강좌5 - NodeJS - 6년 전 등록 / 4년 전 수정

익스프레스 미들웨어, 라우팅

에러 처리

Nodejs교과서 책이 출간되었습니다. 이 포스팅보다는 책이나 동영상 강좌를 보시는 것을 추천합니다.

안녕하세요 이번 시간에는 익스프레스 프레임워크의 **미들웨어**와 **라우팅**에 대해 알아보겠습니다!

미들웨어(Middleware)

익스프레스 프레임워크의 장점 중 하나가 미들웨어를 사용한다는 겁니다. Middleware가 뭘까요? 이름처럼 요청에 대한 응답 과정 **중간**에 껴서 어떠한 동작을 해주는 프로그램입니다. 익스프레스는 요청이 들어올 때 그에 따른 응답을 보내주는데요. 응답을 보내기 전에 미들웨어가 지정한 동작을 수행합니다. 전 시간에 보았던 express.static이 미들웨어의 한 종류로, 정적 파일의 기본 경로를 정해주는 역할을 합니다.

미들웨어에는 수 많은 것들이 있는데 대표적으로 Morgan, Compression, Session, Body-parser, Cookie-parser, Method-override, Cors, Multer 등이 있습니다. 모두 npm에서 다운받을 수 있습니다.

간단히 소개하자면, **Morgan**은 익스프레스 프레임워크가 동작하면서 나오는 메시지들을 콘솔에 표시해줍니다. **Compression**은 이름처럼 페이지를 압축해서 전송해주고요. **Session**은 세션을, **Body-parser**은 폼에서 전송되는 POST 값을, **Cookie-parser**는 쿠키를 사용할 수 있게 해줍니다. **Method-override**는 REST API에서 PUT과 DELETE 메

목록

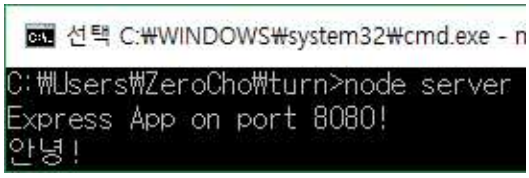
전 시간에 봤던 express.static도 미들웨어의 한 종류입니다. 예전에는 상술했던 대부분의 미들웨어가 express에 포함되어 있었지만 이제는 express.static을 제외하고 다 **분리**되어 따로 **npm install**로 설치해야합니다.

```
const compression = require('compression');
const cors = require('cors');
const express = require('express');
const app = express();
app.use(compression());
app.use(cors());
app.get(...); // 이전과 동일
app.listen(8080, ...); // 이전과 동일
```

express에서는 `app.use(미들웨어)`로 사용할 미들웨어를 응답을 보내기 전에 넣어주면 됩니다. 만약 필요한 게 있다면 npm 검색을 통해서 찾아보세요! 미들웨어는 직접 만들 수도 있습니다. 라우팅 부분(`app.get`, `app.post`,...) 위에만 넣으면 됩니다. 간단히 안녕!이라고 말하는 허접한 미들웨어를 만들어보겠습니다.

```
app.use((req, res, next) => {
  console.log('안녕!');
  next();
});
```

네... 끝입니다. 이제부터 request가 올 때마다 콘솔에 안녕하고 외치게 됩니다.



일단 `const app = express();` 한 후에 `app.use`로 순차적으로 미들웨어가 시작되는데요. 제가 만든 것에서는 안 쓰였지만, 미들웨어는 request와 response를 매개변수로 받아 조작할 수 있습니다. 위에서 언급한 모든 미들웨어가 이런 식으로 request와 response를 조작한 겁니다. 그리고 마지막으로 `next()`를 하면 다음 미들웨어로 넘어갑니다. `next()`를 하지 않으면 더 이상 진행이 되지 않기 때문에 꼭 넣어주셔야합니다.

참고로 **에러**가 발생했을 때는 `next(에러)`처럼 next의 인자로 error 정보를 넣어 **라우팅** 부분으로 넘겨줍니다. 안에 에러 객체를 넣어주었을 때는 상황이 조금 다릅니다. 그 에러는 라우팅에서 처리합니다. 이 글 제일 아래 나와있습니다.

라우팅

익스프레스의 또다른 장점은 **라우팅**이 편리합니다. 라우팅이라 함은 클라이언트에서 보내는 주소에 따라 다른 처리를 하는 것을 의미하는데요. 익스프레스는 **REST API**에 따라 처리하는 데 그 방법이 아주 간단합니다.

app 객체에 `app[REST메소드]('주소', 콜백함수)` 이렇게 연결하는데요. 앞에서 `app.get('/', 콜백)` 하는 것을 보셨죠? 바로 / 주소로 GET 요청이 올 때 콜백하라고 등록한 겁니다. `app.get` 외에, `app.post`, `app.put`, `app.delete` 메소드를 사

와일드카드를 사용할 때는 순서가 중요합니다.

```
app.get('/post/:id', () => {});
app.get('/post/a', () => {});
```

이렇게 되어있다면 /post/a에 요청이 왔을 때 한 눈에 보기엔 사람들은 두 번째 라우터에 걸릴거라고 생각하지만 /post/:id에 먼저 걸립니다. /post/a의 콜백 함수는 실행되지 않습니다. 따라서 와일드카드 라우터는 항상 다른 라우터들보다 뒤에 적어주는 것이 좋습니다.

이렇게 app에 계속 이벤트 리스너처럼 연결하면 되는데, 라우팅은 반복되는 부분이 많기 때문에 주로 모듈로 분리해서 사용합니다. *server.js*에서 라우팅 부분을 지우고, *route.js* 파일을 만들어봅시다.

route.js

```
const express = require('express');
const path = require('path');
const router = express.Router(); // 라우터 분리
router.get('/', (req, res) => { // app 대신 router에 연결
  res.sendFile(path.join(__dirname, 'html', 'main.html'));
});
router.get('/about', (req, res) => {
  res.sendFile(path.join(__dirname, 'html', 'about.html'));
});
module.exports = router; // 모듈로 만드는 부분
```

express에서는 `express.Router()`을 사용해 라우터를 분리할 수 있습니다. `module.exports`가 바로 **모듈**을 만드는 코드입니다. 이 부분이 있어야 다른 파일에서 여기서 **export**한 것을 require할 수 있습니다. **모듈 시스템에 대한 자세한 강좌**는 여기를 참고하세요.

이렇게 만든 *route.js*파일을 *server.js*에서 불러옵니다. 이전 시간과 변화는 없지만 코드를 분리했다는 것에 의미를 둡시다. 이렇게 코드를 잘게 분리해야 나중에 유지보수가 쉽습니다. 수백 수천 줄이 있는 하나의 긴 파일에서 코드를 찾는 것보다는 기능별로 분리된 파일에서 찾는 게 더 쉽겠죠?

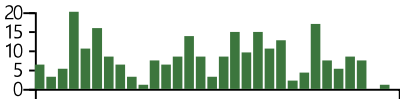
```
const route = require('./route.js');
...
app.use('/', route);
app.use((req, res, next) => { // 404 처리 부분
  res.status(404).send('일치하는 주소가 없습니다!');
});
app.use((err, req, res, next) => { // 에러 처리 부분
  console.error(err.stack); // 에러 메시지 표시
  res.status(500).send('서버 에러!'); // 500 상태 표시 후 에러 메시지 전송
});
app.listen(8080, ...);
```

제일 위에 `const route = require('./routes.js');`를 했는데 이 부분이 route 변수에 아까 `module.exports`로 export했던 router을 연결하겠다는 뜻입니다. `app.use`로 라우팅을 하는 것의 장점은, **그룹화**가 쉽다는 겁니다. `app.use('/category', route1);` 이런 코드가 있으면 route1에 있는 라우터들은 모두 category 주소 아래에 그룹화됩니

그 아래는 서버 에러를 처리하는 부분입니다. 일반 `app.use`에 비해 err 매개변수 하나가 더 있습니다. 바로 `next(err)`로 넘겨줬던 에러가 최종적으로 도착하는 부분입니다. `next(err)`가 호출되는 순간 다른 `app.use`는 모두 건너뛰고 바로 err 매개변수가 있는 `app.use`로 넘어옵니다. 이 부분이 없으면 next로 에러를 넘겨주었을 시 처리할 부분이 없어 서버가 죽어버립니다. 에러를 기록하고 브라우저에 서버에서 에러가 발생했다고 알려줍니다.

다음 시간에는 익스프레스 **템플릿**에 대해 알아보겠습니다!

이전글: Express 프레임워크
다음글: 익스프레스 템플릿(Jade, Pug), Express template

조회수: 

투표로 게시글에 관해 피드백을 해주시면 게시글 수정 시 반영됩니다. 오류가 있다면 어떤 부분에 오류가 있는지도 알려주세요! 잘못된 정보가 퍼져나가지 않도록 도와주세요.

만족해요	**20(66.67%)**
설명이 부족해요	**4(13.33%)**
너무 어려워요	**6(20.00%)**
오류가 있는 거 같아요	**0(0.00%)**
총 투표 수	**30**

이메일 | 비밀번호(수정 및 삭제 시 사용)

댓글을 입력하세요. 근거없는 비난 대신 날카로운 피드백을 원합니다. 답글이 달리면 삭제할 수 없습니다.

등록

10개의 댓글이 있습니다.

익명 안녕하세요 책 구매해서 공부하는 중인데 도저히 해결이 안 되는 오류가 있어서 문의 드립니다. ㅠㅠ...

일 년 전

```
app.get('/', (req, res, next) => {
console.log('GET / 요청에서만 실행됩니다.');
next();
}, (req, res) => {
throw new Error('에러는 에러 처리 미들웨어로 갑니다.')
});
app.use((err, req, res, next) => {
console.error(err);
res.status(500).send(err.message);
});
```

이 부분에서 에러 처리 미들웨어가 4개의 매개변수를 인식하지 못합니다

목록

글을 보는 화면에 수정버튼이 원글의 작성자에게만 보이게 하려고 하는데 이 화면(ejs)을 렌더링하는 파일이랑 로그인