ProcessBuilder	redirectError (ProcessBuilder.Redirect destination) Sets this process builder's standard error destination.
boolean	redirectErrorStream () Tells whether this process builder merges standard error and standard output.
ProcessBuilder	redirectErrorStream (boolean redirectErrorStream) Sets this process builder's redirectErrorStream property.
ProcessBuilder.Redirect	redirectInput () Returns this process builder's standard input source.
ProcessBuilder	redirectInput (File file) Sets this process builder's standard input source to a file.
ProcessBuilder	redirectInput (ProcessBuilder.Redirect source) Sets this process builder's standard input source.
ProcessBuilder.Redirect	redirectOutput () Returns this process builder's standard output destination.
ProcessBuilder	redirectOutput (File file) Sets this process builder's standard output destination to a file.
ProcessBuilder	redirectOutput (ProcessBuilder.Redirect destination) Sets this process builder's standard output destination.
Process	start () Starts a new process using the attributes of this process builder.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ProcessBuilder
<pre>public ProcessBuilder(List<String> command)</pre> <p>Constructs a process builder with the specified operating system program and arguments. This constructor does <i>not</i> make a copy of the command list. Subsequent updates to the list will be reflected in the state of the process builder. It is not checked whether command corresponds to a valid operating system command.</p> <p>Parameters:</p> <p>command - the list containing the program and its arguments</p> <p>Throws:</p> <p><code>NullPointerException</code> - if the argument is null</p>

ProcessBuilder

```
public ProcessBuilder(String... command)
```

Constructs a process builder with the specified operating system program and arguments. This is a convenience constructor that sets the process builder's command to a string list containing the same strings as the command array, in the same order. It is not checked whether command corresponds to a valid operating system command.

Parameters:

command - a string array containing the program and its arguments

Method Detail

command

```
public ProcessBuilder command(List<String> command)
```

Sets this process builder's operating system program and arguments. This method does *not* make a copy of the command list. Subsequent updates to the list will be reflected in the state of the process builder. It is not checked whether command corresponds to a valid operating system command.

Parameters:

command - the list containing the program and its arguments

Returns:

this process builder

Throws:

`NullPointerException` - if the argument is null

command

```
public ProcessBuilder command(String... command)
```

Sets this process builder's operating system program and arguments. This is a convenience method that sets the command to a string list containing the same strings as the command array, in the same order. It is not checked whether command corresponds to a valid operating system command.

Parameters:

command - a string array containing the program and its arguments

Returns:

this process builder

command

```
public List<String> command()
```

Returns this process builder's operating system program and arguments. The returned list is *not* a copy. Subsequent updates to the list will be reflected in the state of this process builder.

Returns:

this process builder's program and its arguments

environment

```
public Map<String,String> environment()
```

Returns a string map view of this process builder's environment. Whenever a process builder is created, the environment is initialized to a copy of the current process environment (see `System.getenv()`). Subprocesses subsequently started by this object's `start()` method will use this map as their environment.

The returned object may be modified using ordinary `Map` operations. These modifications will be visible to subprocesses started via the `start()` method. Two `ProcessBuilder` instances always contain independent process environments, so changes to the returned map will never be reflected in any other `ProcessBuilder` instance or the values returned by `System.getenv`.

If the system does not support environment variables, an empty map is returned.

The returned map does not permit null keys or values. Attempting to insert or query the presence of a null key or value will throw a `NullPointerException`. Attempting to query the presence of a key or value which is not of type `String` will throw a `ClassCastException`.

The behavior of the returned map is system-dependent. A system may not allow modifications to environment variables or may forbid certain variable names or values. For this reason, attempts to modify the map may fail with `UnsupportedOperationException` or `IllegalArgumentException` if the modification is not permitted by the operating system.

Since the external format of environment variable names and values is system-dependent, there may not be a one-to-one mapping between them and Java's Unicode strings. Nevertheless, the map is implemented in such a way that environment variables which are not modified by Java code will have an unmodified native representation in the subprocess.

The returned map and its collection views may not obey the general contract of the `Object.equals(java.lang.Object)` and `Object.hashCode()` methods.

The returned map is typically case-sensitive on all platforms.

If a security manager exists, its `checkPermission` method is called with a `RuntimePermission("getenv.*")` permission. This may result in a `SecurityException` being thrown.

When passing information to a Java subprocess, `system properties` are generally preferred over environment variables.

Returns:

this process builder's environment

Throws:

`SecurityException` - if a security manager exists and its `checkPermission` method doesn't allow access to the process environment

See Also:

`Runtime.exec(String[],String[],java.io.File)`, `System.getenv()`

directory

```
public File directory()
```

Returns this process builder's working directory. Subprocesses subsequently started by this object's `start()` method will use this as their working directory. The returned value may be `null` -- this means to use the working directory of the current Java process, usually the directory named by the system property `user.dir`, as the working directory of the child process.

Returns:

this process builder's working directory

directory

```
public ProcessBuilder directory(File directory)
```

Sets this process builder's working directory. Subprocesses subsequently started by this object's `start()` method will use this as their working directory. The argument may be `null` -- this means to use the working directory of the current Java process, usually the directory named by the system property `user.dir`, as the working directory of the child process.

Parameters:

directory - the new working directory

Returns:

this process builder

redirectInput

```
public ProcessBuilder redirectInput(ProcessBuilder.Redirect source)
```

Sets this process builder's standard input source. Subprocesses subsequently started by this object's `start()` method obtain their standard input from this source.

If the source is `Redirect.PIPE` (the initial value), then the standard input of a subprocess can be written to using the output stream returned by `Process.getOutputStream()`. If the source is set to any other value, then `Process.getOutputStream()` will return a null output stream.

Parameters:

source - the new standard input source

Returns:

this process builder

Throws:

`IllegalArgumentException` - if the redirect does not correspond to a valid source of data, that is, has type `WRITE` or `APPEND`

Since:

1.7

redirectOutput

```
public ProcessBuilder redirectOutput(ProcessBuilder.Redirect destination)
```

Sets this process builder's standard output destination. Subprocesses subsequently started by this object's `start()` method send their standard output to this destination.

If the destination is `Redirect.PIPE` (the initial value), then the standard output of a subprocess can be read using the input stream returned by `Process.getInputStream()`. If the destination is set to any other value, then `Process.getInputStream()` will return a null input stream.

Parameters:

destination - the new standard output destination

Returns:

this process builder

Throws:

`IllegalArgumentException` - if the redirect does not correspond to a valid destination of data, that is, has type `READ`

Since:

1.7

redirectError

```
public ProcessBuilder redirectError(ProcessBuilder.Redirect destination)
```

Sets this process builder's standard error destination. Subprocesses subsequently started by this object's `start()` method send their standard error to this destination.

If the destination is `Redirect.PIPE` (the initial value), then the error output of a subprocess can be read using the input stream returned by `Process.getErrorStream()`. If the destination is set to any other value, then `Process.getErrorStream()` will return a `null` input stream.

If the `redirectErrorStream` attribute has been set `true`, then the redirection set by this method has no effect.

Parameters:

`destination` - the new standard error destination

Returns:

this process builder

Throws:

`IllegalArgumentException` - if the redirect does not correspond to a valid destination of data, that is, has type `READ`

Since:

1.7

redirectInput

```
public ProcessBuilder redirectInput(File file)
```

Sets this process builder's standard input source to a file.

This is a convenience method. An invocation of the form `redirectInput(file)` behaves in exactly the same way as the invocation `redirectInput(Redirect.from(file))`.

Parameters:

`file` - the new standard input source

Returns:

this process builder

Since:

1.7

redirectOutput

```
public ProcessBuilder redirectOutput(File file)
```

Sets this process builder's standard output destination to a file.

This is a convenience method. An invocation of the form `redirectOutput(file)` behaves in exactly the same way as the invocation `redirectOutput(Redirect.to(file))`.

Parameters:

`file` - the new standard output destination

Returns:

this process builder

Since:

1.7

redirectError

```
public ProcessBuilder redirectError(File file)
```

Sets this process builder's standard error destination to a file.

This is a convenience method. An invocation of the form `redirectError(file)` behaves in exactly the same way as the invocation `redirectError(Redirect.to(file))`.

Parameters:

`file` - the new standard error destination

Returns:

this process builder

Since:

1.7

redirectInput

```
public ProcessBuilder.Redirect redirectInput()
```

Returns this process builder's standard input source. Subprocesses subsequently started by this object's `start()` method obtain their standard input from this source. The initial value is `Redirect.PIPE`.

Returns:

this process builder's standard input source

Since:

1.7

redirectOutput

```
public ProcessBuilder.Redirect redirectOutput()
```

Returns this process builder's standard output destination. Subprocesses subsequently started by this object's `start()` method redirect their standard output to this destination. The initial value is `Redirect.PIPE`.

Returns:

this process builder's standard output destination

Since:

1.7

redirectError

```
public ProcessBuilder.Redirect redirectError()
```

Returns this process builder's standard error destination. Subprocesses subsequently started by this object's `start()` method redirect their standard error to this destination. The initial value is `Redirect.PIPE`.

Returns:

this process builder's standard error destination

Since:

1.7

inheritIO

```
public ProcessBuilder inheritIO()
```

Sets the source and destination for subprocess standard I/O to be the same as those of the current Java process.

This is a convenience method. An invocation of the form


```
pb.inheritIO()
```

behaves in exactly the same way as the invocation

```
pb.redirectInput(Redirect.INHERIT)
  .redirectOutput(Redirect.INHERIT)
  .redirectError(Redirect.INHERIT)
```

This gives behavior equivalent to most operating system command interpreters, or the standard C library function `system()`.

Returns:

this process builder

Since:

1.7

redirectErrorStream

```
public boolean redirectErrorStream()
```

Tells whether this process builder merges standard error and standard output.

If this property is `true`, then any error output generated by subprocesses subsequently started by this object's `start()` method will be merged with the standard output, so that both can be read using the `Process.getInputStream()` method. This makes it easier to correlate error messages with the corresponding output. The initial value is `false`.

Returns:

this process builder's `redirectErrorStream` property

redirectErrorStream

```
public ProcessBuilder redirectErrorStream(boolean redirectErrorStream)
```

Sets this process builder's `redirectErrorStream` property.

If this property is `true`, then any error output generated by subprocesses subsequently started by this object's `start()` method will be merged with the standard output, so that both can be read using the `Process.getInputStream()` method. This makes it easier to correlate error messages with the corresponding output. The initial value is `false`.

Parameters:

`redirectErrorStream` - the new property value

Returns:

this process builder

start

```
public Process start()
    throws IOException
```

Starts a new process using the attributes of this process builder.

The new process will invoke the command and arguments given by `command()`, in a working directory as given by `directory()`, with a process environment as given by `environment()`.

This method checks that the command is a valid operating system command. Which commands are valid is system-dependent, but at the very least the command must be a non-empty list of non-null strings.

A minimal set of system dependent environment variables may be required to start a process on some operating systems. As a result, the subprocess may inherit additional environment variable settings beyond those in the process builder's `environment()`.

If there is a security manager, its `checkExec` method is called with the first component of this object's command array as its argument. This may result in a `SecurityException` being thrown.

Starting an operating system process is highly system-dependent. Among the many things that can go wrong are:

- The operating system program file was not found.
- Access to the program file was denied.
- The working directory does not exist.

In such cases an exception will be thrown. The exact nature of the exception is system-dependent, but it will always be a subclass of `IOException`.

Subsequent modifications to this process builder will not affect the returned `Process`.

Returns:

a new `Process` object for managing the subprocess

Throws:

`NullPointerException` - if an element of the command list is null

`IndexOutOfBoundsException` - if the command is an empty list (has size 0)

`SecurityException` - if a security manager exists and

- its `checkExec` method doesn't allow creation of the subprocess, or
- the standard input to the subprocess was `redirected from a file` and the security manager's `checkRead` method denies read access to the file, or
- the standard output or standard error of the subprocess was `redirected to a file` and the security manager's `checkWrite` method denies write access to the file

`IOException` - if an I/O error occurs

See Also:

`Runtime.exec(String[], String[], java.io.File)`

[Overview](#) [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

Java™ Platform
Standard Ed. 7

[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#) [All Classes](#)

Summary: [Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail: Field](#) | [Constr](#) | [Method](#)

Submit a bug or feature

For further API reference and developer documentation, see [Java SE Documentation](#). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright © 1993, 2018, Oracle and/or its affiliates. All rights reserved. Use is subject to [license terms](#). Also see the [documentation redistribution policy](#). [Modify Préférences de cookies](#). [Modify Ad Choices](#).