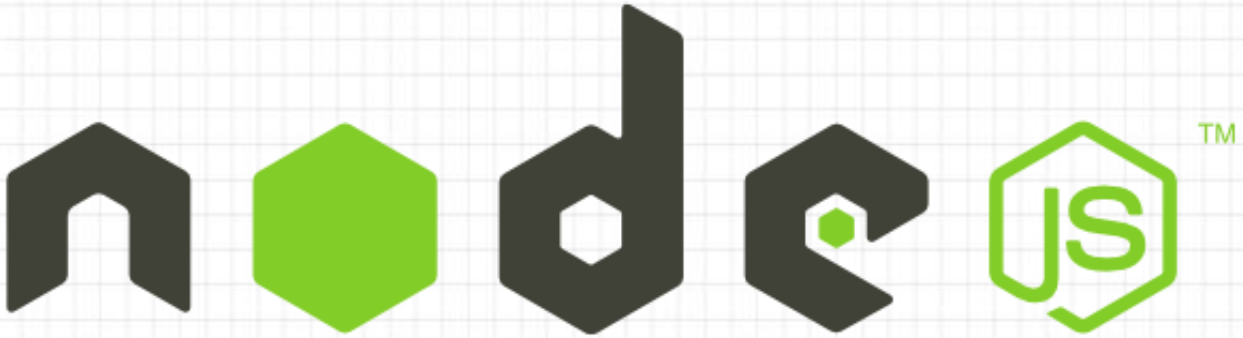


File System & Streams

Eyal Vardi
Microsoft MVP ASP.NET
blog: eyalvardi.wordpress.com

Agenda

- The File System
- Process Object
- Path & fs Modules
- Stream
- Stream Events
- Custom Stream

The 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 __filenameand __dirname depend on the file that **references them**.

Process Object

- **`process.cwd()`**
  The Current Working Directory.
- **`process.chdir("/")`**
  Changing the Current Working Directory.
- **`process.execPath`**
  Locating the nodeExecutable.

The path Module

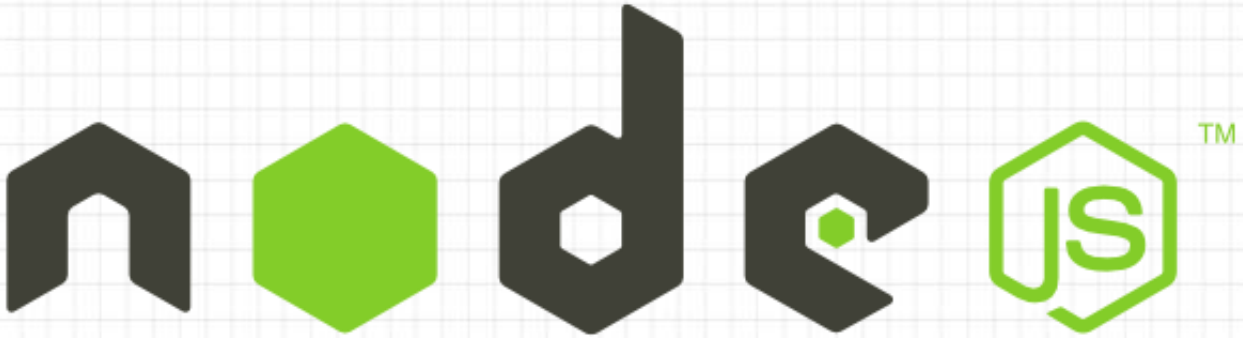
- The path module is a core module that provides a number of utility methods for working with file paths.
  - extname()
  - basename()
  - dirname()
  - Join()

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

The fs Module

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

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



Streams

Stream

- A stream is an abstract interface implemented by various objects in Node.
- Streams are readable, writable, or both.
- All streams are instances of EventEmitter

Working with Streams

```
var Stream = require("stream");
var stream = new Stream();
var duration = 5 * 1000; // 5 seconds
var end = Date.now() + duration;
var interval;

stream.readable = true;

interval = setInterval(function () {
                console.log("Emitting a data event");

                stream.emit("data", new Buffer("foo"));

                if (Date.now() >= end) {
                    console.log("Emitting an end event");

                    stream.emit("end");

                    clearInterval(interval);
                }
}, 1000);
```

Stream Events

- **data Events**
  Indicate that a new piece of stream data, referred to as a chunk, is available.
- **end Event**
  Once a stream sends all of its data, it should emit a single end event.
- **close Event**
  indicate that the underlying source of the stream data has been closed.
- **error Events**
  indicate that a problem occurred with the data stream.

Readable Streams

- The Readable stream interface is the abstraction for a *source* of data that you are reading from.
- A Readable stream will not start emitting data until you indicate that you are ready to receive it.
- Readable streams have two "modes": a **flowing mode** and a **non-flowing mode**.
- Examples of readable streams include:
  - http responses, on the client
  - http requests, on the server
  - fs read streams
  - zlib streams
  - crypto streams
  - tcp sockets
  - child process stdout and stderr
  - process.stdin

Writable Streams

- The Writable stream interface is an abstraction for a *destination* that you are writing data *to*.

- Examples of writable streams include:
  - http requests, on the client
  - http responses, on the server
  - fs write streams
  - zlib streams
  - crypto streams
  - tcp sockets
  - child process stdin
  - process.stdout, process.stderr

File Streams

- createReadStream()

```
var fs = require("fs");
var stream;
stream = fs.createReadStream(__dirname + "/foo.txt");

stream.on("data", function (data) {
    var chunk = data.toString();
    process.stdout.write(chunk);
});

stream.on("end", function() {
    console.log();
});
```

File Streams

- createWriteStream()

```
var fs = require("fs");
var readStream = fs.createReadStream(__dirname + "/foo.txt");
var writeStream = fs.createWriteStream(__dirname + "/bar.txt");
readStream.pipe(writeStream);
```

Compressing a File Using Gzip Compression

```
var fs     = require("fs");
var zlib   = require("zlib");
var gzip   = zlib.createGzip();
var input  = fs.createReadStream("input.txt");
var output = fs.createWriteStream("input.txt.gz");

input
 .pipe(gzip)
 .pipe(output);
```

API for Stream Implementors

- To implement any sort of stream, the pattern is the same:

1. Extend the appropriate parent class in your own subclass. (The **util.inherits** method is particularly helpful for this.)
2. Call the appropriate **parent class constructor in your constructor**, to be sure that the internal mechanisms are set up properly.
3. Implement one or more specific methods, as detailed below.

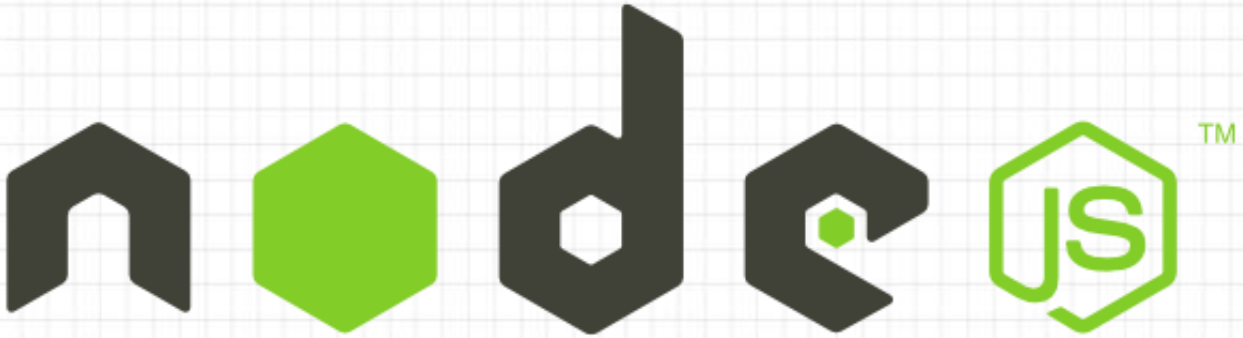
Use-case	Class	Method(s) to implement
Reading only	Readable	_read
Writing only	Writable	_write
Reading and writing	Duplex	_read, _write
Operate on written data, then read the result	Transform	_transform, _flush

Custom Stream (Counting Stream)

```
var Readable = require('stream').Readable;
var util = require('util');
util.inherits(Counter, Readable);

function Counter(opt) {
    Readable.call(this, opt);
    this._max = 1000000;
    this._index = 1;
}

Counter.prototype._read = function () {
    var i = this._index++;
    if (i > this._max)
        this.push(null);
    else {
        var str = '' + i;
        var buf = new Buffer(str, 'ascii');
        this.push(buf);
    }
};
```



Thanks

eyalvardi.wordpress.com

Eyal Vardi
Microsoft MVP ASP.NET
blog: eyalvardi.wordpress.com