**Chapter 05. 웹 서버 만들기**

* 간단한 웹 서버 만들기
* 웹 서버 : 클라이언트(브라우저)에서 HTTP 프로토콜로 요청한 정보를 처리한 후, 응답을 보내주는 역할을 한다. 포트는 6만개의 포트 중 하나를 사용(여기서는 3000 사용)
* http 모듈의 createServer( )로 만ㅅ든 서버 객체는 listen( )메소드를 통해 서버를 호출하여 특정 포트에서 대기하도록 만든다.  
  🡪 listen(port, [hostname], [backlog], [callback]) : 서버를 실행  
   close([callback]) : 서버를 종료

[ch05\_test1.js]

var http = require('http');

// 웹 서버 객체를 생성

var server = http.createServer();

// 웹 서버 시작 후 3000번 포트에서 대기

var port = 3000;

server.listen(port, function(){

console.log('웹 서버가 시작되었습니다. : %d', port);

});

* [hostname]은 서버의 특정 IP를 지정해 실행 시킬 때 사용하고, [backlog]는 동접 수 결정

var host = '192.168.0.6';

var port = 3000;

server.listen(port, host, '50000', function(){

console.log('웹 서버가 시작되었습니다 : %s, %d', host, port);

});

* 클라이언트가 웹 서버에 요청 시 발생하는 이벤트

웹 브라우저가 서버에 접속해 데이터를 요청(req)하면 해당 이벤트가 발생하므로, 해당 이벤트에 맞는 콜백을 등록한다.   
🡪 대표적인 서버 객체 이벤트는 connection, request, close.

var http = require('http');

// 웹 서버 객체 생성

var server = http.createServer();

// 웹 서버 시작

var port = 3000;

server.listen(port, function(){

console.log('웹 서버가 시작되었습니다. : %d', port);

});

// 클라이언트 연결 이벤트 처리

// connection 발생 시, 클라이언트의 연결 정보는 socket으로 전달

server.on('connection', function(socket){

var addr = socket.address(); // 클라이언트 IP, Port 확인을 위한 객체

console.log('클라이언트가 접속했습니다 : %s, %d', addr.address, addr.port);

});

// 클라이언트 요청 이벤트 처리

server.on('request', function(req, res){

console.log('클라이언트 요청이 들어왔습니다.');

console.dir(req);

// 이 부분에 res.write()를 통해 클라이언트에 텍스트 response 가능!

});

// 서버 종료 이벤트 처리

server.on('close', function(){

console.log('서버가 종료됩니다.');

});

* request이벤트에 대한 처리를 아래와 같이 해, 클라이언트 상에 텍스트를 띄움

// 클라이언트 요청 이벤트 처리

server.on('request', function(req, res){

console.log('클라이언트 요청이 들어왔습니다.');

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

res.write("<!DOCTYPE html>");

res.write("<html>");

res.write(" <head>");

res.write(" <title>응답 페이지</title>");

res.write(" </head>");

res.write(" <body>");

res.write(" <h1>Node.JS로부터의 응답페이지</h1>");

res.write(" </body>");

res.write("</html>");

res.end();

});

* 클라이언트 요청 시 파일 읽어 응답하기

// 클라이언트 요청 이벤트 처리

server.on('request', function(req, res){

console.log('클라이언트 요청이 들어왔습니다.');

var filename = 'test.jpg';

fs.readFile(filename, function(err, data){

res.writeHead(200, {"Content-Type": "image/jpg"});

res.write(data);

res.end();

});

});

* 스트림으로 파일 읽어 응답 보내기

// 클라이언트 요청 이벤트 처리

server.on('request', function(req, res){

console.log('클라이언트 요청이 들어왔습니다.');

var filename = 'test.jpg';

var infile = fs.createReadStream(filename, {flags: 'r'});

infile.pipe(res);

});

* 파일 버퍼에 담아두고 일부분만 읽어 응답 보내기

파일을 스트림으로 읽은 후, 일정 크기의 데이터를 읽어 응답을 보내는 방식. 스트림에서 파일을 읽을 때 버퍼의 크기를 정하면 파일을 버퍼 크기만큼씩 읽어 응답 가능

// 클라이언트 요청 이벤트 처리

server.on('request', function(req, res){

console.log('클라이언트 요청이 들어왔습니다-버퍼 스트림 테스트');

var filename = 'test.jpg';

var infile = fs.createReadStream(filename, {flags: 'r'});

var filelength = 0;

var curlength = 0;

// fs.stat() 메소드는 해당 파일의 information을 확인 가능.

fs.stat(filename, function(err, stats){

// 읽어들여야 할 파일의 전체 사이즈를 확인해 filelength에 대입

filelength = stats.size;

});

// 헤더 쓰기

infile.on('readable', function(){

var chunk;

while(null !== (chunk = infile.read())){

console.log('읽어들인 데이터의 크기 : %d 바이트', chunk.length);

curlength += chunk.length;

res.write(chunk, 'utf8', function(err){

console.log('파일 부분 쓰기 완료 : %d, 파일크기 : %d', curlength, filelength);

if(curlength >= filelength);{

res.end();

}

});

}

});

});

* 서버에서 다른 웹 사이트의 데이터를 가져와 응답하기

http 모듈은 서버 뿐 아니라 ‘클라이언트’로서의 기능도 제공하므로, **HTTP클라이언트가 GET과 POST 방식으로 다른 웹 서버에 데이터를 요청할 수 있다**.

HTTP request 포맷은 Header와 Body로 나뉘는데, **GET 방식은 Header에 요청 정보를 넣어 보내고 POST는 Body 부분에 요청 정보를 넣어 보낸다.** 보안 이슈, 파일 업로드 등 파일을 요청 정보로 넣을 경우 POST 방식을 사용한다.

[GET 방식 요청 – ch05\_test7.js]

**http.get( ) 메소드를 사용해 요청 정보를 전달해 res를 받는다**

var http = request('http');

var options = {

host: 'www.google.com',

port: 80,

path: '/'

};

// http.get()메소드로 다른 사이트에 정보 요청

var req = http.get(options, function(res){

// 응답 처리

var resData = '';

res.on('data', function(chunk){ // response 받은 데이터를 chunk 파라미터에 전달한다.

resData += chunk;

});

res.on('end', function(){

console.log(resData);

})

});

req.on('error', function(err){

console.log('오류 발생 :' + err.message);

});

[POST 방식 요청 – ch05\_test8.js]

var http = require('http');

var opts = {

host: 'www.google.com',

port: 80,

method: 'POST',

path: '/',

headers: {}

};

var resData = '';

var req = http.request(opts, function(res){

// 응답처리

res.on('data', function(chunk){

resData += chunk;

});

res.on('end', function(){

console.log(resData);

});

});

opts.headers['Content-Type'] = 'application/x-www-form-urlencoded';

req.data = 'q=actor';

opts.headers['Content-Length'] = req.data.length;

req.on('error', function(err){

console.log('오류 발생 : ' + err.message);

});

// 요청 전송

req.write(req.data);

req.end();

* 요청 파라미터는 req 객체의 data 속성으로 설정한다. Write( ) 메소드로 요청하고자 하는 본문 데이터를 req 객체에 쓴 후 end( ) 메소드로 전송한다.