

KWEB Study Week6: Express.js + α

KWEB 2학기 준회원 스터디



Today's Contents

1. Before Study
2. Express Generator
3. ejs (with Express)
4. DB + SQL Intro
5. 게시판 Intro
6. 과제

M.G. BAE

J.H. BAEK



Before Study

- 지난 주차에 우리는 Routing, Express.js의 express 모듈에 대해 배우고 Node.js에서 http 요청을 받아보고 기본적인 express 모듈 사용에 대한 실습을 해보았습니다.
- 6주차의 목표는 Express Generator라는 걸 알아보고 ejs, DB 등 옛날에 배운 걸 Express.js 위에서도 사용해보고 SQL문까지 한 번 된다면 알아봅시다!
- One.. more.. time.. 죽..여..줘..
- 언제나 처럼 지난 수업 한 번 되짚어보고 시작하겠습니다!



Previous Study - Express.js

- Express.js는 기본적으로 require('express')를 통하여 얻어온 express 인스턴스에 각종 설정을 하고 인스턴스로 하여금 요청을 받도록 하는 listen 메소드를 호출하여 웹 어플리케이션을 구동합니다.
- 인스턴스에 행해지는 설정에는 미들웨어, Error 처리, 요청처리 3종류의 설정이 존재했습니다.
- 위의 3가지 설정을 모두 함수 형태로 이루어 지며 express는 각 함수의 인자수로 이들의 종류를 파악했습니다.



Previous Study - Middleware

- 요청에 대한 응답 과정 **중간**에 끼서 어떠한 동작을 해주는 프로그램
- 대체적으로 (req, res, next) 3개의 인자를 가지는 함수 형태입니다.
- 미들웨어는 전달된 req, res객체를 읽어 적절한 동작을 수행 후 res로 응답을 보내 요청 처리를 끝내거나 next 함수를 호출하여 다음 미들웨어로 실행흐름을 넘깁니다.
- Express.use 함수를 통해서 설정되며 use(fn(req, res, next){}) 형태로 사용되면 전역에 use('[라우팅 패턴]', fn(req, res, next){}) 형태로 사용되면 특정 라우트에만 적용됩니다.



Previous Study - Error 처리

- 사실 Error 처리도 Middleware 중 하나이며, 특이한 점은 (err, req, res, next) 4개의 인자를 가지는 함수 형태입니다.
- Error 처리 함수로 전달된 err 객체에는 앞서 실행 중 발생한 에러를 담고 있으며 함수는 이를 분석하여 자신이 처리할 수 있는 에러라면 처리하고 자신이 처리할 에러가 아니라고 판단되면 next(err)로 다음 에러처리 함수에게 실행흐름을 넘깁니다.
- Express.use 함수를 통해서 설정되며 use(fn(req, res, next){}) 형태로 사용되면 전역에 use('[라우팅 패턴]', fn(req, res, next){}) 형태로 사용되면 특정 라우트에만 적용됩니다.



Express generator

- 앞서 배운대로 익스프레스는 미들웨어, Error 처리, 응답처리 함수를 적절히 설정하고 listen을 실행시켜 구동을 시작합니다.
- 이러한 설정이 하나 둘 늘어나면 해당 내용을 담고 있는 javascript 파일은 점점 커져 관리가 힘들어 집니다.
- 따라서 내용을 적절히 분할하여 모듈화를 하고, 프로젝트 자체의 구조를 잡아야 점점 커지는 개발 작업의 효율성을 높일 수 있습니다.
- 처음 이러한 구조를 잡다보면 막막 할 수도 있지만 express 에는 이와 같은 작업을 도와주는 'express-generator' 라는 도구가 존재합니다.



Express generator

- 아무 것도 없이 코딩하기 힘들셨죠? 치트키 짜잔!

```
$ npm install -g express-generator
```

```
$ express --help

Usage: express [options] [dir]

Options:

  --version           output the version number
  -e, --ejs           add ejs engine support
  --pug              add pug engine support
  --hbs              add handlebars engine support
  -H, --hogan         add hogan.js engine support
  -v, --view <engine> add view <engine> support (dust|ejs|hbs|hjs|jade|pug|twig|vash) (defaults to jade)
  -c, --css <engine> add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css)
  --git              add .gitignore
  -f, --force         force on non-empty directory
  -h, --help          output usage information
```



Express generator 구조

```
$ express --ejs --git kweb-express-sample

warning: option '--ejs' has been renamed to '--view=ejs'

create : kweb-express-sample
create : kweb-express-sample/package.json
create : kweb-express-sample/app.js
create : kweb-express-sample/.gitignore
create : kweb-express-sample/routes
create : kweb-express-sample/routes/index.js
create : kweb-express-sample/routes/users.js
create : kweb-express-sample/public
create : kweb-express-sample/views
create : kweb-express-sample/views/index.ejs
create : kweb-express-sample/views/error.ejs
create : kweb-express-sample/bin
create : kweb-express-sample/bin/www
create : kweb-express-sample/public/javascripts
create : kweb-express-sample/public/images
create : kweb-express-sample/public/stylesheets
create : kweb-express-sample/public/stylesheets/style.css

install dependencies:
$ cd kweb-express-sample && npm install

run the app:
$ DEBUG=kweb-express-sample:* npm start
```

```

v kweb-express-sample
  v bin
    www
  v public
    v images
    v javascripts
    v stylesheets
      style.css
  v routes
    index.js
    users.js
  v views
    error.ejs
    index.ejs
    .gitignore
    app.js
    package.json

```

- express 명령어
 - --ejs: EJS 를 뷰 템플릿 엔진으로써 사용하겠다.
 - --git: Git 연동을 위해서 node_modules 폴더나 이미지 등 기타 파일을 무시하게 해주는 .gitignore를 생성해달라는 옵션
- express generator 폴더 구조
 - /bin : 몇몇 환경 설정이 추가된 구동 파일입니다.
 - /public : 이미지, JS, css파일이 들어가는 폴더
 - /route : 실제 응답처리 함수를 바인딩하는 소스들
 - /views : 뷰 템플릿이 있는 폴더
 - app.js : express 설정 파일
 - package.json : npm 패키지 설정 파일



Express generator 구조 - package.json

```
1 {  
2   "name": "kweb-express-sample",  
3   "version": "0.0.0",  
4   "private": true,  
5   "scripts": {  
6     "start": "node ./bin/www"  
7   },  
8   "dependencies": {  
9     "body-parser": "~1.18.2",  
10    "cookie-parser": "~1.4.3",  
11    "debug": "~2.6.9",  
12    "ejs": "~2.5.7",  
13    "express": "~4.15.5",  
14    "morgan": "~1.9.0",  
15    "serve-favicon": "~2.4.5"  
16  }  
17 }  
18
```

- Scripts:start
 - npm start나 npm run start 명령어를 입력시에 bin/www를 실행시켜 서버를 구동합니다.
- Dependencies
 - 생성기를 통해서 자동으로 추가된 의존성입니다.
 - 각론 미들웨어와 ejs 뷰 엔진, express 가 포함되어 있습니다.



Express generator 구조 - bin/www

```
var app = require('../app');
var debug = require('debug')('kweb-express-sample:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);
```

- 운영체제의 환경변수 또는 기본 설정을 이용해 포트를 설정하고 app.js의 express 모듈을 nodejs 기본 http모듈에 등록시킵니다.
- 따라서 개발자는 express.listen을 실행시킬 필요없이 app.js에서 express 모듈을 export하기만 하면 됩니다.
- 어플의 실행 포트를 바꾸고 싶으면 <윈도우는 SET PORT=8080>, <Unix계열은 export PORT='8080'> 을 통해서 환경 변수를 설정하거나 bin/www를 수정하면 됩니다.



Express generator 구조 - app.js

```
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
```

- 뷰 엔진 미들웨어를 설정합니다.
 - Views 변수는 템플릿엔진 저장 폴더를 가리키며
 - View engine 변수는 사용 엔진을 설정 합니다.

```
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
```

- logger, bodyParser, cookieParser, 정적 파일 서빙 미들웨어를 설정합니다.

```
app.use('/', index);
app.use('/users', users);
```

- 응답처리 함수를 설정합니다.
 - 대부분의 경우에 응답처리 함수는 express의 router 객체에 담겨 계층적으로 설정 됩니다.



Express generator 구조 - app.js

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});
```

- 저번 시간 과제에 있던 내용인 404 Error 처리 미들웨어입니다.

- Express는 명시적으로 404에러를 발생시키지 않기 때문에 라우팅이 끝난 이후에 특수한 미들웨어를 삽입하여 내용을 처리해야 합니다.

- 전역 애러 처리 함수를 설정합니다.

- 기본은 모든 애러를 잡아 로깅후 Error 문자열을 500 응답코드와 함께 응답합니다.



Express generator 구조 - route/index.js

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

- app.js 에서 /에 바인딩되는 라우터 입니다. res.send가 아닌 res.render 함수를 호출하는 것을 볼 수 있습니다.
- express는 render 함수가 호출되면 views 설정에 따라 뷰 템플릿 파일을 찾고 view engine 설정에 맞는 뷰 엔진을 통해 렌더링 된 내용을 응답으로 보내 줍니다.
- 이 파일의 설정으로는 views/index.ejs 파일에 title이 Express인 데이터를 렌더링하여 응답합니다.



Express generator 구조 - route/board.js

```
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res, next) {
  res.send('respond with a resource');
});

module.exports = router;
```

- App.js 에서 /users에 바인딩되는 라우터 입니다.
- 이 파일에서 아래와 같은 코드를 입력해봅시다.

```
router.get('/profile', function(req, res) {
  })
```
- 위와 같은 코드로 바인딩하면 실제로는 /user/profile 로 라우팅 데이터가 설정됩니다.



Express generator 구조 - views/index.ejs

- 앞의 routes/index.js에서 사용되는 ejs 파일 입니다.
- 우리가 앞서 view engine을 ejs로 설정 해서 ejs 파일로써 View가 보여집니다.
- Title 변수를 몇 번 출력하고 있습니다.
- 그렇다면 이제 Express에서 ejs를 쓸 수 있어야겠죠?

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```



- 복습만 잠깐 하고 쪽 실습으로 해봅시다!
- EJS는 View Engine 중 하나로 기존의 HTML과 동일하게 웹 문서에서 사용하는 태그를 템플릿 파일로 집어넣을 수 있습니다. (당연하게도 외장 모듈로써 사용합니다!)
- View Template: 클라이언트에 응답을 보낼 때 사용하려고 미리 만들어 놓은 웹 문서의 원형
- View Engine: 뷰 템플릿을 사용해 결과 웹 문서를 자동으로 생성한 후 응답을 보내는 역할을 수행합니다.



실습 - ejs

- 예전에 해보셨죠? 그대로 입니다!
- 기본 출력 예제인데 따라해봅시다.

```
index.js
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET home page. */
5 router.get('/', function(req, res, next) {
6   res.render('index', {
7     str01: 'Hello',
8     str02: 'World!',
9     arr01: [1, 2, 3, 4],
10    obj01: {
11      key01: 'data01',
12      key02: 'data02'
13    },
14    condition: true
15  });
16 });
17
18 module.exports = router;
19
```

```
index.ejs
1 <b>String</b>
2 <p>
3   <span><%= str01%></span> <span><%= str02%></span>
4 </p>
5 <b>Array</b>
6 <p><%= arr01%></p>
7 <b>Object</b>
8 <p><%= obj01%></p>
9 <b>Boolean</b>
10 <p><%= condition%></p>
11
```

String
Hello World!

Array
1,2,3,4

Object
[object Object]

Boolean
true



실습 - ejs

• 좀 더 해봅시다1 - 조건

```
index.js
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET home page. */
5 router.get('/', function(req, res, next) {
6   res.render('index', {
7     str01: 'Hello',
8     str02: 'World!',
9     arr01: [1, 2, 3, 4],
10    obj01: {
11      key01: 'data01',
12      key02: 'data02'
13    },
14    condition: true
15  });
16 });
17
18 module.exports = router;
19
```

```
index.ejs
1 <ul>
2   <% for (let arr_index=0; arr_index<arr01.length; arr_index++) { %>
3     <li><%= arr01[arr_index]%></li>
4   <% } %>
5 </ul>
6 <ul>
7   <% for (let num of arr01) { %>
8     <li><%= num%></li>
9   <% } %>
10 </ul>
11
```

localhost:3000

- 1
- 2
- 3
- 4

• 좀 더 해봅시다2 - 반복

```
index.js
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET home page. */
5 router.get('/', function(req, res, next) {
6   res.render('index', {
7     str01: 'Hello',
8     str02: 'World!',
9     arr01: [1, 2, 3, 4],
10    obj01: {
11      key01: 'data01',
12      key02: 'data02'
13    },
14    condition: true
15  });
16 });
17
18 module.exports = router;
19
```

```
index.ejs
1 <% if (condition) { %>
2   <p>Condition OK!</p>
3 <% } else { %>
4   <p>Condition BAD !!!</p>
5 <% } %>
6
```

localhost:3000

Condition OK!



DB Remind

- 아마.. 실제 연결할 시간은 이번 시간에 안될 거 같으니! (다음 시간에 합니다 ㅎㅎ) WEB 서비스에서 매우 아주 중요한 DB에 대해 잠깐 Remind 해봅시다.
- DB: 여러 사람에 의해 공유되어 사용될 목적으로 통합하여 관리되는 데이터의 집합
- DBMS (Database Management System): 데이터베이스를 관리하며 응용 프로그램들이 데이터베이스를 공유하며 사용할 수 있는 환경을 제공하는 소프트웨어
- Relational DB vs. NoSQL DB (사실 종류는 더 많지만 우리는 2가지에만 초점!)



NoSQL (with MongoDB)

- 우리는 이미! 3주차에서 대표적인 NoSQL인 MongoDB를 이용하여 실습을 진행해보았습니다.
- Not only SQL → 데이터를 저장하는데 SQL뿐만 아니라 다른 방법이 있다!
- 대표적인 NoSQL: MongoDB, OrientDB, Hadoop, Cassandra, etc.
- 까먹으셨죠? 실습으로 복습 한 번 해봅시다. DB는 이렇게 연결하는거구나 같은 감각을 익혀봅시다. (잘 기억나시는 분은 건너뛰셔도 됩니다!)



실습 - MongoDB

- 오른쪽 코드를 살펴봅시다.
- 위에서부터 차례로 ① 모듈 로딩, ② DB 연결, ③ DB collection 선택이라는 기능을 수행합니다.

```
const MongoDB = require('mongodb');  
const MongoClient = MongoDB.MongoClient;
```

```
MongoClient.connect(url, function (err, _db) {  
  if (err) {  
    throw err;  
  }  
  ...  
});
```

```
book_collection = db.collection('books');
```



실습 - MongoDB

- 오른쪽은 DB가 대표적으로 지니고 있는 CRUD 기능을 사용해본 코드입니다.
- CRUD는 데이터를 처리하는 시스템이 지속성을 갖기 위해 갖춰야 하는 기본적인 데이터 처리 4가지 기능입니다.
- 각각이 Create, Read, Update, Delete 어디에 대응되는지 맞춰보세요~

```
const book = new Book('제 마음도 괜찮아질까요?', '강현식', '서늘한  
const books = [  
  new Book('집 살래 월세 살래', '이재범(핑크팬더)', 15, 17000,  
  new Book('잘 자, 굴삭기 벤!', '되르테 혼', 10, 12000, '116051  
];  
  
book_collection.insertOne(book);  
book_collection.insert(books);
```

```
book_collection.updateMany({  
  ISBN: '116051111X'  
}, {  
  $set: {  
    price: 1000000000,  
    remaining: 9999999999  
  }  
}, () => {  
})
```

```
book_collection.find({  
  ISBN: '116051111X'  
}).toArray((err, data) => {  
  console.log(data);  
});
```

```
book_collection.deleteMany({  
  ISBN: '1187383279'  
}, () => {  
});
```



Relational DB

- 자 그럼, NoSQL은 여기까지 하고! (사실 3주차에서 이미 대충 배웠죠? 어차피 전공은 아니니 사용할 수 있을 수준으로 알면 됩니다.) RDB에 대해 복습해봅시다.
- 관계형 모델을 기반으로 하는 데이터베이스 관리 시스템이다.
 - 관계형 모델이란? → 데이터를 column과 row를 이루는 하나 이상의 테이블(또는 관계)로 정리하며, 고유 키(Primary key)가 각 row를 식별한다.
- DB계의 주류이며, 관계형 모델 기반이니 데이터를 column과 row라는 일종의 표 형태로 저장함! (한 Table의 row는 같은 길이의 column을 가져요!)
- 관계형 DB 종류: MySQL, MariaDB, MSSQL, Oracle, etc.



SQL

- 예전에도 말했듯이 데이터베이스를 사용할 때, 데이터베이스에 접근할 수 있는 데이터베이스 하부 언어를 말한다.
- 요새는 다른 종류의 DB에도 널리 쓰이나 원래는 IBM의 Relational DB에서 사용되었습니다! (NoSQL 계열 DB를 제외하곤 대부분의 DB에서 사용됩니다.)
- 위의 말을 종합해보면 우리가 쓸 DB계의 주류인 RDB에서도 뭔가 쓸 거 같지 않나요?



RDB & SQL

- SQL은 관계형 데이터베이스와 통신하는 데 사용되는 기본 인터페이스입니다.
- 다른 말로 하면! SQL로 데이터베이스에서 질의 기능, 데이터 정의 및 조작 기능이 모두 가능합니다.
- SQL 구문은 DB 종류에 따라 다르지만 표준 SQL이 존재하며 대체로 이와 비슷한 형태를 띄고 있습니다!
- WEB 서비스에서 RDB는 상당히 중요한 부분을 차지하고 있고 따라서 이를 조작하는 SQL문도 알아야할 필요성이 있습니다. (3학년 DB 전공에서도 사용합니다.)



SQL Intro

- But, 오늘은 짧게 Introduction 정도만 배워봅시다.
- 다음 주부터 대표적인 RDB인 MariaDB(MySQL)에서 사용하는 SQL문에 대해 배울 것이고! 오늘은 표준 SQL 구문을 사용해봅시다.
- CRUD 기능 정도만 가볍게 봅시다.
- 다음 시간에 코드에 적용시켜봅시다~



SQL - CRUD

- INSERT INTO: 행 데이터 또는 테이블 데이터의 삽입에 사용됩니다.
 - ex) INSERT INTO A (column1, column2) VALUES ('value1', 'value2')
- SELECT ~ FROM ~ WHERE: 테이블 데이터의 검색 결과를 얻을 수 있습니다.
 - ex) SELECT * FROM A WHERE <조건>
- UPDATE ~ SET: DB 안의 데이터를 수정하는데 사용됩니다.
 - ex) UPDATE A SET `column` = `value` WHERE <조건>
- DELETE FROM: 테이블에서 특정 행, 레코드를 삭제하는데 사용됩니다.
 - ex) DELETE FROM A WHERE <조건>



게시판 Intro

- DB 연결을 시작할 거 같으니! 우리는 홈페이지의 기본 게시판 제작을 다음주까지 목표로 할 것입니다.
- 이것저것 게시판에 필요한 기능들이 많겠죠?
- 준회원 스터디 실습에선 기본적인 게시판 기능들만 구현해볼테니 한 번 추가적으로 Personal한 기능들을 더 넣어보세요!
- 그럼 게시판 만들기 시작하겠습니다~



게시판 Intro

- 인줄 아셨겠지만 이번주는 기본 틀만 잡아볼겁니다.
- Express generator로 프로젝트를 생성해봅시다.
- 그리고 간단히 Router 정도만 잡아봅시다.
 - GET / : 게시판의 게시물을 리스팅 합니다.
 - GET /write : 게시물 작성 폼을 가진 html 페이지를 보여줍니다.
 - POST /write : 작성페이지에서 보낸 post 요청을 읽어 게시물을 생성합니다.

```
router.get('/', function(req, res, next) {  
  res.render('board_list', {  
    articles: BoardService.findAll();  
  });  
});  
  
router.get('/write', function(req, res, next) {  
  res.render('board_write', {  
    articles: BoardService.findAll();  
  });  
});  
  
router.post('/write', function(req, res, next) {  
  res.redirect('/');  
});
```



과제 (~ 11/27 17:00)

- 6주차는 Express.js를 이용해 지금까지 배운 것을 더 사용해보고 추가적으로 DB와 SQL을 맛보았습니다.
- 과제는 다음 슬라이드의 명세대로 하셔서 제출하시면 됩니다! Github에 올려 링크로 카톡으로 제출해주세요~
- 제출기한: 11월 27일 오후 5시까지
- 과제 제출 E-mail
 - 월 7시: 15 배민근 (baemingun@naver.com)
 - 화 7시: 16 백지훈 (bjh970913@gmail.com)



과제

- ① board_list.ejs에 /board/write 로 가는 링크를 추가한다.
- ② board_write.ejs에 /board/write로 POST요청을 통해 제목과 내용을 보내는 폼을 작성한다.
- ③ POST /board/write에서 받은 post내용을 통해 (글을 생성하는 기능은 DB 연결을 안했으니 냅두고!) 게시물 리스팅 페이지로 이동하게 한다.
- ④ GET /board/delete 라우트를 생성하고 url parameter 'id'로 넘어온 글을 (삭제는 DB 연결을 안했으니 또 냅두고!) 보여준다.



It's all today!