6. Introduction to node.js (2/2)

2023학년 2학기 웹응용프로그래밍

권 동 현

Contents

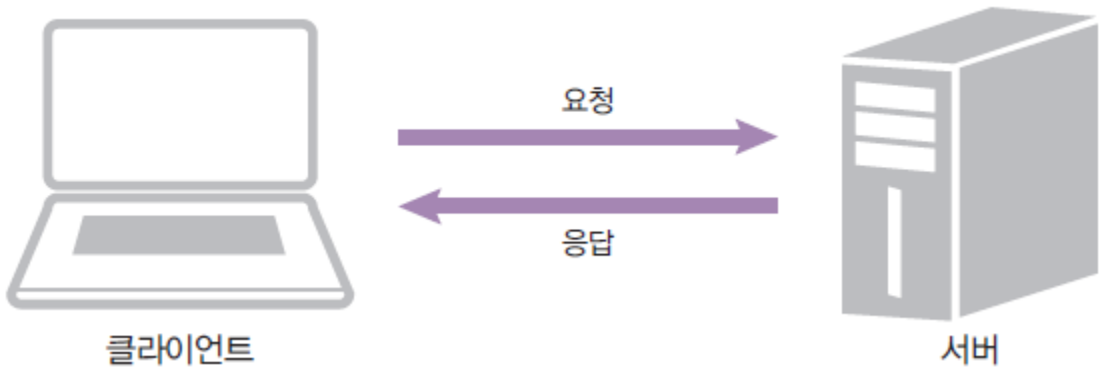
Request and Response

CSLAB

Server and Client

- The relationship between the server and the client
  - The client sends a request to the server
  - The server processes the request
  - After processing, server sends a response back to the client

그림 4-1 클라이언트와 서버의 관계



http server using node.js

- A node.js server responding to HTTP requests
  - ‘createServer’ to listen for request events
  - ‘req’ object contains information about the request
  - ‘res’ object contains information about the response
- The response is sent using the ‘res’ methods
  - ‘write’ to write response content
  - ‘end’ to finish the reponse (you can also include content)
- listen (port) method is used to connect to a specific port.

createServer.js

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

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

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!');
  });
```

http server using node.js

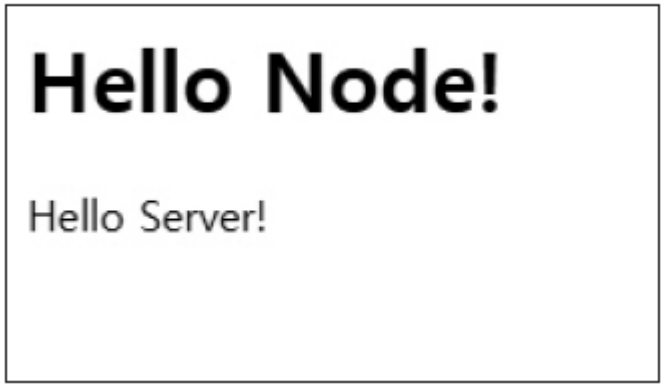
- When you run the script, it connects to port 8080

콘솔

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

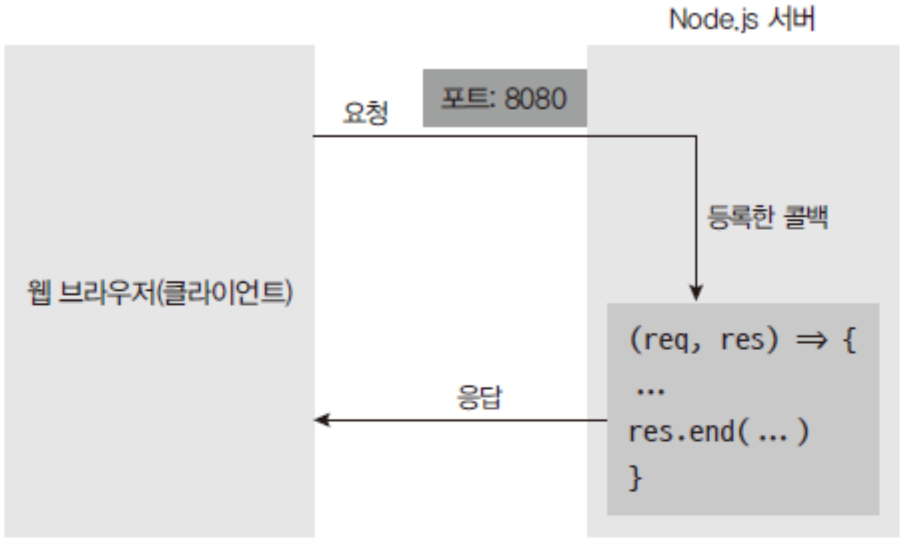
- You can access it through either localhost:8080 or http://127.0.0.1:8080

그림 4-2 서버 실행 화면



Hello Node!

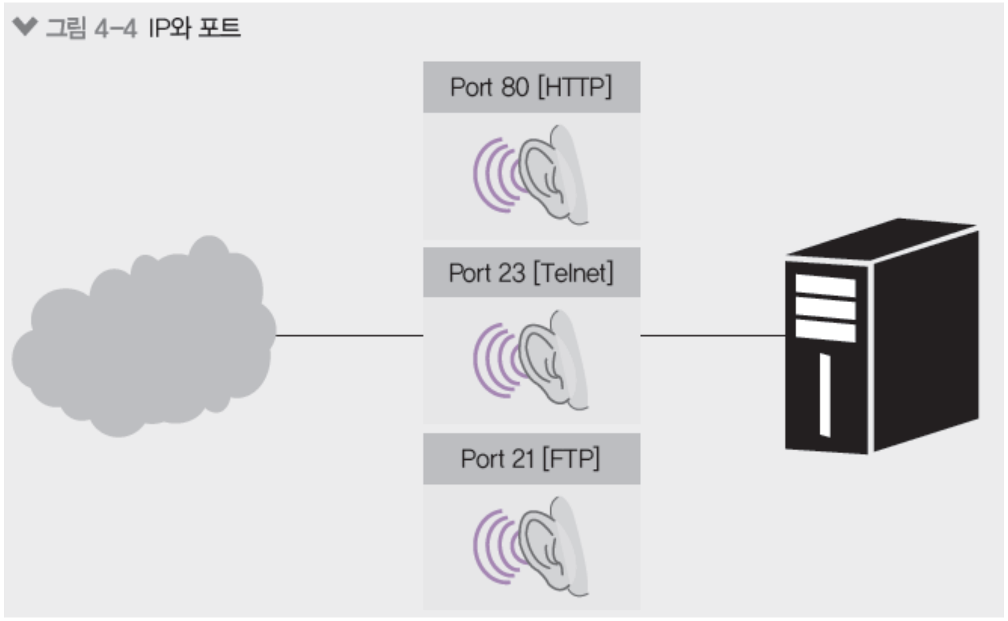
Hello Server!



localhost and port

- 'localhost' is an internal address
  - cannot be accessed from outside the computer

- Ports are numbers that help identify processes within a server
  - By default, HTTP servers use port 80 (can be omitted), while HTTPS uses port 443.
    - www.gilbut.com:80 -> www.github.com
  - Different ports can be used to connect to databases or other servers simultaneously.



Adding Event Listener

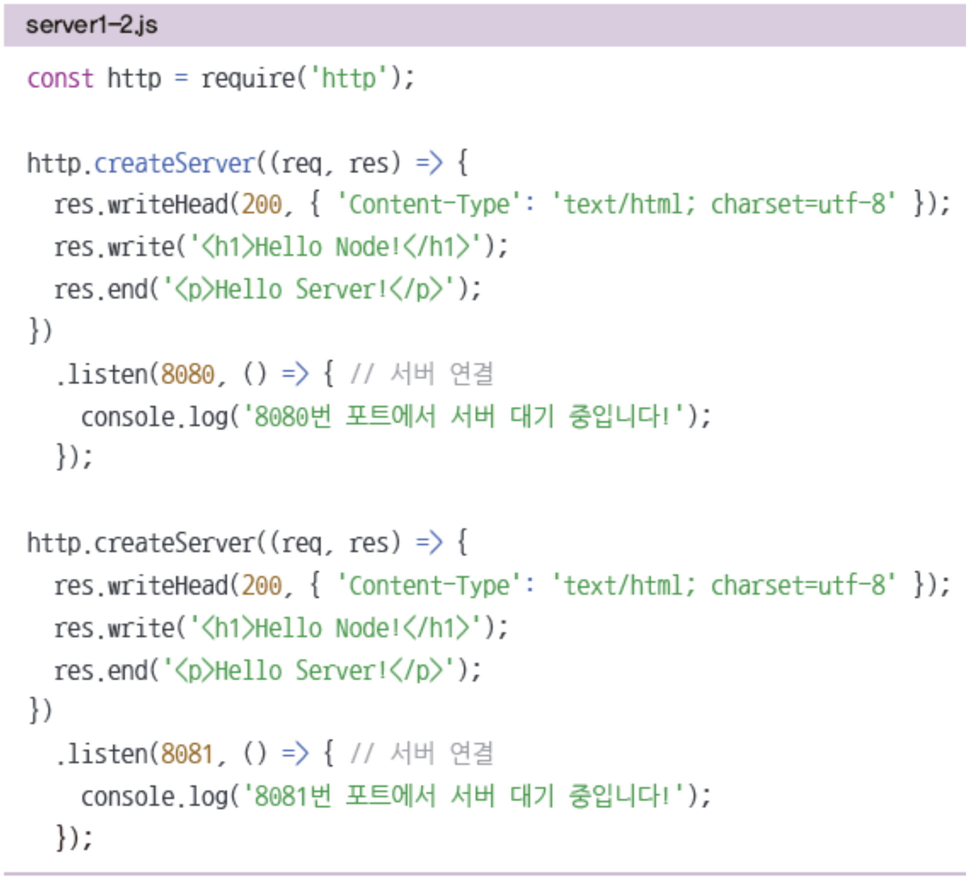
- Attach "listening" and "error" event listeners to a node.js server.

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);
});
```

Multiple http server using node.js

- Call createServer multiple times
  - Each server must be bound to a different port
  - Otherwise, you will encounter the EADDRINUSE error, indicating that the address is already in use.



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!');
  });
```

Send HTML file

- Reading the HTML content from a file using the fs module is a more efficient way to serve HTML content

server2.html

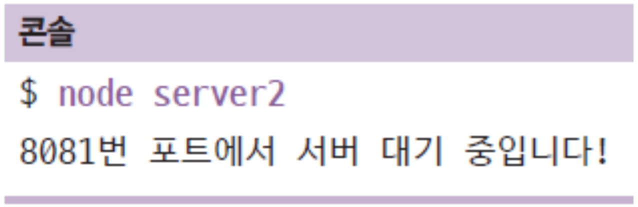
```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Node.js 웹 서버</title>
</head>
<body>
    <h1>Node.js 웹 서버</h1>
    <p>만들 준비되셨나요?</p>
</body>
</html>
```

server2.js

```javascript
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!');
  });
```

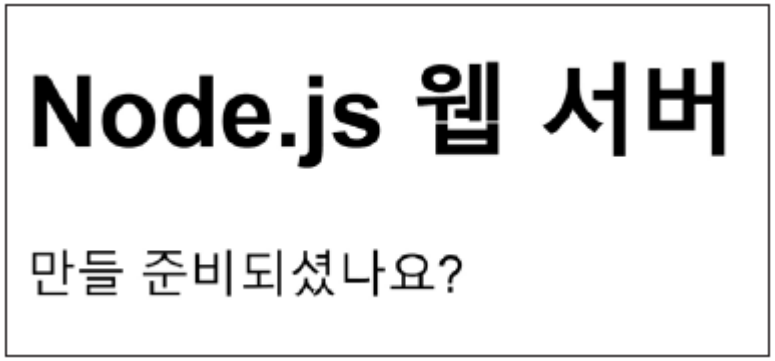
Send HTML file

- Change the port number to 8081
  - If server1.js is stopped, 8080 port can be used
  - Otherwise, it generate ‘EADDRINUSE” eror



콘솔

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



Node.js 웹 서버

만들 준비되셨나요?

REST API and Routing

REST API

- When sending requests to the server, the address expresses the content of the request.
  - For example, "/index.html" means requesting the "index.html" file. It's not always necessary to request HTML
  - using addresses that are easy for the server to understand is beneficial.

- REST API(Representational State Transfer)
  - a method of defining server resources and specifying addresses for those resources.
  - For instance, "/user" might be a request for information about users, and "/post" could be a request related to posts.

- HTTP Request Methods
  - GET: Used when attempting to retrieve server resources.
  - POST: Used when registering new resources on the server or when the desired method is unclear.
  - PUT: Used when wanting to replace server resources with the resources in the request.
  - PATCH: Used when wanting to modify only a part of a server resource.
  - DELETE: Used when wanting to delete a server resource.

HTTP protocol

- Regardless of whether they are using iOS, Android, or the web, can communicate with the server using the HTTP protocol
  - They can send requests to the same address.
  - the separation between the server and the client

- RESTful
  - A server that uses the REST API-based addressing system.
  - For example, a GET request to "/user" is used to retrieve user information, while a POST request to "/user" is used to register a new user.

그림 4-15 REST API

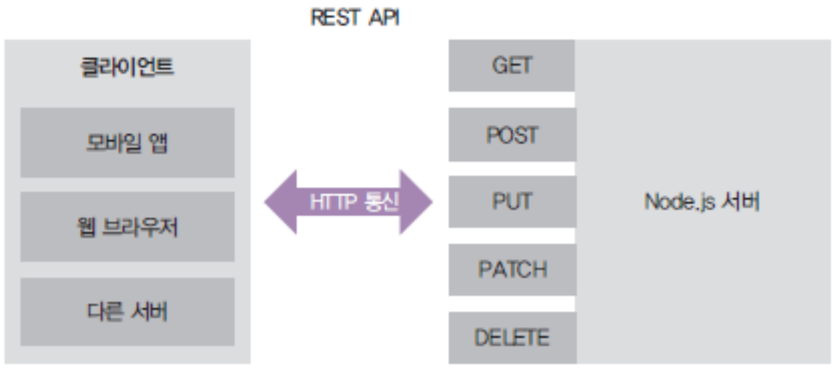
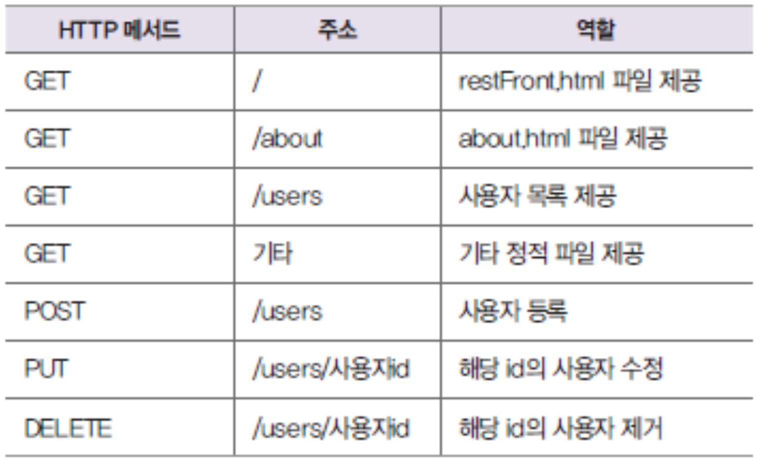


표 4-1 서버 주소 구조

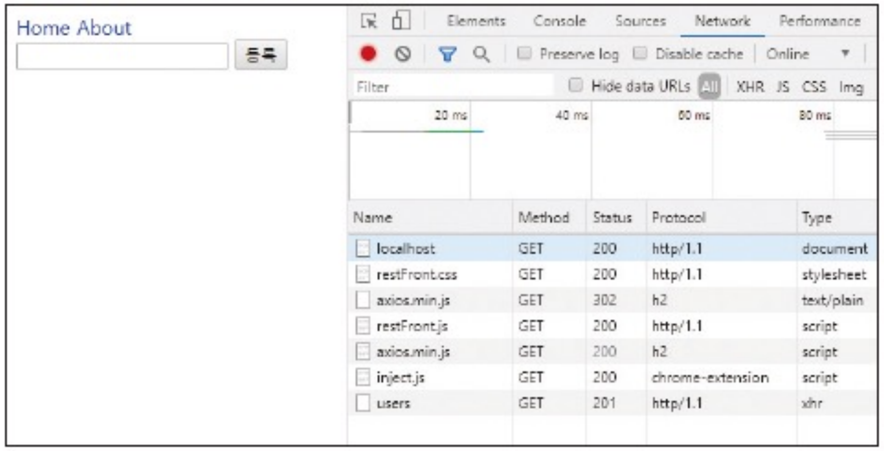


HTTP 메서드	주소	역할
GET	/	restFront.html 파일 제공
GET	/about	about.html 파일 제공
GET	/users	사용자 목록 제공
GET	기타	기타 정적 파일 제공
POST	/users	사용자 등록
PUT	/users/사용자id	해당 id의 사용자 수정
DELETE	/users/사용자id	해당 id의 사용자 제거

REST server example

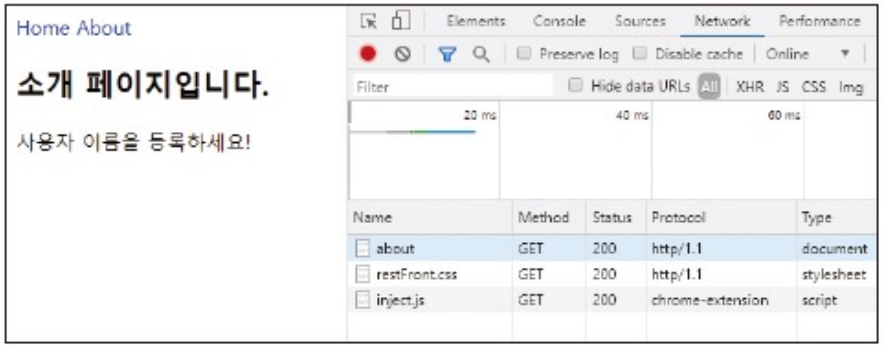
- Reference
  - https://github.com/ZeroCho/nodejs-book/tree/master/ch4/4.2
- restServer.js
  - handles GET, POST, PUT, and DELETE methods for managing data related to users and responds with a 404 NOT FOUND error for non-existing routes.
- restFront.js, restFront.css, restFront.html, about.html
  - client-side code
- To execute the server
  - $ node restServer.js

그림 4-7 Home 클릭 시


Home About
등록
Elements Console Sources Network Performance
Preserve log Disable cache Online
Filter Hide data URLs All XHR JS CSS Img
20 ms 40 ms 60 ms 80 ms

Name	Method	Status	Protocol	Type
localhost	GET	200	http/1.1	document
restFront.css	GET	200	http/1.1	stylesheet
axios.min.js	GET	302	h2	text/plain
restFront.js	GET	200	http/1.1	script
axios.min.js	GET	200	h2	script
inject.js	GET	200	chrome-extension	script
users	GET	201	http/1.1	xhr

그림 4-8 About 클릭 시


Home About
Elements Console Sources Network Performance
Preserve log Disable cache Online
Filter Hide data URLs All XHR JS CSS Img
20 ms 40 ms 60 ms

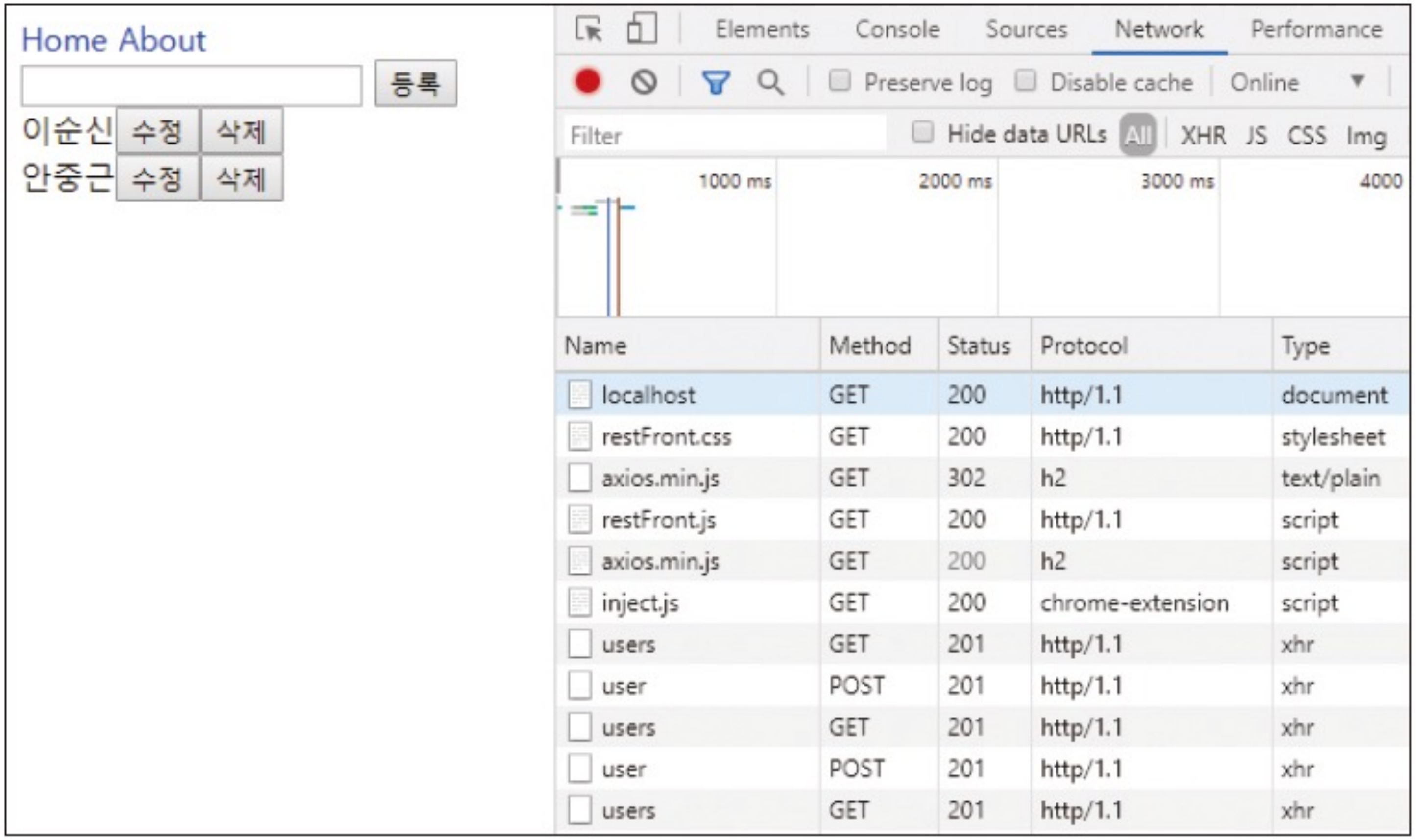
소개 페이지입니다.

사용자 이름을 등록하세요!

Name	Method	Status	Protocol	Type
about	GET	200	http/1.1	document
restFront.css	GET	200	http/1.1	stylesheet
inject.js	GET	200	chrome-extension	script

REST server example

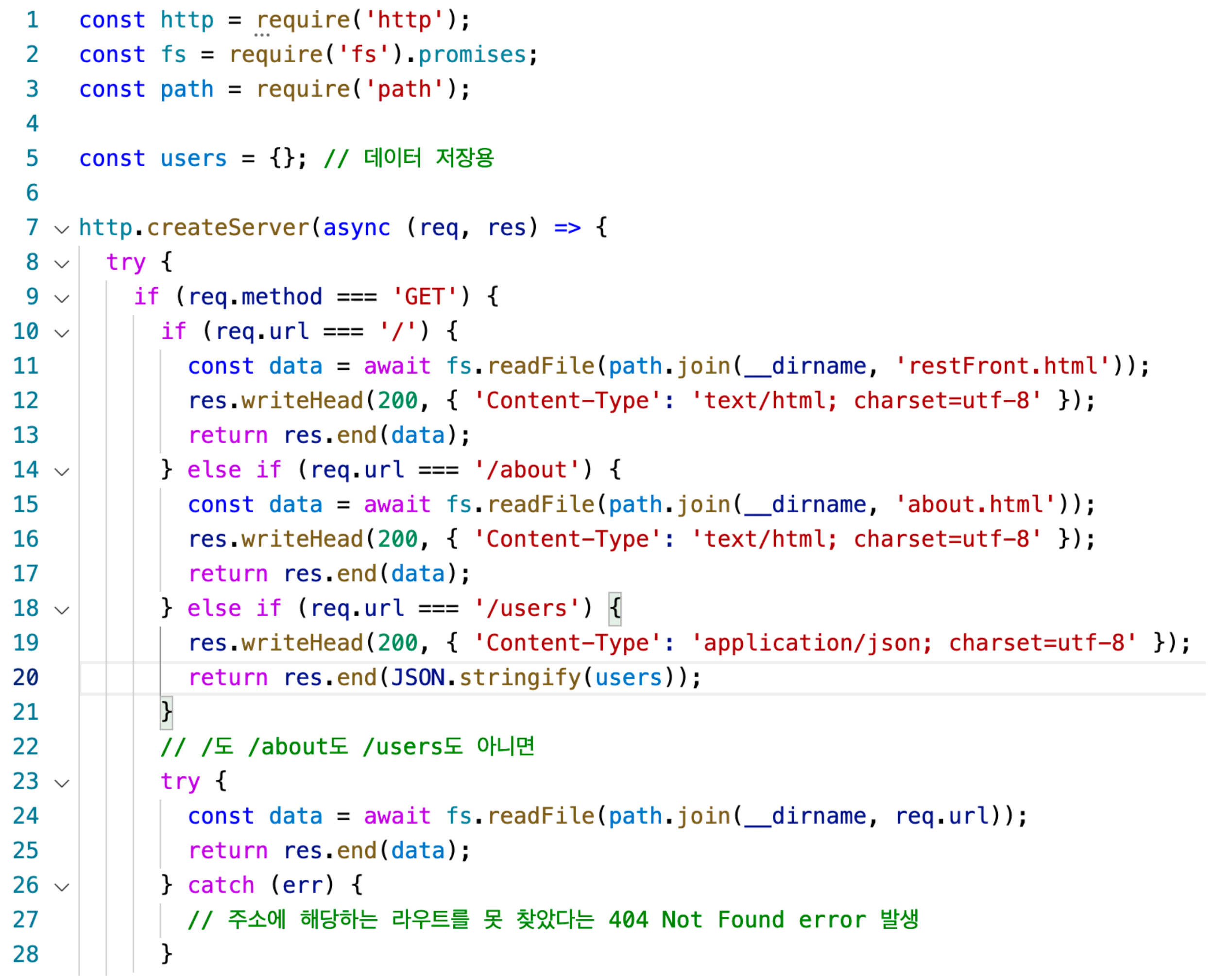
- real-time monitor request details in the "Network" tab of the developer tools (F12)
  - "Name" represents the request address, "Method" indicates the request method, "Status" shows the HTTP response code, "Protocol" specifies the HTTP protocol, and "Type" denotes the type of request (xhr indicates AJAX requests).



Name	Method	Status	Protocol	Type
localhost	GET	200	http/1.1	document
restFront.css	GET	200	http/1.1	stylesheet
axios.min.js	GET	302	h2	text/plain
restFront.js	GET	200	http/1.1	script
axios.min.js	GET	200	h2	script
inject.js	GET	200	chrome-extension	script
users	GET	201	http/1.1	xhr
user	POST	201	http/1.1	xhr
users	GET	201	http/1.1	xhr
user	POST	201	http/1.1	xhr
users	GET	201	http/1.1	xhr

REST server example

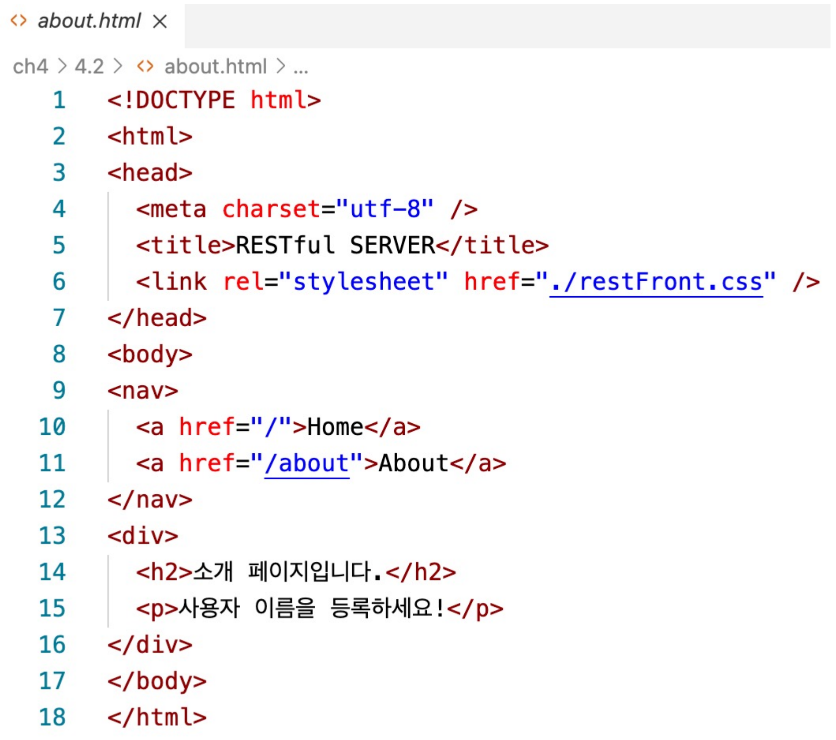
- line 9
  - Block for handling GET method.
- line 10~17
  - Handling requests for "/", "/about" by transmitting HTML files.
- line 18~21
  - Handling requests for "/users" by converting the "users" variable from line 5 to JSON and sending it.
- line 23~28
  - Handling requests for CSS or JS files.



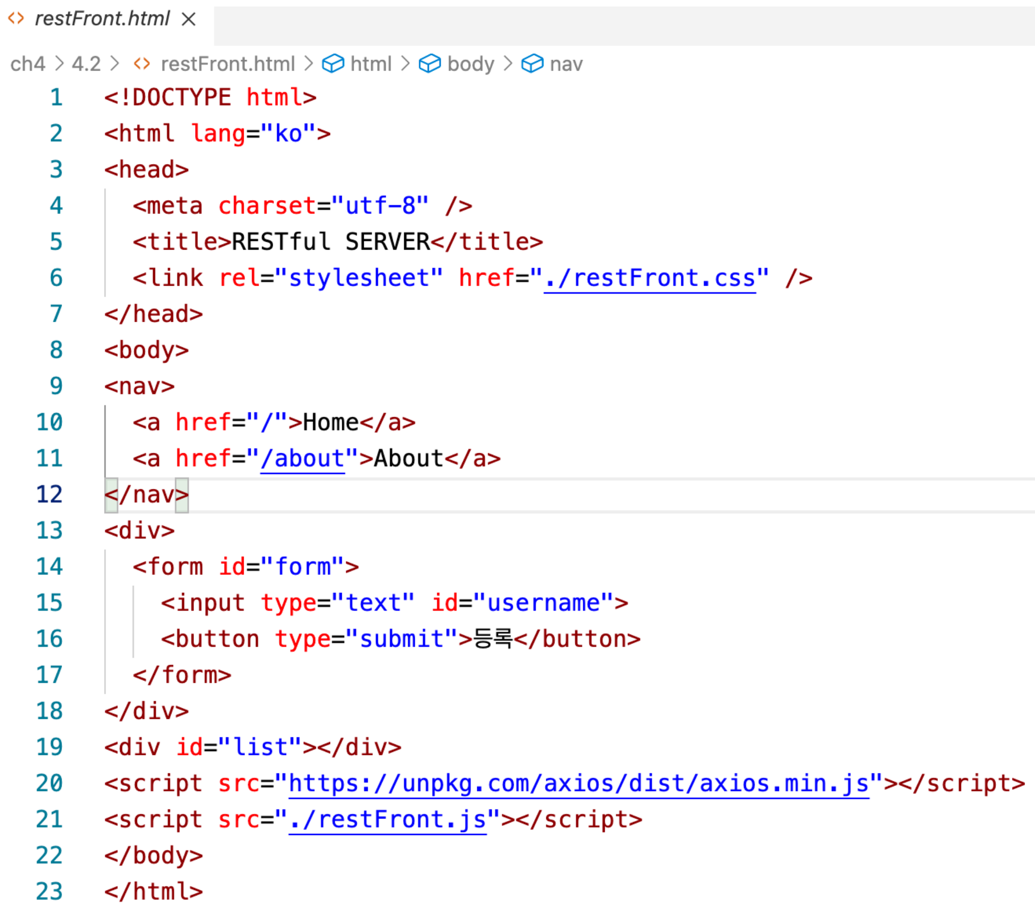
```
1  const http = require('http');
2  const fs = require('fs').promises;
3  const path = require('path');
4
5  const users = {}; // 데이터 저장용
6
7  http.createServer(async (req, res) => {
8    try {
9      if (req.method === 'GET') {
10       if (req.url === '/') {
11         const data = await fs.readFile(path.join(__dirname, 'restFront.html'));
12         res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
13         return res.end(data);
14       } else if (req.url === '/about') {
15         const data = await fs.readFile(path.join(__dirname, 'about.html'));
16         res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
17         return res.end(data);
18       } else if (req.url === '/users') {
19         res.writeHead(200, { 'Content-Type': 'application/json; charset=utf-8' });
20         return res.end(JSON.stringify(users));
21       }
22       // /도 /about도 /users도 아니면
23       try {
24         const data = await fs.readFile(path.join(__dirname, req.url));
25         return res.end(data);
26       } catch (err) {
27         // 주소에 해당하는 라우트를 못 찾았다는 404 Not Found error 발생
28       }
```

REST server example

- about.html
- restFront.html
  - line 19:
    - Setting up the user list dynamically in the restFront.js file.



```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>RESTful SERVER</title>
  <link rel="stylesheet" href="./restFront.css" />
</head>
<body>
<nav>
  <a href="/">Home</a>
  <a href="/about">About</a>
</nav>
<div>
  <h2>소개 페이지입니다.</h2>
  <p>사용자 이름을 등록하세요!</p>
</div>
</body>
</html>
```



```html
1  <!DOCTYPE html>
2  <html lang="ko">
3  <head>
4    <meta charset="utf-8" />
5    <title>RESTful SERVER</title>
6    <link rel="stylesheet" href="./restFront.css" />
7  </head>
8  <body>
9  <nav>
10   <a href="/">Home</a>
11   <a href="/about">About</a>
12 </nav>
13 <div>
14   <form id="form">
15     <input type="text" id="username">
16     <button type="submit">등록</button>
17   </form>
18 </div>
19 <div id="list"></div>
20 <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
21 <script src="./restFront.js"></script>
22 </body>
23 </html>
```

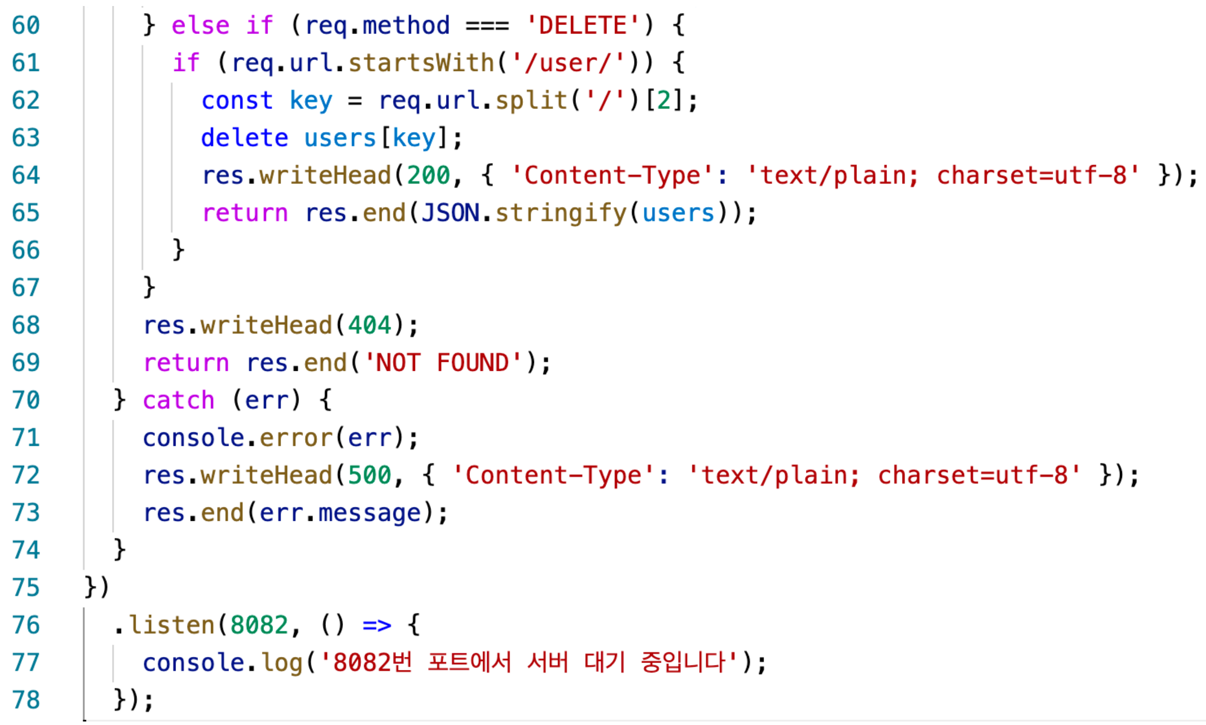
REST server example

- line 29 ~ 45
  - Block for handling POST method.
- line 46 ~ 59
  - Block for handling PUT method.

```
29      } else if (req.method === 'POST') {
30        if (req.url === '/user') {
31          let body = '';
32          // 요청의 body를 stream 형식으로 받음
33          req.on('data', (data) => {
34            body += data;
35          });
36          // 요청의 body를 다 받은 후 실행됨
37          return req.on('end', () => {
38            console.log('POST 본문(Body):', body);
39            const { name } = JSON.parse(body);
40            const id = Date.now();
41            users[id] = name;
42            res.writeHead(201, { 'Content-Type': 'text/plain; charset=utf-8' });
43            res.end('등록 성공');
44          });
45        }
46      } else if (req.method === 'PUT') {
47        if (req.url.startsWith('/user/')) {
48          const key = req.url.split('/')[2];
49          let body = '';
50          req.on('data', (data) => {
51            body += data;
52          });
53          return req.on('end', () => {
54            console.log('PUT 본문(Body):', body);
55            users[key] = JSON.parse(body).name;
56            res.writeHead(200, { 'Content-Type': 'text/plain; charset=utf-8' });
57            return res.end(JSON.stringify(users));
58          });
59        }
```

REST server example

- line 60 ~ 67
  - Block for handling DELETE method.
- line 68 ~ 69
  - For request for other, responds 404 NOT FOUND error
- line 76 ~ 78
  - use 8082 port



```
60      } else if (req.method === 'DELETE') {
61        if (req.url.startsWith('/user/')) {
62          const key = req.url.split('/')[2];
63          delete users[key];
64          res.writeHead(200, { 'Content-Type': 'text/plain; charset=utf-8' });
65          return res.end(JSON.stringify(users));
66        }
67      }
68      res.writeHead(404);
69      return res.end('NOT FOUND');
70    } catch (err) {
71      console.error(err);
72      res.writeHead(500, { 'Content-Type': 'text/plain; charset=utf-8' });
73      res.end(err.message);
74    }
75  })
76    .listen(8082, () => {
77      console.log('8082번 포트에서 서버 대기 중입니다');
78    });
```

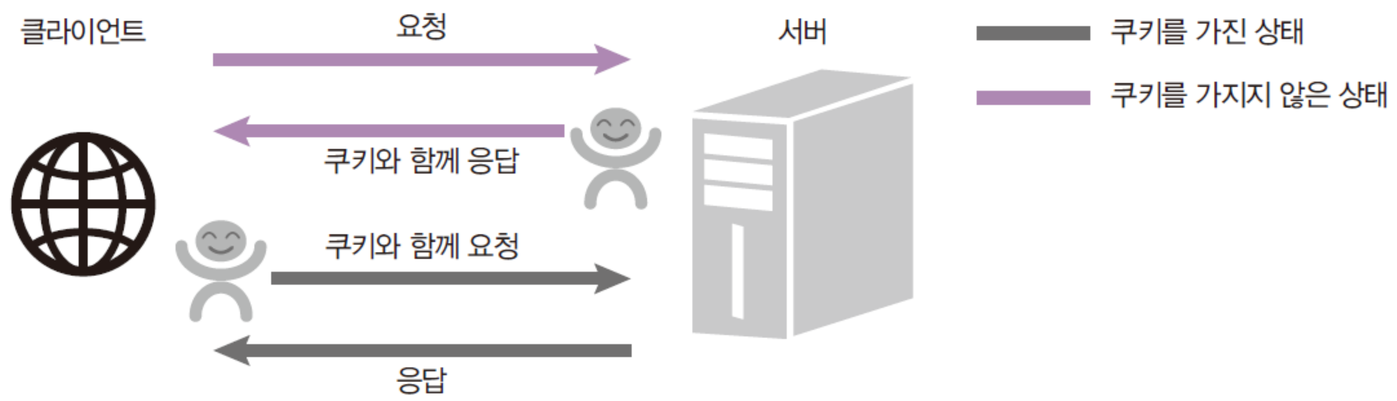
Cookie and Session

CSLAB

Motivation for Cookie

- One drawback to requests
  - The sender's identity is unknown (only IP address and browser information are available)
  - Implementing login is required
  - Cookies and sessions are necessary
- Cookie: Key-value pairs
  - e.g., name=kwondh
  - Enclosed and sent to the server with every request
  - The server reads the cookie to identify the user

▼ 그림 4–13 쿠키



Cookie Server

- Implementing the insertion of cookies directly
  - writeHead: Method to input into the request header
  - Set-Cookie: Command to instruct the browser to set a cookie

콘솔

```
$ node cookie
8083번 포트에서 서버 대기 중입니다!
```

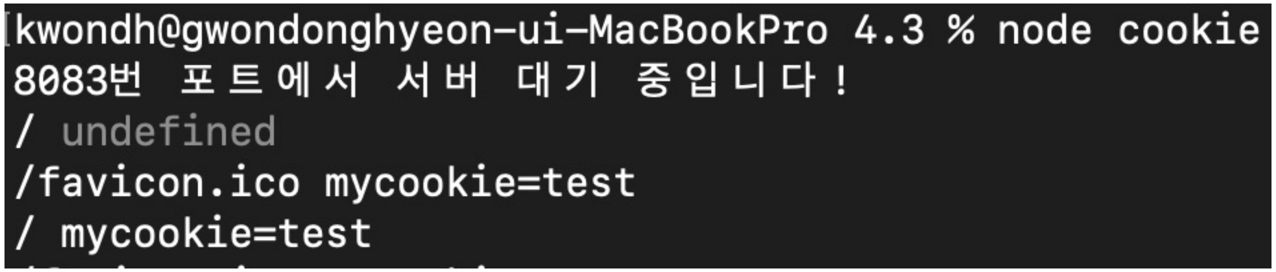
cookie.js

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

http.createServer((req, res) => {
  console.log(req.url, req.headers.cookie);
  res.writeHead(200, { 'Set-Cookie': 'mycookie=test' });
  res.end('Hello Cookie');
})
  .listen(8083, () => {
    console.log('8083번 포트에서 서버 대기 중입니다!');
  });
```

Cookie Server

- req.headers.cookie: Cookies are contained as a string
- req.url: Request address

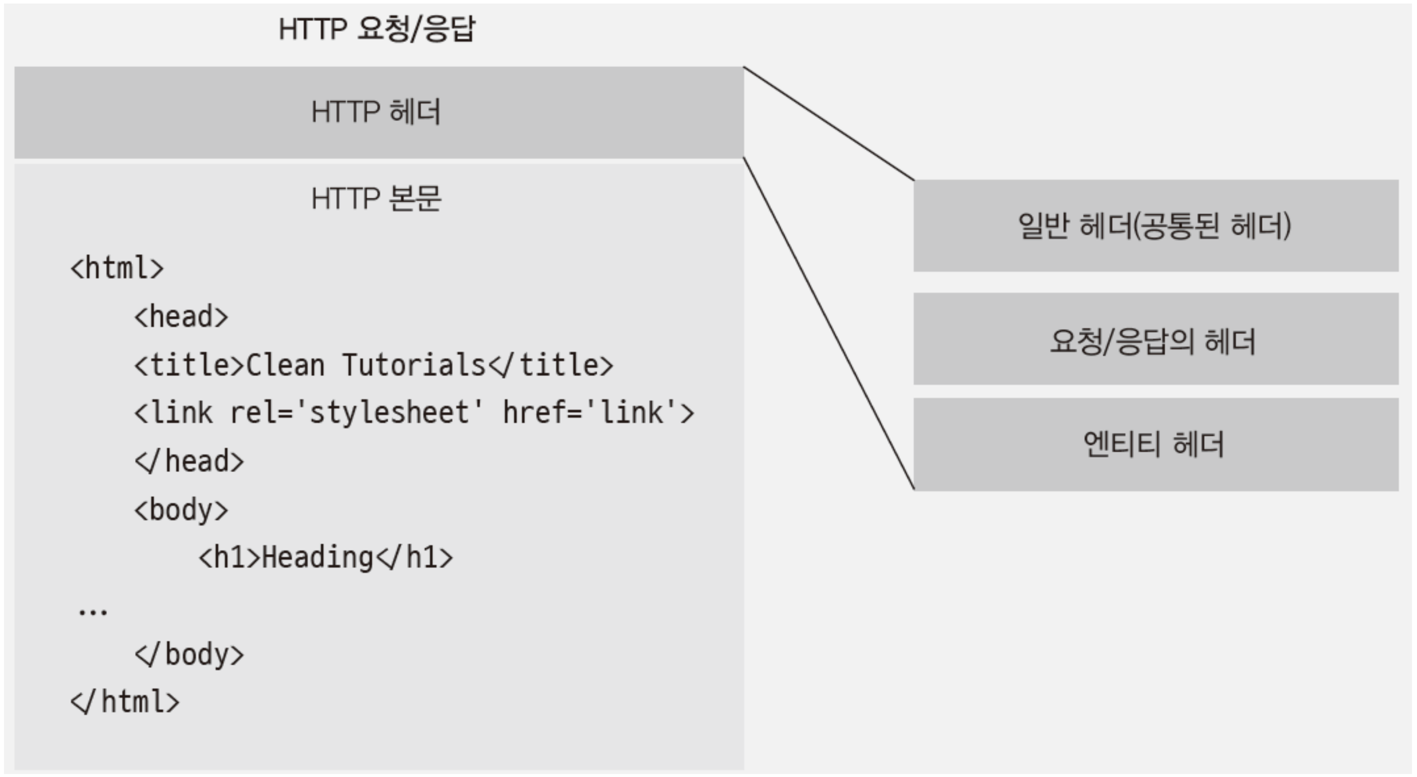
- Connect to localhost:8082
  - When a request is sent and a response is received, cookies are set
  - favicon.ico is a request automatically sent by the browser
  - Cookies are included in the second request, favicon.ico



```
[kwondh@gwondonghyeon-ui-MacBookPro 4.3 % node cookie
8083번 포트에서 서버 대기 중입니다!
/ undefined
/favicon.ico mycookie=test
/ mycookie=test
```

Header and Body

- HTTP requests and responses have headers and bodies
  - Headers contain information about the request or response
  - The body contains the actual data being sent or received
  - Cookies, being supplementary information, are stored in the header



```
<html>
    <head>
    <title>Clean Tutorials</title>
    <link rel='stylesheet' href='link'>
    </head>
    <body>
        <h1>Heading</h1>
...
    </body>
</html>
```

HTTP Status Code

- The value passed as the first argument to the writeHead method indicates whether the request was successful or failed.
  - 2XX: Represents success. Commonly used status codes include 200 (OK) and 201 (Created).
  - 3XX: Indicates redirection (moving to a different page). This code is used when a different address is specified for the requested resource. Commonly used codes are 301 (Moved Permanently) and 302 (Found).
  - 4XX: Represents client errors, indicating that there was an error in the request itself. Commonly used codes include 401 (Unauthorized), 403 (Forbidden), and 404 (Not Found).
  - 5XX: Indicates server errors. This occurs when the request is properly received, but the server encounters an error. It is important to program carefully to avoid triggering these errors. Generally, the server automatically sends a 5XX status code in the event of an unexpected error. Commonly used codes include 500 (Internal Server Error), 502 (Bad Gateway), and 503 (Service Unavailable).

Cookie Example

- Inserting personal information into cookies.
  - parseCookies: Converts the cookie string into an object.
  - The server logic is divided based on whether the address is /login or /.
  - In the case of /login, it checks if a name is received through the query string and stores it as a cookie.
- For other cases, it checks if a cookie exists.
  - If it exists, it displays a welcome greeting.
  - If not, it redirects to the login page.

cookie2.html

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>쿠키&세션 이해하기</title>
</head>
<body>
<form action="/login">
    <input id="name" name="name" placeholder="이름을 입력하세요" />
    <button id="login">로그인</button>
</form>
```

cookie2.js

```javascript
const http = require('http');
const fs = require('fs').promises;
const url = require('url');
const qs = require('querystring');

const parseCookies = (cookie = '') =>
  cookie
    .split(';')
    .map(v => v.split('='))                                   ❶
    .reduce((acc, [k, v]) => {
      acc[k.trim()] = decodeURIComponent(v);
      return acc;
    }, {});

http.createServer(async (req, res) => {
  const cookies = parseCookies(req.headers.cookie);

  // 주소가 /login으로 시작하는 경우
  if (req.url.startsWith('/login')) {
    const { query } = url.parse(req.url);
    const { name } = qs.parse(query);
    const expires = new Date();
    // 쿠키 유효 시간을 현재 시간 + 5분으로 설정
    expires.setMinutes(expires.getMinutes() + 5);             ❷
    res.writeHead(302, {
      Location: '/',
      'Set-Cookie': `name=${encodeURIComponent(name)}; Expires=
➡ ${expires.toGMTString()}; HttpOnly; Path=/`,
    });
    res.end();

  // name이라는 쿠키가 있는 경우
  } else if (cookies.name) {
    res.writeHead(200, { 'Content-Type': 'text/plain; charset=utf-8' });
    res.end(`${cookies.name}님 안녕하세요`);
  } else {                                                    ❸
    try {
      const data = await fs.readFile('./cookie2.html');
      res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });

      res.end(data);
    } catch (err) {
      res.writeHead(500, { 'Content-Type': 'text/plain; charset=utf-8' });   ❸
      res.end(err.message);
    }
  }

})
  .listen(8084, () => {
    console.log('8084번 포트에서 서버 대기 중입니다!');
  });
```

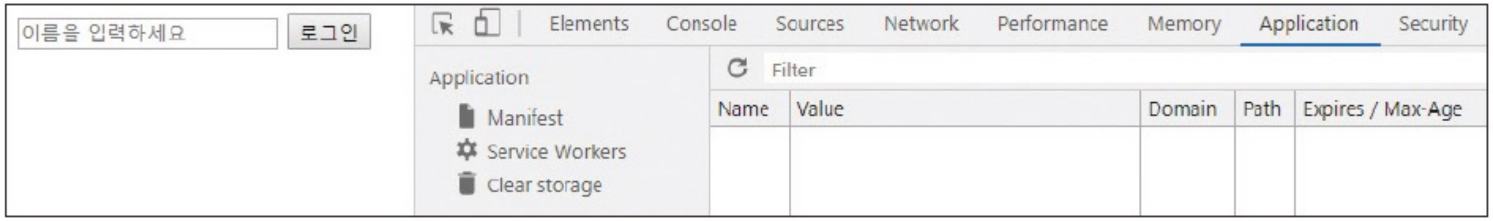
Cookie Example

콘솔

```
$ node cookie2
8084번 포트에서 서버 대기 중입니다!
```

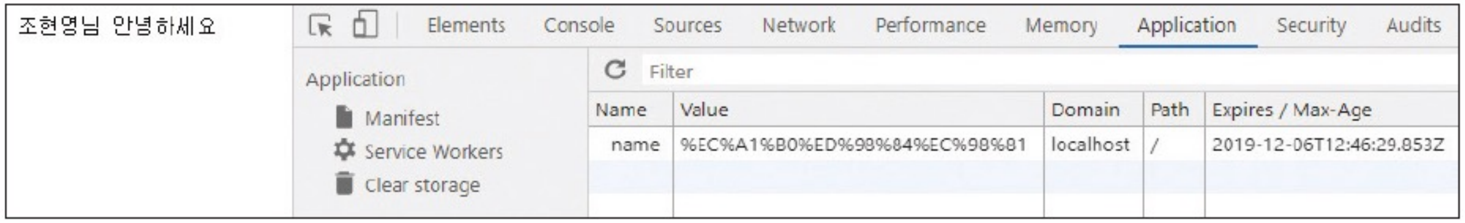
- Connect to localhost:8084 port.
  - Open the Application tab (F12).
  - When logging in, a cookie will be created.

그림 4-15 로그인 이전



Name	Value	Domain	Path	Expires / Max-Age

그림 4-16 로그인 이후



조현영님 안녕하세요

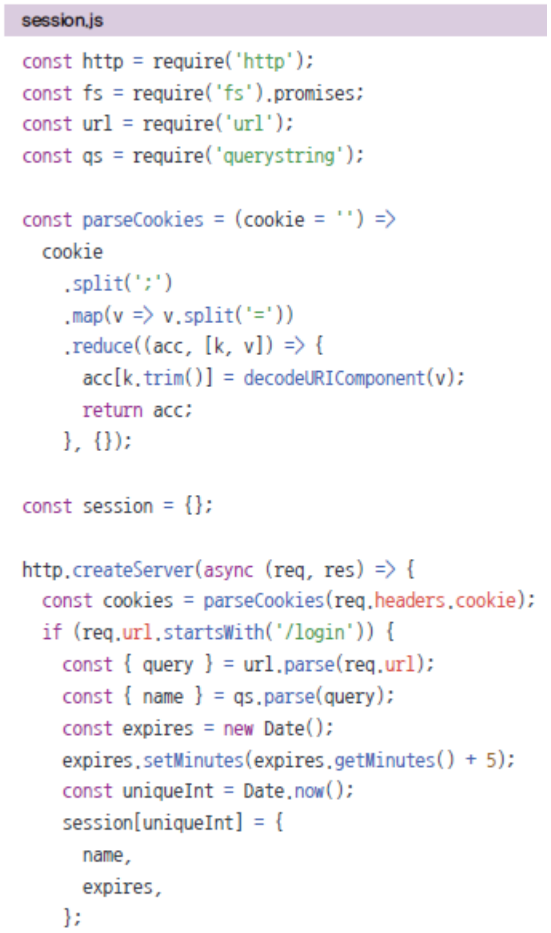
Name	Value	Domain	Path	Expires / Max-Age
name	%EC%A1%B0%ED%98%84%EC%98%81	localhost	/	2019-12-06T12:46:29.853Z

Cookie Options

- There are various options when setting Set-Cookie:
  - CookieName=CookieValue: The basic value of the cookie. It is set as mycookie=test or name=zerocho.
  - Expires=Date: The expiration date of the cookie. Once this date passes, the cookie will be removed. The default is until the client is closed.
  - Max-age=Seconds: Similar to Expires, but you can specify seconds instead of a date. The cookie will be removed after the specified seconds. It takes precedence over Expires.
  - Domain=DomainName: Specifies the domain to which the cookie will be sent. The default is the current domain.
  - Path=URL: Specifies the URL to which the cookie will be sent. The default is '/' and in this case, the cookie can be sent to any URL.
  - Secure: The cookie is only sent if the connection is over HTTPS.
  - HttpOnly: When set, the cookie cannot be accessed by JavaScript. It is recommended to set this to prevent cookie manipulation.

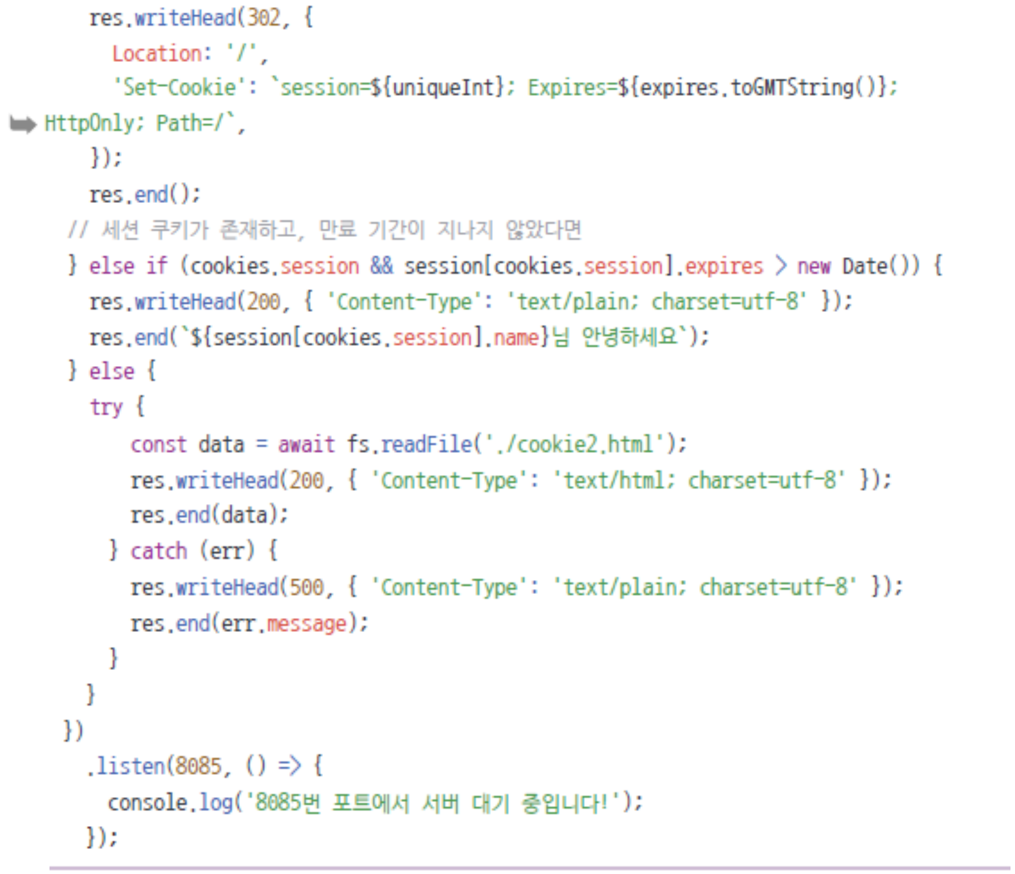
Session

- Cookie information is at risk of exposure and modification. For sensitive information, it is recommended to manage it on the server and provide only the session key to the client.



session.js

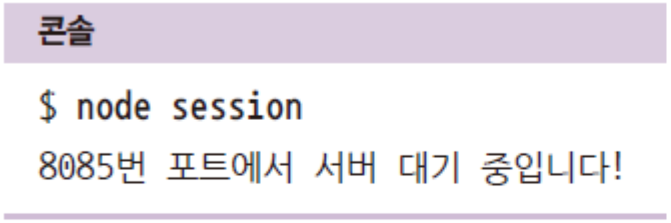
```javascript
const http = require('http');
const fs = require('fs').promises;
const url = require('url');
const qs = require('querystring');

const parseCookies = (cookie = '') =>
  cookie
    .split(';')
    .map(v => v.split('='))
    .reduce((acc, [k, v]) => {
      acc[k.trim()] = decodeURIComponent(v);
      return acc;
    }, {});

const session = {};

http.createServer(async (req, res) => {
  const cookies = parseCookies(req.headers.cookie);
  if (req.url.startsWith('/login')) {
    const { query } = url.parse(req.url);
    const { name } = qs.parse(query);
    const expires = new Date();
    expires.setMinutes(expires.getMinutes() + 5);
    const uniqueInt = Date.now();
    session[uniqueInt] = {
      name,
      expires,
    };
```



```javascript
    res.writeHead(302, {
      Location: '/',
      'Set-Cookie': `session=${uniqueInt}; Expires=${expires.toGMTString()};
➥ HttpOnly; Path=/`,
    });
    res.end();
  // 세션 쿠키가 존재하고, 만료 기간이 지나지 않았다면
  } else if (cookies.session && session[cookies.session].expires > new Date()) {
    res.writeHead(200, { 'Content-Type': 'text/plain; charset=utf-8' });
    res.end(`${session[cookies.session].name}님 안녕하세요`);
  } else {
    try {
        const data = await fs.readFile('./cookie2.html');
        res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
        res.end(data);
      } catch (err) {
        res.writeHead(500, { 'Content-Type': 'text/plain; charset=utf-8' });
        res.end(err.message);
      }
    }
})
  .listen(8085, () => {
    console.log('8085번 포트에서 서버 대기 중입니다!');
  });
```

Session

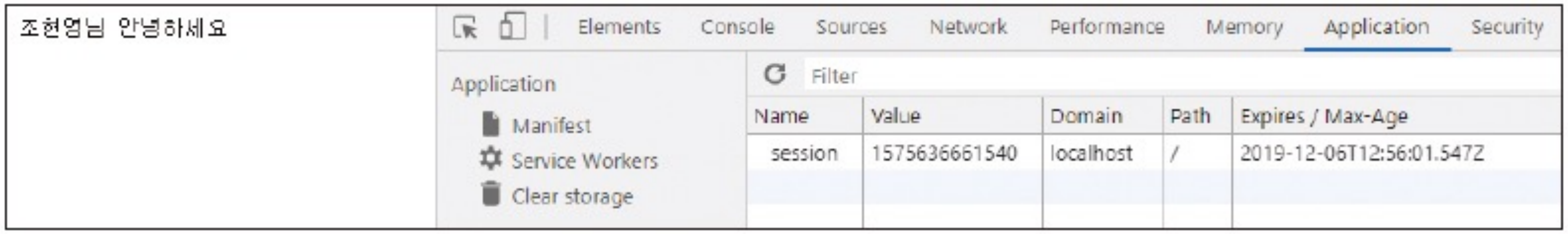
- localhost:8085



콘솔

```
$ node session
8085번 포트에서 서버 대기 중입니다!
```

그림 4-17 로그인 이후



조현영님 안녕하세요

Name	Value	Domain	Path	Expires / Max-Age
session	1575636661540	localhost	/	2019-12-06T12:56:01.547Z

- Avoid implementing sessions directly on a real server.
  - Use express-session, which we will learn about later.

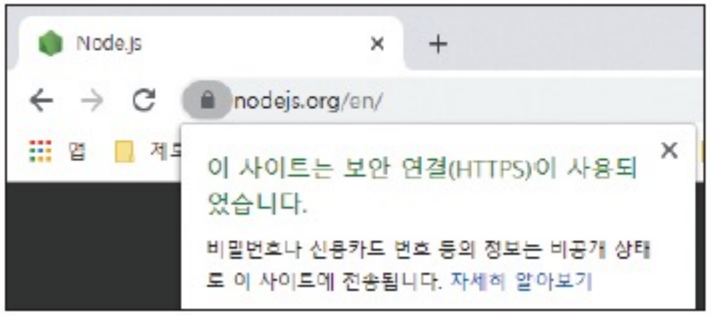
https and http2

CSLAB

https

- The module for adding SSL encryption to a web server Encrypts incoming and outgoing data, ensuring that even if someone intercepts the request in transit, the content cannot be viewed.
- These days, applying HTTPS is essential, especially where personal information is involved.

▼ 그림 4-18 https 적용 화면



이 사이트는 보안 연결(HTTPS)이 사용되었습니다.

비밀번호나 신용카드 번호 등의 정보는 비공개 상태로 이 사이트에 전송됩니다. 자세히 알아보기

https server

- Transforming an HTTP server into an HTTPS server requires an SSL certificate, which needs to be obtained.
- The createServer function takes two arguments:
  - The first argument is an options object related to the certificate. Include files such as pem, crt, key, which you obtain when purchasing the certificate.
  - The second argument is the server logic.

server1.js

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

http.createServer((req, res) => {
  res.writeHead(500, { '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!');
  });
```

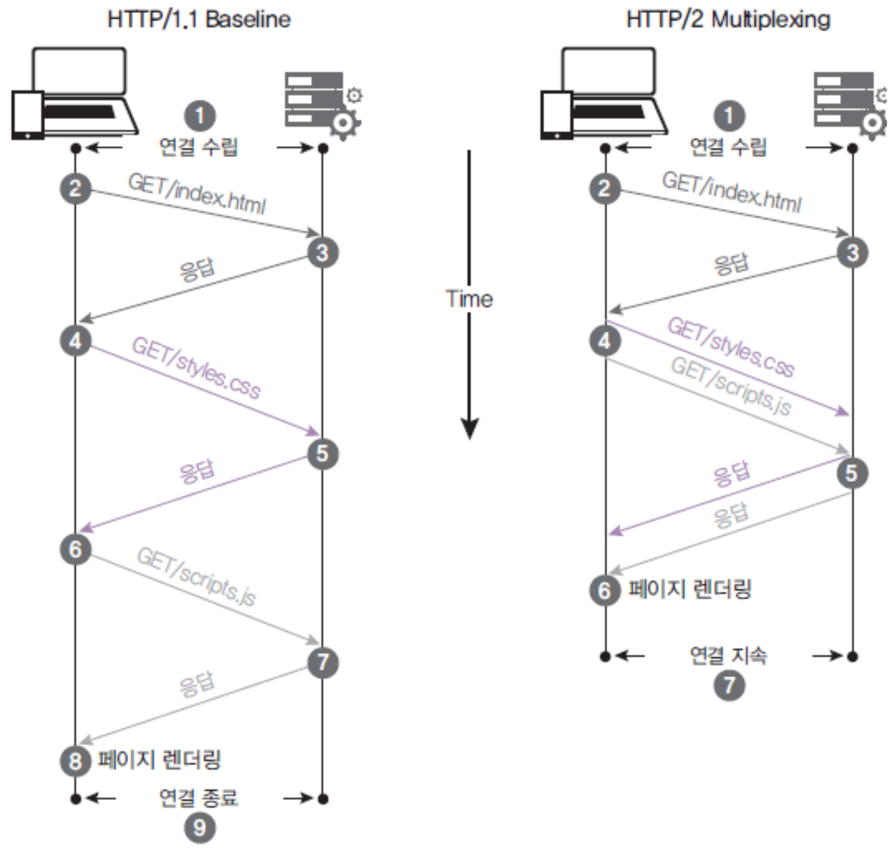
server1-3.js

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

https.createServer({
  cert: fs.readFileSync('도메인 인증서 경로'),
  key: fs.readFileSync('도메인 비밀키 경로'),
  ca: [
    fs.readFileSync('상위 인증서 경로'),
    fs.readFileSync('상위 인증서 경로'),
  ],
}, (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(443, () => {
    console.log('443번 포트에서 서버 대기 중입니다!');
  });
```

http2

- A module that incorporates SSL encryption along with the latest HTTP protocol, http/2.
  - Request and response mechanisms are improved compared to the traditional http/1.1.
  - This results in enhanced web speed.

그림 4-20 http/1.1과 http/2의 비교



http2 server

- Switching from the https module to http2 and changing the createServer method to createSecureServer.

server1-4.js

```
const http2 = require('http2');
const fs = require('fs');

http2.createSecureServer({
  cert: fs.readFileSync('도메인 인증서 경로'),
  key: fs.readFileSync('도메인 비밀키 경로'),
  ca: [
    fs.readFileSync('상위 인증서 경로'),
    fs.readFileSync('상위 인증서 경로'),
  ],
}, (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(443, () => {
    console.log('443번 포트에서 서버 대기 중입니다!');
  });
```

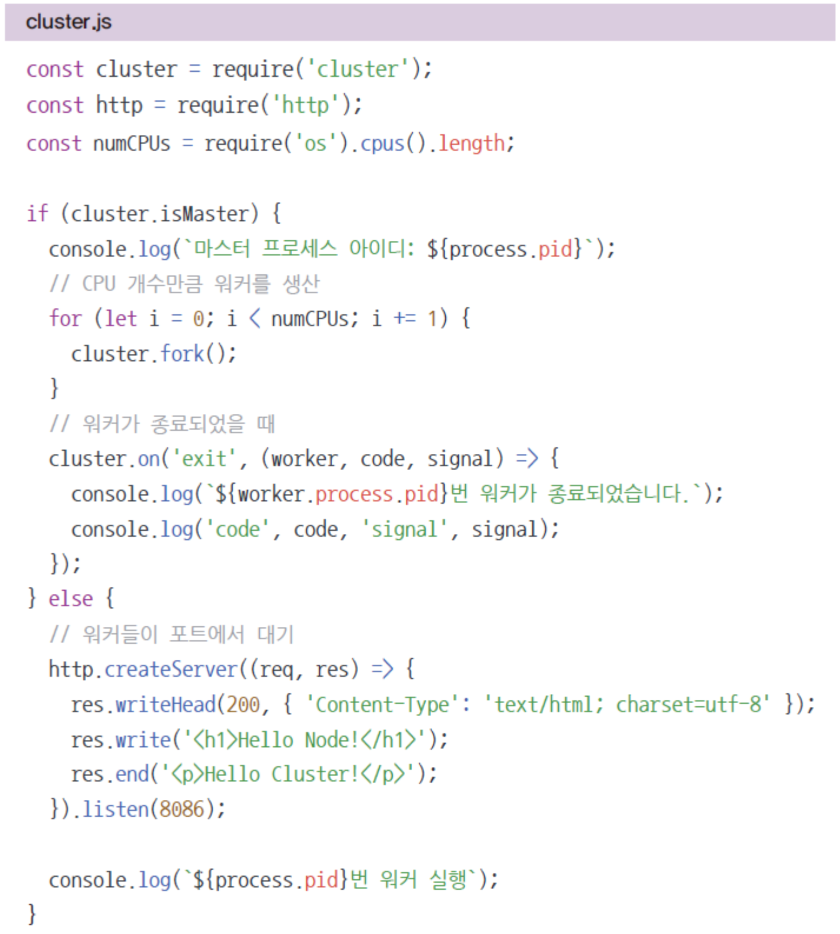
Cluster

cluster

- A module that allows Node.js, which is inherently single-threaded, to make use of all CPU cores.
  - It enables running multiple Node processes that share the same port.
  - When a large number of requests come in, they are distributed among the parallel running servers.
  - This helps alleviate the load on the server.
    - In a server with 8 cores, for example, typically only one core is utilized.
    - However, with clustering, you can assign one Node process per core, improving performance.
  - Note that this doesn't result in an eightfold increase in performance, but it is an enhancement.
  - The downside is that computer resources such as memory and sessions cannot be shared. This can be addressed by using a separate server like Redis.

Server Clustering

- The master process is responsible for creating worker processes, typically one for each CPU core.
  - This setup, resembling the structure of worker_threads, ensures that incoming requests are evenly distributed among the worker processes.

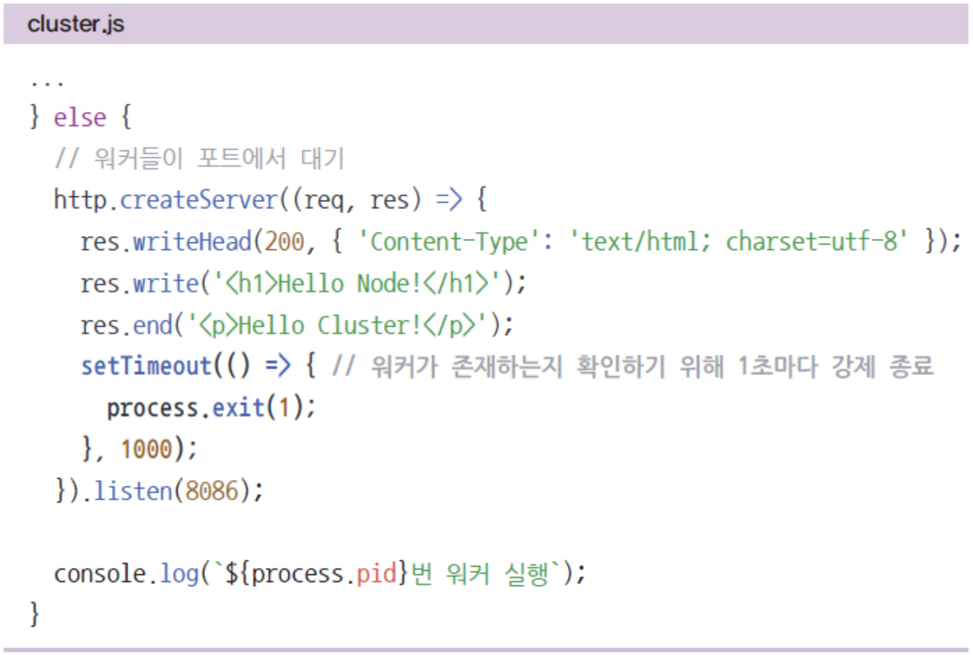


cluster.js

```
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;

if (cluster.isMaster) {
  console.log(`마스터 프로세스 아이디: ${process.pid}`);
  // CPU 개수만큼 워커를 생산
  for (let i = 0; i < numCPUs; i += 1) {
    cluster.fork();
  }
  // 워커가 종료되었을 때
  cluster.on('exit', (worker, code, signal) => {
    console.log(`${worker.process.pid}번 워커가 종료되었습니다.`);
    console.log('code', code, 'signal', signal);
  });
} else {
  // 워커들이 포트에서 대기
  http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
    res.write('<h1>Hello Node!</h1>');
    res.end('<p>Hello Cluster!</p>');
  }).listen(8086);

  console.log(`${process.pid}번 워커 실행`);
}
```

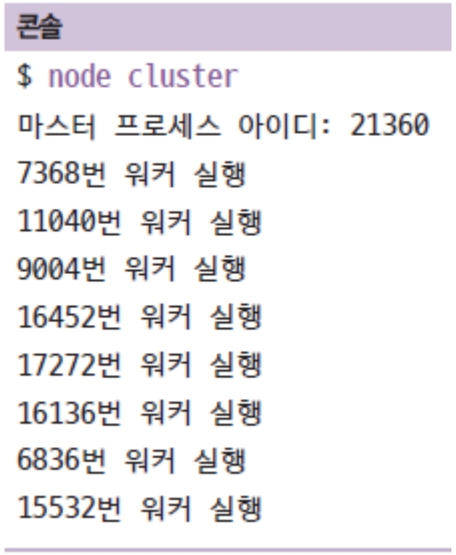
Worker Process Example 1

- Configure the server to shut down after each incoming request.
  - If the computer has 8 cores, the server will handle 8 requests and then terminate.



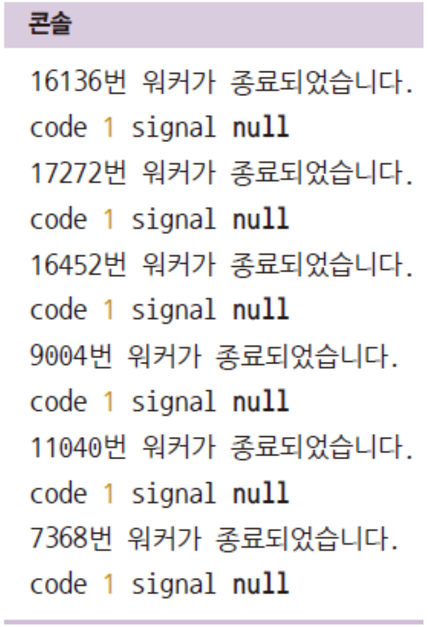
cluster.js

```
...
} else {
  // 워커들이 포트에서 대기
  http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
    res.write('<h1>Hello Node!</h1>');
    res.end('<p>Hello Cluster!</p>');
    setTimeout(() => { // 워커가 존재하는지 확인하기 위해 1초마다 강제 종료
      process.exit(1);
    }, 1000);
  }).listen(8086);

  console.log(`${process.pid}번 워커 실행`);
}
```



콘솔

```
$ node cluster
마스터 프로세스 아이디: 21360
7368번 워커 실행
11040번 워커 실행
9004번 워커 실행
16452번 워커 실행
17272번 워커 실행
16136번 워커 실행
6836번 워커 실행
15532번 워커 실행
```



콘솔

```
16136번 워커가 종료되었습니다.
code 1 signal null
17272번 워커가 종료되었습니다.
code 1 signal null
16452번 워커가 종료되었습니다.
code 1 signal null
9004번 워커가 종료되었습니다.
code 1 signal null
11040번 워커가 종료되었습니다.
code 1 signal null
7368번 워커가 종료되었습니다.
code 1 signal null
```

Worker Process Example 2

- Creating a new worker each time one dies
    - Not a recommended approach
    - It doesn't address the underlying issues and will continue to cause problems. However, it can be used as a temporary solution to prevent the server from shutting down completely until the errors are resolved.

cluster.js

```
...
cluster.on('exit', (worker, code, signal) => {
  console.log(`${worker.process.pid}번 워커가 종료되었습니다.`);
  console.log('code', code, 'signal', signal);
  cluster.fork();
});
...
```

콘솔

```
28592번 워커가 종료되었습니다.
code 1 signal null
10520번 워커 실행
10520번 워커가 종료되었습니다.
code 1 signal null
23248번 워커 실행
```