KWEB Study Week5: Express.js

KWEB 2학기 준회원 스터디

Today's Contents

1. Before Study
2. HTTP with Node.js
3. Express.js Intro
4. Express.js 실습
5. 과제

M.G. BAE

J.H. BAEK

2017 Fall Semester

Before Study

- 지난 주차에 우리는 지난 시간에 HTTP 프로토콜에 대해서 배워 보았습니다.
- 죽..여..줘..
- 5주차의 목표는 Node.js를 통해서 직접 http 요청을 받아보고 요청 처리를 도와주는 express 모듈에 대하여 알아보겠습니다.
- 지난 수업들 한 번 되짚어보고 시작할게요! + Postman이라는 Tool 설치도!

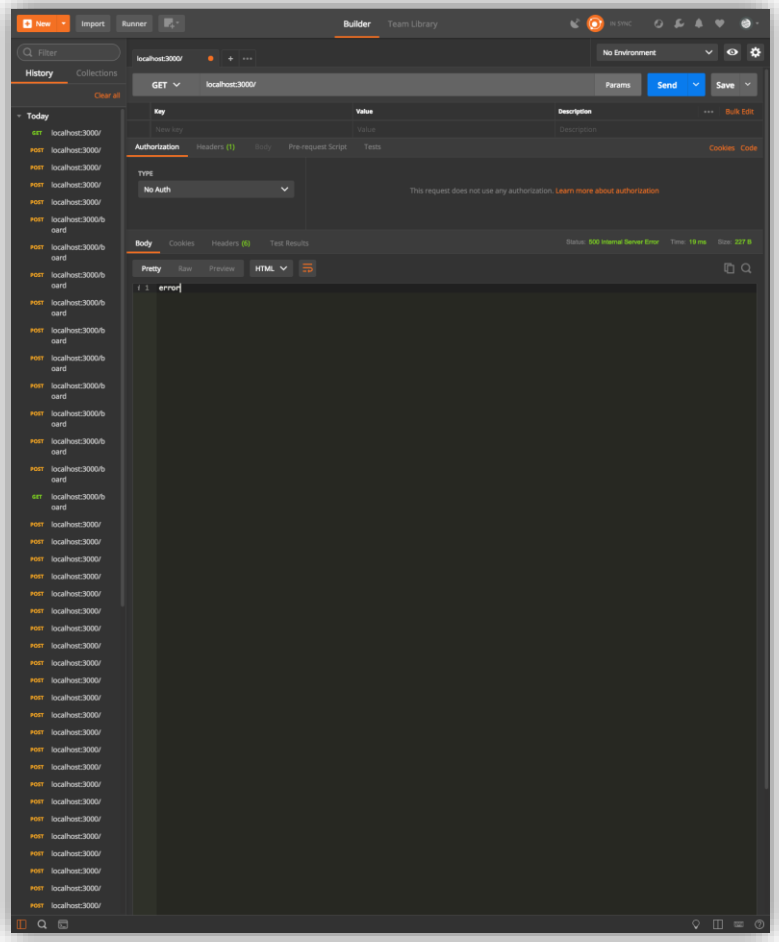
Previous Study

- HTTP 프로토콜은 웹 브라우저(클라이언트)와 웹 서버(서버)간에 HTML을 전송하기 위한 프로토롤로 Header와 Body로 이루어져 있습니다.

- 헤더에는 요청의 타입(메소드), 요청을 받을 서버(호스트), 서버의 리소스 경로(path)와 기타 정보들이 담겨있고

- 바디에는 요청의 실제 데이터가 담겨있었습니다.

Postman 설치



- 아래 주소에서 Postman을 다운받아 설치합시다.
  - https://www.getpostman.com/
- Postman은 http요청을 생성해 주는 프로그램입니다.
- Postman을 이용하면 개발시에 간단한 GET 요청이나 POST요청을 브라우저를 사용하지 않고도 요청할 수 있습니다.

예제 코드

- 오늘은 예제 코드가 실습을 위한 예제 코드가 주어집니다.

- Bit bash 나 Terminal에 아래 명령어를 입력해서 예제 코드를 받아주세요!

```
$ git clone https://github.com/bjh970913/KWEB-2017-2R-NodeJS-study-example-code.git
```

- 성공후에는 생성된 ‘KWEB-2017-2R-NodeJS-study-example-code’ 디렉터리에서 npm install을 한번 실행시켜 주세요

HTTP with Node.js

- 예제 코드 폴더에서 아래 명령어를 치면 node의 http 모듈만을 사용한 웹서버 예제가 실행 됩니다.

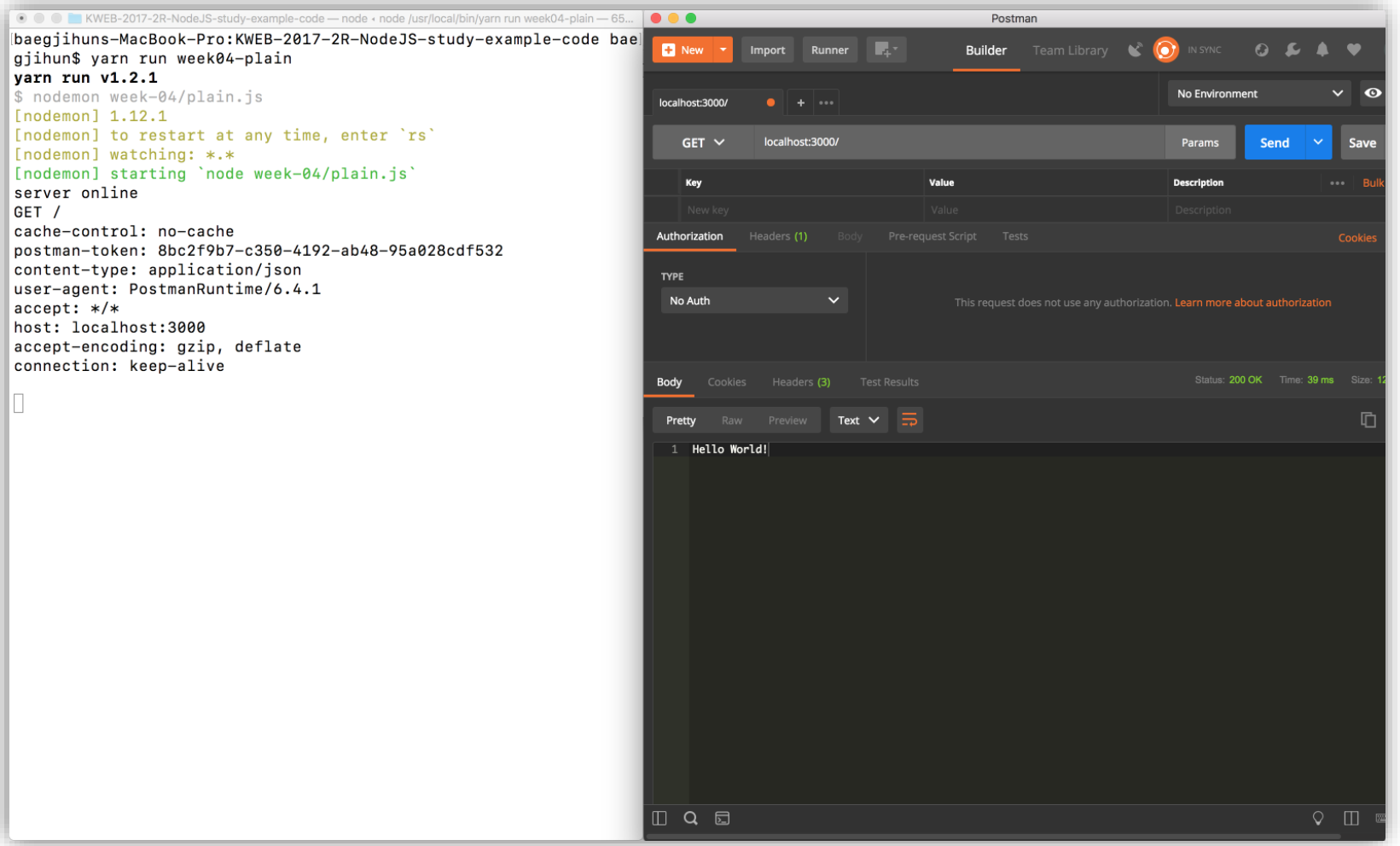
```
$ npm run week05-plain
```

- 이제 Postman을 통해서 직접 웹서버에 요청을 날려 봅시다.

실습 - 예제 코드에 GET / 요청 보내기

- Postman을 이용해 GET / 요청을 보내봅시다.

- 보내고 Console 창 확인!
  - Method: GET
  - 경로: /
  - 호스트: localhost:3000

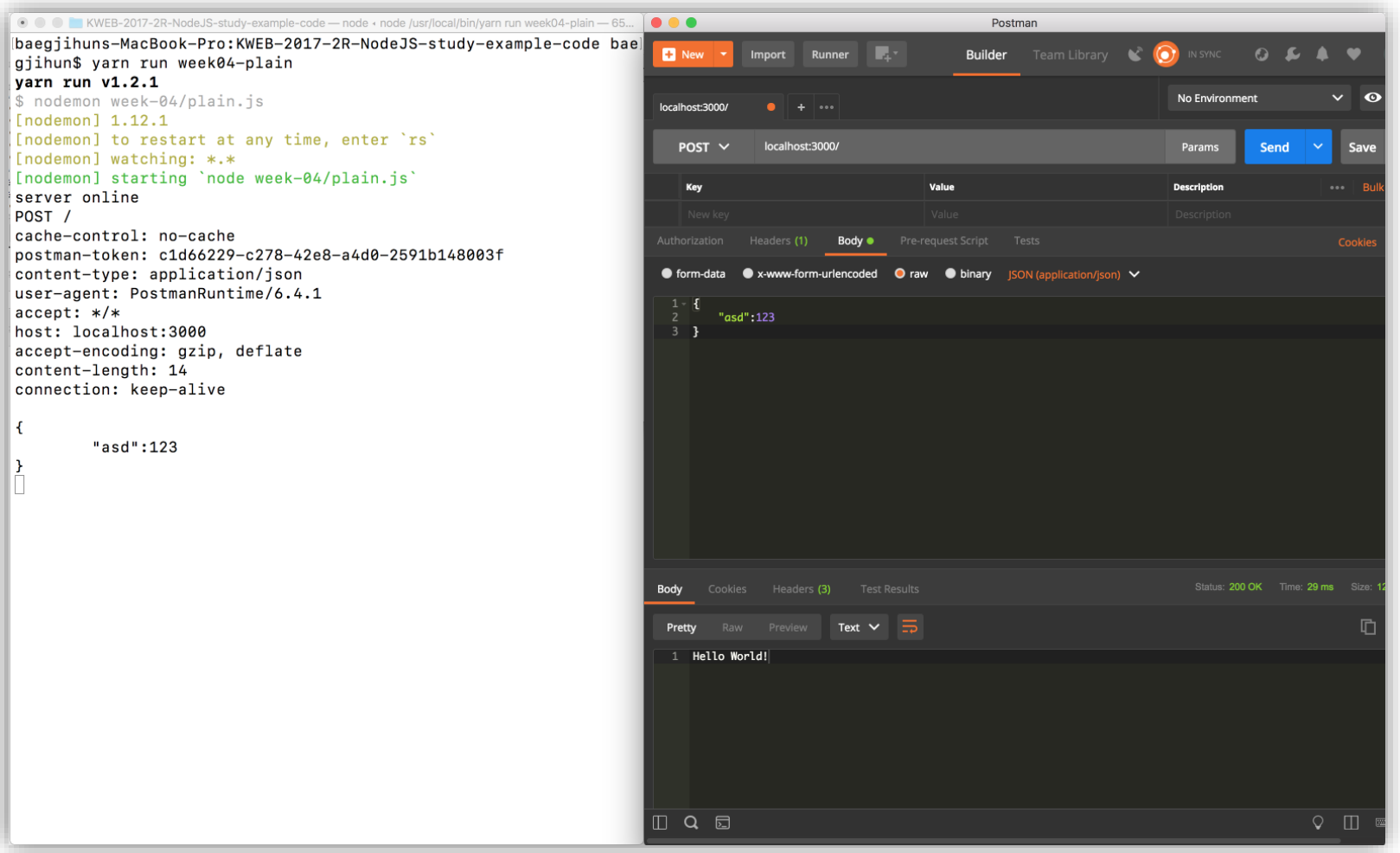


실습 - 예제 코드에 POST / 요청 보내기

- Postman을 이용해 POST / 요청을 보내봅시다.

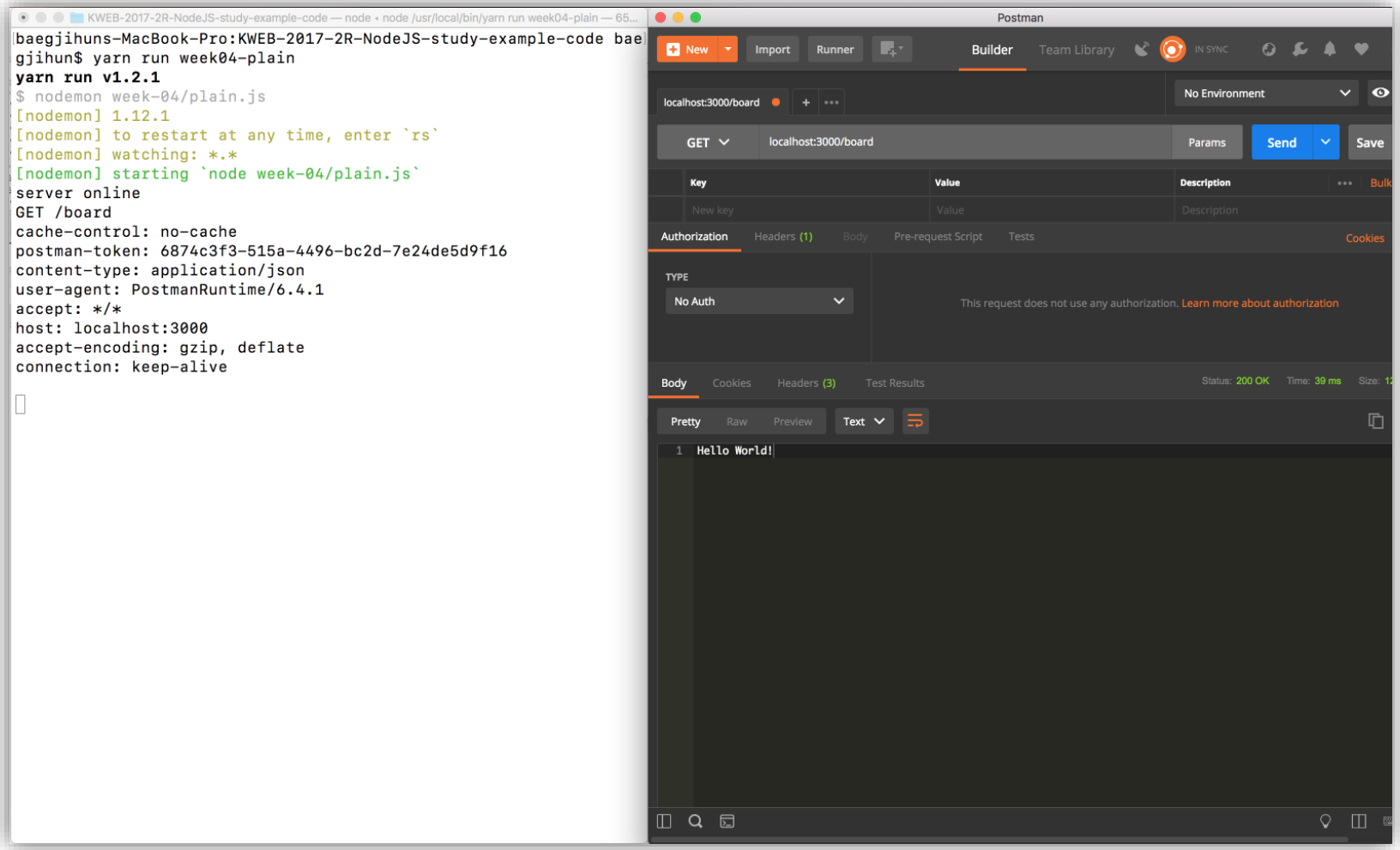
- 보내고 Console 창 확인!
  - Method: POST
  - 경로: /
  - 호스트: localhost:3000

- GET / 요청과 다르게 Body에 내용(Content)가 전달되었습니다.



실습 - 예제 코드에 GET /board 요청 보내기

- 이번엔 Postman을 이용해 GET /board 요청을 보내봅시다.
- 보내고 Console 창 확인!
  - Method: GET
  - 경로: /board
  - 호스트: localhost:3000
- GET / 과 큰 차이를 보이진 않습니다.



URL routing

- 앞서 실습한 내용을 보면 각기 다른 내용으로 요청을 보냈음에도 서버의 응답은 항상 같습니다.
- 이는 우리가 아직 서버에게 어떤 메소드, 어떤 경로일때 어떤 로직을 실행할지 알려주지 않았기 때문에 그렇습니다. → 네, 구현을 안했다고요. 네.
- 각각의 경우에 무엇을 할지는 개발자가 직접 http 프로토콜의 헤더를 참고해 정해주어야 합니다.
- 다시 말하면, 특정 http 요청에 응답할 내용들을 미리 다 만들어놔야한다는 말이죠!

실습 - URL Routing

- 예제 코드 폴더에서 아래의 명령어를 치면 메소드와 경로에 따라 다르게 작동하는 예제 서버가 작동됩니다. (미리 만들어뒀단 얘기죠.)

```
$ npm run week05-plain-route
```

- 이전과 동일하게 Postman을 통해서 직접 웹서버에 요청을 날려 봅시다.

- 어떻게 다른지 알아봅시다.

URL Routing

```
if (req.method.toUpperCase() === 'GET') {
  switch (req.url) {
    case '/':
      main(res);
      break;
    case '/board':
      board(res);
      break;
    default:
      throw new Error('404 Not found');
  }
} else if (req.method.toUpperCase() === 'POST') {
  switch (req.url) {
    case '/board':
      board_write(res);
      break;
    default:
      throw new Error('404 Not found');
  }
} else {
  throw new Error('404 Not found');
}
```

- /week-05/plain-route.js 파일에서 요청에 대한 라우팅을 하는 부분의 코드입니다.
  - 이 소스에서는 단순히 요청의 메소드와 경로(URL)을 문자열 비교하여 동작을 정합니다.
- 하지만 현실에서는 Url 에 포함된 파라메터 파싱, 경로 변수 추출, 정적 파일 요청(CSS, JS)일시에 해당 파일 로딩, Error 처리 등 생각해야 할 부분이 더 많습니다.
- 이걸 모두 구현 하면 힘들어 지겠죠?

Express.js

- 앞에서의 복잡한 문제를 해결해줄 모듈이 있습니다. → Express 모듈!
- Express.js는 Node.js Web Application Framework로 여러 기능들을 제공합니다.
- http 모듈만 사용해서 웹서버를 구성하면 많은 것들을 직접 만들어야합니다. 그러면 시간과 노력이 많이 들겠죠? 그래서 보다 간편하게 해줄 것이 필요합니다!
- Express.js는 다른 여러 모듈과의 조합을 통해 라우팅, 오류처리, 파싱, 파일 로딩 등 여러 기능을 직접 개발하지 않고 사용할 수 있는 기반을 제공 합니다.

실습 - Express.js 사용하기

- 아래의 명령어면 Node.js에서 Express.js 사용가능!

```
$ npm install express --save
```

- 오른쪽 코드를 한번 실행시켜봅시다.

```
const express = require('express');
const app = express();

app.get('/', function (req, res) {
  console.log(`${req.method} ${req.url}
${Object.keys(req.headers).map(k => `${k}: ${req.headers[k]}`).join('\n')}
`);

  req.on('data', d => console.log(d.toString()));

  res.send('Hello World!');
});

app.listen(3000, function () {
  console.log('server online');
});
```

실습 - Express.js 사용하기

```
const express = require('express');
const app = express();

app.get('/', function (req, res) {
  console.log(`${req.method} ${req.url}
${Object.keys(req.headers).map(k => `${k}: ${req.headers[k]}`).join('\n')}
`);

  req.on('data', d => console.log(d.toString()));

  res.send('Hello World!');
});

app.listen(3000, function () {
  console.log('server online');
});
```

- express(): express 객체 생성
- Get 요청에 대한 핸들러를 지정합니다. 메소드별로 함수가 존재하며 첫번째 인자는 path를 두번째 인자로는 함수를 받습니다.
- Express가 특정 주소와 포트에 대해서 요청을 받기 사작합니다.

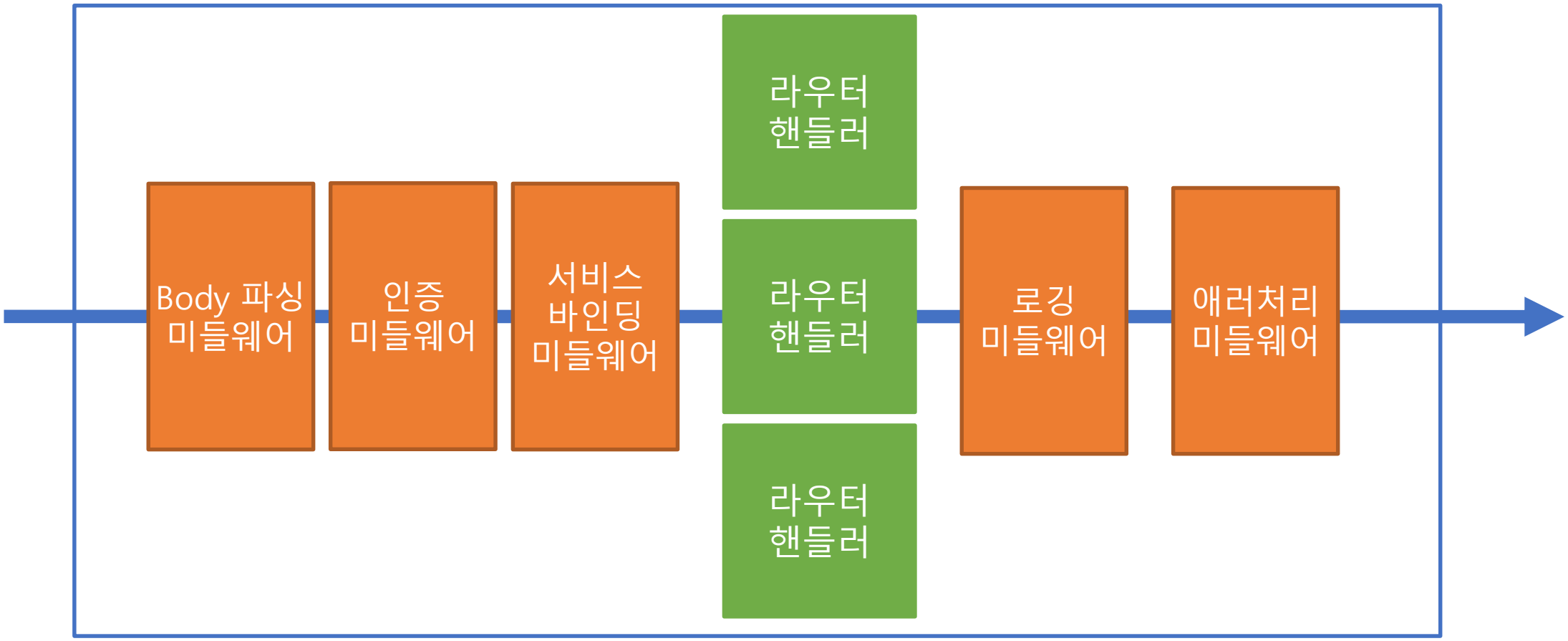
Express Server Object

- 앞의 코드에서 const app = express() 를 살펴봅시다.

- 여기서 app 객체는 express() 메소드 호출로 만들어지는 **익스프레스 서버 객체**입니다!

- Method in Express server object
  - set(name, value) → 서버 설정을 위한 속성을 지정합니다. set() 메소드로 지정한 속성은 get() 메소드로 꺼내어 확인할 수 있습니다.
  - get(name) → 서버 설정을 위해 지정한 속성을 꺼내 옵니다.
  - use([path,] function [,function…]) → 미들웨어 함수를 사용합니다.
  - get([path,] function) → 특정 패스로 요청된 정보를 처리합니다.

Express.js 처리과정



Middleware

- 요청에 대한 응답 과정 **중간**에 껴서 어떠한 동작을 해주는 프로그램

- 미들웨어는 요청을 처리하는 도중 다음과 같은 역할을 수행합니다.
  - 모든 코드를 실행. (ex, 로깅)
  - 요청 및 응답 오브젝트에 대한 변경을 실행. (ex, 파싱)
  - 요청-응답 주기를 종료. (ex, res.end())
  - 스택 내의 그 다음 미들웨어를 호출. (ex, next())

- 각각의 미들웨어는 next() 메소드를 호출하여 그 다음 미들웨어가 처리할 수 있도록 순서를 넘길 수 있다. (다다음 페이지에서 그림으로 봅시다.)

Middleware

- 미들웨어는 3개의 인자를 가지는 함수 형태입니다.

```
function (req, res, next) {
    …
    next()
}
```

- 익스프레스는 미들웨어, 애러처리, 컨트롤러를 인자의 개수로 구분 합니다.

- 대부분의 경우 자신의 로직을 실행 후 next() 메소드를 통해 실행 권한을 넘깁니다.

Express.js

app 객체

미들웨어 설정

라우터 설정

use()로 설정

use()로 설정

use()로 설정

get()으로 설정

클라이언트 요청

미들웨어 #0

미들웨어 #1

미들웨어 #2

next()

next()

next()

미들웨어

요청 패턴 /

요청 패턴 /users

라우터

클라이언트 응답

Routing

- 라우터는 Express 가 켜지면서 정의된 URL path 맵핑에 따라서 적절한 기능을 순서대로 실행 합니다. (라우터에 등록되는 경로는 정규식을 포함 할 수 있습니다.)

- 실행되는 기능에는 미들웨어(parser, Error 처리, etc.), 컨트롤러(핸들러를 대체로 컨트롤러라함!)가 포함 됩니다.
  - app.use('/path/A', middlewareA); → /path/A 에 대해서 middlewareA를 실행
  - app.get('/board', boardController); → /board 에 대해서 boardController를 실행
  - app.use(errorControl); → 전체 path에 대해서 에러처리 미들웨어를 사용

- 라우터에 등록된 데이터는 Express bootstrap 과정에서 정의된 순서대로 실행됩니다.

Req & Res Object

- Express에서 사용하는 요청 객체와 응답 객체는 http 모듈에서 사용하는 객체들과 같지만 아래 몇가지 메소드가 추가되어 있습니다.
  - send([body]) → 클라이언트에 응답 데이터를 보냅니다. (HTML 문서, Buffer 객체, JSON, etc.)
  - status(code) → HTTP 상태 코드를 반환합니다. 상태 코드는 end()나 send()같은 전송 메소드를 추가로 호출해야 전송할 수 있습니다.
  - sendStatus(statusCode) → HTTP 상태 코드를 반환합니다. 상태 코드는 상태 메시지와 함께 전송됩니다.
  - rediret([status,] path) → 웹 페이지 경로를 강제로 이동시킵니다.
  - render(view [, locals][, callback]) → 뷰 엔진을 사용해 문서를 만든 후 전송합니다. (담주에!)

- 요청 객체에 추가된 헤더와 파라미터도 있습니다!
  - req.query: GET 방식 전송 요청 파라미터 확인, req.body: POST 방식 전송 요청 파라미터 확인, req.header(name): 헤더를 확인합니다.

Error 처리

- Express의 Error 처리 함수는 4개의 인자를 가지며 자신보다 앞선 미들웨어나 컨트롤러에서 예외처리 되지 않는 예외가 존재하면 호출 됩니다.

- 이 함수는 Error의 내용을 확인하고 응답을 수정하거나 next를 실행시켜 실행권한을 넘깁니다.

실습

- 오늘은 간단히 Express.js 에 내장된 2가지를 사용해볼 것입니다.
- 대표적인 Express의 Middleware인 body-parser와 express-error.js를 사용해보는 것이 이번 시간의 마지막 실습입니다.
- 이번 주차가 내용이 많아 실습을 더 넣으려다가 생략했습니다! 그래서!
- 다음주에 SQL문과 DB와 함께 Routing, Middleware 실습 등을 더 해볼 것입니다. 이번 실습으로는 감이 오지 않는다고 너무 겁먹지 않아도 됩니다~

실습 - body-parser

- 아래의 명령어를 입력하고 코드를 입력해봅시다.

```
$ npm install body-parser
```

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

app.post('/', function (req, res) {
  console.log(`${req.method} ${req.url}
${Object.keys(req.headers).map(k => `${k}: ${req.headers[k]}`).join('\n')}
`);
  console.log(req.body);
  // req.on('data', d => console.log(d.toString()));
```

실습 - Error 처리

- express-error.js 를 사용해봅시다.
- 아래의 명령어를 입력해서 사용해봅시다.

```
$ npm run week05-express-error
```

- express-error.js 는 익스프레스에서 에러를 처리하는 과정을 보여 줍니다.

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

app.get('/', function (req, res) {
  throw new Error('TEST');
});

app.use(function (err, req, res, next) {
  console.log(err);
  res.statusCode = 500;
  res.send('error');
});

app.listen(3000, function () {
  console.log('server online');
});
```

과제 (~ 11/12 23:59)

- 5주차는 Express.js와 그 안의 Middleware, Routing에 관해 공부했습니다.
- 과제는 필수 제출 1개, Optional 2개입니다!
- 제출기한: 11월 20일 오후 5시까지
- 과제 제출 E-mail
  - 월 7시: 15 배민근 (baemingun@naver.com)
  - 화 7시: 16 백지훈 (bjh970913@gmail.com)

과제

- 필수

→ 앞서 실습한 Week-05/plain-route.js 와 동일한 기능을 수행하는 코드를 expres를 사용하여 작성한 express-board.js 제출하는 것입니다.

- Optional

→ 선택이어서 제출은 없지만! 웬만하면 해오세요!

① 예전에 공부했던 1,2차시에 공부했던 ejs (view template) 공부해오기

② 오늘 Express.js 강의한 내용 복습! 담주에 실습을 더 진행할겁니다!

It’s all today!