1. 노드 시작하기

# REPL : Read, Evaluate, Print, Loop의 약어로 콘솔과 같은 역할을 한다.

# 호출 스택과 이벤트 루프 cf) 이벤트 루프 자세한 설명은 교재 p.53 참조

함수의 호출은 스택에 쌓이고 이벤트 루프는 태스크 큐를 관리한다.

이벤트 루프는 스택에 아무것도 없을 때 태스크 큐에 있는 작업 호출 스택으로 불러 실행한다. 태스크 큐에는 여러 레벨의 큐가 있고 높은 레벨 순으로 이벤트 루프에 의해 스택으로 옮겨져 실행된다.

ex) setTimeout(run, 3000); 을 실행하면 setTimeout 함수는 스택에 들어갔다가 실행되면서 run은태스크 큐로, 타이머는 백그라운드로 보낸다. 백그라운드에서는 3초를 샌 후 이벤트 루프에 의해 태스크 큐에서 run 함수를 조건에 맞게(3초 뒤에) 호출 스택으로 꺼내온다.

# 이벤트기반, 싱글쓰레드, 논블러킹IO

실제 서버는 프로그램이 종료되지 않게 해주는 코드가 필요

서버의 원칙 : 요청이 오면 응답 한다.

요청을 처리하는 것 -> 이벤트 리스너 ex) 방문, 댓글 저장

* 이벤트 기반으로 서버 작동

논블로킹 IO

태스크 큐로 보내서 코드를 효율적인 순서로 실행하는 것

싱글 스레드 (팔이 한 개라고 생각할 수 있음)

* 그렇기 때문에 블로킹이 발생하여 느려지는 것을 방지하고자 논블로킹 IO를 이용하여 순서를 효율적으로 바꿔가며 실행함

# const와 let

ES2015(ES6의 정식명칭, javascript의 정식 문법 레퍼런스)으로 기술되었다. 알아보자

const와 let은 {…} 중괄호 블록을 빠져나갈 수 없다. 중괄호 블록내부에서만 접근할 수 있다.

const와 let의 특성

const는 값 변경이 불가능하다. -> 상수 선언에 사용

let은 값 변경이 가능하다. -> 변수 선언에 사용

const 자체는 변경 불가능 하지만, const에 객체가 할당된 경우 객체 내부 속성은 변경가능

ex) const g={a:1, b:2, c:3}; g.a=3; g.b=5; -> g = {a:3, b:5, c:3}으로 바뀜

# 템플릿 문자열(백틱, `)

문자열을 합칠 때 `으로 감싸줄 수 있다. 변수는 ${변수}로 표시한다

ex) const d=`${a} ${b} ${c}`; // 지저분하게 +로 더해줄 필요가 없음

백틱을 이용하여 문자열 내에 작은 따옴표, 큰 따옴표를 자유롭게 표현 가능하다.

# 객체 리터럴

리터럴로 객체 선언하는 방법에 기능이 추가됨

var foo = { a:’apple’, b:’banana }; 와 같이 리터럴 선언 가능

key:value 가 같은 경우 한 번만 써도 됨

메소드도 key : function(){ } 이 아닌, 메소드이름(){ } 이렇게 선언 가능

동적 속성 할당을 리터럴 안에 표현 가능함 -> { [변수] : 값 } 과 같은 형태로 가능

cf) 동적 속성이란? 변수 이름을 이용하여 속성을 추가할 때는 객체 리터럴에 한 번에 선언할 수가 없고, foo[a + 6] = “six apples” 와 같이 따로 추가해줬어야 함

# 화살표 함수

function(매개){ return 무엇 } 과 같이 선언하던 함수를 화살표를 이용하여 선언할 수 있음.

(매개) => { return 무엇} 과 같이 선언 가능하다.

매개변수에 별다른 가공이 없다면 중괄호를 제거하고 (중괄호가 없으면 return ..과 같다)

(매개) => (x+y); 이런 식으로 대체 가능하다.

화살표 함수가 간단하지만 function의 역할을 대체할 수 없다.

화살표 함수 내부의 this 와 function 내부의 this의 의미가 다르다.

this는 원래 함수의 바로 바깥쪽에 있는 객체를 가리킨다.

* 화살표 함수 내부의 this를 사용하면 바로 바깥쪽의 this가 가리키는 값(즉 바깥의 바깥을 가리킴!)과 같게 만들어준다

# 비구조화(destructure)

const a = 객체.a

const b = 객체.b 를

const {a ,b} = 객체 로 변경가능하다.

***비구조화 할당해서 객체와 메소드를 분리한 경우에는 this가 의도와 다르게 동작할 수 있다.***

배열에도 적용 가능하다.

var array = [‘nodejs’, {}, 10, true]; 일 때,

var node = array[0]

var obj = array[1]

var bool = array[array.length-1];

은 다음과 같다.

const [node, obj, , bool] = array; // 인덱스에 따라서 받아오지 않을 인덱스는 공백으로 비워주는 모습

# rest문법과 Q&A

비구조화 할당에서 1:1로 대응이 아니라 마지막 객체에 전부 할당하고 싶으면

const array = [‘nodejs’, {}, 10, true];

const [node, obj, … bool] = array;

이런 식으로 할당하면 bool 객체는 array 객체가 되어

bool = [10, true]; 이렇게 할당 되는 것과 같게 된다.

ex) const n = (x, …y) => console.log(x,y)

n(5, 6, 7, 8, 9) // (4) [6,7,8,9] 가 출력됨

…변수 -> 이건 나머지를 전부 저 변수에 넣는다는 의미

arguments로 복수 개의 요소를 표현 했으나 이제는

const p = (…rest) => console.log(rest)

p(5,6,7,8,9) // (출력) [5,6,7,8,9]

* arguments는 배열은 아니고 유사배열이라 배열의 메소드를 쓸 수 없지만 rest문법은 배열을 반환하기 때문에 편하다.

const x = { a : 1, b : 2} 라고 선언하면

x는 메모리에 있는 객체의 주소를 가리키는 것이므로 가리키고 있는 메모리 주소는 바꿀 수 없지만, a와 b의 값은 바꿀 수 있다.

# 콜백과 프로미스(Promise) 비교, async / await

콜백 -> 객체에게 명령을 실행시키고 그 일을 기다리지 않고 나를 다시 부를 때까지 다른 명령을 수행하는 것 // 콜백을 사용하는 이유 : 코드가 논블로킹 방식으로 작동하기 때문에!

ex) Users.findOne(‘zero’, (err, use) => {

if(err) {

return console.error(err);

}

console.log(user);

} ) // ‘zero’ 찾으면 다시 이벤트 루프로 들어오는 콜백 함수

과거에는 call-back hell을 막기 위해 변수로 콜백을 빼서 사용함

* 직관성, 코드 가독성 떨어짐
* 해결하기 위해 프로미스 기법이 나옴!

프로미스 작성법

(콘솔에 쳤을 때 첫 글자가 대문자로 나오면 생성자라는 뜻임)

프로미스 사용 예시 ex) const plus = new Promise( (resolve, reject) => {

const a = 1; const b=2;

if(a + b > 2) {

resolve(a+b);

} else {

reject(a+b);

}

});

plus

.then( (success) => { console.log(success); } )

.catch( (fail) => { console.error(fail); } )

resolve 되면 then으로 // reject 되면 catch로 넘어감

* resolve는 프로미스의 성공을 의미, reject는 프로미스의 실패를 의미

resolve의 괄호 안에 들어있는 값은 then의 매개변수인 success로, reject의 괄호 안에 들어있는 값은 catch의 매개변수인 fail로 넘겨줌

프로미스 타입을 리턴해야 then, catch를 사용할 수 있다.

* 프로미스 타입을 return하는 메소드들만 then, catch를 사용할 수 있다.
* promise에서 promise를 또 return하면 아래에 then~catch문으로 이어짐
* promise중첩일 때 한번이라도 실패하면 catch로 감

const promise = new Promise( (res, rej) => {

res(‘성공’);

} ); 와 같이 항상 성공(resolve)하는 promise는

const sucPromise = Promise.resolve(‘성공’); 과 같이 줄일 수 있다.

Promise.all( [ .. , .. , .. ]

.then( (results) => {} )

.catch( (error) => {} )

Promise.all로 여러 프로미스를 동시에 실행 가능하다, 단 하나라도 실패하면 전부 catch로!

Promise가 무엇이냐?

* Promise는 결과값을 가지고 있지만 .then 혹은 .catch를 붙이기 전까지는 반환하지 않음
* 콜백은 결과를 사용하는 부분이 바로 이어져야 해서 융통성이 적어짐
* Promise는 매개변수도 자유롭게 변경이 가능해서 객체지향 적인, 효율적인 코딩가능

가독성이 높아진 건 맞지만 코드가 순서대로 진행되지 않는 다는 건 같음

그래서 만들어진 yield라는 게 있음 (제너레이터와 비슷)

ES2017에서 정식으로 등록됨

async / await를 사용해 프로미스를 순서대로 실행 (눈속임이긴 함) 할 수 있음

async / await 동영상 참조

async를 함수 앞에 붙여야만 await를 쓸 수 있다.

에러가 나지 않기 때문에 try~ catch 문으로 다 나눠줘야 한다는 불편함이 있음

# 동기와 비동기 / 블로킹과 논블로킹

동기와 비동기는 함수가 바로 return 되느지의 여부

블로킹과 논블로킹은 백그라운드 작업 완료 여부

동기 블로킹 : 호출한 함수가 바로 return되지 않고 백그라운드 작업이 끝나야 return됨

비동기 논블로킹 : 호출한 함수가 바로 return되고 나중에 백그라운드가 알림을 줄 때 처리

export 메소드는 module.export에 객체를 복사 붙여넣기 하는 것과 같다.

require는 그 파일의 export에 있는 object를 가져오는 것!

# 서버 생성

http모듈을 require한 후 CreateServer메소드를 이용, 인자로 익명의 콜백 함수를 구현하여 req, res를 이용하여 클라이언트의 요청과 서버의 응답을 처리한다.

Server의 listen메소드를 사용, listen은 포트번호와 콜백함수를 인자로 받는다. 콜백함수를 익명으로 구현하여 console에 서버가 실행됨을 표시하는 로그를 찍는다.

# html파일 불러오기

fs(파일시스템) 모듈을 require한 후 readFile 메소드를 사용하여 파일을 불러온다.

readFile 메소드도 2번째 인자로 콜백함수를 받는다. 익명의 콜백함수를 구현하여 html로부터 받

# 클라이언트 구분 (로그인)

서버가 클라이언트를 구분하여 data를 보내주기 위하여 쿠키, 세션 개념이 필요하다.

쿠키는 단순한 ‘키-값’의 쌍이다.

클라이언트의 웹 브라우저는 서버로부터 쿠키가 오면 쿠키를 저장해두었다가 요청할 때마다 쿠키를 동봉하여 보내준다.

쿠키는 res / req의 각 헤더에 저장된다.

세션은 접속중인 웹 서버에 사용자 data를 저장하는 방식이다.

크롬 개발자 도구의 Network 탭을 사용하여 요청과 응답에 대해 상세하게 살펴볼 수 있다.

# http

200 : 성공

302 : 다른 url로 보내줌

# REST API

서버의 자원들(회원 정보, 게시글, 댓글)을 주소를 사용하여 저장, 구조화 하는 방식

GET : 서버의 자원을 가져옴

POST : 서버에 자원을 새로 등록

PUT : 서버의 자원을 요청에 보내는 데이터로 수정하고자 할 때 사용

PATCH : 서버 자원의 일부만 수정하고자 할 때 사용

DELETE : 서버의 자원을 삭제하고자 할 때 사용

REST API의 장점 : 주소와 메소드만 보고 내용을 명확하게 알아볼 수 있다.

https모듈은 웹 서버에 SSL 암호화를 추가한다. GET / POST 등의 요청을 할 때 오고 가는 데이터를 암호화해서 중간에 다른 사람이 요청을 가로채더라도 내용을 확인할 수 없도록 한다.

요즘은 로그인 / 결제가 필요한 창에서 https적용이 필수가 되어가고 있다.

http2 모듈은 SSL 암호화와 더불어 최신 HTTP 프로토콜인 http/2를 사용하게 해준다. 기존의 http/1.1보다 개선되어 웹의 속도가 많이 개선된다.

# CRUD // 생활코딩 12. Node.js 파일 읽기

Create / Read / Update / Delete

main.js가 바뀌면 재실행을 해야 적용되지만 페이지가 열릴때마다 정적파일을 읽어오고 형식이라면 정적파일을 바꾸고 수정하면 서버를 재실행하지 않아도 적용할 수 있다.

fs모듈의 readFile함수를 이용하여 파일을 읽어온다.

* fs.readFile(파일이름, (형식 ex.utf-8), 콜백함수(err, data){…}와 같은 방식으로 사용 가능

# 콘솔에서의 입력값 (생활코딩)

process.args 함수(?)를 통해 콘솔로부터 입력을 받을 수 있다.

var args = process.args로 입력을 받고 출력을 한다면

args[0] : node runtime이 실행된 경로

args[1] : js파일이 실행된 경로

args[2] : 사용자가 첫번째로 입력한 값, …

이런 방식으로 저장이 된다.

# App – Not found 구현 // 생활코딩

path는 ‘/’으로 구분을 한다.

‘/’ 뒤에 ‘?’ 를 사용하여 붙는 요소는 querystring이므로 pathname이 아님!

* ‘/’ 뒤에 querystring이 붙을 경우의 pathname은 ‘/’이다.
* path와 href는 ‘/?id=CSS’와 같은 방식으로 바뀜

# HTML – FORM // 생활코딩

HTML의 form 기능을 이용하여 form 태그에서 설정해주는 서버로 querystring을 이용해 정보를 전송해줄 수 있다.

* querystring을 이용하여 server에 요청할 data를 구분하는 건 좋은 방법!

but, 요청받은 data나 수정, 삭제할 data를 전부 url에 표시하는 건 좋지 않다.

* why? url에 접속을 하면 url에 쓰여진 정보에 따라 기능이 수행되기 때문에 정보의 수정, 삭제와 관련된 것을 url에 쓰는 건 좋지 않은 방법!

방법 //

<form action=”보낼 서버의 주소” **method=”POST”**>

저렇게 method를 POST로 지정해주면 querystring(querystring을 이용하지 않으므로 당연히 url에도 표시가 안됨)을 이용하지 않고 data를 보낼 수 있다. url을 길이가 유한한데, url을 이용하지 않으므로 큰 data도 전송할 수 있다.

method를 쓰지 않으면 default 값으로 get이 들어간다.//

// 서버로부터 data를 받아오는 것 -> GET

// 서버로 data를 보내는 것 -> POST

# POST 방식으로 전송된 data 받기 // 생활코딩

requset.on 메소드를 사용한다.

requset.on(‘data’,…) 메소드는 스트림을 통해 여러 chunk로 전송되는 data가 들어올 때마다 호출되는 method이다.

request.on(‘data’, function(data){

body += data;

}); 과 같이 사용된다. data가 전부 전송되면

request.on(‘end’, …) 메소드가 호출된다.

* request.on(‘end’, ..) 메소드에서 querystring 모듈의 parse 메소드를 통해서 body에 담긴 data를 객체화 한다!! json형식으로 만들어줌!

# App - 파일생성과 리다이렉션 // 생활코딩

fs.writeFile 메소드를 이용한다.

ex) fs.writeFile(`data/${title}`, data, ‘utf8’, function(err){

});

리다이렉션 : 사용자가 페이지 사용을 완료하면 다른 페이지로 보내는 것

* 302 header를 보낸다 : 이 주소는 영구적으로 바뀌었다는 말! // 일단 이거씀
* response.writeHead(302, {Location : `/?id=${title}`});
* 를 이용해서 리다이렉션을 해준다

# App 제작 - 글수정 / 수정할 정보 전송

id가 undefined를 제외한 모든 경우에 update버튼을 만들어준다.

pathname이 /update 일 때, 즉, 사용자가 update를 눌렀을 때,

사용자에게 보여주는 form태그를 변경하고 파일에서 읽어온 값들을 기본값으로 보여주기 위해서

value 프로퍼티와 textarea의 태그 사이에 백틱문자열로 ${description}을 이용하여 파일에서 읽어온 값들을 보여준다.

그리고 수정할 정보를 전송하기 위해서 title을 이용할 수 있는데 title을 사용자가 바꾸고 update를 하면 파일을 찾지 못하므로, input type=”hidden” name=”id” // name=”title”을 사용하여 id와 title을 구분하고 title은 바뀌어도 id는 불변값으로 하여 파일을 찾을 수 있도록 한다.

# App - 글수정 / 수정된 정보 저장

hidden 태그를 사용하여 이전 제목을 보존한 다음 바뀐 제목으로 바꾼다.

fs.rename(`data/${id}`, `data/${title}`, function(err){ … }); 을 이용하여 파일의 이름을 바꿔주고 qs.parse 메소드로 json형식으로 객체화한 post변수로 내용도 변경하여 저장한다.

# App - 삭제버튼 구현

querystring을 이용하는 방식은 GET방식이다 // 서버에 뭔가를 요구하는 것

그러나 삭제기능은 서버에 변경된 결과를 알려주는 것이므로 POST방식으로 구현해야한다.

* <a href> 를 이용하여 링크로 구현하면 안됨!
* form태그를 이용하여 POST방식으로 구현해야한다.
* form 태그에서 method 변수를 post로 바꿔주고 delete를 누르면 delete\_process url로 이동하도록 한다.

# object의 반복문

var roles = {

‘programmer’ : ‘egoing’,

‘designer’ : ‘k808’,

‘manager’ : ‘hoya’ }

for(var name **in** roles)

in 키워드를 사용하여 object를 반복할 수 있다.

* name에는 value가 아닌 key값이 들어가므로 key 값을 이용해서 roles.key 혹은 roles[‘key’]와 같은 방법으로 접근하면 된다.