



7. Express.js

2023학년 2학기 웹응용프로그래밍

권 동 현

Contents

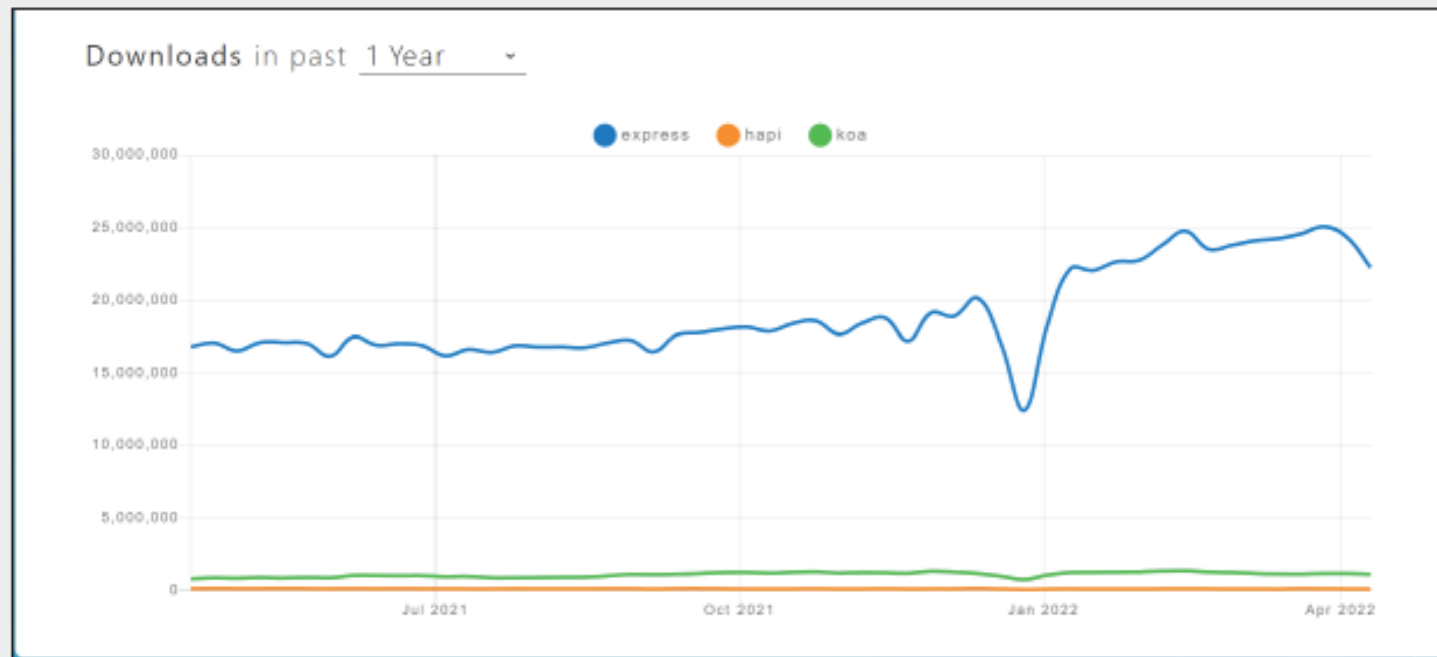
- Introduction to Express.js
- Middleware
- Router
- Req and Res Objects
- Template Engine

Introduction to Express.js

Introduction to Express.js

- Express is a minimal and flexible Node.js web application framework
- When creating a web server using the HTTP module, the code may not be very readable, and its scalability can be limited
 - Using frameworks (e.g., Express, Koa, Hapi)
 - Improves code management and greatly enhances convenience and ease of use

♥ 그림 6-1 express, koa, hapi의 다운로드 수 비교



Getting started with Express.js

- 1. Create a package.json file
 - You can create it manually or generate it using the **npm init** command.
 - Use '**nodemon**' to automatically restart the server when the source code changes.

package.json

```
{
  "name": "learn-express",
  "version": "0.0.1",
  "description": "익스프레스를 배우자",
  "main": "app.js",
  "scripts": {
    "start": "nodemon app"
  },
  "author": "ZeroCho",
  "license": "MIT"
}
```

콘솔

```
$ npm i express
$ npm i -D nodemon
```

Getting started with Express.js

- 2. Write app.js.
 - This file is essential for running the server.
 - Use **app.set('port', port)** to specify the port on which the server will run.
 - Use **app.get('route', router)** to define what the server should do when a GET request is made to a specific route.
 - Use **app.listen(port, callback)** to specify on which port the server should run and provide a callback function.

app.js

```
const express = require('express');

const app = express();
app.set('port', process.env.PORT || 3000);

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

app.listen(app.get('port'), () => {
  console.log(app.get('port'), '번 포트에서 대기 중');
});
```

Getting started with Express.js

- 3. Run the server
 - Execute npm start (defined in package.json) in the console.
 - Open your browser and visit localhost:3000 to check the server.

콘솔

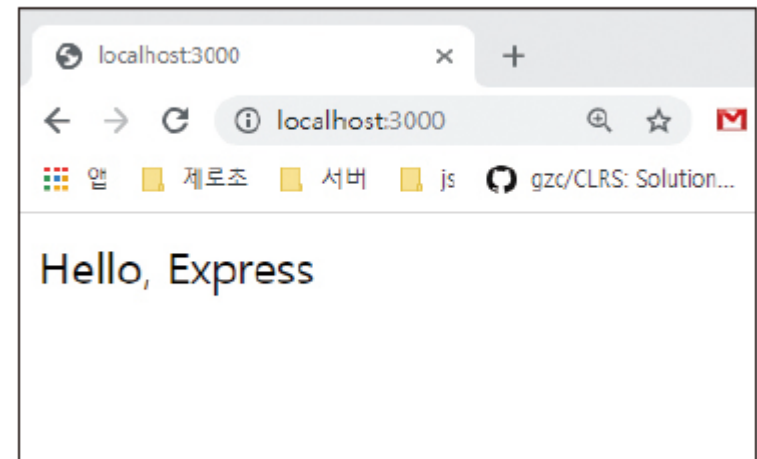
```
$ npm start
```

```
> learn-express@0.0.1 start  
> nodemon app
```

```
[nodemon] 2.0.16
```

```
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching dir(s): *.*  
[nodemon] watching extensions: js,mjs,json  
[nodemon] starting `node app.js`  
3000 번 포트에서 대기 중
```

▼ 그림 6-2 localhost:3000 접속 화면



Getting started with Express.js

- 5. Providing a HTML file
 - Use **res.sendFile** to provide HTML file

index.html

```
<html>
<head>
  <meta charset="UTF-8" />
  <title>익스프레스 서버</title>
</head>
<body>
  <h1>익스프레스</h1>
  <p>배워봅시다.</p>
</body>
</html>
```

app.js

```
const express = require('express');
const path = require('path');

const app = express();
app.set('port', process.env.PORT || 3000);

app.get('/', (req, res) => {
  // res.send('Hello, Express');
  res.sendFile(path.join(__dirname, '/index.html'));
});

app.listen(app.get('port'), () => {
  console.log(app.get('port'), '번 포트에서 대기 중');
});
```

Middleware

Middleware

- Express is built with middleware
 - intermediaries between requests and responses
 - app.use(middleware)** to attach the middleware
 - execute in the order they are defined
 - from top to bottom
 - Middleware functions take three parameters
 - req** for the request
 - res** for the response
 - next** to move to the next middleware

▼ 표 6-1 미들웨어가 실행되는 경우

| | |
|------------------------|-----------------------------|
| app.use(미들웨어) | 모든 요청에서 미들웨어 실행 |
| app.use('/abc', 미들웨어) | abc로 시작하는 요청에서 미들웨어 실행 |
| app.post('/abc', 미들웨어) | abc로 시작하는 POST 요청에서 미들웨어 실행 |

app.js

```
...
app.set('port', process.env.PORT || 3000);

app.use((req, res, next) => {
  console.log('모든 요청에 다 실행됩니다.');
```

```
  next();
});
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);
});

app.listen(app.get('port'), () => {
  ...
});
```

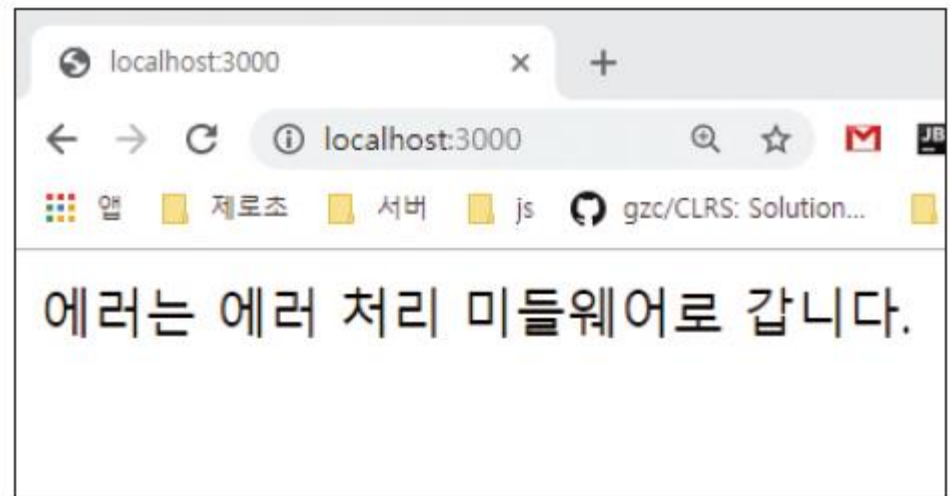
Error handling middleware

- When an error occurs, it can be handled by an error handling middleware
 - There are 4 parameters including `err`, `req`, `res`, and `next`.
 - The first **err** contains information about the error.
 - HTTP status code can be specified with **res.status** method (default 200)
 - Even if you do not connect error handling middleware, Express handles errors on its own.
 - Unless there are special cases, it should be placed at the bottom.

콘솔

모든 요청에 다 실행됩니다.
GET / 요청에서만 실행됩니다.
Error: 에러는 에러 처리 미들웨어로 갑니다.
...

▼ 그림 6-4 localhost:3000 접속 화면



자주 쓰는 미들웨어

- Install morgan, cookie-parser, express-session
 - Equipped with app.use
 - It automatically calls next internally and moves to the next middleware.

콘솔

```
$ npm i morgan cookie-parser express-session dotenv
```

.env

```
COOKIE_SECRET=cookiesecret
```

app.js

```
const express = require('express');
const morgan = require('morgan');
const cookieParser = require('cookie-parser');
const session = require('express-session');
const dotenv = require('dotenv');
const path = require('path');

dotenv.config();
const app = express();
app.set('port', process.env.PORT || 3000);

app.use(morgan('dev'));
app.use('/', express.static(path.join(__dirname, 'public')));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser(process.env.COOKIE_SECRET));
app.use(session({
  resave: false,
  saveUninitialized: false,
  secret: process.env.COOKIE_SECRET,
  cookie: {
    httpOnly: true,
    secure: false,
  },
  name: 'session-cookie',
})));

app.use((req, res, next) => {
  console.log('모든 요청에 다 실행됩니다. ');
  next();
});
...
```

dotenv

- Reads the .env file and creates it as process.env
 - dot(dot) + env
 - cookiesecret value assigned to process.env.COOKIE_SECRET (key=value format)
 - If you write the secret keys in the source code, when the source code is leaked, the secret key will also be leaked.
 - All you have to do is collect secret keys in an .env file and manage the .env file well.

morgan

- The middleware that logs incoming requests and responses to the server.
 - You can choose the level of detail for the logs, including options like dev, tiny, short, common, and combined.

콘솔

```
3000 번 포트에서 대기 중  
모든 요청에 다 실행됩니다.  
GET / 요청에서만 실행됩니다.  
Error: 에러는 에러 처리 미들웨어로 갑니다.  
// 에러 스택 트레이스 생략  
GET / 500 7.409 ms - 50
```

```
app.use(morgan('dev'));
```

- For example: GET / 200 51.267 ms – 1539,
 - which represents the order of HTTP request, request address, status code, response time, and response bytes.
 - In a development environment, the dev option is commonly used, while in a production environment, the combined option is preferred.
- For more detailed logging, the winston package can be used (covered in later).

static

- This is a middleware that serves static files.
 - You provide the path to the static files as an argument.
 - There's no need to manually read the file with `fs.readFile` if the file exists.
 - If the requested file is not found, it will automatically call next to proceed to the next middleware. If the file is found, the subsequent middleware will not execute.

```
app.use('요청 경로', express.static('실제 경로'));
```

```
app.use('/', express.static(path.join(__dirname, 'public')));
```

- You can also make the content request address and the actual content path different.
 - For example, the request address might be **localhost:3000/stylesheets/style.css**, but the actual content path is **/public/stylesheets/style.css**. This approach enhances security by making it difficult to discern the server's structure.

body-parser

- This is middleware that interprets the request body
 - making it useful for processing form data and AJAX request data
 - The json middleware interprets the request body if it's in JSON format, and the urlencoded middleware interprets form requests.
 - When making PUT, PATCH, or POST requests, the data from the frontend is placed in req.body.

```
app.use(express.json());  
app.use(express.urlencoded({ extended: false }));
```

- If you're dealing with buffer data or text data, you'll need to install the body-parser middleware directly

콘솔

```
$ npm i body-parser
```

```
const bodyParser = require('body-parser');  
app.use(bodyParser.raw());  
app.use(bodyParser.text());
```

- For handling multipart data like images or videos, you'll need to use a different middleware, such as the multer package (covered in later).

cookie-parser

- 요청 헤더의 쿠키를 해석해주는 미들웨어
 - parseCookies 함수와 기능 비슷
 - req.cookies 안에 쿠키들이 들어있음

app.js

```
app.use(cookieParser(비밀키));
```

- 비밀 키로 쿠키 뒤에 서명을 붙여 내 서버가 만든 쿠키임을 검증할 수 있음
- 실제 쿠키 옵션들을 넣을 수 있음
 - expires, domain, httpOnly, maxAge, path, secure, sameSite 등
 - 지울 때는 clearCookie로(expires와 maxAge를 제외한 옵션들이 일치해야 함)

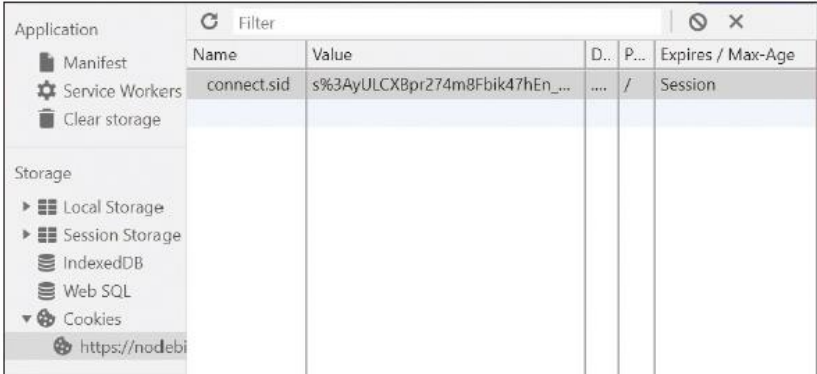
```
res.cookie('name', 'zerocho', {  
  expires: new Date(Date.now() + 900000),  
  httpOnly: true,  
  secure: true,  
});  
res.clearCookie('name', 'zerocho', { httpOnly: true, secure: true });
```

express-session

- Middleware for session management

```
app.use(session({
  resave: false,
  saveUninitialized: false,
  secret: process.env.COOKIE_SECRET,
  cookie: {
    httpOnly: true,
    secure: false,
  },
  name: 'session-cookie',
}));
```

```
req.session.name = 'zerocho'; // 세션 등록
req.sessionID; // 세션 아이디 확인
req.session.destroy(); // 세션 모두 제거
```



The screenshot shows the Chrome DevTools Application tab. The left sidebar has a tree view with 'Application' expanded, showing 'Manifest', 'Service Workers', 'Clear storage', 'Storage' (with sub-items: Local Storage, Session Storage, IndexedDB, Web SQL, Cookies), and 'https://nodebi...'. The main panel shows a table of session storage data. The table has columns: Name, Value, D., P., and Expires / Max-Age. One entry is visible: 'connect.sid' with a long alphanumeric value, '...', '/', and 'Session'.

| Name | Value | D. | P. | Expires / Max-Age |
|-------------|-------------------------------|-----|----|-------------------|
| connect.sid | s%3AyULCXBpr274m8Fbik47hEn... | ... | / | Session |

- Settings for session cookies (secret: cookie encryption, cookie: session cookie option)
- resave**: Whether to save again even if no modifications are made to the session when a request is made.
- saveUninitialized**: Whether to save the session even if there is no history to save in the session. Manual saving is also possible with `req.session.save`, but there is little to do.

Characteristics of middleware

- Function with req, res, next as parameters

```
app.use((req, res, next) => {  
  console.log('모든 요청에 다 실행됩니다.');
```

```
  next();  
});
```

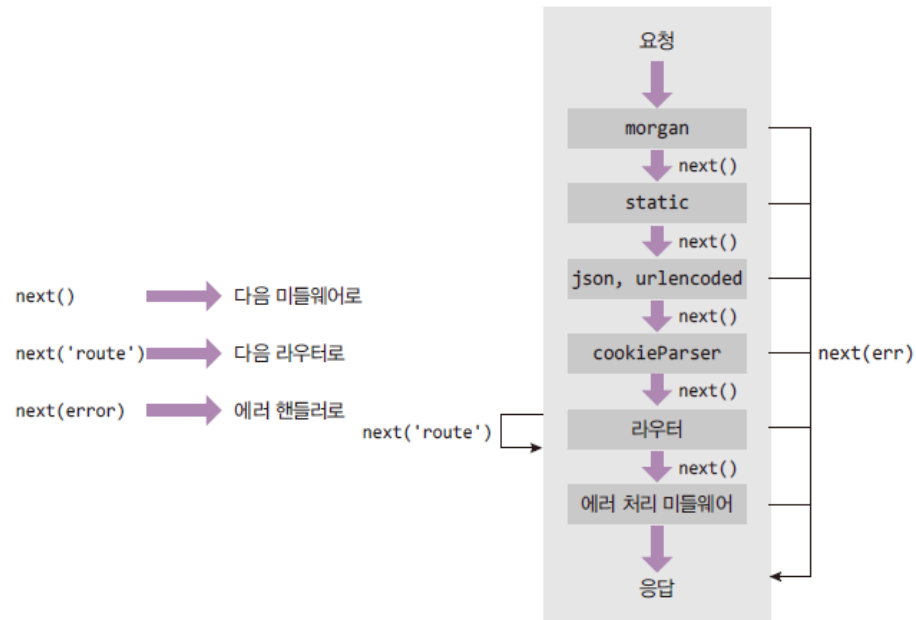
- Express middlewares can also be abbreviated as follows:
 - Order is important
 - When a file is found in static middleware, next is not called, so json, urlencoded, and cookieParser are not executed.

```
app.use(  
  morgan('dev'),  
  express.static('/', path.join(__dirname, 'public')),  
  express.json(),  
  express.urlencoded({ extended: false }),  
  cookieParser(process.env.COOKIE_SECRET),  
);
```

next

- You must call next to move on to the next code.
 - Commenting out next causes no response to be sent
 - This is because it does not move on to the next middleware (router middleware).
 - If you enter a value as an argument in next, it moves to the error handler (in case of 'route', to the next router)

♥ 그림 6-6 next의 동작



♥ 그림 6-7 에러 처리 미들웨어로 에러 보내기

next(err)

(err, req, res, next) => { }

Passing data between middleware

- Data can be passed by putting a value in a req or res object.
- Difference from app.set:
 - **app.set** persists throughout the server, **req** and **res** only persist for one request.
- In general, **res.locals** object is often used for data transfer.

```
app.use((req, res, next) => {  
  res.locals.data = '데이터 넣기';  
  next();  
}, (req, res, next) => {  
  console.log(res.locals.data); // 데이터 받기  
  next();  
});
```

Extending middleware

- How to put middleware inside middleware
 - The two codes below do the same thing

```
app.use(morgan('dev'));  
// 또는  
app.use((req, res, next) => {  
  morgan('dev')(req, res, next);  
});
```

- Can be used in various ways as shown below

```
app.use((req, res, next) => {  
  if (process.env.NODE_ENV === 'production') {  
    morgan('combined')(req, res, next);  
  } else {  
    morgan('dev')(req, res, next);  
  }  
});
```

Multipart data format

- When the enctype of the form tag is multipart/form-data
 - body-parser cannot interpret request body
 - Requires multer package

콘솔

```
$ npm i multer
```

multipart.html

```
<form action="/upload" method="post" enctype="multipart/form-data">
  <input type="file" name="image" />
  <input type="text" name="title" />
  <button type="submit">업로드</button>
</form>
```

▼ Form Data

view parsed

```
-----WebKitFormBoundary0a6rH3D3NJ1cNo85
Content-Disposition: form-data; name="image"; filename="토사.jpg"
Content-Type: image/jpeg
```

```
-----WebKitFormBoundary0a6rH3D3NJ1cNo85
Content-Disposition: form-data; name="title"
```

제목

```
-----WebKitFormBoundary0a6rH3D3NJ1cNo85--
```

Setting up multer

- When calling the multer function:
 - **storage** contains information about where to store the files.
 - **diskStorage** indicates that the uploaded files should be stored on the hard disk.
 - **destination** specifies the storage path and can be passed as the second argument to the done function.
 - **filename** defines the name for the stored file (in the format of filename + date + extension) and can be passed to the done function.
 - **limits** can be used to restrict the number of files or the file size.

```
const multer = require('multer');

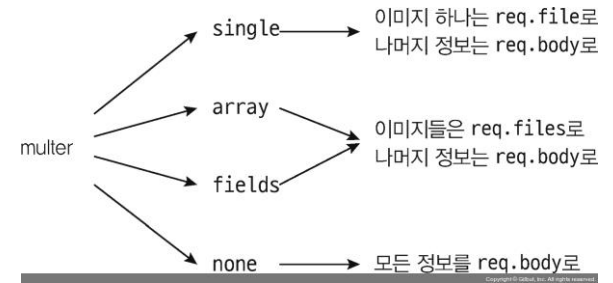
const upload = multer({
  storage: multer.diskStorage({
    destination(req, file, done) {
      done(null, 'uploads/');
    },
    filename(req, file, done) {
      const ext = path.extname(file.originalname);
      done(null, path.basename(file.originalname, ext) + Date.now() + ext);
    },
  }),
  limits: { fileSize: 5 * 1024 * 1024 },
});
```

- In actual server operations, it's often better to store files in storage services like S3 instead of on the server's disk.
 - You can achieve this by simply changing the storage configuration.

multer middlewares

- single, none, array, fields middleware exists
 - single** is for uploading one file, **none** is for not uploading any file at all.
 - Save upload information in **req.file**

```
app.post('/upload', upload.single('image'), (req, res) => {  
  console.log(req.file, req.body);  
  res.send('ok');  
});  
  
app.post('/upload', upload.none(), (req, res) => {  
  console.log(req.body);  
  res.send('ok');  
});
```



- array** and **fields** are used when uploading multiple files
- array** when there are multiple files under one request body name
- fields** is one file under multiple request body names.
- In both cases, uploaded image information exists under req.files

```
app.post('/upload', upload.array('many'), (req, res) => {  
  console.log(req.files, req.body);  
  res.send('ok');  
});  
  
app.post('/upload',  
  upload.fields([ { name: 'image1' }, { name: 'image2' } ]),  
  (req, res) => {  
    console.log(req.files, req.body);  
    res.send('ok');  
  },  
);
```

multer example

- [\[source link\]](#)

```
...
const multer = require('multer');
const fs = require('fs');

try {
  fs.readdirSync('uploads');
} catch (error) {
  console.error('uploads 폴더가 없어 uploads 폴더를 생성합니다.');
```

fs.mkdirSync('uploads');

```
}
const upload = multer({
  storage: multer.diskStorage({
    destination(req, file, done) {
      done(null, 'uploads/');
    },
    filename(req, file, done) {
      const ext = path.extname(file.originalname);
      done(null, path.basename(file.originalname, ext) + Date.now() + ext);
    },
  }),
  limits: { fileSize: 5 * 1024 * 1024 },
});
app.get('/upload', (req, res) => {
  res.sendFile(path.join(__dirname, 'multipart.html'));
});
app.post('/upload', upload.single('image'), (req, res) => {
  console.log(req.file);
  res.send('ok');
});
...
```

Router

express.Router

- Can prevent app.js from becoming long
 - userRouter's get is /user and / combined to become GET /user/

routes/index.js

```
const express = require('express');

const router = express.Router();

// GET / 라우터
router.get('/', (req, res) => {
  res.send('Hello, Express');
});

module.exports = router;
```

routes/user.js

```
const express = require('express');

const router = express.Router();

// GET /user 라우터
router.get('/', (req, res) => {
  res.send('Hello, User');
});

module.exports = router;
```

app.js

```
...
const path = require('path');

dotenv.config();
const indexRouter = require('./routes');
const userRouter = require('./routes/user');
...
name: 'session-cookie',
}));

app.use('/', indexRouter);
app.use('/user', userRouter);

app.use((req, res, next) => {
  res.status(404).send('Not Found');
});

app.use((err, req, res, next) => {
  ...
```

Route Parameter

- If you enter :id, you can access it as req.params.id.
 - Make dynamically changing parts into route parameters

```
router.get('/user/:id', function(req, res) {  
  console.log(req.params, req.query);  
});
```

- Must be located behind a regular router

```
router.get('/user/:id', function(req, res) {  
  console.log('애만 실행됩니다.');
```

```
});  
router.get('/user/like', function(req, res) {  
  console.log('전혀 실행되지 않습니다.');
```

```
});
```

- e.g., /users/123?limit=5&skip=10

콘솔

```
{ id: '123' } { limit: '5', skip: '10' }
```

404 middleware

- Create a 404 router in case no router matches the request

```
app.use((req, res, next) => {  
  res.status(404).send('Not Found');  
});
```

- Without this, the string "Cannot GET address" is simply displayed.

Group routers

- When there is code with the same address but different methods

```
router.get('/abc', (req, res) => {  
  res.send('GET /abc');  
});  
  
router.post('/abc', (req, res) => {  
  res.send('POST /abc');  
});
```

- bundled with router.route

```
router.route('/abc')  
  .get((req, res) => {  
    res.send('GET /abc');  
  })  
  .post((req, res) => {  
    res.send('POST /abc');  
  });
```

req and res objects

req object

- **req.app:** You can access the app object through the req object. It can be used like req.app.get('port').
- **req.body:** This is an object that represents the parsed request body created by the body-parser middleware.
- **req.cookies:** This object contains the parsed cookies from the request, created by the cookie-parser middleware.
- **req.ip:** This property holds the IP address of the request.
- **req.params:** An object that contains information about route parameters.
- **req.query:** An object that holds information about the query string.
- **req.signedCookies:** If you are using signed cookies, they will be available here instead of in req.cookies.
- **req.get(header name):** This method is used to retrieve the value of a specific header from the request.

res object

- **res.app**: Similar to req.app, you can access the app object through the res object.
- **res.cookie(key, value, options)**: This method is used to set a cookie.
- **res.clearCookie(key, value, options)**: This method is used to remove a cookie.
- **res.end()**: It sends a response without any data.
- **res.json(JSON)**: Sends a response in JSON format.
- **res.redirect(url)**: Sends a response with a redirection to the specified URL.
- **res.render(view, data)**: This method is used for rendering a template engine, which will be covered in the next section.
- **res.send(data)**: Sends a response along with data. The data can be a string, HTML, a buffer, an object, or an array.
- **res.sendFile(path)**: Responds with the file located at the specified path.
- **res.setHeader(header, value)**: Sets the response's header.
- **res.status(code)**: Specifies the HTTP status code for the response.

tips

- Supports method chaining

```
res
  .status(201)
  .cookie('test', 'test')
  .redirect('/admin');
```

- Responses must be sent only once



응답을 여러 번 보내는 경우

하나의 요청에 대한 응답은 한 번만 보내야 합니다. 두 번 이상 보내면 다음과 같은 예러가 발생합니다.

콘솔

```
Error: Can't set headers after they are sent.
    at validateHeader (_http_outgoing.js:489:11)
    at ServerResponse.setHeader (_http_outgoing.js:496:3)
    ...
```

이와 같은 예러를 보았다면 라우터에서 res 객체의 응답 메서드가 두 번 이상 사용되지 않았는지 점검해보아야 합니다.

Template Engine

Template Engine

- Improve static shortcomings of HTML
 - Loops, conditional statements, variables, etc. can be used.
 - Dynamic page creation possible
 - Similar to PHP and JSP
- Let's take a look at Pug and Nunjucks.

Pug

- The syntax is similar to Ruby, so the amount of code is greatly reduced.
 - It is very different from HTML, so likes and dislikes differ.
 - Connecting pug with app.set to express
 - **views** specifies the folder where the template files are located.

app.js

```
...  
app.set('port', process.env.PORT || 3000);  
app.set('views', path.join(__dirname, 'views'));  
app.set('view engine', 'pug');  
  
app.use(morgan('dev'));  
...
```

```
routes  
views  
.env  
app.js  
package-lock.json  
package.json
```

♥ 그림 6-11 퍼그 로고



Pug – HTML elements

| 퍼그 | HTML |
|---|---|
| <pre>doctype html html head title= title link(rel='stylesheet', href='/ stylesheets/style.css')</pre> | <pre><!DOCTYPE html> <html> <head> <title>익스프레스</title> <link rel="stylesheet" href="/style.css" /> </head> </html></pre> |
| 퍼그 | HTML |
| <pre>#login-button .post-image span#highlight p.hidden.full</pre> | <pre><div id="login-button"></div> <div class="post-image"></div> <p class="hidden full"></p></pre> |
| 퍼그 | HTML |
| <pre>p Welcome to Express button(type='submit') 전송</pre> | <pre><p>Welcome to Express</p> <button type="submit">전송</button></pre> |

Pug – HTML elements

| 퍼그 | HTML |
|--|---|
| <pre>p 안녕하세요. 여러 줄을 입력합니다. br 태그도 중간에 넣을 수 있습니다.</pre> | <pre><p> 안녕하세요. 여러 줄을 입력합니다. 태그도 중간에 넣을 수 있습니다. </p></pre> |
| 퍼그 | HTML |
| <pre>style. h1 { font-size: 30px; } script. const message = 'Pug'; alert(message);</pre> | <pre><style> h1 { font-size: 30px; } </style> <script> const message = 'Pug'; alert(message); </script></pre> |

Pug - variable

- Put Pug variable in second argument object in res.render

```
router.get('/', function(req, res, next) {  
  res.render('index', { title: 'Express' });  
});
```

extends layout

block content

h1= title

p Welcome to #{title}

- It is also possible to put it in the res.locals object (shared between middleware)

```
router.get('/', function(req, res, next) {  
  res.locals.title = 'Express';  
  res.render('index');  
});
```

- Variables can be rendered with = or #{ } (JavaScript syntax can be used after =)

| 퍼그 | HTML |
|---------------------------------------|---|
| h1= title | <h1>Express</h1> |
| p Welcome to #{title} | <p>Welcome to Express</p> |
| button(class=title, type='submit') 전송 | <button class="Express" type="submit">전송</button> |
| input(placeholder=title + ' 연습') | <input placeholder="Express 연습" /> |

Pug – variables in file

- Variables can be declared in pug files
 - Use JavaScript afterwards

| 퍼그 | HTML |
|--|---|
| <pre>- const node = 'Node.js' - const js = 'Javascript' p # {node}와 # {js}</pre> | <pre><p>Node.js와 Javascript</p></pre> |

- Variable values may not be escaped (auto-escaping)

| 퍼그 | HTML |
|--|---|
| <pre>p= '이스케이프' p!= '이스케이프하지 않음'</pre> | <pre><p>&lt;strong&gt;이스케이프&lt;/strong&gt;</p> <p>이스케이프하지 않음</p></pre> |

Pug – Loops

- You can loop through for in or each in.

| 퍼그 | HTML |
|--|---|
| <pre>ul each fruit in ['사과', '배', '오렌지', '바나나', '복숭아'] li= fruit</pre> | <pre> 사과 배 오렌지 바나나 복숭아 </pre> |

- Can get value and index

| 퍼그 | HTML |
|---|---|
| <pre>ul each fruit, index in ['사과', '배', '오렌지', '바나나', '복숭아'] li= (index + 1) + '번째 ' + fruit</pre> | <pre> 1번째 사과 2번째 배 3번째 오렌지 4번째 바나나 5번째 복숭아 </pre> |

Pug – Conditionals

- if else if else statement, case when statement can be used

| 퍼그 | HTML |
|---|---|
| <pre>if isLoggedIn div 로그인 되었습니다. else div 로그인이 필요합니다.</pre> | <pre><!-- isLoggedIn이 true일 때 --> <div>로그인 되었습니다.</div> <!-- isLoggedIn이 false일 때 --> <div>로그인이 필요합니다.</div></pre> |
| 퍼그 | HTML |
| <pre>case fruit when 'apple' p 사과입니다. when 'banana' p 바나나입니다. when 'orange' p 오렌지입니다. default p 사과도 바나나도 오렌지도 아닙니다.</pre> | <pre><!-- fruit이 apple일 때 --> <p>사과입니다.</p> <!-- fruit이 banana일 때 --> <p>바나나입니다.</p> <!-- fruit이 orange일 때 --> <p>오렌지입니다.</p> <!-- 기본값 --> <p>사과도 바나나도 오렌지도 아닙니다.</p></pre> |

Pug – include

- You can put other pug files in a pug file
 - It is convenient because common parts such as header, footer, and navigation can be managed separately.
 - Specify file path with include

| 퍼그 | HTML |
|--|--|
| header.pug header a(href='/') Home a(href='/about') About | <header> Home About </header> |
| footer.pug footer div 푸터입니다 | <main> <h1>메인 파일</h1> <p>다른 파일을 include할 수 있습니다.</p> </main> <footer> <div>푸터입니다.</div> |
| main.pug include header main h1 메인 파일 p 다른 파일을 include할 수 있습니다. include footer | </footer> |

Pug – extends and block

- You can decide the layout

| 퍼그 | HTML |
|--|--|
| <pre>layout.pug doctype html html head title= title link(rel='stylesheet', href='/style. css') block style body header 헤더입니다. block content footer 푸터입니다. block script</pre> | <pre><!DOCTYPE html> <html> <head> <title>Express</title> <link rel="stylesheet" href="/style.css" /> </head> <body> <header>헤더입니다.</header> <main> <p>내용입니다.</p> </main> <footer>푸터입니다.</footer> <script src="/main.js"></script> </body> </html></pre> |
| <pre>body.pug extends layout block content main p 내용입니다. block script script(src="/main.js")</pre> | |

Nunjucks

- If you are not used to Pug's syntax, it is better to use Nunjucks.
 - Clear Pug and Install Nunjucks
 - The extension is html or njk (view engine as njk)

콘솔

```
$ npm i nunjucks
```

view engine을 퍼그 대신 년적스로 교체합니다.

app.js

```
...  
const path = require('path');  
const nunjucks = require('nunjucks');  
  
dotenv.config();  
const indexRouter = require('./routes');  
const userRouter = require('./routes/user');  
  
const app = express();  
app.set('port', process.env.PORT || 3000);  
app.set('view engine', 'html');  
  
nunjucks.configure('views', {  
  express: app,  
  watch: true,  
});  
  
app.use(morgan('dev'));  
...
```

Nunjucks - variable

- `{{variable}}`

년적스

```
<h1>{{title}}</h1>
<p>Welcome to {{title}}</p>
<button class="{{title}}" type="submit">전송</button>
<input placeholder="{{title}} 연습" />
```

- Internal variables can be declared `{%set JavaScript syntax}`

| 년적스 | HTML |
|--|---|
| <pre>{% set node = 'Node.js' %} {% set js = 'Javascript' %} <p>{{node}}와 {{js}}</p></pre> | <pre><p>Node.js와 Javascript</p></pre> |
| 년적스 | HTML |
| <pre><p>{{ '이스케이프' }}</p> <p>{{ '이스케이프하지 않음' safe }}</p></pre> | <pre><p>&lt;strong&gt;이스케이프&lt;/strong&gt;</p> <p>이스케이프하지 않음</p></pre> |

Nunjucks - Loops

- Write for in in {% %} (index is loop keyword)

| 넌적스 | HTML |
|---|---|
| <pre> {% set fruits = ['사과', '배', '오렌지', '바나나', '복숭아'] %} {% for item in fruits %} {{item}} {% endfor %} </pre> | <pre> 사과 배 오렌지 바나나 복숭아 </pre> |

| 넌적스 | HTML |
|--|---|
| <pre> {% set fruits = ['사과', '배', '오렌지', '바나나', '복숭아'] %} {% for item in fruits %} {{loop.index}}번째 {{item}} {% endfor %} </pre> | <pre> 1번째 사과 2번째 배 3번째 오렌지 4번째 바나나 5번째 복숭아 </pre> |

Nunjucks - Conditionals

- Write conditional statements in {% if %}

| 넌적스 | HTML |
|--|--|
| <pre>{% if isLoggedIn %} <div>로그인 되었습니다.</div> {% else %} <div>로그인이 필요합니다.</div> {% endif %}</pre> | <pre><!-- isLoggedIn이 true일 때 --> <div>로그인 되었습니다.</div> <!-- isLoggedIn이 false일 때 --> <div>로그인이 필요합니다.</div></pre> |

| 넌적스 | HTML |
|--|---|
| <pre>{% if fruit == 'apple' %} <p>사과입니다.</p> {% elif fruit == 'banana' %} <p>바나나입니다.</p> {% elif fruit == 'orange' %} <p>오렌지입니다.</p> {% else %} <p>사과도 바나나도 오렌지도 아닙니다.</p> {% endif %}</pre> | <pre><!-- fruit이 apple일 때 --> <p>사과입니다.</p> <!-- fruit이 banana일 때 --> <p>바나나입니다.</p> <!-- fruit이 orange일 때 --> <p>오렌지입니다.</p> <!-- 기본값 --> <p>사과도 바나나도 오렌지도 아닙니다.</p></pre> |

Nunjucks - include

- Files can load other files
 - You can put the file path in include

| 넌적스 | HTML |
|---|---|
| header.html <pre><header> Home About </header></pre> | <pre><header> Home About </header> <main> <h1>메인 파일</h1> <p>다른 파일을 include할 수 있습니다.</p> </main> <footer> <div>푸터입니다.</div> </footer></pre> |
| footer.html <pre><footer> <div>푸터입니다.</div> </footer></pre> | |
| main.html <pre>{% include "header.html" %} <main> <h1>메인 파일</h1> <p>다른 파일을 include할 수 있습니다.</p> </main> {% include "footer.html" %}</pre> | |

Nunjucks - Layout

- You can decide the layout

| 넉스 | HTML |
|---|--|
| <pre>layout.html <!DOCTYPE html> <html> <head> <title>{{title}}</title> <link rel="stylesheet" href="/style. css" /> {% block style %} {% endblock %} </head> <body> <header>헤더입니다.</header> {% block content %} {% endblock %} <footer>푸터입니다.</footer> {% block script %} {% endblock %} </body> </html></pre> | <pre><!DOCTYPE html> <html> <head> <title>Express</title> <link rel="stylesheet" href="/style.css" /> </head> <body> <header>헤더입니다.</header> <main> <p>내용입니다.</p> </main> <footer>푸터입니다.</footer> <script src="/main.js"></script> </body> </html></pre> |
| <pre>body.html {% extends 'layout.html' %} {% block content %} <main> <p>내용입니다.</p> </main> {% endblock %} {% block script %} <script src="/main.js"></script> {% endblock %}</pre> | |

Error handling middleware

- When an error occurs, template engine variables are set and the error template is rendered regardless of the template engine.
 - Template engine variables can be created using **res.locals.variable**.
 - process.env.NODE_ENV** is a property that distinguishes whether it is a development environment or a deployment environment.

app.js

```
...
app.use((req, res, next) => {
  const error = new Error(`${req.method} ${req.url} 라우터가 없습니다.`);
  error.status = 404;
  next(error);
});

app.use((err, req, res, next) => {
  res.locals.message = err.message;
  res.locals.error = process.env.NODE_ENV !== 'production' ? err : {};
  res.status(err.status || 500);
  res.render('error');
});
...
```

{% extends 'layout.html' %}

{% block content %}

<h1>{{message}}</h1>

<h2>{{error.status}}</h2>

<pre>{{error.stack}}</pre>

{% endblock %}

♥ 그림 6-12 에러 스택 트레이스

GET /abc 라우터가 없습니다.

404

Error: GET /abc 라우터가 없습니다.

```
at C:\Users\spk\OneDrive\Projects\nodejs-book\ch06\5.3\learn-express\app.js:41:18
at Layer.handle [as handle_request] (C:\Users\spk\OneDrive\Projects\nodejs-book\ch06\5.3\learn-express\node_modules\express\lib\router\layer.js:95:5)
at trim_prefix (C:\Users\spk\OneDrive\Projects\nodejs-book\ch06\5.3\learn-express\node_modules\express\lib\router\index.js:317:13)
at C:\Users\spk\OneDrive\Projects\nodejs-book\ch06\5.3\learn-express\node_modules\express\lib\router\index.js:284:7
at Function.process_params (C:\Users\spk\OneDrive\Projects\nodejs-book\ch06\5.3\learn-express\node_modules\express\lib\router\index.js:335:12)
at next (C:\Users\spk\OneDrive\Projects\nodejs-book\ch06\5.3\learn-express\node_modules\express\lib\router\index.js:275:10)
at C:\Users\spk\OneDrive\Projects\nodejs-book\ch06\5.3\learn-express\node_modules\express\lib\router\index.js:635:15
at next (C:\Users\spk\OneDrive\Projects\nodejs-book\ch06\5.3\learn-express\node_modules\express\lib\router\index.js:260:14)
at Function.handle (C:\Users\spk\OneDrive\Projects\nodejs-book\ch06\5.3\learn-express\node_modules\express\lib\router\index.js:174:3)
at router (C:\Users\spk\OneDrive\Projects\nodejs-book\ch06\5.3\learn-express\node_modules\express\lib\router\index.js:47:12)
```