Node.js Server

HTTP & Express.js

2023-1 T.G.WinG 강의식 스터디

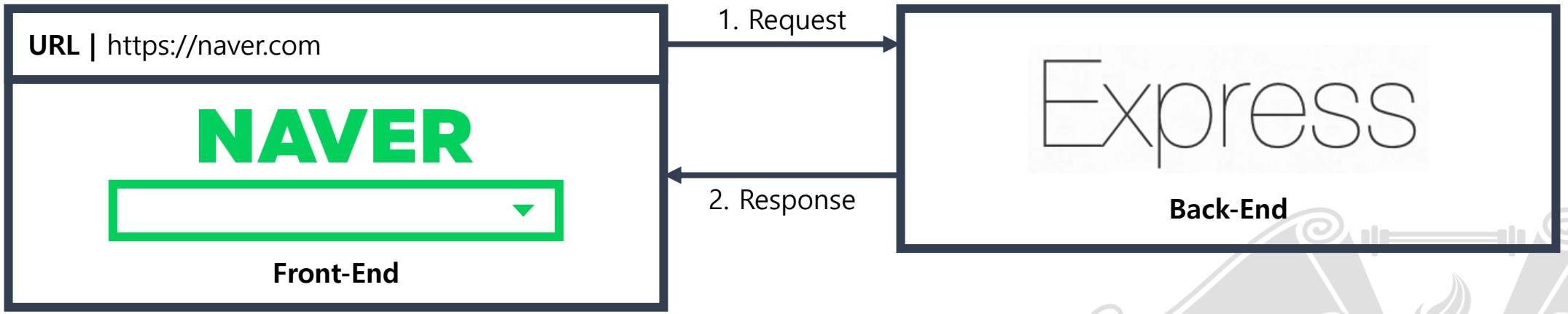
배승호

KHU TGWinG

Web(Front, Back)

❖ Web은 크게 Front-end와 Back-end로 구분되어 진다

- Front-end: 사용자에게 보여주는 **UI를 구성**
- Back-end: 사용자에게 보여줄 **데이터를 관리, 제공**



Network Layer

❖ 네트워크는 **계층적**으로 이루어진다

- 제일 위의 계층부터 데이터를 포장해서 전송을 한 후
- 데이터를 받는 컴퓨터에서 포장지를 하나 하나 까는 방식

1계층 네트워크 액세스 계층(Network Access Layer or Network Interface Layer)

OSI 7계층의 물리계층과 데이터 링크 계층에 해당한다.

물리적인 주소로 MAC을 사용한다.

LAN, 패킷망, 등에 사용된다.

2계층 인터넷 계층(Internet Layer)

OSI 7계층의 네트워크 계층에 해당한다.

통신 노드 간의 IP패킷을 전송하는 기능과 라우팅 기능을 담당한다.

프로토콜 – IP, ARP, RARP

3계층 전송 계층(Transport Layer)

OSI 7계층의 전송 계층에 해당한다.

통신 노드 간의 연결을 제어하고, 신뢰성 있는 데이터 전송을 담당한다.

프로토콜 – TCP, UDP

4계층 응용 계층(Application Layer)

OSI 7계층의 세션 계층, 표현 계층, 응용 계층에 해당한다.

TCP/UDP 기반의 응용 프로그램을 구현할 때 사용한다.

프로토콜 – FTP, HTTP, SSH

Application Layer

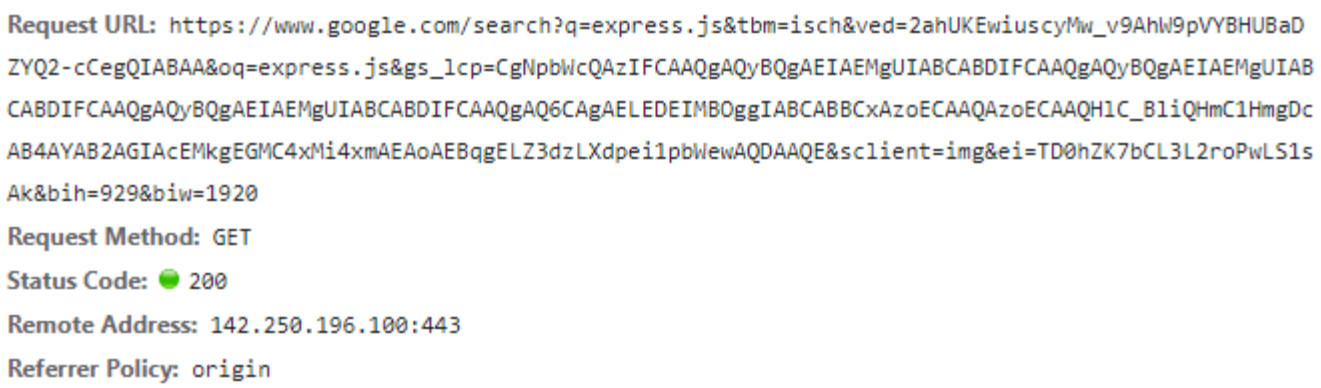
❖ 서버 개발자로써 우리가 코딩하는 부분은 **애플리케이션 계층**이다
- TCP/UDP 기반의 애플리케이션을 개발하는 것
- 즉, 통신을 하는 프로그램을 구현하는 것이 우리의 목표

- **서버 또한 프로그램이다, 그것도 통신을 하는 프로그램**

❖ Request는 **정형화된 양식**이 존재한다 (=Protocol)

- 양식에 맞춰서 요청을 하지 않으면 **서버는 응답을 해줄 수 없다**
- 웹에서 사용하는 양식 표준을 제정해 놓은 프로토콜이 **HTTP**이다

- 요청과 응답을 할 때는 HTTP에 맞춘 형식으로 보내야 한다.



Request HTTP

- **그렇지만 이러한 양식을 우리가 작성할 필요는 없다**
  - Why? 우리는 잘 만들어진 **HTTP 통신 프레임워크**(=Express)를 사용할 것이기 때문

Express

❖ **Node.js**를 위한 빠르고 개방적인 간결한 웹 프레임워크

- HTTP 프로토콜이 어떻게 구현되어 있는지 몰라도,
- 우리같은 사람들이 **추상화된 메서드**를 통해 HTTP 서버를 쉽게 구현할 수 있다
- 비슷한 프레임워크로는 아래와 같은 것들이 있다
  - Java – Spring
  - Python – Django, Flask, FastAPI

Express 시작하기

❖ 설치

1. 애플리케이션을 보관한 디렉토리를 생성하고 그 디렉토리를 작업 영역으로 설정

   WebStorm을 통해 빈 프로젝트를 생성한 후 프로젝트를 여는 과정

2. 명령어를 통해 해당 디렉토리를 Node.js 프로젝트로 설정

   ```
   $ npm init
   ```

3. 명령어를 통해 작업 디렉토리에 express 모듈을 다운로드

   ```
   $ npm install express –save
   ```

Express Hello world

❖ 예제

1. app.js파일을 디렉토리에 생성한 후 다음과 같은 코드를 작성후 저장

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
    res.send('Hello World!')
})

app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

2. 아래의 명령어를 터미널에 입력하여 프로그램을 실행

   $ node app.js

3. 웹 브라우저 주소에 localhost:3000을 입력하여 접속해보자

Express 기본 라우팅

- ❖ 라우팅은 **URL과 메서드에 대한 요청에 응답하는 방법을 결정하는 것**
- ❖ 라우팅 메서드는 아래와 같은 구조를 가짐

```
app.METHOD(PATH, HANDLER)
```

- app은 express의 인스턴스
- METHOD는 HTTP 요청 메서드
- PATH는 서버에서의 경로
- HANDLER는 라우트가 일치할 때 실행되는 Callback 함수

- ❖ 예제

1. app.js에 다음과 같은 코드를 추가해보자

```
app.get('/myinfo', function (req, res) {
    res.send('안녕하세요 {자기 이름}입니다.');
});
```

2. localhost:3000/myinfo로 접속해보자

Express 응답 메서드

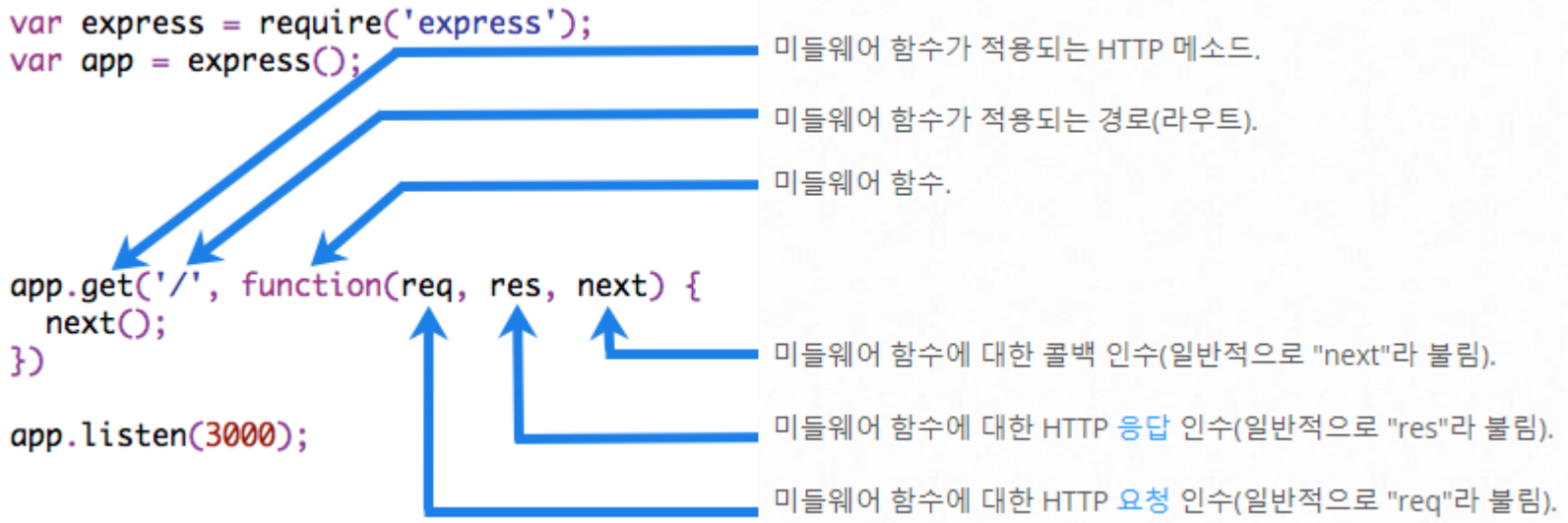
❖ Express는 다양한 응답 메서드를 지원함

- res.download()
  - 파일이 다운로드되도록 프롬프트합니다
- res.end()
  - 응답 프로세스를 종료합니다
- res.json()
  - JSON 응답을 전송합니다
- res.jsonp()
  - JSONP 지원을 통해 JSON 응답을 전송합니다
- res.redirect()
  - 요청의 경로를 재지정합니다
- res.render()
  - 뷰 템플릿을 렌더링합니다
- res.send()
  - 다양한 타입의 응답을 보냅니다
- res.sendFile()
  - Octet stream 파일을 보냅니다
- res.sendStatus()
  - 응답 상태 코드를 설정한 후 해당 코드를 문자열로 표현한 내용을 응답 본문으로서 전송합니다

Express 미들웨어

- ❖ 요청-응답 과정에서 다음 **미들웨어 함수에 대한 액세스 권한**을 갖는 함수
- ❖ 미들웨어는 다음과 같은 태스크를 수행할 수 있음
  - 모든 코드를 실행
  - 요청 및 응답 오브젝트에 대한 변경을 실행
  - 요청-응답 주기를 종료
  - 스택 내의 그 다음 미들웨어를 호출

- ❖ 현재 미들웨어가 요청-응답 과정을 종료하지 않는 경우는 next()를 호출하여 다음 미들웨어로 제어를 전달해야 함



Express 미들웨어

❖ 예제

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

var myLogger = function (req, res, next) {
    console.log('LOGGED');
    next();
};

app.use(myLogger);

app.get('/', function (req, res) {
    res.send('Hello World!');
});

app.listen(3000);
```

감사합니다


The World is
a Global Village and
the Peoples of the World
are One Human Family.
May We Strive for
Peace and Humanity
with the Spirit of
Global Cooperation
Society.