KB금융그룹 | 국민의 평생 금융파트너

2024년 상반기 K-디지털 트레이닝

네트워크 기초 및 서버 만들기

[KB] IT's Your Life

KB 국민은행

- HTTP 프로토콜
- 요청과 응답
- 중요한 헤더 정보 살펴보기
  - 요청 URL
  - 요청 메서드

요청 메서드	설명
GET 메서드	서버에서 정보를 가져올 때 사용합니다. 예를 들어 구글 웹 사이트 URL을 입력해 서버로 보내거나 웹 사이트에 있는 링크를 클릭하면 GET 요청이 서버로 전송되고, 서버는 해당 URL의 문서를 응답으로 반환합니다.
POST 메서드	서버에 데이터를 저장할 때 사용합니다. 예를 들어 회원 가입을 하거나 로그인할 때 사용자가 입력한 정보는 POST 메서드를 사용해 서버로 넘겨줍니다.
PUT 메서드	서버에 있는 데이터를 수정(업데이트)할 때 사용합니다. 예를 들어 서버에 이미 저장되어 있는 사용자 정보에서 일부를 수정할 때 PUT 메서드를 사용해 서버로 보냅니다.
DELETE 메서드	서버에서 데이터를 삭제할 때 사용합니다. 예를 들어 블로그 글이나 파일을 삭제할 때 DELETE 메서드를 사용해 삭제할 정보를 서버로 전송합니다.

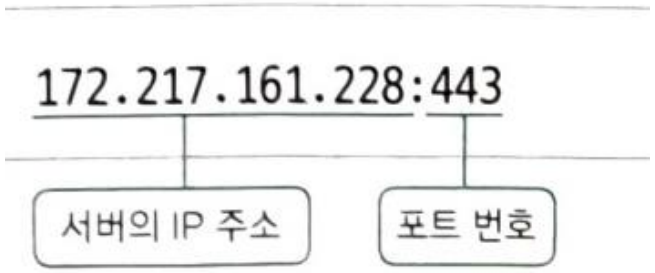
중요한 헤더 정보 살펴보기

- 상태 코드

코드	메시지	설명
1xx	Informational	계속 처리 중
2xx	Successful	요청 성공
200	OK	요청이 성공적으로 처리되었습니다.
201	Created	요청이 성공적으로 처리되어 새로운 자료가 생성되었습니다.
204	No Content	요청이 성공적으로 처리되었지만 응답으로 반환할 내용이 없습니다.
3xx	Redirection	다른 위치로 이동
301	Moved Permanently	요청한 데이터가 새 URL로 옮겨졌습니다.
4xx	Client Error	클라이언트 오류
400	Bad Request	클라이언트 요청이 잘못되었거나 유효하지 않습니다.
401	Unauthorized	권한이 없어 거절되었지만 인증을 다시 시도할 수 있습니다.
403	Forbidden	권한이 없어 거절되었고 인증을 시도하면 계속 거절됩니다.
404	Not Found	해당 데이터를 찾을 수 없습니다.
5xx	Server Error	서버 오류
500	Internal Server Error	서버에 요청을 처리하는 동안 오류가 발생했습니다.
503	Service Unavailable	요청한 서비스를 이용할 수 없습니다.

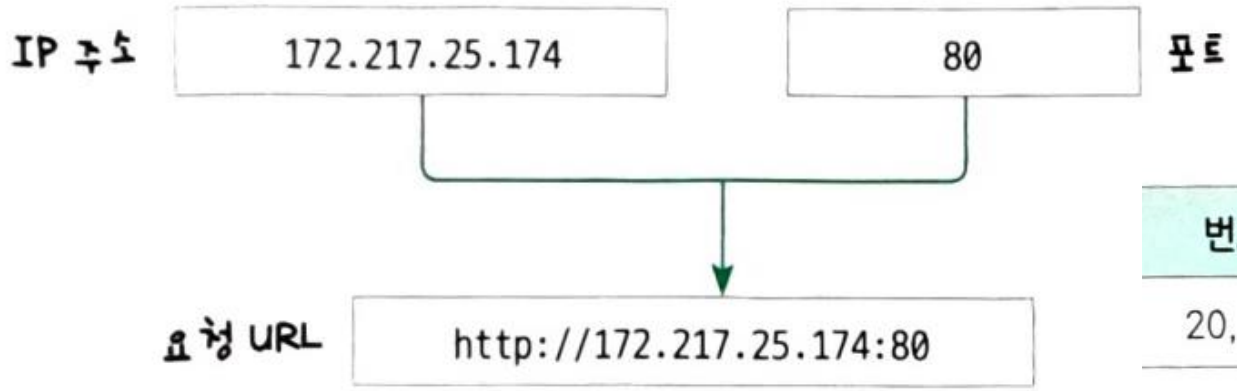
중요한 헤더 정보 살펴보기

- 원격 주소



IP 주소와 포트

- IP 주소
- 포트



IP 주소와 포트로 구성된 요청 URL

번호	기능
20, 21	FTP(파일 전송 프로토콜)
25	SMTP(이메일 발송)
53	DNS 서버
80	웹(HTTP)
110	POP3(이메일 수신)
443	HTTPS

서버 만들기

- http 모듈
- 서버 만들기 – createServer 함수

http.createServer([옵션][,콜백])

- 콜백: (req, res) => { }

서버 만들기

서버 실행하기 - listen

server.listen(포트[, 호스트][, 콜백])

- 호스트: ip가 여러 개인 경우 어디에 바인딩할지 지정

chapter04/server-1.js

```
const http = require('http');

const server = http.createServer((req, res) => {

  console.log('요청 발생');
});

server.listen(3000, () => {
  console.log('3000번 포트에서 서버 실행 중');
});
```

요청 객체 살펴보기

- 클라이언트가 보낸 요청 정보를 분석하여 객체로 저장
- request 또는 req 이름 사용
- http.createServer()에 요청 처리 함수 인자
  - (req, res) => { ... }
  - 클라이언트가 요청을 보낼 때마다 실행
  - 함수 인자
    - request
    - response

chapter04/server-2.js

```
const http = require('http');

const server = http.createServer((req, res) => {

  console.log(req);
});

server.listen(3000, () => {
  console.log('3000번 포트에서 서버 실행 중');
});
```

```
<ref *2> IncomingMessage {
  _readableState: ReadableState {
    highWaterMark: 16384,
    buffer: BufferList { head: null, tail: null, length: 0 },
    length: 0,
    pipes: [],
    awaitDrainWriters: null,
    [Symbol(kState)]: 1185840
  },
  _events: [Object: null prototype] {},
  _eventsCount: 0,
...
}
```

응답 객체 살펴보기

응답 헤더 만들기

- res.setHeader(이름, 값)
- res.writeHead(상태코드 [, 상태메시지][, 헤더])

```
res.setHeader("Content-Type", "text/plain")

res.writeHead(200, { "Content-Type" : "text/plain")
```

응답 객체 살펴보기

응답 본문 만들기

- 본문 내용 작성

res.write(내용[, 인코딩][, 콜백])

  - 여러 번 호출 가능

- 응답 종료

res.end(내용[, 인코딩][, 콜백])

```
res.write("내용1")
res.write("내용2")
res.write("내용3")
res.end()

또는
res.end("내용")
```

chapter04/server-3.js

```
const http = require('http');

const server = http.createServer((req, res) => {
  console.log(req.method);

  res.setHeader('Content-Type', 'text/pain'); // 응답 헤더
  res.write('Hello Node'); // 응답 본문
  res.end(); // 응답 종료
});

server.listen(3000, () => {
  console.log('http://localhost:3000 서버 실행 중');
});
```

http://localhost:3000 서버 실행 중
GET
GET
GET

HTML 페이지 서빙하기

- 웹 브라우저가 수신 데이터가 HTML 문서임을 알 수 있도록 헤더 구성
  - "Content-Type" : "text/html"
  - body는 html 태그가 있는 텍스트로 전송
  - html 파일을 res로 출력
  - res는 스트림 객체임
    - 파이프로 연결하여 간단하게 복사 가능

chapter04/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <h1>WELCOME TO NODE.JS</h1>
    <p>Node.js 세계에 오신 것을 환영합니다.</p>
  </body>
</html>
```

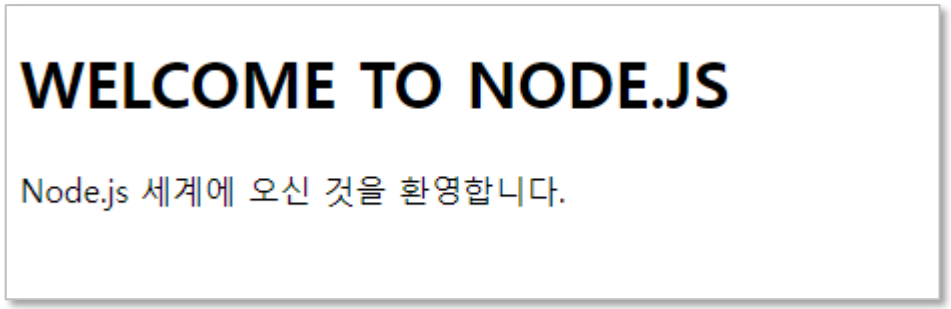
chapter04/server-4.js

```
const http = require('http');
const fs = require('fs');

const server = http.createServer((req, res) => {
  res.setHeader('Content-Type', 'text/html'); // 응답 헤더
  const readStream = fs.createReadStream(__dirname + '/index.html', 'utf8');

  readStream.pipe(res);
});

server.listen(3000, () => {
  console.log('http://localhost:3000 서버 실행 중');
});
```



라우팅 routing

- 클라이언트에서 들어오는 요청에 따라 그에 맞는 함수를 실행하는 것
- url과 요청 메서드(GET, POST, PUT, DELETE)에 따라 각각 다른 함수를 실행

chapter04/server-5.js

```
const http = require('http');

const server = http.createServer((req, res) => {
  // 요청 메서드와 URL 가져오기
  const { method, url } = req;
  res.setHeader('Content-Type', 'text/plain'); // 응답 헤더

  // URL과 메서드에 따라 응답을 다르게 처리
  if (method === 'GET' && url === '/home') {
    res.statusCode = 200;
    res.end('HOME');
  } else if (method === 'GET' && url === '/about') {
    res.statusCode = 200;
    res.end('ABOUT');
  } else {
    res.statusCode = 404;
    res.end('NOT FOUND');
  }
});

server.listen(3000, () => {
  console.log('http://localhost:3000 서버 실행 중');
});
```