**[4장. http 모듈로 웹 서버 만들기]**

1. 요청과 응답 이해하기

const http = require('http');

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

// 여기에 어떻게 응답할지 적어줍니다.

});

const http = require('http');

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

res.write('<h1>Hello Node!</h1>');

res.end('<p>Hello Server!</p>');

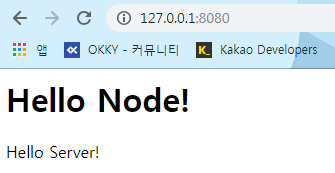
}).listen(8080, ()=>{

console.log('8080번 포트에서 서버 대기 중입니다!');

});

C:\materK1m\newmimac\test\_kes\http\_module>node server1

8080번 포트에서 서버 대기 중입니다!



const http = require('http');

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

res.write('<h1>Hello Node!</h1>');

res.end('<p>Hello Server!</p>');

});

server.listen(8080);

server.on('listening', ()=>{

console.log('8080서버 포트에서 서버 대기 중입니다!');

});

server.on('error', (error)=>{

console.error(error);

});

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8"/>

<title>Node.js 웹 서버</title>

</head>

<body>

<h1>Node.js 웹 서버</h1>

<p>만들 준비 되셨나요 ? </p>

</body>

</html>

const http = require('http');

const fs = require('fs');

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

fs.readFile('./server2.html', (err, data) =>{

if(err) throw err;

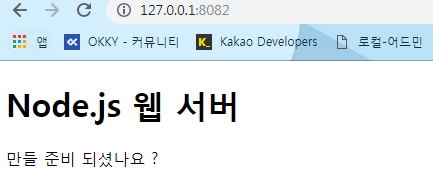
res.end(data);

});

}).listen(8082, ()=>{

console.log('8082번 포트에서 서버 대기중입니다!');

});



2. 쿠키와 세션 이해하기

const http = require('http');

const parseCookies = (cookie = '') =>

cookie

.split(';')

.map(v => v.split("="))

.map(([k, ...vs])=>[k,vs.join('=')])

.reduce((acc, [k,v])=>{

acc[k.trim()] = decodeURIComponent(v);

return acc;

}, {});

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

const cookie = parseCookies(req.headers.cookie);

console.log(req.url, cookie);

res.writeHead(200, {'Set-Cookie' : 'mycookie=test'});

res.end('Hello Cookie');

})

.listen(8083, ()=>{

console.log('8083번 포트에서 서버 대기중입니다!');

});

C:\materK1m\newmimac\test\_kes\http\_module>node server3

8083번 포트에서 서버 대기중입니다!

/ { '': '' }

/favicon.ico { mycookie: 'test' }

<!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>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>쿠키&세션 이해하기</title>

</head>

<body>

<form action="/login">

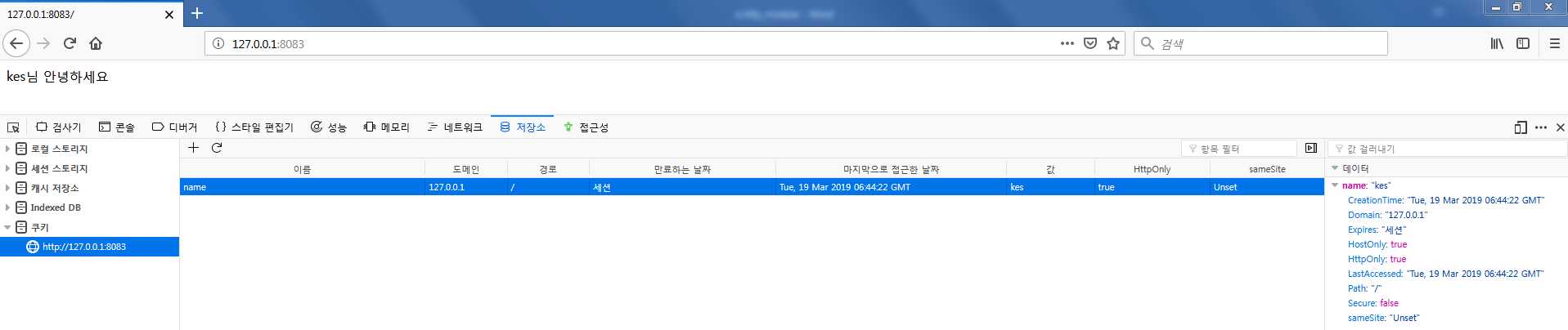
<input id="name" name="name" placeholder="이름을 입력하세요."/>

<button id="login">로그인</button>

</form>

</body>

</html>



const http = require('http');

const fs = require('fs');

const url = require('url');

const qs = require('querystring');

const parseCookies = (cookie = '') =>

cookie

.split(';')

.map(v => v.split("="))

.map(([k, ...vs])=>[k,vs.join('=')])

.reduce((acc, [k,v])=>{

acc[k.trim()] = decodeURIComponent(v);

return acc;

}, {});

const session = {};

http.createServer((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 randomInt = +new Date();

session[randomInt] = {

name,

expires,

};

res.writeHead(302,{

Location : '/',

'Set-Cookie' : `session=${randomInt}; Expires=${expires.toGMTString()}; HttpOnly; Path=/`,

});

res.end();

}else if(cookies.session && session[cookies.session].expires > new Date()){

res.writeHead(200, {'Content-Type':'text/html; charset=utf-8'});

res.end(`${session[cookies.session].name}님 안녕하세요`);

}else{

fs.readFile('./server4.html', (err, data) => {

if(err){

throw err;

}

res.end(data);

});

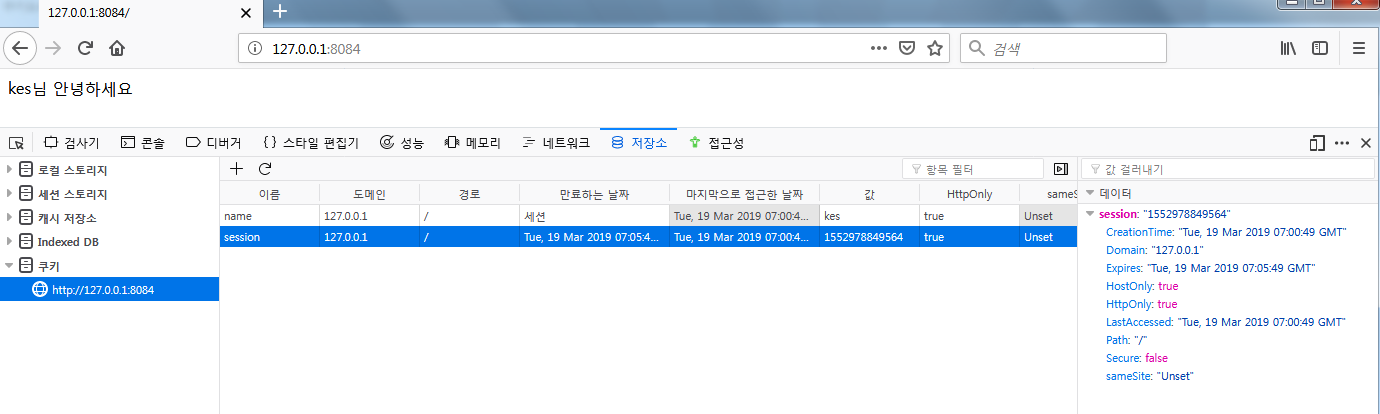
}

})

.listen(8084, () => {

console.log('8084번 포트에서 서버 대기중입니다!');

});



**3. REST API와 라우팅**

a {

color: blue;

text-decoration: none;

}

<!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>

<form id="form">

<input type="text" id="username">

<button type="submit">등록</button>

</form>

</div>

<div id="list"></div>

<script src="./restFront.js"></script>

</body>

</html>

function getUser() { // 로딩 시 사용자 가져오는 함수

var xhr = new XMLHttpRequest();

xhr.onload = function () {

if (xhr.status === 200) {

var users = JSON.parse(xhr.responseText);

var list = document.getElementById('list');

list.innerHTML = '';

Object.keys(users).map(function (key) {

var userDiv = document.createElement('div');

var span = document.createElement('span');

span.textContent = users[key];

var edit = document.createElement('button');

edit.textContent = '수정';

edit.addEventListener('click', function () { // 수정 버튼 클릭

var name = prompt('바꿀 이름을 입력하세요');

if (!name) {

return alert('이름을 반드시 입력하셔야 합니다');

}

var xhr = new XMLHttpRequest();

xhr.onload = function () {

if (xhr.status === 200) {

console.log(xhr.responseText);

getUser();

} else {

console.error(xhr.responseText);

}

};

xhr.open('PUT', '/users/' + key);

xhr.setRequestHeader('Content-Type', 'application/json');

xhr.send(JSON.stringify({ name: name }));

});

var remove = document.createElement('button');

remove.textContent = '삭제';

remove.addEventListener('click', function () { // 삭제 버튼 클릭

var xhr = new XMLHttpRequest();

xhr.onload = function () {

if (xhr.status === 200) {

console.log(xhr.responseText);

getUser();

} else {

console.error(xhr.responseText);

}

};

xhr.open('DELETE', '/users/' + key);

xhr.send();

});

userDiv.appendChild(span);

userDiv.appendChild(edit);

userDiv.appendChild(remove);

list.appendChild(userDiv);

});

} else {

console.error(xhr.responseText);

}

};

xhr.open('GET', '/users');

xhr.send();

}

window.onload = getUser; // 로딩 시 getUser 호출

// 폼 제출

document.getElementById('form').addEventListener('submit', function (e) {

e.preventDefault();

var name = e.target.username.value;

if (!name) {

return alert('이름을 입력하세요');

}

var xhr = new XMLHttpRequest();

xhr.onload = function () {

if (xhr.status === 201) {

console.log(xhr.responseText);

getUser();

} else {

console.error(xhr.responseText);

}

};

xhr.open('POST', '/users');

xhr.setRequestHeader('Content-Type', 'application/json');

xhr.send(JSON.stringify({ name: name }));

e.target.username.value = '';

});

<!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>

const http = require('http');

const fs = require('fs');

const users = {};

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

if (req.method === 'GET') {

if (req.url === '/') {

return fs.readFile('./restFront.html', (err, data) => {

if (err) {

throw err;

}

res.end(data);

});

} else if (req.url === '/about') {

return fs.readFile('./about.html', (err, data) => {

if (err) {

throw err;

}

res.end(data);

});

} else if (req.url === '/users') {

return res.end(JSON.stringify(users));

}

return fs.readFile(`.${req.url}`, (err, data) => {

if (err) {

res.writeHead(404, 'NOT FOUND');

return res.end('NOT FOUND');

}

return res.end(data);

});

} else if (req.method === 'POST') {

if (req.url === '/users') {

let body = '';

req.on('data', (data) => {

body += data;

});

return req.on('end', () => {

console.log('POST 본문(Body):', body);

const { name } = JSON.parse(body);

const id = +new Date();

users[id] = name;

res.writeHead(201);

res.end('등록 성공');

});

}

} else if (req.method === 'PUT') {

if (req.url.startsWith('/users/')) {

const key = req.url.split('/')[2];

let body = '';

req.on('data', (data) => {

body += data;

});

return req.on('end', () => {

console.log('PUT 본문(Body):', body);

users[key] = JSON.parse(body).name;

return res.end(JSON.stringify(users));

});

}

} else if (req.method === 'DELETE') {

if (req.url.startsWith('/users/')) {

const key = req.url.split('/')[2];

delete users[key];

return res.end(JSON.stringify(users));

}

}

res.writeHead(404, 'NOT FOUND');

return res.end('NOT FOUND');

})

.listen(8085, () => {

console.log('8085번 포트에서 서버 대기중입니다');

});

C:\materK1m\newmimac\test\_kes\http\_module>node restServer

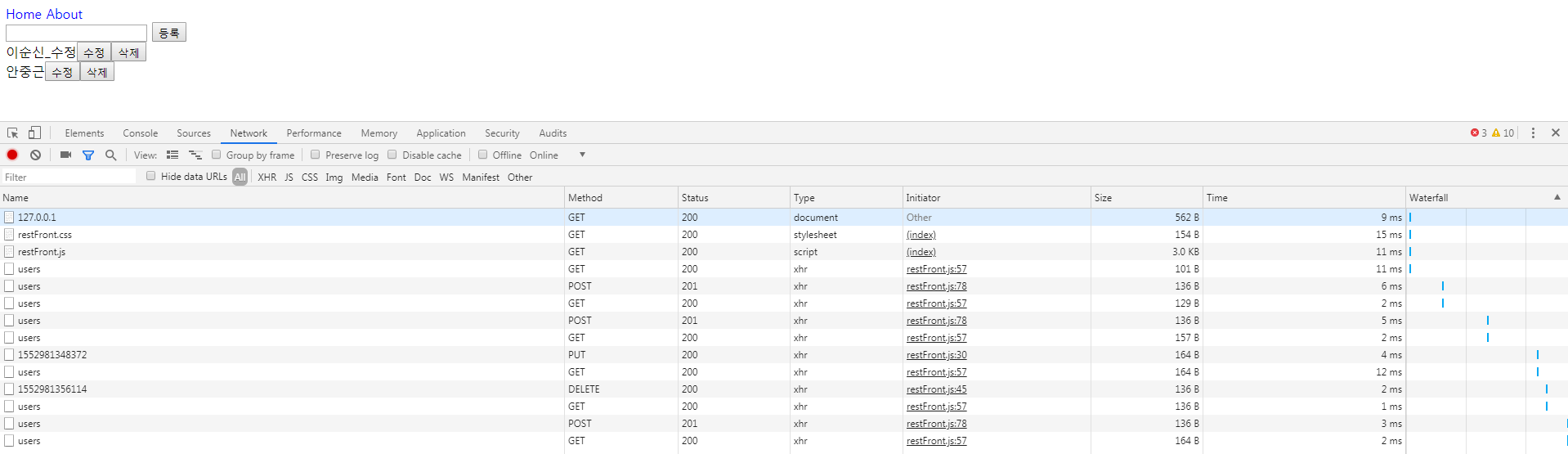
8085번 포트에서 서버 대기중입니다

POST 본문(Body): {"name":"이순신"}

POST 본문(Body): {"name":"안중근"}

PUT 본문(Body): {"name":"이순신\_수정"}

POST 본문(Body): {"name":"안중근"}



4. https와 http2

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.write('<h1>Hello Node!</h1>');

res.end('<p>Hello Server!</p>');

}).listen(443, () => {

console.log('443번 포트에서 서버 대기중입니다!');

});

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) => {

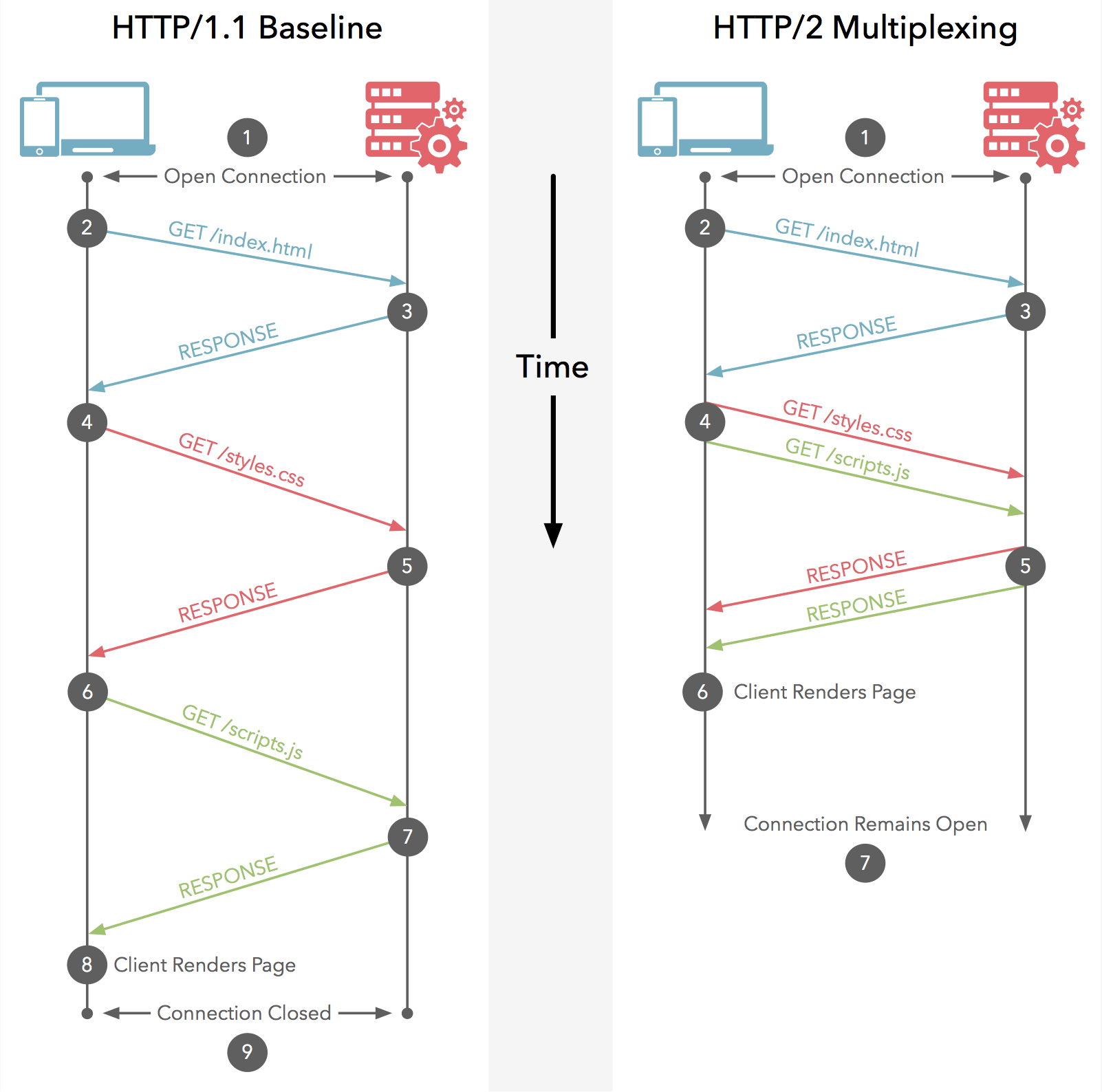
res.write('<h1>Hello Node!</h1>');

res.end('<p>Hello Server!</p>');

}).listen(443, () => {

console.log('443번 포트에서 서버 대기중입니다!');

});



5. cluster

cluster 모듈은 싱글 스레드인 노드가 CPU 코어를 모두 사용할 수 있게 해주는 모듈입니다. 포트를 공유하는 노드 프로세스를 여러 개 둘 수도있어 요청이 많이 들어왔을 때 병렬로 실행된 서버의 개수만큼 요청이 분산되게 할 수도 있습니다.

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.`);

cluster.fork();

});

} else {

// 워커들이 포트에서 대기

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

res.write('<h1>Hello Node!</h1>');

res.end('<p>Hello Cluster!</p>');

setTimeout(() => {

process.exit(1);

}, 1000);

}).listen(8085);

console.log(`${process.pid}번 워커 실행`);

}

C:\materK1m\newmimac\test\_kes\http\_module>node cluster

마스터 프로세스 아이디: 11436

7276번 워커 실행

13844번 워커 실행

12912번 워커 실행

11188번 워커 실행

11188번 워커가 종료되었습니다.

12364번 워커 실행

13844번 워커가 종료되었습니다.

13344번 워커 실행