

K-Digital Training 웹 풀스택 과정

Node.js 기초

Node.js란?



- 구글 크롬의 자바스크립트 엔진 (V8 Engine) 에 기반해 만들어진 Javascript 런타임
- 이벤트 기반, 비동기 I/O 모델을 사용해 가볍고 효율적
- npm 패키지는 세계에서 가장 큰 오픈 소스 라이브러리

런타임이란?

- 프로그래밍 언어가 구동되는 환경



- javascript의 런타임 환경은 웹 브라우저만 존재 했었음.
 - javascript 를 서버단 언어로 사용하기 위해 나온 것이 node.js
 - 웹 브라우저 없이 실행 가능

왜 배워야할까?

- 풀 스택 자바스크립트: 프론트엔드와 백엔드 모두 자바스크립트 사용으로 생산성 향상.
 - 높은 성능: 비동기 이벤트 기반 모델로 실시간 애플리케이션에 최적화.
 - 활발한 커뮤니티: 풍부한 오픈 소스 라이브러리와 모듈 제공 (NPM).
 - 빠른 학습 곡선: 자바스크립트를 알고 있다면 쉽게 학습 가능.
 - 크로스 플랫폼: 다양한 운영 체제에서 실행 가능, 코드 이식성 높음.
- => 앞으로 사용하게 될 모든 js들은 node.js 기반으로 관리됨

Node.js 설치

Node.js 설치 - 윈도우

[Node.js \(nodejs.org\)](https://nodejs.org)




다운로드


최신 LTS 버전: 16.16.0 (includes npm 8.11.0)


플랫폼에 맞게 미리 빌드된 Node.js 인스톨러나 소스코드를 다운받아서 바로 개발을 시작하세요.

LTS
대다수 사용자에게 추천

현재 버전
최신 기능


Windows Installer
node-v16.16.0-x64.msi


macOS Installer
node-v16.16.0.pkg


Source Code
node-v16.16.0.tar.gz

Windows Installer (.msi)

Windows Binary (.zip)

| | |
|--------|--------|
| 32-bit | 64-bit |
| 32-bit | 64-bit |

Node.js 설치 – MAC

1. HomeBrew 설치

https://brew.sh/index_ko 접속

2. Node js 설치

```
brew install node
```


Node.js 설치 - 버전확인

node -v
npm -v

```
C:\Users\>node -v
v16.17.1

C:\Users\>npm -v
8.7.0

C:\Users\>
```

npm 이란?

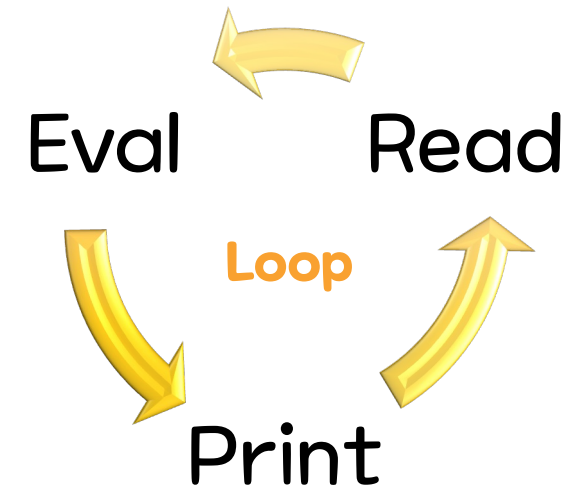
- Javascript로 개발된 각종 모듈의 설치, 업데이트, 구성, 제거 과정을 자동화하여 관리해주는 기능

REPL

R(Read) → E(Evaluate) → P(Print) → L(Loop)

윈도우에서의 cmd, 맥에서의 terminal처럼
노드에는 REPL 콘솔이 있다.

```
C:\Users\inda>node
Welcome to Node.js v12.22.12.
Type ".help" for more information.
>
```



REPL 사용하기

```
C:\Users\linda>node
Welcome to Node.js v12.22.12.
Type ".help" for more information.
> var a = "안녕";
undefined
> var b = "반가워";
undefined
> console.log ( a + " 000. " + b );
안녕 000. 반가워
undefined
> .exit

C:\Users\linda>
```

› 에서 javascript 코드 입력

➤ 간단한 코드 테스트 용도

Node.js 특징

Node.js 특징

1. 자바스크립트 언어 사용
2. Single Thread
3. Non-blocking I/O
4. 비동기적 Event-Driven

특징 2) Single Thread

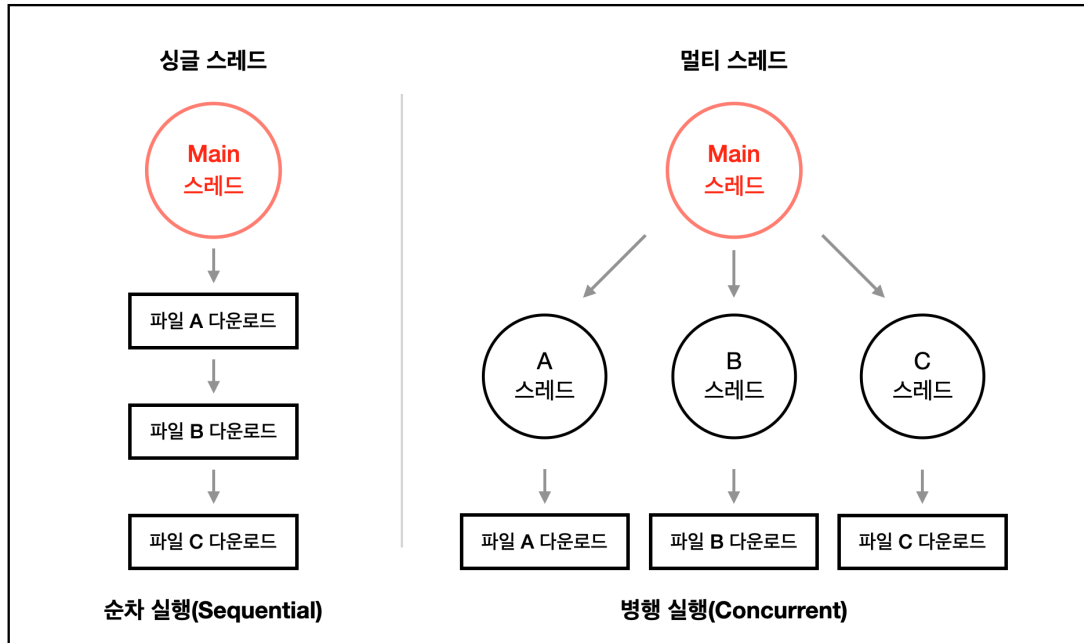
프로세스

- 실행 중인 프로그램
- 운영체제에서 할당하는 작업의 단위

Thread(스레드)

- 프로세스 내에서 실행되는 흐름의 단위
- 하나의 프로세스에는 n 개의 스레드가 존재하며 동시에 작동할 수 있다(병렬처리)

특징 2) Single Thread



Node.js는 사용자가 직접 제어할 수 있는 스레드는 하나이다.

- 싱글 스레드라 주어진 일을 하나밖에 처리 못한다.
- Non-blocking I/O 기능으로 일부 코드는 백그라운드(다른 프로세스) 에서 실행 가능
- **에러를 처리하지 못하는 경우 멈춘다.**
- 프로그래밍 난이도가 쉽고, cpu, 메모리 자원을 적게 사용한다.

특징 2) Single Thread

~~싱글 스레드?
멀티 스레드 프로세스?
CPU?~~

에러를 처리하지 못하면 프로그램이 아예 중단됨



예외처리의 중요성 ↑

특징 3) Non-blocking I/O

- 블로킹(Blocking)

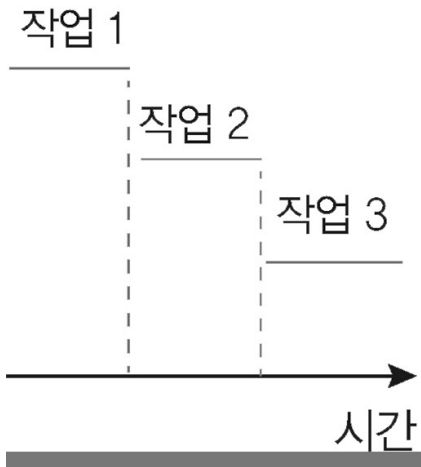
- 요청 후 대기: 요청한 작업이 완료될 때까지 기다려야 하며 다른 작업을 할 수 없다.
- 작업 중단: 하나의 작업이 끝날 때까지 모든 다른 작업이 중단된다.
- 예시: 파일을 읽는 동안 그 작업이 끝날 때까지 기다림.

- 논블로킹(Non-blocking)

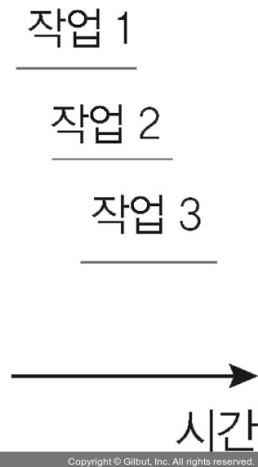
- 요청 후 진행: 요청한 작업이 완료되지 않아도 바로 다음 작업을 계속할 수 있다.
- 작업 병행: 여러 작업을 동시에 처리할 수 있다.
- 예시: 파일을 읽는 동안 다른 작업을 계속할 수 있다. 작업이 끝나면 결과를 처리함

특징 3) Non-Blocking I/O

블로킹



논블로킹



• 블로킹 (Blocking)

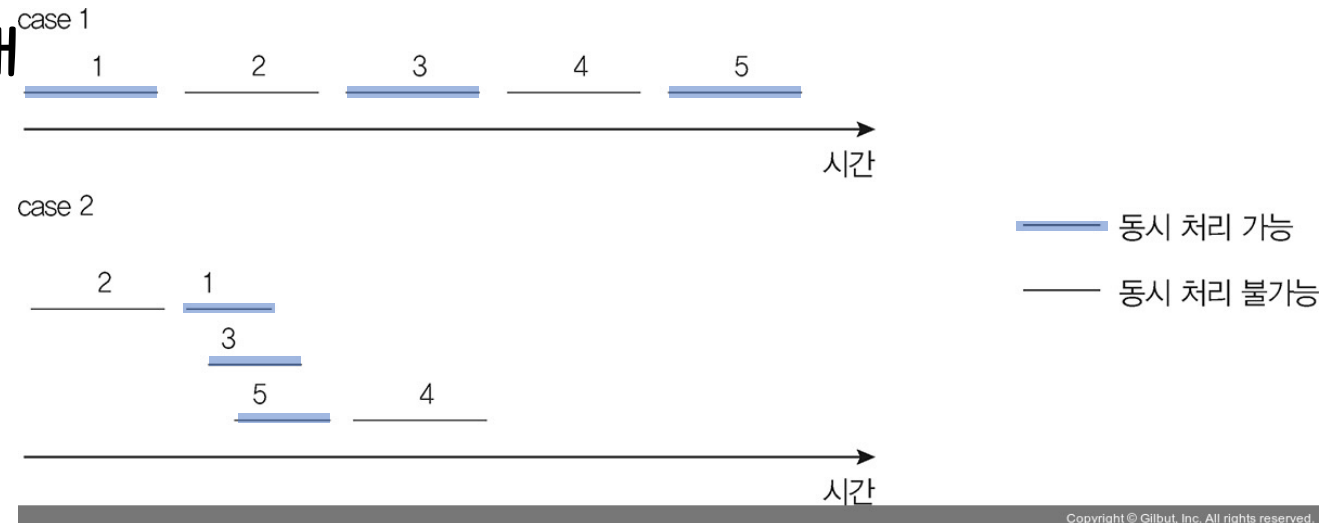
- 해당 작업이 끝나야만 다음 작업을 수행

• 논 블로킹 (Non-Blocking)

- 작업이 완료될 때까지 대기하지 않고 다음 작업 수행. 즉, 빨리 완료된 순서로 처리
- 블로킹 방식보다 같은 작업을 더 짧은 시간에 처리 가능

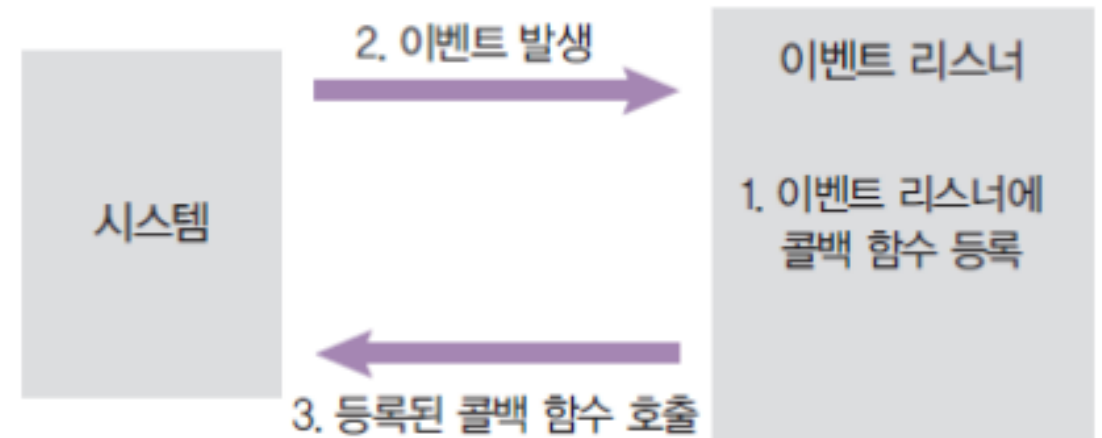
특징 3) Non-Blocking I/O

- I/O
 - 입출력(input/output) 작업
 - ex. 파일 시스템 접근 (읽기, 쓰기, 만들기 등), 네트워크 요청 등
- Node는 I/O 작업을 할 때 논블로킹 방식으로 처리
 - 동시에 처리될 수 있는 작업을 최대
 - 시간적 이득을 획득



특징 4) Event-Driven

- Event-Driven : 이벤트가 발생할 때 미리 지정해둔 작업을 수행
- Ex) 클릭, 네트워크 요청, 타이머 등
- 이벤트 리스너 (Event Listener)
 - 이벤트 등록 함수
- 콜백 함수 (Callback Function)
 - 이벤트가 발생했을 때 실행되는 함수

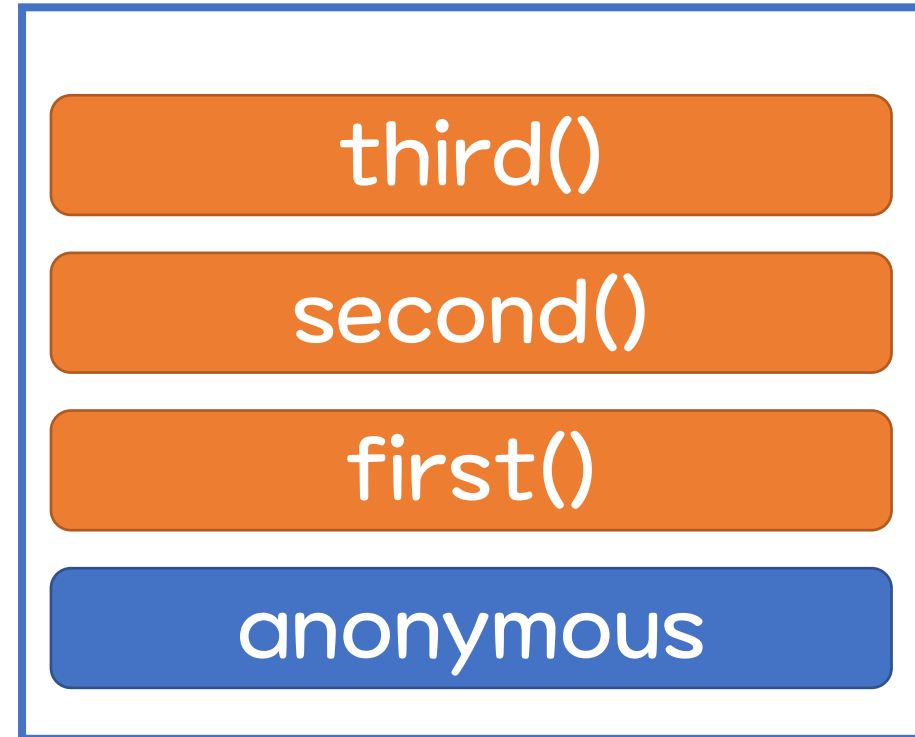


Call Stack

```
1 function first() {
2     second();
3     console.log( "first" );
4 }
5 function second() {
6     third();
7     console.log( "second" );
8 }
9 function third() {
10    console.log( "thrid" );
11 }
12 first();
13
14
```

문제 출력 디버그 콘솔 터미널 GITLENS

thrid
second
first



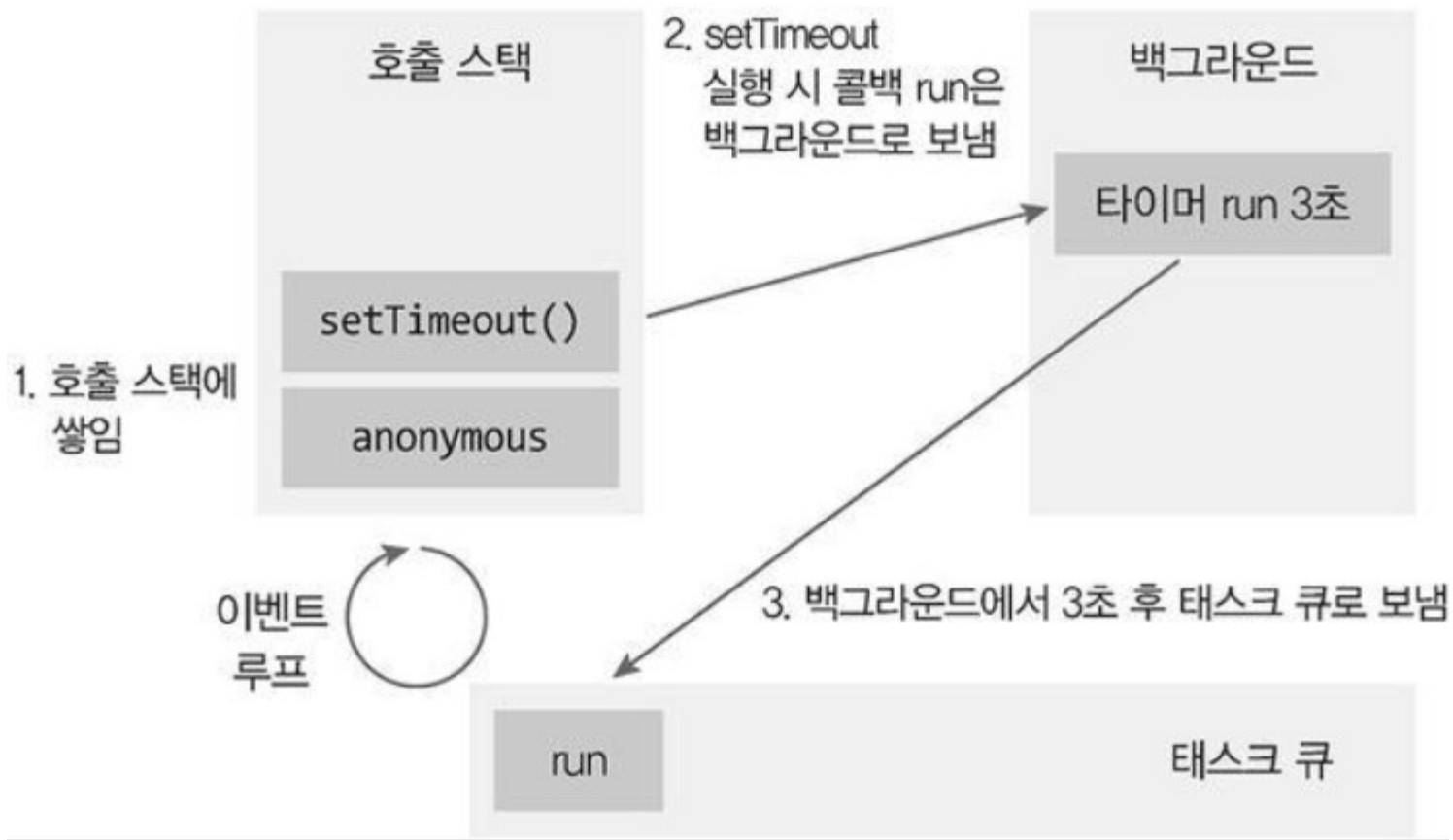
Call Stack (호출 스택)
LIFO 방식

Event Loop

```
function run() {  
  console.log("3초 뒤 실행");  
}
```

```
console.log("시작");  
setTimeout(run, 3000);  
console.log("끝");
```

```
/*  
시작  
끝  
3초 후 실행  
*/
```



Event Loop

4. 호출 스택 실행이
끝나 비워지면

호출 스택

백그라운드

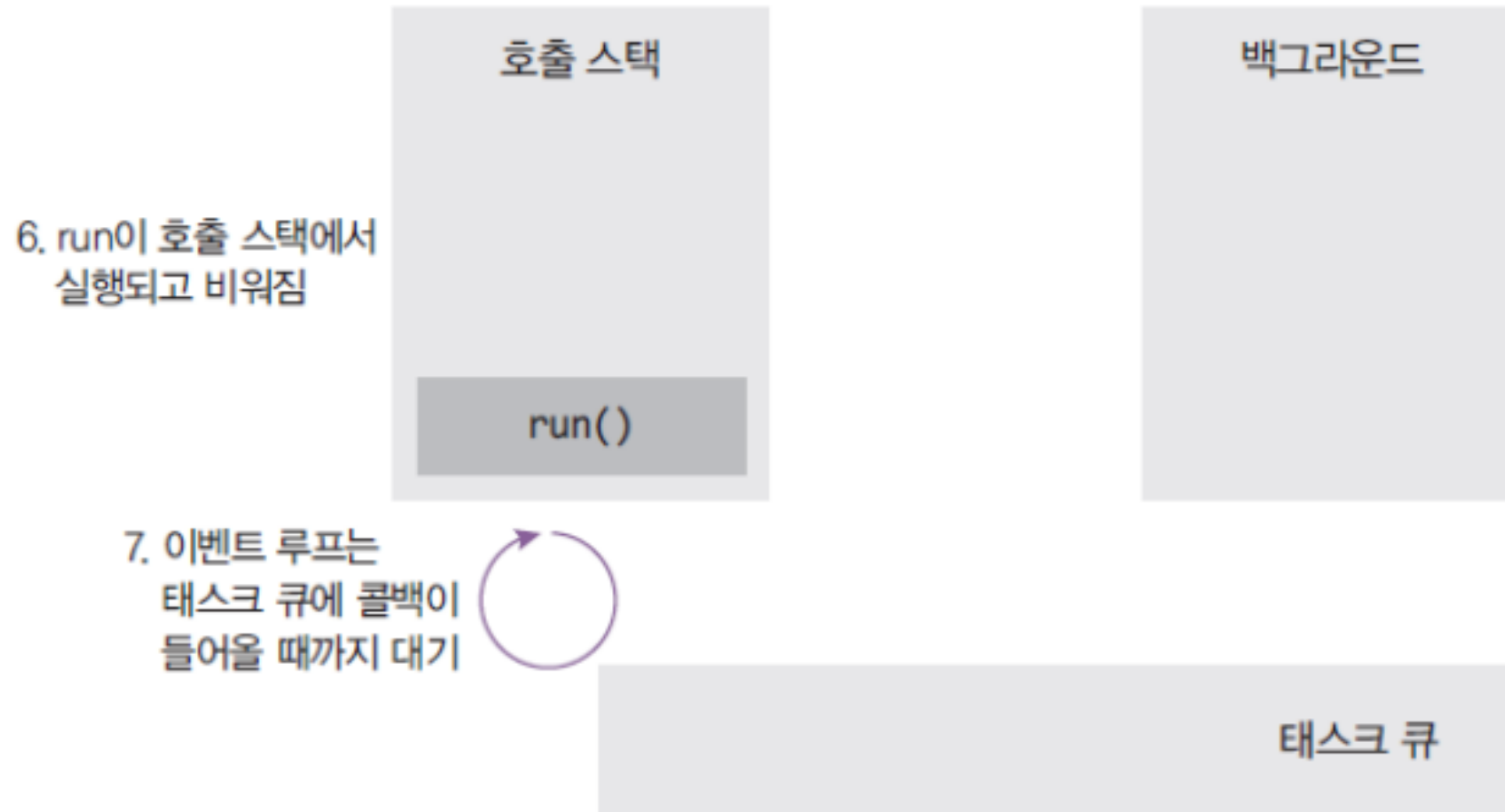
5. 이벤트 루프가
태스크 큐의 콜백을
호출 스택으로 올림



run

태스크 큐

Event Loop



Node.js 의 역할

- 간단한 로직
- 대량의 클라이언트가 접속하는 서비스 (입출력이 많은 서비스)
- 빠른 개발 요구
- 빠른 응답시간 요구
- 비동기 방식에 어울리는 서비스 (스트리밍 서비스, 채팅 서비스 등)

NPM

npm

- Node Package Manager (<https://www.npmjs.com/>)
- 노드 패키지를 관리해주는 툴

- Npm에 업로드 된 노드 모듈
- 패키지들 간 의존 관계가 존재



npm 사용하기

```
npm init
```

- 프로젝트를 시작할 때 사용하는 명령어
- **package.json**에 기록될 내용을 문답식으로 입력한다.

```
npm init --yes
```

- **package.json**이 생성될 때 **기본 값으로** 생성된다.

```
npm install 패키지 이름
```

- 프로젝트에서 사용할 **패키지를 설치**하는 명령어
- 설치된 패키지의 이름과 정보는 **package.json**의 **dependencies**에 입력된다.

package.json

- 패키지들이 서로 의존되어 있어, 문제가 발생할 수 있는데 이를 관리하기 위해 필요한 것
- 프로젝트에 대한 정보와 사용 중인 패키지 이름 및 버전 정보가 담겨 있는 파일

```
{  
  "name": "220721",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  ▶ 디버그  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC"  
}
```

package.json

“name” : 패키지 이름

“version” : 패키지의 버전

“main” : 자바스크립트 실행 파일 진입점 (문답식에서의 entry point)

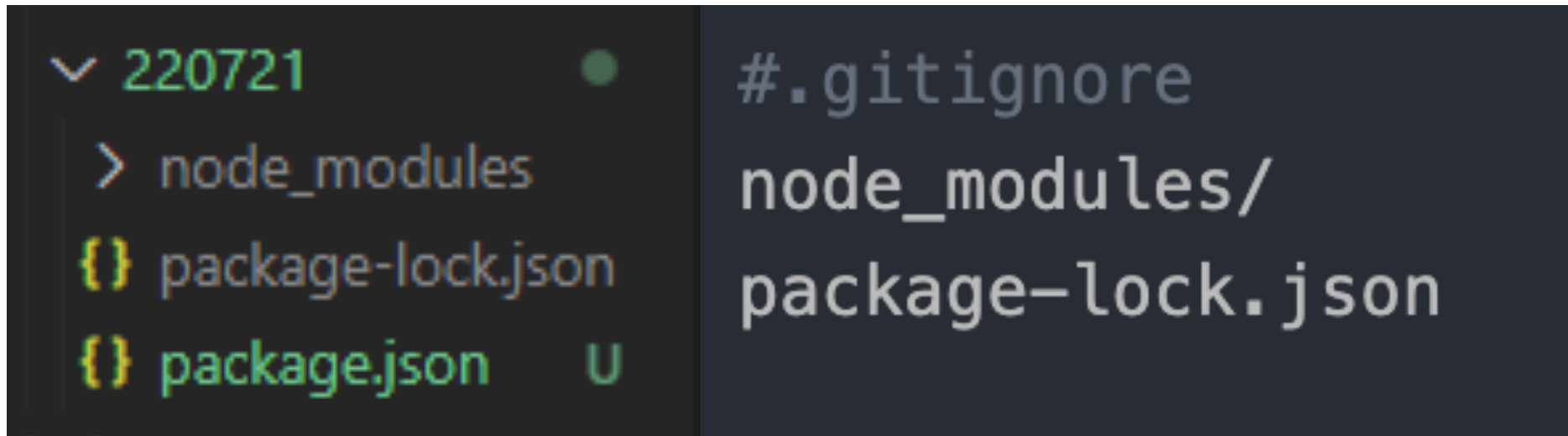
“description” : 패키지에 대한 설명

“**scripts**” : npm run 을 이용해 정해놓는 스크립트 명령어

“license” : 해당 패키지의 라이선스

.gitignore

- node_modules 폴더는 깃 저장소에 업로드 되게 하지 말자!



The image shows a code editor with a dark theme. On the left, a file explorer shows a folder named '220721' which contains a subfolder 'node_modules' and two files: 'package-lock.json' and 'package.json'. On the right, the content of the '.gitignore' file is displayed, showing the following text:

```
#.gitignore
node_modules/
package-lock.json
```

.gitignore

.gitignore?

- Git 버전 관리에서 **제외할 파일 목록을 지정**하는 파일
- Git 관리에서 특정 파일을 제외하기 위해서는 git에 올리기 전에 .gitignore에 파일 목록을 미리 추가해야 한다.

.gitignore

***.txt** → 확장자가 txt로 끝나는 파일 모두 무시

!test.txt → test.txt는 무시되지 않음.

test/ → test 폴더 내부의 모든 파일을 무시 (b.exe와 a.exe 모두 무시)

/test → (현재 폴더) 내에 존재하는 폴더 내부의 모든 파일 무시 (b.exe무시)

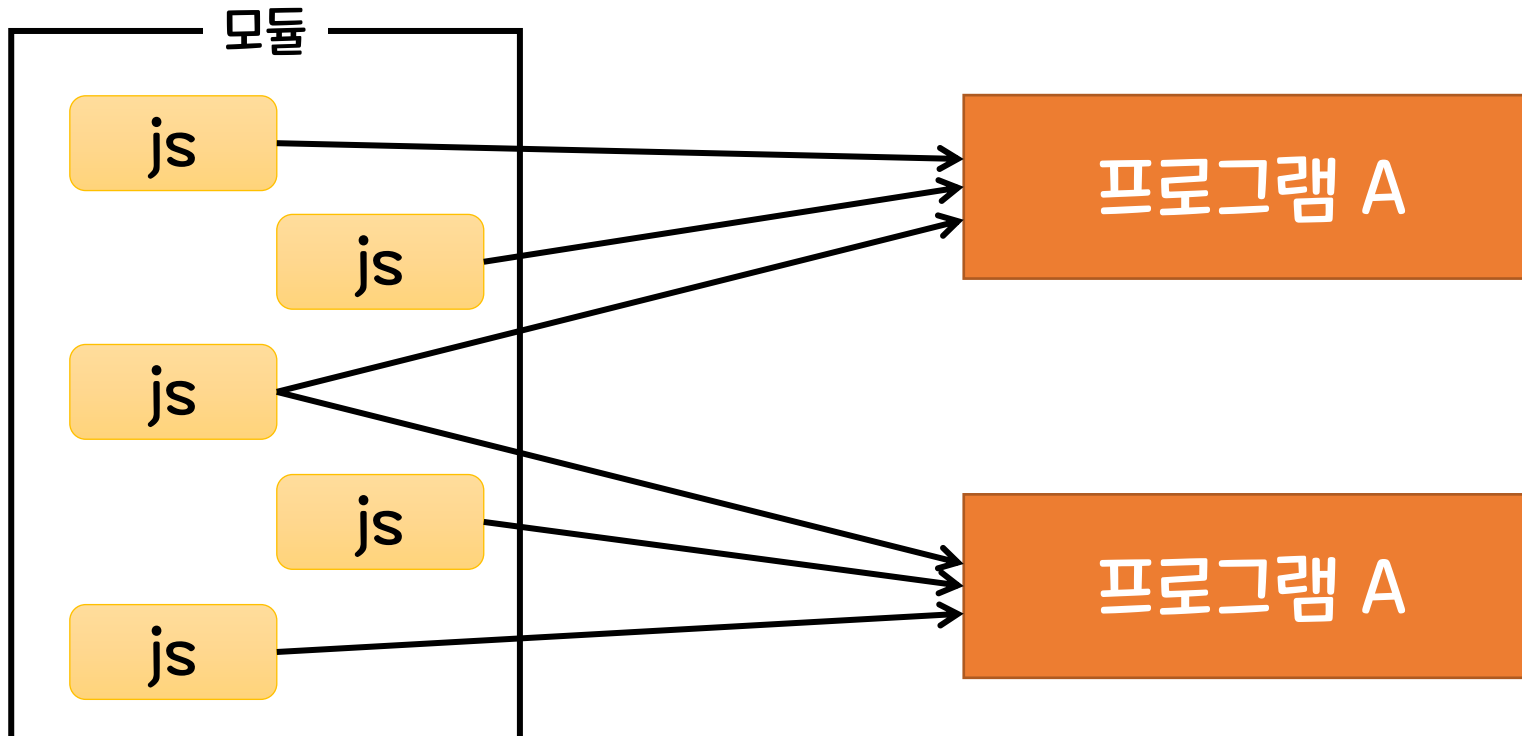
```
(base) [07:25 PM] cwjcsk:~/99_test/99_tmp$ tree -a
.
├── .gitignore
├── test
│   └── b.exe
└── tmp
    └── test
        └── a.exe

3 directories, 3 files
```

모듈 (Module)

모듈이란?

- 특정한 기능을 하는 함수나 변수들의 집합
- 재사용 가능한 코드 조각



모듈의 장점

- 코드 추상화
 - 복잡한 시스템이나 객체를 단순화하여 핵심적인 부분에 집중하는 프로그래밍 원칙
- 코드 캡슐화
 - 코드의 무분별한 변경을 막기 위해 모듈 내부에 코드를 숨길 수 있다.
- 코드 재사용
- 의존성 관리



모듈 만들기(1)

- 하나의 모듈 파일에 하나 만들기
- 하나의 모듈을 `module.exports` 로 내보내기 하여 다른 파일에서 사용하여야 함

`module.exports` = 내보내려는 항목

```
//module1.js
const a = 10;
const b = 20;

//하나의 모듈 파일에 하나의 모듈 만들기
function connect() {
    //~~~ 로직
    return a + b;
}

module.exports = connect;
```

모듈 사용하기(1)

- 하나의 모듈 가져와서 사용하기

```
//index.js
const connect = require("./module1.js");
const result = connect();
console.log(result); // 결과 30
```

모듈 만들기(2)

- 하나의 모듈 파일에 여러 개 만들기

```
//module2.js
//방법1
function add(x, y) {
    return x + y;
}
function subtract(x, y) {
    return x - y;
}
function multiply(x, y) {
    return x * y;
}
module.exports = {
    add,
    subtract,
    multiply,
};
```

```
//방법2
module.exports.add = function(x, y) {
    return x + y;
};
module.exports.subtract = function(x, y) {
    return x - y;
};
module.exports.multiply = function(x, y) {
    return x * y;
};
//방법2 생략버전
exports.add = function(x, y) {
    return x + y;
};
exports.subtract = function(x, y) {
    return x - y;
};
exports.multiply = function(x, y) {
    return x * y;
};
```

모듈 사용하기(2)

- 하나의 모듈 파일에 있는 여러 개 모듈 가져와서 사용하기

```
//index.js
const calculator = require('./module2.js');
console.log(calculator.add(1, 3));
console.log(calculator.subtract(6, 4));
console.log(calculator.multiply(5, 7));

//객체 구조분해 이용
const { add, subtract, multiply } = require('./module2.js');
console.log(add(2, 3));
console.log(subtract(5, 3));
console.log(multiply(4, 5));
```


모듈 불러오기 주의사항

- const { } 로 가져올 때는 구조분해해 가져오기에 이름이 동일해야 한다.

```
const { a, b } = require("../var.js");  
const returnString = require("../func.js");
```

- 하나만 내보낸 모듈은 다른 이름이어도 불러올 수 있다.

ES2015 모듈

자바스크립트 자체 모듈 시스템 문법

-> package.json 에 “type”: “module” 을 추가해 사용

```
"main": "index.js",  
"type": "module",  
  Debug  
"scripts": {
```

ES2015 모듈

export : 모듈 내보내기

```
const add = (x, y) => x + y;
const subtract = (x, y) => x - y;
const multiply = (x, y) => x * y;
module.exports = {
  add,
  subtract,
  multiply,
};
```



```
const add = (x, y) => x + y;
const subtract = (x, y) => x - y;
const multiply = (x, y) => x * y;
export { add, subtract, multiply };
```

```
2
3  ✓ function connect() {
4    |   return a + b;
5    | }
6
7  module.exports = connect;
```

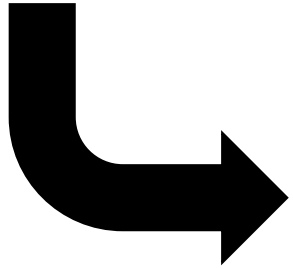


```
✓ function connect() {
  |   return a + b;
  | }
  export default connect;
```

ES2015 모듈

import ~ from ~ : 모듈 가져오기

```
const { a, b } = require("./var.js");  
const returnString = require("./func.js");
```



```
import { a, b } from './var.js';  
import returnString from './func.js';
```

내장 모듈

fs(File System)

- 파일 시스템 작업을 위한 모듈로, 파일 읽기, 쓰기, 삭제, 디렉토리 생성 등을 할 수 있다.

```
const fs = require("fs");

// 파일 쓰기
fs.writeFile("file.txt", "Hello, World!", (err) => {
  if (err) throw err;
  console.log("저장완료");
});

// 파일 읽기
fs.readFile("file.txt", "utf8", (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

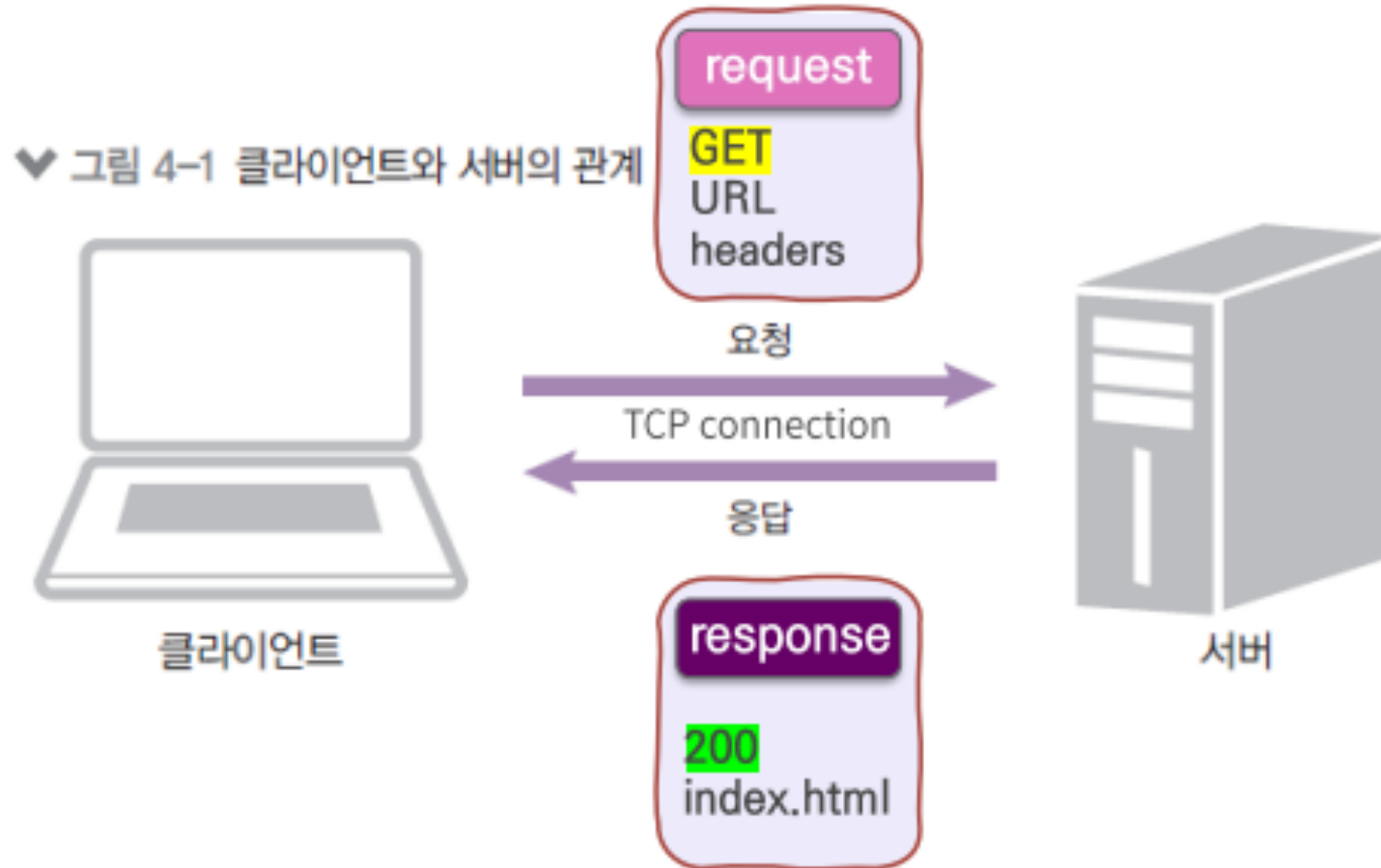
- 파일 및 디렉토리 경로 작업을 보다 쉽게 할 수 있도록 도와줌
- 운영 체제 간 경로 표기법 차이를 자동으로 처리
- 주요 메서드
 - join(경로들) : 여러 개의 경로 조각을 결합하여 하나의 경로로 생성
 - resolve(경로들) : 여러 개의 경로 조각을 결합하여 절대 경로 생성(없으면 현재 경로)
 - basename(경로, 확장자[선택]) : 주어진 경로의 마지막 부분(파일 이름)을 반환
 - extname(경로): 주어진 경로의 확장자를 반환
 - dirname(경로): 주어진 경로의 디렉토리 부분을 반환

- URL 문자열을 처리하고 구문 분석하기 위한 모듈(URL은 전역객체)

```
const myURL = new URL("http://localhost.com:8000/path/name?query=string#hash");
//속성
console.log(myURL.href); //http://localhost.com:8000/path/name?query=string#hash
console.log(myURL.hostname); // localhost.com
console.log(myURL.pathname); // /path/name
console.log(myURL.search); // ?query=string
console.log(myURL.origin); // http://localhost.com:8000
console.log(myURL.protocol); // http:
console.log(myURL.host); // localhost.com:8000
console.log(myURL.port); // 8000
console.log(myURL.hash); // #hash
console.log(myURL.searchParams.get("query")); // string
myURL.searchParams.append("newParam", "value");
console.log(myURL.search); // ?query=string&newParam=value
//매서드
console.log(myURL.toString()); // URL 객체를 문자열로 반환
const params = new URLSearchParams("?query=string&newParam=value");
console.log(params.get("query")); // string
params.set("query", "newString");
console.log(params.toString()); // query=newString&newParam=value
params.delete("newParam");
console.log(params.toString()); // query=newString
```


서버 만들기

http 통신



- Nodejs 를 통해 서버를 구축하는 방법은 http 와 express 두 개
- http 모듈
 - 웹 서버를 구동하기 위한 node.js 내장 웹 모듈
 - server 객체, request 객체, response 객체를 사용한다.
 - server 객체 : 웹 서버를 생성할 때 사용하는 객체
 - request 객체 : 요청 메시지를 작성할 때 첫 번째 매개변수로 전달되는 객체
 - response 객체 : 응답 메시지를 작성할 때 두 번째 매개변수로 전달되는 객체

http 모듈 서버 만들기

```
const http = require('http');  
  
const server = http.createServer();  
  
server.listen(8080, function(){  
  console.log( '8080번 포트로 서버 실행' );  
});
```

listen(port, callback)

서버를 첫번째 매개변수의 포트로 실행한다.

http 모듈 서버 만들기

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

const server = http.createServer( function(req, res){
  res.writeHead( 200 );
  res.write( "<h1>Hello!</h1>" );
  res.end("<p>End</p>");
});

server.listen(8080, function(){
  console.log( '8080번 포트로 서버 실행' );
});
```

Response 객체

writeHead : 응답 헤더 작성

write : 응답 본문 작성(여러번 작성가능)

end : 응답 본문 작성 후 응답 종료

localhost 와 port

- localhost
 - localhost는 컴퓨터 내부 주소 (127.0.0.1)
 - 자신의 컴퓨터를 가리키는 호스트이름(hostname)
- Port
 - 서버 내에서 데이터를 주고받는 프로세스를 구분하기 위한 번호
 - 기본적으로 http 서버는 80번 포트 사용 (생략 가능, https는 443)

server 객체

* 메서드

| | |
|----------|-----------------------|
| listen() | 서버를 실행하고 클라이언트를 기다린다. |
| close() | 서버를 종료한다. |
| on() | server 객체에 이벤트를 등록한다. |

* 이벤트

| | |
|---------------|-----------------------------------|
| request | 클라이언트가 요청할 때 발생하는 이벤트 |
| connection | 클라이언트가 접속할 때 발생하는 이벤트 |
| close | 서버가 종료될 때 발생하는 이벤트 |
| checkContinue | 클라이언트가 지속적인 연결을 하고 있을 때 발생하는 이벤트 |
| upgrade | 클라이언트가 http 업그레이드를 요청할 때 발생하는 이벤트 |
| clientError | 클라이언트에서 오류가 발생할 때 발생하는 이벤트 |

server 객체 - 이벤트

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

const server = http.createServer( function(req, res){
  res.writeHead( 200 );
  res.write( "<h1>Hello!</h1>" );
  res.end("<p>End</p>");
});

server.on('request', function(code){
  console.log( "request 이벤트" );
});

server.on('connection', function(code){
  console.log( "connection 이벤트" );
});

server.listen(8080, function(){
  console.log( '8080번 포트로 서버 실행' );
});
```


html 파일 전송

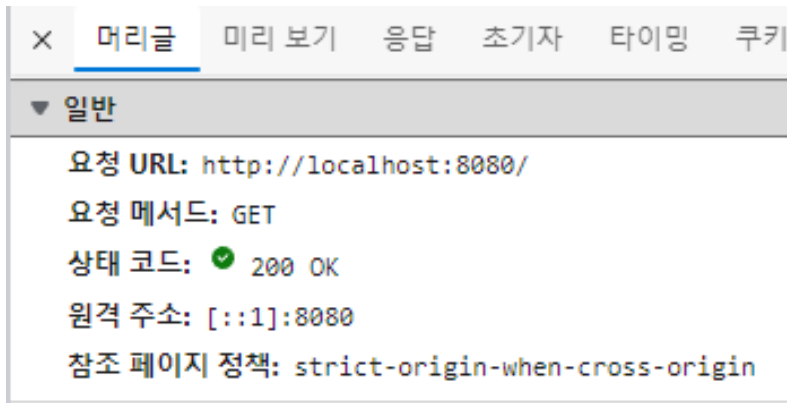
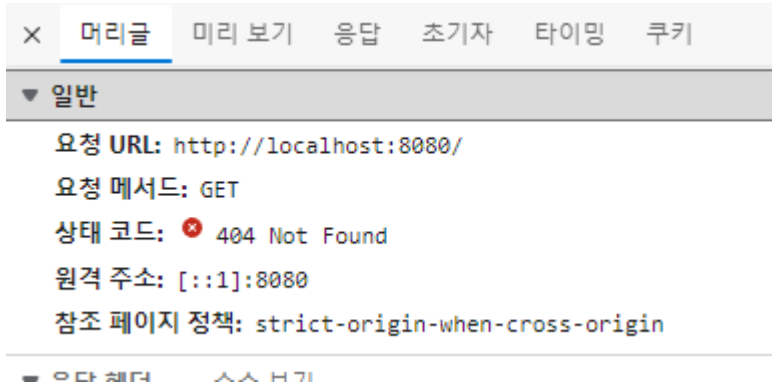
```
<html>
  <head>
    <title>http모듈</title>
  </head>
  <body>
    <h1>Hello http</h1>
    <p>p태그</p>
  </body>
</html>
```

```
const http = require("http");
const fs = require("fs");

const server = http.createServer((req, res) => {
  //파일전송
  fs.readFile("./index.html", (err, data) => {
    if (err) {
      console.error(err);
      res.writeHead(404, { "Content-Type": "text/plain" });
      res.write(err.message);
      res.end();
      return;
    }
    res.writeHead(200, "OK", { "Content-Type": "text/html" });
    res.write(data);
    res.end("<div>END</div>");
  });
});

//listen: 서버를 실행
server.listen(8000, function () {
  console.log("8000번 포트 실행");
});
```

http 응답



- 1XX : 처리중
 - 100: Continue, 102: Processing
- 2XX : 성공
 - 200: OK, 201: Created, 202: Accepted
- 3XX : 리다이렉트(다른 페이지로 이동)
- 4XX : 요청 오류
 - 400: 잘못된 요청, 401: 권한 없음, 403: 금지됨
 - 404: 찾을 수 없음(Page not found)
- 5XX : 서버 오류

<https://developer.mozilla.org/ko/docs/Web/HTTP/Status>