<생활코딩 Node.js 요약>

기술설계팀 35기 장진희

1. 필요한 배경지식
2. 생활코딩 WEB1 수업 (HTML, CSS, JavaScript)
3. Java의 기본 개념

- 문법 : for, while, if 등

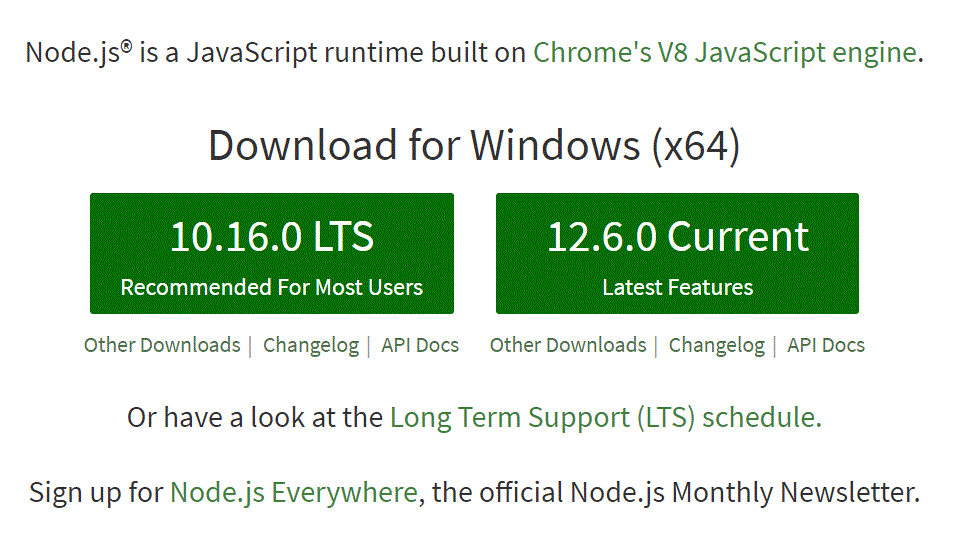
- 개념 : 변수형, 함수, 객체, 메소드, 배열 등

// 이 요약문에서는 설명을 생략함

1. Git 사용법

* cd 로 위치 이동하기
* Ctrl + C 누르면 종료

1. Node.js 공식 홈페이지에서 문서 -> 내 설치버전(10.16.0)에 들어가면 다양한 함수 설명서를 볼 수 있음
2. 설치법(Node.js runtime) – window 기준
3. <https://nodejs.org/en/> 사이트 접속
4. 10.16.0(최신버전) LTS 클릭



1. 전부 동의 누르고 설치
2. Cmd 창 열어서 확인

* node -v : 설치버전의 번호 확인가능
* node

>console.log(1+1);

: 결과가 2가 나오면 설치 성공

1. 사전 준비
2. 바탕화면에 nodejs 파일 생성
3. Atom으로 nodejs 파일 열기
4. <https://opentutorials.org/module/3549/21032> 사이트 접속하여 소스코드를 nodejs 파일로 복사
5. Git bash 켜서 nodejs의 파일 경로까지 이동(cd)하고 node main.js입력
6. local:3000 의 url 입력하여 서버가 정상적으로 동작하는지 확인
7. 기초 문법
8. 모든 변수형을 포괄하는 var을 주로 사용

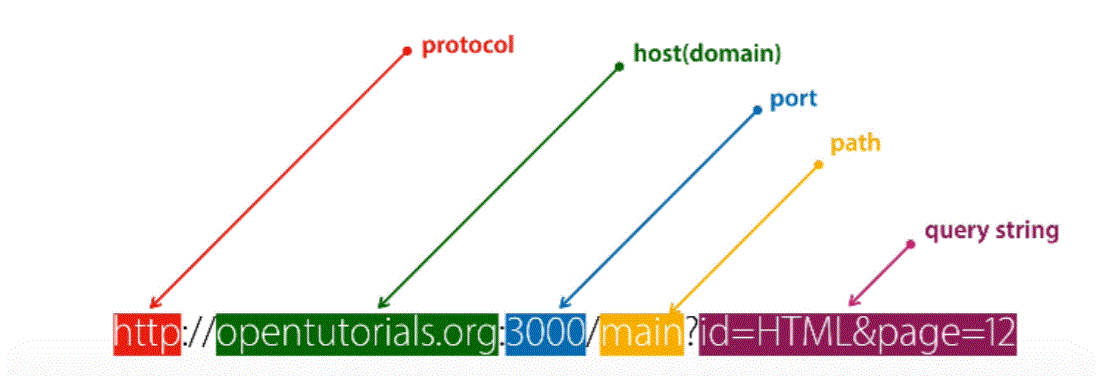
Ex) int a = 3; -> var a= 3;

str a = ‘A’; -> var a= ‘A’;

1. 문장 끝에 ; 로 마무리
2. Lorem을 치고 Tab을 누르면 의미 없는 긴 문장을 생성 가능
3. console.log(“내용”)으로 원하는 내용을 콘솔창에 출력할 수 있다.

(“내용”은 변경가능한 부분)

1. syntax폴더안의 sample.js를 실행시키고 싶으면 node syntax/sample.js를 입력
2. 같은지를 물어보는 비교 연산자 : ===
3. 템플릿 리터럴
4. 시작과 끝에 “” 대신 `` 를 사용
5. ${“내용”} 으로 변수 표현 (C언어에서 %d와 비슷한 역활)
6. URL



1. Protocol : 통신규칙
2. Host(domain) : 특정한 인터넷에 연결되어 있는 컴퓨터를 가리키는 주소
3. Port : 한 대의 컴퓨터 안에 여러 개의 서버들 중에 접속시킬 서버를 가리킴

(web server 의 port는 80)

1. Path : 어떤 디렉토리의 어떤 파일인지를 가리킴
2. Query string

* 물음표로 시작
* 값과 값 사이에 &를 사용
* 값과 값의 이름은 = 로 구분
* query data 구하기

var url = require(‘url’);

var \_url = request.url;

var queryData = url.parse(\_url, true).query;

console.log(queryData);

1. queryData.id에 따라 웹페이지 변경하기

var title = queryData.id;

타이틀 자리에 ${title} 넣기

<a href = “queryData의 내용 쓰기”>

1. CRUD
2. Create
3. Read
4. Update,
5. Delete
6. 파일 읽기

var fs = require(‘fs’);

fs.readFile(‘읽고싶은 문서’, ‘utf8’, function(err, data){

console.log(data);

})

* 실습 예제(HTML, CSS, JavaScript)를 data 폴더에 파일로 넣어서 불러오고 수정할 수 있도록 하기

1. 404 Not Found

var pathname = url.parse(\_url, true).pathname;

If (pathname === ‘/’){

var template = ~~~

response.writeHead(200);

response.end(template);

}

else {

response.writeHead(404);

response.end(‘Not found’);

}

//404 Not found는 파일을 찾을 수 없을 때의 예외처리

1. 홈페이지 구현

if(queryData.id===undefined){

var title = 'Welcome';

var description = 'Hello, Node.js';

~~~~~

}else{

~~~~

}

// 홈에 가면 queryData.id 가 undefined 로 뜨므로

1. 파일목록 알아내기

fs.readdir( ‘./data’, function(error, filelist){

console.log(filelist);

}

1. 글 목록 출력

var list = '<ul>';

var i = 0;

while(i < filelist.length){

list = list + `<li><a href = "/?id=${filelist[i]}">${filelist[i]}</a></li>`;

i = i + 1;

}

list = list + '</ul>';

//filelist 배열안에 들어있는 글 목록을 순차적으로 출력

1. Template을 함수로 바꾸기

전 : var template = `<!doctype html>

<html>

<head>

<title>WEB1 - ${title}</title>

<meta charset="utf-8">

</head>

<body>

<h1><a href="/">WEB</a></h1>

${list}

${control}

${body}

</body>

</html>`

후 : function templateHTML(title, list, body, control){

return `

<!doctype html>

<html>

<head>

<title>WEB1 - ${title}</title>

<meta charset="utf-8">

</head>

<body>

<h1><a href="/">WEB</a></h1>

${list}

${control}

${body}

</body>

</html>`

}

var template = templateHTML(title, list, `<h2>${title}</h2>${description}` ,`<a href = "/create">create</a>`);

// templateList도 같은 방식으로 함수로 변경

1. 동기/비동기(synchronous/asynchronous)

동기 : 순차적으로 주어진 일을 수행, 직렬적

비동기 : 일을 하면서 오래 걸리는 일을 실행시키고 다음일을 한 후 다시 이전 일로 돌아가며 수행, 병렬적

1. Callback

-작업이 끝난 뒤 함수를 호출하는 것을 가리킴

1. NPM(패키지 매니저) – PM2
2. 시작 : npm install pm2 -g
3. 프로그램 시행 : pm2 start main.js
4. 프로그램의 상태를 모니터링 : pm2 monit (Q를 눌러서 종료)
5. 현재 시행중인 프로그램 리스트 : pm2 list
6. 프로그램 종료 : pm2 stop main
7. 프로그램 수정 후 시행시 자동 반영 : pm2 start main.js --watch
8. 프로그램의 문제점 확인 : pm2 log
9. Start 와 log를 동시에 함 : pm2 start main.js -- watch --no-daemon
10. 파일 수정시 사이트가 종료되는 것을 방지 :
11. pm2 start main.js -- watch --ignore-watch=”data/\*” --no-daemon
12. PM2를 모두 종료 : pm2 kill
13. Form

<form action="http//localhost:3000/process create" method = "post"> //이 주소에 query string의 형태로 입력받은 정보를 전송

<p><input type = "text" name = "title"></p> //사용자가 입력1

<p>

<textarea name = "description"></textarea> //사용자가 입력2

</p>

<p>

<input type = "submit"> //입력 후 submit을 누름

</p>

</form>

1. Create 버튼 생성

`<a href = "/create">create</a>`

else if (pathname === '/create'){

fs.readdir('./data',function(error,filelist){

var title = 'WEB - create';

var list = templateList(filelist);

var template = templateHTML(title, list, //form을 활용

`<form action="http//localhost:3000/create\_process" method = "post"> <p><input type = "text" name = "title" placeholder = "title"></p>

<p>

<textarea name = "description" placeholder = "description"></textarea>

</p>

<p>

<input type = "submit">

</p>

</form>

`,'');

response.writeHead(200);

response.end(template);

});

}

1. Create 기능 완성(post 방식으로 데이터를 전달)

else if(pathname === '/http//localhost:3000/create\_process'){

var body = '';

request.on('data', function (data) { //정보를 수신할 때마다 callback 함수를 수행

body += data;

});

request.on('end', function () {

var post = qs.parse(body);

var title = post.title;

var description = post.description; //파일 생성

fs.writeFile( `data/${title}`, description, 'utf8', function(err){

response.writeHead(302, {Location : `/?id=${title}`});

response.end();

})

});

1. Update 버튼 생성

<a href = "/update?id=${title}">update</a>

else if (pathname === '/update'){

fs.readdir('./data',function(error,filelist){

fs.readFile(`data/${queryData.id}`,'utf8', function(err,description){

var title = queryData.id;

var list = template.list(filelist);

var html = template.HTML(title, list,

`<form action="http//localhost:3000/update\_process" method = "post">

<input type = "hidden" name = "id" value ="${title}" > //기존 내용 전송을 위해 저장

<p><input type = "text" name = "title" placeholder = "title" value = "${title}"></p>

<p> //기존 내용 value에 넣기

<textarea name = "description" placeholder = "description">${description}</textarea> //기존 내용 placeholder에 넣기

</p>

<p>

<input type = "submit">

</p>

</form>

`

,`<a href = "/create">create</a> <a href = "/update?id=${title}">update</a>`);

response.writeHead(200);

response.end(html);

});

});}

1. update 기능 완성

else if (pathname === '/http//localhost:3000/update\_process'){

var body = '';

request.on('data', function (data) {

body += data;

});

request.on('end', function () {

var post = qs.parse(body);

var id = post.id; //create\_process와의 차이점

var title = post.title;

var description = post.description;

fs.rename(`data/${id}`, `data/${title}`, function(error){ //파일 내용 수정

fs.writeFile( `data/${title}`, description, 'utf8', function(err){

response.writeHead(302, {Location : `/?id=${title}`});

response.end();

})

})

});

}

1. delete 버튼 생성

<form action = "delete\_process" method = "post">

<input type = "hidden" name = "id" value = "${title}">

<input type = "submit" value = "delete">

</form>

// delete 버튼을 링크 형태로 만들면 내용이 삭제되지 못하고 해킹당할 위험이 높음

1. delete 기능 완성

else if (pathname === '/delete\_process'){

var body = '';

request.on('data', function (data) {

body += data;

});

request.on('end', function () {

var post = qs.parse(body);

var id = post.id;

fs.unlink(`data/${id}`, function(error){ //삭제

response.writeHead(302, {Location : `/`}); //삭제 후 홈페이지로 이동

response.end();

})

});

}

1. template 객체 만들기

var template = {

HTML : function(title, list, body, control){

~~~~~~~

},

list : function (filelist){

~~~~~~~

}

}

var list = template.list(filelist);

var html = template.HTML(title, list, `<h2>${title}</h2>${description}`, `<a href = "/create">create</a>`);

1. 모듈(module)

<mpart.js>

var M = {

v : 'v',

f : function(){

console.log(this.v);

}

}

module.exports = M; //모듈 M을 밖에서도 이용할 수 있음

var part = require('./mpart.js'); //모듈을 가져옴

part.f();

1. Template 모듈 만들기

<template.js>

module.exports = {

HTML : function(title, list, body, control){

~~~~~~~~~

},

list : function (filelist){

~~~~~~~~~~

}

}

1. 보안
2. 입력정보에 대한 보안 : queryData에 ..을 입력하여 상위 파일이 해킹 당할 수 있음

, 오염된 정보가 들어올 수 있음

1. 출력정보에 대한 보안 : JavaScript의 형식으로 입력이 들어오면 사이트의 내용이 변경되어 엉뚱한 사이트나 내용으로 우회될 수 있음 - <script>를 삭제
2. API

- Application Programming Interface

Ex) Node.js 가 내장하고 있는 함수, 라이브러리 등