

K-Digital Training 웹 풀스택 과정

Node.js란?

WITH 팀 리처드



Node.js란?

Node.js



- 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

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?

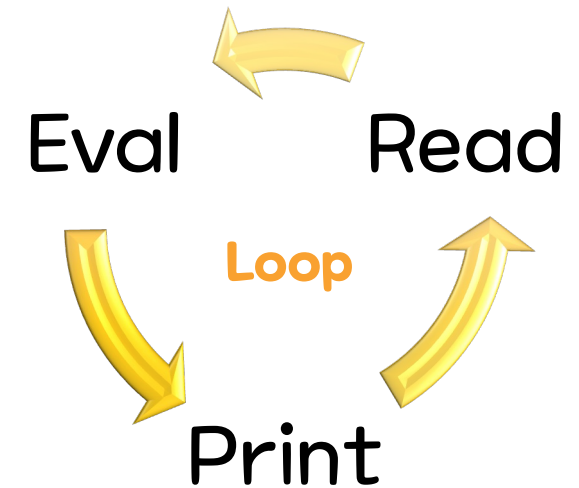
- Javascript로 개발된 각종 모듈의 설치, 업데이트, 구성, 제거 과정을 자동화하여 관리해주는 기능

REPL

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

윈도우에서의 cmd, 맥에서의 terminal처럼
노드에는 REPL 콘솔이 있다.

```
C:\Users\linda>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

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",  
  "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

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

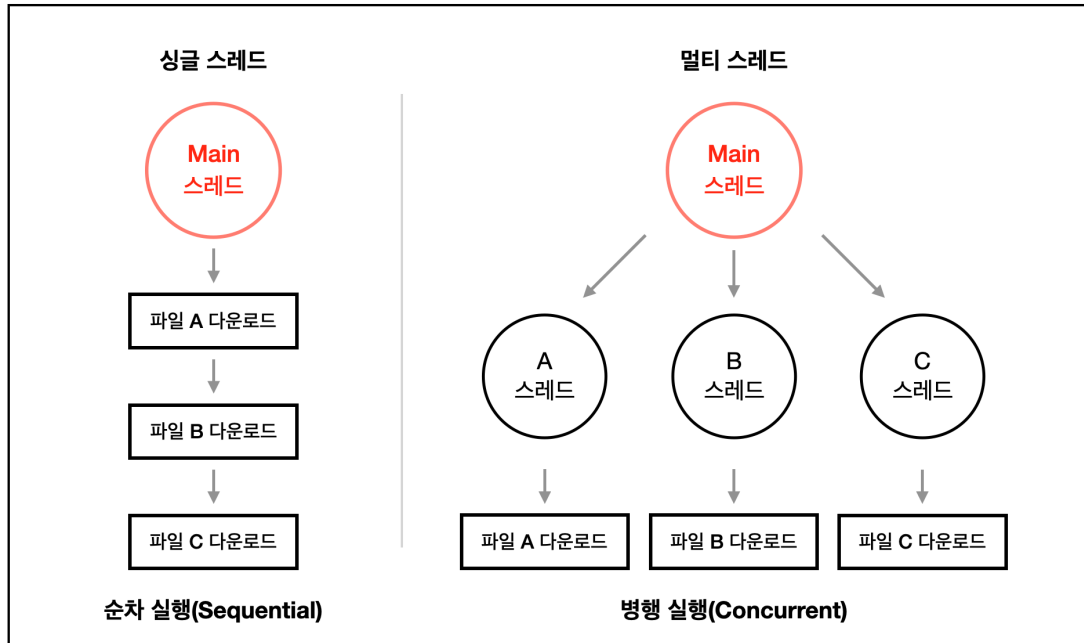
프로세스

- 실행 중인 프로그램
- 운영체제에서 할당하는 작업의 단위

Thread(스레드)

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

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 - Single Thread

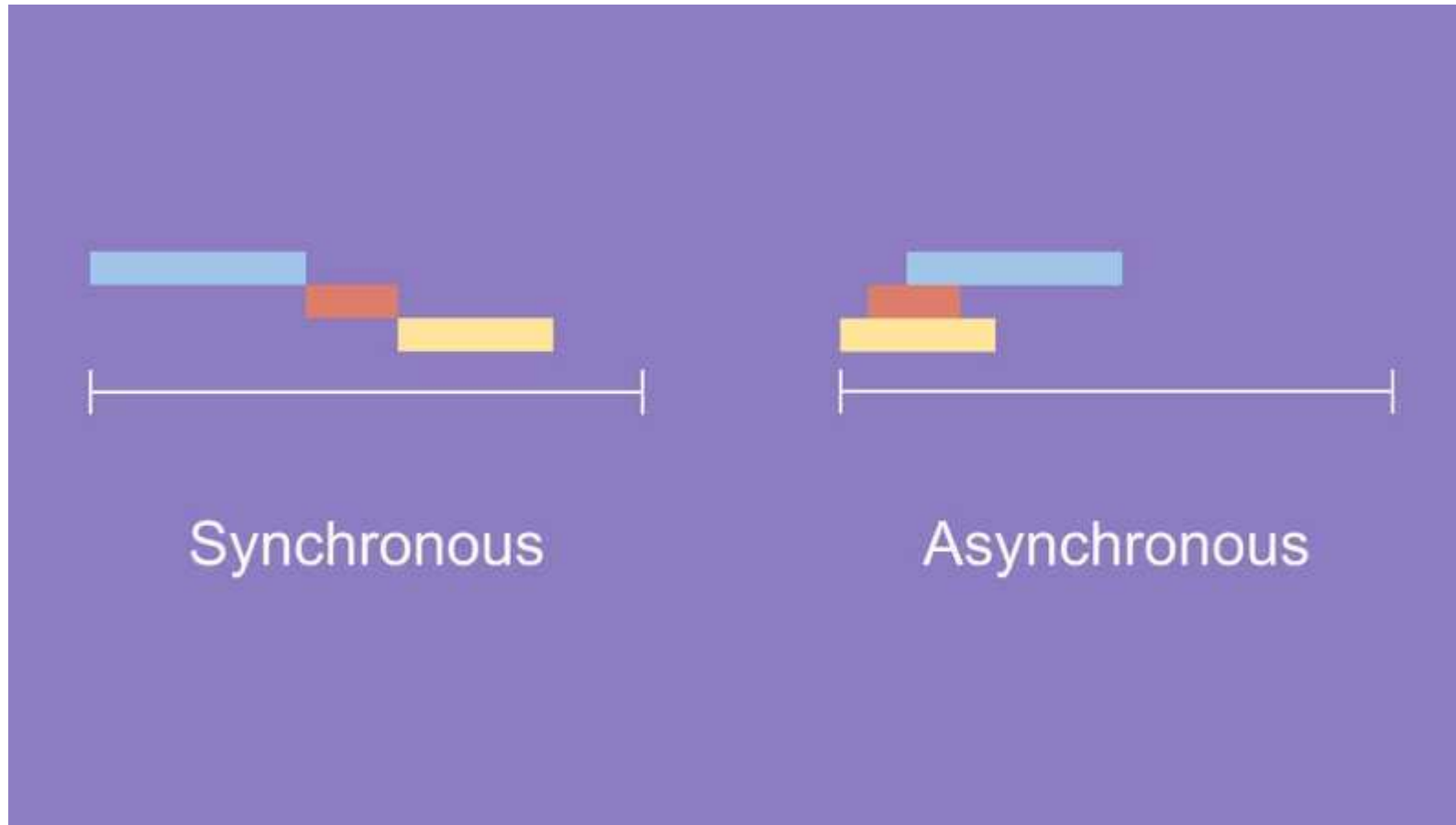
~~상용 스레드?
멀티 스레드 프로세스?
CPU?~~

에러를 처리하지 못하면 프로그램이 아예 중단됨



예외처리의 중요성 ↑

특징 3) Non-blocking I/O



특징 3) Non-blocking I/O

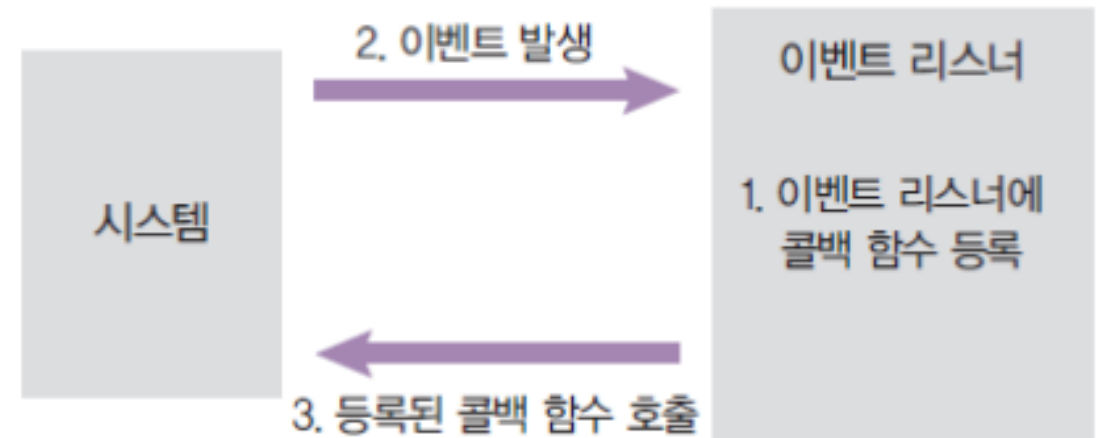
- 동기 (Synchronous)
 - 데이터의 요청과 결과가 한 자리에서 동시에 일어나는 것
 - 시간이 얼마가 걸리든지 요청한 자리에서 결과가 주어진다.
 - 한 요청에 서버의 응답이 이루어질 때까지 계속 대기해야 한다.
- 비동기 (Asynchronous)
 - 동시에 일어나지 않는 것
 - 요청한 후 응답을 기다리지 않고 다른 활동을 한다.

특징 3) Non-blocking I/O

- I/O 작업 : 파일 시스템 접근 (읽기, 쓰기, 만들기 등), 네트워크 요청
- Node.js는 표준 라이브러리의 모든 I/O 메서드를 비동기 방식으로 제공한다.

특징4) 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

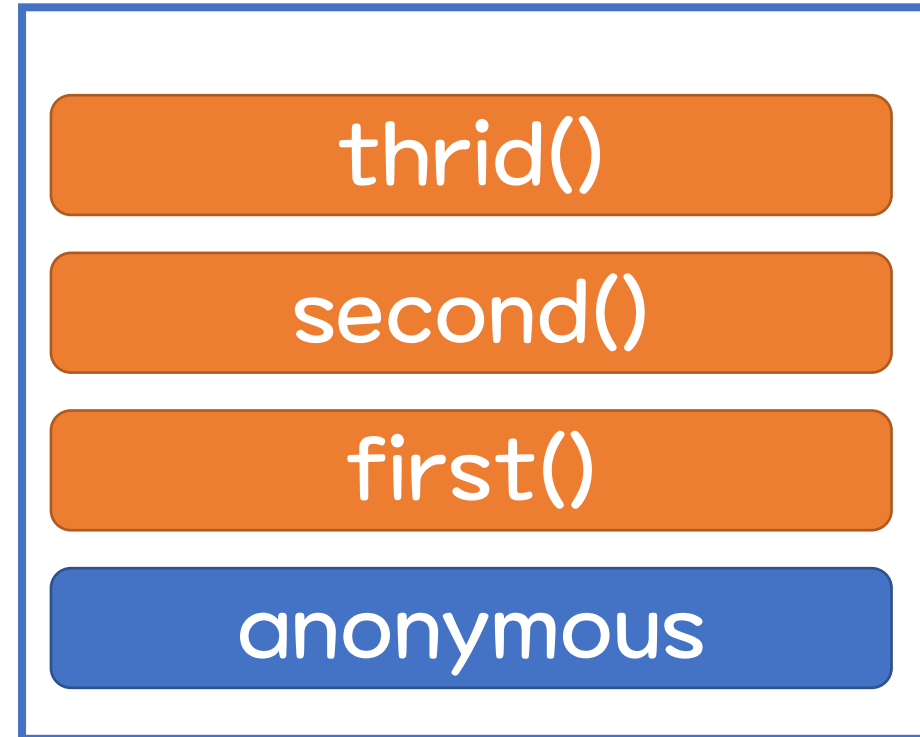


Call Stack

```
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



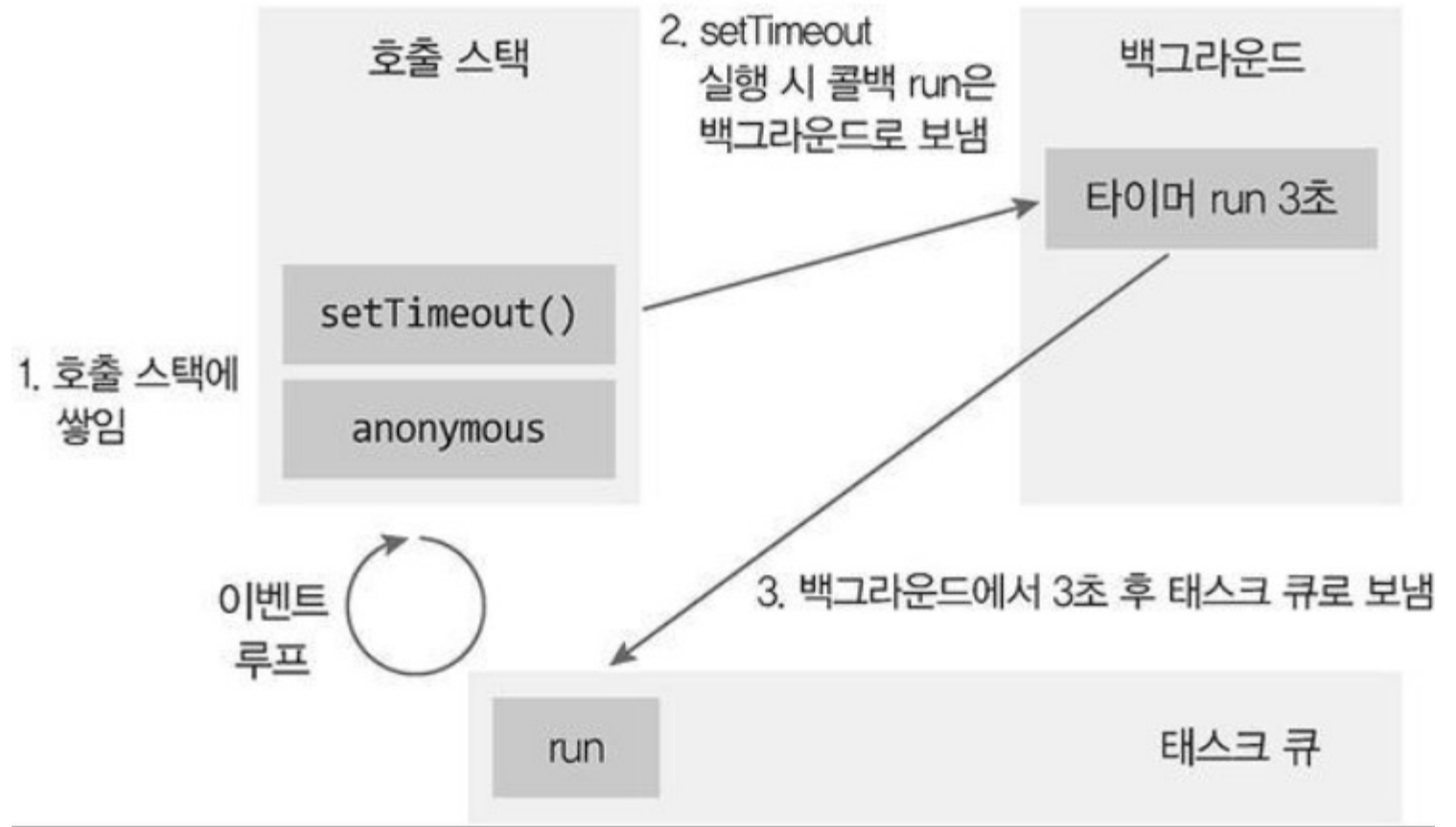
Call Stack (호출 스택)
LIFO 방식

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

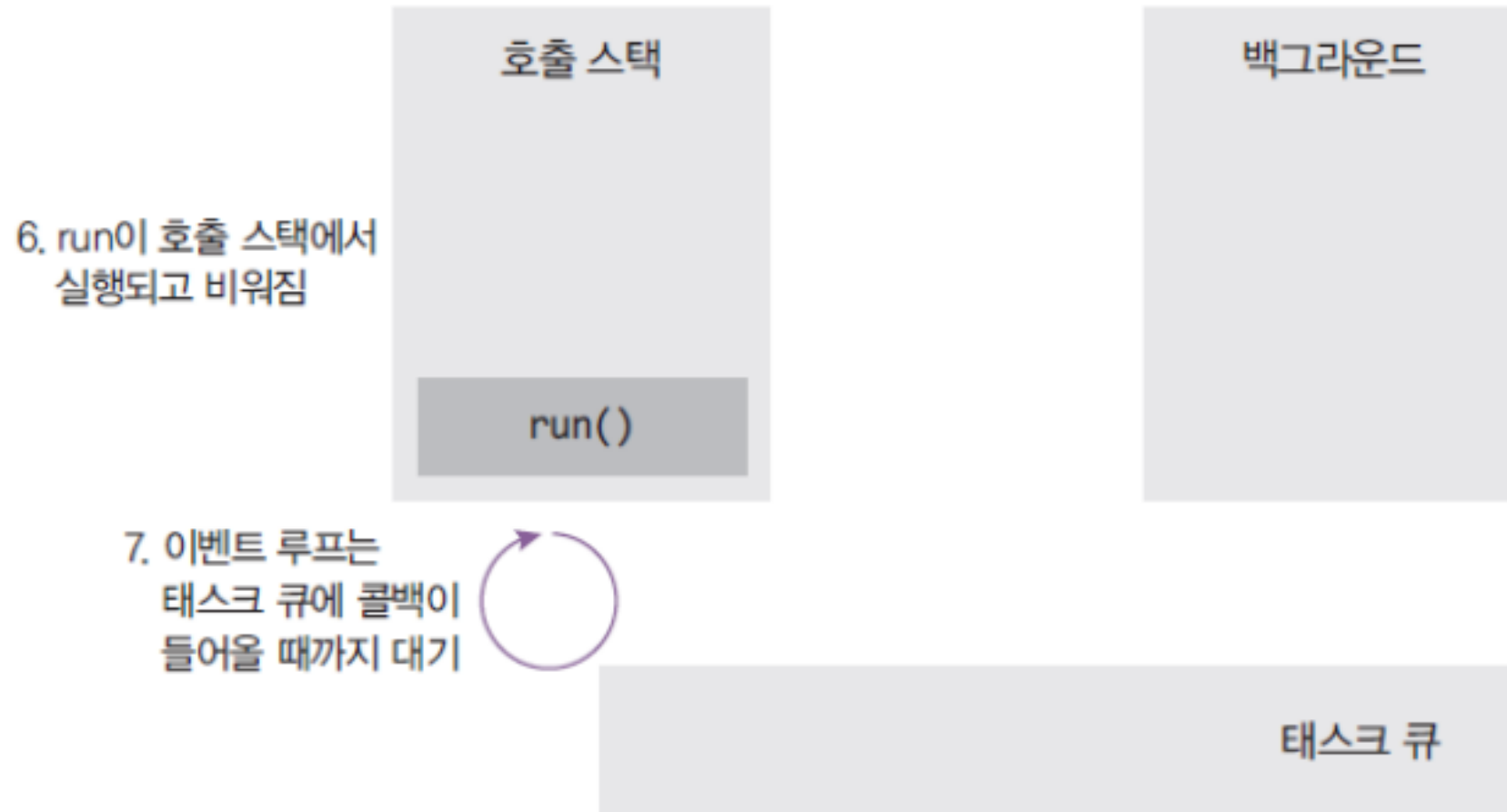
5. 이벤트 루프가
태스크 큐의 콜백을
호출 스택으로 올림



run

태스크 큐

Event Loop



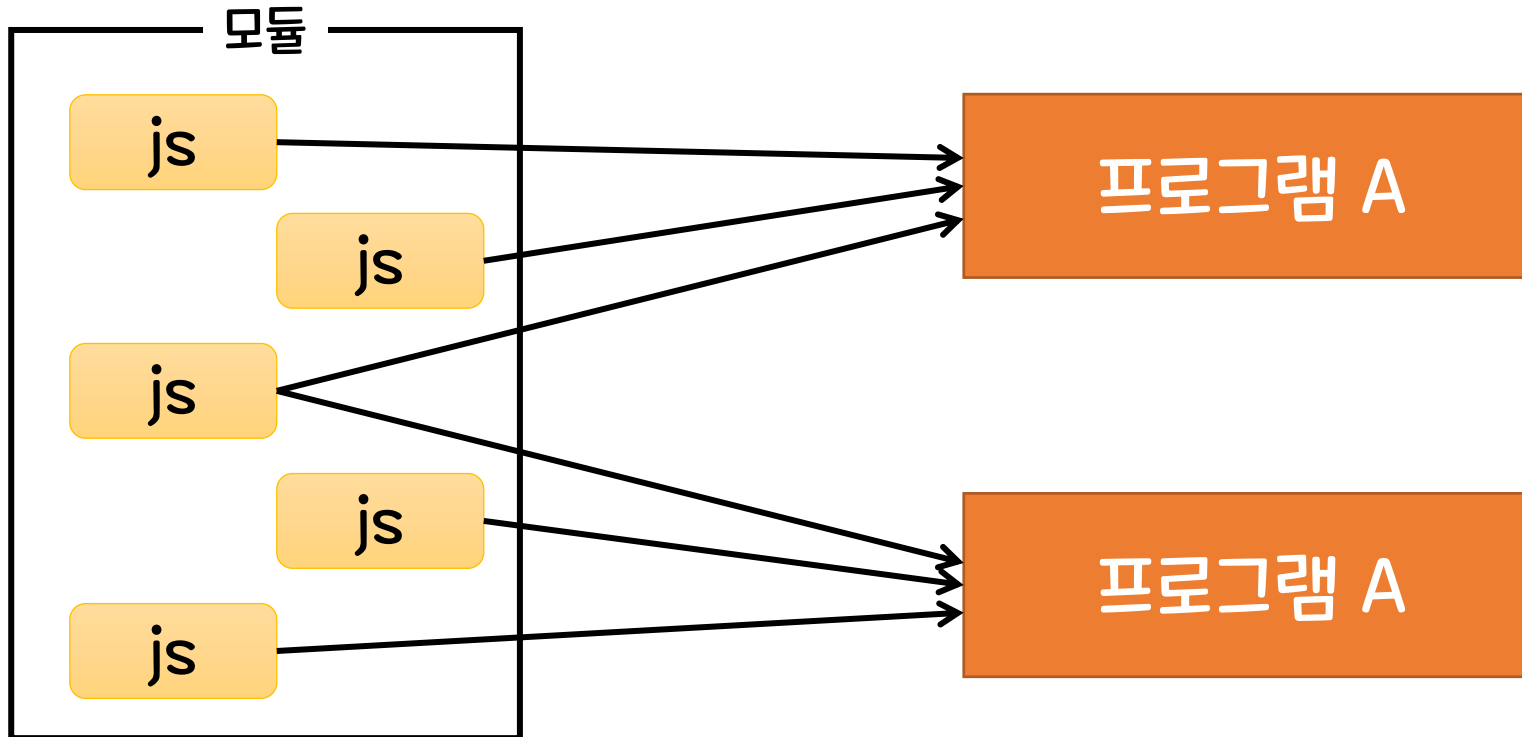
Node.js 의 역할

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

모듈 (Module)

모듈이란?

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



모듈의 장점

- 코드 추상화
- 코드 캡슐화
- 코드 재사용
- 의존성 관리



모듈 만들기

하나의 모듈 파일에 여러 개 만들기

```

1  const a = "a 변수";
2  const b = "b 변수";
3
4  module.exports = {
5      a,
6      b
7  };

```

하나의 모듈 파일에 하나 만들기

```

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

```


모듈 불러오기

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

```
1  const a = "a 변수";  
2  const b = "b 변수";  
3  
4  module.exports = {  
5    a,  
6    b  
7  };|
```



```
1  const a = "a 변수";  
2  const b = "b 변수";  
3  
4  export { a, b };|
```

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

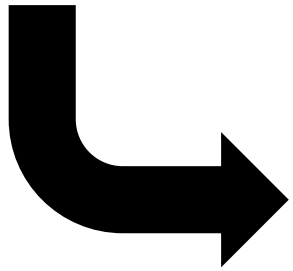


```
2  
3  function connect() {  
4    return a + b;  
5  }  
6  
7  export default connect;|
```

ES2015 모듈

import ~ from ~ : 모듈 가져오기

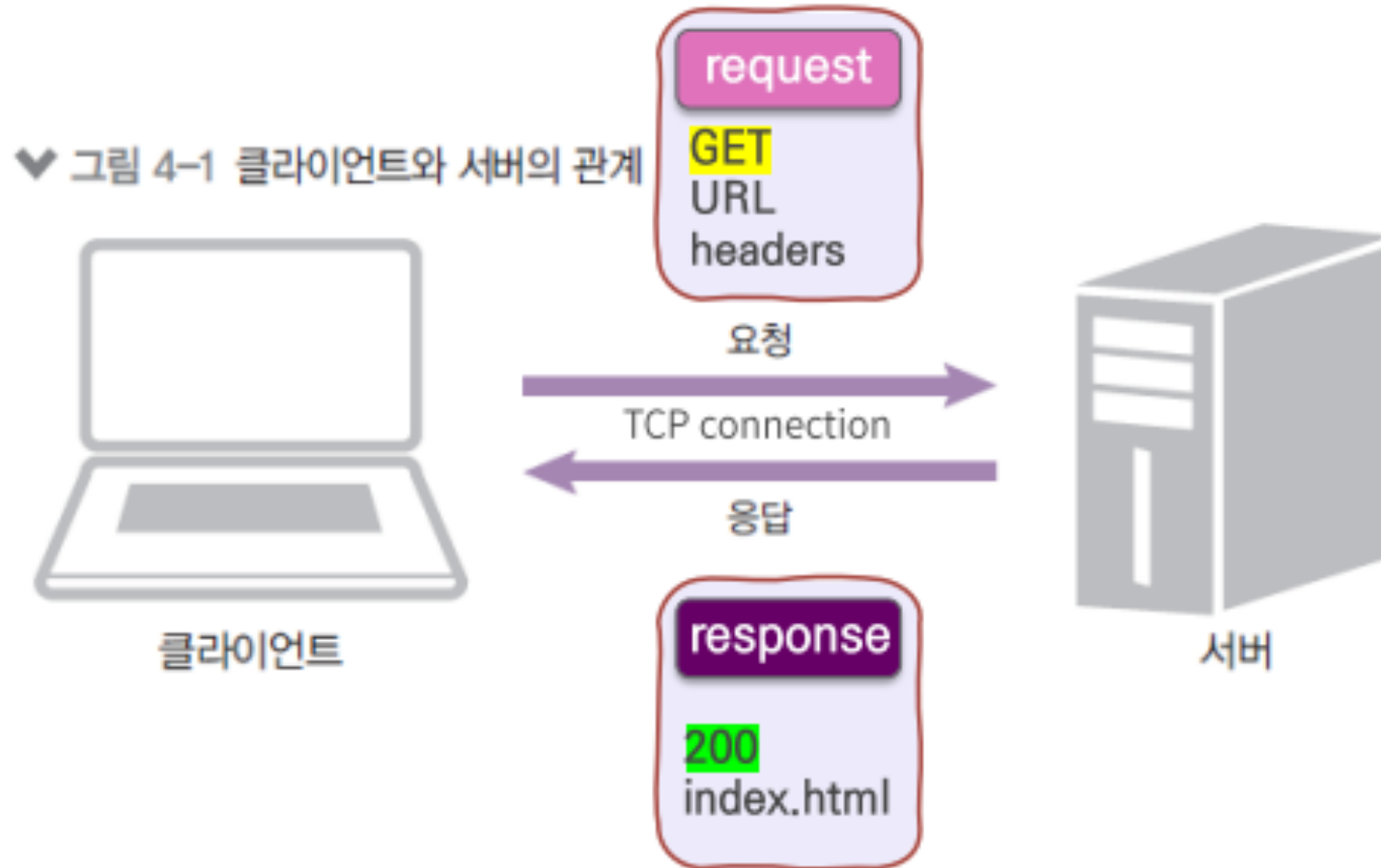
```
const { a, b } = require("./var.js");  
const returnString = require("./func.js");
```



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

서버 만들기

http 통신



http 모듈

- 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
 - 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' );
});
```

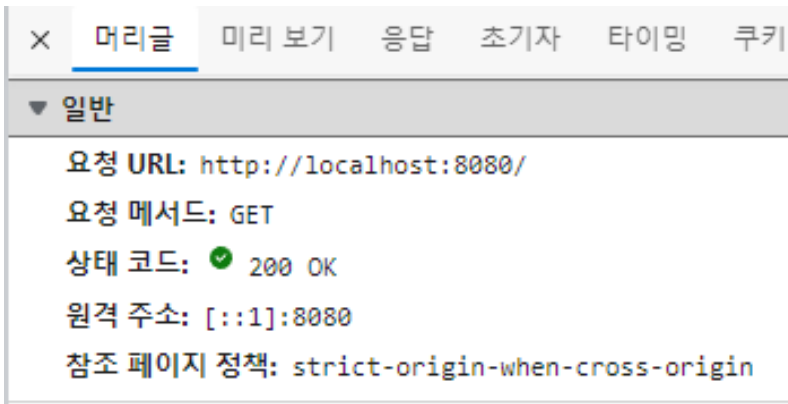
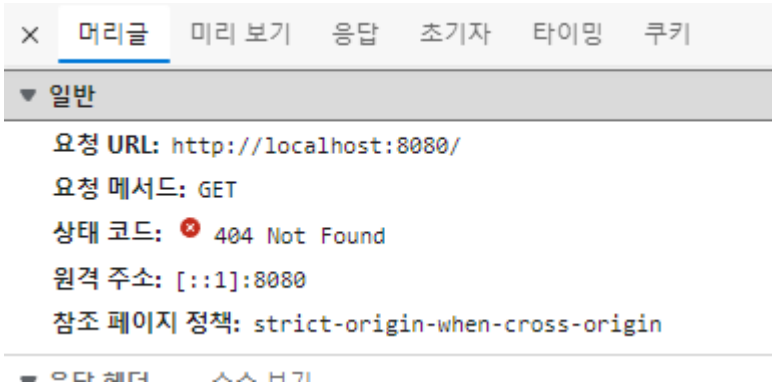
html 파일 전송

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

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

const server = http.createServer( async function(req, res){
  try {
    const data = await fs.readFile('./5_http.html');
    res.writeHead(200);
    res.end(data);
  } catch(err) {
    console.error(err);
    res.writeHead(404);
    res.end(err.message);
  }
});
```

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

Express 모듈

Express

- 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.json**의 **dependencies** 에 **express** 기록

```
> node_modules
```

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

.gitignore

```
220721
  > node_modules
  {} package-lock.json
  {} package.json U

.gitignore U X
.gitignore
1  /node_modules
2  package-lock.json
```

[복습] .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.

[복습] .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
```

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 application**을 만듦
- **app 객체**
 - Express() 함수를 호출함으로써 만들어진 **express application**

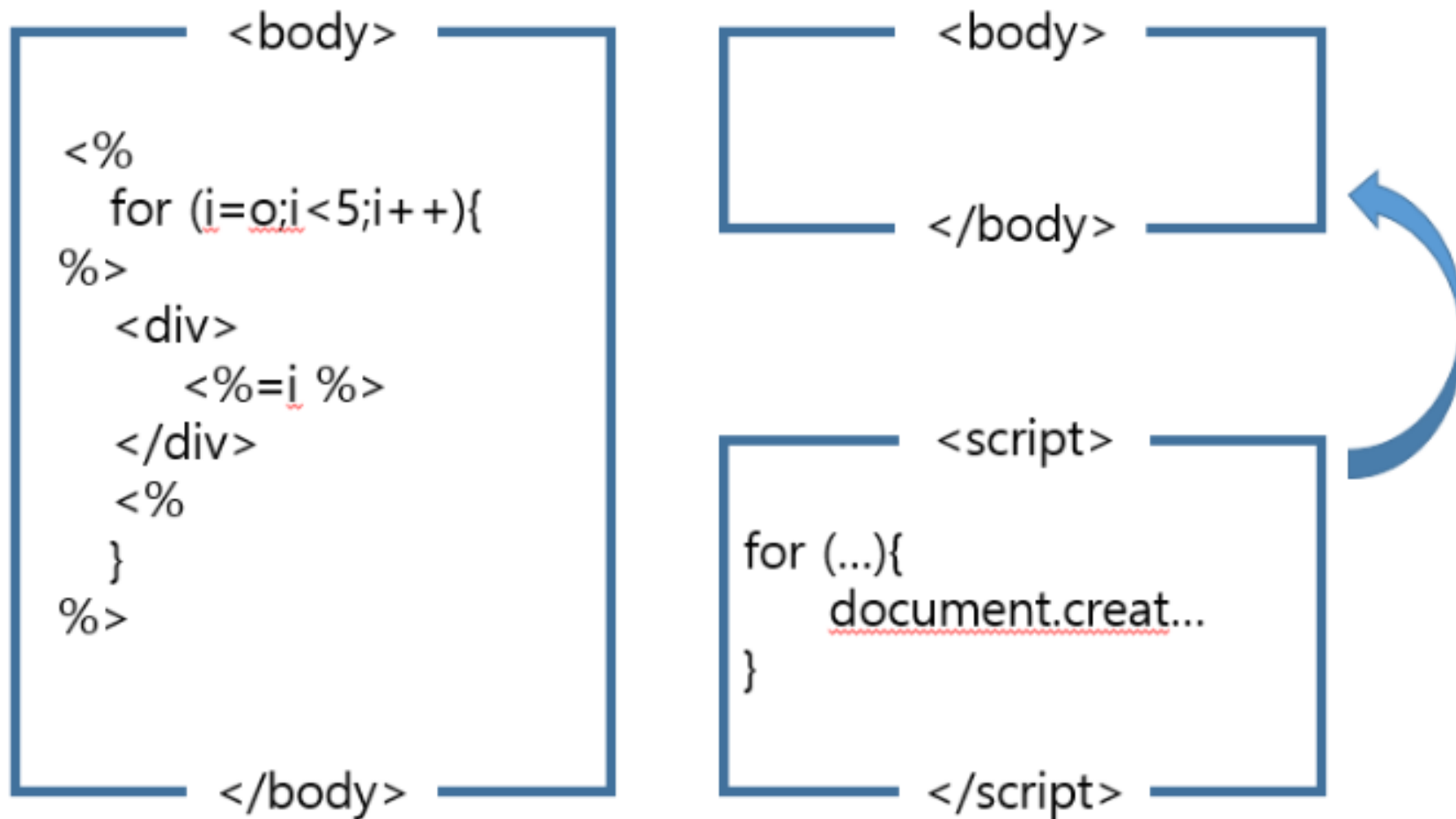
```
1  const express = require('express');  
2  const app = express();
```

템플릿 엔진

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



ejs 템플릿

```
$ npm install ejs
```

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

ejs 템플릿

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

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

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

app.get('/test', function (req, res) {
  res.render('test');
});

app.listen(PORT, function () {
  console.log(`Listening on port ${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 이용해야 한다.

```
<%= %>
```

- 값을 템플릿에 출력할 때 사용

```
<%- include('view의 상대주소') %>
```

- 다른 view 파일을 불러올 때 사용

미들웨어

- 요청이 들어옴에 따라 응답까지의 **중간 과정**을 **함수로 분리**한 것
- **서버와 클라이언트를 이어주는** **중간 작업**
- **use()** 를 이용해 **등록**할 수 있다.

```
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
- 등록 방법

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

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

app.get('/test', function (req, res) {
  res.render('test');
});

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



정적 파일 로드 코드

[keyword] 미들웨어, static