컴퓨터 네트워크

4장

4.1 요청과 응답 이해하기

4.2 **REST API**와 라우팅

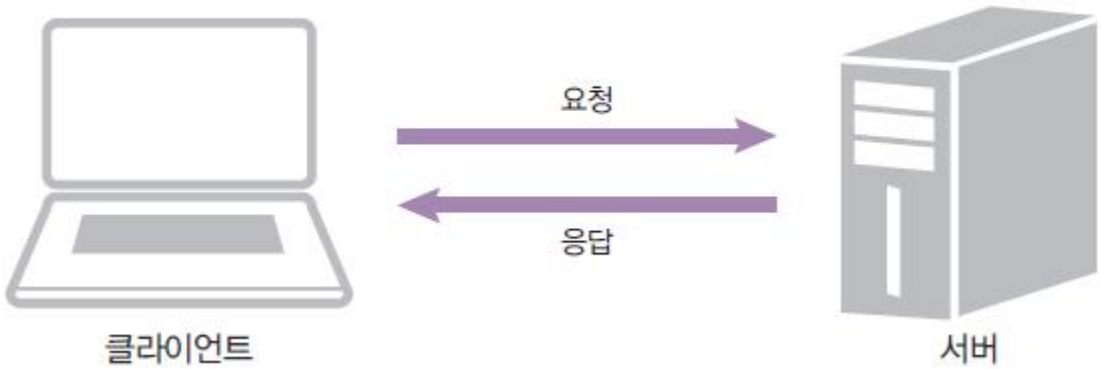
4.1 요청과 응답 이해하기

1. 서버와 클라이언트

서버와 클라이언트의 관계

- 클라이언트가 서버로 요청(request)을 보냄
- 서버는 요청을 처리
- 처리 후 클라이언트로 응답(response)을 보냄

그림 4-1 클라이언트와 서버의 관계



2. 노드로 http 서버 만들기

HTTP 모듈

- Node.js 내장 모듈
- 웹 서버와 클라이언트 생성 등 관련된 모든 기능 담당

개념	설명
요청	웹 서버에 보내는 모든 요청
응답	요청을 받은 웹 서버가 처리하는 작업
Http 모듈	http 웹 서버와 관련된 모든 기능을 담은 내장 모듈
Server 객체	웹 서버를 생성할 때 사용하는 객체
Response 객체	응답 메시지 작성 시 request 이벤트 리스너의 두 번째 매개변수로 전달되는 객체
Request 객체	응답 메시지 작성 시 request 이벤트 리스너의 첫 번째 매개변수로 전달되는 객체

2. 노드로 http 서버 만들기

server 객체

- createServer로 요청 이벤트에 대기
- req 객체는 요청에 관한 정보가, res 객체는 응답에 관한 정보가 담겨 있음

listen(포트) 메서드로 특정 포트에 연결

createServer.js

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

http.createServer((req, res) => {
  // 여기에 어떻게 응답할지 적습니다.
});
```

```
  // http모듈을 추출합니다.
const http = require('http')

// 웹 서버를 생성합니다.
const server = http.createServer()

// 웹 서버를 3001 포트로 실행합니다.
server.listen(3001, function(){
  console.log('3001번 포트로 서버가 실행되었습니다.')
})
```

2. 노드로 http 서버 만들기

» server 객체 메서드

메서드	설명
listen(port, callback)	서버를 첫번째 매개변수의 포트로 실행
close(callback)	서버 종료

2. 노드로 http 서버 만들기

response 객체

- response 객체를 통해 응답 메시지 작성
- write로 응답 내용을 적고
- end로 응답 마무리(내용을 넣어도 됨)

메서드	설명
writeHead(statusCode, statusMessage, headers)	응답 헤더 작성 / statusCode: 200, 404 등 한 번만 호출 가능
write(chunk, encoding, callback)	응답의 본문을 작성하는 부분
end(data, encoding, callback)	모든 응답 헤더와 본문이 전송되었음을 서버에 알림.

2. 노드로 http 서버 만들기

- res 메서드로 응답 보냄
  - write로 응답 내용을 적고
  - end로 응답 마무리(내용을 넣어도 됨)
- listen(포트) 메서드로 특정 포트에 연결

server1.js

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

http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
  res.write('<h1>Hello Node!</h1>');
  res.end('<p>Hello Server!</p>');
})
  .listen(8080, () => { // 서버 연결
    console.log('8080번 포트에서 서버 대기 중입니다!');
  });
```

3. 상태 코드(Status Code)

» writeHead 메서드에 첫 번째 인수로 넣은 값

- 요청이 성공했는지 실패했는지를 알려줌
- 2XX: 성공을 알리는 상태 코드입니다. 대표적으로 200(성공), 201(작성됨)이 많이 사용됩니다.
- 3XX: 리다이렉션(다른 페이지로 이동)을 알리는 상태 코드입니다. 어떤 주소를 입력했는데 다른 주소의 페이지로 넘어갈 때 이 코드가 사용됩니다. 대표적으로 301(영구 이동), 302(임시 이동)가 있습니다.
- 4XX: 요청 오류를 나타냅니다. 요청 자체에 오류가 있을 때 표시됩니다. 대표적으로 401(권한 없음), 403(금지됨), 404(찾을 수 없음)가 있습니다.
- 5XX: 서버 오류를 나타냅니다. 요청은 제대로 왔지만 서버에 오류가 생겼을 때 발생합니다. 이 오류가 뜨지 않게 주의해서 프로그래밍해야 합니다. 이 오류를 클라이언트로 res.writeHead로 직접 보내는 경우는 없고, 예기치 못한 에러 발생 시 서버가 알아서 5XX대 코드를 보냅니다. 500(내부 서버 오류), 502(불량 게이트웨이), 503(서비스를 사용할 수 없음)이 자주 사용됩니다.

4. 8080 포트로 접속하기

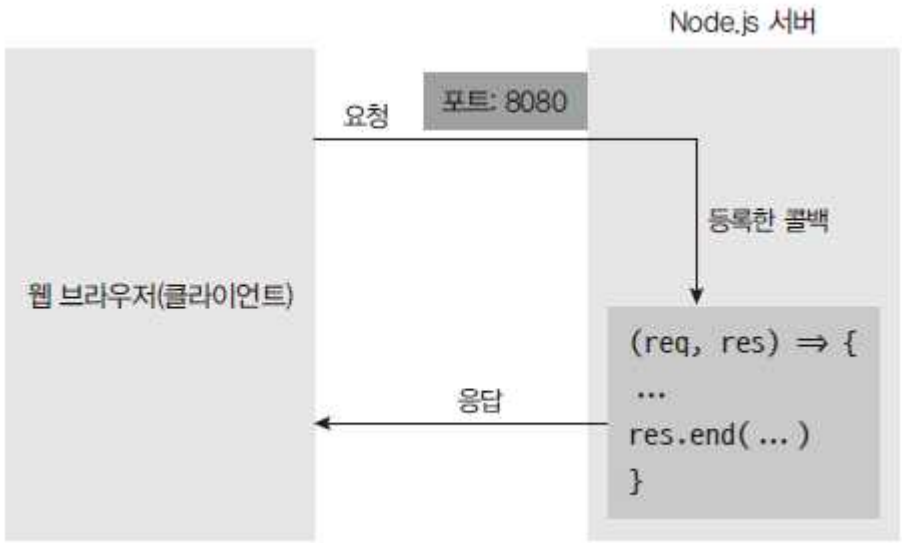
» 스크립트를 실행하면 8080 포트에 연결됨

콘솔

```
$ node server1
8080번 포트에서 서버 대기 중입니다!
```

» localhost:8080 또는 http://127.0.0.1:8080에 접속

그림 4-2 서버 실행 화면

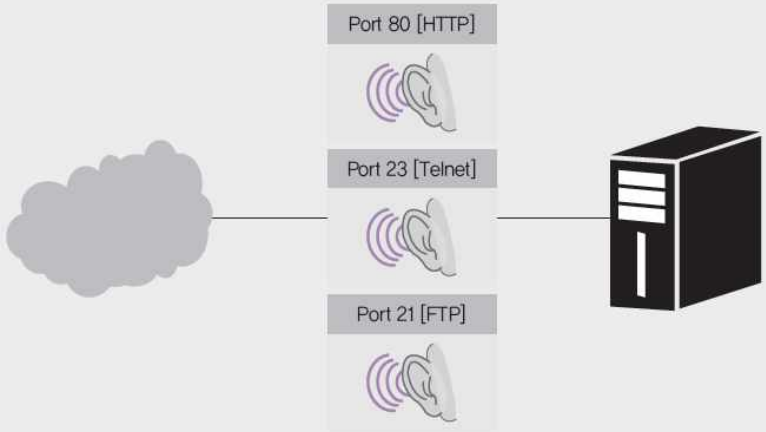


5. localhost와 포트

- localhost는 컴퓨터 내부 주소
  - 외부에서는 접근 불가능

- 포트는 서버 내에서 프로세스를 구분하는 번호
  - 기본적으로 http 서버는 80번 포트 사용(생략가능, https는 443)
  - 예) www.gilbut.com:80 -> www.github.com
  - 다른 포트로 데이터베이스나 다른 서버 동시에 연결 가능

그림 4-4 IP와 포트



6. 이벤트 리스너 붙이기

» listening과 error 이벤트를 붙일 수 있음.

- on 메서드를 이용해 server에 이벤트 연결

server1-1.js

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

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
  res.write('<h1>Hello Node!</h1>');
  res.end('<p>Hello Server!</p>');
});
server.listen(8080);

server.on('listening', () => {
  console.log('8080번 포트에서 서버 대기 중입니다!');
});
server.on('error', (error) => {
  console.error(error);
});
```

6. 이벤트 리스너 붙이기

Server 객체의 이벤트

이벤트	설명
request	클라이언트가 요청할 때 발생하는 이벤트
connection	클라이언트가 접속할 때 발생하는 이벤트
close	서버가 종료될 때 발생하는 이벤트
checkContinue	클라이언트가 지속적인 연결을 하고 있을 때 발생하는 이벤트
clientError	클라이언트에서 오류가 발생할 때 발생하는 이벤트

- createServer() 메서드의 매개변수로 request 이벤트를 on 없이 정의 가능

7. 한 번에 여러 개의 서버 실행하기

» createServer를 여러 번 호출하면 됨.

- 단, 두 서버의 포트를 다르게 지정해야 함.
- 같게 지정하면 EADDRINUSE 에러 발생

server1-2.js

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

http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
  res.write('<h1>Hello Node!</h1>');
  res.end('<p>Hello Server!</p>');
})
  .listen(8080, () => { // 서버 연결
    console.log('8080번 포트에서 서버 대기 중입니다!');
  });

http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
  res.write('<h1>Hello Node!</h1>');
  res.end('<p>Hello Server!</p>');
})
  .listen(8081, () => { // 서버 연결
    console.log('8081번 포트에서 서버 대기 중입니다!');
  });
```

8. html 읽어서 전송하기

» write와 end에 문자열을 넣는 것은 비효율적

- fs 모듈로 html을 읽어서 전송하자
- write가 버퍼도 전송 가능

server2.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Node.js 웹 서버</title>
</head>
<body>
    <h1>Node.js 웹 서버</h1>
    <p>만들 준비되셨나요?</p>
</body>
</html>
```

server2.js

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

http.createServer(async (req, res) => {
  try {
    const data = await fs.readFile('./server2.html');
    res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
    res.end(data);
  } catch (err) {
    console.error(err);
    res.writeHead(500, { 'Content-Type': 'text/plain; charset=utf-8' });
    res.end(err.message);
  }
})
  .listen(8081, () => {
    console.log('8081번 포트에서 서버 대기 중입니다!');
  });
```

8. html 읽어서 전송하기

fs(File System) 모듈

- fs 모듈을 이용하여 html 뿐만 아니라 이미지, mp3 파일 등 제공

Content Type	설명
text/plain	기본적인 텍스트
text/html	HTML 문서
text/css	CSS 문서
text/xml	XML 문서
image/jpeg	JPG/JPEG 파일
image/png	PNG파일
video/mpeg	MPEG 비디오 파일
audio/mp3	MP3 파일

8. html 읽어서 전송하기

fs(File System) 모듈

- fs 모듈을 이용하여 html 뿐만 아니라 이미지, mp3 파일 등 제공

```
var http = require('http');
var fs = require('fs');
var app = http.createServer(function(req, res){
    fs.readFile('ditto.png', (error, data)=>{
        res.writeHead(200, {'Content':'image/png'});
        res.end(data);
    })

});
app.listen(3000);
app.on('error', (error)=>{
    console.error(error);
});

```

9. server2 실행하기

» 포트 번호를 8081로 바꿈

- server1.js를 종료했다면 8080번 포트를 계속 써도 됨
- 종료하지 않은 경우 같은 포트를 쓰면 충돌이 나 에러 발생

콘솔

```
$ node server2
8081번 포트에서 서버 대기 중입니다!
```

4.2 REST API와 라우팅

1. REST API

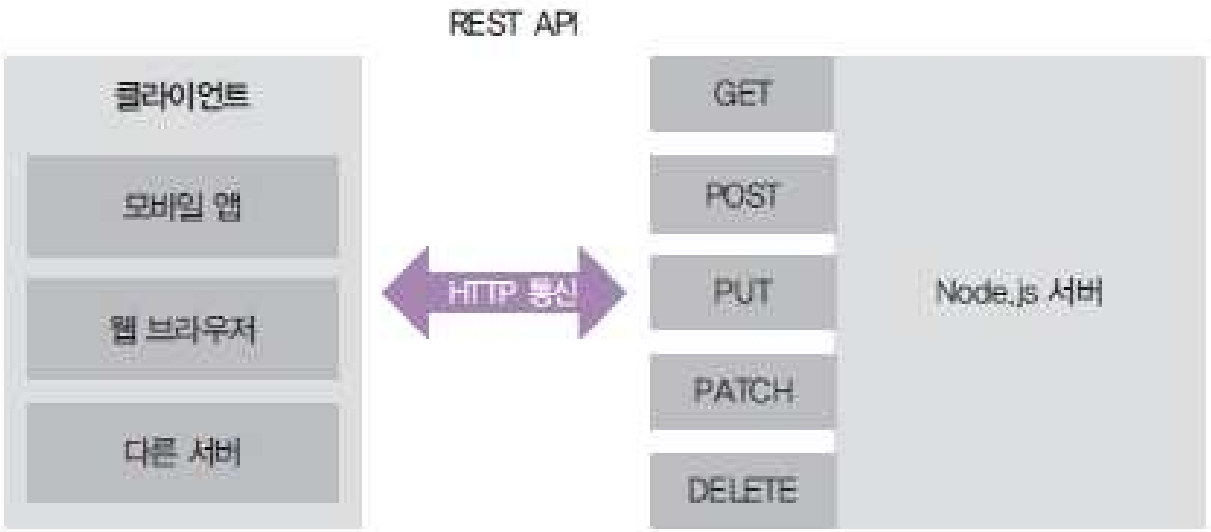
- 서버에 요청을 보낼 때는 주소를 통해 요청의 내용을 표현
  - /index.html이면 index.html을 보내달라는 뜻
  - 항상 html을 요구할 필요는 없음
  - 서버가 이해하기 쉬운 주소가 좋음
- REST API(Representational State Transfer)
  - 서버의 자원을 정의하고 자원에 대한 주소를 지정하는 방법
  - /user이면 사용자 정보에 관한 정보를 요청하는 것
  - /post면 게시글에 관련된 자원을 요청하는 것
- HTTP 요청 메서드
  - GET: 서버 자원을 가져오려고 할 때 사용
  - POST: 서버에 자원을 새로 등록하고자 할 때 사용(또는 뭘 써야할 지 애매할 때)
  - PUT: 서버의 자원을 요청에 들어있는 자원으로 치환하고자할 때 사용
  - PATCH: 서버 자원의 일부만 수정하고자 할 때 사용
  - DELETE: 서버의 자원을 삭제하고자할 때 사용

2. HTTP 프로토콜

» 클라이언트가 누구든 서버와 HTTP 프로토콜로 소통 가능

- iOS, 안드로이드, 웹이 모두 같은 주소로 요청 보낼 수 있음
- 서버와 클라이언트의 분리

그림 4-15 REST API



2. HTTP 프로토콜

RESTful

- REST API를 사용한 주소 체계를 이용하는 서버
- GET/user는 사용자를 조회하는 요청, POST/user는 사용자를 등록하는 요청

표 4-1 서버 주소 구조

HTTP 메서드	주소	역할
GET	/	restFront.html 파일 제공
GET	/about	about.html 파일 제공
GET	/users	사용자 목록 제공
GET	기타	기타 정적 파일 제공
POST	/users	사용자 등록
PUT	/users/사용자id	해당 id의 사용자 수정
DELETE	/users/사용자id	해당 id의 사용자 제거

수고하셨습니다♥