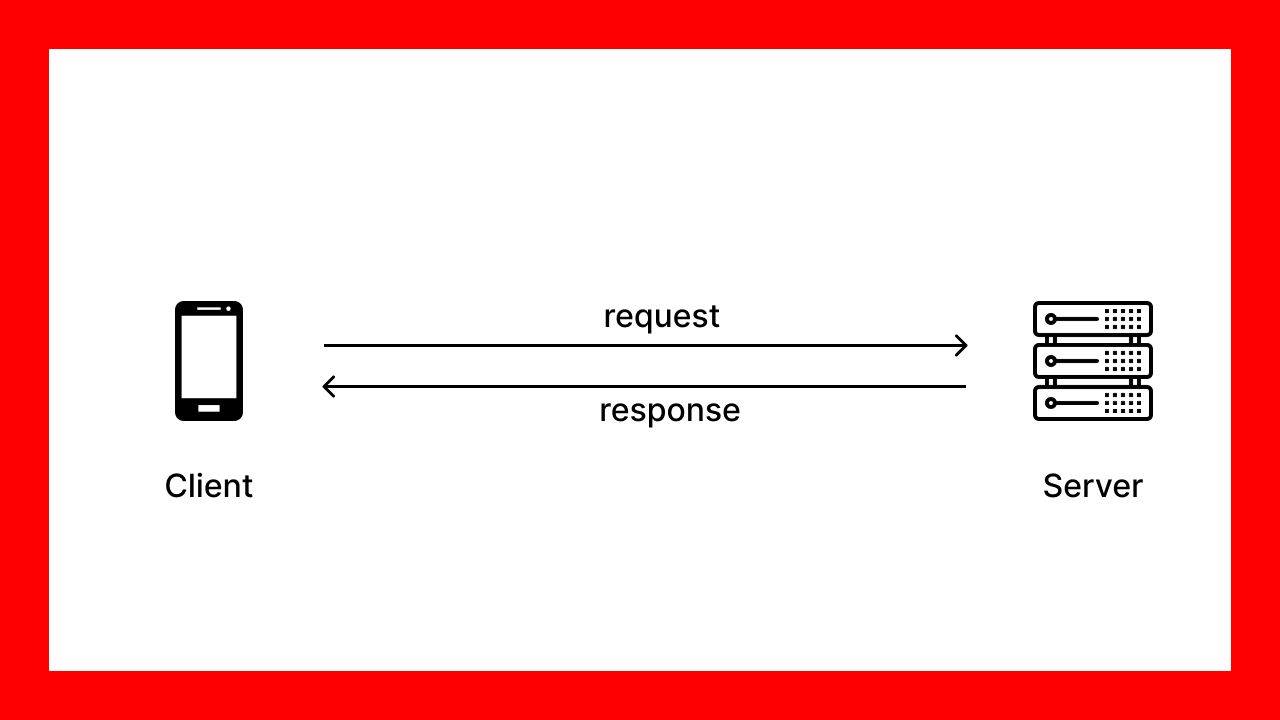
**1. HTTP (Hypertext Transfer Protocol)**

* Computers connected to the internet are called clients and servers
* Clients are internet-connected devices such as computers or mobiles phones along with web-accessing software available on those devices such as a web browser.
* Servers on the other hand are computers that store web pages, sites, or apps.
* Clients make requests for the specific resource to the server, server process the request and send the response according to it.



* A protocol (Set of Rules) that defines a format for clients and servers to speak to each other.
* The clients sends the HTTP request and the server responds with the HTTP response.

**HTTP and Node**

* Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).
* We can create a web server using node.js
* nodde.js has access to operating system functionality like networking.
* Node has an event loop to run tasks asynchronously and is perfect for creating web servers that can simultaneously handle large volumes of requests.
* The node server we create should still respect the HTTP format.
* The HTTP module allows creation of web servers that can Transfer data over HTTP.

**How to use http module**

1. import the module

const http = require("http");

2. create server using method of http module

This method accept the callback function with two arguments as request and response

const server = http.createServer((req, res) => {

………….

………….

});

3. listen method is use to start to listen the request of specific port number as specified.

server.listen(3000);

4. using the response methods within the createServer method for sending the response.

4.1 writeHead method: This method can have two arguments first is the status code and other argument is content-type

res.writeHead(200,{'content-type':'text/html'});

4.2 write method: This method can have argument (messege or data ) you want to write.

res.write("Hello");

4.3 end method : you must have to write end method when you want to end the server response. or else it will be response loop and will not start server next time on same port.

res.end();

**Final Example**

const http = require("http");

const server = http.createServer((req, res) => {

res.writeHead(200,{'content-type':'text/html'});

res.write("Hello");

res.end();

});

server.listen(3000);

**2. Path Module**

Node.js Path

* The Node.js path module is used to handle and transform files paths. This module can be imported by using the following syntax:

Syntax:

var path =  require ("path")

**Node.js Path Methods**

Let's see the list of methods used in path module:

|  |  |  |
| --- | --- | --- |
| Index | Method | Description |
| 1. | path.normalize(p) | It is used to normalize a string path, taking care of '..' and '.' parts. |
| 2. | path.join([path1][, path2][, ...]) | It is used to join all arguments together and normalize the resulting path. |
| 3. | path.resolve([from ...], to) | It is used to resolve an absolute path. |
| 4. | path.isabsolute(path) | It determines whether path is an absolute path. an absolute path will always resolve to the same location, regardless of the working directory. |
| 5. | path.relative(from, to) | It is used to solve the relative path from "from" to "to". |
| 6. | path.dirname(p) | It return the directory name of a path. It is similar to the unix dirname command |
| 7. | path.basename(p[, ext]) | It returns the last portion of a path. It is similar to the Unix basename command. |
| 8. | path.extname(p) | It returns the extension of the path, from the last '.' to end of string in the last portion of the path. if there is no '.' in the last portion of the path or the first character of it is '.', then it returns an empty string. |
| 9. | path.parse(pathstring) | It returns an object from a path string. |
| 10. | path.format(pathobject) | It returns a path string from an object, the opposite of path.parse above. |

**Node.js Path Example**

const path = require("path");

console.log(\_\_filename);

console.log(\_\_dirname);

console.log(path.basename(\_\_filename));

console.log(path.basename(\_\_dirname));

console.log(path.extname(\_\_filename));

console.log(path.parse(\_\_filename));

console.log(path.format(path.parse(\_\_filename)));

console.log(path.isAbsolute(\_\_filename));

console.log(path.parse("./data.json"));

console.log(path.join("folder1", "folder2", "index.html"));

console.log(path.join(\_\_dirname, "data.json"));

console.log(path.resolve("folder1", "folder2", "index.html"));

**3. File System module**

Difference between Asynchronous and Synchronous methods:

|  |  |  |
| --- | --- | --- |
| Sr.no | Synchronous methods | Asynchronous methods |
| 1. | Synchronous functions are called blocking functions | Asynchronous functions are called non-blocking functions. |
| 2. | It blocks the execution of the program until the file operation has finished processing. | It does not block the execution of the program. |
| 3. | These functions take File Descriptor as the last argument. | These functions take a callback function as the last argument. |
| 4. | Examples: fs.readFileSync(), fs.appendFileSync(), fs.writeFileSync() etc. | Examples: fs.readFile(), fs.appendFile(), fs.writeFile(), fs.stat() etc. |

**Synchronous methods for file system**:

**readFileSync(<filename/path>,<format>):**

* This method is use to reading the file in synchronous form.
* First argument is the file name or the path if the file is exists in different folder.
* The second argument is the format in which you want to read the file. By default it is hexa code representation of the ascii value of character. It is required to pass as “UTF-8” if you want to read in text format.

Example:

const fs = require("fs");

data = fs.readFileSync("./ReadMe.txt");

console.log(data);

data = fs.readFileSync("./ReadMe.txt", "utf-8");

console.log(data);

Output:

PS C:\NODEJS> node module\_fs.js

<Buffer 48 65 6c 6c 6f 20 49 53 54 41 52 21>

Hello ISTAR!

Note: The 72 is the ascii value of “H” and 48 is the hexa value of 72. So, it represents 48 by default.

**writeFileSync(<filename/path>,<data>):**

* This method is use to writing the file in synchronous form. It will create a file if specified file does not exist. If it exists it will replace new content with old contents.
* First argument is the file name or the path if the file is exists in different folder.
* The second argument is the data which you want to write in a file.

Example:

const fs = require("fs");

fs.writeFileSync("./ReadMe.txt",”Hello MCA!”);

data = fs.readFileSync("./ReadMe.txt", "utf-8");

console.log(data);

Output:

PS C:\NODEJS> node module\_fs.js

Hello MCA!

**appendFileSync(<filename/path>,<data>):**

* This method is use to adding content in the file in synchronous form. It will add new content after the existing contents.
* First argument is the file name or the path if the file is exists in different folder.
* The second argument is the data which you want to write in a file.

Example:

const fs = require("fs");

fs.appendFileSync("./ReadMe.txt",”Hello ISTAR!”);

data = fs.readFileSync("./ReadMe.txt", "utf-8");

console.log(data);

Output:

PS C:\NODEJS> node module\_fs.js

Hello MCA!Hello MCA!

**Asynchronous methods for file system**:

**readFile(<filename/path>,<format>,<callback(err,data)):**

* This method is use to reading the file in asynchronous form.
* First argument is the file name or the path if the file is exists in different folder.
* The second argument is the format in which you want to read the file. By default it is hexa code representation of the ascii value of character. It is required to pass as “UTF-8” if you want to read in text format.
* The third argument is callback with two arguments, one is error and other is data. This will either return error or data.

Example:

data = fs.readFile("./ReadMe.txt", "utf-8", function (err, data) {

  if (err) {

    console.log("No file found!");

  } else {

    console.log(data);

  }

});

Output:

PS C:\NODEJS> node module\_fs.js

Hello MCA!

The “ReadMe.txt” is available at the specified path. So, it will return data. Now, lets try to read a file which does not exist.

Example:

data = fs.readFile("./Read.txt", "utf-8", function (err, data) {

  if (err) {

    console.log("No file found!");

  } else {

    console.log(data);

  }

});

Output:

PS C:\NODEJS> node module\_fs.js

No file found!

**writeFile(<filename/path>,<data>,<callback(err)):**

* This method is use to writing the file in asynchronous form.
* First argument is the file name or the path if the file is exists in different folder.
* The second argument is the data which you want to write in a file.
* The third argument is callback with an argument as error. It will return error if error occurred.

Example:

data = "Hello ISTAR!";

fs.writeFile("./ReadMe.txt", data, function (err) {

  if (err) throw err;

  console.log("Saved!");

});

Output:

PS C:\NODEJS> node module\_fs.js

Saved!

**appendFile(<filename/path>,<data>,<callback(err)):**

* This method is use to appending the file in asynchronous form.
* First argument is the file name or the path if the file is exists in different folder.
* The second argument is the data which you want to append in a file.
* The third argument is callback with an argument as error. It will return error if error occurred.

Example:

data = "Hello ISTAR!";

fs.appendFile("./ReadMe.txt", data, function (err) {

  if (err) throw err;

  console.log("Updated!");

});

Output:

PS C:\NODEJS> node module\_fs.js

Updated!

Difference between using synchronous method and asynchronous method.

Example:

const fs = require("fs");

console.log("Reading data synchronously");

console.log("Start!");

data = fs.readFileSync("./ReadMe.txt", "utf-8");

console.log(data);

console.log("Process!");

console.log("End!");

console.log("----------------------------");

console.log("Reading data asynchronously");

console.log("Start!");

data = fs.readFile("./ReadMe.txt", "utf-8", function (err, data) {

  if (!err) console.log(data);

});

console.log("Process!");

console.log("End!");

Output:

Reading data synchronously

Start!

Hello ISTAR!Hello ISTAR!

Process!

End!

----------------------------

Reading data asynchronously

Start!

Process!

End!

Hello ISTAR!Hello ISTAR!

Notice: The next statement for printing “process” and “end” will execute immediately after the printing “start” statement. It will not wait for the file reading operation completion in asynchronous.