

민건호 Node 스터디 정리

Node

node.js를 사용하는 이유

- : 브라우저환경과 컴퓨터환경 두마리 토끼를 잡을 수 있다.
- : 프론트와 백엔드 둘다 같은 언어로 구현 가능하다.
- : 50퍼 이상이 node.js를 사용하고 있다.
- : 많은 대기업들이 node.js를 사용하고 있다.
- : 많은 커뮤니티와 툴을 가지고 있고 성능은 이미 검증을 받았다.
- : 쉽게 접근가능하고 좋은 생산성을 가지고 있다.

Node의 4가지 포인트

1. javascript runtime
2. single Thread
3. non-blocking i/o
4. event-driven

크롬 웹 브라우저는 v8 엔진이 탑재된 자바스크립트 런타임이다.

- ? 런타임이란 프로그래밍언어가 구동되는 환경을 말한다.
- ? 자바스크립트 런타임이란 자바스크립트가 구동되는 환경을 말한다.
- ? 이러한 자바스크립트 런타임의 종류로는 웹브라우저(크롬, 파이어폭스 등)프로그램과 Node.js라는 프로그램이 있다.
- ? 이러한 프로그램들에서 자바스크립트가 구동되기 때문에 자바스크립트 런타임이라고 한다.

- ? v8이란 자바스크립트 엔진 중 하나이며 오픈 소스 자바스크립트 엔진 중 하나이다.
- ? 크롬 웹 브라우저와 Node.js등에서 사용되고 있다.
- ? 자바스크립트와 웹어셈블리 엔진이다.
 - > 웹어셈블리란 C나 C++와 같은 프로그래밍 언어를 컴파일해서 어느 브라우저에서나 빠르게 실행되는 형식으로 바꿔주는 기술을 뜻한다. (자바스크립트를 대체하는 것이 아니라 보완하는 기술)

자바스크립트는 싱글 스레드이다.

- ? 자바스크립트는 전통적으로 단일 스레드이다.
- ? 코어가 여러 개 있어도 메인 스레드라고 하는 단일 스레드에서만 작업을 행할 수 있다.

EX)

```
function SignThreadTest() {
  console.log(1);
  for( let i=0; i<10000; i++ ) {
    console.log('자바스크립트는 싱글 스레드 프로그래밍 언어이다. ');
  }
  console.log(2);
}
// '자바스크립트는 싱글 스레드 프로그래밍 언어이다.'가 10,000번 실행이 완료 될 때까지
// 콘솔 창에 '2' 이 찍히는 것을 기다려야한다.
SignThreadTest();
```

? 위 예시 코드와 같이 앞에 일이 완료 될 때까지 다음 코드는 다른 일은 하지 못하고 기다려야 해서 웹사이트에서 다른일을 할 수 없게 된다.
? 이렇게 싱글 스레드는 일을 동기적으로 처리한다.

비동기 콜백으로 싱글 스레드 프로그래밍 언어에서의 블로킹을 해결할 수 있다. -> 논 블로킹

? Node.js에는 동기로 처리하는 블로킹 메소드가 거의 없다.
> 대부분 지원 메소드가 비동기 처리를 하게끔 만들어졌다.
? 그렇다면 Node.js에서 요청을 비동기로 어떻게 처리할 수 있는 것일까
> 이는 어떤 코드를 실행하면 결국 콜백을 받고 이걸 나중에 실행한다는 말이다.
> 자세하게 풀어서 이야기하면 함수 호출 시 당장 실행하는 것이 아니라(동기->블로킹) 일단 어느 곳에 쌓아놓고 동시에 요청을 처리하고(비동기->논 블로킹) 요청이 완료된 순서대로 처리(스택 이용) 한다는 말이다.
? 콜백함수는 간단하게 다른 함수에 매개변수로 넘겨준 함수를 말한다.
> 매개변수로 넘겨받은 함수는 일단 넘겨받고, 때가 되면 나중에 호출한다는 것이다.

```
EX)
function checkGit(count, link, good) {
  count < 3 ? link() : good();
}

function linkGit() {
  console.log('링크를 들어가주세요 !');
  console.log('https://github.com/mingeonho1');
}

function goodGit() {
  console.log('오늘 할당량은 모두 채우셨습니다! :)')
}

check(2, linkGit, goodGit);
```

? 위처럼 예를 들자면 이런 것이다.
코드를 살펴보면 checkGit, linkGit, goodGit 총 3가지 함수를 선언하고

checkGit 함수를 호출할 때 매개변수로 count에 숫자값을,

그리고 link와 good에 각각 linkGit과 goodGit함수를 전달했다.

여기서 linkGit함수와 goodGit함수가 콜백함수 인 것이다.

checkGit함수가 먼저 호출되고, 매개변수로 들어온 count의 값에 따라 linkGit과 goodGit함수 둘 중 한 가지가 나중에 호출된다.

위 코드는 count가 2이기 때문에 linkGit이 실행된다.

이벤트 기반(Event-Driven)


> Event-Driven은 non-blocking i/o 와 밀접한 관계가 있는데 >

? 이벤트 기반이란 이벤트가 발생할 때 미리 지정해 둔 작업을 수행하는 것을 의미한다.

EX) 콜백을 던져주고나서 "파일을 다 읽으면 (파일을 다 읽어지는 이벤트가 발생하면) 콜백을 호출해" 라고 이벤트를 통해서 내가 등록한 콜백을 호출할 수 있게 해준다.

GitHub - mingeonho1/Node.js

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

 <https://github.com/mingeonho1/Node.js>

mingeonho1/
Node.js



 1 Contributor
 0 Issues
 0 Stars
 0 Forks



Node Modules

global

global은 전역 객체로, node 환경 내에서 모든 파일에서 접근 가능하고 내부의 메서드를 global 표기를 생략하고 사용이 가능하다. 내부에 message, console ...

console

```
console.log('logging...');  
console.clear(); // 콘솔창을 다 지워줌
```

// log level

```
console.log('log'); // 개발  
console.info('info'); // 정보  
console.warn('warn'); // 경고  
console.error('error'); // 에러, 사용자 에러, 시스템 에러
```

// assert

```
console.assert(2 === 3, 'not same!'); // True가 아닌경우에만 출력  
console.assert(2 === 2, 'same!');
```

// print object

```
const student = {name:'geonho', age: 22, company : {name: 'unionc'}};  
console.log(student);  
console.table(student); // 테이블로 보기좋게 출력 됨  
console.dir(student, {showHidden: true, colors: false, depth: 0 }); //옵션을 줄 수 있음
```

// measuring time

```
console.time('for loop');  
for(let i=0; i<10; i++){  
    console.log(i);  
}  
console.timeEnd('for loop'); // 걸린시간 측정
```

// counting

```
function a() {  
    console.count('a function'); // 함수가 몇번 호출된지 카운트  
}
```

```
a();  
a();
```

```
console.countReset('a function'); // 카운트 초기화
```

// trace

```
function f1(){
    f2();
}
function f2(){
    f3();
}
function f3(){
    console.log('f3');
    console.trace();    // 어디서 함수를 호출했는지 알려줌    // 디버깅할 때 좋음
}
f1();
```

```
# module

const counter = require('./counter.js');

counter.increase();
counter.increase();
counter.increase();
console.log(counter.getCount());
```

```
let count = 0;

function increase(){
    count++;
}

function getCount(){
    return count;
}

module.exports.getCount = getCount;
module.exports.increase = increase;
```

```
import * as counter from './counter.js';

counter.increase();
counter.increase();
counter.increase();
console.log(counter.getCount());
```

```
let count = 0;

export function increase(){
    count++;
}

export function getCount(){
    return count;
}
```

```
# os

const os = require('os');

console.log(os.EOL === '\n');    // 맥 줄바꿈
console.log(os.EOL === '\r\n');    // 윈도우 줄바꿈

console.log(os.totalmem());
console.log(os.freemem());
console.log(os.type());
console.log(os.userInfo());
console.log(os.cpus());
```

```

console.log(os.homedir());
console.log(os.hostname());

// 다양한 운영체제
// 내 서버가 동작하고 있는 그 환경에 대한 정보를 얻어올 때 os모듈을 사용함.

# process

const process = require('process');

console.log(process.execPath); // 현재 실행되고 있는 노드의 위치
console.log(process.version); // 노드의 버전
console.log(process.pid); // 프로세스 아이디
console.log(process.ppid); // 프로세스 부모의 아이디
console.log(process.platform); // 플랫폼에 대한 정보들
console.log(process.env); // 현재 컴퓨터에 저장된 환경변수의 모든 정보들
console.log(process.uptime());
console.log(process.cwd());
console.log(process.cpuUsage());

setTimeout(() => {
  console.log('setTimeout'); // 모든 코드가 다 끝난후에 0초뒤에 setTimeout 실행
}, 0);

process.nextTick(()=>{
  console.log('nextTick'); // process.nextTick()은 액션의 실행을 이벤트루프의 다음
}); // 차례까지 실제로 연기한다.

// nextTick()은 Task큐에 다른 콜백함수가 들어있어도 그 순서를 무시하고 이 콜백함수를 제일
// 우선순위를 높여서 Task큐 제일 앞부분에다가 넣어준다.

for(i=0; i<1000; i++){
  console.log("for loop");
}

// 실행결과

/*
C:\Program Files\nodejs\node.exe
v16.14.2
13032
26424
win32
{
  ALLUSERSPROFILE: 'C:\\ProgramData',
  APPDATA: 'C:\\Users\\UNI-S\\AppData\\Roaming',
  APPLICATION_INSIGHTS_NO_DIAGNOSTIC_CHANNEL: 'true',
  CHROME_CRASHPAD_PIPE_NAME: '\\\\.\\pipe\\crashpad_28484_MZAWXFVVIEVAJBZI',
  .
  .
  .
  VSCODE_NLS_CONFIG: '{"locale":"ko","availableLanguages":{"*":"ko"},"_languagePackI
d":"738ac08552cca045871a73764b8c446f.ko","_translationsConfigFile":"C:\\\\Users\\UNI
-S\\AppData\\Roaming\\Code\\cpl\\738ac08552cca045871a73764b8c446f.ko\\tcf.
json","_cacheRoot":"C:\\\\Users\\UNI-S\\AppData\\Roaming\\Code\\cpl\\738ac
08552cca045871a73764b8c446f.ko","_resolvedLanguagePackCoreLocation":"C:\\\\Users\\UN
I-S\\AppData\\Roaming\\Code\\cpl\\738ac08552cca045871a73764b8c446f.ko\\c35
11e6c69bb39013c4a4b7b9566ec1ca73fc4d5","_corruptedFile":"C:\\\\Users\\UNI-S\\AppDa

```

```

ta\\\\Roaming\\\\Code\\\\clp\\\\738ac08552cca045871a73764b8c446f.ko\\\\corrupted.inf
o", "_languagePackSupport": true}',
  VSCODE_PID: '28484',
  VSCODE_PIPE_LOGGING: 'true',
  VSCODE_VERBOSE_LOGGING: 'true',
  windir: 'C:\\WINDOWS'
}
0.0763542
c:\\Users\\UNI-S\\Desktop\\mgh\\Node.js
{ user: 15000, system: 46000 }
for loop
for loop
for loop
for loop
.
.
.
for loop
for loop
for loop
nextTick
setTimeout
*/

# timer

let num = 1;

const interval = setInterval(() =>{
  console.log(num++);
}, 1000);

setTimeout(() => {
  console.log("Timeout!");
  clearInterval(interval);
}, 6000);

// 6초 뒤에 interval을 멈추게 함.

# path

// dirname
console.log(path.dirname(__filename)); // 디렉토리 이름

//extension
console.log(path.extname(__filename)); // 확장자만

// parse
const parsed = path.parse(__filename);
console.log(parsed); // 오브젝트 형태로 하나씩
parsed.root
parsed.name

const str = path.format(parsed);
console.log(str); // 스트링 형태로

// isAbsolute
console.log('isAbsolute?', path.isAbsolute(__dirname)); // 절대경로

```

```

console.log('isAbsolute?', path.isAbsolute('../')); // 상대경로

// normalize
console.log(path.normalize('../folder/////sub')); // 이상한 경로를 알아서 고쳐줌

// join
console.log(__dirname + path.sep + 'image');
console.log(path.join(__dirname, 'image')); // 조인을 사용하면 자동으로 경로를 만들어줌

// 경로를 직접 넣기보다는 path를 사용해서 운영체제별로 경로를 만드는데 좋음

# file

const { error } = require('console');
const fs = require('fs');

// 3
// rename(..., callback(error, data))
// try { renameSync(...)} catch(e){}
// promises.rename().then().catch(0)

try {
  fs.renameSync('4-10-file/text.txt', '4-10-file/text-new.txt'); // 동기
} catch (error) {
  console.log(error);
}

fs.rename('4-10-file/text-new.txt', '4-10-file/text.txt', (error) => { // 비동기
  console.log('ERROR : ' + error);
});

fs.promises
  .rename('4-10-file/text2.txt', '4-10-file/text-new.txt')
  .then(() => console.log('Done!'))
  .catch(console.error(error));

/*
  동기적 처리
  : 일을 처리할 때 순서대로 처리하는 것이다.
  비동기적 처리와 비교했을 때 효율이 떨어진다.
  구현이 간단하다.
*/

/*
  비동기적 처리
  : 일을 처리할 때 병렬적으로 동시에 처리하는 것이다.
  동기적 처리와 비교했을 때 효율이 높다.
  구현이 복잡하다.
*/

const fs = require('fs').promises;

// read a file
fs.readFile('4-10-file/text.txt', 'utf-8') // 파일 읽기
  .then((data) => console.log(data))
  .catch(console.error);

```

```
// writing a file1
fs.writeFile('4-10-file/text.txt', 'Hello! Geon HO ') // 텍스트 쓰기
.catch(console.error);

fs.appendFile('4-10-file/text.txt', 'YO! Geon HO ') // 텍스트 추가
.then((data) => {
  fs.copyFile('4-10-file/text.txt', '4-10-file/text2.txt') // 비동기 처리이기 때문에
  텍스트를 추가하고 복사
  .catch(console.error);
})
.catch(console.error);

// copy
fs.copyFile('4-10-file/text.txt', '4-10-file/text2.txt') // 파일 복사
.catch(console.error);

// folder
fs.mkdir('sub-folder') // 폴더생성
.catch(console.error);

fs.readdir('.') // 현재 경로에 있는 모든 폴더를 읽음
.then(console.log)
.catch(console.error);

// api를 사용할 때 인자는 어떤 것들이 있는지
// 추가적으로 전달해야하는 옵션과 같은 인자들이 있는지
// 함수에서 리턴되는 값은 무엇인지
// promissess 라면 catch를 이용해서 에러를 잡는것이 중요함 !

# buffer와 stream은 친구

서버 -----> 사용자
      mp4

: 서버가 사용자에게 동영상 파일을 보내주고 또 사용자가 동영상 파일을 다운로드 받을 때 까지
다 기다렸다가 동영상을 보게되면 오랜시간이 걸리게 된다
> 이와는 다르게 서버에서 동영상 전체가 아닌 잘게잘게 나뉜진 데이터를 조금씩 보내주는
것, 이것을 스트리밍이라고 한다.
= Progressive Download

: 사용자가 동영상을 보는 속도보다 조금씩 다운로드 받는 속도가 더 빠르다면 버퍼링을 이용해서
조금조금씩 더 버퍼를 채워넣을 수 있다.
> 반대로 사용자가 동영상을 보는 속도가 더 빠르다면 충분히 쌓여있는 버퍼가 없기 때문에
우리가 버퍼링에 걸렸다고 하는 그런 것이다.

const fs = require('fs');
const data = [];

fs.createReadStream('./file.txt', { // 스트림은 조금조금씩 읽어오기 때문에 이벤트 베이스!
  // highWaterMark: 8, // 기본값 64 kbytes
  encoding: 'utf-8',
}).on('data', chunk => {
  console.log(chunk);
  data.push(chunk);
  // console.count('data');
}).on('end', () => {
  console.log(data.join(''));
}).on('error', error => {
```



```
    console.log(error);  
  });
```