Session 5.2

Node File System

Topics

- Interacting with the File System
- Process Object
- Scalability & Child Processes

Interacting with File System

File System

_filename:

- The absolute path of the currently executing file.

_dirname:

- The absolute path to the directory containing the currently executing file.

- The values of _filename and _dirname depend on the file that references them

fs module

- Node applications perform file I/O via the fs module, a core module whose methods provide wrappers around standard file system

Wrappers around POSIX function *(both asynchronous and synchronous versions)*

- rename
- truncate
- chmod
- unlink
- rmdir
- mkdir
- close
- open
- readFile
- writeFile
- appendfile

fs module

Stream oriented functions

- fs.createReadStream() - returns a ReadableStream
- fs.createWriteStream() - returns a WritableStream

Watch a file or directory for changes

- fs.watch() - returns a fs.FsWatcher (an EventEmitter)
- 'change' event: the type of change and filename that changed
- 'error' event: emitted when an error occurs

fs flags

Flag	Description
r	Open file for reading. An exception occurs if the file does not exist.
r+	Open file for reading and writing. An exception occurs if the file does not exist.
w	Open file for writing. The file is created ifitdoesnotexist or truncated ifitexists.
w+	Open file for reading and writing. The file is created ifitdoesnotexist or truncated ifitexists.
a	Open file for appending. The file is created if it does not exist.
a+	Open file for reading and appending. The file is created if it does not exist.

fs methods

fs method	Description
fs.close(fd, callback)	Asynchronous close. No arguments other than a possible exception are given to the completion callback
fs.ftruncate(fd, len, callback)	Asynchronous ftruncate (remove data from the file).
fs.mkdir(path [, mode], callback)	Asynchronous mkdir. Mode defaults to 0777.
fs.open(path, flags [,mode], callback)	Asynchronous file open.
fs.rmdir(path, callback)	Asynchronous rmdir.
fs.stat(path, callback)	Asynchronous stat. The callback gets two arguments err, stats where stats is a fs.stats object.
fs.unlink(path, callback)	Asynchronous unlink (physically remove a file).
fs.write(fd, buffer, offset, length, position, callback)	Write buffer to the file specified by fd. Note: buffer can be Buffer or String; offset and length determine the part of the buffer to be written; position refers to the offset from the beginning of

readFile synchronous & asynchronous

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

// Asynchronous read
fs.readFile('./text.txt', function (err, data) {
   if(err)
     console.log(err);

    console.log(`Asynchronous read:  + ${data.toString()}`);
})

// Synchronous read
const data = fs.readFileSync('/test.txt');
console.log(`Synchronous read:  + ${data.toString()}`);

console.log('Program Ended');
```

writeFile asynchronous & synchronous

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

var out_data = 'Autobots, transform and roll out!';

// Asynchronous write

fs.writeFile('./output_async.txt', out_data, function (err, data) {
   if(err)
     console.log(err);

    console.log(`Output Async file content:  ${out_data}`);
})

// Synchronous write
fs.writeFileSync('/output_sync.txt');
console.log(`Output Sync file content:   ${out_data}`);

console.log('Program Ended');
```

file sync example

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

if(fs.existsSync('temp')) {
    console.log('Directory exists, removing...');
    if(fs.existsSync('temp/new.txt')) {
        fs.unlinkSync('temp/new.txt');
    }
    fs.rmdirSync('temp');
}

fs.mkdirSync('temp');
if(fs.existsSync('temp')) {
    process.chdir('temp');
    fs.writeFileSync('test.txt', 'this is a test');
    fs.renameSync('test.txt', 'new.txt');
    console.log('File has size', fs.statSync('new.txt').size);
    console.log('File contents: ' + fs.readFileSync('new.txt').toString());
}
```

Video - Creating & Removing Directories