IO

> IO

- [Doc] Buffer
- [Doc] String Decoder ( )
- [Doc] Stream ( )
- [Doc] Console ( )
- [Doc] File System ( )
- [Doc] Readline
- [Doc] REPL

Node.js IO . , IO, IO ?

Buffer

Buffer Node.js , IO , (/ V8 Buffer. Buffer Buffer , , .

Node.js v6.x `new Buffer()` , Buffer , Buffer ,

Buffer.from()	Buffer
Buffer.alloc()	Buffer
Buffer.allocUnsafe()	Buffer

TypedArray

Node.js Buffer ES6 TypedArray , Buffer , TypedArray Uint8Array ,

, :

```
const arr = new Uint16Array(2);
arr[0] = 5000;
arr[1] = 4000;
```

```
const buf1 = Buffer.from(arr); //     buffer
const buf2 = Buffer.from(arr.buffer); //

console.log(buf1);
//   : <Buffer 88 a0>,     buffer
console.log(buf2);
//   : <Buffer 88 13 a0 0f>

arr[1] = 6000;
console.log(buf1);
//   : <Buffer 88 a0>
console.log(buf2);
//   : <Buffer 88 13 70 17>
```

String Decoder

(String Decoder) Buffer decode string , Buffer.toString , UTF-8

```
const StringDecoder = require('string_decoder').StringDecoder;
const decoder = new StringDecoder('utf8');

const cent = Buffer.from([0xC2, 0xA2]);
console.log(decoder.write(cent)); // ¢

const euro = Buffer.from([0xE2, 0x82, 0xAC]);
console.log(decoder.write(euro)); // □
```

stringDecoder.write Buffer buffer stringDecoder.wr

```
const StringDecoder = require('string_decoder').StringDecoder;
const decoder = new StringDecoder('utf8');

decoder.write(Buffer.from([0xE2]));
decoder.write(Buffer.from([0x82]));
console.log(decoder.end(Buffer.from([0xAC])));  // □
```

Stream

Node.js `stream` . (stream) . C :

```
int copy(const char *src, const char *dest)
{
    FILE *fpSrc, *fpDest;
    char buf[BUF_SIZE] = {0};
    int lenSrc, lenDest;

    //    src
    if ((fpSrc = fopen(src, "r")) == NULL)
    {
        printf("   '%s'     \n", src);
        return FAILURE;
    }

    //   dest
    if ((fpDest = fopen(dest, "w")) == NULL)
    {
        printf("   '%s'     \n", dest);
        fclose(fpSrc);
        return FAILURE;
    }

    //   src     BUF_SIZE      buf
    while ((lenSrc = fread(buf, 1, BUF_SIZE, fpSrc)) > 0)
    {
        //   buf       dest
        if ((lenDest = fwrite(buf, 1, lenSrc, fpDest)) != lenSrc)
        {
            printf("    '%s'   \n", dest);
            fclose(fpSrc);
            fclose(fpDest);
            return FAILURE;
        }
        //        buf
        memset(buf, 0, BUF_SIZE);
    }

    //
```

```
    fclose(fpSrc);
    fclose(fpDest);
    return SUCCESS;
}
```

, 20G , 20G , , . 1MB (buf) 1M

Node.js , C , libuv Ev tEmitter . . linux/unix

Stream

Readable		_read
Writable		_write
Duplex		_read, _write
Transform	,	_transform, _flush

Node API , buffer . Ja `objectMode` null, .) . "

Node.js stream , `src` `buf` `dest` , ,(`dest`) . , , ( ) .

Readable Writable `writable._writableState.getBuffer()` `readable._readableState.buffer` .
stream `highWaterMark` `objectMode` stream, .

`stream.push()` `stream.read()` `highWaterMark` . , , .

writable.w `highWaterMark` , writable.write() true ( ),

```
// Write the data to the supplied writable stream one million times.
// Be attentive to back-pressure.
function writeOneMillionTimes(writer, data, encoding, callback) {
  let i = 1000000;
  write();
  function write() {
    var ok = true;
    do {
      i--;
      if (i === 0) {
        // last time!
        writer.write(data, encoding, callback);
```

```
      } else {
        // see if we should continue, or wait
        // don't pass the callback, because we're not done yet.
        ok = writer.write(data, encoding);
      }
    } while (i > 0 && ok);
    if (i > 0) {
      // had to stop early!
      // write some more once it drains
      writer.once('drain', write);
    }
  }
}
```

Duplex Transform

Duplex Transform , , , , . net.Socket

pipe

stream `.pipe()`, `objectMode` `objectMode` . pipe . ,

pipe Node.js pipe . pipe David Cai

Console

console.log `os`. 6.x) `this._stdout` `process.stdout`:

```
// As of v8 5.0.71.32, the combination of rest param, template string
// and .apply(null, args) benchmarks consistently faster than using
// the spread operator when calling util.format.
Console.prototype.log = function(...args) {
  this._stdout.write(`${util.format.apply(null, args)}\n`);
};
```

console.log :

```
let print = (str) => process.stdout.write(str + '\n');

print('hello world');
```

: , (util.format).

console.log.bind(console)

```
//     https://github.com/nodejs/node/blob/v6.x/lib/console.js
function Console(stdout, stderr) {
  // ... init ...

  // bind the prototype functions to this Console instance
  var keys = Object.keys(Console.prototype);
  for (var v = 0; v < keys.length; v++) {
    var k = keys[v];
    this[k] = this[k].bind(this);
  }
}
```

File

“ ” Unix/Linux , Unix/Linux , fd (

Node.js POSIX I/O . require('fs') . . fs.open

// TODO

UTF8, GBK, es6 ,

BOM

stdio

stdio (standard input output) , (stdin), (stdout), (`process.stdin` (Readable), `process.stdout` (Writable) `process.stderr` (Writable) stream.

`printf("hello, world!");` python C/ruby `print 'hello, world!'` JavaScript `console.log('hello, world!');`

C , :

```
int printf(FILE *stream,      )
{
  // ...

  // 1.
  char *s = malloc(4096);

  // 2.          ,      s
  //      ...

  // 3.   s        stream
  fwrite(s, stream);

  // 4.
  free(s);

  // ...
}
```

3 , stream stdout ( ). shell , shell fork shell (, shell stdout, shell .

, shell stdin, stdin , shell . (PS: shell windows cmd, po

ssh , shell stdout, shell sshd fork , std

, C stdio, stdio , shell , .

```
for (; i < getdtablesize(); ++i) {
   close(i);  //      fd
}
```

Linux/unix fd , 0 . .

```
console.log(process.stdin.fd); // 0
console.log(process.stdout.fd); // 1
console.log(process.stderr.fd); // 2
```

IPC , fd? , , IPC , ?fd .

?

, Node.js , Node.js (process.stdin stream) .

, read , readSync stdin . stackoverflow:

```
/*
 * http://stackoverflow.com/questions/3430939/node-js-readsync-from-stdin
 * @mklement0
 */
var fs = require('fs');

var BUFSIZE = 256;
var buf = new Buffer(BUFSIZE);
var bytesRead;

module.exports = function() {
  var fd = ('win32' === process.platform) ? process.stdin.fd : fs.openSync('/dev/stdin',
'rs');
  bytesRead = 0;

  try {
    bytesRead = fs.readSync(fd, buf, 0, BUFSIZE);
  } catch (e) {
    if (e.code === 'EAGAIN') { // 'resource temporarily unavailable'
      // Happens on OS X 10.8.3 (not Windows 7!), if there's no
      // stdin input - typically when invoking a script without any
      // input (for interactive stdin input).
      // If you were to just continue, you'd create a tight loop.
      console.error('ERROR: interactive stdin input not supported.');
      process.exit(1);
    } else if (e.code === 'EOF') {
      // Happens on Windows 7, but not OS X 10.8.3:
      // simply signals the end of *piped* stdin input.
      return '';
    }
    throw e; // unexpected exception
  }
```

```
  if (bytesRead === 0) {
    // No more stdin input available.
    // OS X 10.8.3: regardless of input method, this is how the end
    //   of input is signaled.
    // Windows 7: this is how the end of input is signaled for
    //   *interactive* stdin input.
    return '';
  }
  // Process the chunk read.

  var content = buf.toString(null, 0, bytesRead - 1);

  return content;
};
```

Readline

`readline` Readble stream (process.stdin) . net, http stream

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

const rl = readline.createInterface({
  input: fs.createReadStream('sample.txt')
});

rl.on('line', (line) => {
  console.log(`Line from file: ${line}`);
});
```

, realine `input.on('keypress', onkeypress)` line , stream ,

PS: , , scanf (, ts). Node.js C

REPL

Read-Eval-Print-Loop (REPL)

Revision #2

Created 19 July 2021 15:13:52 by

Updated 19 July 2021 15:18:56 by