웹 서버 측 프로그래밍 1

1장

- 소개

: node.js , express, mongodb를 이용한 블로그 응용 개발

: 사용자 인증, 데이터 검증, 비동기적 자바스크립트, 암호해싱 ..

- 블로그 응용

: 기능 - 회원 등록, 로그인, 블로그 포스팅

- Node.js란?

: chrome 브라우저의 v8 자바스크립트 엔진을 수정하여 서버 프로그램용 실행 환경으로 제작한 것

: 장점 - v8 엔진의 속도가 빠름, node.js를 비 동기적 프로그래밍 방식 장려, 다양한 라이브러리 존재

- 첫 웹 서버 만들기

<코드>

const http=require('http'); //http 모듈 가져오기

const server=http.createServer((req, res)⇒{ //서버 생성

console.log(req.url); //요청 url을 콘솔에 출력

res.end("Hello Node.js"); //클라이언트에 응답 보내기

});

server.listen(3000); //서버를 3000번 포트에서 시작

⇒ 접속 : url 창에 localhost:3000 입력

- 자바스크립트 모듈

⇒ 모듈 : 자바스크립트에서 관련 있는 변수, 함수 등을 모아 놓은 파일

수입하는 파일

```
const ss=require('./sam');
ss.A();
console.log(ss.persons);
```

수출하는 파일(sam.js)

```
function A() {...}
function B() {...}
persons=[....]

module.exports={A,
persons};
```

⇒ 패키지 : 관련 모듈들을 모아 놓은 디렉터리, package.json 파일에 존재

⇒ 내장 패키지 : http,fs,util,net 등

- 서버 코드 설명

⇒ createServer 함수 : 함수를 인수로 받음 → 이 함수를 콜백 함수라고 부름

⇒ req, res 인수 : 각각 httprequset, httpresponse 객체

⇒ server.listen(port) : 서버를 시작하고 주어진 포트로부터 요청을 받아들이기 시작함.

- request와 response 객체

⇒ 다양한 url에 서로 다른 응답 보내기

```
const server=http.createServer((req, res)⇒{
if (req.url == “/about”)
    res.end(’The about page’)
else if (req.url == “/contact”)
    res.end(’The about page’)
else if (req.url == “/”)
```

```
    res.end('The home page')
else {
    res.writeHead(404)
    res.end('page not found')
    }
});
```

- html 보내기

⇒ html 파일을 보내는 서버

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

const homePage=fs.readFileSync("./homepage.html");
const aboutPage=fs.readFileSync("./aboutpage.html");
const server=http.createServer((req, res)⇒{
if (req.url == "/about")
    res.end('The about page')
else if (req.url == "/contact")
    res.end('The about page')
else if (req.url == "/")
    res.end('The home page')
else {
    res.writeHead(404)
    res.end('page not found')
    }
});
```

⇒ 파일 시스템 패키지 : 파일 읽기 (readFileSync)

- 정리

⇒ 한계 : 서버는 1개의 요청 처리 함수를 사용하고 있음

⇒ 이미지,css등을 사용하지 못함

⇒ 작은 규모의 사이트에서만 사용가능

2장 : NPM과 Express

- npm으로 패키지 설치 → 다양한 개발자 패키지 존재

- Express

: Node.js 위에 만들어진 경량 웹 프레임 워크

: 장점 → node.js의 api를 단순화 , 미들웨어와 라우팅을 사용하여 응용을 구조화, http 객체에 기능 추가, 동적 html 작성 가능

⇒ 설치 해야함 (npm init → npm install express)

- Express로 웹 서버 만들기

⇒ Express 웹 서버

<코드>

```
const express=require('express')'
const app = express();
app.listen(3000,()⇒{
    console.log('express server running on port 3000');
});
```

⇒ 라우팅으로 요청 처리하기

```
app.get("/",(req,res)⇒ {
    res.json({
        name: 'park'
        age: '20'
    })
});
```

⇒ 서버 실행 : node index.js

⇒ 응답으로 파일 보내기

```
const path=require('path);
app.get("/about",(req,res)⇒{
    res.SendFile(path.resolve(__dirname,"./aboutpage.html));
});
```

→ path 모듈 : 파일 경로와 관련 (resolve : 절대 경로 생성)

→ __dirname : 현재 작업 폴더의 경로

- 정적 파일 서비스

: app.use(express.static('public')); ⇒ 디렉토리의 정적파일 제공!

→ express.static 패키지 : 정적 파일 서비스를 위한 패키지

→ app.use : 기능을 미들웨어 스택에 추가

3장: 블로그 응용 시작하기

- 블로그 응용

⇒ index.js 작성

```
const express=require('express');
const path=require('path");
app = express();
app.use(express.static('public"));
app.listen(3000,()⇒{
    console.log('express server running on port 3000');
});
```

- nodemon을 통한 서버 자동 재 실행

→ 서버를 바꾸면 서버 자동 재실행

⇒ nodemon 설치 : npm install —save-dev nodemon

⇒ package.json의 script의 start 작성

```
"scripts": {
  "start": "nodemon index.js",
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

⇒ 실행 : npm start

- 라우트를 통하여 파일 서비스 하기

⇒ index.js 수정

```
app.get("/", (req, res) =>{
    res.sendFile(path.resolve(__dirname,"pages/index.html"));
});
app.get("/about", (req, res) =>{
    res.sendFile(path.resolve(__dirname,"pages/about.html"));
});
app.get("/contact", (req, res) =>{
    res.sendFile(path.resolve(__dirname,"pages/contact.html"));
});
app.get("/post", (req, res) =>{
    res.sendFile(path.resolve(__dirname,"pages/post.html"));
});
```