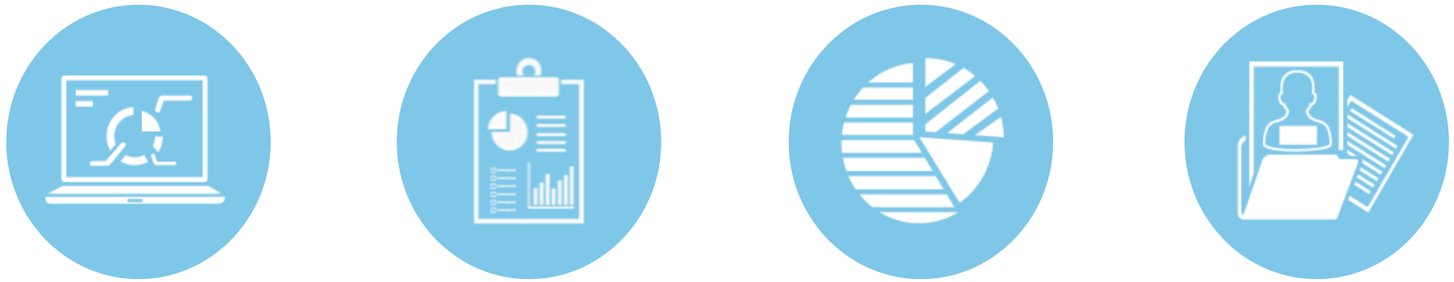
Node.js

Lesson 08—Working with Multiprocess



Lesson Overview

In this lesson, you will be able to work with Child process API and Cluster API.

Learning Objectives

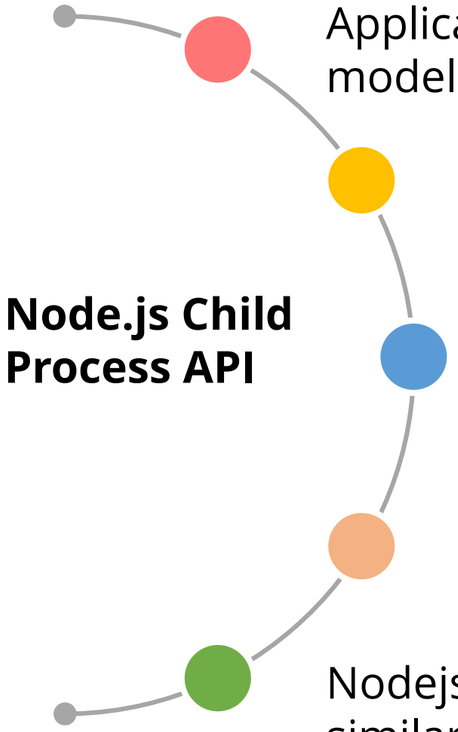
- Working with Node.js Child Process API
- Working with Cluster API

Working with Multiprocess

Topic 1—Working with Node.js Child Process API

Node.js Child Process API

INTRODUCTION



Applications based on node.js deal with I/O very efficiently because of event-driven asynchronous model.

While dealing with CPU-intensive processes, it blocks the event loop and reduces the application responsiveness.

CPU-intensive tasks must be maintained by a different process to keep the event loop free.

Node helps developers to maintain the load balance by spawning a new process and assigning the new task to it using child_process.

Nodejs module “child_process” provides the capabilities to spawn all child processes in similar, but not identical ways.

Node.js Child Process API

CREATING A CHILD PROCESS

Node.js comes pre-loaded with child_process module which uses three major methods to create a child process.

Exec(), spawn(), fork()

child_process.exec: It runs a command in a console and buffers the output.

child_process.spawn: It launches a new process with a given command.

child_process.fork: It is a specialized version of spawn() method that develops child processes.

Node.js Child Process API

EXEC() METHOD

The exec() method is the easiest one to develop a child process.

When exec() method is invoked, a new command prompt is launched to execute the command string.

exec() method takes three standard arguments:

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

`Command`: Takes command to execute

`Options`: Is an optional configurable object

`Callback`: Comes with three arguments, i.e., error, *stdout*, *stderr*.

Node.js Child Process API

EXEC() METHOD - EXAMPLE

```
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 will exit code '+code);
 });
}
```

Node.js Child Process API

SPAWN() METHOD

Spawn method is designed to offer a more complex connection between two processes (i.e., child and parent).

Spawn method is preloaded with a number of events and methods to connect with the child process.

Spawn() takes three arguments:

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

`command()` executes without additional arguments. An array to provide arguments is `args`, which will contain the list of arguments.

Spawn() method does not accept callback function; it returns a child process Object.

Demo for Node.js Child Process API

Node.js Child Process API

FORK() METHOD

child_process.fork method is a unique type of spawn() to develop Node processes.

fork() method returns an object with a pre-loaded channel, apart having all the methods in a normal ChildProcess instance.

Syntax:

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

```
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);
 });
}
```

Demo for Node.js Child Process API

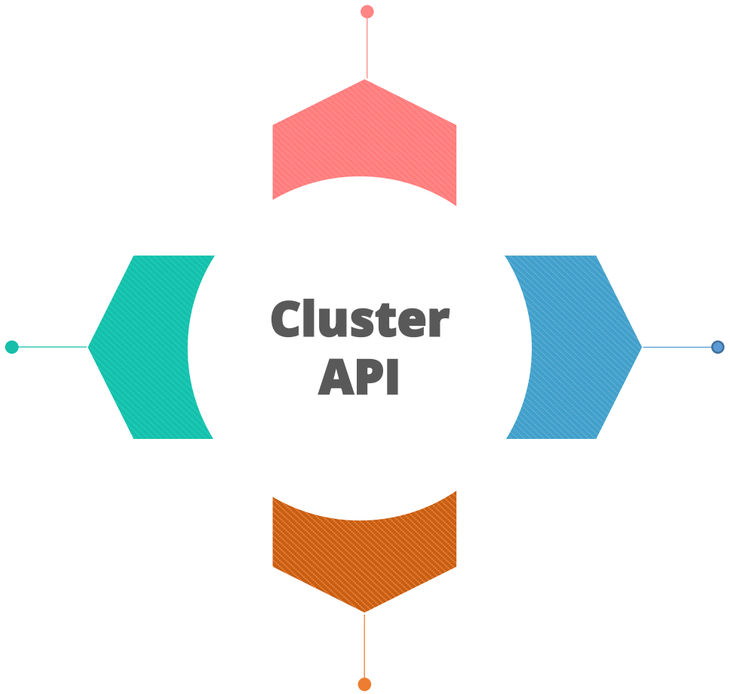
Working with Events and Buffers

Topic 2—Working with Cluster API

Working with Cluster API

Node.js is gaining more popularity as a server-side environment, primarily for high-traffic websites.

Node.js is single thread model and has 512MB memory limit for 32-bit systems and 1GB for 64-bit systems; however, lack of memory and processing power may create bottlenecks when multiple processes must be executed.



The most important features of Node.js is its scalability.

This is the main reason why big corporates with high volume traffic are integrating Node.js with their platform (e.g., Microsoft, Yahoo, Uber, and Walmart) or completely transferring their server-side operations to Node.js.

Working with Cluster API

FEATURES OF NODE.JS CLUSTER API

A cluster is a collection of similar instances (i.e., workers) running in the control of a parent Node.

All Worker instances can be spawned by implementing child_processes fork() method.

The Cluster module of node.js helps developers to create a small network of separate processes that can share server ports and give you access to utilize the full power of node.js-based server.

Node.js runs in a single thread. While it's still very fast in most cases, it really doesn't take advantage of multiple processors if they're available. The cluster module helps to achieve scalability.

Working with Cluster API

CLUSTER IMPLEMENTATION IN NODE.JS

Import cluster module.

```
var userCluster = require('cluster);
```

Cluster module executes the same process various times. This specifies which portions of the application are for the master process and which portions are for workers.

```
if(userCluster.isMaster) { ... }
```

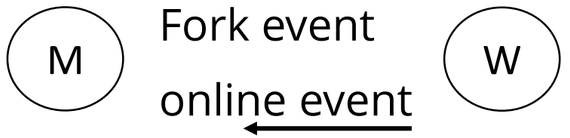
We initialize the master process, and this in turn initializes the child workers. To initiate a worker process within a master process, use the fork() method:

```
cluster.fork();
```

Working with Cluster API

THE CLUSTER EVENTS

- fork
- message
- online
- listening
- disconnect
- exit
- error



The difference between **'fork**' and **'online**' is that fork is emitted when the master forks a worker, and 'online' is emitted when the worker is running.

Working with Cluster API

EXAMPLE

```
var http = require("http");
var cluster = require("cluster");
var numCPUs = require("os").cpus().length;

if (cluster.isMaster) {
 cluster.on("fork", function (worker) {
  console.log("Attempting to fork worker");
 });
 cluster.on("online", function (worker) {
  console.log("Successfully forked worker");
 });
 for (var i = 0; i < numCPUs; i++) {
  cluster.fork();
 }
}
else {
 // implement worker code
}
```

Demo for Node.js Cluster API

Quiz

QUIZ 1

We can create child processes in Node applications.

a. True

b. False

QUIZ 1

We can create child processes in Node applications.

a. True

b. False

The correct answer is **a. True.**

We can create child processes in Node applications.

QUIZ 2

Which of the following modules is required to create a child process?

a. process module

b. child_process module

c. child module

d. web module

QUIZ 2

Which of the following modules is required to create a child process?

a. process module

b. child_process module

c. child module

d. web module

The correct answer is **b. child_process module.**

child_process module is required to create a child process.

QUIZ 3

___ method is the easiest one to develop a child process.

a. exec

b. createchild

c. cprocess

d. All of the above

QUIZ 3

___ method is the easiest one to develop a child process.

a. exec

b. createchild

c. cprocess

d. All of the above

The correct answer is **a. exec.**

exec method is the easiest one to develop a child process.

Key Takeaways

- Node.js comes pre-loaded with child_process module which has the three major ways to create a child process: Exec(), spawn(), and fork().
- The Cluster module of node.js helps developers to create a small network of separate processes that can share server ports. The modules give developers access to utilize the full power of node.js-based server.

Thank You