1. http 모듈로 서버 만들기

❖ server3.js

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

const server = http.createServer(async (req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
    res.write('<h1>Node.js로 서버만들기</h1>');
    res.end(`<p>요청된 경로 : ${req.url}</p>`)
}).listen(8080);

/* Listening Event Listener */
server.on('listening', () => {
    console.log('8080포트로 서버가 시작되었습니다.');
});

/* Error Event Listener */
server.on('error', () => {
    console.error(error);
});
```

1. http 모듈로 서버 만들기

여기서 잠깐

포트번호는 0~65535번까지 사용할 수 있고 종류는 아래와 같습니다.

- Well-known ports : 0~1023번까지이며 이미 사용되고 있는 포트이기 때문에 따로 사용할 수 없습니다.
- Registered ports : 1024~49151번까지이며 벤더가 할당 받아 사용하는 포트입니다.
- Dynamic ports : 49152~65535번까지이며 주로 시스템에서 쓰입니다.

서버를 위해서는 주로 80포트 혹은 8080포트를 이용합니다.

여기서 잠깐

http 상태 코드(요청 코드)

http 요청이 성공적으로 완료되었는지, 실패했는지 알려주는 기능을 합니다. 3자리 숫자로 이루어져 있고 첫 번째 자리는 1~5까지이며 숫자별로 알려주는 상태가 다릅니다.

- 100번대 : 정보 / 서버가 요청을 받았고 클라이언트는 작업을 계속 진행해도 된다는 의미입니다.
- 200번대 : 성공 / 요청을 성공적으로 받았고 수용했다는 코드입니다.
- 300번대 : 리다이렉션 / 요청을 완료하기 위해서 추가적인 작업이 필요하다는 의미입니다.
- 400번대 : 클라이언트 서버 오류를 의미합니다.
- 500번대 : 서버 오류를 의미합니다.

그리고 주로 사용하는 코드는 다음과 같습니다.

- 200 : 요청이 성공했습니다.
- 204 : 요청은 성공했으나 제공할 내용이 없습니다.
- 304 : 이전 요청과 동일합니다.
- 400 : 클라이언트 요청 오류입니다.
- 401 : 요청을 위한 권한을 요구합니다.
- 403 : 요청이 서버에 의해 거부되었습니다.
- 404 : 요청한 URL을 찾을 수 없습니다.
- 500 : 서버에 오류가 발생하여 응답이 불가능합니다.

2. request 객체의 속성과 response 객체의 메서드

❖ request 객체가 제공하는 속성

- req 인자
- 클라이언트(예:브라우저)가 서버로 요청시 전달하는 정보 포함

종류	속성	설명
req	req.body	POST 방식으로 들어오는 요청 정보 파라미터를 가집니다.
	req.query	GET 방식으로 들어오는 요청 쿼리 스트링 파라미터를 가집니다.
	req.params	개발자가 붙인 라우터 파라미터 정보를 가집니다.
	req.headers	HTTP 헤더 정보를 가집니다.
	req.route	현재 라우트에 대한 정보를 가집니다.
	req.cookies	클라이언트가 전달한 쿠키 값을 가집니다.
	req.accepts	인자에 타입을 넣고 클라이언트가 해당 타입을 받을 수 있는지 확인합니다.
	req.ip	클라이언트의 ip 주소 값을 가집니다.
	req.path	클라이언트가 요청한 경로를 가집니다.
	req.host	요청 호스트 이름을 반환합니다.
	req.xhr Ajax	요청 시 true를 반환합니다.
	req.protocol	현재 요청의 프로토콜(http, https 등)입니다.
	req.secure	현재 요청이 보안된 요청이면 true를 반환합니다.
	req.url	url 경로와 쿼리스트링을 반환합니다.

2. request 객체의 속성과 response 객체의 메서드

❖ response 객체가 제공하는 속성, 메서드

- res 인자
- 웹서버가 클라이언트로 응답하는 정보를 생성, 설정할 때 사용하는 메

res	res.send	클라이언트에게 응답을 보냅니다.
	res.sendFile	인자로 넣은 경로의 파일을 클라이언트에 전송합니다.
	res.json	클라이언트에게 json 형태의 응답을 보냅니다.
	res.render	템플릿 엔진을 사용하여 뷰를 렌더링합니다.
	res.locals	뷰를 렌더링하는 기본 문맥을 포함합니다.
	res.end	인자로 넣은 응답을 마지막으로 보내고 응답을 종료합니다.
	res.status	HTTP 응답 토드를 설정합니다.
	res.set	응답 헤더를 설정합니다.
	res.cookie	클라이언트에 저장될 쿠키를 설정합니다.
	res.redirect	인자로 넣은 URL으로 redirect합니다(기본 응답 값은 302입니다).
	res.type	헤더의 Content-Type를 설정합니다.

3. http + fs + html

❖ fs_test.js

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

http.createServer(async (req, res) => {
    try {
        const f = await fs.readFile('./fs_test.html');
        res.writeHead(200, { 'Content-Type': 'text.html; charset=utf-8' }); // 200이면 요청 성공
        res.end(f); // 요청 종료
    } catch (err) { // 에러 처리
        console.error(err); // 요청에 실패했을 경우 에러 출력
        res.writeHead(500, { 'Content-Type': 'text.html; charset=utf-8' }); // 500이면 서버에 오류 발생
        res.end(err.message); // 에러 메세지와 함께 요청 종료
    }
})
.listen(8080, () => {
    console.log('8080포트에서 서버 연결 중 ..')
});
```

3. http + fs + html

❖ fs_test.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>Node.js로 서버만들기</h1>
    <p>3장 http모듈 공부중입니다.</p>
  </body>
</html>
```