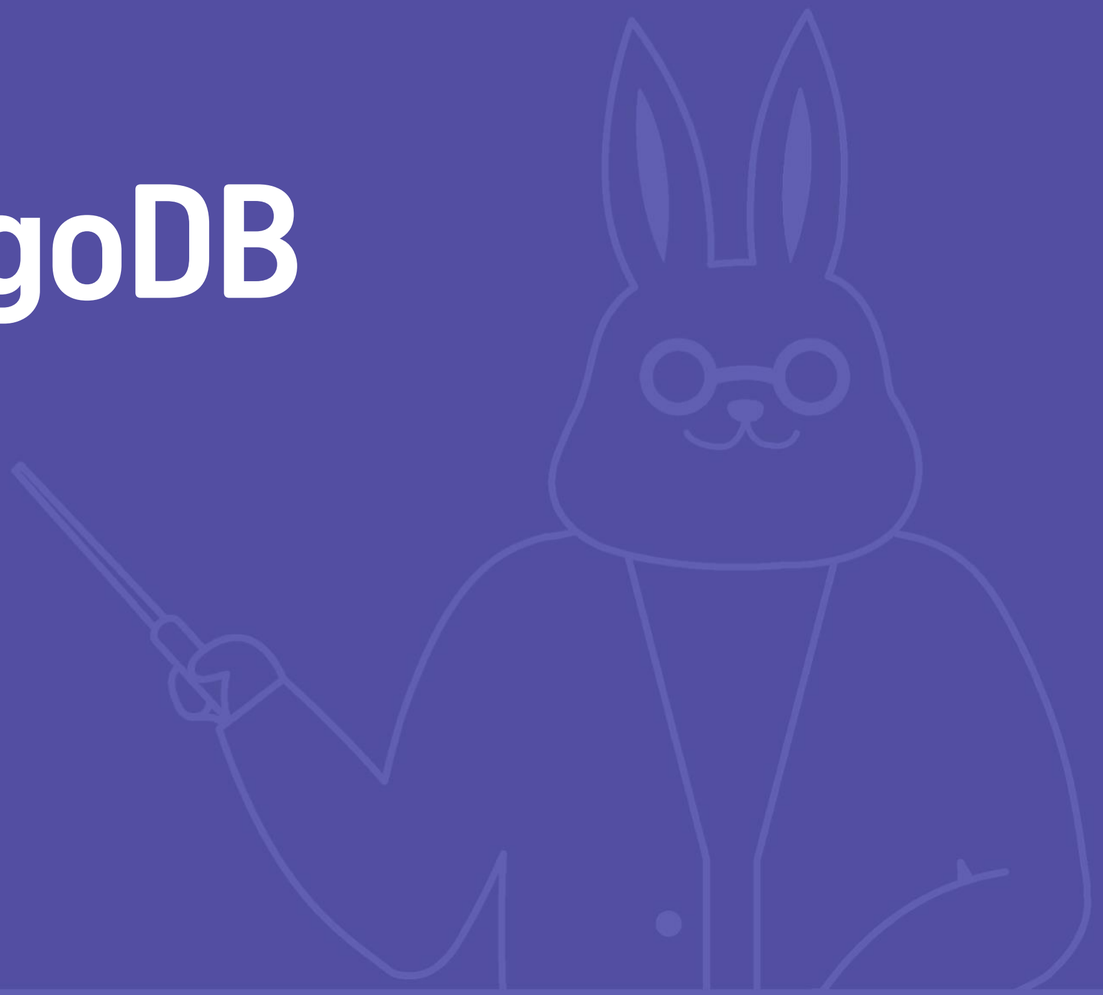
Express.js와 MongoDB

01 Express.js의 미들웨어



/* elice */

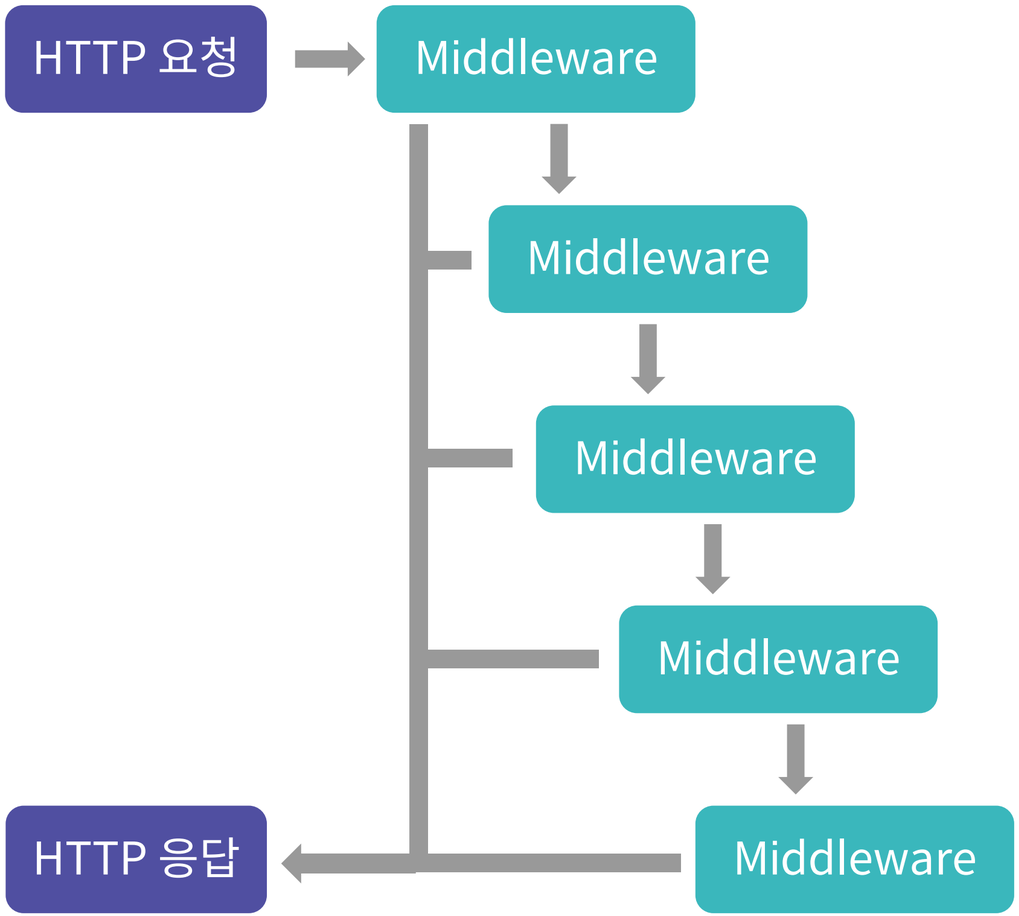
01

Express.js의 Middleware

Middleware란?

미들웨어는 Express.js **동작의 핵심**
HTTP 요청과 응답 사이에서 **단계별 동작을 수행해주는 함수**

Middleware 동작 원리



Express.js의 미들웨어는 HTTP 요청이 들어온 순간부터 시작이 됨

미들웨어는 **HTTP 요청과 응답 객체를 처리**하거나, **다음 미들웨어를 실행**할 수 있음

HTTP 응답이 마무리될 때까지 미들웨어 동작 사이클이 실행됨

02

Middleware의 작성과 사용

middleware 작성법

req, res, next를 가진 함수를 작성하면 해당 함수는 미들웨어로 동작할 수 있음

- ✓ req는 HTTP 요청을 처리하는 객체
- ✓ res는 HTTP 응답을 처리하는 객체
- ✓ next는 다음 미들웨어를 실행하는 함수

Route Handler와 middleware

Route Handler도 **미들웨어의 한 종류**
Route Handler는 **라우팅 함수(get, post, put, delete 등)에 적용된** 미들웨어
일반적인 미들웨어와는 다르게 **path parameter를 사용**할 수 있음

middleware 작성법

middleware-examples

```
const logger = (req, res, next) => {
  console.log(`Request ${req.path}`);
  next();
}

const auth = (req, res, next) => {
  if (!isAdmin(req)) {
    next(new Error('Not Authorized'));
    return;
  }
  next();
}
```

req, res, next를 인자로 갖는 함수를 작성하면 미들웨어가 됨

req, res 객체를 통해 HTTP 요청과 응답을 처리하거나
next 함수를 통해 다음 미들웨어를 호출해야 함

next() 함수가 호출되지 않으면
미들웨어 사이클이 멈추기 때문에 주의

middleware 사용법

middleware 는 적용되는 위치에 따라서
어플리케이션 미들웨어, 라우터 미들웨어, 오류처리 미들웨어로 분류 가능
필요한 동작 방식에 따라 미들웨어를 **적용할 위치**를 결정

middleware 사용법 - 어플리케이션 미들웨어

application middleware

```
app.use((req, res, next) => {
  console.log(`Request ${req.path}`);
  next(); 1
});

app.use(auth); 2

app.get('/', (req, res, next) => {
   res.send('Hello Express'); 3
});
```

use 나 **http method** 함수를 사용하여
미들웨어를 연결할 수 있음

미들웨어를 모든 요청에 공통적으로 적용하기
위한 방법

HTTP 요청이 들어온 순간부터 적용된
순서대로 동작 함

middleware 사용법 - 라우터 미들웨어

router middleware

```
router.use(auth); 3

router.get('/', (req, res, next) => {
  res.send('Hello Router');
}); 4

app.use((req, res, next) => {
  console.log(`Request ${req.path}`);
  next(); 1
});

app.use('/admin', router); 2
```

router 객체에 미들웨어가 적용되는 것 외에는 어플리케이션 미들웨어와 사용 방법은 동일

특정 경로의 라우팅에만 미들웨어를 적용하기 위한 방법

app 객체에 라우터가 적용된 이후로 순서대로 동작함

middleware 사용법 - 미들웨어 서브 스택

middleware sub-stack

```
app.use(middleware1, middlware2, ...);

app.use('/admin', auth, adminRouter);

app.get('/', logger, (req, res, next) => {
  res.send('Hello Express');
});
```

use 나 http method 함수에 **여러 개의 미들웨어를 동시에 적용**할 수 있음

주로 한 개의 경로에 특정해서 미들웨어를 적용하기 위해 사용

전달된 인자의 순서 순으로 동작

오류처리 미들웨어

오류처리 미들웨어는 **일반적으로 가장 마지막에 위치**하는 미들웨어
다른 미들웨어들과는 달리 **err, req, res, next 네 가지 인자**를 가지며,
앞선 미들웨어에서 **next 함수에 인자가 전달되면 실행**됨

오류처리 미들웨어

error handling middleware

```
app.use((req, res, next) => {
  if (!isAdmin(req)) {
    next(new Error('Not Authorized')); 1
    return;
  }
  next();
});

app.get('/', (req, res, next) => {
    res.send('Hello Express');
});

app.use((err, req, res, next) => { 2
  res.send('Error Occurred');
});
```

가장 아래 적용된 err, req, res, next를 인자로 갖는 함수가 오류처리 미들웨어

이전에 적용된 미들웨어 중 next에 인자를 넘기는 경우 중간 미들웨어들은 뛰어넘고

오류처리 미들웨어가 바로 실행됨

함수형 middleware

하나의 미들웨어를 작성하고, **작동 모드를 선택하면서 사용**하고 싶을 경우
미들웨어를 함수형으로 작성하여 사용할 수 있음

Ex) API별로 사용자의 권한을 다르게 제한하고 싶은 경우

함수형 middleware

함수형 미들웨어

```
const auth = (memberType) => {
  return (req, res, next) => {
    if (!checkMember(req, memberType)) {
      next(new Error(`member not ${memberType}`));
      return;
    }
    next();
  }
}

app.use('/admin', auth('admin'), adminRouter);

app.use('/users', auth('member'), userRouter);
```

auth 함수는 **미들웨어 함수를 반환하는 함수**

auth **함수 실행 시 미들웨어의 동작이 결정**되는 방식으로 작성됨

일반적으로 **동일한 로직에 설정값만 다르게** 미들웨어를 사용하고 싶을 경우에 활용됨

Middleware Libraries

Express.js는 다양한 미들웨어들이 **이미 만들어져 라이브러리로 제공**됨
유용한 미들웨어를 npm 을 통해 추가하여 사용할 수 있음
Express.js 홈페이지나 **npm 온라인 저장소**에서 찾아볼 수 있음

Ex) cors, multer, passport 등

Middleware 요약

미들웨어는 **HTTP 요청과 응답 사이**에서 동작하는 **함수**
req, res, next를 인자로 갖는 함수는 미들웨어로 동작할 수 있음
app 혹은 router 객체에 연결해서 사용 가능
next에 인자를 넘기는 경우 **오류처리 미들웨어**가 실행됨
미들웨어에 값을 설정하고 싶은 경우는 **함수형 미들웨어**로 작성 가능

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io