Node.js

목차

1. Node.js 란?

Node.js는 크로스플랫폼 오픈소스 자바스크립트 런타임 환경으로 윈도우, 리눅스, macOS 등을 지원한다.
Node.js는 V8 자바스크립트 엔진으로 구동되며, 웹 브라우저 바깥에서 자바스크립트 코드를 실행할 수 있다.
주로 확장성 있는 네트워크 애플리케이션과 서버 사이드 개발에 사용되는 소프트웨어 플랫폼이며, 논블로킹(Non-blocking) I/O와 단일 스레드 이벤트 루프를 통한 높은 처리 성능을 가지고 있다.

2. Node.js vs JavaScript

- NodeJS는 자바스크립트 런타임 환경 (실행환경)입니다.
- 자바스크립트는 웹사이트용 스크립트를 만드는 데 사용되는 컴퓨터 프로그래밍 언어입니다. 브라우저만 자바스크립트를 실행할 수 있습니다
- 브라우저 외부에서 자바스크립트를 실행하려면 node.js 가 필요합니다

3. Node.js 특징

- NodeJs는 Google Chrome의 V8 엔진을 기반으로 구축되어 있으며, 이러한 이유로 실행 시간이 매우 빠르고 매우 빠르게 실행
- Node.js는 다양한 플랫폼(Windows, Linux, Unix, Mac OS X 등)에서 실행
- Node.js는 무료
- Node.js는 서버에서 자바스크립트를 사용합니다

A. 자바스크립트 언어 사용

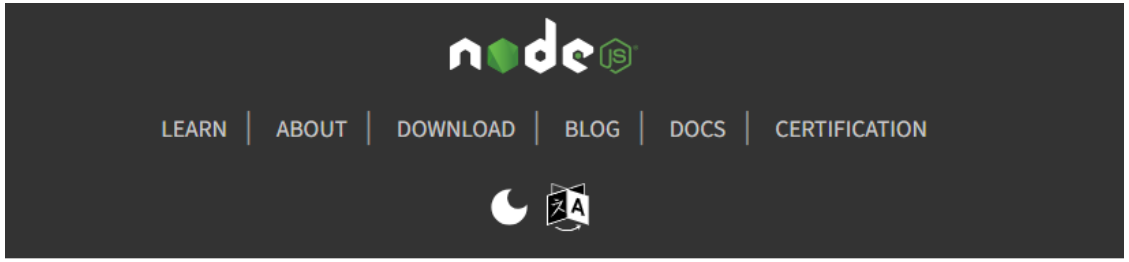
B. 비동기적 Event-Driven

C. Single Thread

D. Non-blocking I/O

4. Node.js 설치

설치 주소: nodejs.org



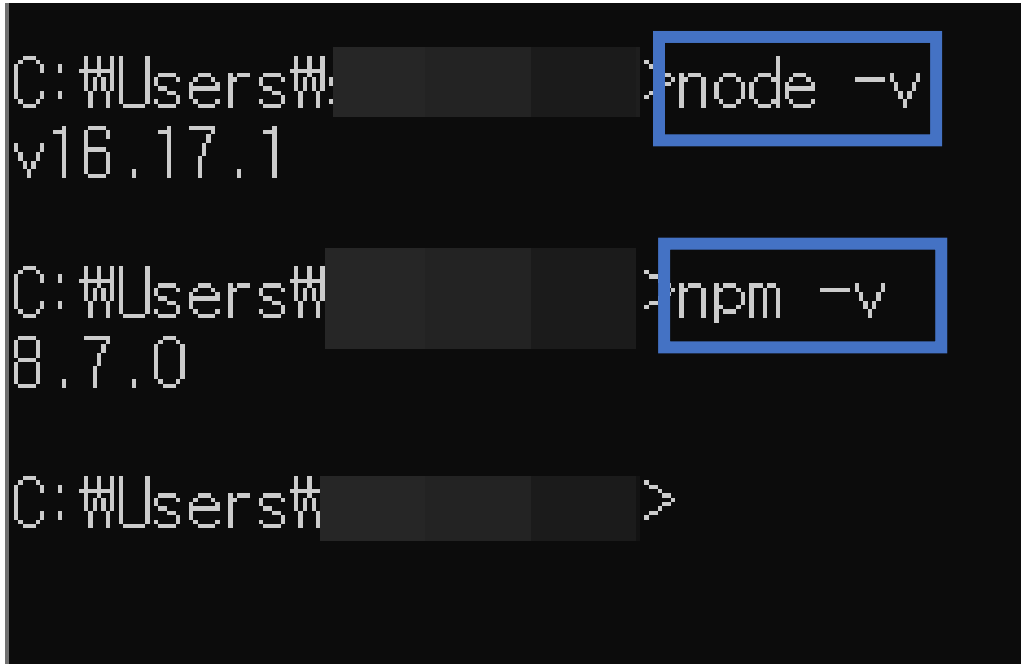
[윈도우]
LTS 버전 다운로드 – 실행파일 클릭

[Mac]

- 1. HomeBrew 설치
  - 이미 설치되어 있다면 생략
  - 터미널에 *brew –v* 명령어 입력 후 버전이 뜬다면 설치된 것
  - https://brew.sh/index_ko

- 2. Node.js 설치
  - *brew install node*

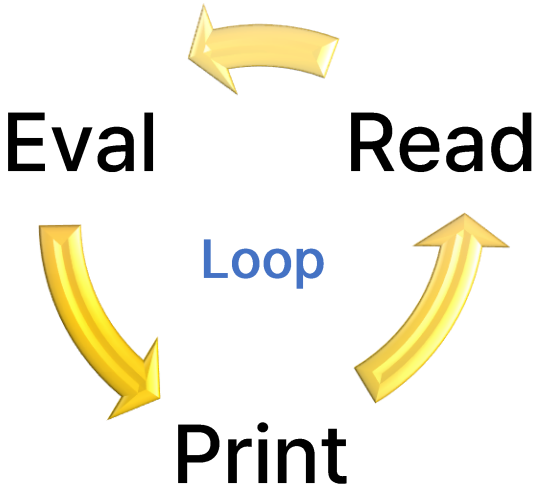
a. Node.js 설치 후 확인



b. Node.js의 대화형 모드, REPL

- Read-Eval-Print-Loop 줄임말로 셸이 동작하는 방식
- 윈도우에서의 cmd, 맥에서의 terminal처럼 노드에는 REPL 콘솔이 있음
- 시작 : node
- 종료 : .exit

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

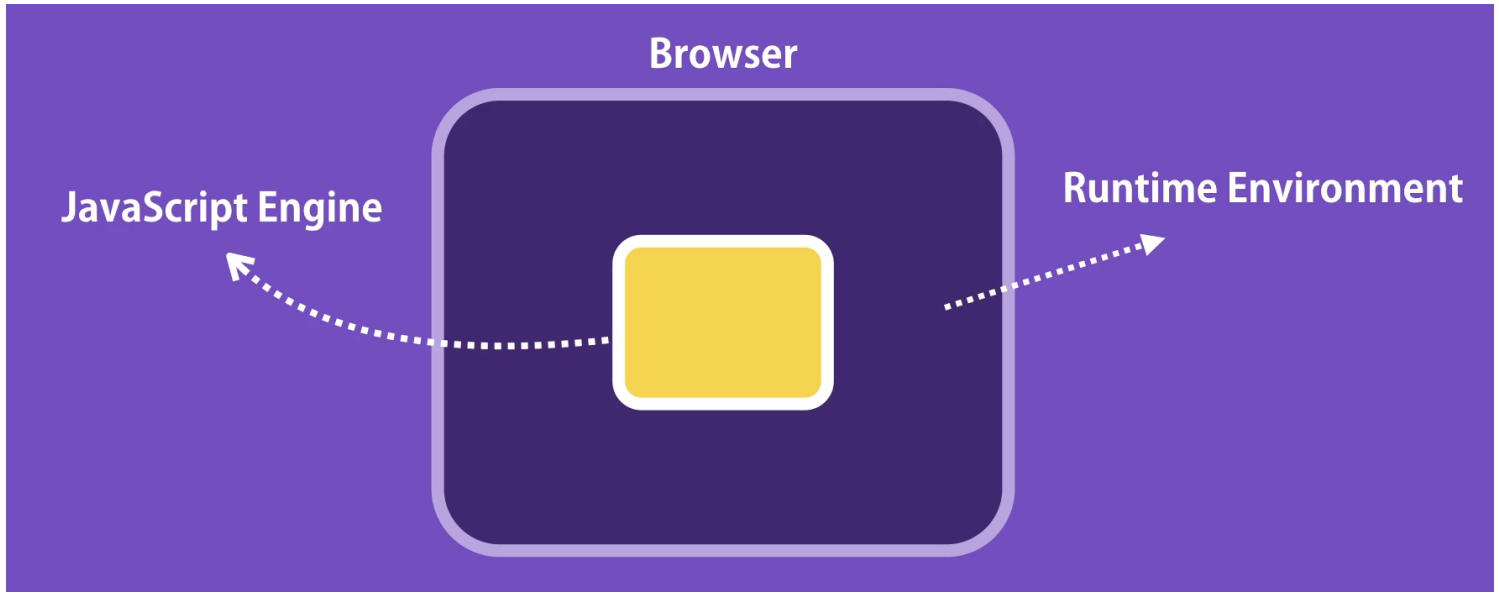


c. REPL 체험해 보기

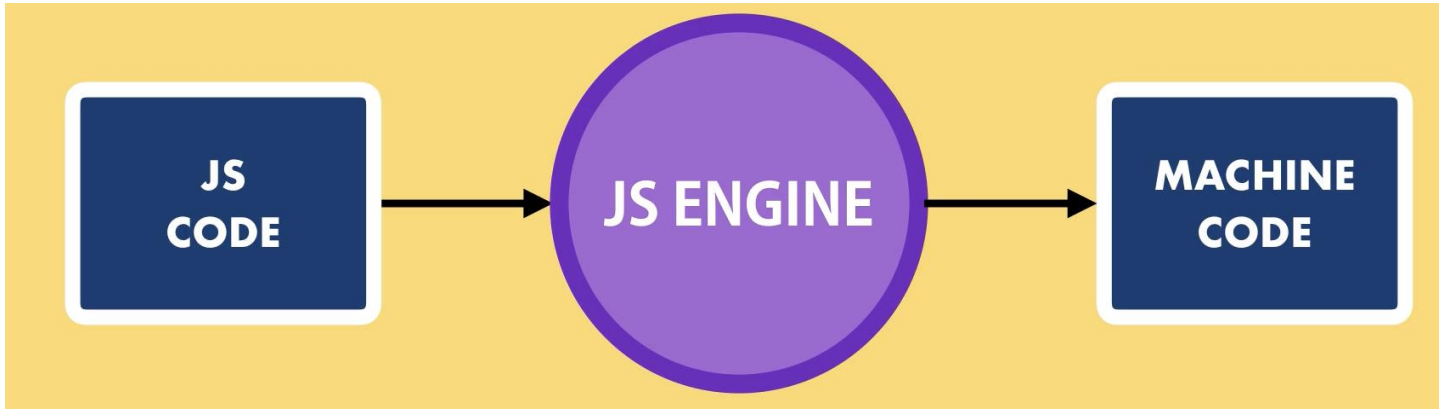
```
C:₩Users₩Linda>node
Welcome to Node.js v12.22.12.
Type ".help" for more information.
> var a = "안녕";
undefined
> var b = "반가워";
undefined
> console.log ( a + " 000. " + b );
안녕 000. 반가워
undefined
> .exit

C:₩Users₩Linda>
```

터미널에서 JS 코드 입력
(간단한 코드 테스트 용도)

5. 자바스크립트의 런타임

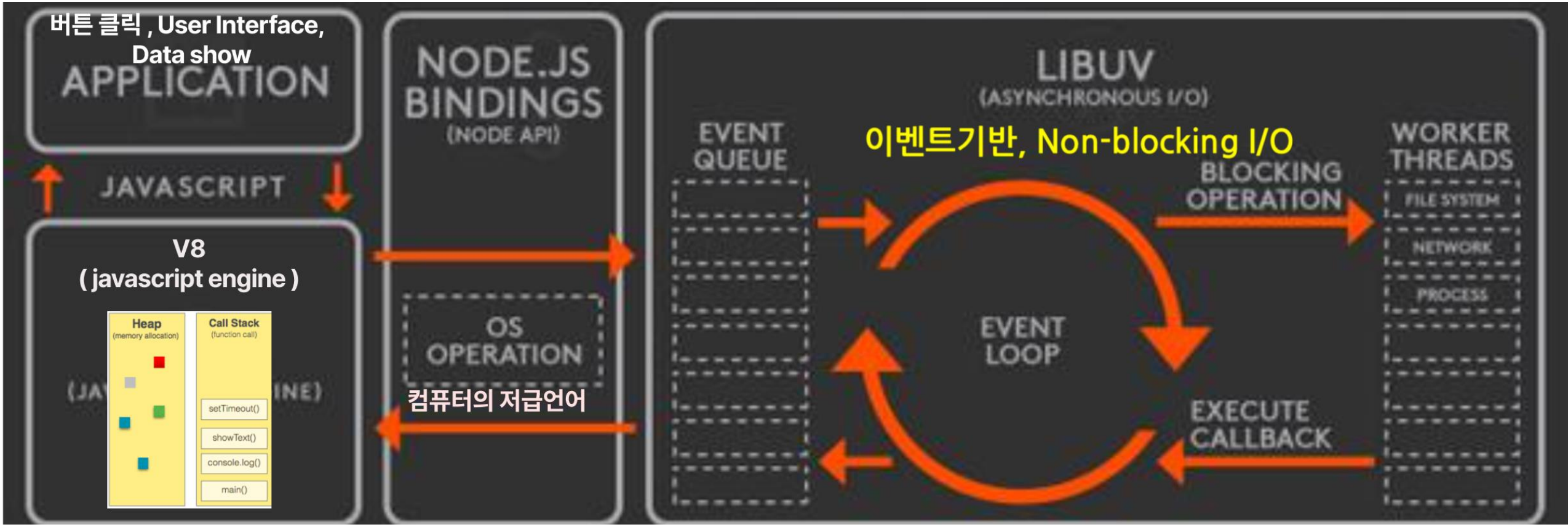


JavaScript 엔진



6. Node.js 아키텍처

THE NODE.JS SYSTEM



7. 비동기적 Event-Driven

이벤트 리스너 , 콜백

- 이벤트 기반 : 이벤트가 발생할 때 미리 저장해둔 작업을 수행하는 방식을 말한다. ex) 클릭이나 네트워크 요청 등이 있다.
- Node.js = V8 + libuv (c , c++로 작성된 이벤트기반, 논블로킹 I/O 모델 구현 라이브러리)
- 이벤트 루프(Event Loop) : 여러 이벤트 동시 발생시 어떤 순서로 콜백 함수를 호출할지 판단하는 역할 담당. 노드가 종료될 때까지 이벤트 처리를 위한 작업을 반복하므로 루프라 부린다.

Promise

- microtasks (Event Queue) : process.nextTick, Promises, queueMicrotask(f), MutationObserver
- macrotasks (Job Queue): I/O, setTimeout, setInterval, requestAnimationF rame , , UI rendering ...

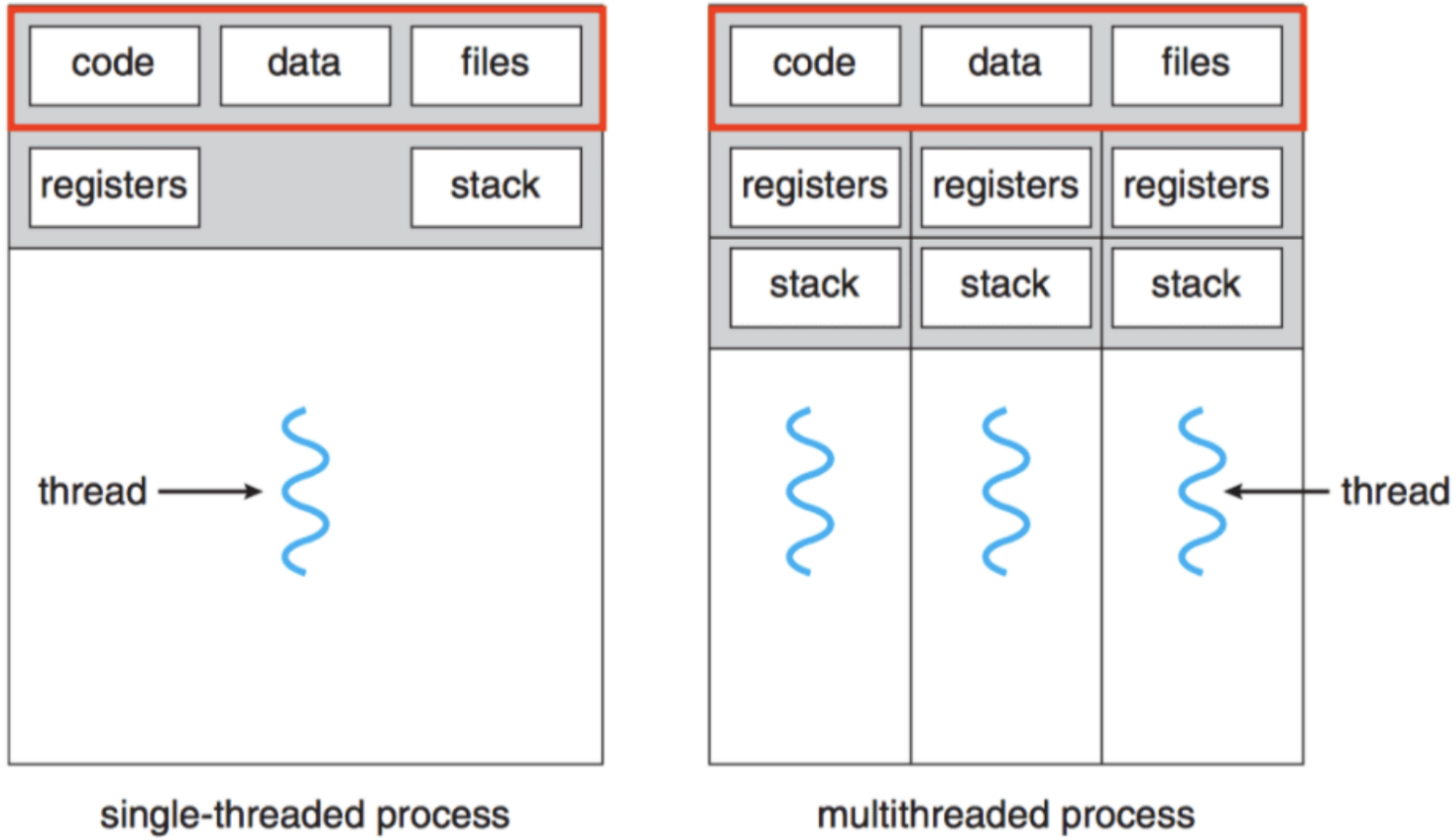
8. 싱글스레드란

자바스크립트는 Single Thread 이다

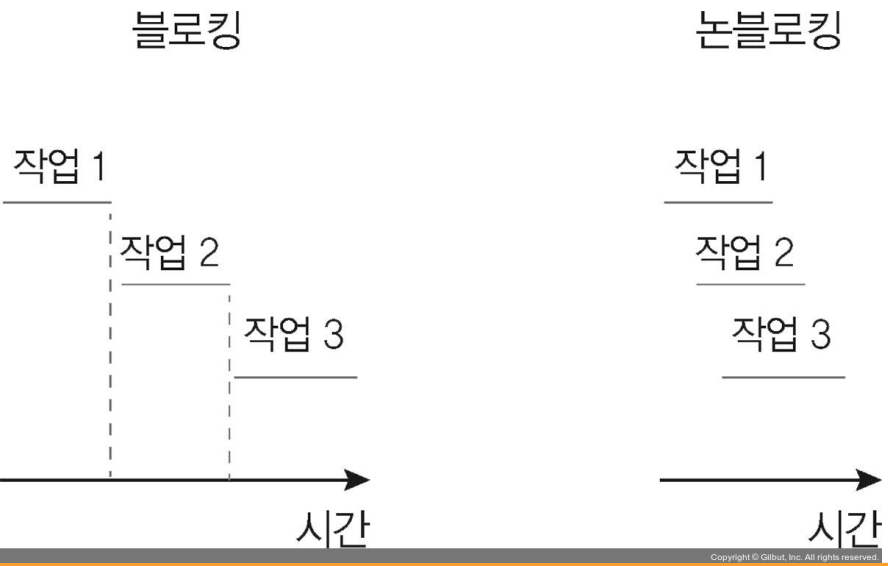
- 프로세스(Process) :

운영체제에서 할당하는 작업의 단위. Node.js나 인터넷 브라우저 같은 프로그램은 개별적인 프로세스

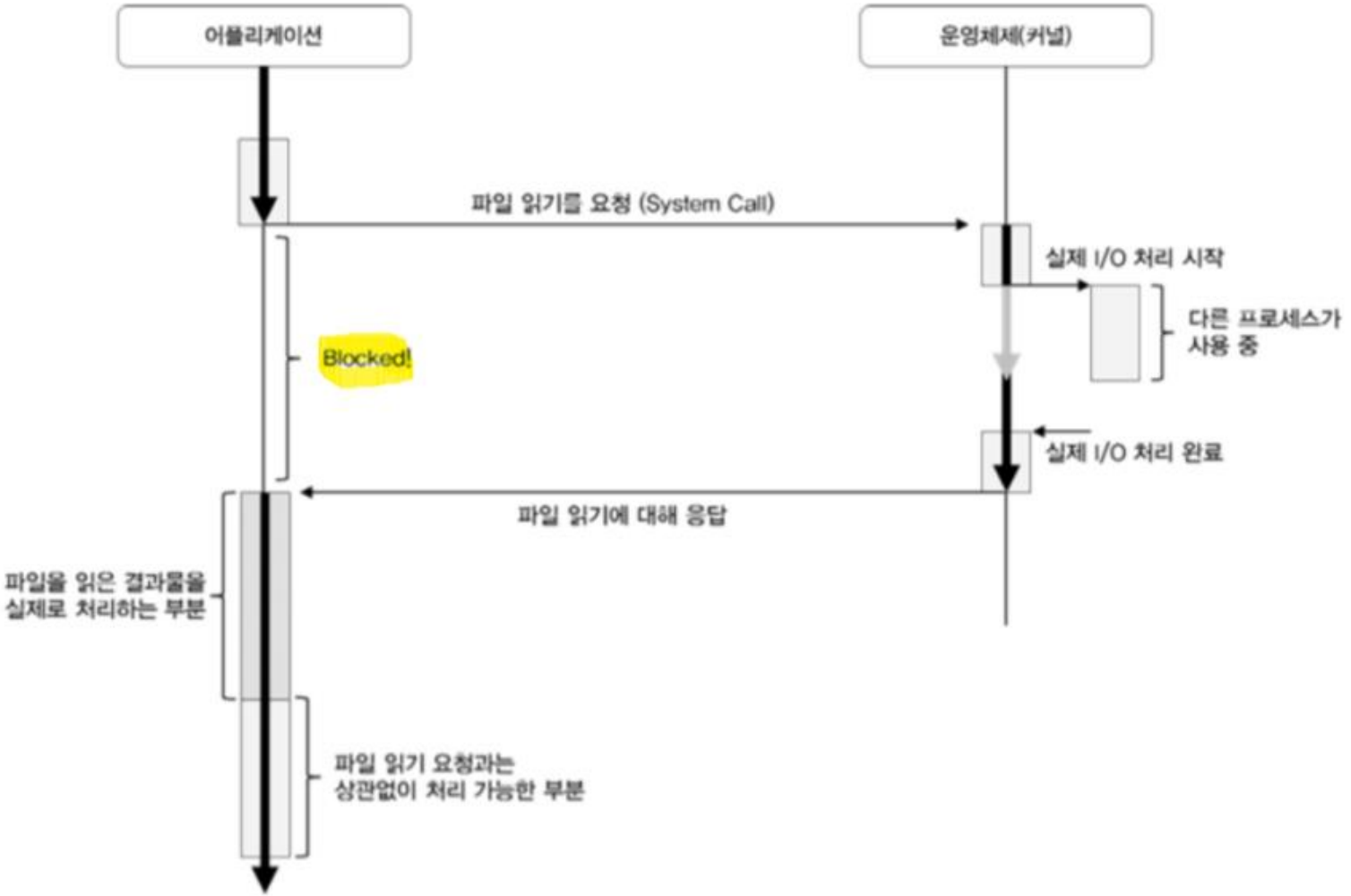
- 스레드(Thread)
  프로세스 내에서 실행되는 흐름의 단위. 하나의 프로세스는 스레드를 여러개 가질 수 있음



9. 블로킹 I/O, Non-blocking I/O

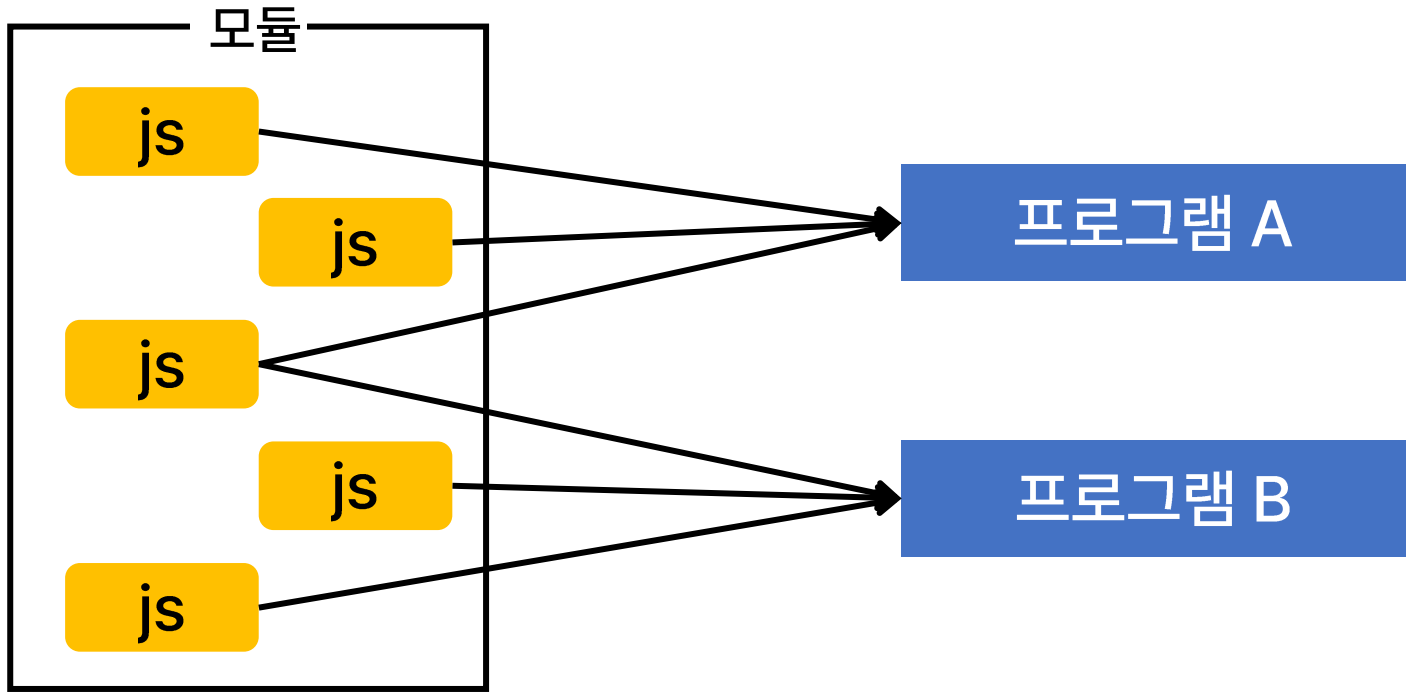


Single Thread 에서 Blocking 방식



10. 모듈이란?

- a few of the modules built into the core of Node such as operating system, file system, events and http 등 그리고 우리가 직접 만드는 모듈도 있다.
- 특정한 기능을 하는 함수나 변수들의 집합
- 재사용 가능한 코드 조각



11. 모듈의 장점

- 코드 추상화
- 코드 캡슐화
- 코드 재사용
- 의존성 관리

11. 구조분해문법 복습

[html 파일로 작성 , 브라우저의 콘솔창에서 확인]

```
let user ={name:'Hong', age:30}
console.log(user.name, user.age);
```

```
let user ={name:'Hong', age:30}
let {name, age} =user;
console.log(name,age)
```

12. 노드 모듈 만들기-1

- 하나의 모듈에 하나 만들기

[math.js]

```
const add = (a, b) => a + b;

module.exports = add;
```

- module.exports 구문으로 내보내야 다른 파일에서 사용 가능

```
module.exports = 내보내려는 항목
```

노드 모듈 불러오기-1

- math 모듈에서 add 함수 불러오기

[app.js]

```
const add = require('./math');

console.log(add);
```

```
>node app.js
```

노드 모듈 만들기-2

- 하나의 모듈 파일에 여러 개 만들기

```
const add = (a, b) => a + b;
const E = 2.718;
const PI = 3.141592;

// case1
module.exports = {
add,
E,
PI
};
```

```
// case2
module.exports.add = add;
module.exports.E = E;
module.exports.PI = PI;

// case2 생략
exports.add = add;
exports.E = E;
exports.PI = PI;
```

math2.js

노드 모듈 불러오고 사용하기-2

- math 모듈 불러오기

  [app2.js]

```
const math = require('./math2');
console.log(math);
```

- math 모듈에서 불러온 함수, 변수 사용하기

  [app2.js]

```
const math = require('./math2');
console.log(math);
console.log(math.add(math.PI, math.E));
```

노드 모듈 불러올 때 주의할 점

- const { } 로 가져올 때는 객체 구조분해해 가져오기에 이름이 동일해야 함

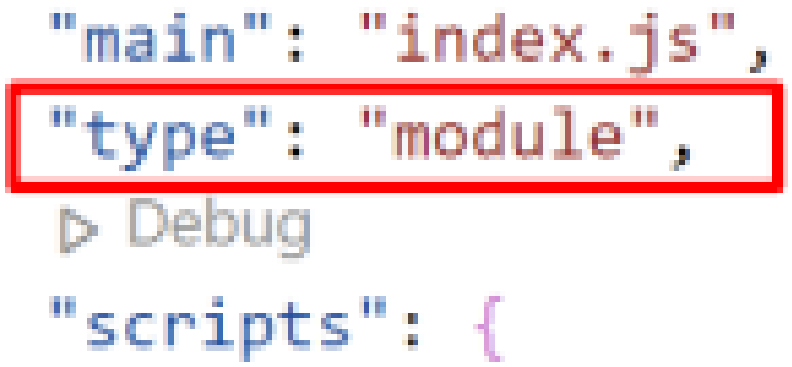
```
const math = require('./math2');
console.log(math);

const { add, E, PI } = require('./math2');
console.log(add(E, PI));
```

- 하나만 내보낸 모듈은 이름이 달라져도 불러올 수 있음

```
const add = require('./math');
console.log(add);

const onlyOne = require('./math');
console.log(onlyOne);
```

13. ES2015 모듈

- 자바스크립트 자체 모듈 시스템 문법
- 노드 모듈 시스템과 방식이 약간 다름
- package.json 에 “type”: “module” 을 추가해 사용



ES2015 모듈

export : 모듈 내보내기

```
const a = "a 변수";
const b = "b 변수";

module.exports = {
    a,
    b
};
```

```
const a = "a 변수";
const b = "b 변수";

export { a, b };
```

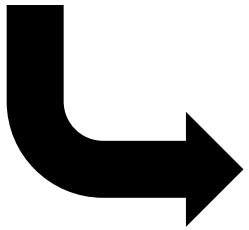
```
function connect() {
    return a + b;
}

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';
```

여기서는 사용방법을 알아보자 실습을 위해서는 package.json 파일이 필요하다

14. NPM

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

Npm에 업로드 된 노드 모듈

패키지들 간 의존 관계가 존재

NPM 사용하기

```
npm init
```

- 프로젝트를 시작할 때 사용하는 명령어
- package.json에 기록될 내용을 문답식으로 입력한다.
- 앞에 있는 간단한 예를 실습해 보자 – 결과는 `[Function: connect]`

```
npm init --yes
```

- package.json이 생성될 때 기본 값으로 생성된다.

```
npm install 패키지 이름
```

- 프로젝트에서 사용할 패키지를 설치하는 명령어
- 설치된 패키지의 이름과 정보는 package.json의 dependencies 에 입력된다.

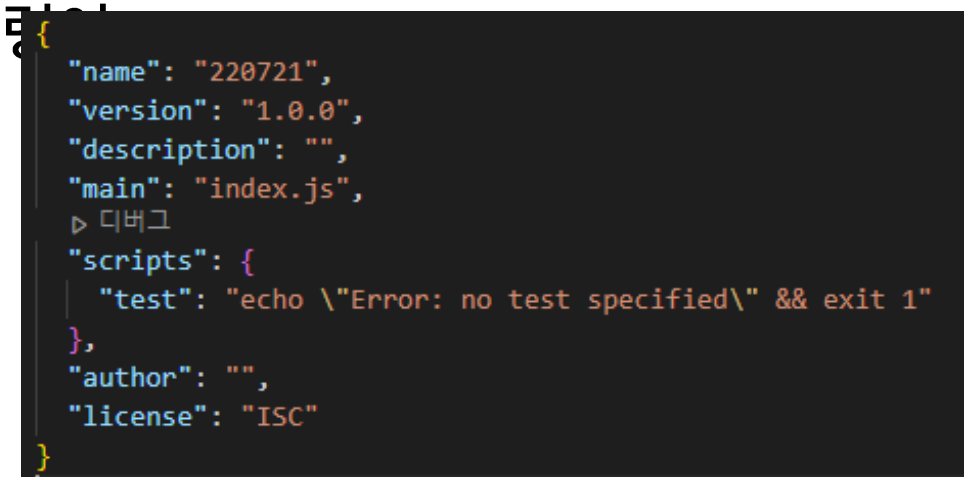
package.json

- 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. 우리가 어플리케이션을 배포할 때, 사용할 라이브러리들이 담겨있는 곳이다
- scripts: npm run 을 이용해 정해놓는 스크립트 명령어
- license: 해당 패키지의 라이센스



15. 서버 만들기

http 통신

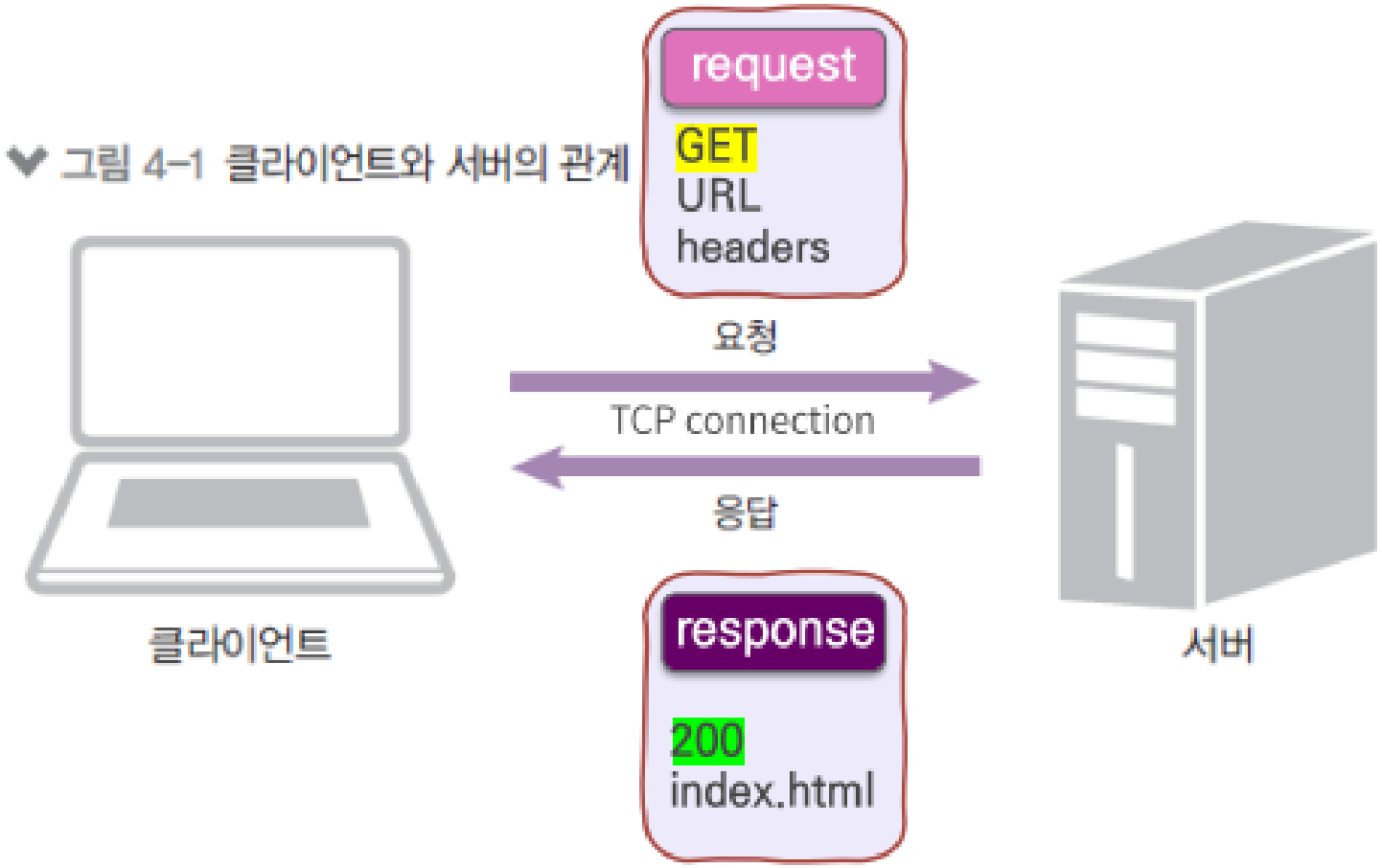


그림 4-1 클라이언트와 서버의 관계
request
GET
URL
headers
요청
TCP connection
응답
response
200
index.html
클라이언트
서버

http 모듈

- Nodejs 를 통해 서버를 구축하는 방법
  - http
  - express
- 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
  - response 객체 : 응답 메시지를 작성할 때 두 번째 매개변수로 전달되는 객체
  - request 객체 : 응답 메시지를 작성할 때 첫 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

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

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

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

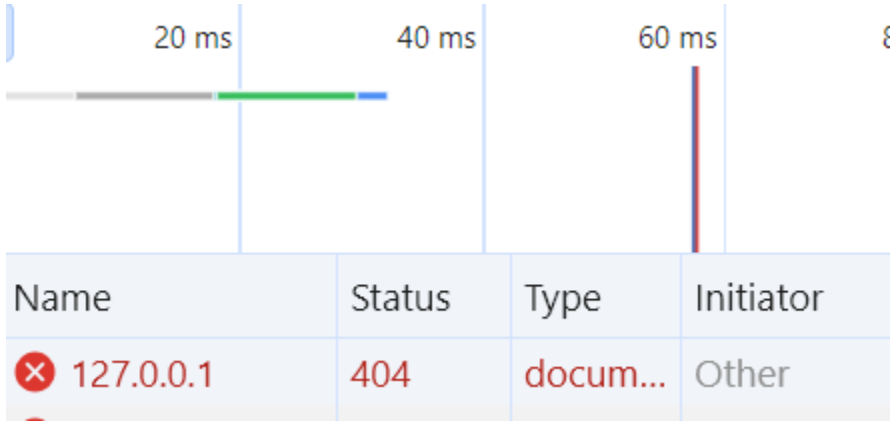
```
const http = require('http');

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' );
});
```

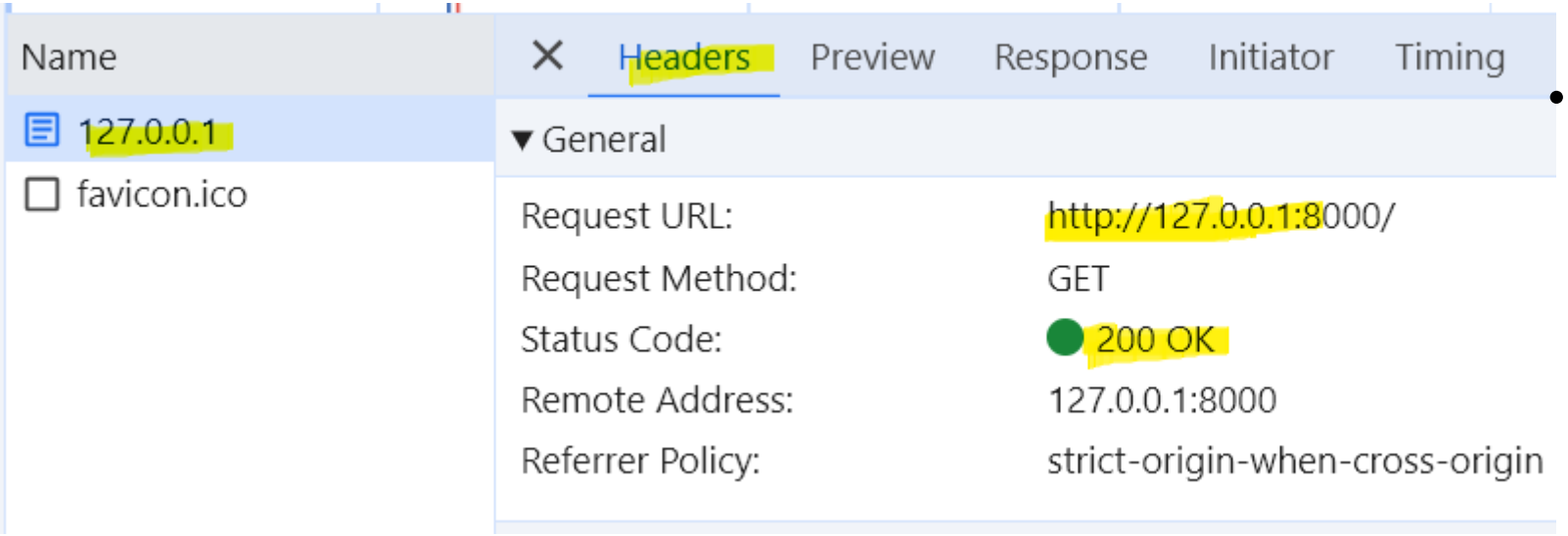
html 파일 전송

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

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

const server = http.createServer((req, res) => {
    try {
        const data = fs.readFileSync("index.html");
        res.writeHead(200);
        res.write(data);
        res.end();
    } catch (err) {
        console.error(err);
        res.writeHead(404);
        res.write(err.message);
        res.end();
    }
});

server.listen(8000, () => {
    console.log(`http://localhost:8000`);
});
```

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

16. Express로 서버만들기

- 웹 서버를 생성하는 것과 관련된 기능을 담당하는 프레임워크
- 웹 애플리케이션을 만들기 위한 각종 메소드와 미들웨어 등이 내장되어 있다.

- http 모듈 이용 시 코드의 가독성⬇, 확장성⬇
  → 이를 해결하기 위해 만들어진 것이 Express 프레임워크

Express 설치

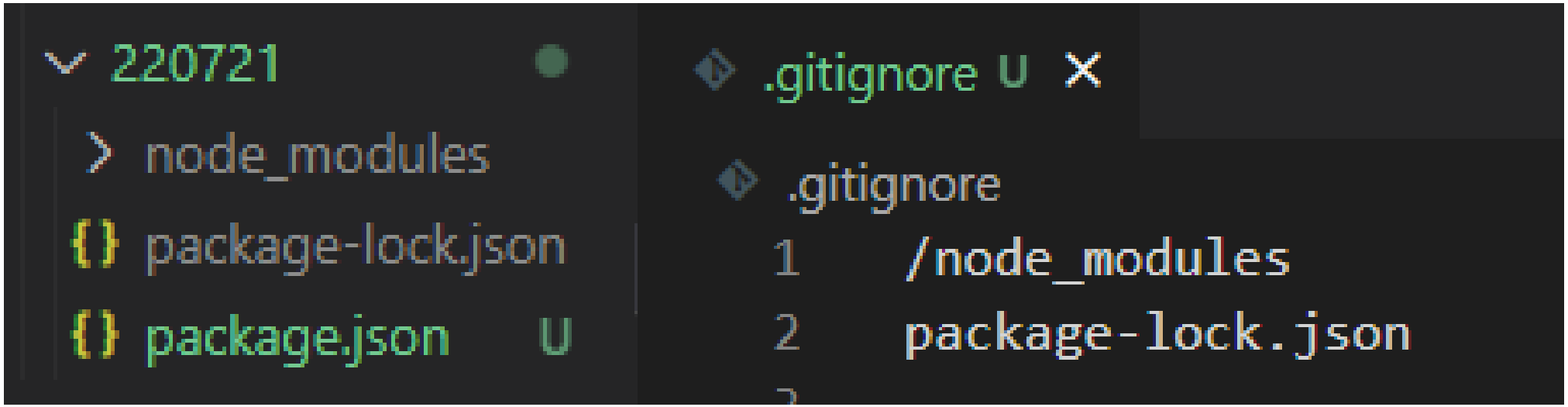
```
> npm install express
```

- npm_modules 가 만들어지며 express에 관련된 폴더가 생성
- package.json의 dendencies 에 express 기록

```
> node_modules
```

```
"dependencies": {
  "express": "^4.18.1"
}
```

.gitignore


220721
node_modules
package-lock.json
package.json U
.gitignore U
.gitignore
1 /node_modules
2 package-lock.json

npm 사용

1. (프로젝트를 진행할 폴더에서) `npm init` → (npm 을 사용할거야!)

- package.json : 프로젝트 폴더에서 npm init 명령어를 사용하는 순간 자동으로 생기는 파일, 모듈의 정보(버전 등)와 프로젝트 정보 등을 담고 있음

2. 사용하고 싶은 모듈 설치는 npm 모듈이름

- node_modlues : 모듈 저장 공간, 모듈을 설치하게 되면 이 곳에 설치됨

node_modules는 용량이 너무 커요

- node_modules는 아주 많은 폴더와 파일로 이루어져 있기 때문에 용량이 아주 큽니다.
- 깃허브에 올리지 않아요! 항상 .gitignore에 추가해주세요!
- 실제 모듈이 없으면 다른 컴퓨터에서는 사용할 수 없는데, node_modules를 github에 올리지 않는다면, 어떻게 할까요?
  - 모듈에 대한 모든 정보를 가지고 있는 package.json이 있기 때문에 문제 없습니다!
  - node_modules가 없어도 package.json이 있다면 node_modules를 설치할 수 있어요!
  - 바로 `npm install` 이라는 명령어를 통해서요!

.gitignore

.gitignore?

- Git 버전 관리에서 제외할 파일 목록을 지정하는 파일

- Git 관리에서 특정 파일을 제외하기 위해서는 git에 올리기 전에 .gitignore에 파일 목록을 미리 추가해야 한다.

.gitignore

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

3 directories, 3 files
```

*.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)

Express 사용

```
const express = require('express');
const app = express();
const PORT = 8000;

app.get('/', function (req, res) {
  res.send('hello express');
});

app.listen(PORT, function () {
  console.log(`Listening on port ${PORT}! http://localhost:${PORT}`);
});
```

Express 사용

- express()
  - Express 모듈이 export 하는 최상위 함수로, express application을 만듦
- app 객체
  - Express() 함수를 호출함으로써 만들어진 express application

```
1   const express = require('express');
2   const app = express();
```

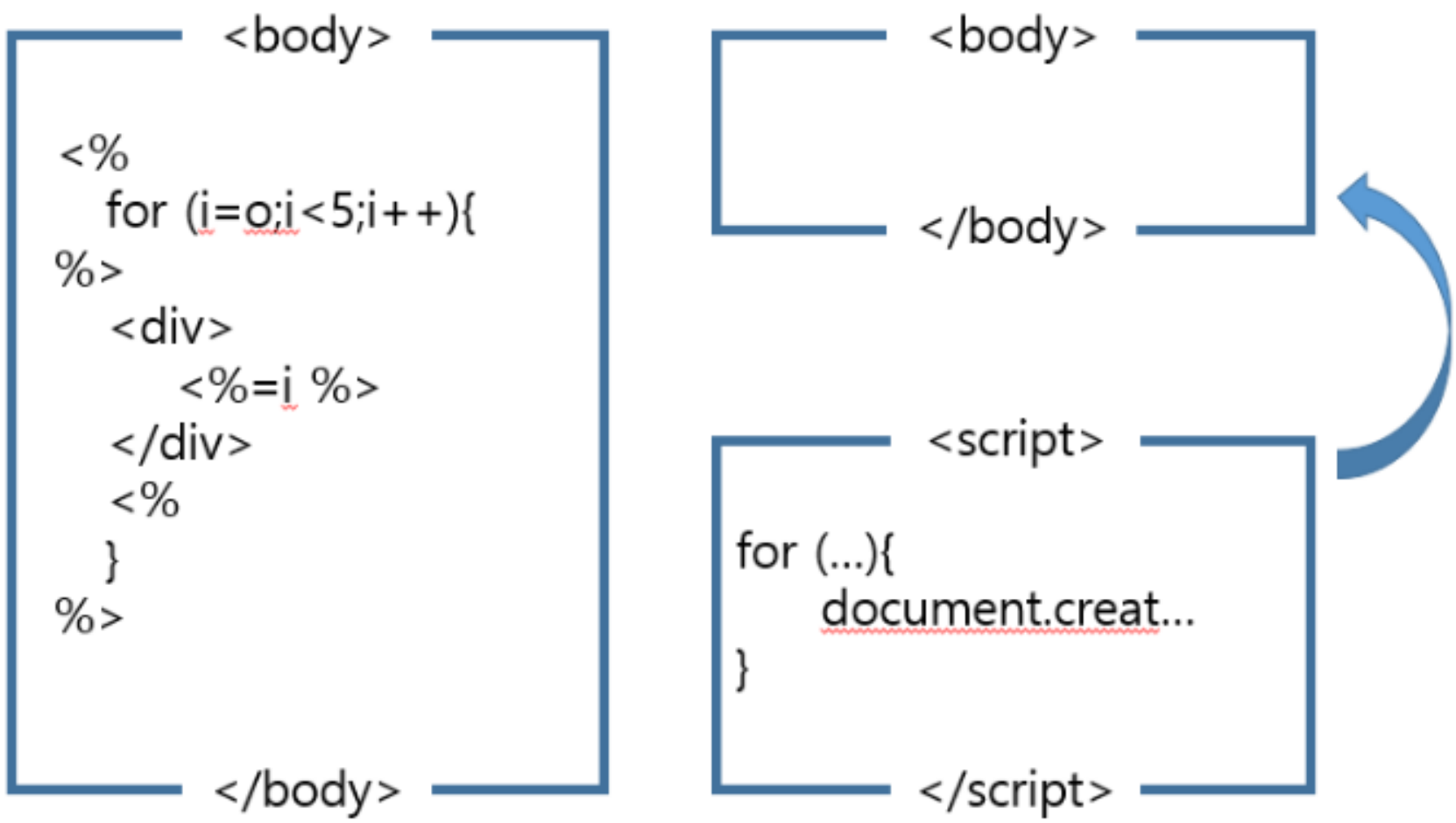
RESTful

- Representational State Transfer의 약자
- REST는 기본적으로 이러한 http 서비스를 구축하기 위한 규약입니다. 간단한 http 프로토콜 원리를 사용하여 읽기 업데이트를 생성하고 데이터를 삭제할 수 있도록 지원합니다.
- 우리는 이러한 작업을 모두 CRUD 작업이라고 부릅니다

17. EJS 템플릿

- 템플릿 엔진
  - 문법과 설정에 따라 파일을 html 형식으로 변환시키는 모듈
- ejs
  - Embedded JavaScript 의 약자로, 자바스크립트가 내장되어 있는 html 파일
  - 확장자는 .ejs

ejs 템플릿


<body>
<%
for (i=o;i<5;i++){
%>

<%=i %>

<%
}
%>
</body>
<body>
</body>
<script>
for (...){
document.creat...
}
</script>

ejs 템플릿

```
$ npm install ejs
```

```
app.set('view engine', 'ejs');
app.set('views', './views');
```

ejs 템플릿

```
const express = require("express");
const app = express();
const PORT = 8000;

app.set("view engine", "ejs");
app.set("views", "./views");

app.get("/", (req, res) => {
    res.send("Hello Express");
});

app.get("/test", (req, res) => {
    res.render("test");
});

app.listen(PORT, () => {
    console.log(`http://localhost:${PORT}`);
});
```

ejs 템플릿 설정

ejs 템플릿 렌더링

ejs 템플릿

```
<html>
    <head>
        <title>EJS TEST</title>
    </head>
    <body>
        <% for (var i = 0; i < 5; i++) { %>
            <h1>안녕</h1>
        <% } %>
    </body>
</html>
```

ejs 문법 사용하기

```
<% %>
```

- 무조건 자바스크립트 코드가 들어가야 하고, 줄바꿈을 할 경우에는 새로운 <% %> 를 이용

```
<%= %>
```

- 값을 템플릿에 출력할 때 사용

```
<%- include('view의 상대주소') %>
```

- 다른 view 파일을 불러올 때 사용

미들웨어

- 요청이 들어옴에 따라 응답까지의 중간 과정을 함수로 분리한 것
- 서버와 클라이언트를 이어주는 중간 작업
- use() 를 이용해 등록할 수 있다.
- app.use() 요청을 받을 때마다 실행하는 코드

```
app.set('view engine', 'ejs');
app.use('/views', express.static(__dirname + '/views'));
```

미들웨어 - static

- 이미지, CSS 파일 및 JavaScript 파일(front)과 같은 정적 파일 제공
- Express 에 있는 static 메소드를 이용해 미들웨어로 로드
- 사용이유 : 이미지, css 파일 및 Javascript 파일과 같은 정적 파일을 제공하는 역할

- 사용방법 : use() 를 이용해 등록한다

```
app.use('/static', express.static(__dirname + '/static'));
```

ejs 템플릿

```
const express = require("express");
const app = express();
const PORT = 8000;

app.set("view engine", "ejs");
app.set("views", "./views");
app.use("/public", express.static(__dirname + "/public"));

app.get("/", (req, res) => {
    res.send("Hello Express");
});

app.get("/test", (req, res) => {
    res.render("test");
});

app.listen(PORT, () => {
    console.log(`http://localhost:${PORT}`);
});
```

정적 파일 로드 코드
[keyword] 미들웨어, static