Session 5.1

Node Local System

Topics

- Interacting with the Local System
  - Buffer
  - Buffers vs Streams

Buffer

Buffer

The official Node.js docs states in part...

... mechanism for reading or manipulating streams of binary data. The Buffer class was introduced as part of the Node.js API to make it possible to interact with octet streams in the context of things like TCP streams and file system operations.

To simplify it, we can say the Buffer class was introduced as part of the Node.js API to make it possible to manipulate or interact with streams of binary data.

What is Binary Data?

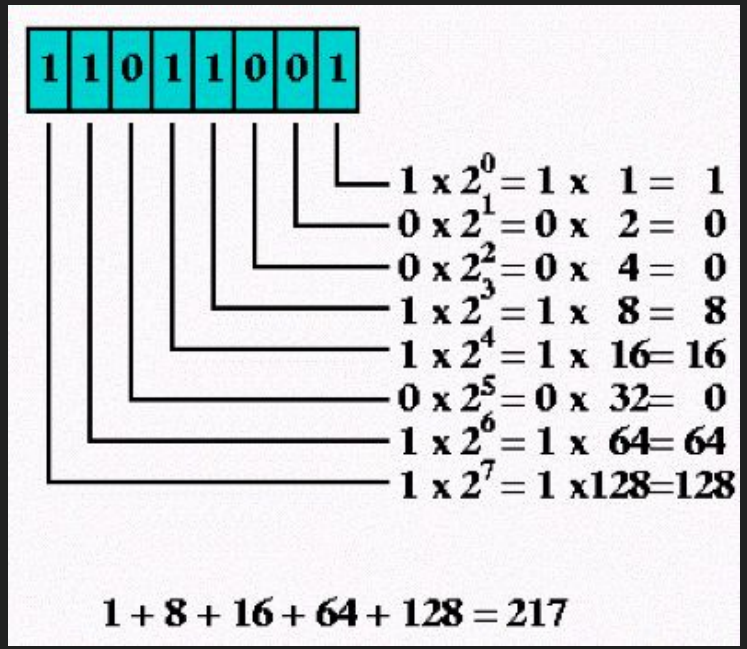
We know computers store and represent data in binaries.

Binary is a set or a collection of 1s and 0s. ie, 10, 01, 001, 110, 00101000

Each number in a binary, each 1 and 0 in a set is called a Bit, which is short for Binary digIT

- To store a piece of data, the computer needs to convert that data to binary representation. ie. to store the number 12, it needs to convert it to binary 1100

Binary Conversion



- Computers use simple pure math to do the binary conversion. Expressing a number in base-2 numeral system.
- Computers know how to represent all types of data in binaries ie. numbers, strings, images and videos

  ie. "L" string will be converted to the binary number that represents L

Character Sets & Encoding

- Character Sets are defined rules of what exact number represents each character ie. UniCode and ASCII

- Character Encoding is the rules that define how that number should be represented in binaries. ie. how many bits
  - UTF-8 states the characters should be encoded in bytes.
  - A byte is a set of eight bits - eight 1s and 0s

The point is that computer stores all data types in binaries, known as binary data.

character	encoding	bits
A	UTF-8	01000001
A	UTF-16	00000000 01000001
A	UTF-32	00000000 00000000 00000000 01000001

Streams & Buffer

- There is a huge amount of data to process, we don't need to wait for all the data to be available before you process it.
- The big data is broken down and sent in chunks over time.
- The "waiting area" for the data is the buffer. ie. bus station
- A small physical location in RAM *(raw C memory location)*, where data is temporally gathered, wait and event sent out for processing during streaming

Interacting with Buffer

```
//in Node.js:
//buffer class is global in Node, usually won't need to require it
const Buffer = require('buffer').Buffer;

//create a buffer
const buffer = new Buffer('One way to create a new buffer', 'utf-8');
//utf-8 is standard, but can be altered

//to change buffer data back to string
bufferText = buffer.toString();
//can give different encoding as argument if not utf-8

console.log(bufferText); //'This is one way to create a new buffer'
```

Buffers vs Streams

Buffers & Streams

- **Buffer:** Data Structure to store and transfer arbitrary binary data

```
> require('fs').readFileSync('./assets/divine-comedy.txt')
<Buffer 49 6e 63 69 70 69 74 20 43 6f 6d 6f 65 64 69 61 20 44 61 6e 74 69 7
3 20 41 6c 61 67 68 65 72 69 69 2c 0a 46 6c 6f 72 65 6e 74 69 6e 69 20 6e 6
1 74 69 ... 209962 more bytes>*
```

- **Stream:** Abstract interface for working with streaming data

```
> const stream = require('fs').createReadStream('./assets/divine-comedy.txt')
undefined
> stream.bytesRead
65536*
```

Read file contents & Copy to another file

Buffer

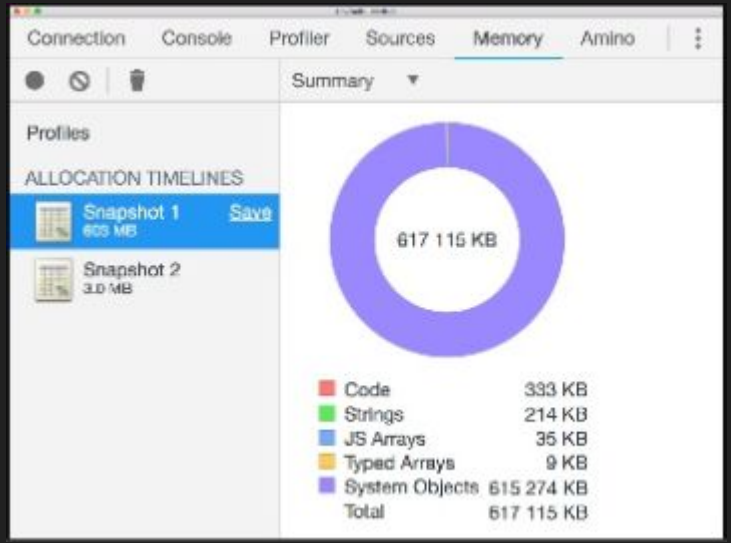
```
const { readFileSync, writeFileSync } = require('fs')

const [,, src, dest] = process.argv
const content = readFileSync(src)
writeFileSync(dest, content)
```

Streams

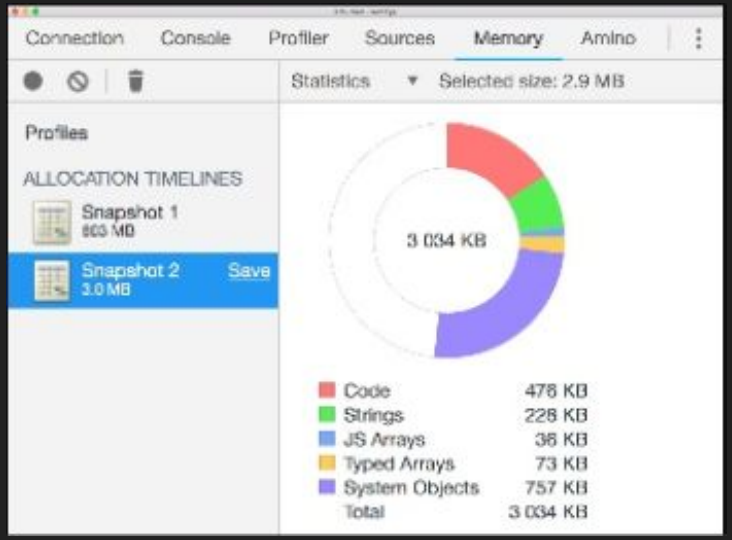
```
const {
  createReadStream,
  createWriteStream
} = require('fs')

const [,, src, dest] = process.argv
const srcStream = createReadStream(src)
const destStream = createWriteStream(dest)
srcStream.on('data', (data) => destStream.write(data))
```

Memory Comparison (~600 MB File)

Buffer



Streams



Streams vs Buffers

- Streams keep a low memory footprint even with large amounts of data
- Streams allow you to process data as soon as it arrives
- Stream processing generally does not block the event loop

All Streams are Event Emitters

- A stream instance is an object that emits events when its internal state changes, for instance

```
s.on('readable', () => {}) // ready to be consumed
s.on('data', (chunk) => {}) // new data is available
s.on('error', (err) => {}) // some error happened
s.on('end', () => {}) // no more data available
```

- The events available depend on the type of stream (ie. file readstream)

Video - Advantages of streams