2 주차

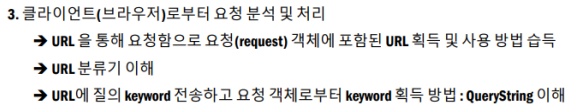
과제 - MYSQL 설치

과제 보낼때 node_modules까지 압축해서 보내면 용량이 너무 크므로 보내지마라. (감점 요소)

오늘 배울 내용들

1장 뒷부분 + 2장

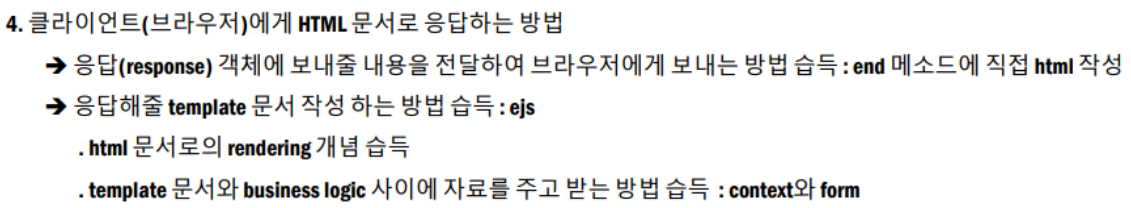
요청하기 (request)



URL을 획득하는 방법, URL 요청을 분류하는 방법

응답하기 (response)

기본적으로 html으로 응답한다.



+비지니스로직은 다음주 (3주차)

▼ 지난 시간 복습 (1장 내용)

```
var http = require('http'); //웹 서버 기능의 모듈
var fs = require('fs'); // 파일 처리 모듈

// http의 멤버 함수인 createServer 실행한게 app에 담겨져 있음
// 즉 app이 웹서버이다.
var app = http.createServer(function(request, response){ //function이 reque
    var url = request.url; //클라이언트가 요청한 url을 request객체로 부터 가져온다.
    if(request.url == '/'){ //루트 경로 인지 체크
        url = '/index.html'; // 루트경로면 05-index.html 페이지로 리다이렉트
    }
    if(request.url == '/favicon.ico'){ //favicon.ico는 브라우저가 요청하는 기본
        return response.writeHead(404);
    }
    response.writeHead(200); // 응답 헤더에 200 상태 코드 작성.
```

```
    console.log(__dirname + url); // __dirname은 현재 디렉토리 경로 (Node.js에서
    response.end(fs.readFileSync(__dirname + url)); //fs.readFileSync를 사용
    // end: 응답하는 액션에 해당하는 메서드
})
app.listen(3000); //request를 계속 듣기 위한 코드
```

```
var http = require('http') // 웹 서버 모듈인 'http' 모듈 import하기
```

```
var http = require('http') // 웹 서버 모듈인 'http' 모듈 import하기
var app = http.createServer(



)

app.listen(3000)
// 위 코드만으로 웹 서버가 실행된다.
// 요청은 콜백함수로 받아야한다.
```

```
var http = require('http') // 웹 서버 모듈인 'http' 모듈 import하기
var app = http.createServer(
    function(request, response){ //파라미터명은 맘대로 (req, res)도 가능

 }
)

app.listen(3000)
// function에 중요한 두가지는 요청 정보와 응답 내용이 있어야한다.
// 요청의 정보는 'request'에 있다. 이를 콜백함수의 첫번째 파라미터로 넘겨준다. (약속이다)
// 응답의 내용은 'response'이고 이를 두번째 파라미터로 넘겨준다. (약속)
```

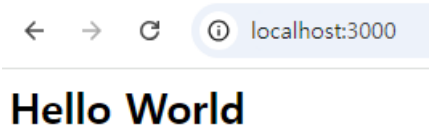
이 후

```
var url = request.url; // 코드를 통해서 url을 분류하는 과정을 진행한다.
```

```
response.end() // <- 이 부분에 String이 아니라 html개게를 넣어줄 수 있다.

// 응답에 html 객체를 실어서 응답을 보낸ㄷ.
response.end("<html><head></head><body><h1>Hello World</h1></body></html>")
```

실행 결과




```
/, /main, /main/side, /main?id=HTML&page=12 와 같은 단어들을 요청의 종류라고 한다.
```

여기서 ?: 이후에 나오는 것이 query string이다.

query string은 질의 단어, 조건을 제시하는 기능이다.

```
/ain?id=HTML&page=12 에서 &는 두 변수를 연결하는 역할이다.
//HTML 여러개 중에서 page 12에해당하는 걸 달라는 요청으로 추측할 수 있다.
```

---

loccalhost:3000/main 으로 들어오는 요청 처리하기

1장때 사용한 코드에 추가하여 loccalhost:3000/main 로 들어오는 요청을 처리한다.

```
var http = require('http'); //웹 서버 기능의 모듈
var fs = require('fs'); // 파일 처리 모듈

// http의 멤버 함수인 createServer 실행한게 app에 담겨져 있음
// 즉 app이 웹서버이다.
var app = http.createServer(function(request, response){ //function이 request
    var url = request.url; //클라이언트가 요청한 url을 request객체로 부터 가져온다.
    if(request.url == '/'){ //루트 경로 인지 체크
        url = '/index.html'; // 루트경로면 05-index.html 페이지로 리다이렉트
    }
    // 추가한 내용
    if(request.url == '/main'){
```

```
        url = '/index2.html'
    }
    if(request.url == '/favicon.ico'){ //favicon.ico는 브라우저가 요청하는 기본 아
        return response.writeHead(404);
    }
    response.writeHead(200); // 응답 헤더에 200 상태 코드 작성.
    console.log(__dirname + url); // __dirname은 현재 디렉토리 절대 경로 (홈 디렉토

    // response.end("<html><head></head><body><h1>Hello World</h1></body></ht

    response.end(fs.readFileSync(__dirname + url)); //fs.readFileSync를 사용해
    // end: 응답하는 액션에 해당하는 메서드
})
app.listen(3000); //request를 계속 듣기 위한 코드
```

요청(링크, URL)의 종류를 만드는 사람 - 서버

요청(링크, URL)의 종류를 사용하는 사람 - 클라이언트

URL에서 쿼리스트링 추출하기

url 모듈에는 parse라는 메서드를 사용한다. (파싱 기능)

```
var urlm = require('url')

// 위 코드는 url 모듈을 사용하겠다는 의미이다. (url 모듈은 urlm 이라는 변수에 저장)
// request 객체의 멤버 변수명으로서 url과 혼동하지말자. (url 정보가 들어있음 이를 _url에
```

```
var queryData = urlm.parse(_url, true).query // url 파싱 후 쿼리 객체 반환을 저장한
```

```
var http = require('http');
var fs = require('fs')
var urlm = require('url'); //url 모듈을 요청한다. url 모듈에 담긴 기능(parse)를 사용
var app = http.createServer(function(req, res){
    var _url = req.url;
    var queryData = urlm.parse(_url, true).query // url에서 quertstring 문자열을

    console.log(_url);
    console.log(queryData); //query.id로 바꾸어도 실행해보자
    console.log(queryData.id); //.id를 사용시 쿼리 객체의 내용을 출력할 수 있다.

    if(req.url =='/'){
```

```
        _url = '/index.html';
    }
    if(req.url =='/favicon.ico'){
        return res.writeHead(404);
    }
    res.writeHead(200);
    res.end(fs.readFileSync(__dirname + _url));
});
app.listen(3000);
```

---

http://localhost:3000/?id=HTML&name=wooseok 으로 url 입력시

```
    console.log(queryData.id); //.id를 사용시 쿼리 객체의 내용을 출력할 수 있다.
    console.log(queryData.name); //.id를 사용시 쿼리 객체의 내용을 출력할 수 있다.
```

위처럼 .id .name을 사용하여 출력할 수 있다.

```
[Object: null prototype] { id: 'HTML', name: 'wooseok' }
HTML
wooseok
```

다양한 메서드 사용해보기

.query 대신 .path .pathname 사용해보기

```
    var queryData = urlm.parse(_url, true).path // pathname, query을 포함 (전체
    var queryData = urlm.parse(_url, true).pathname // 쿼리스트링 앞부분
```

true 빼보기 → 객체가 아닌 문자열로 출력된다.

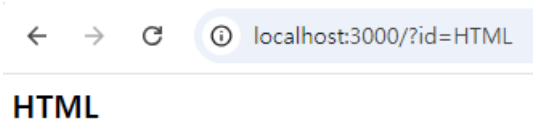
```
    var queryData = urlm.parse(_url).path // url에서 quertstring 문자열을 추출해
```

```
var http = require('http');
var fs = require('fs')
var urlm = require('url'); //url 모듈을 요청한다. url 모듈에 담긴 기능(parse)를 사용
var app = http.createServer(function(req, res){
    var _url = req.url;
    var queryData = urlm.parse(_url, true).query // url에서 quertstring 문자열을

    // console.log(_url);
```

```
    console.log(queryData); //query.id로 바꾸어도 실행해보자

    if(req.url =='/'){
        _url = '/index.html';
    }
    if(req.url =='/favicon.ico'){
        return res.writeHead(404);
    }
    res.writeHead(200);
    // res.end(fs.readFileSync(__dirname + _url));
    res.end("<html><body><h2>" + queryData.id + "</h2></body></html>")
});
app.listen(3000);
```



response 객체에 HTML 문서를 전달해주는 방법

1. end 메소드에 직접 html 작성
2. end 메소드에 fs 모듈을 이용해 디렉토리 경로에 html 파일을 읽어온다.
3. template 언어 이용 (맨 마지막에 할 예정, 좀 복잡함)

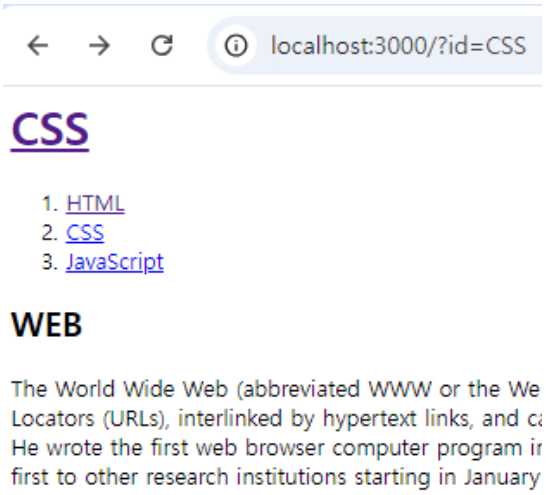
```
var http = require('http');
var fs = require('fs')
var urlm = require('url'); //url 모듈을 요청한다. url 모듈에 담긴 기능(parse)를 사용
var app = http.createServer(function(req, res){
    var _url = req.url;
    var queryData = urlm.parse(_url, true).query // url에서 quertstring 문자열을
    var title = queryData.id;
    // console.log(_url);
    console.log(queryData); //query.id로 바꾸어도 실행해보자

    if(req.url =='/'){
        _url = '/index.html';
        title = 'Welcom';
    }
    if(req.url =='/favicon.ico'){
```

```
        return res.writeHead(404);
    }
    res.writeHead(200);

    var template = `
    <html>
    <head>
        <title>WEB - ${title}</title>
    </head>
    <body>
    <h1><a href="index.html">${queryData.id}</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>WEB</h2>
    <p>The World Wide Web (abbreviated WWW or the Web) is an information spac
    </p>
  </body>
    </html>
    `;
    res.end(template);
});
app.listen(3000);
```

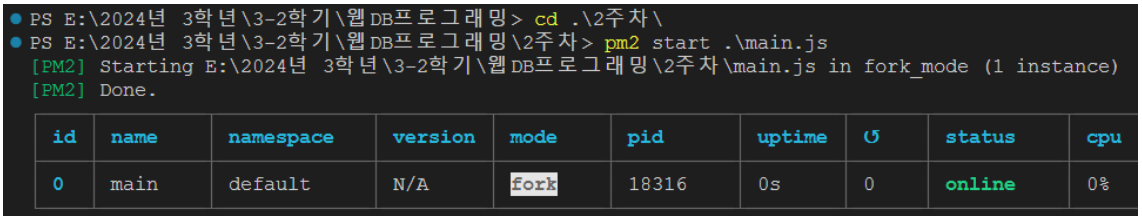
```
// 이코트 또한 사용자가 클릭시 요청으로 작용한다.
// 클릭시 localhost:3000/?id=HTML or CSS or JavaScript로 요청하는 거랑 같다.
          <li><a href="/?id=HTML">HTML</a></li>
      <li><a href="/?id=CSS">CSS</a></li>
      <li><a href="/?id=JavaScript">JavaScript</a></li>
```



pm2 설치

install pm2 -g

pm2 start “js명” — js 실행 시작



pm2 stop “js명” — js 실행 종료

변경 사항을 실시간 반영을 위해서

pm2 start “js명” --watch 를 사용한다.

Express 설치

디렉토리 구조

웹DB프로그래밍/1주차

웹DB프로그래밍/2주차

...

Express는 루트디렉토리인 웹DB프로그래밍에 설치

npm install express --save

Express로 서버 만들기

```
const express = require('express') // 1. express 모듈을 import한다. const에 의해
const app = express()//http.createServer() 와 비슷, express() 함수에 의해 Applica
app.get('/',(req, res)=>res.send('Hello World'))
app.get('/author',(req, res)=>res.send('/author'))
app.get('/main', (req, res)=>res.send("song wooseok"))
app.listen(3000, () => console.log('Example app listening on port 3000')) //
```

- get(path, callback): 사용자가 해당 경로를 요청하면 callback을 실행한다(화살표 함수 사용)
  get 메소드는 라우팅(URL 분류)기능을 수행한다. - 요청된 경로마다 응답해 주는 기능
  더 이상 if else로 URL을 분류할 필요가 없다.
- res.send는 end메소드와 같은 기능이다.

기존 코드 Express로 대체하기

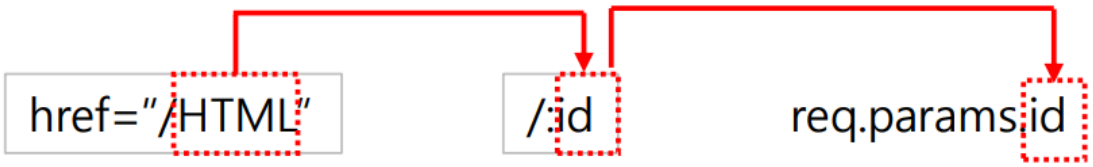
```
var express = require('express');
var app = express()
var urlm = require('url'); //url 모듈을 요청한다. url 모듈에 담긴 기능(parse)를 사용

app.get('/', (req, res) => {
    var _url = req.url;
    var queryData = urlm.parse(_url, true).query // url에서 quertstring 문자열을
    console.log(queryData); //query.id로 바꾸어도 실행해보자
    var title = query.id;


    var template = `
    <html>
    <head>
        <title>WEB - ${title}</title>
    </head>
    <body>
    <h1><a href="index.html">${title}</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>WEB</h2>
    <p>The World Wide Web (abbreviated WWW or the Web) is an information spac
    </p>
  </body>
    </html>
    `;
    res.send(template);
});

app.get('/favicon.icon', (req,res)=>res.writeHead(404));
app.listen(3000, ()=> console.log("Example app is listening on port 3000"));
```

/:id 사용하기 (시맨틱 URL 사용하기)




```
var express = require('express');
var app = express()
var urlm = require('url'); //url 모듈을 요청한다. url 모듈에 담긴 기능(parse)를 사용

app.get('/:id', (req, res) => {
    var id = req.params.id;
    title = 'welcome';
    console.log(id);
    var title =id;

    var template = `
    <html>
    <head>
        <title>WEB - ${title}</title>
    </head>
    <body>
    <h1><a href="index.html">${title}</a></h1>
    <ol>
      <li><a href="/HTML">HTML</a></li>
      <li><a href="/CSS">CSS</a></li>
      <li><a href="/JavaScript">JavaScript</a></li>
    </ol>
    <h2>WEB</h2>
    <p>The World Wide Web (abbreviated WWW or the Web) is an information spac
    </p>
  </body>
    </html>
    `;
    res.send(template);
});

app.get('/favicon.icon', (req,res)=>res.writeHead(404));
app.listen(3000, ()=> console.log("Example app is listening on port 3000"));
```

(원래 4주차 진행인데 진도는 2주차라 여기에 추가)

ejs태그를 사용하면 .html 파일이 .ejs로 바뀐다.

2. template 언어 문법

표기	의미
<% js 문법 %>	흐름 제어문
<%= %>	변수 값
<%- %>	변수 내용을 태그로 인식
<%- include(view의 상대주소) %>	다른 view파일을 불러 옴.

```
app.get('/:id',(req,res)=>{
        //파라미터 값을 먼저 꺼내온다.
    var id=req.params.id;

        // 저장한 파라미터를 json형식으로 context 객체에 저장
    var context = {title:id};

    res.render('home', context, (err,html)=>{
    // render -> 어떤 파일이든지 html로 만드는 것을 의미
        res.end(html)
    })
})
```

res.render() 메소드의 파라미터를 알아보자.

1. ejs 파일
2. ejs 파일에 전달할 data
3. response객체에 있는 end로 html(ejs파일)을 클라이언트에 전달한다.
   첫번째 파라미터인 ejs파일이 세번째 파라미터인 콜백함수의 html로 전달된다.

렌더 함수 때문에 사용자에게 보여지는 화면 html화면은 서버가 새롭게 만들어주는 html파일이다.
기존의 html의 틀을 유지한채 값만 바꿔끼는 방식이 아니다.

② res.render(view [, locals] [, callback]) : locals 위치에 있는 자료를 view에 넘겨주고 view를 html로 만들어서 클라이언트에 보내는 함수

3. ejs : main.js가 있는 폴더에 **views** 폴더 생성

▼ 위 코드 실습

main.js

```
const express = require('express');
const app = express();
app.set('views',__dirname+'/views') //뷰 디렉토리 경로 설정
app.set('view engine','ejs') //뷰 엔진으로 ejs를 사용하겠다는 뜻.

//루트 디렉토리 경로
app.get('/', (req,res)=>{
    var context = {title: 'welcome-1'}
    res.render('home', context, (err, html)=>{
        res.end(html)
    })
})

app.get('/:id',(req,res)=>{
    var id=req.params.id;

    var context = {title:id};

    res.render('home', context, (err,html)=>{
    // render -> 어떤 파일이든지 html로 만드는 것을 의미
    // 1. ejs 2.ejs에 전달할 data 3.만들어진 ejs파일(html)을 브라우저에 전달
        res.end(html)
    })
})
```

```
app.get('/favicon.ico',(req,res)=>res.writeHead(404));
app.listen(3000, ()=>console.log("Example app listening on port 3000"));
```

./views/home.ejs

```
<!DOCTYPE html>
<html>
<head>
    <title>Web1 - <%=title%></title>
    <meta charset="utf-8">
</head>
<body>
    <h1><a href="/">WEB</a></h1>
    <ol>
        <li><a href="/HTML">HTML</a></li>
        <li><a href="/CSS">CSS</a></li>
        <li><a href="/JS">Javasciprt</a></li>
    </ol>
    <hr>
    <h2><%=title%></h2>
    <p>Test</p>
</body>
</html>
```

주의 사항

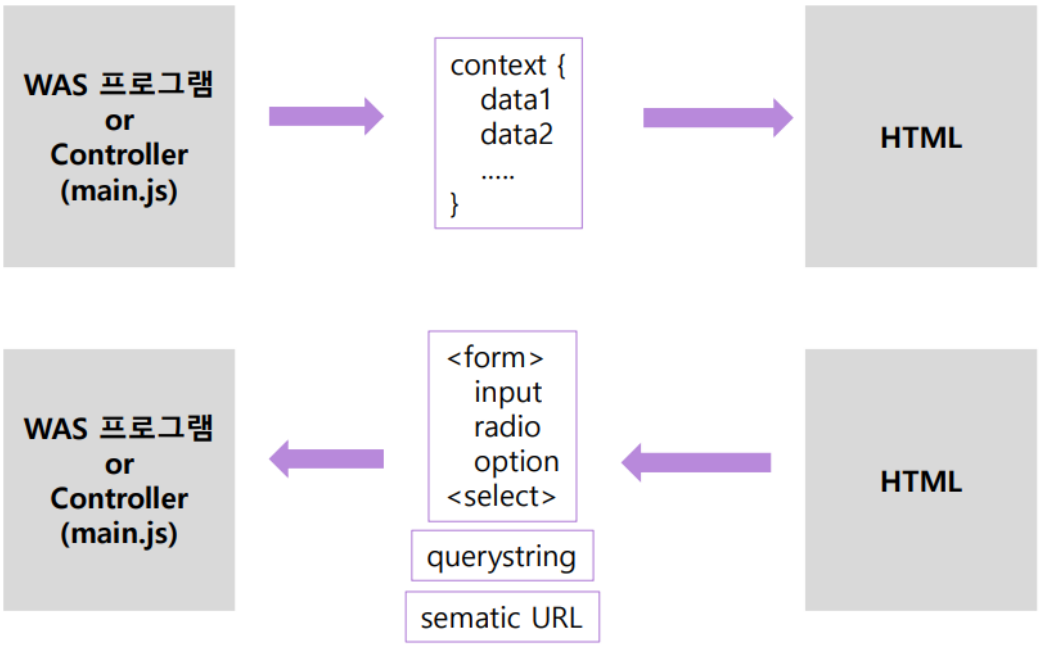
ejs에서 필요로 하는 모든 정보를 다 보내줘야한다.

app.get('/',~)과 app.get('/:id',~)가 같은 home.ejs를 호출한다고 하자.

home.ejs는 <%=title%>, <%=name%>이라는 두 변수를 필요로한다.

이때 app.get('/:id',~)에만 title과 name을 넘겨주고 app.get('/',~)에서 title만 넘겨주면 에러가 뜬다. (흰화면 - 화면을 꾸미는데 필요한 data부족)

즉, ejs에 data를 부족하게 전달하면 안된다.



HTML에서 WAS로 전달하는 3가지 방법 암기

1. form 태그 활용
2. 쿼리스트링 활용
3. 시맨틱 URL 활용