Node.js

2019년 2월 20일 수요일 오후 6:24

아파치 웹 서버가 못하는 걸 Node.js가 할 수 있다.

response.writeHead(200); //파일 전송이 성공적일 때
response.writeHead(404); //파일을 찾을 수 없을 때

Template Literal

2019년 2월 20일 수요일 오후 7:01

```
var name = 'k8805';
// String literals
var letter = 'Dear '+name+'₩n₩nLorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. '+name+' Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa egoing qui officia deserunt mollit anim id est laborum. '+name;

// Template literals 그레이브 엑센트(`)와 ${}를 이용
var letter = `Dear ${name}
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. ${name} Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. ${1+1}
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa egoing qui officia deserunt mollit anim id est laborum. ${name}`;
console.log(letter);
```

URL

2019년 2월 20일 수요일 오후 7:22

Ex) http :// opentutorials.org : 3000 / main ? id=HTML
protocol host(domain) port path query string

---

main.js

```
var http = require('http');
var fs = require('fs');
var url = require('url'); // url모듈 사용

var app = http.createServer(function(request,response){
        var _url = request.url;
        var queryData = url.parse(_url, true).query; // query string 값을 객체형태로 가져온다. {id : HTML}
        console.log(queryData.id); // HTML을 출력
        if(_url == '/'){
                _url = '/index.html';
        }
        if(_url == '/favicon.ico'){
                return response.writeHead(404);
        }
        response.writeHead(200);
        response.end(queryData.id);
});
app.listen(3000);
```

Template Literal & URL

2019년 2월 20일 수요일 오후 9:19

<페이지 이동에 따른 동적 변화 - main.js>

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

var app = http.createServer(function(request,response){
    var _url = request.url;
    var queryData = url.parse(_url, true).query;
    var title = queryData.id;
    if(_url == '/'){  // WEB을 클릭했을 때 (=최상위URL로 이동했을 때)
      title = 'Welcome';
    }
    if(_url == '/favicon.ico'){
      response.writeHead(404);
      response.end();
      return;
    }
    response.writeHead(200);

/*html파일을 변수에 담아서 페이지 이동에 따라 페이지가 동적으로 바뀌게 함 --> 아직 본문내용(<p>태그 안 내용)이 바뀌지는 않음*/
    var template = `
    <!doctype html>
    <html>
    <head>
      <title>WEB1 - ${title}</title>
      <meta charset="utf-8">
    </head>
    <body>
      <h1><a href="/">WEB</a></h1>
      <ol>
        <li><a href="/?id=HTML">HTML</a></li>
        <li><a href="/?id=CSS">CSS</a></li>
        <li><a href="/?id=JavaScript">JavaScript</a></li>
      </ol>
      <h2>${title}</h2>
      <p><a href="https://www.w3.org/TR/html5/" target="_blank" title="html5 speicification">Hypertext Markup Language (HTML)</a> is the standard markup language for <strong>creating <u>web</u> pages</strong> and web applications.Web browsers receive HTML documents from a web server or from local storage and render them into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.
      <img src="coding.jpg" width="100%">
      </p><p style="margin-top:45px;">HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects, such as interactive forms, may be embedded into the rendered page. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets.
      </p>
    </body>
    </html>
    `;
```

```
    response.end(template);

});
app.listen(3000);
```

파일 읽기

2019년 2월 20일 수요일 오후 9:33

```
var fs = require('fs');
fs.readFile('sample.txt', 'utf8', function(err, data){
console.log(data);
});
```

이걸 이용하여 적용

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

var app = http.createServer(function(request,response){
    var _url = request.url;
    var queryData = url.parse(_url, true).query;
    var title = queryData.id;
    if(_url == '/'){
      title = 'Welcome';
    }
    if(_url == '/favicon.ico'){
      response.writeHead(404);
      response.end();
      return;
    }
    response.writeHead(200);
    fs.readFile(`data/${queryData.id}`,'utf8',function(err, description){
      var template = `
      <!doctype html>
      <html>
      <head>
        <title>WEB1 - ${title}</title>
        <meta charset="utf-8">
      </head>
      <body>
        <h1><a href="/">WEB</a></h1>
        <ol>
          <li><a href="/?id=HTML">HTML</a></li>
          <li><a href="/?id=CSS">CSS</a></li>
```

```
            <li><a href="/?id=JavaScript">JavaScript</a></li>
          </ol>
          <h2>${title}</h2>
          <p>
            ${description}
          </p>
        </body>
        </html>
        `;
        response.end(template);
    });
});
app.listen(3000);
```

콘솔에서의 입력 값

2019년 2월 20일 수요일 오후 11:08

콘솔에서..

node example.js 2 //2가 입력값

example.js 내용..

```
var args = process.argv; //배열의 형태로 반환됨
console.log(args[2]); //3번째 값이 입력 값임.
```

조건문(URL)

2019년 2월 21일 목요일 오후 6:23

var _url = request.url;
url.parse{_url, true} --> 아래와 같은 형태

```
url{
  protocol : null,
  slashes : null,
  auth : null,
  host : null,
  port : null,
  hostname : null,
  hash : null,
  search : '?id=CSS',
  query : {id : 'CSS'},
  pathname : '/',
  path : '/?id=CSS',
  href : '/?id=CSS'
}
```

이 중 pathname을 이용하여 조건문을 만든다.
pathname은 query가 있어도 query를 뺀 것

파일 목록 알아내기

2019년 2월 21일 목요일 오후 7:41

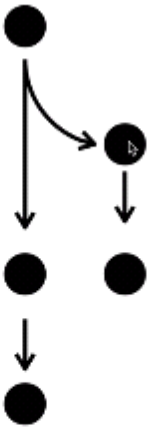
```
var testFolder = './data'; //경로는 실행시키는 파일 위치가 아니라 콘솔 창 현재 위치(G:₩Node.js)
var fs = require('fs');

fs.readdir(testFolder, function(error, filelist){
  console.log(filelist);
})
```

<적용시킨 main.js>

```
fs.readdir('./data', function(error, filelist){
        var title = 'Welcome';
        var description = 'Hello, Node.js';
        var list = '<ul>';
        var i = 0;
        while(i < filelist.length){
          list = list + `<li><a href="/?id=${filelist[i]}">${filelist[i]}</a></li>`;
          i = i + 1;
        }
        list = list+'</ul>';
        var template = `
        <!doctype html>
        <html>
        <head>
          <title>WEB1 - ${title}</title>
          <meta charset="utf-8">
        </head>
        <body>
          <h1><a href="/">WEB</a></h1>
          ${list}
          <h2>${title}</h2>
          <p>${description}</p>
        </body>
        </html>
        `;
        response.writeHead(200);
        response.end(template);
      })
```

동기와 비동기

2019년 2월 21일 목요일 오후 8:13



synchronous

asynchronous

<동기 vs 비동기>

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

//readFileSync 동기적
/*
console.log('A');
var result = fs.readFileSync('sample/sample.txt','utf8');
console.log(result);
console.log('C');
*/
//결과 : A B C

//readFile 비동기적
console.log('A');
//3번째 인자는 callback 함수 --> 파일을 읽어오는 작업이 끝나면 실행되는 함수
fs.readFile('sample/sample.txt','utf8',function(err,result){
  console.log(result);
});
console.log('C');
//결과 : A C B

//비동기적인게 좋음
```

callback

2019년 2월 21일 목요일 오후 8:57

```
/*
function a(){
  console.log('A');
}
a();
*/
//같은 의미
var a = function(){
  console.log('A');
}
//a();

function slowfunc(callback){
  callback();
}
slowfunc(a);
```

NPM(패키지 매니저)

2019년 2월 21일 목요일 오후 8:58

PM2

- 의도치 않게 실행중인 프로그램(프로세스) 가 꺼졌을 때 다시 실행시켜줌
- 프로그램 수정 시 자동으로 껐다 켜줌

http://pm2.keymetrics.io/ 참고(설치과정 필요)

`pm2 start main.js --watch` : 프로그램 수정 시 자동으로 껐다 켜주는 명령

`pm2 log` : 실시간으로 실행되거나 오류를 보여줌

HTML - Form

2019년 2월 21일 목요일 오후 9:37

웹 브라우저에서 서버로 데이터를 전송할 때 사용하는 기능

```
<form action="http://localhost:3000/process_create" method="post"> // 주소의 서버로  post 형식으로 데이터 전송
<p><input type="text" name="title"></p> // name을 꼭 해줘야 query가 생성될 때 그 이름으로 전송이 됨.
<p>
<textarea name="description"></textarea>
</p>
<p>
<input type="submit">
</p>
</form>
```

이게 나중에 query가 되어 http://localhost:3000/process_create/?title=...&description=... 가 됨.
그러나, 주소에 데이터가 포함되어 있는 것은 굉장히 위험
따라서 데이터를 전송(수정, 삭제, 생성)할 땐, post형식을 사용하는 것!! default값은 get (서버로부터 가져올 때)

앞에서 http://localhost:3000/?id=HTML --> 이것은 사용자가 입력하는 데이터 값이 아니라 링크에 대한 정보이므로 괜찮음

<input type="hidden" name="id" value="${title}"> --> hidden 속성을 주면 보이지 않음.

POST 방식으로 전송된 데이터 받기

2019년 2월 22일 금요일 오후 10:02

```
var qs = require('querystring');

else if (pathname === '/create_process') {
        var body = '';
        request.on('data',function(data){
          body += data;
        });
        request.on('end',function(){
          var post = qs.parse(body); //객체 형태로 반환
          var title = post.title;
          var description = post.description;
        });
        response.writeHead(200);
        response.end('success');
    }
```

파일 쓰기 & redirection

2019년 2월 22일 금요일 오후 10:04

```
fs.writeFile(`data/${title}`, description, 'utf8',function(err){ // 인자로 경로, 내용, 형식, 콜백함수
        response.writeHead(302,{Location: `/?id=${title}`}); // redirection할 땐 302
        response.end();
      });
```

파일 이름 수정하고 파일 쓰기

2019년 2월 22일 금요일 오후 10:12

```
fs.rename(`data/${id}`,`data/${title}`,function(err){ // 이전경로, 바뀔 경로, callback
        fs.writeFile(`data/${title}`, description, 'utf8',function(err){
          response.writeHead(302,{Location: `/?id=${title}`});
          response.end();
      });
```

주의사항

2019년 2월 22일 금요일 오후 11:36

삭제하는 기능은 링크로 구현하면 안됨. (매우 위험)

<form>의 <input> 버튼구현

파일 삭제

2019년 2월 22일 금요일 오후 11:48

```
else if (pathname === '/delete_process') {
        var body = '';
        request.on('data',function(data){
          body += data;
        });
        request.on('end',function(){
          var post = qs.parse(body);
          var id = post.id;
          fs.unlink(`data/${id}`,function(err){
            response.writeHead(302,{Location: `/`});
            response.end();
          });
        });
    }
```

자바 스크립트(객체&배열)

2019년 2월 23일 토요일 오후 10:05

객체 vs 배열

<배열>

```
var members = ['egoing', 'k8805', 'hoya'];
console.log(members[1]); // k8805
var i = 0;
while(i < members.length){
console.log('array loop', members[i]);
i = i + 1;
}
```

<객체>

```
var roles = {
'programmer':'egoing', // key : value
'designer' : 'k8805',
'manager' : 'hoya'
}
console.log(roles.designer); //k8805
console.log(roles['designer']); //k8805
for(var n in roles){
console.log('object => ', n, 'value => ', roles[n]);
}
```

객체 이용 코드 정리

2019년 2월 23일 토요일 오후 10:08

```
var template = {
HTML : function(title, list, body, control){
return `
<!doctype html>
<html>
<head>
<title>WEB1 - ${title}</title>
<meta charset="utf-8">
</head>
<body>
<h1><a href="/">WEB</a></h1>
${list}
${control}
${body}
</body>
</html>
`;
},list : function(filelist){
var list = '<ul>';
var i = 0;
while(i < filelist.length){
list = list + `<li><a href="/?id=${filelist[i]}">${filelist[i]}</a></li>`;
i = i + 1;
}
list = list+'</ul>';
return list;
}
}
```

모듈

2019년 2월 24일 일요일 오전 12:52

기능들을 별도의 파일에 저장하고 require로 이용하는 방법 - 정리 정돈의 큰 틀
- 하나만 exports할 때는 module.exports
- 여러 개를 exports할 때는 exports

<nodejs/mpart.js>

```
var M = {
v:'v',
f:function(){
console.log(this.v);
}
}
module.exports = M;
```

OR

<nodejs/mpart.js>

```
module.exports = {
v:'v',
f:function(){
console.log(this.v);
}
}
```

<muse.js>

```
var part = require('./mpart.js');
part.f();
```

<topic.js>

```
var db = require('./db.js');
var template = require('./template.js');
exports.home = function(request,response){
  db.query(`SELECT * FROM topic`,function(error,topics){
    if(error){
      throw error;
    }
    var title = 'Welcome';
    var description = 'Hello, Node.js';
    var list = template.list(topics);
```

```
    var html = template.HTML(title,list,
      `<h2>${title}</h2>${description}`,
      `<a href="/create">create</a>`);
    response.writeHead(200);
    response.end(html);
  });
}
```

입력 정보에 대한 보안

2019년 2월 24일 일요일 오후 1:40

<url 경로의 위험성>

예를 들어 사용자의 중요한 정보가 담긴 파일 password.js 가 존재한다고 하면
url에 localhost:3000/?id=../password.js 라고 친다면 ../는 상위 디렉토리에서부터 검색하기 때문에 사용자의 정보가 노출될 위험이 있다.

이걸 해결하기 위해 path 모듈을 이용.
파일을 읽어들이는 fs.readFile에서
path.parse().base를 사용해서 ../가 적용되지 않게 한다.

```
path.parse('/home/user/dir/file.txt');
// Returns:
// { root: '/',
//   dir: '/home/user/dir',
//   base: 'file.txt',
//   ext: '.txt',
//   name: 'file' }
```

```
var filteredId = path.parse(queryData.id).base; //사용자가 입력할 수 있는 url 창으로부터 받은 queryData.id에 ../ 이런 게 있어도 base만 입력됨.
fs.readFile(`data/${filteredId}`, 'utf8', function(err, description){
        .....
}
```

출력 정보에 대한 보안

2019년 2월 24일 일요일 오후 2:51

npm sanitize-html : html태그들을 무력화시킴.
https://www.npmjs.com/package/sanitize-html

npm init
npm install -S sanitize-html (-S는 -g와 달리 이 프로젝트 내에서만 사용하기 위한 설치 옵션)

<사용 방법>

```
var sanitizeHtml = require('sanitize-html');

var dirty = 'some really tacky HTML';
var clean = sanitizeHtml(dirty); // script 태그일 경우에는 아예 날려버리고 일반 html태그는 태그만 날리고 그 안에 내용은 살린다.
```

만약 html의 h1,h2태그의 기능은 살리고 싶다면,

```
ex) var sanitizedDescription = sanitizeHtml(description, { //두 번째 인자로 허용할 태그를 명시해주면 됨.
        allowedTags:['h1','h2']
    });
```

create 부분에 title과 description에 적용

PM2

2019년 2월 24일 일요일 오후 3:30

pm2 kill // pm2로 실행중인 프로그램 다 끔.

pm2 start main.js --watch --ignore-watch="data/* sessions/*" --no-daemon // : 이게 변경 되도 다시 껐다가 켜지지 않음.(난감한 상황 방지) : console.log와 일반 log를 같이 보여줌.

Mysql(삽입, 수정, 삭제)

2019년 2월 24일 일요일 오후 4:31

node.js에 mysql 모듈을 설치

https://www.npmjs.com/package/mysql

npm install --save mysql (package.json파일에 dependencies에 자동으로 기록)

<연결>

```
var mysql = require('mysql');
var db = mysql.createConnection({
host:'localhost',
user:'root',
password:'111111',
database:'opentutorials'
});
db.connect();
```

<DB에 저장되어 있는 값 사용>

```
db.query(`SELECT * FROM topic`, function(error,topics){
    if(error){
    throw error;
    }
    db.query(`SELECT * FROM topic WHERE id=?`,[queryData.id], function(error2, topic){ //?에 그 다음 인자 값이 자동으로 들어감.
        if(error2){
        throw error2;
        }
        var title = topic[0].title; //topic 의 결과는 한 줄씩 배열 값이므로 [0]을 해줘야 함.
        var description = topic[0].description;
        var list = template.list(topics);
        var html = template.HTML(title, list,
        `<h2>${title}</h2>${description}`,
        ` <a href="/create">create</a>
        <a href="/update?id=${queryData.id}">update</a>
        <form action="delete_process" method="post">
        <input type="hidden" name="id" value="${queryData.id}">
        <input type="submit" value="delete">
        </form>`
        );
        response.writeHead(200);
        response.end(html);
    })
});
```

<값을 DB에 저장>

```
var body = '';
request.on('data', function(data){
body = body + data;
});
request.on('end', function(){
var post = qs.parse(body); // POST형식으로 전달받은 값 저장
db.query(`
    INSERT INTO topic (title, description, created, author_id)
    VALUES(?, ?, NOW(), ?)`,
```

```
        [post.title, post.description, 1],  // ? ? ? 세 개의 ?에 각각 값을 지정해줌.
        function(error, result){
            if(error){
            throw error;
            }
            response.writeHead(302, {Location: `/?id=${result.insertId}`}); //삽입된 행의 primary key값을 가져옴.
            response.end();
        })
});

<값 수정>
var body = '';
request.on('data', function(data){
        body = body + data;
});
request.on('end', function(){
        var post = qs.parse(body);
        db.query('UPDATE topic SET title=?, description=?, author_id=1 WHERE id=?', [post.title, post.description, post.id], function(error, result){
            response.writeHead(302, {Location: `/?id=${post.id}`});
            response.end();
        })
});

<값 삭제>
var body = '';
request.on('data', function(data){
        body = body + data;
});
request.on('end', function(){
        var post = qs.parse(body);
        db.query('DELETE FROM topic WHERE id = ?', [post.id], function(error, result){
            if(error){
            throw error;
            }
            response.writeHead(302, {Location: `/`});
            response.end();
        });
});
```

Mysql(조인)

2019년 2월 24일 일요일 오후 11:26

SELECT * FROM topic LEFT JOIN author ON topic.author_id=author.id WHERE topic.id=?

보안(SQL Injection)

2019년 2월 27일 수요일 오후 6:02

외부에서 들어온 정보는 오염되어 있다고 가정하는 것이 좋음.

SQL Injection : SQL문을 주입하는 공격

<필터링 기법>

보안 방법1) 공격자가 id=(입력값)에 위험한 쿼리 문을 입력해도 문자열이 됨.

```
db.query(`SELECT * FROM author WHERE id=?`, [queryData.id], function(error2, author) {
        ...
});
```

보안 방법2)

```
db.query(`SELECT * FROM author WHERE id=${db.escape(queryData.id)}`, function(error2, author) {
        ...
});
```

ex) http://localhost:3000/?id=1; DROP TABLE topic; (굉장히 위험)
그러나 오류가 뜸. 왜냐면 기본적으로 db.query는 한번에 여러 sql문을 실행 시킬 수 없기 때문.

보안(Escaping)

2019년 2월 27일 수요일 오후 7:10

출력 정보에 대한 보안에 이어서..

sanitizedHtml을 이용한 방법

사용자가 입력한 정보에 대한 처리