Node.js

Vraj Suratwala

Department of ICT, VNSGU, Surat

Node.js - Important Node Packages

URL

pm2

process

readline

fs

events

console

buffer

querystring

http

v8

os Module

URL

- The URL Module is the core module in Node.js that allows you to parse, contruct, normalize and encode the URL’s.
- you do not need to install it because it’s core module and it will because direct usable to you.
- Syntax:
  - CJS
    - const Data = require(‘url’);
  - ESM
    - import {url} from ‘url’;

URL - Methods() and Property

- url.parse(urlstr) : Parses a url string into component.
- url.format(urlobj) : format an object into url string.
- url.resolve(from,to) : Resolves a targets URL relative to a base.
- new URL(input,base) : Creates a URL objects with full features.

URL - Features

- 1. Built-in Modules
- 2. Useful for
- 3. Legacy vs New URL()
    - -url.parse() vs new URL()

pm2

- pm2 refers to advanced production grade process manager which is an External Package so we do not need to install it externaly.

- Key Features:
  - auto restart
  - Load Balancing
  - Log Management
  - Monitering
  - startup script

- How to Install it ?
- npm install -g pm2
  - Basic Commands
  - pm2 list
  - pm2 start app.js
  - pm2 save
  - pm2-startup
  - pm2-save
  - pm2-logs

Process

- The Process is the Global Object in Node that Provide the Information and Status od Current Process which is executing curruntly.

- Methods.

- Process.argv() - Command Line Arguments.
- Process.env - Environment Variables.
- Process.exit() - Exit the Process with Status Code.
- Process.pid() - Current PID.
- Process.cwd() - Current Working Directory.
- Process.MemoryUsage() - Memory Information.

Process - Example

- console.log(“Arg : ” , process.argv)

- How to run this?
- NODE_ENV = production node demo.js arg1 arg2

readline

- this readline module in node.js provides an interface to read the input which is give by user (like the terminal) and writing an output,also used to read static data.

- Usecase : Command Line Tool, INteractive Prompts, and Simple Input and Output.

- How to Import The readline module.
- const readline = require('readline');

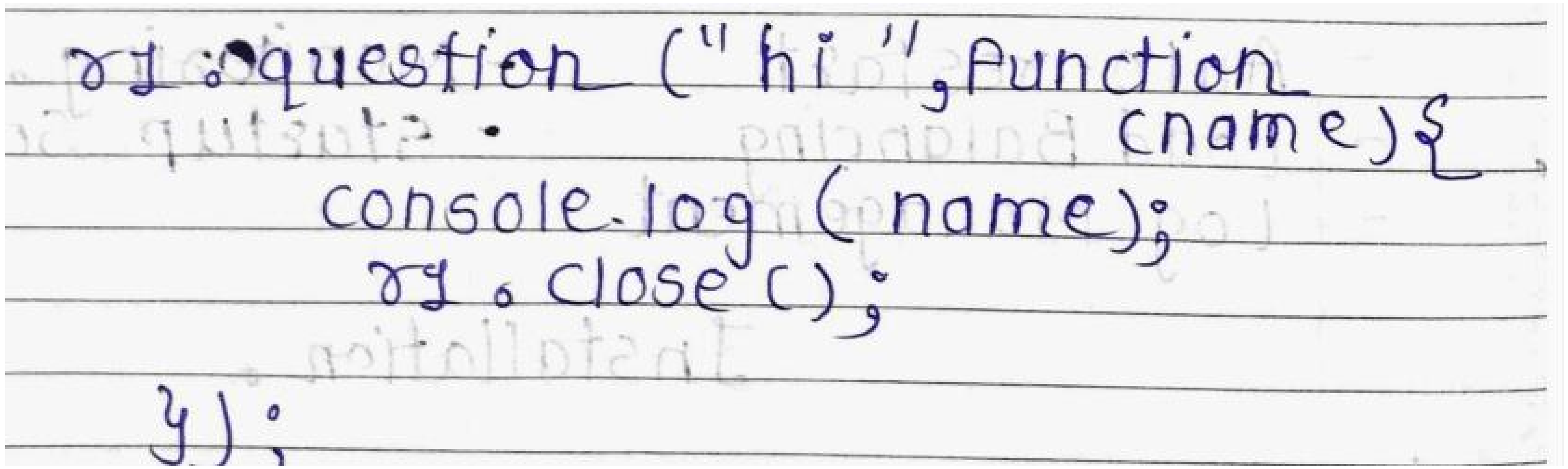
Example

- const r1 = readline.CreateInterface({

  input: process.stdin,

  output :process.srdout

- });



```
r1.question ("hi",function
                        (name){
      console.log(name);
       r1.close();

   });
```

Methods of readline module

- readline.createInterface()
- r1.question(query,callback())
- r1.close()
- r1.on(‘line’,callback())
- r1.on(‘close’,callback())

fs

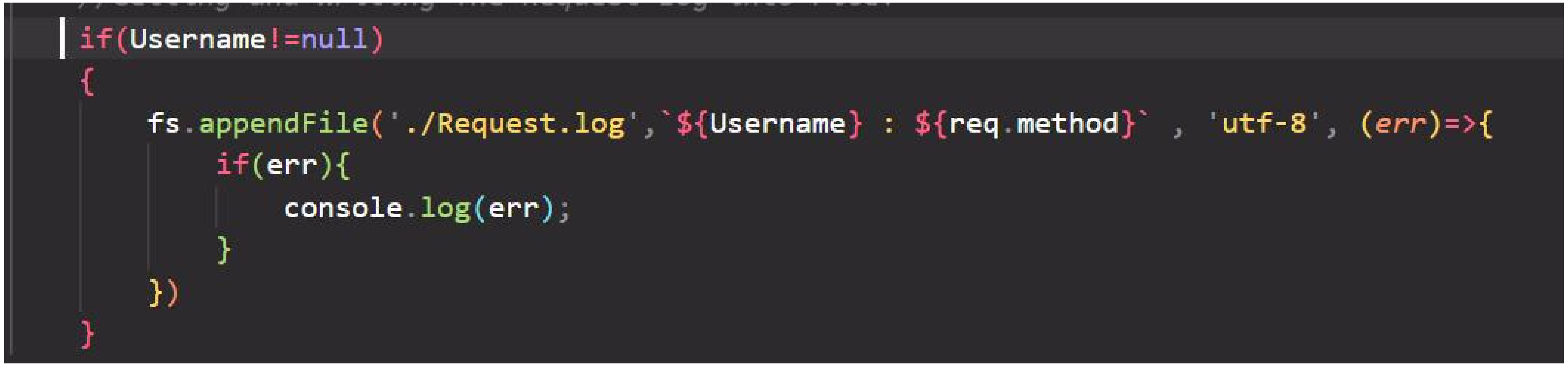
- again it is core module which comes with node.js so you do not need to install.
- the fs refers to file system on your computer you can can create,read,update and delete,rename files and directory and more.

- how to import ‘fs’?
  - const fs = require(‘fs’);

fs-Methods

- fs.readFile()
- fs.readFileSync()
- fs.writeFile()
- fs.appendFile()
- fs.unlink()
- fs.mkdir()
- fs.rmdir()
- fs.start()

Example :



```
if(Username!=null)
{
    fs.appendFile('./Request.log',`${Username} : ${req.method}` , 'utf-8', (err)=>{
        if(err){
            console.log(err);
        }
    })
}
```

fs - Features

- 1. Core Module
- 2. File handling
- 3. Use case
- 4. API TYPE
  - Sync.
  - Async.

  - So this is the information about fs module.

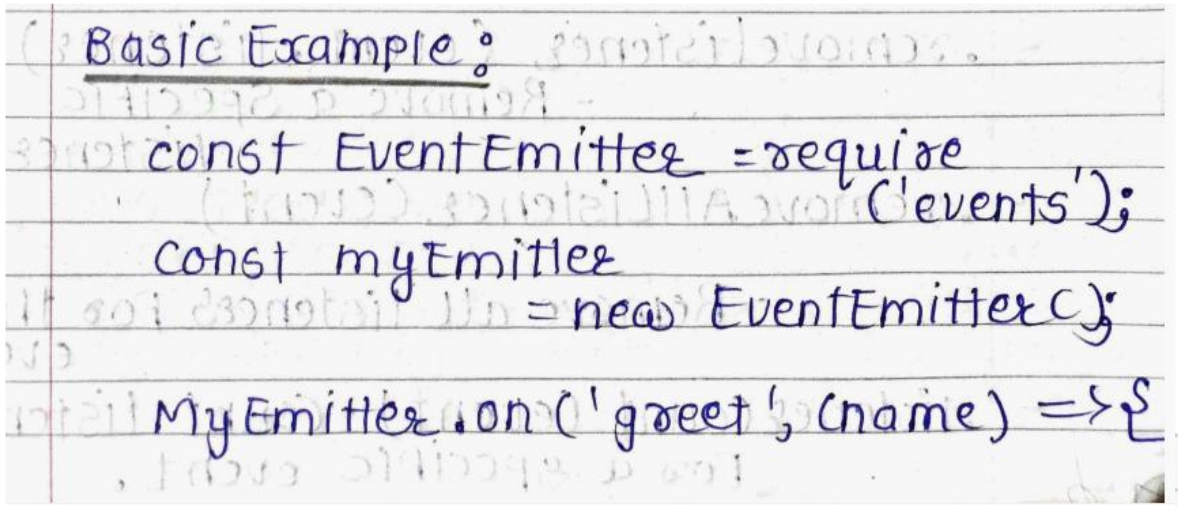
Events

- The events modules in Node.js allows you to create ,listen for and handle custom events.
- Events Based on The
  - Observer Design Pattern
  - Core Module

- Why use the Events Module ?
  - Event Driven Architecture
  - Hepls to decouple Components
  - Server, user action, logging etc.

How to Import it and use it ?

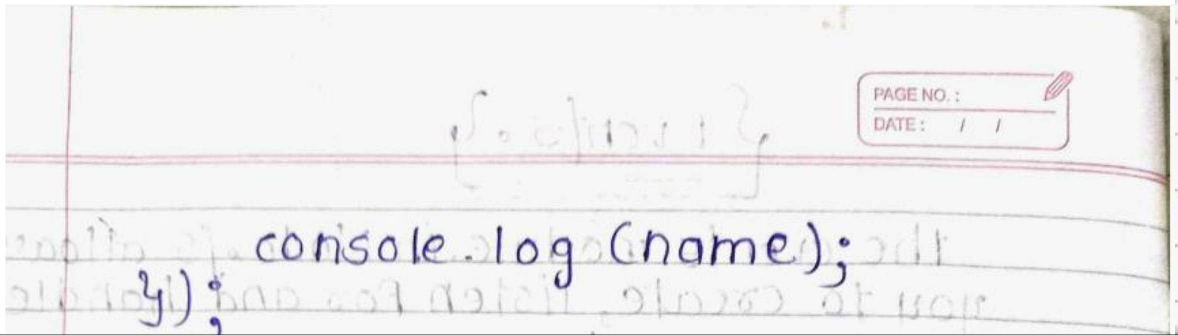
- importing the event module.
  - const eventEmitter = require('events');



Basic Example:

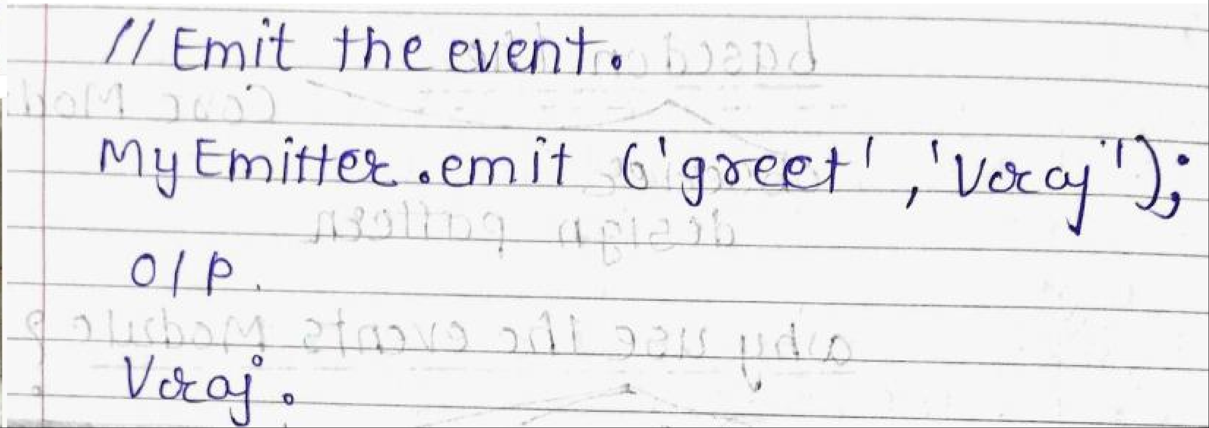
```
const EventEmitter = require
                       ('events');
const myEmitter
        = new EventEmitter();

MyEmitter.on ('greet', (name) => {
```



PAGE NO.:
DATE: / /

```
    console.log (name);
});
```



```
// Emit the event.

MyEmitter.emit ('greet', 'Vraj');

O/P.
Vraj.
```

Common EventEmmitter Event

- .on(event,listener) : Register an event ! (Multiple Allowed)
- .once(event,listener) : Register a one time listener.
- .emit(event, [args]) : Trigger an Event.
- .removeListener (event,listener) : Remove a Specific Listener.
- .removeAllListener (event) : Remove all listeners for all the events.
- .listenerCount(event) : Count Listeners for a Specific events.

Real world use cases.

- Logging system
- HTTP Server.
- File Processing.
- Chat App etc...

console

- We can say that the console is providing a simple debugging and logging interface.
- again it is global module so we do not need to include it.

- Why should we use console.
  - print messages
  - log output
  - display errors and warning

console

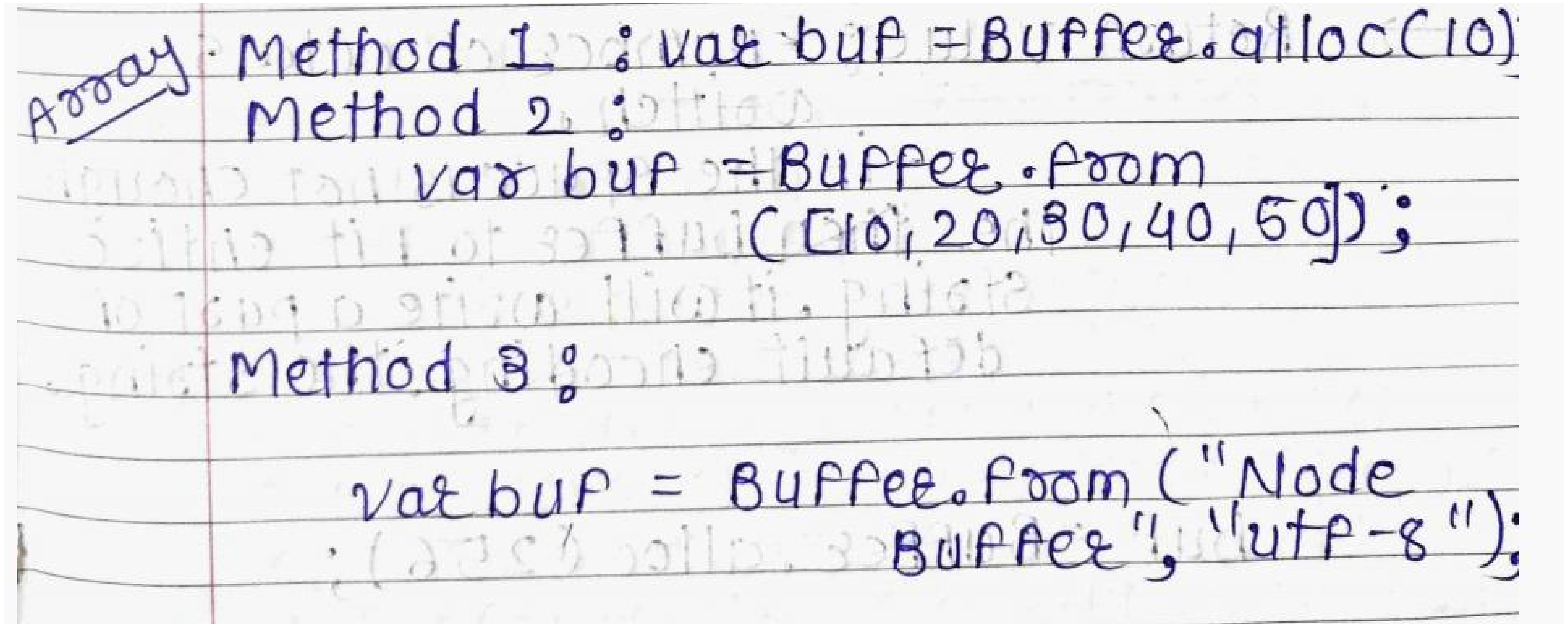
- if the destination is a file then that time it will use Sync. use.
- and in the case Async. way it will use pipe.

- Common console Methods.
  - console.log()
  - console.error()
  - console.info()
  - console.warn()
  - console.debug()
  - console.dir()
  - console.table()
  - console.time()
  - console.timeEnd()
  - console.assert()

Buffers

- pure js is unicode friendly rathar then Binary data!
- while dealing with file systems , it’s necessary to handle the Octal Streams too.
- Node Provides buffer class which provides instances to store raw data similar to an array of int but corresponds to a raw memory allocation outside the v8 heap!

Creation of Buffers



Array

Method 1 : var buf = Buffer.alloc(10)

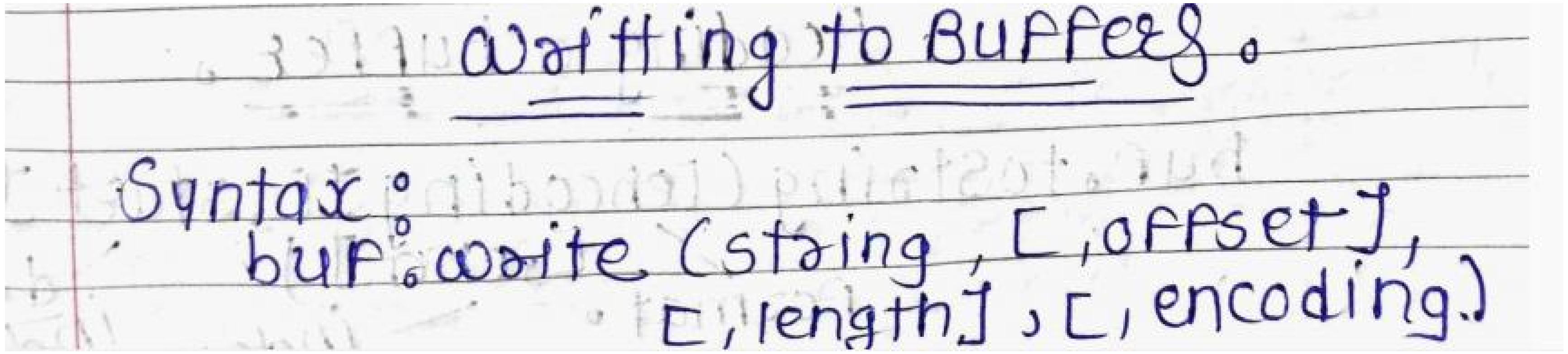
Method 2 :

```
var buf = Buffer.from
        ([10, 20, 30, 40, 50]);
```

Method 3 :

```
var buf = Buffer.from ("Node
          Buffer", "utf-8");
```

Writing to buffers



Writting to Buffers.

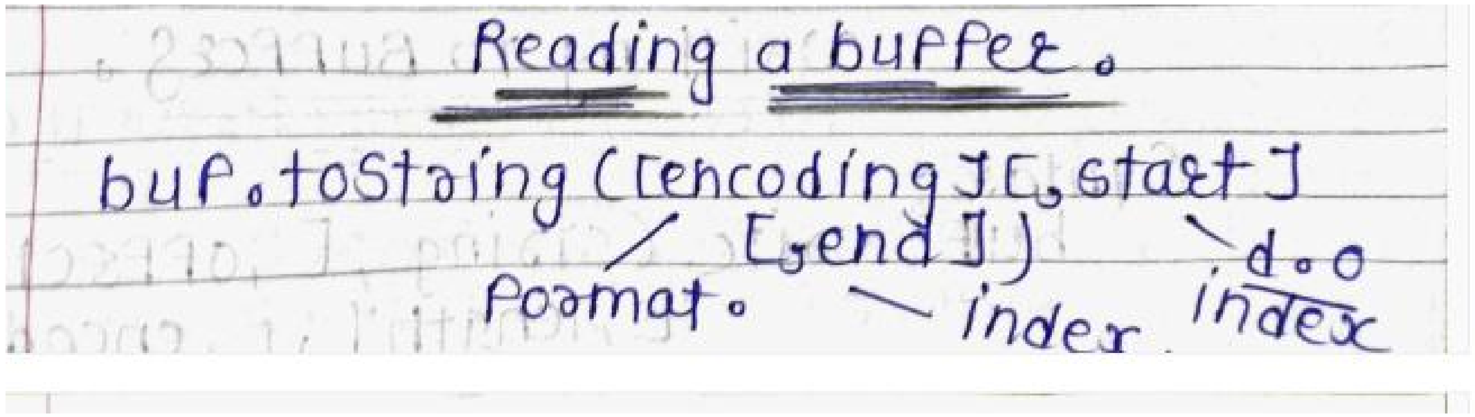
Syntax:
buf.write (string , [,offset],
[,length] , [, encoding.)

Parameters

- 1. String : the string data to be written to buffer.
- 2. offset : this is the index of the buffer to start writing at.
  - default value is 0.
- 3.length : This is number of bytes to write.
- 4.Encoding : Encoding to use.
  - ‘utf-8 is by default encoding!’

Return Value

- number of octals return.
- if the space is not enough the buffer to fit entire string.
- it will write the part of string.



Reading a buffer.

buf.tostring([encoding][, start] [, end])

format. index. d.o index

Operation with buffers.

Convert Buffer to JSON:

```
var buff = Buffer.from('Node Buffer');
var json = buf.toJSON(buf);
console.log(json);
```

Concatenate Buffers:

```
Buffer.concat(list, [, totalLength])
```

list → ArrayList of Buffer objects to be concatenate.

totalLength → while concatenate.

```
var buff1 = Buffer.from('buffer1');
var buff2 = Buffer.from('buffer2');
var buffer3 = buffer.concat([buff1, buff2]);
console.log(buffer3.tostring());
```

```
Buffer.isEncoding(encoding);
Buffer.isBuffer(obj);
```

→ Copy : buf.compare(otherBuffer);

→ Compare : buf.copy(targetBuffer [, targetStart], [, sourceStart], [, sourceEnd])

Slice : buf.slice([start], [, end])

Length : buf.length;

byteLegth(string, [encoding]) :

↓ gives the Actual byte length of a string.

encoding defaults to 'utf-8'.

querystring

- again it is an core module that we do not need to install.
- used to parse and format url query string.

- why to use ‘querystring’ ?
  - to convert a query string to js object and vise-versa.
  - useful for to hadle the form with different methods.
- Importing this module :
- const qs = require(‘querystring’);

Common Methods

- querystring.parse - converts query string into an object.
- querystring.stringify(obj) - Converts Object into query string.
- queryString.eacape() - Esacape a string for use in a query.
- querystring.unscape() - unscapes a query string components.

The Difference Between Querystring vs URL Search Pattern

querystring	URLsearchpattern
legacy	Modern
Simeple and Widly Used	Recommanded in new Apps
Traditional way	New way

Http

- Again it is a core module.
- allows us to create HTTP servers and handle HTTP REQ. and RES.
- build servers without frameworks.
- handle routes, headers, status code and data manually.

- Example : const http = require(‘http’);
- const server = http.createServer((req,res) =>{
    - console.log(‘Hello World!’);
    - });

    - server.listen(3000, ()=>{//code});

Summary - http

- 1. Required Module
- 2. Create Server
- 3. Testing Request and Response

v8

- it is again core module.
- an interface to the v8 js engine, which runs JS code inside node.js.
- developed by google.
- Compiled js directory to native machine code.
- execute js at server side.

Common Methods - v8

- v8.getHeapstatistics() - return memory usage statistics of the v heap.
- v8.getHeapSpaceStatistics() - Returns Detailed State about memory Spaces.
- v8.serialize(value) - Converts js to buffer(binary).
- v8.deserialize(buffer) - Converts a serializeed buffer back to JS.

Usecase - v8

- Performance Monitering.
- Debugging Memory Leaks.
- Data serialization
- De-serialization
- Internal Tooling.

os - module

- provide a way to access an operating system required related information like the
    - platform, CPU Architecture, Memory Usage, and more.

- again it’s a core module.
- Importing a module :
    - const os = required(‘os’);

os - Common Methods

- os.platform()
- os.type()
- os.arch()
- os.cpus()
- os.hostname()
- os.freemem()
- os.totalmem()
- os.uptime()
- os.userInfo()
- os.homedir()

Real time use cases - os

- 1. System Monitering
- 2. CLI Tools
- 3. Server Scripts

Thank you

Vraj Suratwala - Department of ICT, VNSGU, Surat