VENTURESITY
VIT

Node JS Cheat Sheet

About Node JS

Node.JS is evented I/O for V8 JavaScript. It is asynchronous in nature,
with handlers to I/O and other events being function callbacks. It is particularly suited to distributed computing environments with high concurrency.

- node script.js ⟶ Run script
- npm install ⟶ Install packages with npm

Global Objects

In browsers, the top-level scope is the global scope.That means that in browsers if you're in the global scope var something will define a global variable.In Node this is different. The top-level scope is not the global scope; var something inside a Node module will be local to that module.

__filename; ⟶The filename of the code being executed. (absolute path)

__dirname; ⟶The name of the directory that the currently executing script resides in. (absolute path)

module; ⟶A reference to the current module. In particular module.exports is used for defining what a module exports and makes available through require().

exports; ⟶A reference to the module.exports that is shorter to type.

process; ⟶The process object is a global object and can be accessed from anywhere. It is an instance of EventEmitter.

Buffer; ⟶The Buffer class is a global type for dealing with binary data directly.

Console

console.log([data], [...]); ⟶Prints to stdout with newline.

console.info([data], [...]); ⟶Same as console.log.

console.error([data], [...]); ⟶Same as console.log but prints to stderr.

console.warn([data], [...]); ⟶Same as console.error.

console.dir(obj); ⟶Uses util.inspect on obj and prints resulting string to stdout.

console.time(label); ⟶Mark a time.

Modules

var module = require('./module.js'); ⟶Loads the module module.js in the same directory.

module.require('./another_module.js'); ⟶load another_module as if require() was called from the module itself.

for eg- exports.area = function (r) {
    return Math.PI * r * r;
};

Process

process.on(SIGNAL, callback) ⟶Signal events emitted when process receives a signal
exit ⟶Process is about to exit
uncaughtException ⟶Exception bubbled back to event loop
Properties: process.stdout ⟶A writable stream to stdout.
processs.stderr ⟶A writable stream to stderr.
process.stdin ⟶A writable stream to stdin.
process.argv ⟶An array containing the command line arguments.
process.env⟶An object containing the user environment

Node JS Cheat Sheet

File System

Use require('fs') to use this module.All the methods have asynchronous and synchronous forms.

fs.readFile(fileName [,options], callback)——→Reads existing file.
fs.writeFile(filename, data[, options], callback)——→Writes to the file. If file exists then overwrite the content otherwise creates new file.
fs.open(path, flags[, mode], callback)——→Opens file for reading or writing.
fs.rename(oldPath, newPath, callback) ——→Renames an existing file.
fs.mkdir(path[, mode], callback)——→Creates a new directory.
fs.readdir(path, callback)——→Reads the content of the specified directory.
fs.exists(path, callback) ——→Determines whether the specified file exists or not.
fs.appendFile(file, data[, options], callback)——→Appends new content to the existing file

Path

Use require('fs') to use this module.All the methods have asynchronous and synchronous forms.

fs.readFile(fileName [,options], callback)——→Reads existing file.
fs.writeFile(filename, data[, options], callback)——→Writes to the file. If file exists then overwrite the content otherwise creates new file.
fs.open(path, flags[, mode], callback)——→Opens file for reading or writing.
fs.rename(oldPath, newPath, callback) ——→Renames an existing file.
fs.mkdir(path[, mode], callback)——→Creates a new directory.
fs.readdir(path, callback)——→Reads the content of the specified directory.
fs.exists(path, callback) ——→Determines whether the specified file exists or not.
fs.appendFile(file, data[, options], callback)——→Appends new content to the existing file

HTTP

To use the HTTP server and client one must require('http').

http.STATUS_CODES; ——→A collection of all the standard HTTP response status codes, and the short description of each.
http.request(options, [callback]); ——→This function allows one to transparently issue requests.
http.get(options, [callback]); ——→Set the method to GET and calls req.end() automatically.
server = http.createServer([requestListener]); // Returns a new web server object. The requestListener is a function which is automatically added to the 'request' event.
server.listen(path, [callback]); ——→Start a UNIX socket server listening for connections on the given path.
server.setTimeout(msecs, callback); ——→Sets the timeout value for sockets, and emits a 'timeout' event on the Server object, passing the socket as an argument, if a timeout occurs.

server.on('request', function (request, response) { }); ——→Emitted each time there is a request.
server.on('connection', function (socket) { }); ——→When a new TCP stream is established.
server.on('close', function () { }); ——→Emitted when the server closes.
request.write(chunk, [encoding]); ——→Sends a chunk of the body.
request.end([data], [encoding]); ——→Finishes sending the request. If any parts of the body are unsent, it will flush them to the stream.
request.abort(); ——→Aborts a request.
response.write(chunk, [encoding]);——→This sends a chunk of the response body.
response.writeHead(statusCode, [reasonPhrase], [headers]); ——→Sends a response header to the request.
response.setHeader(name, value); ——→Sets a single header value for implicit headers. If this header already exists in the to-be-sent headers, its value will be replaced. Use an array of strings here if you need to send multiple headers with the same name.