Name :- Dholey Drushti Ishwarbhai
Roll no :- 12
Subject :- 701 Date :- 19-07-'23

Que :- 1 Nodejs : Introduction , features , execution architecture

Introduction

- Nodejs is a server-side platform.
- it uses an open source , cross-platform run-time environment for developing server-side and networking applications.
- Nodejs applications are written in javascript, and can be run within the Nodejs run-time on OS X , Microsoft windows and Linux.
- Nodejs also provides a rich library of various Javascript modules which simplifies the development of web applications using Nodejs to a great extent.

Node.js = Runtime Environment + Js Library.

Features

following are some of the important features that make Node.js the first choice of software architects.

1. Asynchronous and Event Driven

All APIs of Nodejs library are async. that is, non-blocking.

it essentially means a Nodejs based server never waits for an API to return data.

The server moves to the next API after calling it and a notification mechanism of events of Node.js helps the server to get a response from the previous API call.

2. Very fast

Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.

3. Single Threaded but Highly Scalable.

Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which creates limited threads to handle requests.

4) No Buffering

Node.js applications never buffer any datas.

This applications simply output datas in chunks.

Execution Architecture

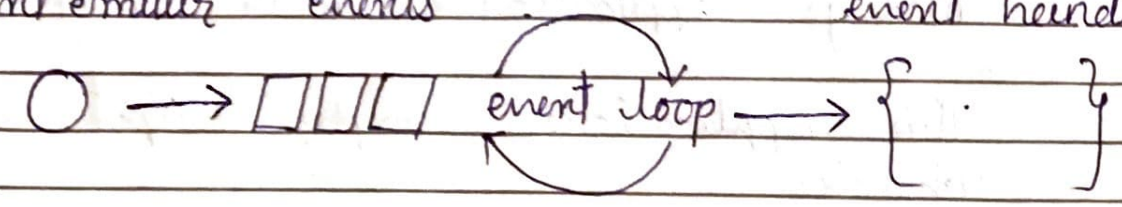
- Node.js follows an event-driven, single-threaded architecture.

- when a Node.js application starts, it initializes an event loop that continuously listens for events and triggers appropriate callbacks when an event occurs.

- The event loop enables Node.js to handle multiple concurrent connections efficiently.

- it uses an event driven programming model based on the EventEmitter class, where objects that emit events are known as eventEmitters.

- These events can be asynchronous op., such as file I/O, network requests, custom events triggered by the applications.



when an event occurs, Node.js executes the associated callback functions, allowing the application to respond to an event.

Since I/O operations in Node.js are non-blocking, the event loop can continue processing other events while waiting for I/O operations to complete.

To handle CPU-intensive tasks without blocking the event loop, Node.js provides a features called "child processes".

It allows spawning additional Node.js processes to offload heavy computations and leverage multiple CPU cores.

Que:- Note on module with example

- modules are a fundamental concept used to organize and share code.
- A module is essentially a reusable block of code that encapsulates related functionality.
- It can be a single file or a directory containing multiple files.
- Nodejs provides a built-in module system that allows you to create, import and use modules in your applications.
- This module system follows the CommonJS module specification, which is the standard used in Nodejs.

example : mathUtils.js

```
const add = (a, b) => {
    return a+b;
};
const substract = (a, b) => {
    return a-b;
};
module.exports = {
    add, substract
};
```

- in this file we exports to modules using module.exports object, which makes them accessible to other parts of our applications.

- in app.js file, we can import mathUtils.js

```
const mathUtils = require('./mathUtils');

const result1 = mathUtils.add(5, 3);
console.log(result1);

const result2 = mathUtils.substract(10, 7);
console.log(result2);
```

- The module system allows you to organize your code into reusable units, making it easier to manage and maintain large-scale Node.js applications.

Que:- Note on package with example.

- a package refers to a collection of reusable code, usually grouped together in a directory and distributed as a compressed archive files.

- Packages can contain modules, dependencies, configuration files, and other resources that make it easier to share and use code across different projects.

- The primary tool for managing packages in Node.js is npm, which comes bundled with Node.js installation. npm provides a vast registry of packages and offers commands to install, update and publish packages.

package.json

- The package.json file is a vital component of any Node.js package.

- it serves as a manifest file containing metadata about the package, its dependencies, scripts, and other configuration settings.

it allows you to define project-specific details and manage dependencies efficiently.

to install package :

npm install <package-name>

ex npm install express

use of package :

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
    res.send('Hello, world');
});

app.listen(3000, () => {
    console.log('server started on port 3000');
});
```

Que :- use of package.json and package-lock.json

- The package.json and package-lock.json files are essential components of Node.js projects
- They serve different purpose and play crucial roles in managing dependencies and ensuring consistent builds.

Package.json

- this file is a menifest file that contains metadata about your Node.js project, including its name, version, discription, author, scripts, dependencies, and more. It serves the following purposes.

Dependency Menegement

npm install

Script Definitions

npm run / start / test / build

Project Configuration.

Package-lock.json

- it is automatically generated by npm when installing or modifying package dependencies. It serves the following process:

Dependency locking

Dependency Resolution

Faster installation

Que:- npm - introduction and commands with use...

- npm stands for Node Package Manager.
- It is a package manager for the Node Javascript platform
- npm is known as the world's largest software registry.
- npm consist of three component
- The website allows you to find third-party packages, set up profiles and manage your packages.
- The command-line interface or npm CLI that runs from a terminal to allow you to interact with npm.
- large public database for JS code.

Commands

npm init - to initialize the project directory

npm install <package-name>

- used to install package

npm install <package-name> -g

- to install executable package globally.

npm update <package-name>

- to update any package in your project

npm reinstall <package-name>

- to reinstall package from project

npm ls

- to list out all the packages installed

npm search <package-name>

- to used to search the package.

npm install <package-name> --save.

Que:- Describe use and working of following Nodejs packages, important properties and methods and relevant programs.

1) url :

- provides a detailed explanation of the 'url' package in Nodejs, including its use, working, important properties, method and a relevant program.

- it allows you to parse, format, and manipulate URLs, making it easier to handle various URL-related tasks.

use :

- Parse URLs : Extract different components of a URL, such as the protocol, hostname, pathname, query parameters, etc.

- Format URLs : Convert an object with URL components into a formatted URL string.

- Manipulate URLs : Modify or combine parts of a URL to create new URLs.

working :

- The 'url' module works with URL strings and objects representing URL components.

- It uses the 'url.parse()' method to parse a URL string and return an object containing its components.

- Similarly, it uses the 'url.format()' method to format an object with URL component into a URL string.

Important properties and methods :

'url.parse(urlString[, parseQueryString[, slashesDenoteHost]])' :
parses a URL and returns an object containing its components.

urlString : The URL string to parse.

parseQueryString : (Optional) A boolean value indicating whether to parse the query string. Default is 'false'.

url.format(urlObject) : Formats an object with URL components into a URL string.

urlObject : An object with URL components like protocol, hostname, pathname, query, etc.

Relevant program :

```
const url = require('url');

const urlString = 'http://www.example.com/param1=value1&param2=value2';

const parseUrl = url.parse(urlString);

console.log(parseUrl);
```

eg process :

- built-in module which provides info. and control over the current Nodejs process.
- It allows you to access environment variables, command-line arguments, standard i/o streams, and communicate with the parent process.

use :

- access command-line arguments and environment variables
- communicate with the parent process through standard i/o streams.
- Handle process-related events like process termination.

working :

- it automatically available in every Nodejs app. It represents the current Nodejs process and provides various properties and methods to interact with it.

Important properties and methods

process.argv : an array containing the command-line arguments passed to the Node's process.

process.env : An object containing the environment variables.

Relevant program:

```
const args = process.argv.slice(2);
console.log('command-line arguments:', args);

const nodeEnv = process.env.NODE_ENV;
console.log('NODE_ENV', nodeEnv);
```

9) PM2 (External Package)

- external process manager for Node.js applications.

- provides features like process clustering, automatic restarts, and monitoring, making it well-suited for managing Node.js applications in production environments.

use:

- Run Node.js applications as background processes (daemons).
- automatically restart applications on crashes or failures.
- scale applications across multiple CPU cores to improve performance.

working:

- installed globally using npm
npm install pm2 -g
pm2 start app.js
pm2 monit

4) readline

- built-in module that provides an interface for reading input from a readable stream line by line.
- it allows you to interact with the user through prompts and get user input in a non-blocking manner.

use:

- create an interface for reading input from the user line by line.
- prompt the user for input and receive responses.
- Handle user input asynchronously without blocking the event loop.

working:

- creates an interface that reads input from the specified readable stream and triggers events when a line is received
- It provides methods to prompt the user and get input, and these operations are asynchronous, allowing other tasks to proceed while waiting for user input.

vision

Important properties and methods:

readline.createInterface(options):
create readline interface
options: an object containing options for the interface, typically specifying the input and output streams.

interface.question(query, callback):
asks a question and waits for user input

query: The question to be displayed to the user

callback: A function that will be called with the user's response as an argument

Relevant Program:

```
const readline = require('readline');
const rl = readline.createInterface({
    input: process.stdin;
    output: process.stdout });

rl.question('what is your name?', (name)=>{
    console.log(`Hello, ${name}!`);
    rl.close();
});
```

5) fs

- it provides file system related functionality, allowing you to read, write and manipulate files and directories.
- it enables you to interact with the file system on your machine and perform various file-related operations.

use:

- read files and directories
- write data to files.
- manipulate files and directories, such as renaming, deleting and moving.
- create and manage directories.
- work with file system streams for handling large files efficiently.

working:

- provides both synchronous and asynchronous methods for file system operations.
- synchronous method blocks the event loop until the operation is completed, while asynchronous methods are non-blocking and use callbacks or promises to handle results.

VISION

important properties & methods

fs.readfile (path [, options], callback)
path - path to the file

fs.writeFile (file, data [, options], callback)

Relevant program:

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

fs.readfile ('input.txt', 'utf-8', (err, data) => {
    if (err) {
        console.log (err);
        return;
    }
    Console.log ('File content '; data);

fs.writefile ('output.txt', data.toUpperCase(), 'utf8',
        (err) => {
        if (err) {
            console.error (err);
            return;
        }
        console.log ('Data written ');
    });
});
```

6) events

- provides an event-driven architecture and implements the EventEmitter class, allowing you to create and handle custom events.

properties & methods:

events.EventEmitter : allows you to work with event

emitter.on(eventName, listener) : adds a listener function

emitter.emit(eventName[, ...args]) : emits specified event

Relevant program:

```
const eventEmitter = require('events');

class MyEmitter extends EventEmitter();

const myEmitter = new MyEmitter();
myEmitter.on('greet', (name) => {
        console.log('Hello, ${name}!');
});

myEmitter.emit('greet', "Drishti");
```

7) console

use :

provides methods for writing to the standard output and standard error streams, making it useful for debugging and logging.

properties & methods :

console.log (message) : writes a message to the standard output

console.error (message) : writes an error message to standard error.

Relevant program :

```
Console.log (" Hello ! log message ");
console.error (" Hello ! error message ");
```

8) buffer

use :

provides a way to handle library / binary data, allowing you to manipulate and work with raw data.

properties & methods

Buffer.from (data) : creates a new buffer object from the provided data

buffer.toString ([encoding [, start [, end]]]) :
returns the string representation of a buffer.

Relevant program :

```
const buffer = Buffer.from ('Hello, world', 'utf8')

console.log (buffer.toString ('utf8'));
```

9) querystring

use :

provides utilities for working with URL query strings, parsing and formatting them.

properties & methods

querystring.parse(str[, sep[, eq[, options]]])

querystring.stringify(obj[, sep[, eq[, options]]])

Relevent program :

```
const qs = require('querystring');
const qS = 'param1=value1&param2=value2';
const parsedQ = querystring.parse(queryS);
console.log(parsedQ);
```

10) http

use :

provids HTTP server and client implementation, allowing you to build HTTP-based app.

properties & methods

http.createServer([requestListener])

http.get(options[, callback])

Relevant program:

```
const http = require('http');

const server = http.createServer((req, res) => {
    res.writeHead(200, {'content-Type': 'text/plain'});
    res.end('Hello world');
});

server.listen(3000);
```

11) V8

- provides access to V8 engine-specific features and information.
- javascript engine developed by google and used in Nodejs to execute JS code.

use:

- access v8 engine stastics and information
- inspect memory usage and garbage collection metrics.

working

allows you to interact with the underlying v8 engine through its API's and retrieve various performance-related information and metrics.

Important properties and methods.

v8.getHeapStatistics(): returns object containing total heap size, used heap size and heap limits

v8.getHeapSpaceStatistics():

Relevant program

```
const v8 = require('v8');
const heapStats = v8.getHeapStatistics();
console.log(heapStats);
```

vision

12) OS

- provides operating system related utilities. allowing you to access information about the underlying os.

use :

- retrieve info about os such as platform, architecture and hostname.
- get info about system's CPU, network interface, and memory
- Handle and manipulate file path.

Important properties & methods

os.platform()
os.arch()
os.hostname()

Relevent program

```
const os = require('os');

console.log(os.platform);

console.log("architecture", os.arch());

console.log("hostname", os.hostname());
```

14) zlib

- provides compression and decompression functionality using the zlib library.

use :

- Compress data using gzip, deflate or zlib algorithms
- decompress data that was previously compressed using gzip, deflate or zlib.

Important properties & methods :-

zlib.createGzip(options)
zlib.createGunzip(options)
zlib.createDeflate(options)
zlib.createInflate(options)
zlib.createDeflateRaw(options)
zlib.createInflateRaw(options)
zlib.createUnzip(options)

Relevant programs:

```
const zlib = require('zlib');

const dataToCompress = 'This is some data to
                        compress and decompress.';
```

```
zlib.gzip(dataToCompress, (err, compressedData) => {
    if (err) {
        console.error('Compression error:', err);
        return;
    }

    console.log('Compressed data:', compressedData);

    zlib.gunzip(compressedData, (err, decompressedData) => {
        if (err) {
            console.error('Decompression error:', err);
            return;
        }

        console.log('Decompressed data:',
                    decompressedData.toString());
    });
});
```