

생활코딩

Node.js 노드제이에스 프로그래밍

2024.10.
7주 수업



7주차 수업의 범위

* 세션

* **body-parser**

1. 쿠키는 위험 ➔ 검사 메뉴에서 모두 볼 수 있음.

현대의 앱들은 인증을 쿠키로 구현하지 않음.

2. 세션

세션 아이디와 값을 분리하는 기법

사용자의 민감한 정보(값)는 서버에 저장

세션 아이디 - 사용자를 식별하는 값으로서 기능

사용자의 실제 정보 - 사용자의 로그인 여부 닉네임, 마지막 접속 시간 등이 **sessions** 디렉토리에 파일로 저장

1. express-session 미들웨어 설치

```
dejs\prj> npm install -s express-session
```

2. session 사용

```
var express = require('express') ;
var parseurl = require('parseurl');
var session = require('express-session');
var app = express() ;

app.use(session({
  secret : 'keyboard cat',
  resave : false,
  saveUninitialized : true
}));

app.use(function(req,res,next){
  if(!req.session.views){
    req.session.views={};
  }

  //get the url pathname
  var pathname = parseurl(req).pathname;

  //count the views
  req.session.views[pathname] = (req.session.views[pathname] || 0) + 1;

  next();
});
```

2. session 사용

```
app.get('/foo', function(req,res,next){
  res.send('you viewed this page' + req.session.views['/foo'] + ' times');
});

app.get('/bar', function(req,res,next){
  res.send('you viewed this page' + req.session.views['/bar'] + ' times');
});

app.listen(3000, function(){
  console.log('3000!');
});
```

2. session 사용 - 실행 결과

DevTools is now available in Korean! [Always match Chrome's language](#) [Switch DevTools to Korean](#) [Don't show again](#)

Elements Console Sources Network Performance Memory Application Security Lighthouse >> 1

Application

- Manifest
- Service Workers
- Storage

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies

Filter

Only show cookies with an issue

Name	Value	Dom...	Path	E..	S.	Ht...	Sec...	Sa...	Sa...	Part..	Pr...
connect.sid	s%3AY0LBxYhYcy...	local...	/	S...	91	✓					Me...
Permanent	cookies	local...	/	2...	16						Me...

**connect.sid 쿠키 : 서버에 접속할 때마다 웹브라우저가 서버 쪽에 전송
이 쿠키를 전송받은 서버는 쿠키에 담긴 세션 아이디에 해당하는 사용자를
식별해서 그에 맞는 데이터를 처리**

localhost:3000/bar

localhost:3000/bar

YouTube 지도

you viewed this page2 times

1. 코드 설명

① `var session = require('express-session');`

모듈을 불러와서 session 이라는 객체에 저장.

② `app.use(session({
 secret : 'keyboard cat',
 resave : false,
 saveUninitialized : true
}));`

모든 경로의 클라이언트 요청에 대해
session 함수를 실행하라

사용자의 요청이 있을 때마다 app.use 안의 session 함수가 호출되어 세션이 시작.

그러면 내부적으로 express-session 미들웨어가 개입해서 애플리케이션이 세션을 사용할 수 있게 처리

use 메소드 : 클라이언트로부터 오는 모든 메소드에 대해 실행

use 메소드의 형식

use(경로, 미들웨어함수)

경로로 들어오는 요청이 get 요청이든 post 요청이든 상관없이 미들웨어 함수 실행. 만약, 경로가 없다면

모든 경로의 클라이언트 요청에 대해 미들웨어 함수 실행



Express 라우트 메소드

* get Method

- GET 요청이면서 요청 URL이 첫번째 인자에 해당할 경우 두 번째 인자의 함수를 실행하고 res.send와 같은 응답 메소드로 사이클을 마친다.

또는 응답으로 마치지 않고 next('route')를 써서 다음 get 메소드로 진행할 수 있다.

* post Method

- POST 요청이라는 것만 다르고 get method와 같다.

* use Method

- 첫번째 인자인 URL로 요청이 들어오면 두번째 인자인 미들웨어 함수를 실행한다.
- 첫번째 인자가 생략되면 모든 URL 요청에 대해 미들웨어 함수를 실행한다.
- next() 메소드를 사용해야 다음으로 진행할 수 있다.



미들웨어의 실행 순서

1. 애플리케이션 레벨 미들웨어

① 형식

```
var app = express()

app.use(function (req, res, next) {
  console.log('Request Type: ', req.method) ;
  next() ;
})
```

use안의 함수가 미들웨어

요청(req), 응답(res) 객체를 받아서

- ② 특징 : 요청(req), 응답(res) 객체를 받아서 활용할 수 있다. 그리고 next 함수를 이용해서 그 다음 미들웨어 실행 여부를 이전 미들웨어에서 호출(next()) 여부로 결정할 수 있다.
use 메소드의 첫 번째 인수로 경로를 넘겨줌으로써 해당 미들웨어가 특정 경로에서만 동작하는 것이 가능.



미들웨어의 실행 순서

2. 애플리케이션 레벨 미들웨어 실행 순서 - 1

```
app.use('/user/:id', function(req,res,next){ ①  
  console.log('Request URL:', req.originalUrl);  
  next();  
}, function(req,res,next){ ②  
  console.log('Request Type:', req.method);  
  next();  
})
```

①번, ②번 함수를 순서대로 실행



미들웨어의 실행 순서

2. 애플리케이션 레벨 미들웨어 실행 순서 - 2

```
app.get('/user/:id', function(req,res,next){ ①  
    console.log('ID:', req.params.id);  
    next();  
}, function(req,res,next){ ②  
    res.send('User Info');  
})  
  
app.get('/user/:id', function(req,res,next){ ③  
    res.end(req.params.id);  
})
```

첫번째 라우트에 있는 첫 번째 미들웨어 ①이 실행

next 함수가 호출되면 다음에 있는 미들웨어 ②가 실행

②번 미들웨어에는 next 함수 호출이 없으므로 해당 경로에 대한 라우팅을 종료

두번째 라우트에 있는 미들웨어 ③은 실행되지 않음



미들웨어의 실행 순서

2. 애플리케이션 레벨 미들웨어 실행 순서 - 3

```
app.get('/user/:id', function(req,res,next){ ①  
  if (req.params.id === 0) next('route')  
  else next();  
}, function(req,res,next){ ②  
  res.send('regular');  
})  
  
app.get('/user/:id', function(req,res,next){ ③  
  res.send('special');  
})
```

첫번째 라우트에 있는 첫 번째 미들웨어 ①이 실행.

if 문에서 요청 객체 req에 있는 params.id값이 0이면 next('route')는 다음 라우트의 미들웨어를 실행하라는 뜻임.

그래서 /user/:id에 대한 두 번째 라우트에 있는 미들웨어 ③이 실행.

params.id 값이 0이 아니면 인자가 없는 next()를 호출해 다음 미들웨어 ②가 실행

1. 코드 설명

```
② app.use(session({  
  secret : 'keyboard cat',  
  resave : false,  
  saveUninitialized : true  
}));
```

secret 옵션 – 필수 옵션. 세션 ID 쿠키를 서명하는 데 사용할 문자열

다른 사람이 보서는 안 되는 내용이라서 노출하면 안 되고 자신만 아는 내용으로 입력

실제 서버에 올릴 때는 이 코드를 변수 등으로 처리해야 함. 보안사항이기 때문

resave 옵션 – 데이터를 세션 저장소에 저장할지를 설정. false로 지정하면 세션 데이터가 바뀌지 않는 한 세션 저장

소에 저장하지 않고, true이면 세션 데이터의 변경 여부와 상관없이 무조건 세션 저장소에 저장

saveUninitialized – 세션의 구동 여부를 설정. true로 지정하면 세션이 필요하기 전까지는 세션을 구동하지 않고,

false이면 세션의 필요 여부와 상관없이 무조건 세션을 구동

session 함수에는 내부적으로 next()가 들어 있어서 다음 use로 넘어간다.

1. 코드 설명

```
③ app.use(function(req,res,next){  
    if(!req.session.views){  
        req.session.views={};  
    }  
  
    //get the url pathname  
    var pathname = parseurl(req).pathname;  
  
    //count the views  
    req.session.views[pathname] = (req.session.views[pathname] || 0) + 1;  
  
    next();  
});
```

req.session.views : session 객체의 멤버변수로 views를 생성 빈 객체 {} 으로 초기화.

views[pathname] = : views의 멤버변수 pathname에 값 저장

※ java script의 객체 정의 객체 이름 = { 멤버변수 : 값,

 멤버함수 : fuctions (인수) { }

}

1. session 객체 생성

```
var express = require('express') ;
var parseurl = require('parseurl');
var session = require('express-session');
var app = express() ;

app.use(session({
  secret : 'keyboard cat',
  resave : false,
  saveUninitialized : true
}));

app.get('/', function(req,res,next){
  console.log(req.session); session 미들웨어는 request 객체의 속성으로
  res.send('Hello session'); session 객체를 추가함.
});

app.listen(3000, function(){
  console.log('3000!');
});
```

```
Session {
  cookie: { path: '/', _expires: null, originalMaxAge: null, httpOnly: true }
}
```

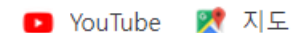

2. session 객체 num 속성 사용

```
var express = require('express') ;
var parseurl = require('parseurl');
var session = require('express-session');
var app = express() ;

app.use(session({
  secret : 'keyboard cat',
  resave : false,
  saveUninitialized : true
}));

app.get('/', function(req,res,next){
  console.log(req.session);
  if(req.session.num === undefined){
    req.session.num = 1;
  }
  else{
    req.session.num += 1;
  }
  res.send( `Hello session : ${req.session.num}` );
});

app.listen(3000, function(){
  console.log('3000!');
});
```



Hello session : 2

session 데이터는 서버에 저장됨으로 웹 서버를 종료하면
세션이 지워짐

1. **session** 저장소는 데이터베이스나 파일 저장 가능
2. **session** 저장 방법에 따라 모듈이 나누어져 있음.
3. 파일로 세션 저장 - 다음 모듈 설치

```
ejs\prj> npm install session-file-store
```

4. 코드 추가

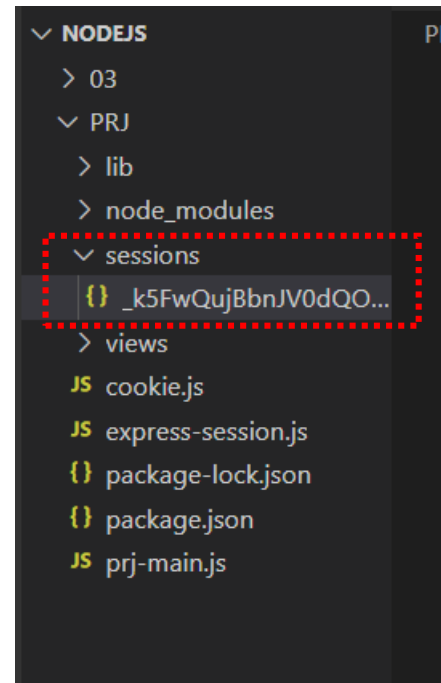
```
var express = require('express') ;
var parseurl = require('parseurl');
var session = require('express-session');
var FileStore = require('session-file-store')(session);
var app = express() ;

app.use(session({
  secret : 'keyboard cat',
  resave : false,
  saveUninitialized : true,
  store : new FileStore()
}));

app.get('/', function(req,res,next){
  console.log(req.session);
  if(req.session.num === undefined){
    req.session.num = 1;
  }
  else{
    req.session.num += 1;
  }

  res.send(`Hello session : ${req.session.num}`);
});

app.listen(3000, function(){
  console.log('3000!');
});
```

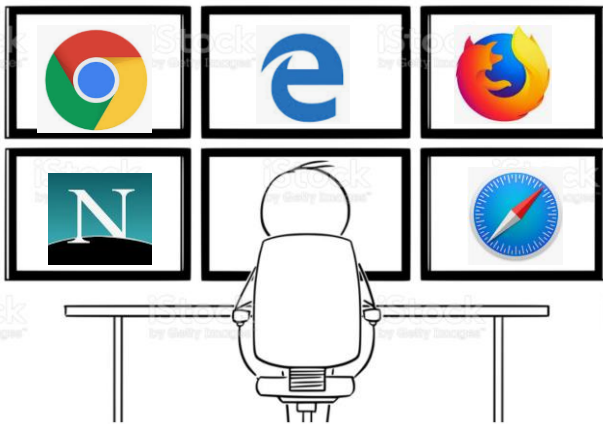


웹페이지 요청하면 sessions 폴더 생성되고 json 파일 생성

익스프레스 세션 미들웨어는 사용자가 세션 아이디를 가지고 있는 상태에서 서버로 접속하면 요청 헤더에 쿠키 값으로 세션 아이디를 서버에 전달.

5. session 메커니즘

클라이언트



'/' 요청

sessions 폴더
에 json 생성

session id 전송

쿠키에 id 저장

'/' 요청 session id 전송

웹서버

```
...  
var session = require('express-session');  
var FileStore = require('session-file-store')(session);  
...  
app.use(session({  
  secret : 'keyboard cat',  
  resave : false,  
  saveUninitialized : true,  
  store : new FileStore()  
}));  
  
app.get('/', function(req,res,next){  
  ...  
});  
  
app.listen(3000, function(){  
  console.log('3000!');  
});
```

필요하다면 json에
정보갱신

1. **session** 저장소는 데이터베이스나 파일 저장 가능
2. **session** 저장 방법에 따라 모듈이 나누어져 있음.
3. **mysql**로 세션 저장 - 다음 모듈 설치

```
prj> npm install -s express-mysql-session
```

4. 코드 추가

```

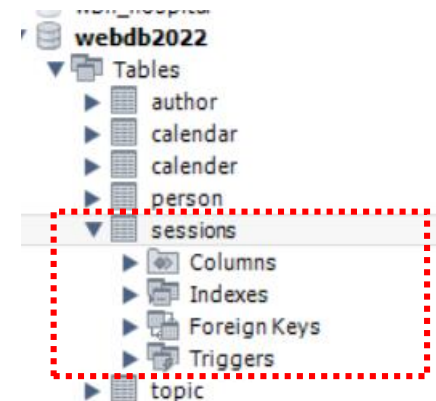
var express = require('express') ;
var parseurl = require('parseurl');
var session = require('express-session');
var MySQLStore = require('express-mysql-session')(session);
var options = {
  host      : 'localhost',
  user      : 'nodejs',
  password  : 'nodejs',
  database  : 'webdb2024'
};
var sessionStore = new MySQLStore(options);
var app = express() ;

app.use(session({
  secret : 'keyboard cat',
  resave : false,
  saveUninitialized : true,
  store : sessionStore
}));

app.get('/', function(req,res,next){
  ...
});

app.listen(3000, function(){
  console.log('3000!');
});

```



session_id	expires	data
zSbpJxPfaFQqBBBqn4wtu0EHC0Mw_m	1666079185	{ "cookie": { "originalMaxAge": null, "expires": null, ...

session_id	expires	data
RTYnFtuGzjW_KYRu22qqr8YU1cPNRWAg	1666079448	{ "cookie": { "originalMaxAge": null, "expires": null, "httpOnly": true, "path": "/", "num": 1 }
zSbpJxPfaFQqBBBqn4wtu0EHC0Mw_m	1666079416	{ "cookie": { "originalMaxAge": null, "expires": null, "httpOnly": true, "path": "/", "num": 3 }

웹페이지 요청하면 sessions 테이블 생성 만약 테이블이 있다면 레코드 추가

1. 로그인 링크 만들기

main.js에 session 관련 모듈 require하는 코드 추가

```
var session = require('express-session');
var MySQLStore = require('express-mysql-session')(session);
var options = {
  host : 'localhost',
  user : 'nodejs',
  password : 'nodejs',
  database : 'webdb2024'
};
var sessionStore = new MySQLStore(options);

app.use(session({
  secret : 'keyboard cat',
  resave : false,
  saveUninitialized : true,
  store : sessionStore
}));
```

1. 로그인 링크 만들기

home.ejs

```
<body>
  <a href="/login">login</a>
  <h1><a href="/"><%=title%></a></h1>
  <a href="/author">저자관리</a>
  <ol type="1">
```

로그아웃하고 로그인할 때마다 바뀌어야 하니까 링크 전체를 템플릿 변수로 한다

```
<body>
  <%=login%>
  <h1><a href="/"><%=title%></a></h1>
```



1. 로그인 링크 만들기

main.js에 /login URL 분류기 추가

```
app.get('/login',(req,res)=>{
  topic.login(req,res);
})
```

topic.js 코드 수정

```
home : (req,res) => {
  db.query('SELECT * FROM topic', (error,topics)=>{
    var login = ''
    login = `
```

2. login 을 클릭하면 다음과 같은 화면이 나오도록 **topic.js**를 수정하기

login

Topic List

[저자관리](#)

1. [MySQL](#)
2. [NodeJs](#)
3. [HTML](#)
4. [CSS](#)
5. [express](#)
6. [날씨](#)
7. [XSS](#)

[create](#)

email

password

```
login : (req,res) => {
  db.query('SELECT * FROM topic', (error,topics)=>{
    var m = '<a href="/create">create</a>'
    var b = `<form action="/login_process" method="post">
      <p><input type = "text" name="email" placeholder="email"</p>
      <p><input type = "text" name="password" placeholder="password"</p>
      <p><input type="submit"></p>
    </form>`

    var context = { login : `<a href="/login">login</a>`,
      title : 'Login ID/PW 생성',
      list : topics,
      menu : m,
      body : b};

    req.app.render('home', context, (err,html)=>{
      res.end(html) })
  });
},
```

3. 제출 버튼을 클릭하면 로그인 표시가 로그아웃 표시로 바뀌면서 세션을 생성하는 프로그램 작성하기



main.js에 다음 코드 추가하기

```
app.post('/login_process', (req, res) => {
  topic.login_process(req, res);
})
```

3. 제출 버튼을 클릭하면 로그인이 로그아웃 되면서 쿠키를 생성하는 프로그램 작성하기

topic.js에 login_process 메소드 추가하기

```
login_process : (req, res) => {  
  var body = '';  
  req.on('data', (data)=> {  
    body = body + data;  
  });  
  req.on('end', () => {  
    var post = qs.parse(body);  
    if(post.email==='bhwang99@gachon.ac.kr' && post.password==='123456'){  
      req.session.is_logged_in = true;  
      res.redirect('/');  
    }  
    else {  
      res.end('who?');  
    }  
  });  
},
```

세션 생성코드

3. 제출 버튼을 클릭하면 로그인이 로그아웃 되면서 쿠키를 생성하는 프로그램 작성하기

topic.js에 authIsOwner 함수 수정하기

```
function authIsOwner(req,res) {  
    if(req.session.is_logged)  
        { return true; }  
    else {  
        return false }  
}
```

3. 제출 버튼을 클릭하면 로그인이 로그아웃으로 되면서 세션을 생성하는 프로그램 작성하기

topic.js에 home 메소드 그대로 사용

```
home : (req,res)=>{
  db.query('select * from topic', (error, topics)=>{
    var login = authStatusUI(req,res)

    var m = '<a href="/create">create</a>'
    var b = '<h2> Welcome </h2><p>Node.js Start Page </p>'

    if (topics.length == 0){
      b = '<h2> Welcome </h2><p>자료가 없으니 create 링크를 이용하여 자료를 입력하세요 </p>'
    }

    var context = {title : "WEB Topic 테이블",
      login: login,
      list : topics,
      menu : m,
      body : b };
    res.render('home',context,(err,html)=>{
      res.end(html)
    });
  });
},
```

4. 로그아웃 처리

main.js 에
logout_process추가

```
app.get('/logout_process', (req, res) => {  
  topic.logout_process(req, res);  
})
```

topic.js 에
logout_process추가

```
logout_process : (req, res) => {  
  req.session.destroy((err) => {  
    res.redirect('/');  
  })  
},
```

5. 접근 제어

로그인한 사람만 생성, 갱신, 삭제 가능하도록

1. 미들웨어

익스프레스가 기본으로 제공하는 기능이 아닌 다른 사람이 만든 소프트웨어

2. body-parser 미들웨어

■ **post** 방식으로 전달되는 데이터의 크기가 크기 때문에 **data**라는 이벤트가 발생할 때마다 **body** 변수에 데이터를 추가하다가 **end** 이벤트가 발생하면 **body** 변수에 담긴 데이터를 처리

```
create_process : (req, res) => {  
  var body = '';  
  req.on('data', (data) => {  
    body = body + data;  
  });  
  req.on('end', () => {  
    var post = qs.parse(body);  
  });  
}
```

■ **body-parser** 미들웨어를 사용하면 좀 더 간결한 코드 작성 가능

- **body**란 웹 브라우저에서 요청한 정보의 본문을 의미
- 요청 정보의 본문을 해석해서 우리에게 필요한 형태로 가공해주는 프로그램

2. body-parser 미들웨어

설치

```
> npm install body-parser
```

사용법

main.js에 다음 코드 추가

```
var bodyParser = require('body-parser');  
app.use(bodyParser.urlencoded({extended: false}));
```

topic.js의 create_process를 다음과 같이 수정

```
create_process : (req,res) => {  
  // var body = '';  
  // req.on('data', (data)=> {  
  //   body = body + data;  
  // });  
  // req.on('end', () => {  
  //   var post = qs.parse(body);  
  var post = req.body;
```

※ body-parser는 사용자가 POST 방식으로 전송한 데이터를 내부적으로 분석하여 create_process의 이전 코드와 같은 동작을 한 다음 그 결과를 전달해 준다.