6. NodeJS

영진전문대학
컴퓨터정보계열
김종율

학습내용

- Node.js
  - https://nodejs.org/en/
- Express
  - https://expressjs.com/ko/

Node.js 런타임 설치

- Node.js 사이트 접속 및 다운로드
  - https://nodejs.org/
- 설치확인
  - cmd창에서 node -v 입력후 엔터
- node repl(read evaluate print loop) 실행
  - cmd창에서 node 입력후 엔터
  - 프롬프트가 '>'으로 변경된 후 자바스크립트 코드 실습해보기

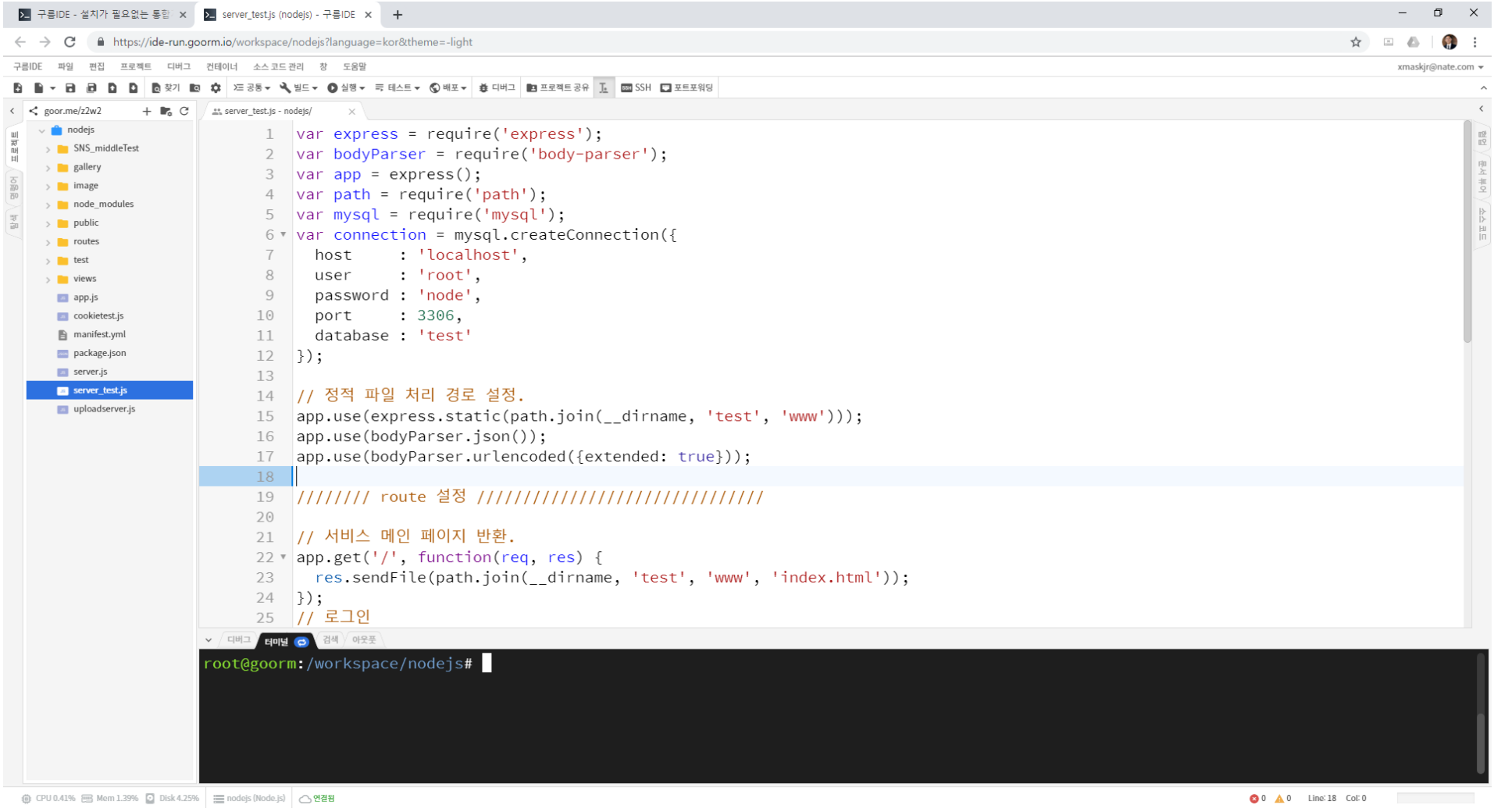
Node.js 웹 프레임워크 Express환경 설정

- Express 애플리케이션 생성기 설치
  - npm install express-generator –g
- 설치후 Express 애플리케이션 생성기의 명령어로 Expresss 애플리케이션을 위한 프로젝트 생성
  - express myapp
  - express --view=pug myapp
- 생성된 프로젝트 폴더로 이동
  - cd myapp
- 프로젝트 폴더에서 express환경 설치
  - npm install
- 설치확인
  - set DEBUG=myapp:*; npm start
- 브라우저에서 실행
  - url: http://localhost:3000/

용어정리

- Server
  - 원격지에 설치되어 접속하는 사용자(Client)에게 서비스를 제공하는 컴퓨터
  - Request-Response
- Server-side Language
  - Server에서 해석되고 실행되는 언어
  - PHP, JSP, ASP, node.js, Ruby 등
- Client-side Language
  - 사용자 쪽에서 해석되고 실행되는 스크립트 언어
  - 대표: JavaScript

goorm.io


구름IDE - 설치가 필요없는 통합
server_test.js (nodejs) - 구름IDE
https://ide-run.goorm.io/workspace/nodejs?language=kor&theme=-light
server_test.js - nodejs/
nodejs
SNS_middleTest
gallery
image
node_modules
public
routes
test
views
app.js
cookietest.js
manifest.yml
package.json
server.js
server_test.js
uploadserver.js
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
var path = require('path');
var mysql = require('mysql');
var connection = mysql.createConnection({
host : 'localhost',
user : 'root',
password : 'node',
port : 3306,
database : 'test'
});
// 정적 파일 처리 경로 설정.
app.use(express.static(path.join(__dirname, 'test', 'www')));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));
//////// route 설정 ////////////////////////////////
// 서비스 메인 페이지 반환.
app.get('/', function(req, res) {
res.sendFile(path.join(__dirname, 'test', 'www', 'index.html'));
});
// 로그인
root@goorm:/workspace/nodejs#

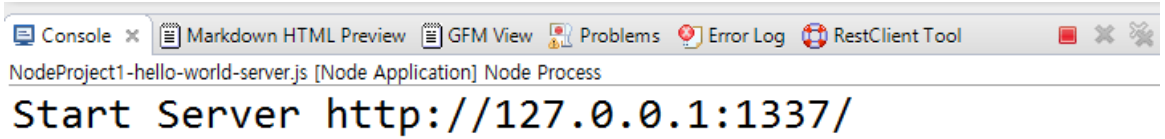
http module

- HTTP Server

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

1. http module의 import
2. 웹 서버 객체 생성(createServer)
   client로부터 request가 있을 때 처리하는 callback함수 정의
3. callback 함수 내부`
   1)header 설정(response code, 출력헤더)
   2)전송메시지를 파라미터로 전송 완료(end)
4. 생성된 웹서버 객체의 접속 대기 설정 (listen: 포트번호, 접속 IP)



http module

- HTTP Client: POST 방식
- 서버로 request전송:
  - http.request(options, callback);
    - options: object or string(url.parse()로 자동object 화 됨)
    - callback
    - http.ClientRequest instance return
      - writable stream
  - req.end()로 HTTP요청 완료 명시해야 함

http module

- options 내용
  - protocol: 기본값-http:, 사용되는 프로토콜
  - host: 기본값-localhost, 서버의 domain명이나 IP주소
  - hostname: url.parse()을 위한 옵션, host보다 더 선호되는 옵션
  - family: IP주소 family, 4 또는 6 (IP v4 or v6)
  - port: 기본값-80, 서버의 port 번호
  - localAddress: client의 IP
  - socketPath: Unix 소켓, host:port나 socketPath둘 중에 하나 사용
  - method: 기본값-GET, HTTP request방식 지정
  - path: 기본값-/, 요청경로를 지정(쿼리문장도 포함함, /index.html?p=12)
  - headers: request header에 포함되는 객체
  - auth: 기본 인증(user:password)
  - agent: Agent(Socket pool관련)제어, Agent가 사용되면 request 는 Connection: keep-alive로 설정됨
    - undefined: 기본값, host:port를 위한 global Agent 사용
    - Agent object: 전달받은 Agent를 명시적으로 사용함
    - false: pool사용 안함. request는 Connection: close으로 설정됨
  - createConnection: agent 옵션이 사용불가시 request를 위해 사용할 socket/stream을 생성하는 함수 지정
  - timeout: socket timeout 지정

http module

- options 내용

`options` `<Object>` | `<string>`

- `protocol` `<string>` Protocol to use. Defaults to `http:`.
- `host` `<string>` A domain name or IP address of the server to issue the request to. Defaults to `localhost`.
- `hostname` `<string>` Alias for `host`. To support `url.parse()`, `hostname` is preferred over `host`.
- `family` `<number>` IP address family to use when resolving `host` and `hostname`. Valid values are `4` or `6`. When unspecified, both IP v4 and v6 will be used.
- `port` `<number>` Port of remote server. Defaults to 80.
- `localAddress` `<string>` Local interface to bind for network connections.
- `socketPath` `<string>` Unix Domain Socket (use one of host:port or socketPath).
- `method` `<string>` A string specifying the HTTP request method. Defaults to `'GET'`.
- `path` `<string>` Request path. Defaults to `'/'`. Should include query string if any. E.G. `'/index.html?page=12'`. An exception is thrown when the request path contains illegal characters. Currently, only spaces are rejected but that may change in the future.
- `headers` `<Object>` An object containing request headers.
- `auth` `<string>` Basic authentication i.e. `'user:password'` to compute an Authorization header.
- `agent` `<http.Agent>` | `<boolean>` Controls `Agent` behavior. Possible values:
  - `undefined` (default): use `http.globalAgent` for this host and port.
  - `Agent` object: explicitly use the passed in `Agent`.
  - `false`: causes a new `Agent` with default values to be used.
- `createConnection` `<Function>` A function that produces a socket/stream to use for the request when the `agent` option is not used. This can be used to avoid creating a custom `Agent` class just to override the default `createConnection` function. See `agent.createConnection()` for more details.
- `timeout` `<number>`: A number specifying the socket timeout in milliseconds. This will set the timeout before the socket is connected.

http module

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

var options = {
  hostname: '127.0.0.1',
  port: 1337,
  path: '/',
  method: 'POST'
};

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 (chunk) {
    console.log('BODY: ' + chunk);
  });
});

req.on('error', function(e) {
  console.log('problem with request: ' + e.message);
});

// write data to request body
req.write('11111data11111111111111\n');
//req.write('data1\n');
req.end();
```

```
STATUS: 200
HEADERS: {"content-type":"text/plain","date":"Wed, 15 Jul 2015 03
BODY: Hello World
```

서버에서 받은 상태코드

서버에서 받은 헤더 정보

서버 응답 인코딩 설정

서버 응답 정보 출력
data: 받은 정보가 있으면 발생하는 event

요청에 에러 발생시 처리

실제 데이터 요청

요청 종료

http module

- HTTP Client: GET방식: http.get(options,callback);

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

http.get("http://localhost:1337/",
        function(res) {
            console.log("Got response: " + res.statusCode);
            console.log('HEADERS: ' +
                    JSON.stringify(res.headers));
            res.setEncoding('utf8');
            res.on('data', function (chunk) {
                console.log('BODY: ' + chunk);
            });
        }
).on('error',
      function(e) {
        console.log("Got error: " + e.message);
      }
);
```

```
Got response: 200
HEADERS: {"content-type":"text/plain","date":"Wed, 15 Jul 2015 03:24:17
BODY: Hello World
```

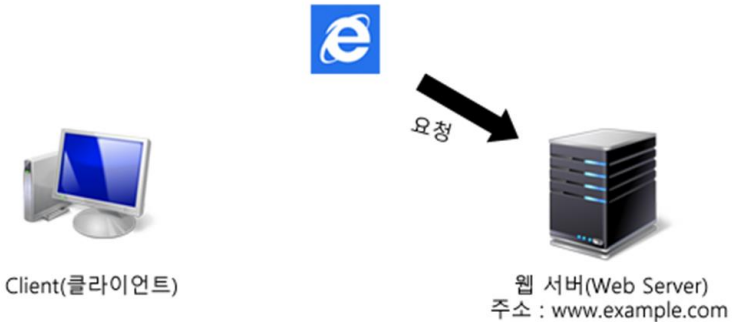
Web 동작원리

- Server와 Client가 존재. Client는 user를 뜻함

- Client가 Web Browser 주소창에 www.example.com를 입력하고 Enter

Web 동작원리

- Web Browser는 www.example.com의 주소에 있는 파일을 Request 함



- Web Server는 해당 주소에 있는 파일을 반환함. 보통 index.html이 이에 해당함

Web 동작원리

- Web Browser는 Server로 부터 받은 파일을 읽어 화면에 출력하여 user에게 보여줌

Response는 HTML이외에도 이미지나 동영상,
문서 등 모든 유형이 될 수 있음

Server-side Script 동작원리

- Client가 Web Browser 주소창에 www.example.com/publish/exam1.php를 입력하고 Enter

※ Server-side 언어의 실행
- Web Server에서 HTML로 변환하는 과정
- 이때 보통 Session이나 DB와 같은 Server 쪽 Resource를 이용

- Server는 Request한 URL에 있는 파일을 찾아 HTML 파일로 변환


```
<?php
    echo "<h1>Hello PHP</h1>";
?>
```

```
<html>
<head></head>
<body>
    <h1>Hello PHP</h1>
</body>
</html>
```

Server-side Script 동작원리

- 변환된 HTML 파일을 전송

- Web Browser는 Server로 부터 받은 파일을 읽어 화면에 출력하여 user에게 보여줌