**Chapter 05. 웹 서버 만들기**

* 익스프레스로 웹 서버 만들기
* 미들웨어와 라우터로 좀 더 편리하게 서버 개발 가능

[app.js]

var express = require('express');

var http = require('http');

// express 서버 객체 생성

var app = express();

// 기본 포트를 app 개체에 속성으로 설정

app.set('port', process.env.PORT || 3000);

// express 서버 시작

http.createServer(app).listen(app.get('port'), function(){

console.log('익스프레스 서버를 시작했습니다. : ' + app.get('port'));

});

* Express 객체 app을 만들고, 이를 createServer메소드의 파라미터로 전달한다.
* **Express 서버 객체의 주요 메소드**

1. set(name, value) : 서버 설정을 위한 속성을 지정하며 get( )으로 불러낼 수 있다.  
   set 속성 name은 env(서버 모드), views, view engine 등이 있다.
2. get(name) : get(‘port’)처럼 name을 설정해 value를 불러올 수 있다.
3. use([path,]function[,function…]) : 미들웨어 함수를 사용
4. get([path,] function) : 특정 패스로 요청된 정보를 처리한다.(라우팅)

* 미들웨어로 클라이언트에 응답 보내기
* **미들웨어** : **use( ) 함수를 사용해** 특정 함수를 등록해서 **모든 요청이 해당 미들웨어를 거치도록** 미리 설정하는 것. 복수의 미들웨어를 설정하고 next( )를 이용해 순서대로 거칠 수 있도록 설정 가능하다.  
  🡪 라우팅은 후반 챕터에서 별도로 학습!

[app2.js]

var express = require('express');

var http = require('http');

var app = express();

app.use(function(req, res, next){

console.log('첫 번째 미들웨어에서 요청을 처리함');

res.writeHead('200', {'Content-Type': 'text/html;charset=utf8'});

res.end('<h1>Express 서버에서 미들웨어로 응답한 결과입니다.</h1>');

});

http.createServer(app).listen(3000, function(){

console.log('Express 서버가 3000번 포트에서 시작됨.');

});

* Express 서버 객체 app에서 use( ) 메소드로 미들웨어를 설정함
* 여러 개의 미들웨어 등록하여 사용

[app3.js]

app.use(function(req, res, next){

console.log('첫 번째 미들웨어에서 요청을 처리함');

req.user = 'IZONE';

next();

});

app.use('/', function(req, res, next){

console.log('두 번째 미들웨어에서 요청을 처리함');

res.writeHead('200', {'Content-Type': 'text/html;charset=utf8'});

res.end('<h1>Express 서버에서 두 번째 미들웨어로 ' + req.user + '이 응답함</h1>');

});

* 참고로 이 미들웨어 안에서는 미들웨어 함수를 호출한 app 객체도 req.app을 사용해 호출할 수 있다.
* 익스프레스의 응답객체

・ res.send([body]) : 클라이언트에 응답 데이터를 보낸다. HTML 문자열, Buffer & JSON 객체, JSON 배열

app.use(function(req, res, next){

console.log('첫 번째 미들웨어에서 요청을 처리함');

res.send({name: '아이즈원', age: 20});

});

* 클라이언트에서 localhost…호출 시 화면에 {"name":"아이즈원","age":20} 출력

・ res.status( ) : 상태 코드를 작성하는 기능만 있음  
 e.g.) res.status(403).send(‘Forbidden’);

・ res.sendStatus( ) : 코드만 파라미터로 전달

・ res.redirect( ) : 다른 주소 또는 프로젝트 폴더 내 다른 페이지 전달

app.use(function(req, res, next){

console.log('첫 번째 미들웨어에서 요청을 처리함');

res.redirect('http://google.com');

});

・ res.render( ) : 뷰 엔진을 사용해 템플릿 파일에 있는 내용을 HTML 페이지로 변환

* 익스프레스에서 요청 객체에 추가한 헤더 & 파라미터

클라이언트에서는 요청 파라미터를 함께 (GET/POST) 보낼 수 있다. GET방식일 경우, 요청 파라미터들은 req 객체의 query 객체안에 들어가게 된다.  
e.g.)<http://localhost:3000/?name=mike> 접속할 경우, 요청 파라미터(query string)인 name의 값을 req 객체의 query에 넣고, 요청하는 헤더 값은 header( ) 메소드로 확인이 가능하다.

app.use(function(req, res, next){

console.log('첫 번째 미들웨어에서 요청을 처리함');

var userAgent = req.header('User-Agent');

var paramName = req.query.name; // /?name=mike

res.writeHead('200', {'Content-Type':'text/html;charset=utf8'});

res.write('<h1>Express 서버에서 응답한 결과입니다.</h1>');

res.write('<div><p>User-Agent : ' + userAgent + '</p></div>');

res.write('<div><p>Param name : ' + paramName + '</p></div>');

res.end();

});