NODE JS

Prof.Vinay Joshi and Dr.Sarasvathi V
Department of Computer Science and Engineering

Acknowledgement
The slides are created from various internet resources with valuable contributions from multiple professors

PES UNIVERSITY

NODE JS

File System Module

Department of Computer Science and Engineering

NODE JS

File System Module

- Node implements File I/O using simple wrappers around standard POSIX functions.
- The Node File System (fs) module can be imported using the following syntax –

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

Synchronous vs Asynchronous

- Every method in the fs module has synchronous as well as asynchronous forms.
- Asynchronous methods take the last parameter as the completion function callback and the first parameter of the callback function as error.
- It is better to use an asynchronous method instead of a synchronous method, as the former never blocks a program during its execution, whereas the second one does.

NODE JS

Fs module operations

PES UNIVERSITY

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

NODE JS

Fs File Open

What is Synchronous and Asynchronous approach?

Synchronous approach: They are called blocking functions as it waits for each operation to complete, only after that, it executes the next operation, hence blocking the next command from execution i.e. a command will not be executed until & unless the query has finished executing to get all the result from previous commands.

Asynchronous approach:

- They are called non-blocking functions as 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 i.e. each command will be executed soon after the execution of the previous command.
- While the previous command runs in the background and loads the result once it is finished processing the data.

NODE JS

Fs File Open

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

// Asynchronous read
fs.readFile('input.txt', function (err, data) {
   if (err) {
      return console.error(err);
   }
   console.log("Asynchronous read: " + data.toString());
});
```

Fs File Open


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

// Synchronous read
var data = fs.readFileSync('input.txt');
console.log("Synchronous read: " + data.toString());
```

PES UNIVERSITY

Syntax

Following is the syntax of the method to open a file in asynchronous mode −

fs.open(path, flags[, mode], callback)

Parameters

Here is the description of the parameters used −

path − This is the string having file name including path.

flags − Flags indicate the behavior of the file to be opened. All possible values have been mentioned below.

mode − It sets the file mode (permission and sticky bits), but only if the file was created. It defaults to 0666, readable and writeable.

callback − This is the callback function which gets two arguments (err, fd).

Flags for read/write operations are − r,r+,rs,rs+,w,wx,w+,wx+,a,ax,a+,ax+

Fs File Open

The fs.open() method is used to create, read, or write a file. The fs.readFile() method is only for reading the file and fs.writeFile() method is only for writing to the file, whereas fs.open() method does several operations on a file. First, we need to load the fs class which is a module to access the physical file system.

Syntax:

fs.open(path, flags, mode, callback)

Fs File Open


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

// Asynchronous - Opening File
console.log("opening file!");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File open successfully");
});
```

Fs File Write

Syntax

fs.writeFile(filename, data[, options], callback)

This method will over-write the file if the file already exists. If you want to write into an existing file then you should use another method available.

Parameters

path – This is the string having the file name including path.

data – This is the String or Buffer to be written into the file.

options – The third parameter is an object which will hold {encoding, mode, flag}. By default. encoding is utf8, mode is octal value 0666. and flag is 'w'

callback – This is the callback function which gets a single parameter err that returns an error in case of any writing error.

NODE JS

Fs File Open

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

fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

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

fs.open('mynewfile2.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('Saved!');
});
```

NODE JS

Fs File Open

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

console.log("writing into existing file");
fs.writeFile('input.txt', 'Web tech', function(err) {          //write data to a file
   if (err) {
      return console.error(err);
   }

   console.log("Data written successfully!");
   console.log("Let's read newly written data");

   fs.readFile('input.txt', function (err, data) {          //Read data to a file
      if (err) {
         return console.error(err);
      }
      console.log("Asynchronous read: " + data.toString());
   });
});
```

NODE JS

Fs File Open

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

fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Fs File Read

Syntax

fs.read(fd, buffer, offset, length, position, callback) This method will use file descriptor to read the file. If you want to read the file directly using the file name, then you should use another method available.

Parameters

fd – This is the file descriptor returned by fs.open().

buffer – This is the buffer that the data will be written to.

offset – This is the offset in the buffer to start writing at.

length – This is an integer specifying the number of bytes to read.

position – This is an integer specifying where to begin reading from in the file. If position is null, data will be read from the current file position.

callback – This is the callback function which gets the three arguments, (err, bytesRead, buffer).

PES UNIVERSITY

Unlinking a File

Use fs.unlink() method to delete an existing file.

fs.unlink(path, callback);

Closing a File

fs.close(fd, callback)

fd – This is the file descriptor returned by file fs.open() method.

callback – This is the callback function No arguments other than a possible exception are given to the completion callback.

Truncate a File

fs.truncate(fd, len, callback)

fd – This is the file descriptor returned by fs.open().

len – This is the length of the file after which the file will be truncated.

callback – This is the callback function No arguments other than a possible exception are given to the completion callback.

NODE JS

Fs Module – Other Important Methods

Method	Description
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.chown(path, uid, gid, callback)	Asynchronous chown.
fs.stat(path, callback)	Returns fs.stat object which includes important file statistics.
fs.link(srcpath, dstpath, callback)	Links file asynchronously.
fs.symlink(destination, path[, type], callback)	Symlink asynchronously.
fs.rmdir(path, callback)	Renames an existing directory.
fs.mkdir(path[, mode], callback)	Creates a new directory.
fs.readdir(path, callback)	Reads the content of the specified directory.
fs.utimes(path, atime, mtime, callback)	Changes the timestamp of the file.
fs.exists(path, callback)	Determines whether the specified file exists or not.
fs.access(path[, mode], callback)	Tests a user's permissions for the specified file.
fs.appendFile(file, data[, options], callback)	Appends new content to the existing file.

NODE JS

Fs File delete

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

fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

THANK YOU

Prof.Vinay Joshi and Dr.Sarasvathi V

Department of Computer Science and Engineering