Seoul Software ACademy

x CODINGON

Express

- API, Express, ejs 모듈 -

SeSAC 도봉 1기
웹 풀스택 과정

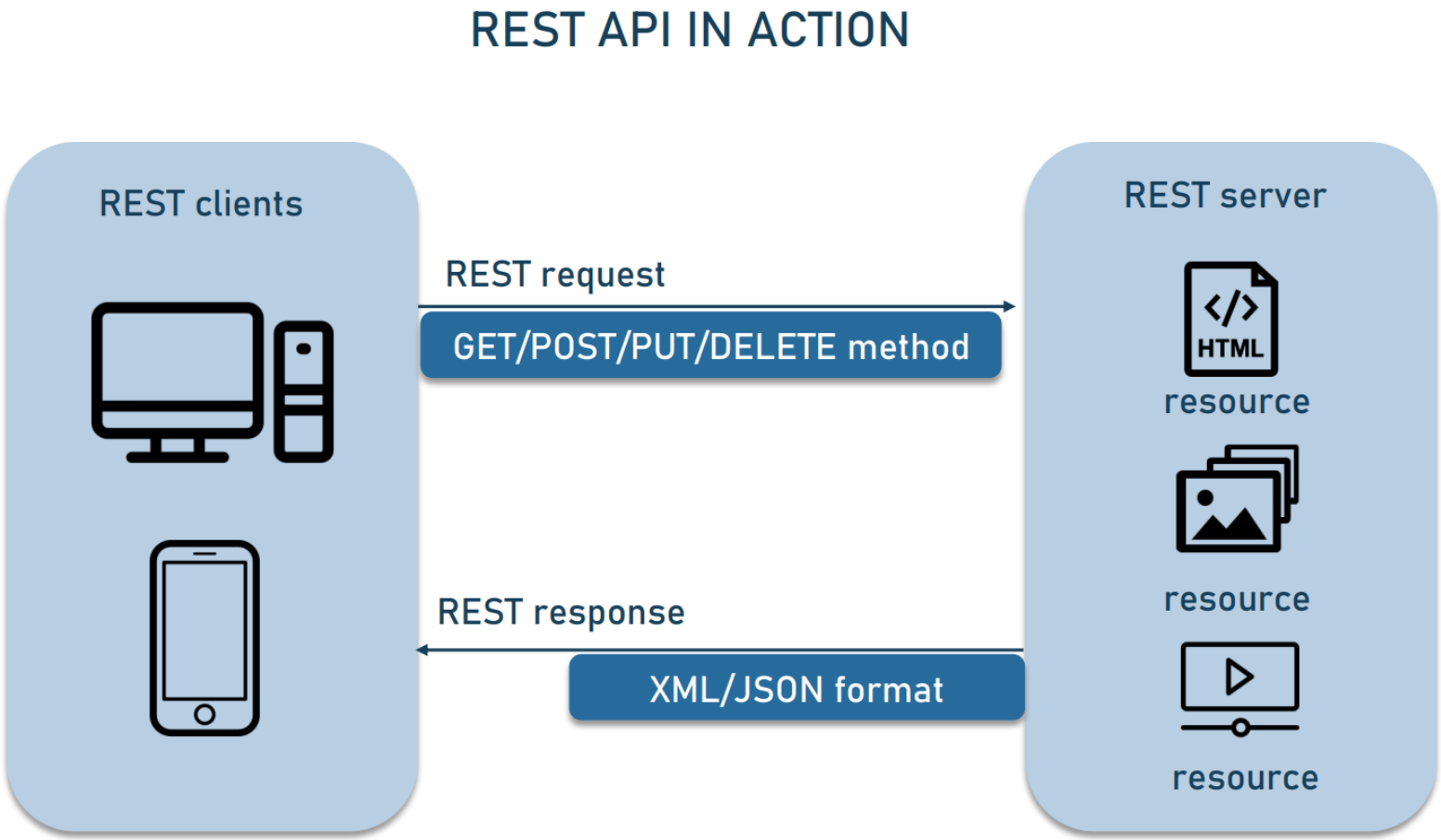
API

API

- **A**pplication **P**rogramming **I**nterface

프론트엔드에서 백엔드로 데이터를 요청하면 (request),
백엔드에서 프론트엔드로 응답(response)

REST API


REST API IN ACTION
REST clients
REST request
GET/POST/PUT/DELETE method
REST server
HTML
resource
resource
REST response
XML/JSON format
resource
altexsoft

REST API

- Representational State Transfer API
- HTTP 통신에서 CRUD 요청을 resource + method로 표현, 특정한 형태로 전달하는 방식.
- resource(자원)? method?
  - resource : 내가 필요한 데이터, 모든 데이터들은 특정 장소에 보관되어 있음 (HTTP URI를 통해 자원을 명시)
  - method : CRUD 중 어떤 방식으로 요청할건지 (next page!)

(참고) CRUD

	Create	Read	Update	Delete
블로그 게시글	생성	읽기	수정	삭제

“REST”

GET	/movies	Get list of movies
GET	/movies/:id	Find a movie by its ID
POST	/movies	Create a new movie
PUT	/movies	Update an existing movie
DELETE	/movies	Delete an existing movie

Method, 어떤 방식으로?

GET 가져와! (읽기, 검색 - Read)

특정 뉴스기사 클릭했을 때 해당 뉴스의 제목 & 내용 가져오기

POST 입력, 등록! (생성 - Create)

댓글 등록

게시판 글 등록

Method, 어떤 방식으로?

PUT 모두 수정! (Update)

원래 있는 글 {title:'~', content:'~'} 모두 수정

(POST로 대체해서 많이 사용)

DELETE 삭제! (Delete)

PATCH 일부 수정! (Update)

원래 있는 글 {title:'~'} 하나만 PATCH 날려도 전체 수정됨

(POST로 대체해서 많이 사용)

http 통신

http 통신

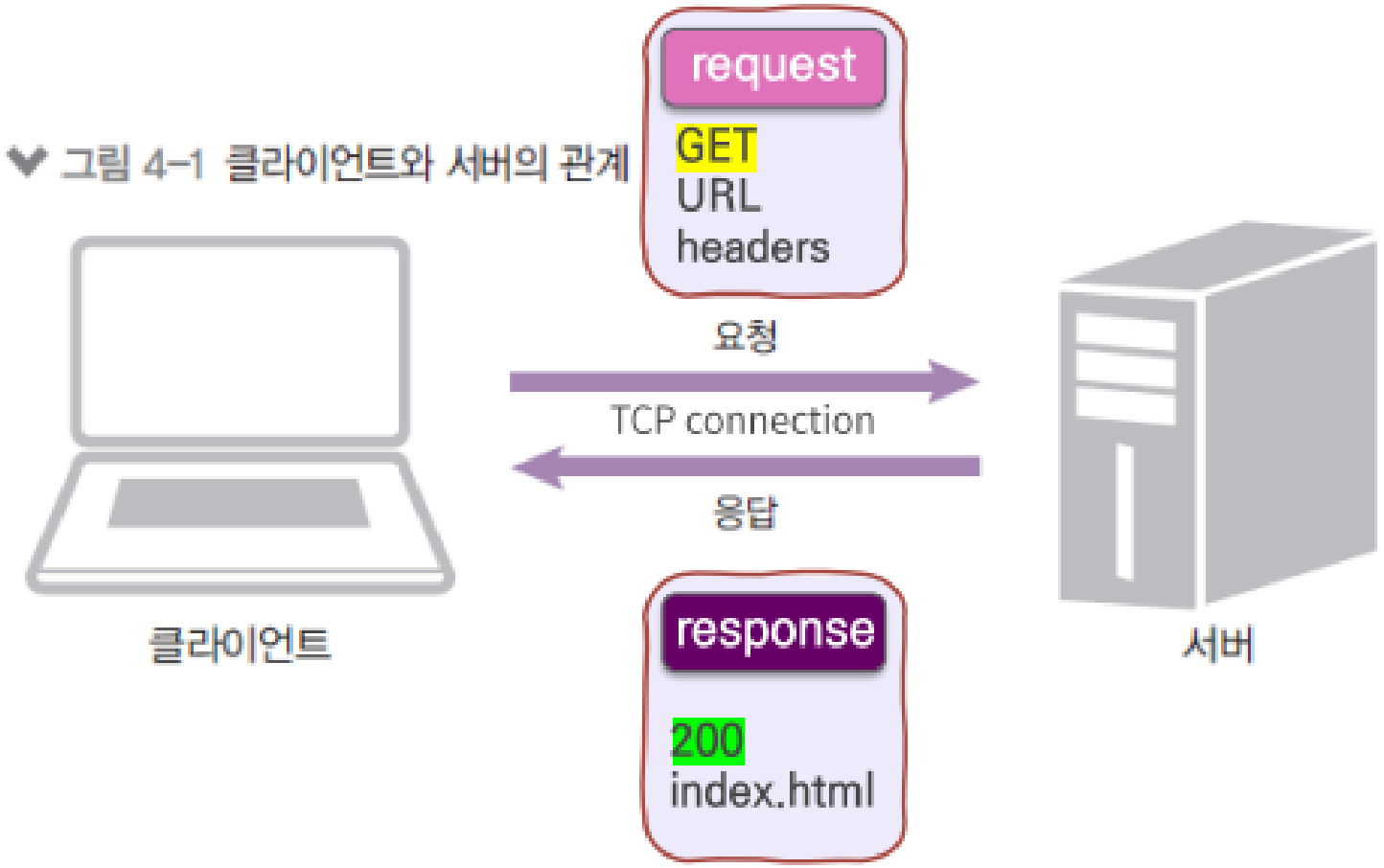


그림 4-1 클라이언트와 서버의 관계
request
GET
URL
headers
요청
TCP connection
응답
response
200
index.html
클라이언트
서버

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

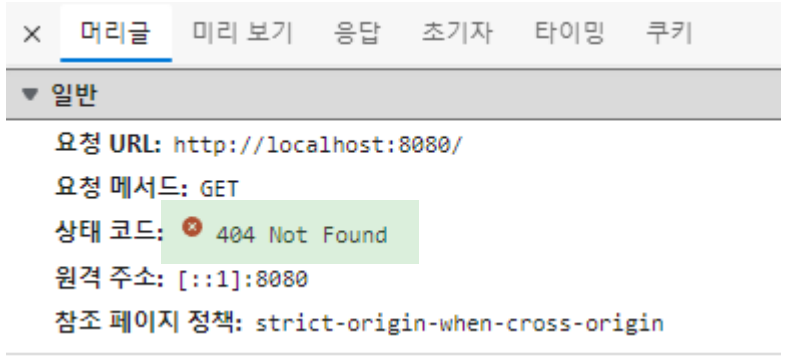
```html
<html>
    <head>
        <title>http모듈</title>
    </head>
    <body>
        <h1>Hello http</h1>
        <p>p태그</p>
    </body>
</html>
```

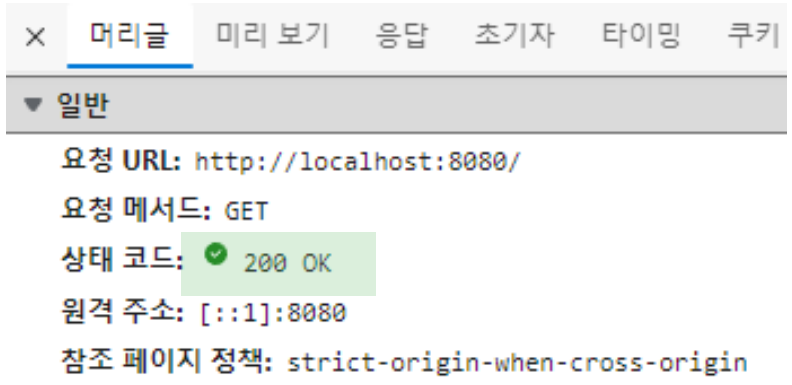
```js
const http = require("http");
const fs = require("fs");

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

HTTP Status Codes

Level 200 (Success)

200 : OK

201 : Created

203 : Non-Authoritative Information

204 : No Content

Level 400

400 : Bad Request

401 : Unauthorized

403 : Forbidden

404 : Not Found

409 : Conflict

Level 500

500 : Internal Server Error

503 : Service Unavailable

501 : Not Implemented

504 : Gateway Timeout

599 : Network timeout

502 : Bad Gateway

Express 모듈

Express

- **웹 서버를 생성**하는 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"
}
```

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**

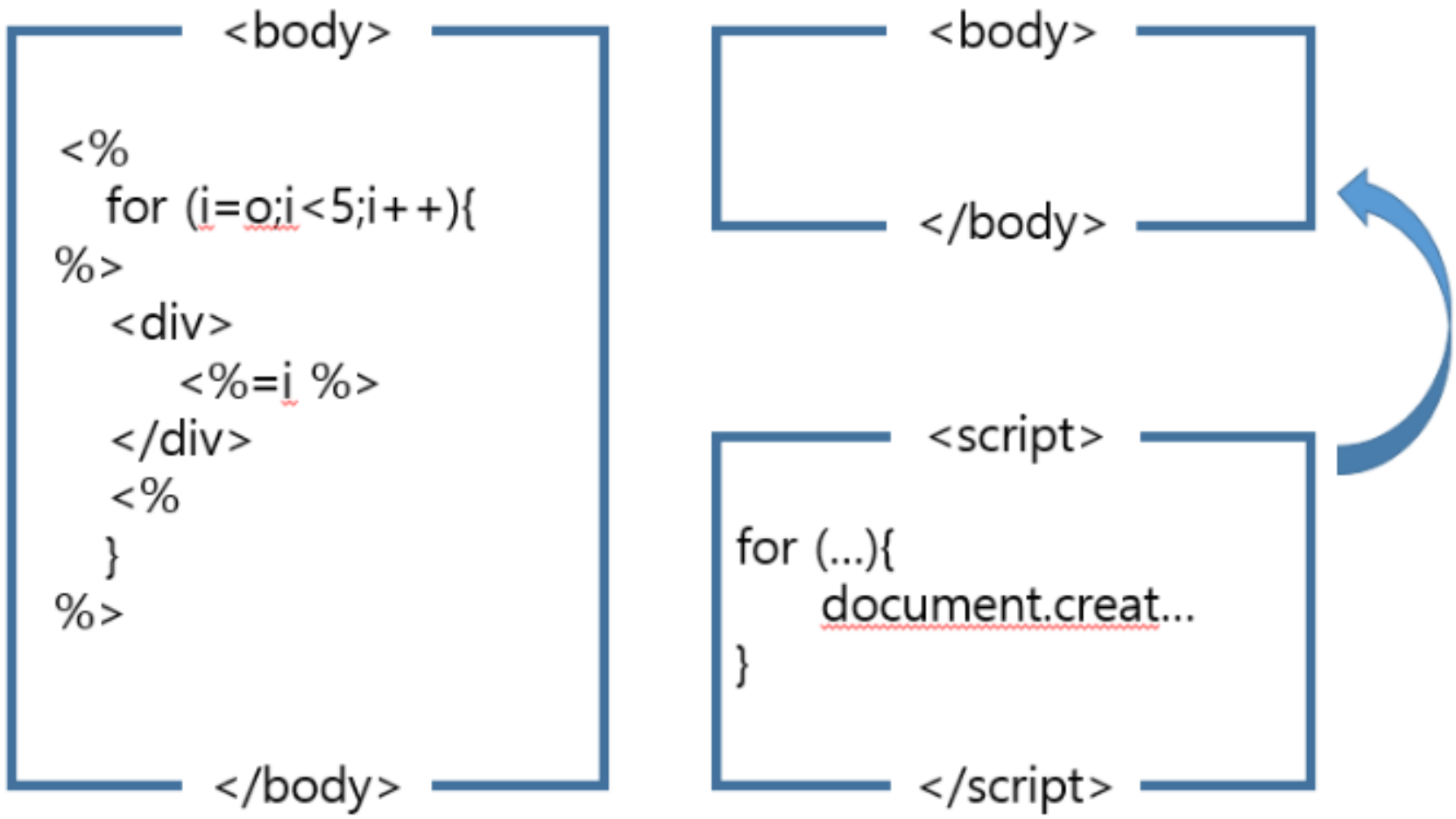
```
1   const express = require('express');
2   const app = express();
```

템플릿 엔진

EJS 템플릿

- 템플릿 엔진
  - 문법과 설정에 따라 **파일을 html 형식으로 변환**시키는 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 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.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