Streams in NodeJs | Lecture Notes

Notes by : Milind Mishra

Some more Modules in Node Js

- We are going to talk about how we can start reading files in Node Js.
- Node Js as a runtime environment is that we have essentially brought the capablities of the javascript browser into your os and we can do a lot of things with it, like read files and write files and do a lot of things.
- We are going to talk about how we can start reading files in Node Js, with respect to ES6 Moduling and Common JS Moduling.

> Sidenote : We remember from the previous lessons that ES6 Moduling is the new way of doing things and Common JS Moduling is the old way of doing things. And ES6 Moduling can be enables in a file by renaming the extension of the file to .mjs and Common JS Moduling can be enabled in a file by renaming the extension of the file to .cjs.

- To start with a project we can `npm init` or `npm init -y` (supresses the questions) to create a package.json file for our project.
- We can install the fs module using `npm install fs` and we can import it in our file using `import fs from 'fs'` or `const fs = require('fs')` depending on the type of moduling we are using.
- You can add property type with value module, this would make the current folder/project as ES Module Type.

> A level up the directory where *package.json* exists would not support the ES6 Moduling.

- Global `__dirname` variable is available in Node Js, which gives the path of the current directory.

> Sidenote : We can use `__dirname` to get the path of the current directory and we can use `__filename` to get the path of the current file. But ES6 Moduling does not support `__dirname` and `__filename` variables. So we can use `import.meta.url` to get the path of the current file. And generally people shifting from an Express code base to a react code base they would face the change in the module type and hence support for `__dirname` and `__filename` variables vanishes, this is a thing to note.

Reading files in Node Js

- The `fs module` helps us to read files in Node Js. We can use `fs.readFile` to read a file in Node Js.
- The best part is that fs is an inbuilt module in Node Js, so we do not need to install it.
- The modules gives us support to two types of functions one being callback supported and the other being promise supported. So user may use fsPromises or fs as per their choice.
- The node:fs enables us to interact with the file system in a way modeled on standard POSIX functions.
- To use promise based APIs:

```
import * as fs from 'node:fs/promises';
```

- To use callback based APIs:

```
import * as fs from 'node:fs';
```

- We can use `fs.readFile` to read a file in Node Js. And readLines is a function which takes a file path as an argument and returns a promise which resolves to an array of lines in the file.

> ESM Syntax

```
import { open } from 'node:fs/promises';

const file = await open(filePath, 'r');

for await (const line of readLines(file)) {
  console.log(line);
}
```

> *Top level await* : You can use await outside of an async function, at the top level of a module. This means that modules with child modules that use await will wait for the child modules to finish loading before continuing to execute.

- lets say we make a new file index.html and we want to read it in Node Js, we can do that using `fs.readFile` function.

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <h1>Hello World</h1>
  </body>
</html>
```

- Lets access this file in Node Js.

```
import { readFile } from 'fs/promises';

// creating a url object for the file path
const filePath = new URL("./index.html", import.meta.url);
// import.meta.url gives the path of the current file in ES6 Moduling
// the syntax of URL object is new URL(path, base) where path is relative and base
```

```
is absolute
let contents = await readFile(filePath, { encoding: 'utf-8' }); // since its a top
level module we can use await
console.log(contents); // prints the contents of the file
```

- Now we are able to read the contents of a file in Node Js, and no this can not be done in the browser as the browser does not have access to the file system.
- To add some data to the file we can use `fs.writeFile` function.

```
import { writeFile } from 'fs/promises';

// preparing a data object to write to the file

const data = {
    name : "Milind Mishra",
    profession: "Software Engineer"
    age: 23,
}

for(const [key,value] of Object.entries(data)) {
    contents = contents.replace(`{{${key}}}`, value);
}
// Object.entries() is a method that returns an array of a given object's own
enumerable string-keyed property [key, value] pairs, in the same order as that
provided by a for...in loop (the difference being that a for-in loop enumerates
properties in the prototype chain as well).
// There are other helper functions liek Object.keys() Object.values() which can
be used to get the keys and values of an object.

console.log(contents); // prints the contents of the file with the data object
values
```

- Making a template string to write to the file.

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
      <h1>Hello World</h1>
    <h2>My name is ${data.name}</h2>
    <h2>I am a ${data.profession}</h2>
    <h2>I am ${data.age} years old</h2>
```

```
  </body>
</html>
```

> for ... of loop : The for...of statement creates a loop iterating over iterable objects, including: built-in String, Array, array-like objects (e.g., arguments or NodeList), TypedArray, Map, Set, and user-defined iterables. It invokes a custom iteration hook with statements to be executed for the value of each distinct property of the object.

```
// for ... of loop
const iterable = [10, 20, 30];

for (const value of iterable) {
  console.log(value);
}
```

- There are multiple templating libs like ejs handlebar pug etc. But we are using the native templating engine of Node Js. All the Server Side Rendering is done using the native templating engine of Node Js. And the SSR is quite good for SEO and performance. While client side was all the craze and SPAs all over the place, SSR was not that popular. But now with the advent of Next Js and Gatsby Js, SSR is becoming more and more popular. And slowly react is moving towards SSR.

> One such instance from interview that happened was use to the usage of `for(let i = 0; i <= 10; i++)` syntax rejection occured as this syntax has too many error prone points like in a project we need to write a for loop and we write `for(let i = 0; i <= 10; i++)` and we forget to increment the value of i, so the loop will run forever. So to avoid such errors we use `for(let i = 0; i < 10; i++)` syntax. Like wise if we start the iteration wrong and increment wrongly that is error prone as well. So instead we try to use for of loops and for in loops etc. So this is the reason why the interviewer rejected the code.

- Now we are moving the discussion to a file and description will be written as comments to better understand topic and move up to speed.