Node.js

http 모듈 - 김근형 강사

http 모듈

- http 모듈
  - node.js 에서 웹 사이트를 구축할 수 있게 해주는 가장 핵심적인 모듈
  - 웹 어플리케이션을 구축해 클라이언트와 서버와의 통신을 도와줄 수 있게 하는 모듈이다.
  - http 역시 require를 통해 선언하며 아래와 같이 선언이 가능하다.

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

http 모듈

- http 모듈 기본적인 형태

```
// http 모듈 요청
const http = require('http');

// 사용자의 요청을 받는 부분
const server = http.createServer((req, res) => {

});

// 서버 오픈
server.listen(port, callback);
```

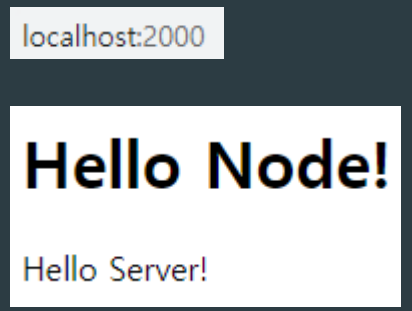
http 모듈

- http 예제 - 1

```
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(2000, () => { // 서버 연결
    console.log('서버 대기 중...  port : 2000');
});
```

```
서버 대기 중...  port : 2000
```



http 모듈

- http 예제 - 2

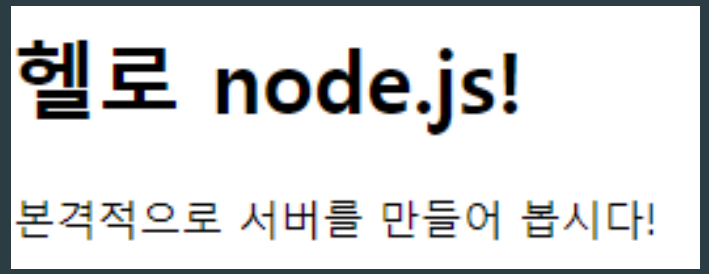
```
const http = require('http');

const server = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
    res.write('<h1>헬로 node.js!</h1>');
    res.end('<p>본격적으로 서버를 만들어 봅시다!</p>');
});
server.listen(2000);

server.on('listening', () => {
    console.log('서버 대기 중...  port : 2000');
});

server.on('error', (err) => {
    console.error(err);
});
```

```
서버 대기 중...  port : 2000
```



http 모듈

- http 예제 - 3

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

http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
    res.write('<h1>Serverr1</h1>');
    res.end('<p>2000번 포트를 사용하고 있는 서버입니다.</p>');
}).listen(2000, () => { // 서버 연결
    console.log('1번 서버... port : 2000');
});

http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });
    res.write('<h1>Server2</h1>');
    res.end('<p>2001번 포트를 사용하고 있는 서버입니다.</p>');
}).listen(2001, () => { // 서버 연결
    console.log('1번 서버... port : 2001');
});
```

```
1번 서버... port : 2000
1번 서버... port : 2001
```

Serverr1

2000번 포트를 사용하고 있는 서버입니다.

Server2

2001번 포트를 사용하고 있는 서버입니다.

http 모듈

- http 예제 - 4
  - fs 모듈을 활용하게 되면 html 파일을 읽어서 사용이 가능하다.

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>html1 파일</title>
</head>
<body>
    <h1>Node.js 웹 서버</h1>
    <p>이 파일을 읽어 사용자에게 출력합니다.</p>
</body>
</html>
```

Node.js 웹 서버

이 파일을 읽어 사용자에게 출력합니다.

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

http.createServer(async (req, res) => {
    try {
        const data = await fs.readFile(__dirname+'/html/html1.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(2000, () => {
    console.log('1번 서버... port : 2000');
});
```

http 주요 객체

- http.ClientRequest
  - 내부적으로 Reqeust를 생성해서 내보내야 할 경우 생성하는 객체
  - CreateServer시 callback 함수에서 제공하는 request 가 이 객체는 아니다.
- http.ServerResponse
  - 이벤트 수신 시 사용하는 객체
  - ServerResponse 객체를 사용해 클라이언트 요청에 대한 응답을 정형화해 보낸다.
  - CreateServer시 callback 함수 두번째 인자로 제공하는 response가 이 객체의 성질을 가지고 있다.

http 주요 객체

- http.IncomingMessage
  - 클라이언트의 요청을 처리하는 객체
  - CreateServer의 콜백함수에서 첫번째 매개변수로 제공하는 객체
- http.Server
  - HTTP 서버를 구현하는 데 기조가 되는 프레임워크를 제공하는 객체
  - 서버를 시작하려면 createServer() 함수를 사용해 Server 객체를 생성해야 한다.

http 주요 객체

- http.ClientRequest 를 이용한 Get 방식과 Post 방식 요청

```
var http = require('http');
var options = {
    hostname: 'httpbin.org',
    path: '/ip'
};

function handleResponse(response) {
  var serverData = '';
  response.on('data', function (chunk) {
    serverData += chunk;
  });
  response.on('end', function () {
    console.log("received server data:");
    console.log(serverData);
  });
}

http.request(options, function(response){
  handleResponse(response);
}).end();
```

```
var http = require("http");
var options = {
  hostname: 'httpbin.org',
  path: '/post',
  method: 'POST',
  headers: {
    'Content-Type': 'text/html',
  }
};

var req = http.request(options, function(res) {
  console.log('Status: ' + res.statusCode);
  console.log('Headers: ' + JSON.stringify(res.headers));
  res.setEncoding('utf8');
  res.on('data', function (body) {
    console.log('Body: ' + body);
  });
});
req.on('error', function(e) {
  console.log('problem with request: ' + e.message);
});

req.write(
    '{"text": "test string"}'
);
req.end();
```

http 주요 객체

- http.Server 함수

함수	설명
close()	서버 연결 닫기
listen()	서버에서 시스템의 포트를 수신 대기하도록 설정
listening	서버가 연결을 수신하는 경우 true를 반환하고 그렇지 않으면 false
maxHeadersCount	수신 헤더 수 제한
setTimeout()	타이머 값 설정. 기본값은 2분
timeout	타이머 값 설정 또는 가져오기

http 주요 객체

- http.Server 함수 예제

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

var srvr = http.createServer(function (req, res) {
  res.write('Hello World!');
  res.end();
});
srvr.listen(2000);
console.log(srvr.listening); // true
srvr.close();  // 서버 종료
console.log(srvr.listening); // false
```

http 주요 객체

- http.IncomingMessage 함수

함수	설명
headers	헤더 이름 및 값을 포함하는 키 값 쌍 개체 반환
httpVersion	클라이언트에서 보낸 HTTP 버전 반환
method	요청 방법 반환
rawHeaders	요청 헤더 배열을 반환
rawTrailers	원시 요청 트레일러 키 및 값의 배열을 반환함
setTimeout()	지정된 시간(밀리초) 후에 지정된 함수를 호출함
statusCode	HTTP 응답 상태 코드 반환
socket	연결에 대한 소켓 개체 반환
trailers	트레일러를 포함하는 개체 반환
url	요청 URL 문자열을 반환함

http 주요 객체

- http.ServerResponse 함수

함수	설명
addTrailers()	HTTP 추적 헤더 추가
end()	서버가 응답이 완료되었다고 간주해야 하는 신호
finished	응답이 완료되면 true를 반환하고 그렇지 않으면 false
getHeader()	지정된 헤더 값을 반환함
headersSent	헤더가 전송된 경우 true를 반환하고 그렇지 않은 경우 false
removeHeader()	지정된 머리글 제거
sendDate	응답에서 날짜 헤더를 전송하지 않을 경우 false로 설정하십시오. 기본 참
setHeader()	지정된 머리글 설정
setTimeout	소켓의 시간 초과 값을 지정된 밀리초 수로 설정
statusCode	클라이언트로 보낼 상태 코드 설정
statusMessage	클라이언트로 보낼 상태 메시지 설정
write()	클라이언트로 텍스트 또는 텍스트 스트림 전송
writeContinue()	클라이언트에 HTTP Continue 메시지 보내기
writeHead()	상태 및 응답 헤더를 클라이언트로 보내기

http 주요 객체

- http.IncommingMessage , http.ServerResponse 예제

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

let server = http.createServer(function(req, res){

    let req_url = new URL(req.url, `http://${req.headers.host}`);
    let params = req_url.searchParams;

    res.writeHead(200, {'content-type':'text/html'});

    res.write("<!DOCTYPE html>");
    res.write("<html>");
    res.write("<head>");
    res.write("<meta charset='utf-8'>");
    res.write("<title>req, res 예제 </title>");
    res.write("</head>");
    res.write("<body>");

    if(req_url.pathname === '/'){
        res.write("<h1>root 입니다</h1>");
        res.write("<a href='test1?data=111'>test1</a></br>");
        res.write("<a href='test1?data=222'>test2</a></br>");
    }else if(req_url.pathname === '/test1'){
        if(params.get('data')==111){
            res.write("<h1>test1 페이지 입니다.</h1>");
        }else if(params.get('data')==222){
            res.write("<h1>test2 페이지 입니다.</h1>");
        }
    }

    res.write("</body>");
    res.write("</html>");
    res.end();
});

server.listen(2000);

server.on('listening', () => {
    console.log('서버 대기 중...  port : 2000');
});

server.on('error', (err) => {
    console.error(err);
});
```

http 주요 객체

- http.IncommingMessage , http.ServerResponse 예제