**[3장. 노드 기능 알아보기]**

1. PERL 사용하기

입력한 코드를 읽고(Read), 해석하고(Eval), 결과물을 반환하고(Print), 종료할 때까지 반복(Loop)한다고 해서 REPL(Read Eval Print Loop)이라고 부릅니다.

[실습]

vsCode에서 ctrl+`을 누르면 터미널을 켤 수 있습니다.

C:\materK1m\newmimac\test\_kes>node

> const str = 'Hello world, hello node';

undefined

> console.log(str);

Hello world, hello node

undefined

REPL을 종료하려면 ctrl+C를 두 번 누르거나, REPL 창에 .exit를 입력하면 됩니다.

1. JS 파일 실행하기

[실습]

function helloWorld(){

console.log('Hello World');

helloNode();

}

function helloNode(){

console.log('Hello Node');

}

helloWorld();

C:\materK1m\newmimac\test\_kes> cd node\_basic

C:\materK1m\newmimac\test\_kes\node\_basic>node jsExec

Hello World

Hello Node

1. 모듈로 만들기

[실습]

module\_var.js

const odd = '홀수 입니다.';

const even = '짝수 입니다.';

module.exports={ // 변수들을 담은 객체를 대입

odd,

even

}

module\_func.js

const{odd,even} = require('./module\_var'); // 불러옴

function checkOddOrEven(num){

if(num%2){ //홀수면

return odd;

}

return even;

}

module.exports = checkOddOrEven; // 대입

module\_index.js

const{odd,even} = require('./module\_var');

const checkNumber = require('./module\_func');

function checkStringOddOrEven(str){

if(str.length % 2){ // 홀수면

return odd;

}

return even;

}

console.log(checkNumber(10));

console.log(checkStringOddOrEven('hello'));

C:\materK1m\newmimac\test\_kes\node\_basic>node module\_index

짝수 입니다.

홀수 입니다

1. 노드 내장 객체 알아보기

* global 객체의 속성에 대입하여 파일 간에 데이터를 공유할 수 있지만, 프로그램의 규모가 커질 수록 어떤 파일에서 global 객체에 값을 대입했는지 찾기 힘들어 유지보수에 어려움을 겪게 되오니 남용은 하지 말 것

module.exports = () => global.message;

const A = require('./globalA');

global.message = '안녕하세요';

console.log(A());

C:\materK1m\newmimac\test\_kes\node\_basic>node globalB

안녕하세요

* console

consoletime(레이블) : console.timeEnd(레이블)과 대응되어 같은 레이블을 가진 time과 timeEnd 사이의 시간을 측정합니다.

console.log(내용) : 평범한 로그를 콘솔에 표시합니다. console.log(내용, 내용, …) 처럼 여러 내용을 동시에 표시할 수도 있습니다.

console.error(내용) : 에러를 콘솔에 표시합니다.

console.dir(객체, 옵션) : 객체를 콘솔에 표시할 때 사용합니다. 첫 번째 인자로 표시할 객체를 넣고, 두 번째 인자로 옵션을 넣습니다. 옵션의 color를 true로 하면 콘솔에 색이 추가되어 보기가 한결 편해집니다. depth 객체 안의 객체를 몇 단계까지 보여줄지를 결정합니다. 기본값은 2입니다.

console.trace(레이블) : 에러가 어디서 발생하는지 추적할 수 있게 해줍니다. 보통은 에러 발생 시 에러 위치를 알려주므로 자주 사용하지는 않지만, 위치가 나오지 않는다면 사용할만합니다.

[실습]

const string = 'abc';

const number = 1;

const boolean = true;

const obj = {

outside : {

inside : {

key: 'value',

},

},

};

console.time('전체 시간');

console.log('평범한 로그입니다 쉽표로 구분해 여러 값을 찍을 수 있습니다');

console.log(string, number, boolean);

console.error('에러 메시지는 console.error에 담아주세요');

console.dir(obj, {color : false, depth : 2});

console.dir(obj, {color : true, depth : 1});

console.time('시간 측정');

for(let i=0; i < 100000; i++){

continue;

}

console.timeEnd('시간 측정');

function b(){

console.trace('에러 위치 추적');

}

function a(){

b();

}

a();

console.timeEnd('전체 시간');

C:\materK1m\newmimac\test\_kes\node\_basic>node console

평범한 로그입니다 쉽표로 구분해 여러 값을 찍을 수 있습니다

abc 1 true

에러 메시지는 console.error에 담아주세요

{ outside: { inside: { key: 'value' } } }

{ outside: { inside: [Object] } }

시간 측정: 2.932ms

Trace: 에러 위치 추적

at b (C:\materK1m\newmimac\test\_kes\node\_basic\console.js:26:13)

at a (C:\materK1m\newmimac\test\_kes\node\_basic\console.js:30:5)

at Object.<anonymous> (C:\materK1m\newmimac\test\_kes\node\_basic\console.js:33:1)

at Module.\_compile (internal/modules/cjs/loader.js:721:30)

at Object.Module.\_extensions..js (internal/modules/cjs/loader.js:732:10)

at Module.load (internal/modules/cjs/loader.js:620:32)

at tryModuleLoad (internal/modules/cjs/loader.js:560:12)

at Function.Module.\_load (internal/modules/cjs/loader.js:552:3)

at Function.Module.runMain (internal/modules/cjs/loader.js:774:12)

at executeUserCode (internal/bootstrap/node.js:499:15)

전체 시간: 39.522ms

* 타이머
* /\* setTimeout, setInterval, setImmediate는 노드에서 window 대신 global 객체 안에 들어 있습니다.
* \* setTimeout(콜백함수, 밀리초) : 주어진 밀리초(1000분의 1초) 이후에 콜백함수를 실행합니다.
* \* setInterval(콜백함수, 밀리초) : 주어진 밀리초마다 콜백 함수를 반복 실행합니다.
* \* setImmediate(콜백함수) : 콜백함수를 즉시 실행합니다.
* \*
* \* clearTimeout(아이디) : setTimeout을 취소합니다.
* \* clearInterval(아이디) : setInterval을 취소합니다.
* \* clearImmediate(아이디) : setImmediate를 취소합니다.
* \*/
* const timeout = setTimeout(() => {
* console.log('1.5초 후 실행');
* }, 1500);
* const interval = setInterval(() =>{
* console.log('1초마다 실행');
* }, 1000);
* const timeout2 = setTimeout(() => {
* console.log('실행되지 않습니다.');
* }, 3000);
* setTimeout(() => {
* clearTimeout(timeout2);
* clearInterval(interval);
* }, 2500);
* const immediate = setImmediate(()=>{
* console.log('즉시 실행');
* });
* const immediate2 = setImmediate(() => {
* console.log('실행되지 않습니다.');
* });
* clearImmediate(immediate2);

[결과값]

C:\materK1m\newmimac\test\_kes\node\_basic>node Timer

즉시 실행

1초마다 실행

1.5초 후 실행

1초마다 실행

* \_\_filename, \_\_dirname
* console.log(\_\_filename);
* console.log(\_\_dirname);

C:\materK1m\newmimac\test\_kes\node\_basic>node \_\_filename\_\_dirname

C:\materK1m\newmimac\test\_kes\node\_basic\\_\_filename\_\_dirname.js

C:\materK1m\newmimac\test\_kes\node\_basic

* module, exports

exports.odd="홀수입니다";

exports.even="짝수입니다";

const{odd,even} = require('./moduleExports');

const checkNumber = require('./module\_func');

function checkStringOddOrEven(str){

if(str.length % 2){ // 홀수면

return odd;

}

return even;

}

console.log(checkNumber(10));

console.log(checkStringOddOrEven('hello'));

C:\materK1m\newmimac\test\_kes\node\_basic>node moduleExports\_index

짝수 입니다.

홀수입니다

* process

C:\materK1m\newmimac\test\_kes\node\_basic>node

> process.version

'v11.7.0'

> process.platform

'win32'

> process.pid

12812

> process.uptime()

26.11

> process.execPath

'C:\\Program Files\\nodejs\\node.exe'

> process.cwd()

'C:\\materK1m\\newmimac\\test\_kes\\node\_basic'

> process.cpuUsage()

{ user: 140000, system: 46000 }

* process.env

process.env는 서비스의 중요한 키를 저장하는 공간으로도 사용됩니다.

서버나 데이터 베이스의 비밀번호와 각종 API 키를 코드에 직접 입력하는 것은 위험하기 때문에 process.env의 속성으로 대체 합니다

const secretId = process.env.SECRET\_ID;

const secretCode = process.env.SECRET\_CODE;

이제 process.env에 직접 SECRET\_ID와 SECRET\_CODE를 넣어주면 됩니다.

C:\materK1m\newmimac\test\_kes\node\_basic>set SECRET\_ID = kes

C:\materK1m\newmimac\test\_kes\node\_basic>set SECRET\_CODE = kes

* process.nextTick(콜백)

이벤트 루프가 다른 콜백 함수들 보다 nextTick의 콜백함수를 우선으로 처리하도록 하는 예제

setImmediate(()=>{

console.log("immediate");

});

process.nextTick(()=>{

console.log("nextTick");

})

setTimeout(()=>{

console.log("timeout");

},0);

Promise.resolve().then(()=>console.log('promise'));

C:\materK1m\newmimac\test\_kes\node\_basic>node process.nextTick

nextTick

promise

timeout

immediate

* process.exit(코드)

let i = 1;

setInterval(()=>{

if(i === 5){

console.log("종료!");

process.exit();

}

console.log(i);

i += 1;

}, 1000);

C:\materK1m\newmimac\test\_kes\node\_basic>node process.exit

1

2

3

4

종료!

1. 노드 내장 모듈 사용하기

* OS

const os = require('os');

console.log('운영체제 정보--------------------');

console.log('os.arch() : ', os.arch()); // process.arch와 동일합니다.

console.log('os.platform() : ', os.platform()); // process.platform과 동일합니다.

console.log('os.type() : ', os.type()); // 운영체제의 종류를 보여줍니다.

console.log('os.uptime() : ', os.uptime()); // 운영체제 부팅 이후 흐른 시간(초)를 보여줍니다.

console.log('os.hostname() : ', os.hostname()); // 컴퓨터의 이름을 보여줌니다.

console.log('os.release() : ', os.release()); // 운영체제의 버전을 보여줍니다.

console.log("경로-----------------------");

console.log("os.homedir() : ", os.homedir()); // 홈 디렉터리 경로를 보여줍니다.

console.log("os.tmpdir() : ", os.tmpdir()); // 임시 파일 저장 경로를 보여줍니다.

console.log("cpu 정보-----------------------");

console.log("os.cpus() : ", os.cpus()); // 컴퓨터의 코어 정보를 보여줍니다.

console.log("os.cpus().length : ", os.cpus().length);

console.log("메모리 정보-----------------------");

console.log("os.freemem() : ", os.freemem()); // 사용 가능한 메모리(RAM)를 보여줍니다.

console.log("os.totalmem() : ", os.totalmem()); // 전체 메모리 용량을 보여줍니다.

C:\materK1m\newmimac\test\_kes\node\_basic>node os

운영체제 정보--------------------

os.arch() : x64

os.platform() : win32

os.type() : Windows\_NT

os.uptime() : 18898

os.hostname() : ±èÀº¼±\_NOT

os.release() : 6.1.7601

경로-----------------------

os.homedir() : C:\Users\Administrator

os.tmpdir() : C:\Users\ADMINI~1\AppData\Local\Temp

cpu 정보-----------------------

os.cpus() : [ { model: 'Intel(R) Core(TM) i5-4210M CPU @ 2.60GHz',

speed: 2594,

times:

{ user: 1003850,

nice: 0,

sys: 1022648,

idle: 16872147,

irq: 25646 } },

{ model: 'Intel(R) Core(TM) i5-4210M CPU @ 2.60GHz',

speed: 2594,

times:

{ user: 814278, nice: 0, sys: 748305, idle: 17335704, irq: 73913 } },

{ model: 'Intel(R) Core(TM) i5-4210M CPU @ 2.60GHz',

speed: 2594,

times:

{ user: 1133862, nice: 0, sys: 1342716, idle: 16421585, irq: 3104 } },

{ model: 'Intel(R) Core(TM) i5-4210M CPU @ 2.60GHz',

speed: 2594,

times:

{ user: 391702, nice: 0, sys: 255202, idle: 18251134, irq: 1138 } } ]

os.cpus().length : 4

메모리 정보-----------------------

os.freemem() : 3796910080

os.totalmem() : 8493228032

* path

const path = require("path");

const string = \_\_filename;

console.log("path.sep : ", path.sep); // 경로의 구분자

console.log("path.delimiter : ", path.delimiter); // 환경변수의 구분자. process.env.PATH를 입력하면 여러개의 경로가 이 구분자로 구분되어 있다. Window는 세미콜론(;), POSIX는 콜론(:)

console.log("---------------------------------");

console.log("path.dirname() : ", path.dirname(string)); // 파일이 위치한 폴더 경로를 보여줌

console.log("path.extname() : ", path.extname(string)); // 파일의 확장자를 보여줌

console.log("path.basename() : ", path.basename(string)); // 파일의 이름(확장자 포함)을 보여줌

console.log("path.basename() : ", path.basename(string, path.extname(string))); // 파일의 이름만 표시하고 싶다면 basename의 두번 째 인자로 파일의 확장자를 넣어주면 됨

console.log("---------------------------------");

console.log("path.parse()", path.parse(string)); // 파일의 경로를 root, dir, base, ext, name으로 분리

console.log("path.format() : ", path.format({ // path.parse() 한 객체를 파일 경로로 합침

dir : "C:\\user\\zerocho",

name : "path",

ext : ".js",

}));

console.log("path.normalize() : ", path.normalize("C://users\\\zerocho\\\path.js")); // /나 \를 실수로 여러번 사용했거나 혼용햇을 때 정상적인 경로로 변환해줌

console.log("---------------------------------");

console.log("path.isAbsolute() : ", path.isAbsolute("C:\\")); // 파일의 경로가 절대경로인지 상대경로인지 true나 false로 알려줌

console.log("path.isAbsolute() : ", path.isAbsolute("./home")); // 경로를 두 개 넣으면 첫번째 경로에서 두번째 경로로 가는 법을 알려줌

console.log("---------------------------------");

console.log("path.relative() : ", path.relative("C:\\users\\zerocho\\path.js","C:\\"));

// path.join(경로, ...) 여러 인자를 넣으면 하나의 경로로 합쳐줍니다. 상대경로인 ..(부모 디렉터리)와 .(현 디렉터리)도 알아서 처리해줍니다.

console.log("path.join() : ", path.join(\_\_dirname, "..", "users", ".", "zerocho")); // path.join()과 비슷하지만 차이가 있습니다.

C:\materK1m\newmimac\test\_kes\node\_basic>node path

path.sep : \

path.delimiter : ;

---------------------------------

path.dirname() : C:\materK1m\newmimac\test\_kes\node\_basic

path.extname() : .js

path.basename() : path.js

path.basename() : path

---------------------------------

path.parse() { root: 'C:\\',

dir: 'C:\\materK1m\\newmimac\\test\_kes\\node\_basic',

base: 'path.js',

ext: '.js',

name: 'path' }

path.format() : C:\user\zerocho\path.js

path.normalize() : C:\users\zerocho\path.js

---------------------------------

path.isAbsolute() : true

path.isAbsolute() : false

---------------------------------

path.relative() : ..\..\..

path.join() : C:\materK1m\newmimac\test\_kes\users\zerocho

* url

const url = require("url");

const URL = url.URL;

const myURL = new URL("http://www.gilbut.co.kr/book/bookList.aspx?sercate1=001001000#andchor");

console.log("new URL() : ", myURL);

console.log("url.format() : ", url.format(myURL));

console.log("------------------------------------");

const parsedUrl = url.parse("http://www.gilbut.co.kr/book/bookList.aspx?sercate1=001001000#andchor");

console.log("url.parse() : ", parsedUrl); // 주소를 분해합니다.

console.log("url.format() : ", url.format(parsedUrl)); // 분해되었던 url 객체를 다시 원래 상태로 조립합니다.

C:\materK1m\newmimac\test\_kes\node\_basic>node url

new URL() : URL {

href:

'http://www.gilbut.co.kr/book/bookList.aspx?sercate1=001001000#andchor',

origin: 'http://www.gilbut.co.kr',

protocol: 'http:',

username: '',

password: '',

host: 'www.gilbut.co.kr',

hostname: 'www.gilbut.co.kr',

port: '',

pathname: '/book/bookList.aspx',

search: '?sercate1=001001000',

searchParams: URLSearchParams { 'sercate1' => '001001000' },

hash: '#andchor' }

url.format() : http://www.gilbut.co.kr/book/bookList.aspx?sercate1=001001000#andchor

------------------------------------

url.parse() : Url {

protocol: 'http:',

slashes: true,

auth: null,

host: 'www.gilbut.co.kr',

port: null,

hostname: 'www.gilbut.co.kr',

hash: '#andchor',

search: '?sercate1=001001000',

query: 'sercate1=001001000',

pathname: '/book/bookList.aspx',

path: '/book/bookList.aspx?sercate1=001001000',

href:

'http://www.gilbut.co.kr/book/bookList.aspx?sercate1=001001000#andchor' }

url.format() : <http://www.gilbut.co.kr/book/bookList.aspx?sercate1=001001000#andchor>

* searchParams.js

const{URL} = require("url");

const myURL = new URL("http://www.gilbut.co.kr/?page=3&limit=10&category=nodjs&category=javascript");

console.log("searchParams : ", myURL.searchParams);

console.log("searchParams.getAll() : ", myURL.searchParams.getAll("category")); // getAll(key) : 키에 해당하는 모든 값들을 가져옵니다.

console.log("searchParams.get() : ", myURL.searchParams.get("limit")); // get(key) : 키에 해당하는 첫번 째 값만 가져옵니다.

console.log("searchParams.has() : ", myURL.searchParams.has("page")); // has(key) : 해당키가 있는지 없는지를 검사합니다.

console.log("searchParams.key() : ", myURL.searchParams.keys()); // keys() : searchParams의 모든 키를 반복기 객체로 가져옵니다.

console.log("searchParams.values() : ", myURL.searchParams.values()); // values() : searchParams의 모든 값을 반복기 객체로 가져옵니다.

myURL.searchParams.append("filter","es3"); // append(키, 값) : 해당 키를 추가합니다.

myURL.searchParams.append("filter","es5");

console.log(myURL.searchParams.getAll("filter"));

myURL.searchParams.set("filter","es6"); // set(키, 값) : append와 비슷하지만 같은 키의 값들을 모두 지우고 새로 추가합니다.

console.log(myURL.searchParams.getAll("filter"));

myURL.searchParams.delete("filter"); // delete(키) : 해당 키를 제거합니다.

console.log(myURL.searchParams.getAll("filter"));

console.log("searchParams.toString() : ", myURL.searchParams.toString()); // toString() 조작한 searchParams 객체를 다시 문자열로 만듭니다.

myURL.search = myURL.searchParams.toString();

C:\materK1m\newmimac\test\_kes\node\_basic>node searchParams

searchParams : URLSearchParams {

'page' => '3',

'limit' => '10',

'category' => 'nodjs',

'category' => 'javascript' }

searchParams.getAll() : [ 'nodjs', 'javascript' ]

searchParams.get() : 10

searchParams.has() : true

searchParams.key() : URLSearchParams Iterator { 'page', 'limit', 'category', 'category' }

searchParams.values() : URLSearchParams Iterator { '3', '10', 'nodjs', 'javascript' }

[ 'es3', 'es5' ]

[ 'es6' ]

[]

searchParams.toString() : page=3&limit=10&category=nodjs&category=javascript

* querystring

const url = require("url");

const querystring = require("querystring");

const parsedUrl = url.parse("http://www.gilbut.co.kr/?page=3&limit=10&category=nodejs&category=javascript");

const query = querystring.parse(parsedUrl.query);

console.log('querystring.parse() : ', querystring.parse(parsedUrl.query));

console.log("querystring.parse() : ", query);

console.log('querystring.stringfy() : ', querystring.stringify(query));

C:\materK1m\newmimac\test\_kes\node\_basic>node querystring

querystring.parse() : [Object: null prototype] { page: '3', limit: '10', category: [ 'nodejs', 'javascript' ] }

querystring.parse() : [Object: null prototype] { page: '3', limit: '10', category: [ 'nodejs', 'javascript' ] }

querystring.stringfy() : page=3&limit=10&category=nodejs&category=javascript

* crypto

[단방향 암호화]

const crypto = require("crypto");

console.log('base64 : ', crypto.createHash('sha512').update('비밀번호').digest('base64'));

console.log('hex : ', crypto.createHash('sha512').update('비밀번호').digest('hex'));

console.log('base64 : ', crypto.createHash('sha512').update('다른 비밀번호').digest("base64"));

C:\materK1m\newmimac\test\_kes\node\_basic>node hash

base64 : dvfV6nyLRRt3NxKSlTHOkkEGgqW2HRtfu19Ou/psUXvwlebbXCboxIPmDYOFRIpqav2eUTBFuHaZri5x+usy1g==

hex : 76f7d5ea7c8b451b773712929531ce92410682a5b61d1b5fbb5f4ebbfa6c517bf095e6db5c26e8c483e60d8385448a6a6afd9e513045b87699ae2e71faeb32d6

base64 : cx49cjC8ctKtMzwJGBY853itZeb6qxzXGvuUJkbWTGn5VXAFbAwXGEOxU2Qksoj+aM2GWPhc1O7mmkyohXMsQw==

const crypto = require("crypto");

crypto.randomBytes(64,(err,buf)=>{

const salt = buf.toString('base64');

console.log('salt : ', salt);

crypto.pbkdf2('비밀번호', salt, 100000, 64, 'sha512', (err, key)=>{

console.log('password : ', key.toString('base64'));

})

})

C:\materK1m\newmimac\test\_kes\node\_basic>node pbkdf2

salt : T7PhK6fXFYEGlGhBlZ4B1CSdYaH8PqSZMchhqBTV4ucqGSEvN3crNvcn/C8zANGrihMXFF1iHG277Iz+iyjlyA==

password : z6RjnTRvdoKUKrPVLgFa+U/ImdJHKLDhGOHPAYlxCkxTjGj2jJiSAv2K0ixZHfeOC6RdnH+WwmD1IaFgGwNHjw==

[양방향 암호화]

const crypto = require('crypto');

const cipher = crypto.createCipher('aes-256-cbc', '열쇠');

let result = cipher.update('암호화할 문장', 'utf8', 'base64');

result += cipher.final('base64');

console.log('암호화 : ', result);

const decipher = crypto.createDecipher('aes-256-cbc', '열쇠');

let result2 = decipher.update(result, 'base64','utf8');

result2 += decipher.final('utf8');

console.log('복호화 : ', result2);

C:\materK1m\newmimac\test\_kes\node\_basic>node cipher

암호화 : ooogp/vac4l26/ezEglCluFn9vjfixVtCUCaqiaMr28=

복호화 : 암호화할 문장

(node:13752) [DEP0106] DeprecationWarning: crypto.createCipher is deprecated.

* util

const util = require('util');

const crpto = require('crypto');

const dontUseMe = util.deprecate((x,y)=>{

console.log(x+y);

}, 'donstUseMe 함수는 deprecated되었으니 더 이상 사용하지 마세요!');

dontUseMe(1,2);

const randomBytesPromise = util.promisify(crypto.randomBytes);

randomBytesPromise(64)

.then((buf)=>{

console.log(buf.toString('base64'));

})

.catch((error)=>{

console.error(error);

});

C:\materK1m\newmimac\test\_kes\node\_basic>node util

3

C:\materK1m\newmimac\test\_kes\node\_basic\util.js:9

const randomBytesPromise = util.promisify(crypto.randomBytes);

^

ReferenceError: crypto is not defined

at Object.<anonymous> (C:\materK1m\newmimac\test\_kes\node\_basic\util.js:9:43)

at Module.\_compile (internal/modules/cjs/loader.js:721:30)

at Object.Module.\_extensions..js (internal/modules/cjs/loader.js:732:10)

at Module.load (internal/modules/cjs/loader.js:620:32)

at tryModuleLoad (internal/modules/cjs/loader.js:560:12)

at Function.Module.\_load (internal/modules/cjs/loader.js:552:3)

at Function.Module.runMain (internal/modules/cjs/loader.js:774:12)

at executeUserCode (internal/bootstrap/node.js:499:15)

at startMainThreadExecution (internal/bootstrap/node.js:436:3)

* 파일 시스템 접근하기

const fs = require('fs');

fs.readFile('./readme.txt', (err,data) => {

if(err){

throw err;

}

console.log(data);

console.log(data.toString());

})

C:\materK1m\newmimac\test\_kes\node\_basic>node readFile

<Buffer ec a0 80 eb a5 bc 20 ec 9d bd ec 96 b4 ec a3 bc ec 84 b8 ec 9a 94 2e 20>

저를 읽어주세요.

const fs = require('fs');

fs.writeFile('./writeme.txt', '글이 입력됩니다.', (err)=>{

if(err){

throw err;

}

fs.readFile('./writeme.txt',(err,data)=>{

if(err){

throw err;

}

console.log(data.toString());

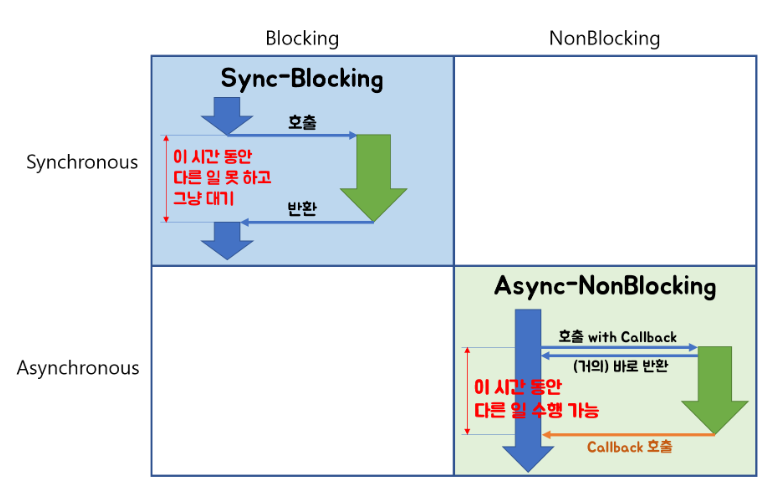
});

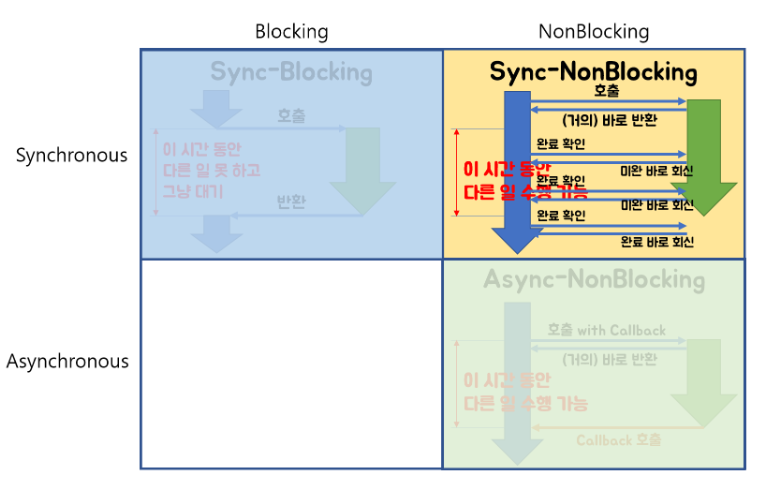
});

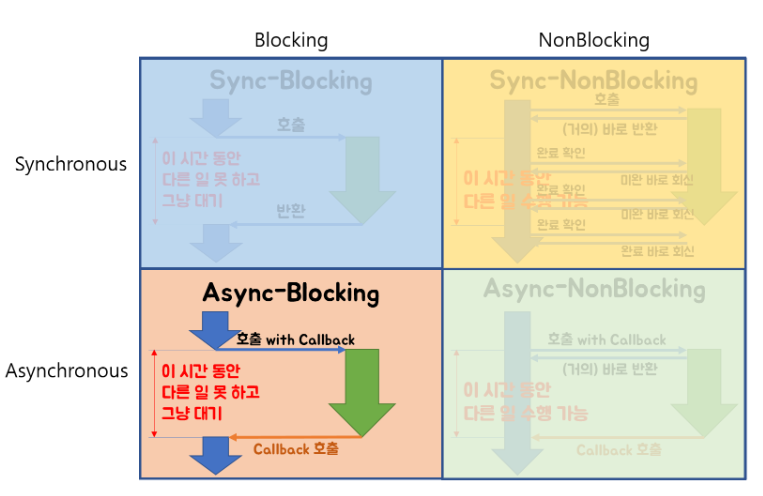
C:\materK1m\newmimac\test\_kes\node\_basic>node writeFile

글이 입력됩니다.

* 동기 메서드와 비동기 메서드







const fs = require("fs");

console.log('시작');

fs.readFile('./readme2.txt', (err, data) => {

if(err){

throw err;

}

console.log('1번', data.toString());

});

fs.readFile('./readme2.txt', (err, data)=> {

if(err){

throw err;

}

console.log('2번', data.toString());

});

fs.readFile('./readme2.txt', (err, data) => {

if(err){

throw err;

}

console.log('3번', data.toString());

});

console.log('끝');

C:\materK1m\newmimac\test\_kes\node\_basic>node async

시작

끝

2번 저를 여러번 읽어보세요.

1번 저를 여러번 읽어보세요.

3번 저를 여러번 읽어보세요.

const fs = require("fs");

console.log('시작');

let data = fs.readFileSync('./readme2.txt');

console.log('1번', data.toString());

data = fs.readFileSync('./readme2.txt');

console.log('2번', data.toString());

data = fs.readFileSync('./readme2.txt');

console.log('3번', data.toString());

console.log('끝');

C:\materK1m\newmimac\test\_kes\node\_basic>node sync

시작

1번 저를 여러번 읽어보세요.

2번 저를 여러번 읽어보세요.

3번 저를 여러번 읽어보세요.

끝

const fs = require('fs');

console.log('시작');

fs.readFile('./readme2.txt', (err,data) => {

if(err) throw err;

console.log('1번', data.toString());

fs.readFile('./readme2.txt', (err, data)=>{

if(err) throw err;

console.log('2번', data.toString());

fs.readFile('./readme2.txt', (err, data)=>{

if(err) throw err;

console.log('3번', data.toString());

});

});

});

console.log('끝');

C:\materK1m\newmimac\test\_kes\node\_basic>node asyncOrder

시작

끝

1번 저를 여러번 읽어보세요.

2번 저를 여러번 읽어보세요.

3번 저를 여러번 읽어보세요.

* 버퍼와 스트림

const buffer = Buffer.from('저를 버퍼로 바꿔보세요.');

console.log('from() : ', buffer);

console.log('length : ', buffer.length);

console.log('toString() : ', buffer.toString());

const array = [Buffer.from('띄엄 '), Buffer.from('띄엄 '), Buffer.from('띄어쓰기 ')];

const buffer2 = Buffer.concat(array);

console.log('concat() : ', buffer2.toString());

const buffer3 = Buffer.alloc(5);

console.log('alloc() : ', buffer3);

C:\materK1m\newmimac\test\_kes\node\_basic>node buffer

from() : <Buffer ec a0 80 eb a5 bc 20 eb b2 84 ed 8d bc eb a1 9c 20 eb b0 94 ea bf 94 eb b3 b4 ec 84 b8 ec 9a 94 2e>

length : 33

toString() : 저를 버퍼로 바꿔보세요.

concat() : 띄엄 띄엄 띄어쓰기

alloc() : <Buffer 00 00 00 00 00>

const fs = require('fs');

const writeStream = fs.createWriteStream('./writeme2.txt');

writeStream.on('finish', ()=>{

console.log('쓰기 완료');

});

writeStream.write('이 글을 씁니다. \n');

writeStream.write('한번 더 씁니다.');

writeStream.end();

C:\materK1m\newmimac\test\_kes\node\_basic>node createWriteStream

쓰기 완료

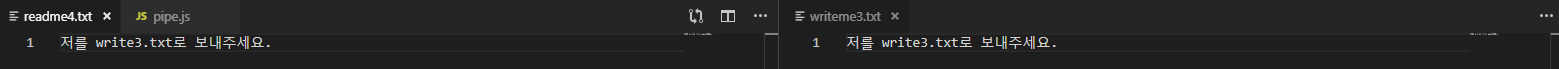
const fs = require('fs');

const readStream = fs.createReadStream('readme4.txt');

const writeStream = fs.createWriteStream('writeme3.txt');

readStream.pipe(writeStream);

C:\materK1m\newmimac\test\_kes\node\_basic>node pipe



const zlib = require('zlib');

const fs = require('fs');

const readStream = fs.createReadStream('./readme4.txt');

const zlibStream = zlib.createGzip();

const writeStream = fs.createWriteStream('./readme4.txt.gz');

readStream.pipe(zlibStream).pipe(writeStream);

C:\materK1m\newmimac\test\_kes\node\_basic>node gzip

* 기타 fs 메서드

const fs = require('fs');

/\*

fs.access(경로, 옵션, 콜백) : 폴더나 파일에 접근할 수 있는지를 체크합니다.

두 번째 인자로 상수들을 넣었습니다.

F\_OK는 파일 존재여부,

R\_OK는 읽기 권한여부,

W\_OK는 쓰기 권한여부를 체크합니다

파일/폴더나 권한이 없다면 에러가 발생하는데, 파일/폴더가 없을 때의 에러코드는 ENOENT입니다.

fs.mkdir(경로, 콜백) : 폴더를 만드는 메서드입니다. 이미 폴더가 있다면 에러가 발생하므로 먼저 access() 메서드를 호출해서 확인하는 것이 중요합니다.

fs.open(경로, 옵션, 콜백) : 파일의 아이디(fd 변수)를 가져오는 메서드입니다.

파일이 없다면 파일을 생성한 뒤 그 아이디를 가져옵니다.

가져온 아이디를 사용해 fs.read()나 fs.write()로 읽거나 쓸 수 있습니다.

두 번째 인자로 어떤 동작을 할 것인지 설정할 수 있습니다.

쓰려면 w, 읽으려면 r, 기존 파일에 추가하려면 a입니다.

예제에서는 w로 설정했으므로 파일이 없을 때 새로 만들 수 있었습니다.

fs.rename(기존경로, 새경로, 콜백) : 파일의 이름을 바꾸는 메서드입니다. 기존 파일 위치와 새로운 파일 위치를 적어주면 됩니다.

반드시 같은 폴더를 지정할 필요는 없으므로 잘라내기 같은 기능을 할 수도 있습니다.

\*/

fs.access('./folder', fs.constants.F\_OK | fs.constants.R\_OK | fs.constants.W\_OK, (err)=>{

if(err){

if(err.code == 'ENOENT'){

console.log('폴더없음');

fs.mkdir('./folder', (err)=>{

if(err) throw err;

console.log('폴더 만들기 성공');

fs.open('./folder/file.js','w',(err, fd)=>{

if(err) throw err;

console.log('빈 파일 만들기 성공', fd);

fs.rename('./folder/file.js', './folder/newfile.js', (err)=>{

if(err) throw err;

console.log('이름 바꾸기 성공!');

});

});

});

}else{

throw err;

}

}else{

console.log('폴더 이미 있음');

}

});

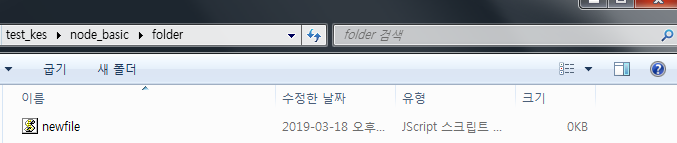
C:\materK1m\newmimac\test\_kes\node\_basic>node fsCreate

폴더없음

폴더 만들기 성공

빈 파일 만들기 성공 3

이름 바꾸기 성공!



const fs = require('fs');

/\*

fs.readdir(경로, 콜백) : 폴더 안의 내용물을 확인할 수 있습니다. 배열 안에 내부 파일과 폴더명이 나옵니다.

fs.unlink(경로, 콜백) : 파일을 지울 수 있습니다. 파일이 없다면 에러가 발생하므로 먼저 파일이 있는지를 꼭 확인해야 합니다.

fs.rmdir(경로, 콜백) : 폴더를 지울 수 있습니다. 폴더 안에 파일이 있다면 에러가 발생하므로 먼저 내부 파일을 모두 지우고 호출해야 합니다.

\*/

fs.readdir('./folder', (err, dir)=>{

if(err) throw err;

console.log('폴더 내용 확인', dir);

fs.unlink('./folder/newFile.js', (err)=>{

if(err) throw err;

console.log('파일 삭제 성공');

fs.rmdir('./folder',(err)=>{

if(err) throw err;

console.log('폴더 삭제 성공');

});

});

});

C:\materK1m\newmimac\test\_kes\node\_basic>node fsDelete

폴더 내용 확인 [ 'newfile.js' ]

파일 삭제 성공

폴더 삭제 성공

C:\materK1m\newmimac\test\_kes\node\_basic>node fsDelete

C:\materK1m\newmimac\test\_kes\node\_basic\fsDelete.js:10

if(err) throw err;

^

Error: ENOENT: no such file or directory, scandir 'C:\materK1m\newmimac\test\_kes\node\_basic\folder'

const fs = require('fs');

fs.copyFile('readme4.txt', 'writeme4.txt', (err)=>{

if(err) return console.error(err);

console.log('복사완료');

});

C:\materK1m\newmimac\test\_kes\node\_basic>node copyFile

복사완료

* 이벤트 이해하기

const EventEmitter = require('events');

const myEvent = new EventEmitter();

myEvent.addListener('event1',()=>{

console.log('이벤트1');

});

myEvent.on('event2',()=>{

console.log('이벤트 2');

});

myEvent.on('event2',()=>{

console.log('이벤트2 추가');

});

myEvent.emit('event1');

myEvent.emit('event2');

myEvent.once('event3', ()=>{

console.log('이벤트 3');

});

myEvent.emit('event3');

myEvent.emit('event3');

myEvent.on('event4', ()=>{

console.log('이벤트 4');

});

myEvent.removeAllListeners('event4');

myEvent.emit('event4');

const listener = () => {

console.log('이벤트5');

};

myEvent.on('event5',listener);

myEvent.removeListener('event5',listener);

myEvent.emit('event5');

console.log(myEvent.listenerCount('event2'));

/\*

on(이벤트명, 콜백) : 이벤트 이름과 이벤트 발생 시의 콜백을 연결해줍니다. 이렇게 연결하는 동작을 이벤트 리스닝이라고 합니다. event2처럼 이벤트 하나에 이벤트 여러 개를 달아줄 수 있습니다.

addListener(이벤트명, 콜백) : on과 기능이 같습니다.

emit(이벤트명) : 이벤트를 호출하는 메서드입니다. 이벤트 이름을 인자로 넣어주면 미리 등록해뒀던 이벤트 콜백이 실행됩니다.

once(이벤트명, 콜백) : 한번만 실행되는 이벤트입니다. myEvent.emit('event3')을 두 번 연속 호출했지만 콜백이 한 번만 실행됩니다.

removeAllListeners(이벤트명) : 이벤트에 연결된 모든 이벤트 리스너를 제거합니다. event4가 호출되기 전에 리스너를 제거했으므로 event4는 호출되지 않습니다.

removeListener(이벤트명, 리스너) : 이벤트에 연결된 리스너를 하나씩 제거합니다. 역시 event5의 콜백도 호출되지 않습니다.

off(이벤트명, 콜백) : 노드 10버전에서 추가된 메서드로, removeListener와 기능이 같습니다.

listenerCount(이벤트명) : 현재 리스너가 몇개 연결되어 있는지 확인합니다.

\*/

C:\materK1m\newmimac\test\_kes\node\_basic>node event

이벤트1

이벤트 2

이벤트2 추가

이벤트 3

2

* 예외처리하기

setInterval(()=>{

console.log('시작');

try {

throw new Error('서버를 고장내주마!');

} catch (error) {

console.error(error);

}

},1000)

C:\materK1m\newmimac\test\_kes\node\_basic>node error1

시작

Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error1.js:4:15)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)

시작

Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error1.js:4:15)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)

시작

Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error1.js:4:15)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)

시작

Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error1.js:4:15)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)

시작

Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error1.js:4:15)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)

^C

C:\materK1m\newmimac\test\_kes\node\_basic>node error1

시작

Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error1.js:4:15)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)

시작

Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error1.js:4:15)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)

시작

Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error1.js:4:15)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)

const fs = require('fs');

setInterval(()=>{

fs.unlink('./abcdefg.js', (err)=>{

if(err) console.error(err);

});

}, 1000);

C:\materK1m\newmimac\test\_kes\node\_basic>node error2

{ [Error: ENOENT: no such file or directory, unlink 'C:\materK1m\newmimac\test\_kes\node\_basic\abcdefg.js']

errno: -4058,

code: 'ENOENT',

syscall: 'unlink',

path: 'C:\\materK1m\\newmimac\\test\_kes\\node\_basic\\abcdefg.js' }

{ [Error: ENOENT: no such file or directory, unlink 'C:\materK1m\newmimac\test\_kes\node\_basic\abcdefg.js']

errno: -4058,

code: 'ENOENT',

syscall: 'unlink',

path: 'C:\\materK1m\\newmimac\\test\_kes\\node\_basic\\abcdefg.js' }

{ [Error: ENOENT: no such file or directory, unlink 'C:\materK1m\newmimac\test\_kes\node\_basic\abcdefg.js']

errno: -4058,

code: 'ENOENT',

syscall: 'unlink',

path: 'C:\\materK1m\\newmimac\\test\_kes\\node\_basic\\abcdefg.js' }

{ [Error: ENOENT: no such file or directory, unlink 'C:\materK1m\newmimac\test\_kes\node\_basic\abcdefg.js']

errno: -4058,

code: 'ENOENT',

syscall: 'unlink',

path: 'C:\\materK1m\\newmimac\\test\_kes\\node\_basic\\abcdefg.js' }

process.on('uncaughtException',(err)=>{

console.error('예기치 못한 에러',err);

});

setInterval(()=>{

throw new Error('서버를 고장내주마!');

},1000);

setTimeout(()=>{

console.log('실행됩니다.');

},2000)

C:\materK1m\newmimac\test\_kes\node\_basic>node error3

예기치 못한 에러 Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error3.js:6:11)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)

실행됩니다.

예기치 못한 에러 Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error3.js:6:11)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)

예기치 못한 에러 Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error3.js:6:11)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)

예기치 못한 에러 Error: 서버를 고장내주마!

at Timeout.setInterval [as \_onTimeout] (C:\materK1m\newmimac\test\_kes\node\_basic\error3.js:6:11)

at listOnTimeout (timers.js:324:15)

at processTimers (timers.js:268:5)