Node.js - Scaling Application

Node.js runs in a single-thread mode, but it uses an event-driven paradigm to handle concurrency. It also facilitates creation of child processes to leverage parallel processing on multi-core CPU based systems.

Child processes always have three streams **child.stdin**, **child.stdout**, and **child.stderr** which may be shared with the stdio streams of the parent process.

Node provides **child_process** module which has the following three major ways to create a child process.

exec − child_process.exec method runs a command in a shell/console and buffers the output.
spawn − child_process.spawn launches a new process with a given command.
fork − The child_process.fork method is a special case of the spawn() to create child processes.

The exec() method

child_process.exec method runs a command in a shell and buffers the output. It has the following signature −

```
child_process.exec(command[, options], callback)
```

Parameters

Here is the description of the parameters used −

command (String) The command to run, with space-separated arguments
options (Object) may comprise one or more of the following options −
cwd (String) Current working directory of the child process
env (Object) Environment key-value pairs
encoding (String) (Default: 'utf8')
shell (String) Shell to execute the command with (Default: '/bin/sh' on UNIX, 'cmd.exe' on Windows, The shell should understand the -c switch on UNIX or /s /c on Windows. On Windows, command line parsing should be compatible with cmd.exe.)
timeout (Number) (Default: 0)
maxBuffer (Number) (Default: 200*1024)

killSignal (String) (Default: 'SIGTERM')
uid (Number) Sets the user identity of the process.
gid (Number) Sets the group identity of the process.

callback The function gets three arguments **error**, **stdout,** and **stderr** which are called with the output when the process terminates.

The exec() method returns a buffer with a max size and waits for the process to end and tries to return all the buffered data at once.

Example

Let us create two js files named support.js and master.js −

File: support.js

```
console.log("Child Process " + process.argv[2] + " executed." );
```

File: master.js

```
const fs = require('fs');
const child_process = require('child_process');

for(var i=0; i<3; i++) {
   var workerProcess = child_process.exec('node support.js '+i,function
      (error, stdout, stderr) {

      if (error) {
         console.log(error.stack);
         console.log('Error code: '+error.code);
         console.log('Signal received: '+error.signal);
      }
      console.log('stdout: ' + stdout);
      console.log('stderr: ' + stderr);
   });

   workerProcess.on('exit', function (code) {
      console.log('Child process exited with exit code '+code);
   });
}
```

Now run the master.js to see the result −

```
$ node master.js
```

Verify the Output. Server has started.

```
Child process exited with exit code 0
stdout: Child Process 1 executed.

stderr:
Child process exited with exit code 0
stdout: Child Process 0 executed.

stderr:
Child process exited with exit code 0
stdout: Child Process 2 executed.
```

The spawn() Method

child_process.spawn method launches a new process with a given command. It has the following signature −

```
child_process.spawn(command[, args][, options])
```

Parameters

Here is the description of the parameters used −

command (String) The command to run
args (Array) List of string arguments
options (Object) may comprise one or more of the following options −
cwd (String) Current working directory of the child process.
env (Object) Environment key-value pairs.
stdio (Array) String Child's stdio configuration.
customFds (Array) Deprecated File descriptors for the child to use for stdio.
detached (Boolean) The child will be a process group leader.
uid (Number) Sets the user identity of the process.
gid (Number) Sets the group identity of the process.

The spawn() method returns streams (stdout &stderr) and it should be used when the process returns a volume amount of data. spawn() starts receiving the response as soon as the process starts executing.

Example

Create two js files named support.js and master.js −

File: support.js

```
console.log("Child Process " + process.argv[2] + " executed." );
```

File: master.js

```
const fs = require('fs');
const child_process = require('child_process');

for(var i = 0; i<3; i++) {
   var workerProcess = child_process.spawn('node', ['support.js', i]);

   workerProcess.stdout.on('data', function (data) {
      console.log('stdout: ' + data);
   });

   workerProcess.stderr.on('data', function (data) {
      console.log('stderr: ' + data);
   });

   workerProcess.on('close', function (code) {
      console.log('child process exited with code ' + code);
   });
}
```

Now run the master.js to see the result −

```
$ node master.js
```

Verify the Output. Server has started

```
stdout: Child Process 0 executed.

child process exited with code 0
stdout: Child Process 1 executed.

stdout: Child Process 2 executed.

child process exited with code 0
child process exited with code 0
```

The fork() Method

child_process.fork method is a special case of spawn() to create Node processes. It has the following signature −

```
child_process.fork(modulePath[, args][, options])
```

Parameters

Here is the description of the parameters used −

modulePath (String) The module to run in the child.
args (Array) List of string arguments
options (Object) may comprise one or more of the following options −
cwd (String) Current working directory of the child process.
env (Object) Environment key-value pairs.
execPath (String) Executable used to create the child process.
execArgv (Array) List of string arguments passed to the executable (Default: process.execArgv).
silent (Boolean) If true, stdin, stdout, and stderr of the child will be piped to the parent, otherwise they will be inherited from the parent, see the "pipe" and "inherit" options for spawn()'s stdio for more details (default is false).
uid (Number) Sets the user identity of the process.
gid (Number) Sets the group identity of the process.

The fork method returns an object with a built-in communication channel in addition to having all the methods in a normal ChildProcess instance.

Example

Create two js files named support.js and master.js −

File: support.js

```
console.log("Child Process " + process.argv[2] + " executed." );
```

File: master.js

```
const fs = require('fs');
const child_process = require('child_process');

for(var i=0; i<3; i++) {
   var worker_process = child_process.fork("support.js", [i]);

   worker_process.on('close', function (code) {
      console.log('child process exited with code ' + code);
   });
}
```

Now run the master.js to see the result −

```
$ node master.js
```

Verify the Output. Server has started.

```
Child Process 0 executed.
Child Process 1 executed.
Child Process 2 executed.
child process exited with code 0
child process exited with code 0
child process exited with code 0
```

Revision #1
Created 16 December 2022 10:32:24 by Admin
Updated 16 December 2022 10:33:02 by Admin