Ganpat University

॥ विद्यया समाजोत्कर्षः ॥

#StudentsFirst
#CharacterMust

(Software Packages)

By

Prof. Hiteshri Modi

Node Modules

- Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

- Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope. Also, each module can be placed in a separate .js file under a separate folder.

Node.js Module Types

1. Core Modules
2. Local Modules
3. Third Party Modules

Node.js Core Modules

- The core modules include bare minimum functionalities of Node.js.
- These core modules are compiled into its binary distribution and load automatically when Node.js process starts.
- However, you need to import the core module first in order to use it in your application.

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods, and events to work with file I/O.
util	util module includes utility functions useful for programmers.

Loading Core Modules

- need to import it using require() function as shown below.

  var module = require('modulename');
- Example:
  - to use Node.js http module to create a web server.

```
var http = require('http');
var server = http.createServer(function(req, res)
{
 //write code here
});
 server.listen(5000);
```

Node.js Local Module

- Local modules are modules created locally in your Node.js application.
- These modules include different functionalities of your application in separate files and folders.
- You can also package it and distribute it via NPM, so that Node.js community can use it.
- For example, if you need to connect to MongoDB and fetch data then you can create a module for it, which can be reused in your application.

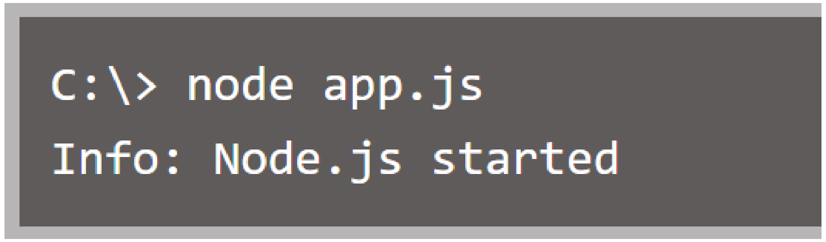
Writing Simple Module

- write simple logging module which logs the information, warning or error to the console.

```
//log.js file

var log = {
          info: function (info) {
             console.log('Info: ' + info);
          },
          warning:function (warning) {
             console.log('Warning: ' + warning);
          },
          error:function (error) {
             console.log('Error: ' + error);
          }
   };

module.exports = log
```

Loading Local Module

```
//app.js

var myLogModule = require('./Log.js');

myLogModule.info('Node.js started');
```



```
C:\> node app.js
Info: Node.js started
```

Export Module in Node.js

- The **module.exports** is a special object which is included in every JavaScript file in the Node.js application by default.
- The module is a variable that represents the current module, and exports is an object that will be exposed as a module.
- So, whatever you assign to module.exports will be exposed as a module.

Node.js Third Party Module

- Modules are available online and they are installed using the Node Package Manager (NPM) are called third party modules.
- **Examples:** express, mongoose, angular, react etc.

Node Package Manager (NPM)

- **Node Package Manager (NPM)** is a library and registry for JavaScript software packages. NPM has command-line tools to help install the different packages and manage their dependencies.
- Official website: https://www.npmjs.com
- In September 2022 over 2.1 million packages were reported being listed in the npm registry, making it the biggest single language code repository.
- NPM helps:
  - Install a Node.js module
  - Extend a Node.js module
  - Publish a Node.js module to repositories like Github

Node Package Manager (NPM)

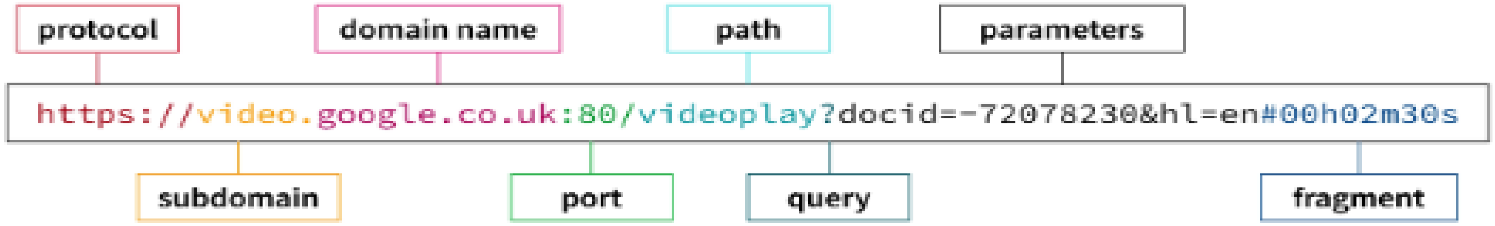
- npm install -g module_name // Install Globally
- npm install --save module_name //Install and save in package.json

Example:

- npm install --save express //Install express module
- npm install --save express mongoos //Install multiple modules at once

Core Modules: URL

- URL module provides utilities for URL resolution and parsing.


protocol
domain name
path
parameters
https://video.google.co.uk:80/videoplay?docid=-72078230&hl=en#00h02m30s
subdomain
port
query
fragment

- [illegible] to
  URL components.

Syntax: url.parse(urlString, parseQueryString);

Core Modules: URL

The **url.parse()** method takes a URL string, parses it and return URL object with each part of the address as properties of URL object.

Syntax:url.parse(urlString, parseQueryString, slashesDenoteHost)

Return Value: The url.parse() method returns an object with each part of the address like protocol, hostname, pathname, query, href

Core Modules: URL

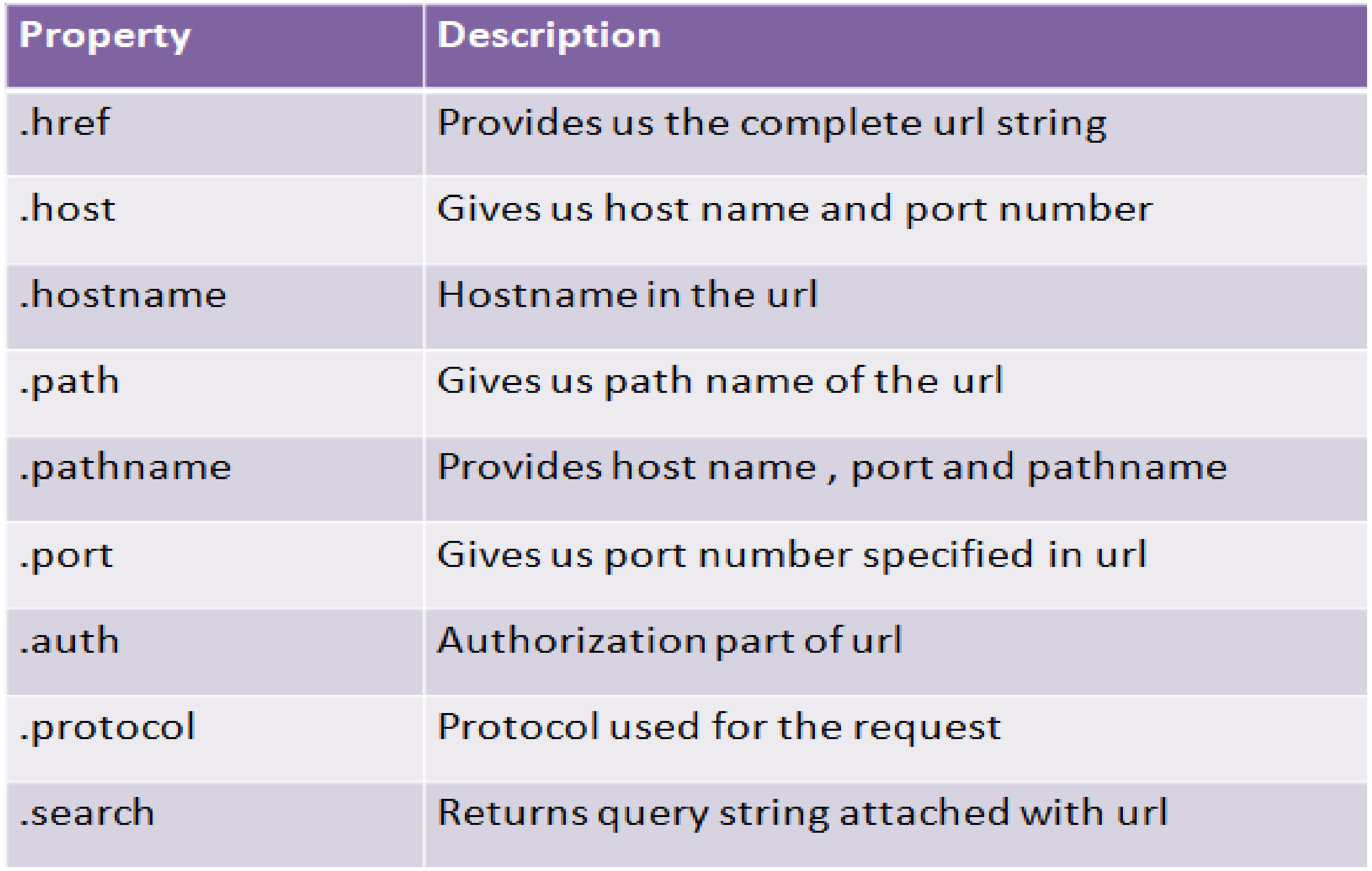
Parameters: This method accepts three parameters:

1) **urlString:** It holds the URL string which need to parse.
2) **parseQueryString:** It is a boolean value. If it set to true then the query property will be set to an object returned by the querystring module’s parse() method. If it set to false then the query property on the returned URL object will be an unparsed, undecoded string. Its **default value is false.**
3) **slashesDenoteHost:** It is a boolean value. If it set to true then the first token after the literal string // and preceding the next / will be interpreted as the host.
   eg https://www.ganpatuniversity.ac.in/exams/cbcs-regulations result {host: ‘www.ganpatuniversity.ac.in, pathname: ‘/exams/cbcs-regulations’} rather than {pathname: ‘//www.ganpatuniversity.ac.in/exams/cbcs-regulations’}.
   Its **default value is false.**

```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2017&month=february'

var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
console.log(qdata.month); //returns 'february'
```

```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';
var q = url.parse(adr, true);

console.log(q.href);
console.log(q.host); //returns 'localhost:8080'
console.log(q.hostname);
console.log(q.path);
console.log(q.pathname); //returns '/default.htm'
console.log(q.port);
console.log(q.auth);
console.log(q.protocol);
console.log(q.search); //returns '?year=2017&month=february'
var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
console.log(qdata.month); //returns 'february'

var adr = 'http://localhost:8080/#hiteshri';
var q = url.parse(adr, true);
console.log(q.hash);
```

Core Modules: URL



Property	Description
.href	Provides us the complete url string
.host	Gives us host name and port number
.hostname	Hostname in the url
.path	Gives us path name of the url
.pathname	Provides host name , port and pathname
.port	Gives us port number specified in url
.auth	Authorization part of url
.protocol	Protocol used for the request
.search	Returns query string attached with url

Core Modules: URL

Example:

```
var http = require('http');
var url = require('url');

http.createServer(function (req, res) {
    var components = url.parse(req.url, true);
    console.log(components);
}).listen(4200);
```

Core Modules: Fs

- FS (File System) in-built module to handle file operations like creating, reading, writing, deleting, renaming file.

- It gives the functionality of file I/O by providing wrappers around the standard POSIX(IEEE STANDARD) functions.

- All file system operations can have synchronous and asynchronous forms depending upon user requirements.

Core Modules: Fs

Synchronous approach: called **blocking functions**

- It waits for each operation to complete, only after that, it executes the next operation, hence blocking the next command from execution

Asynchronous approach: called **non-blocking functions**

- It never waits for each operation to complete, rather it executes all operations in the first go itself. The result of each operation will be handled once the result is available

- If your operations are not doing very heavy lifting like querying huge data from DB then go ahead with Synchronous way otherwise Asynchronous way.

- In an Asynchronous way, you can show some progress indicator to the user while in the background you can continue with your heavyweight works. This is an ideal scenario for GUI based apps.

readFile()	**readFileSync()**
Non-Blocking process	Blocking process
Uses callback	uses variable to store content
Asynchronous nature	Synchronous nature

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

fs.readFile('file.txt','utf8', function (err, data) {
    if (err)
        throw err;
    console.log("\nasync read : file.txt : \n" + data);
});

const data = fs.readFileSync('file1.txt','utf8');
console.log("\nsync read : file1.txt : \n"+ data);
```

Core Modules: path

- Node.js provides path module which allows to interact with file paths easily.

- The path module has many useful properties and methods to access and manipulate paths in the file system.

Properties:

- path.sep - represents the platform-specific path separator.
  It returns \ on Windows and / on Linux and macOS.

- path.delimiter - represents the path delimiter.
  It returns ; on Windows and : on Linux and macOS.

Method	Description
basename()	Returns the last part of a path
delimiter	Returns the delimiter specified for the platform
dirname()	Returns the directories of a path
extname()	Returns the file extension of a path
format()	Formats a path object into a path string
isAbsolute()	Returns true if a path is an absolute path, otherwise false
join()	Joins the specified paths into one
normalize()	Normalizes the specified path
parse()	Formats a path string into a path object
posix	Returns an object containing POSIX specific properties and methods
relative()	Returns the relative path from one specified path to another specified path
resolve()	Resolves the specified paths into an absolute path
sep	Returns the segment separator specified for the platform
win32	Returns an object containing Windows specific properties and methods

Core Modules: path

Methods:

path.basename(path, [,ext])
path.dirname(path)
path.extname(path)
path.format(pathObj)
path.isAbsolute(path)
path.join(...path)
path.normalize(path)
path.parse(path)
path.relative(from, to)
path.resolve(...path)

Core Modules: util

- The node.js "util" module provides some functions to print formatted strings as well as some 'utility' functions are helpful for debugging purposes.

- Use *require('util')* to access these functions.

Core Modules: util

Methods:

util.format(format, [...])
util.debug(string)
util.error([...])
util.puts([...])
util.print([...])
util.log(string)
util.inspect(object, [options])
Customizing util.inspect colors
util.isArray(object)
util.isRegExp(object)
util.isDate(object)
util.isError(object)
util.inherits(constructor, superConstructor)

Core Modules: OS

- OS module in Node.js establishes interaction between the operating system and the application.
- The OS module consists of certain properties and methods which help in performing system-related activities.
- The OS module provides several information about the Operating System to Node.js like hostname, free (unused) memory, total memory, operating system name, user information, etc.

Core Modules: OS

Properties:

- **os.EOL -** end of the line marker
- **os.constants -** constants for process signals, error codes

Methods:

- **os.type() -** name of the operating system
- **os.arch() -** architecture of the operating system
- **os.platform() –** operating system platform name
- **os.cpus()** - information of the CPU like model, speed

Core Modules: OS

Methods:

- **os.userInfo() -** information about the current user
- **os.uptime() -** system uptime in seconds
- **os.hostname() -** hostname of the operating system
- **os.totalmem() -** total memory in bytes
- **os.freemem()** - unused memory in the hard disk of the operating system in bytes
- **os.networkInterfaces() -** information about the network interfaces like IP address, MAC address, netmask, etc..