노드 내장모듈 사용하기 필기노트

2021년 1월 20일 수요일 오후 3:51

1. os
   운영체제의 정보를 담고 있음, 모듈은 require로 가져옴

```
const os = require('os');
console.log('운영체제 정보-------------------------');
console.log('os.arch():',os.arch());
console.log('os.platform():',os.platform());
console.log('os.type():',os.type());
console.log('os.uptime():',os.uptime());
console.log('os.hostname():',os.hostname());
console.log('os.release():',os.release());
console.log('경로---------------------------------');
console.log('os.homedir():',os.homedir());
console.log('os.tmpdir():',os.tmpdir());
console.log('cpu 정보---------------------------');
console.log('os.cpus():',os.cpus());
console.log('os.cpus().length:',os.cpus().length);
console.log('메모리 정보--------------------------');
console.log('os.freemem():',os.freemem());
console.log('os.totalmem():',os.totalmem());
```

```
os.arch(): x64
os.platform(): win32
os.type(): Windows_NT
os.uptime(): 404696
os.hostname(): DESKTOP-IGM1AKI
os.release(): 10.0.19042
경로---------------------------------
os.homedir(): C:\Users\jhn06
os.tmpdir(): C:\Users\jhn06\AppData\Local\Temp
cpu 정보---------------------------
os.cpus(): [
 {
  model: 'AMD Ryzen 5 2600 Six-Core Processor          ',
  speed: 3400,
  times: {
   user: 34646265,
   nice: 0,
   sys: 41187046,
   idle: 122954328,
   irq: 7119359
  }
 },
 {
  model: 'AMD Ryzen 5 2600 Six-Core Processor          ',
 },
]
os.cpus().length: 12
메모리 정보--------------------------
os.freemem(): 10260762624
os.totalmem(): 17126883328
```

나머지 코어의 정보

- os.cpus() : 컴퓨터의 코어 정보를 보여줍니다
  node는 싱글 스레드를 사용하므로 효율적으로 서버를 구현하려면 서버를 코어 개수만큼 돌리면 된다

2. path
   폴더와 파일의 경로를 쉽게 조작하도록 도와주는 모듈
   운영체제 별로 구분자가 다름 (Windows : '₩', POSIX : '/')

```
const path = require('path')
const string = __filename;
console.log('path.sep:',path.sep);
console.log('path.delmiter:',path.delimiter);
console.log("----------------------");
console.log("path.dirname():",path.dirname(string));
console.log("path.extname():",path.extname(string));
console.log("path.basename():",path.basename(string));
console.log("path.basename - extname:",path.basename(string, path.extname(strin
g)));
console.log("-------------------");
console.log("path.parse()",path.parse(string));
console.log("path.format():",path.format({
   dir: "E:\프로그래밍\Nodejs 교과서\3. 노드 기능",
   name: 'path',
   ext: '.js',
}));
console.log('path.normalize():',path.normalize("E:\프로그래밍\Nodejs 교과서\3. 노
드 기능\path.js"));
console.log("--------------------------");
console.log("path.isAbsolute(C:\\):",path.isAbsolute("C:\\"));
console.log("path.isAbsolute(./home):",path.isAbsolute("./home"));
console.log("--------------------------");
console.log("path.relative():",path.relative("E:\프로그래밍\Nodejs 교과서\3. 노
드 기능\path.js","E:\\"));
console.log("path.join()",path.join(__dirname,"..","..","/users",".","/jhn06"))
;
console.log("path.resolve():",path.resolve(__dirname,"..","users",".","/jhn06")
);
```

```
path.sep: \
path.delmiter: ;
----------------------
path.dirname(): e:\프로그래밍\Nodejs 교과서\3. 노드 기능\os와path
path.extname(): .js
path.basename(): path.js
path.basename - extname: path
-------------------
path.parse() {
 root: 'e:\\',
 dir: 'e:\\프로그래밍\\Nodejs 교과서\\3. 노드 기능\\os와path',
 base: 'path.js',
 ext: '.js',
 name: 'path'
}
path.format(): E:프로그래밍Nodejs 교과서. 노드 기능\path.js
path.normalize(): E:프로그래밍Nodejs 교과서. 노드 기능path.js
--------------------------
path.isAbsolute(C:\): true
path.isAbsolute(./home): false
```

```
--------------------------
path.relative(): ..\..\..
path.join() e:\프로그래밍\Nodejs 교과서\users\jhn06
path.resolve(): e:\jhn06
```

- path.join(경로, ...): 여러 인자를 넣으면 하나의 경로로 합쳐준다. 상대 경로인 ..(부모 디렉터리)와 .(현 위치)도 알아서 처리
- path.resolve(경로, ..): path.join()와 비슷하지만, join은 절대경로 무시, resolve는 절대 경로를 존중하여 앞의 것들이 무시
- path.sep : 경로의 구분자, 윈도우는 ₩, posix는 / 사용
- path.dirname(경로) : 폴더명 추출
- path.extname(경로) : 확장자 추출
- path.basename(경로, 확장자) : 파일의 이름을 확장자 포함하여 보여줌
- path.parse(경로) : 분해
- path.format(객체) : 합침
- path.normalize(경로) : /나 ₩를 실수로 여러 번 사용하였을 경우 정상 경로로 반환
- path.isAbsolute(경로) : 파일의 경로를 절대경로/상대경로인지 True/false로 반환
- path.relative(기준경로, 비교경로):경로를 두개 넣으면 첫 번째 경로에서 두 번째 경로로 가는 방법을 알려줌

3. url
   인터넷 주소를 쉽게 조작하도록 도와주는 모듈
   url처리에 크게 두 가지 방식이 있음 ->기존 노드 방식, WHATWG방식
   URL | Node.js v14.15.4 Documentation
   외우지 말고 옆에 두고 사용하면서 익히는것이 좋음

기존 Node

WHATWG

```
┌─────────────────────────────────────────────────────────────────────────────────────────────┐
│                                            href                                             │
├──────────┬──┬─────────────────────┬────────────────────┬───────────────────────────┬──────┤
│ protocol │  │        auth         │        host        │           path            │ hash │
│          │  │                     ├─────────────┬──────┼─────────────┬─────────────┤      │
│          │  │                     │   hostname  │ port │  pathname   │   search    │      │
│          │  │                     │             │      │             ├┬────────────┤      │
│          │  │                     │             │      │             ││   query    │      │
"  https:   //    user   :   pass   @ sub.example.com : 8080   /p/a/t/h  ?  query=string   #hash "
│          │  │                     │   hostname  │ port │             │             │      │
│          │  │                     ├─────────────┴──────┤             │             │      │
│ protocol │  │ username │ password │        host        │             │             │      │
├──────────┴──┼──────────┴──────────┼────────────────────┤             │             │      │
│   origin    │                     │       origin       │  pathname   │   search    │ hash │
├─────────────┴─────────────────────┴────────────────────┴─────────────┴─────────────┴──────┤
│                                            href                                             │
└─────────────────────────────────────────────────────────────────────────────────────────────┘
(All spaces in the "" line should be ignored. They are purely for formatting.)
```

```
const url = require('url');
const {URL} = url;
const myURL = new URL('http://www.gilbut.co.kr/book/bookList.aspx?sercate1=001001000#anchor');
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#anchor');
console.log('url.parse():',parsedUrl);
console.log('url.format():',url.format(parsedUrl));
```

WHATWG

```
new URL(): URL {
  href: 'http://www.gilbut.co.kr/book/bookList.aspx?sercate1=001001000#anchor',
  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: '#anchor'
}
url.format(): http://www.gilbut.co.kr/book/bookList.aspx?sercate1=001001000#anchor
----------------------
url.parse(): Url {
```

```
url.format(): http://www.gilbut.co.kr/book/bookList.aspx?sercate1=001001000#anchor
----------
url.parse(): Url {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'www.gilbut.co.kr',
  port: null,
  hostname: 'www.gilbut.co.kr',
  hash: '#anchor',
  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#anchor'
}
url.format(): http://www.gilbut.co.kr/book/bookList.aspx?sercate1=001001000#anchor
```

- 기존 모듈 메서드

  url.parse(주소) : 주소를 분해, WHATWG방식과 비교하면 username과password 대신 auth속성, searchParams 대신 query

  url.format(객체) : WHATWG방식의 url과 기존 url 모두 가능, 분해된 url을 원래대로 조립

- searchParams

  WHATWG 방식에서 쿼리 스트링(search)부분 처리를 도와주는 객체

  ?page=3&limit=10&category=nodejs&category=javascript

```
const {URL} = require('url');
const myURL = new URL('http://www.gilbut.co.kr/?page=3&limit=10&category=nodejs&category=javascript');
console.log('searchParams:',myURL.searchParams);
console.log('searchParams.getAll():',myURL.searchParams.getAll('category'));
console.log('searchParams.get():',myURL.searchParams.get('limit'));
console.log('searchParams.has():',myURL.searchParams.has('page'));
console.log('searchParams.keys():',myURL.searchParams.keys());
console.log('searchParams.valuse():',myURL.searchParams.values());
myURL.searchParams.append('filter','es3');
myURL.searchParams.append('filter','es5');
console.log(myURL.searchParams.getAll('filter'));
myURL.searchParams.set('filter','es6');
console.log(myURL.searchParams.getAll('filter'));
myURL.searchParams.delete('filter');
console.log(myURL.searchParams.getAll('filter'));
console.log('searchParams.toString():',myURL.searchParams.toString());
myURL.search = myURL.searchParams.toString();
```

```
searchParams: URLSearchParams {
  'page' => '3',
  'limit' => '10',
  'category' => 'nodejs',
  'category' => 'javascript' }
searchParams.getAll(): [ 'nodejs', 'javascript' ]
searchParams.get(): 10
searchParams.has(): true
searchParams.keys(): URLSearchParams Iterator { 'page', 'limit', 'category', 'category' }
searchParams.valuse(): URLSearchParams Iterator { '3', '10', 'nodejs', 'javascript' }
[ 'es3', 'es5' ]
[ 'es6' ]
[]
searchParams.toString(): page=3&limit=10&category=nodejs&category=javascript
```

- queryString

  기존 노드 방식에서 쿼리 스트링(search)부분 처리르 ㄹ도와주는 객체

  querystring.parse(쿼리) : url의 query부분을 자바스크립트 객체로 분해해준다

  querystring.stringify(객체) : 분해된 query객체를 문자열로 다시 조립

4. crypto(단방향 암호화)

   암호화는 가능, 복호화는 불가능

   암호화 : 평문을 암호로 만듦　　복호화 : 암호를 평문으로 해독

   단방향 암호화의 대표 주자는 해시 기법

   abcdefgh문자열 -> qvwv

비밀번호	← X	해시 함수	⇄	다이제스트

   해시는 복호화가 안되지만, 해시 함수는 항상 그 단어를 동일하게 변경시킨다

5. Hash 사용하기

- createHash(알고리즘) : 사용할 해쉬 알고리즘을 넣어준다

  md5, sha1, sha256, sha512 등이 가능하지만, sha1와 md5는 취약점이 발견됨

  현재는 sha512로 충분하지만 추후에 더 강화된 알고리즘으로 넣어야 함

- update(문자열) : 변환할 문자열을 넣어준다
- digest(인코딩) : 인코딩할 알고리즘을 넣어준다

```
const crypto = require('crypto');
```

```
base64:
```

```
console.log('base64:',crypto.createHash('sha512')
.update('비밀번호').digest('base64'));
console.log('hex:',crypto.createHash('sha512').up
date('비밀번호').digest('hex'));
console.log('base64:',crypto.createHash('sha512')
.update('다른 비밀번호').digest('base64'));
```

```
dvfV6nyLRRt3NxKSlTHOkkEGgqW2HRtfu19Ou/psUXvwlebbXCboxIPmDYOFRIpqav2eUTBFu
HaZri5x+usy1g==
hex:
76f7d5ea7c8b451b773712929531ce92410682a5b61d1b5fbb5f4ebbfa6c517bf095e6db5c2
6e8c483e60d8385448a6a6afd9e513045b87699ae2e71faeb32d6
base64:
cx49cjC8ctKtMzwJGBY853itZeb6qxzXGvuUJkbWTGn5VXAFbAwXGEOxU2Qksoj+aM2GWPhc
1O7mmkyohXMsQw==
```

5-1. ~~pbkdf2~~ → db에 pwd, salt 둘다 암호화해야함

컴퓨터의 발달로 기존 암호화 알고리즘이 위협받고 있음

sha512가 취약해지면 sha3으로 넘어가야 함, 현재는pbkdf2나 bcrypt, scrypt 알고리즘으로 비밀번호 암호화

Nodejs는 pbkdf2와 scrypt2를 지원함

```
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'));
    });
});
```

↳ 소금, sha512 해쉬도 더 어렵게함

```
salt: HIXT6rZfea4Gi7zOv61mpZ7su+QdR3tfyBjcQk/T3
+pfsqHtOu4YD07y24xwjrPXM8Bh7n19dBGpTNG7EzUC7g==
password: N5EBD2mNxrgNbZ3MHahFbcTk3lTGBAAd2LLMgwxtOV2MMw/0j44
+J1tPzyrGJpJBJUKpJoSFr1jmfrLXNEMeAg==
```

6. 양방향 암호화 → Aes 추천 단방향암호화 → Res 추천 / 그냥 Aws kms 등을 사용

대칭형 암호화(암호문 복호화 가능) -> Key가 사용됨, 암호화할 때와 복호화할 때 같은 Key를 사용해야 함

```
const crypto = require('crypto');
const algorithm = 'aes-256-cbc';
const key = 'abcdefghijklmnopqrstuvwxyz123456'
const iv = '1234567890123456';
const cipher = crypto.createCipheriv(algorithm, key, iv);
let result = cipher.update('암호화할 문장','utf8','base64');
result += cipher.final('base64');
console.log('암호화 : ',result);
const decipher = crypto.createDecipheriv(algorithm, key, iv);
let result2 = decipher.update(result, 'base64', 'utf8');
result2 += decipher.final('utf8');
console.log('복호화 : ',result);
```

```
암호화 : iiopeG2GsYlk6ccoBoFvEH2EBDMWv1kK9bNuDjYxiN0=
복호화 : iiopeG2GsYlk6ccoBoFvEH2EBDMWv1kK9bNuDjYxiN0=
```

- crypto.createCipheriv(알고리즘, 키, iv) : 암호화 알고리즘과 키, 초기화백토를 넣어준다
- cipher.update(문자열, 인코딩, 출력 인코딩) : 암호화할 대상과 대상의 인코딩, 출력 결과물의 인코딩을 넣어준다
- cipher.final(출력 인코딩) : 출력 결과물의 인코딩을 넣어주면 암호화 완료
- crypto.createDecipheriv(알고리즘, 키, iv) : 복호화할 때 사용
- decipher.update(문자열, 인코딩, 출력 인코딩) : 암호화된 문장, 그 문장의 인코딩, 복호화할 인코딩을 넣어준다
- decipher.final(출력 인코딩) : 복호화 결과물의 인코딩을 넣어준다

7. util

각종 편의 기능을 모아둔 모듈, 주로 deprecated와 promisify가 자주 쓰임

```
const util = require('util');
const crypto = require('crypto');
const dontUseMe = util.deprecate((x, y) => {
    console.log(x + y);
}, 'dontUseMe 함수는 deprecated 되었으니 더이상 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);
    });
```

```
3
(node:29952) DeprecationWarning: dontUseMe 함수는 deprecated 되었으니 더이상 사용하지 마세요
q0CZ9Zbahkc1tu7fVrYmWDzDcNUjaJ0Yr7cw2ruWEtwsfkdR642Y4Smo0R/I/NCXrfBbCTjlRXmPQK2U/GNViA==
```

- utll.deprecate : 함수가 deprecated 처리되었음을 알려줌

  첫 번째 인자 : 함수를 사용할 때 경고 메시지 출력

두 번째 인자 : 출력될 경고 메시지 입력
함수가 조만간 사라지거나 변경될 때 알려줄 수 있어 유용

- util.promisify :콜백 패턴을 프로미스 패턴으로 바꿔준다
  바꿀 함수를 인자로 제공하면 됨. 이렇게 바꾸면 async/await 패턴을 사용 가능해 유용

8. worker_threads
   노드에서 멀티 스레딩 방식을 사용하기 위한 모듈
   isMainThread : 현재 코드가 메인 스레드에서 실행되는지, 워커 스레드에서 실행되는지 구분
   메인 스레드에서는 new Worker를 통해 현재 파일(__filename)을 워커 스레드에서 실행시킴

```
const { Worker, isMainThread, parentPort } = require('worker_threads');
// 메인 스레드
// 메인 스레드에서 일을 여러개 워커 스레드로 분배를 해주고,
// 그것을 합치는 것도 직접 해야 함
if(isMainThread){
    const worker = new Worker(__filename);
    worker.on('message',(value)=> console.log('워커로부터', value));
    worker.on('exit', () => console.log('워커 끝~'));
    worker.postMessage('ping');
}
// 워커 스레드
else{
    parentPort.on('message', (value)=>{
        console.log('부모로부터', value);
        parentPort.postMessage('pong');
        parentPort.close();
    })
}
```

```
부모로부터 ping
워커로부터 pong
워커 끝~
```

스레드 여러 개

```
const { Worker, isMainThread, parentPort, workerData } = require('worker_threads');
const {worker} = require('cluster');
// 메인 스레드
// 메인 스레드에서 일을 여러개 워커 스레드로 분배를 해주고,
// 그것을 합치는 것도 직접 해야 함
if(isMainThread){
    const threads = new Set();
    threads.add(new Worker(__filename, {
        workerData: { start: 1 },
    }));
    threads.add(new Worker(__filename, {
        workerData: { start: 2 },
    }));
    for( let worker of threads){
        worker.on('message',(value)=> console.log('워커로부터', value));
        worker.on('exit', () => {
            threads.delete(worker);
            // 일이 마무리 됨을 알려줌
            if(threads.size === 0){
                console.log('워커 끝~');
            }
        });
    }
}
// 워커 스레드
else{
    const data = workerData;
    parentPort.postMessage(data.start + 100);
}
```

```
워커로부터 101
워커로부터 102
워커 끝~
```

- 워커스레드를 사용하지 않고 2~10,000,000까지 소수 찾기

```
const min  = 2;
const max = 10_000_000;
const primes = [];
function generatePrimes(start, range){
    let isPrime = true;
    const end = start + range;
    for (let i = start; i < end; i++){
        for(let j = min; j < Math.sqrt(end); j++){
            if(i !== j && i % j === 0){
```

```
prime: 13.571s
664579
```

```
            isPrime = false;
            break;
        }
    }
    if(isPrime){
        primes.push(i);
    }
    isPrimes = true;
  }
}
console.time('prime');
generatePrimes(min, max);
console.timeEnd('prime');
console.log(primes.length);
console.log(primes);
```

- 워커스레드를 사용해서 2~10,000,000까지 소수 찾기

```
const { Worker, isMainThread, parentPort, workerData } = require('worker_threads');
const min = 2;
let primes = [];
function findPrimes(start, range){
  let isPrime = true;
  const end = start + range;
  for (let i = start; i < end; i++){
    for(let j = min; j < Math.sqrt(end); j++){
      if(i !== j && i % j === 0){
        isPrime = false;
        break;
      }
    }
    if(isPrime){
      primes.push(i);
    }
    isPrime = true;
  }
}
if(isMainThread){
  const max = 10_000_000;
  // 워커 스레드는 각각의 함수에 각각 분배를 해주어야 한다
  const threadCount = 8;
  const threads = new Set();
  const range = Math.ceil((max - min) / threadCount);
  let start = min;
  console.time('prime');
  for(let i = 0; i < threadCount - 1; i++){
    const wStart = start;
    threads.add(new Worker(__filename, {workerData: {start: wStart, range}}));
    start += range;
  }
  threads.add(new Worker(__filename, {workerData: {start, range: range + ((max - min + 1) % threadCount) }}));
  for (let worker of threads){
    worker.on('error', (err) => {
      throw err;
    });
    worker.on('exit', () => {
      threads.delete(worker);
      if (threads.size === 0){
        console.log("dd");
        console.timeEnd('prime');
        console.log(primes.length);
      }
    });
    // 워커의 결과를 합침
    worker.on('message', (msg)=>{
      primes = primes.concat(msg);
    });
  }
}
else{
  findPrimes(workerData.start, workerData.range);
  parentPort.postMessage(primes);
}
```

```
prime: 2.919s
664579
```

결론 : 스레드를 사용하는 것이 효율은 더 좋음

하지만 어려우므로, Node 말고 다른 언어에서 쓰자

9. child_process

노드에서 다른 프로그램을 실행하고 싶거나 명령어를 수행하고 싶을 때 사용

현재 노드 프로세스 외에 새로운 프로세스를 띄워 명령을 수행

명령 프롬포트의 명령어인 dir를 노드를 통해 실행(리눅스는 ls를 대신 적는다)

```
const exec = require('child_process').exec;
var process = exec('cmd /c chcp 65001>nul && dir');
process.stdout.on('data',function(data){
    console.log(data.toString());
})
process.stdout.on('data',function(data){
    console.error(data.toString());
})
```

```
2021-01-20  오후 04:43    <DIR>          .
2021-01-20  오후 04:43    <DIR>          ..
2021-01-12  오후 07:37    <DIR>          .vscode
2021-01-11  오후 09:40    <DIR>          1. 노드 시작하기
2021-01-13  오후 08:32    <DIR>          2. 알아두어야 할 Javascript
2021-01-24  오후 11:15    <DIR>          3. 노드 기능
2021-01-19  오전 10:03               232 README.md
               1 File(s)            232 bytes
               6 Dir(s)  1,538,113,421,312 bytes free
```

```
const spawn = require('child_process').spawn;
const process = spawn('python',['test.py']);
process.stdout.on('data',function(data){
    console.log(data.toString());
    console.log("dd");
})
process.stdout.on('data',function(data){
    console.error(data.toString());
})
```

```
hello python
```

단, 파이썬 등을 실행하려면 컴퓨터에 파이썬 등이 깔려 있어야 함

노드가 직접 실행하는 것이 아니라 대신 실행해주는 것