

KWEB Study Week2: JS with Node.js

KWEB 2학기 준회원 스터디



Today's Contents

1. Before Study
2. Object & Function
3. Asynchronous
4. Clojure
5. Class in JS
6. 과제

M.G. BAE

J.H. BAEK



Before Study

- 지난 2주차에는 npm, Node.js의 Module과 ejs Module에 관해 배웠습니다.
- 우리가 2학기에 다루는 Node.js는 Javascript 기반의 플랫폼이기 때문에 Javascript에 대한 지식이 있어야합니다.
- 2주차의 목표는 지금까지 배웠던 Javascript를 갈고 닦는 것입니다. (물론 새로운 내용도 등장합니다!) 그리고 비동기의 개념을 똑바로 익혀가도록 합시다.
- 지난 주차 복습과 함께 시작할게요!



Previous Study: npm

- npm
 - NodeJS, Javascript 의 패키지 매니저 + 오픈소스 생태계
 - 주소 URL: <https://www.npmjs.com/>
- yarn
 - npm의 몇가지 문제를 개선하기 위해 나온 npm 기반의 패키지 매니저
 - 보안영역 강화
 - 캐시, 멀티 다운로드를 통한 보다 빠른 패키지 관리
 - Yarn 전용 lock 파일을 통한 프로젝트 버전 관리의 용이성
 - 프로덕션 에서는 안정성을 위해 yarn의 사용을 선호함



Previous Study: npm

- npm init
 - 로컬 프로젝트에서 npm을 통해 설치하거나 사용할 기본 정보들을 기록하는 package.json 파일의 생성을 도와줌
- npm install
 - 패키지 설치
 - node_modules 폴더에 패키지 관련 파일들을 설치해 줌
 - 기본적으로 node는 require 함수 사용시에 별도의 위치를 지정하는 문자(EX "./")를 사용하지 않으면 node_modules 폴더에서 패키지를 찾아서 require 해줌
- npm uninstall
 - 패키지 삭제
 - node_modules에 설치된 패키지를 지워줌
- npm --save (npm 명령어 옵션 중 하나이다.)
 - Install, uninstall 등의 작업후에 package.json에 작업 내용을 기록함



Previous Study: Module

- 큰 프로그램을 만들다보면 여러 곳에서 빈번하게 사용되는 코드문치가 있기 마련입니다.
- 반복되는 내용을 각각 코드에 모두 작성하게 되면 수정/유지보수시에 헬게이트를 보게 될 것입니다. → (서비스 비즈니스 로직(가격계산, 통계 등)을 처리하는 코드가 여러 곳에 퍼져 있소 요구사항에 변경 되었을 때 등등)
- 따라서 반복되는 코드는 한곳에 모아두고 사용하는 곳에서 가져다 쓰는 것이 바람직합니다.
- Node는 Module이라는 기능을 통해 위의 과정을 쉽게 만들어 줍니다.



Previous Study: ejs (module)

- ejs (Embedded JavaScript)는 자바스크립트의 대표적인 템플릿 엔진입니다.
 - 주소 URL: <http://www.embeddedjs.com/>
- 템플릿 + 데이터 통해 html을 생성하는 역할을 함
- 주로 Express 패키지와 함께 MVC 패턴을 통한 서버 웹 프로그래밍에 사용됨 (MVC 패턴은 이후 Express.js와 함께 배울 예정)
- 비슷한 역할을 하는 패키지로 pug가 있음(구 Jade) → pug도 역시 템플릿 엔진입니다!



Object

- Javascript는 객체(Object) 기반의 스크립트 프로그래밍 언어이다.
- 우리가 살아가는 세상은 많은 객체 들이 상호 작용하며 돌아가며, 객체 지향은 이러한 개념을 프로그래밍에 가져온 것이다.
- 객체지향 언어에서 객체는 데이터(주체)와 그 데이터에 관련되는 동작(절차, 방법, 기능)을 모두 포함하고 있는 개념이다.
- 모든 객체는 자신의 속성값(**Property**)과 행동을 정의 하는 메소드(**Method**)를 가진다.
 - 예를 들어 JS에서 문자열 변수 “TEXT” 는 내용인 ‘text’를 속성값으로 문자열 처리에 관련된 함수를 메소드로 가진다.



Object

- 객체라는 껍데기 속에서 실제 객체를 완성하는 구성요소들은 앞 슬라이드에서 말한 프로퍼티와 메소드이다.
- 프로퍼티: 객체의 속성을 나타내는 접근 가능한 이름과 활용 가능한 값을 가지는 특별한 형태
- 메소드: 객체가 가지고 있는 동작
- 오른쪽은 Class를 제외하고 원래 Javascript에서 객체를 정의하는 예제입니다.

```
//방법 1
var someone={
  name: "Team",
  height: 170,
  weight: 65,
  eat: function(){this.weight+=1;},
  diet: function(){this.weight-=1;}
}

//방법 2
function Person(name, height, weight){
  this.name=name,
  this.height=height,
  this.weight=weight,
  this.eat=function(){this.weight+=1;},
  this.diet=function(){this.weight-=1;}
}

var Team=new Person("Team",170,65);
Team.height=173;
Team.eat();
```




Function

- Javascript에서 함수는 작업을 수행하고 값을 반환할 수도 있습니다.
- Javascript에서는 여러 방식의 함수 선언이 가능합니다.
 - 입력을 받고 출력을 주는 함수
 - 입력을 받고 출력을 주지 않는 함수
 - 입력을 받지않고 출력을 주는 함수
 - 입력을 받지않고 출력을 주지 않는 함수
- 함수는 모든 객체에서 사용 가능하므로 함수를 "전역 메서드"라고도 합니다.

```
1 // function take input and return output
2 function func1(a){
3     return a + 1;
4 }
5
6 // function take input and don't return output
7 function func1(a){
8     console.log(a);
9 }
10
11 // function no take input and return output
12 function func1(){
13     return 1;
14 }
15
16 // function take no input and don't return output
17 function func1(){
18     console.log(1);
19 }
```



First Class Object

- First Class Object는 Object를 First Class Citizen로써 취급하는 것이다. → 그냥 1급 시민인 객체이다. 조건은 1급 시민과 동일하다.
- First Class Citizen: 아래 3가지 조건을 만족하면 First-Class이다.
 - 변수에 담을 수 있다. (stored in a variable)
 - 인자(parameter)로써 전달할 수 있다. (passed as an argument)
 - 반환값으로 전달할 수 있다. (returned from other functions)
- 한국어 책에는 1급 객체라고 쓰여 있습니다. (First-Class는 프언 시간에 배웁니다!)



First Class Object

- Javascript의 함수는 1급 객체입니다.
- Javascript에서는 함수를 변수와 인자로써 답을 수 있다
 - 정확히는 함수는 변수와 같은 하나의 객체로 인식된다. 함수를 하나의 객체로 인식하기 때문에 이를 변수(객체로) 저장 할 수 있다.
- Javascript의 함수는 또한 1급 함수이기도 합니다. → 이걸 따로 찾아봅시다!

```
1  function func1(a){
2      return a + 1;
3  }
4
5  func1(1);
6
7  // OUTPUT
8  // 2
9
10 var func1_as_var = func1;
11
12 func1_as_var(1);
13
14 // OUTPUT
15 // 2
```



Asynchronous 이해하기

- 다들 윈도우나 안드로이드 운영체제에서 어플리케이션을 사용하다가 ‘응답없음’ 메시지가 뜨면서 갑자기 화면상의 UI가 작동하지 않는 것을 경험해보았죠..?
- 이는 프로그램의 흐름이 UI를 처리하는 로직 밖에서 오랜 시간 실행되어 UI 처리 로직이 정상적인 작동을 하지 못해 일어나는 겁니다. → (말이 어렵죠? 그냥 프로그램이 해야 할 걸 못해서 에러난 겁니다.)
- 파일 읽기, 네트워크 통신(API 사용, DB 처리 등), 입출력(I/O)을 실행하면 대부분의 프로그램은 흐름에 지연이 발생합니다. (OS 시간에 자세히 배웁니다.)
- 일괄작업(순차적으로 처리)을 이용하는 프로그램이라면 지연이 큰 문제가 없으나, 웹브라우저와 같이 사용자와 상호작용하는 UI가 있는 환경에서 지연은 주요한 문제가 됩니다.



Asynchronous 이해하기

- 일상에서 우리가 커피를 주문할 때를 생각해봅시다.
- 커피를 주문 할때 우리는 1. 어떤 커피를 주문할지 정하고, 2. 카운터에 가서 커피를 주문하고 3. 아래 2가지 중 하나의 일을 할 수 있습니다.
- 카운터 앞에서 커피가 나오기를 기다리거나
- 진동벨을 받아가서 다른 일을 하다가 커피를 받아가거나



Asynchronous 이해하기

- 일상에서 우리가 커피를 주문할 때를 생각해봅시다.
- 커피를 주문 할때 우리는 1. 어떤 커피를 주문할지 정하고, 2. 카운터에 가서 커피를 주문하고 3. 아래 2가지 중 하나의 일을 할 수 있습니다.
- 카운터 앞에서 커피가 나오기를 기다리거나 → 동기 (Synchronous)
 - 진동벨을 주고 받지 않아도 되지만 커피가 나올 때까지 카운터 앞에서 꼭 대기해야 합니다. (카운터 앞에서 대기하면 다른 업무는 못 보겠죠?)
- 진동벨을 받아가서 다른 일을 하다가 커피를 받아가거나 → 비동기 (Asynchronous)
 - 커피가 준비되는 동안 카운터 앞에서 대기할 필요없이 다른 일을 볼 수 있다.



Asynchronous 이해하기

- Chrome에서 F12나 Cmd + Alt + I 를 눌러 개발자 도구를 열고, Console 탭에서 “while(1){}”을 입력해봅시다.
- while문을 실행하고 나면 아마 브라우저가 응답하지 않을 거예요!
- 위의 결과는 자바스크립트에 끊임없이 돌아가는 무한 루프가 브라우저의 렌더링을 담당하는 코드로 실행 흐름을 넘겨주지 않아서 일어나는 현상입니다.
- 이와 같은 현상이 일어나지 않도록 하려고 등장한 개념이 “비동기” 이다.



Callback Function

- 비동기 방식으로 만든 함수는 연산이 끝났을 때 파라미터로 함수를 전달합니다.
- 이때, 파라미터로 전달되는 함수를 콜백 함수라고 합니다.
- **Callback function:** 객체의 상태 변화(이벤트)가 발생한 경우에 전달되는 함수
- 반환되는 함수안에도 새로운 함수가 있을 수 있음!

```
function sum_and_mod7(a, b, callback) {  
    var result = (a+b) % 7;  
    callback(result);  
}  
  
sum_and_mod7 (17,191, function(result) {  
    console.log('Processing sum_and_mod7\n');  
    console.log('( %d + %d ) % 7 = %d', a, b, result);  
});
```




실습 - fs 모듈 비동기 읽기

- 파일을 읽을 때 우리는 1. 어떤 파일을 읽을지 정하고 2. fs 모듈의 함수에게 파일을 읽을 것을 요청하고. 3. 아래와 같은 동작을 선택할 수 있다.
- 파일을 읽어 내용을 돌려줄 때까지 기다린다. → `readFileSync` (동기)
- 파일읽기가 끝나면 수행할 동작을 넘기고 다른 일을 계속한다. → `readFile` (비동기)

```
1 const http = require('http');
2 const path = require('path');
3 const fs = require('fs');
4
5 const hostname = '127.0.0.1';
6 const port = 3000;
7
8 const server = http.createServer((req, res) => {
9
10     fs.readFile(
11         path.join(__dirname, 'data.txt'),
12         'UTF-8',
13         function(err, content){
14             res.statusCode = 200;
15             res.setHeader('Content-Type', 'text/html');
16             res.end();
17         });
18 });
19
20 server.listen(port, hostname, () => {});
```



실습 - 동기 방식과 비교

- 한 번 동기 방식으로 파일 읽는 것과 비교해봅시다.

```
1  const http = require('http');
2  const path = require('path');
3  const fs = require('fs');
4
5  const hostname = '127.0.0.1';
6  const port = 3000;
7
8  const server = http.createServer((req, res) => {
9
10     var content = fs.readFileSync(path.join(__dirname, 'data.txt'), 'UTF-8');
11
12     res.statusCode = 200;
13     res.setHeader('Content-Type', 'text/html');
14     res.end();
15 });
16
17 server.listen(port, hostname, () => {});
18
```



생각해보기

- 근데 한 번 생각해보면 뭔가 이상합니다.
- fs 모듈의 readFile 함수에 data.txt 파일을 읽고 읽기가 완료되었을 때 수행할 함수를 넘겨준 것은 OK
- 그런데 분명히 콜백의 실행은 fs 모듈안일텐데 그 속에는 createServer의 콜백함수의 res 변수가 정의 되어 있지 않지 않나요..?
- 어떻게 readFile 함수의 콜백이 res 변수를 사용할 수 있는지..?

```
1  const http = require('http');
2  const path = require('path');
3  const fs = require('fs');
4
5  const hostname = '127.0.0.1';
6  const port = 3000;
7
8  const server = http.createServer((req, res) => {
9
10     fs.readFile(
11         path.join(__dirname, 'data.txt'),
12         'UTF-8',
13         function(err, content){
14             res.statusCode = 200;
15             res.setHeader('Content-Type', 'text/html');
16             res.end();
17         });
18 });
19
20 server.listen(port, hostname, () => {});
```



Clojure

- 방금 전 슬라이드와 같은 의문점은 Clojure라는 개념이 해결해줍니다.
- 1급 객체 함수의 개념을 이용하여 스코프(scope)에 묶인 변수를 바인딩 하기 위한 일종의 기술
- 기능상으로, 클로저는 함수를 저장한 레코드(record)이며, 스코프(scope)의 인수(factor)들은 클로저가 만들어질 때 정의(define)되며, 스코프 내의 영역이 소멸(remove)되었어도 그에 대한 접근(access)은 독립된 복사본인 클로저를 통해 이루어질 수 있다.
- 아래는 책과 주요 사이트에서 써놓은 클로저의 개념입니다.
 - 함수 객체와, 함수의 변수가 해석되는 유효범위
 - 독립적인 변수를 참조하는 함수들이다.
 - 내부함수가 외부함수의 맥락(context)에 접근할 수 있는 것
 - 함수와 함수가 선언된 어휘적 환경의 조합이다



실습 - Clojure

```
1 function make_closure(){
2     var a = 1;
3
4     function closure(){
5         console.log(a);
6     }
7
8     return closure;
9 }
10
11 cl = make_closure();
12
13 cl()
14 // 1
15
16 a
17 // VM332:16 Uncaught ReferenceError: a is not defined
```

- make_closure 함수의 지역변수인 a는 함수가 종료되면 소멸되는 것이 보통입니다.
- 하지만 make_closure가 return한 함수를 실행하면 정상적으로 변수 a에 접근합니다.
- 이는 make_closure 함수 내부에서 closure 함수를 선언할 때 Javascript의 클로저 기능에 의해 변수 a의 선언이 기억되기 때문!
- 이렇게 함수에 클로저 정보가 저장되는 것을 바인딩 이라고 한다.



실습 - Clojure

```
1  var a = 1;
2
3  function may_return_1(){
4      return a;
5  }
6
7  a = 2;
8
9  function may_return_2(){
10     return a;
11 }
12
13
14 may_return_1();
15 // 2
16
17 may_return_2();
18 // 2
```

- 예제를 하나 더 살펴봅시다.
- 클로저가 기억하는 것은 변수 그 자체를 기억하는 것이 아니라 변수의 선언 사실과 그 주소를 기억합니다.
- 만약 클로저가 변수의 내용까지 기억한다면 옆의 예시 코드가 1과 2 모두 출력해야 하겠지만 그렇지 않습니다.
- 변수의 주소가 기억되기 때문에 모두 2를 출력한다.



Class

- 앞에서 객체는 자신의 성질을 담는 속성과 동작을 정의한 메소드를 가진다고 했습니다.
- Javascript는 객체 기반의 스크립트 프로그래밍 언어입니다. 2015년 이전에는 class 키워드가 없어 함수를 이용해 우회적으로 클래스를 정의하여 사용했습니다. (함수도 객체니까요!)
- Class는 객체의 속성과 메소드를 명시적으로 선언할 수 있는 방법을 제공하며, 다양한 프로그래밍 언어에서 사용됩니다. 현재는 Javascript도 지원합니다!
- 1학년 컴퓨터프로그래밍2에서 배우는 Java의 Class랑 사용방법이 비슷합니다!



Object 선언 - Before 2015

- 함수 실행시에 this 키워드를 통해 속성을 지정하고 prototype 키워드를 통해 메소드를 지정한다.

```
1  function Book(title, author) {  
2      this.title = title;  
3      this.author = author;  
4  }  
5  
6  Book.prototype.getTitle = function() {  
7      return this.title;  
8  }  
9  
10 Book.prototype.getAuthor = function() {  
11     return this.author;  
12 }  
13  
14 var es5 = new Book('ECMAScript 5 - 2009', 'Ecma International');  
15 es5.getTitle()  
16 // ECMAScript 5 - 2009  
17 es5.getAuthor()  
18 // 'Ecma International'  
19
```




Object 선언 - After 2015 (Class)

- 이전 코드와 완전히 같은 기능을 하는 코드이다. Class를 이용해 메소드의 선언이 좀더 깔끔해 졌다.

```
1  class Book {  
2      constructor(title, author) {  
3          this.title = title;  
4          this.author = author;  
5      }  
6  
7      getTitle() {  
8          return this.title;  
9      }  
10  
11     getAuthor() {  
12         return this.author;  
13     }  
14 }  
15  
16 var es5 = new Book('ECMAScript 5 - 2009', 'Ecma International');  
17 es5.getTitle()  
18 // ECMAScript 5 - 2009  
19 es5.getAuthor()  
20 // 'Ecma International'
```



Object 선언 - Typescript

- 이전 코드와 완전히 같은 기능을 하는 코드입니다.
- Typescript는 Javascript의 superset이며 타입의 지정을 가능하게 합니다.
- 직전 코드보다 좀더 선언이 명확해졌다.
- Typescript는 알아만 둥시다.

```
1  class Book {  
2      private title: String;  
3      private author: String;  
4  
5      constructor(title, author) {  
6          this.title = title;  
7          this.author = author;  
8      }  
9  
10     public getTitle() {  
11         return this.title;  
12     }  
13  
14     public getAuthor() {  
15         return this.author;  
16     }  
17 }  
18  
19 var es5 = new Book('ECMAScript 5 - 2009', 'Ecma International');  
20 es5.getTitle()  
21 // ECMAScript 5 - 2009  
22 es5.getAuthor()  
23 // 'Ecma International'
```



과제 (~ 10/9 23:59)

- 2주차는 앞으로의 Node.js 공부를 위해 Node.js에서 많이 쓰이는 Javascript의 개념들을 공부했습니다.
- 과제는 1,2 두개 입니다!
- 제출기한: 10월 9일 자정 - 제출은 각 스터디장에게!
- 과제 제출 E-mail
 - 월 7시: 15 배민근 (baemingun@naver.com)
 - 화 7시: 16 백지훈 (bjh970913@gmail.com)



과제 1

- 오른쪽은 파일을 하나씩 쓰고 쓴 파일의 내용을 다시 읽어 새로운 파일을 만드는 코드입니다.
- 현재 오른쪽 코드는 Sync로 끝나는 동기 함수들을 사용하고 있습니다.
- 과제 1은 오른쪽과 같이 동작하는 코드를 비동기 함수를 이용하여 작성하는 것입니다. (Sync가 들어가면 안됩니다.)

```
1  const fs = require('fs');
2
3  fs.writeFileSync('./1.txt', '11111');
4  console.log('1.txt created.');
```

...

```
19 fs.writeFileSync('./5.txt', content4 + '55555');
20 console.log('5.txt created.');
```

...

```
23 console.log(content5);
```



과제 2

- 과제 2는 간단합니다. 오늘 배운 객체를 3가지 방법으로 정의하는 것입니다.
- 같은 객체를 Class 키워드를 사용하는 코드를 포함하여 2가지 이상의 방법으로 정의해주세요.
- 단, 3개 이상의 속성과 2개 이상의 메소드를 가져야 합니다.

〈Class를 이용해 코딩한 예제〉

```
1 class Book {  
2     constructor(title, author) {  
3         this.title = title;  
4         this.author = author;  
5     }  
6  
7     getTitle() {  
8         return this.title;  
9     }  
10  
11     getAuthor() {  
12         return this.author;  
13     }  
14 }  
15  
16 var es5 = new Book('ECMAScript 5 - 2009', 'Ecma International');  
17 es5.getTitle()  
18 // ECMAScript 5 - 2009  
19 es5.getAuthor()  
20 // 'Ecma International'
```



It's all today!