

네트워크서비스 프로토콜

12주차 2

» 소프트웨어학부

» 김형균 교수



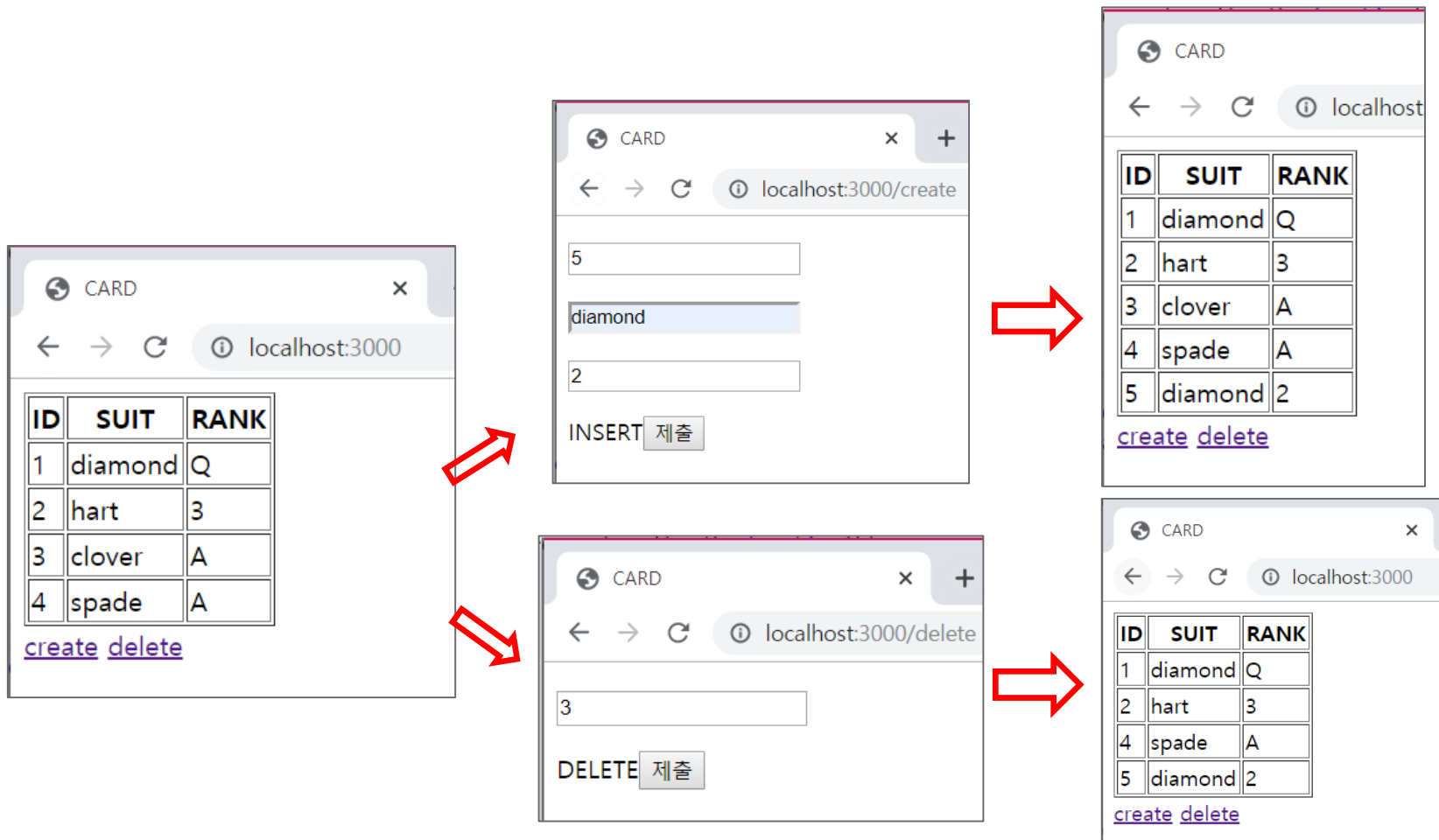
수업에 들어가며

- 지난 시간 복습
 - express 라우팅 실습
- 오늘 학습할 내용
 - Express 미들웨어
 - 미들웨어의 유형
 - 기존 미들웨어 사용하기 body-parser
 - 기존 미들웨어 사용하기 compression
 - Express 미들웨어 만들기

실습



- 앞서 작성된 본 사이트를 익스프레스 모듈을 이용해 변환하시오





기존 라우팅 처리 확인

```
1  var url = require('url');
2  var qs = require('querystring');
3  var http = require('http');
4  var mysql = require('mysql');
5  var db = mysql.createConnection({
6    host: 'localhost',
7    user: 'root',
8    password: '50031',
9    database: 'games'
10 });
11 db.connect();
12
13 var app = http.createServer(function(request, response){
14   var _url = request.url;
15   var pathname = url.parse(_url, true).pathname;
16 >   if(pathname == '/') { ...
40 >   } else if(pathname == '/create') { ...
56 >   } else if(pathname === '/create_process') { ...
74 >   } else if(pathname == '/delete') { ...
88 >   } else if(pathname === '/delete_process') { ...
103   };
104 });
105 app.listen(3000);
```



» 작업 폴더에 익스프레스 모듈 설치

```
PS C:\test> npm install express --save
npm WARN saveError ENOENT: no such file or directory, open 'C:\test\package.json'
npm WARN enoent ENOENT: no such file or directory, open 'C:\test\package.json'
npm WARN test No description
npm WARN test No repository field.
npm WARN test No README data
npm WARN test No license field.

+ express@4.17.1
added 49 packages from 36 contributors and audited 158 packages in 2.045s
found 0 vulnerabilities

PS C:\test> █
```



선언부 처리와 라우팅 모듈 정의

```
1  var express = require('express')
2  var app = express()
3  var qs = require('querystring');
4  var mysql = require('mysql');
5  var db = mysql.createConnection({
6    host: 'localhost',
7    user: 'root',
8    password: '50031',
9    database: 'games'
10 });
11 db.connect();
12
104 app.listen(3000, function() {
105   console.log('app listening on port 3000!')
106 });
```











Express 미들웨어

» 익스프레스 프레임워크의 장점 중 하나가 미들웨어를 사용한다는 것

» Middleware 란

- 이름처럼 요청에 대한 응답 과정 중간에 끼서 어떠한 동작을 해주는 프로그램
- 익스프레스는 요청이 들어올 때 그에 따른 응답을 보내주는데, 응답을 보내기 전에 미들웨어가 지정한 동작을 수행함

» 대표적인 미들웨어

- Morgan, Compression, Session, Body-parser, Cookie-parser, Method-override, Cors, Multer 등이 있습니다. 모두 npm에서 다운받을 수 있습니다.



미들웨어의 유형

» Application

- 어플리케이션 전역에서 처리가 가능한 미들웨어로 어플리케이션 자체에 request가 발생할 때마다 실행됩니다.

» Router

- 동작방식은 Application-level 미들웨어랑 같습니다. 하지만 라우터 단위로 묶어 놓고 `express.Router()`의 객체를 사용해야 한다는 차이점이 있습니다. Router-level 미들웨어를 사용한다면 어플리케이션에 Path별 요청에 따른 동작 방식을 모듈화하여 관리할 수 있습니다.

» Error Handling

- 에러 처리를 담당하는 미들웨어입니다. 이러한 유형의 미들웨어는 반드시 네 개의 인자를 매개변수로 받아 이 미들웨어가 에러를 담당하는 미들웨어라는 것을 식별해야 합니다.

» Third-party

- 기본적으로 주어지는 Built-in middleware 외에 추가로 설치하여 사용해야하는 미들웨어를 Third-party middleware라고 합니다.

미들웨어의 유형

1. Application-level middleware



- » 어플리케이션 전역에서 처리가 가능한 미들웨어로 어플리케이션 자체에 request가 발생할 때마다 실행됩니다. 이런 어플리케이션 레벨의 미들웨어는 다음과 같이 사용합니다.
 - `app.use()`
 - `app.METHOD()` - HTTP method
- » 미들웨어를 사용한다고 선언을 하면 기본적으로 미들웨어는 미들웨어 스택에 쌓이게 됩니다.
- » 어플리케이션에 대한 request가 들어올 때마다 이 스택을 통과하면서 request에 관한 처리를 하고 response를 하게 됩니다.

미들웨어의 유형

1. Application-level middleware



```
app.use(function(req, res, next){
  console.log('Time: ', Date.now())
  next()
}) // ----- 1

app.use('/index', function(req, res, next){
  /*
    Code
  */
}) // ----- 2
```

- » app.use() 를 사용할 때는 path를 지정할 수 있습니다.
- » 첫 번째 미들웨어는 어떤 request가 들어와도 작동을 하는 것이고
- » 두 번째 미들웨어는 /index의 path를 통해 들어오는 요청에 대해서만 작동하는 미들웨어입니다.

미들웨어의 유형

2. Router-level middleware



- 동작방식은 Application-level 미들웨어랑 같습니다.
- 하지만 라우터 단위로 묶어 넣고 `express.Router()`의 객체를 사용해야 한다는 차이점이 있습니다.
- Router-level 미들웨어를 사용한다면 어플리케이션에 Path별 요청에 따른 동작 방식을 모듈화하여 관리할 수 있습니다.
- 만일 `/user/mypage`의 path로 GET 요청이 들어온다면 `user`라는 라우터 미들웨어가 처리하는데 그 안의 내용을 보면 `router.get('/mypage')` 에서 처리를 해주는 것을 알 수 있습니다. 이런 방법으로 path별로 처리해줄 미들웨어를 모듈화하여 관리할 수 있습니다.

```
// user.js
var app = express()
var router = express.Router()

router.get('/mypage', function(req, res, next){
  // Code
})

router.get('/articles', function(req, res, next){
  // Code
})

router.post('/newArticle', function(req, res, next){
  // Code
})

module.exports = router
```

```
// app.js
var app = express();
var user = require('user.js')

app.use('/user', user)
```




기존 미들웨어 사용하기 body-parser

» 요청의 본문을 해석해주는 미들웨어

- 폼 데이터나 AJAX 요청의 데이터 처리

» 설치

- `$ npm install body-parser --save`

» API

- `var bodyParser = require('body-parser')`

» parse application/x-www-form-urlencoded

- `app.use(bodyParser.urlencoded({ extended: false }))`

```
1  var express = require('express')
2  var app = express()
3  var fs = require('fs');
4  var template = require('./lib/template.js');
5  var path = require('path');
6  var qs = require('querystring');
7  var sanitizeHtml = require('sanitize-html');
8  var bodyParser = require('body-parser');
9
10 app.use(bodyParser.urlencoded({ extended: false }));
11
```



```
69 app.post('/create_process', function(request, response){
70   var post = request.body;
71   var title = post.title;
72   var description = post.description;
73   fs.writeFile(`data/${title}`, description, 'utf8', function(err){
74     response.redirect(`/page/${title}`);
75   });
76 });
```

```
69 app.post('/create_process', function(request, response){
70   var body = '';
71   request.on('data', function(data){
72     body = body + data;
73   });
74   request.on('end', function(){
75     var post = qs.parse(body);
76     var title = post.title;
77     var description = post.description;
78     fs.writeFile(`data/${title}`, description, 'utf8', function(err){
79       response.writeHead(302, {Location: `/?id=${title}`});
80       response.end();
81     })
82   });
83 });
```



```

106  ✓ app.post('/update_process', function(request, response){
107      var post = request.body;
108      var id = post.id;
109      var title = post.title;
110      var description = post.description;
111  ✓   fs.rename(`data/${id}`, `data/${title}`, function(error){
112  ✓       fs.writeFile(`data/${title}`, description, 'utf8', function(err){
113           response.redirect(`/page/${title}`);
114       })
115   });
116 });

```

```

106  ✓ app.post('/update_process', function(request, response){
107      var body = '';
108  ✓   request.on('data', function(data){
109       body = body + data;
110   });
111  ✓   request.on('end', function(){
112       var post = qs.parse(body);
113       var id = post.id;
114       var title = post.title;
115       var description = post.description;
116  ✓   fs.rename(`data/${id}`, `data/${title}`, function(error){
117  ✓       fs.writeFile(`data/${title}`, description, 'utf8', function(err){
118           response.redirect(`/page/${title}`);
119       })
120   });
121 });
122 });

```



```
118 app.post('/delete_process', function(request, response){
119     var post = request.body;
120     var id = post.id;
121     var filteredId = path.parse(id).base;
122     fs.unlink(`data/${filteredId}`, function(error){
123         response.redirect('/');
124     });
125 });
```

```
118 ✓ app.post('/delete_process', function(request, response){
119     var body = '';
120 ✓ request.on('data', function(data){
121     body = body + data;
122 });
123 ✓ request.on('end', function(){
124     var post = qs.parse(body);
125     var id = post.id;
126     var filteredId = path.parse(id).base;
127 ✓ fs.unlink(`data/${filteredId}`, function(error){
128     response.redirect('/');
129 });
130 });
131 });
```



기존 미들웨어 사용하기 compression

» 대용량 데이터 테스트 페이지 작성후 개발자 도구 "Network" 탭에서 용량확인

The screenshot shows a web browser at `localhost:3000/page/compression`. The page content includes a link list, action buttons, and a heading. The Network tab in the developer tools is open, showing a list of network requests. The 'compression' request is highlighted with a red box, showing a status of 200, a size of 74.4 KB, and a time of 24 ms.

WEB

- [compression](#)
- [css](#)
- [html](#)
- [node](#)

[create](#) [update](#)

[delete](#)

compression

compression NPM Version NPM Downloads Build Status Test Coverage Node.js compression middleware. The following

Name	Status	Type	Initiator	Size	Time	Waterfall
create_process	302	text/h...	Other	209 B	14 ms	
compression	200	docu...	:3000/create...	74.4 KB	24 ms	



compression 미들웨어

» compression 미들웨어를 이용해서 콘텐츠를 압축해서 전송하는 방법

» Install

- `$ npm install compression --save`

» API

- `var compression = require('compression')`

» Examples

- `app.use(compression())`



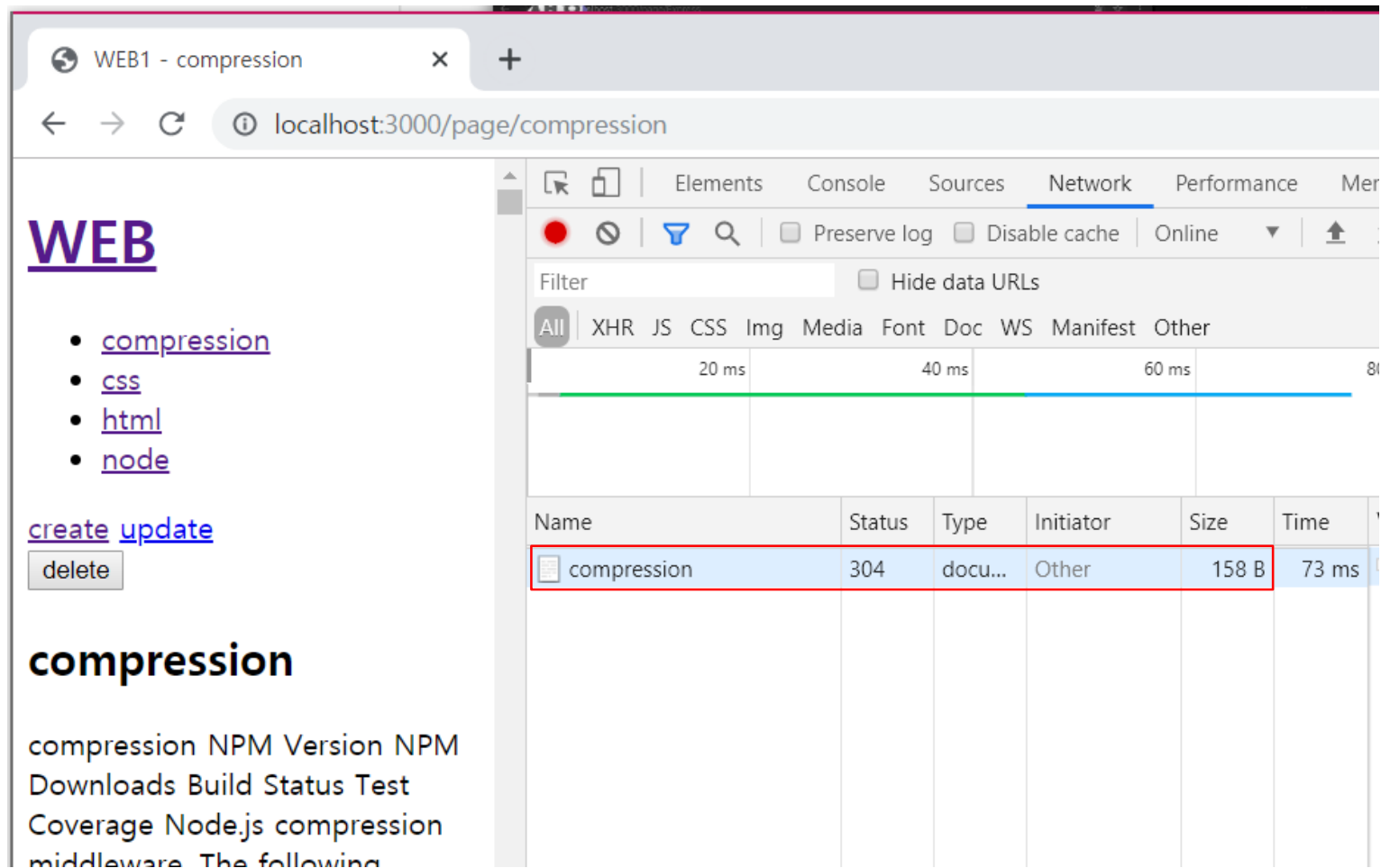
compression 미들웨어

express > JS main.js >  app.get('/') callback >  fs.readdir('./data') callback

```
1  var express = require('express')
2  var app = express()
3  var fs = require('fs');
4  var template = require('./lib/template.js');
5  var path = require('path');
6  var qs = require('querystring');
7  var sanitizeHtml = require('sanitize-html');
8  var bodyParser = require('body-parser');
9  var compression = require('compression')
10
11  app.use(bodyParser.urlencoded({ extended: false }));
12  app.use(compression());
13
```


compression 미들웨어

» compression 미들웨어 적용후 리로드하고 개발자 도구 "Network" 탭에서 용량확인



The screenshot shows a web browser window with the address bar displaying `localhost:3000/page/compression`. The page content includes a heading **WEB**, a list of links ([compression](#), [css](#), [html](#), [node](#)), and buttons for [create](#), [update](#), and [delete](#). Below the links, the heading **compression** is followed by text describing the compression NPM package.

The Network tab in the developer tools is open, showing a list of network requests. The request for `compression` is highlighted, showing a status of 304, a type of `docu...`, and a size of 158 B. The time taken for this request is 73 ms.

Name	Status	Type	Initiator	Size	Time
compression	304	docu...	Other	158 B	73 ms



Express 미들웨어 만들기

» 미들웨어 함수는

- 요청 오브젝트(req), 응답 오브젝트 (res),
- 그리고 애플리케이션의 요청-응답 주기 중 그 다음의 미들웨어 함수 대한 액세스 권한을 갖는 함수입니다.
- 그 다음의 미들웨어 함수는 일반적으로 next라는 이름의 변수로 표시됩니다.

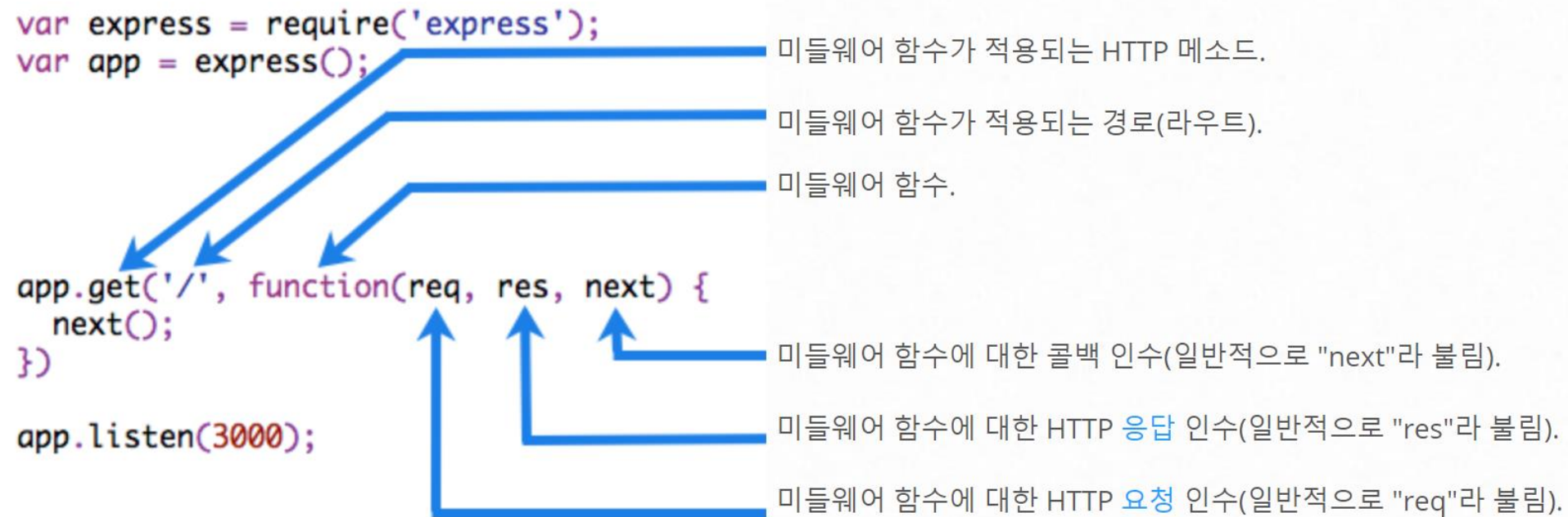
» 미들웨어 함수는 다음과 같은 태스크를 수행할 수 있습니다.

- 모든 코드를 실행.
- 요청 및 응답 오브젝트에 대한 변경을 실행.
- 요청-응답 주기를 종료.
- 스택 내의 그 다음 미들웨어를 호출.



Express 미들웨어 만들기

» 미들웨어 함수 호출의 요소



Express 미들웨어 만들기

» 반복되는 모듈 찾기

```
20 app.get('/', function(request, response) {
21 >   fs.readdir('./data', function(error, filelist){ ...
30   });
31 });
32
33 app.get('/page/:pageId', function(request, response) {
34 >   fs.readdir('./data', function(error, filelist){ ...
55   });
56 });
57
58 app.get('/create', function(request, response){
59 >   fs.readdir('./data', function(error, filelist){ ...
74   });
75 });
76
77 > app.post('/create_process', function(request, response){ ...
84   });|
85
86 app.get('/update/:pageId', function(request, response){
87 >   fs.readdir('./data', function(error, filelist){ ...
110   });
111 });
```



Express 미들웨어 만들기

» 반복되는 fs.readdir() 함수를 미들웨어 함수로 정의하기

```
13  app.use(function(request, response, next){
14    fs.readdir('./data', function(error, filelist){
15      request.list = filelist;
16      next();
17    });
18  });
```

```
20  ✓ app.get('/', function(request, response) {  
21      var title = 'Welcome';  
22      var description = 'Hello, Node.js';  
23      var list = template.list(request.list);  
24      var html = template.HTML(title, list,  
25          `

## ${title}</h2>${description}`, 26 `27 ); 28 response.send(html); 29 });


```

```
20  ✓ app.get('/', function(request, response) {  
21  ✓  fs.readdir('./data', function(error, filelist){  
22      var title = 'Welcome';  
23      var description = 'Hello, Node.js';  
24      var list = template.list(filelist);  
25      var html = template.HTML(title, list,  
26          `

## ${title}</h2>${description}`, 27 `28 ); 29 response.send(html); 30 }); 31 });


```



» 다른 모듈의 fs.readdir() 함수 부분도 미들웨어로 변경

```

20 app.get('/', function(request, response) {
21 >   fs.readdir('./data', function(error, filelist){...
30   });
31 });
32
33 app.get('/page/:pageId', function(request, response) {
34 >   fs.readdir('./data', function(error, filelist){...
55   });
56 });
57
58 app.get('/create', function(request, response){
59 >   fs.readdir('./data', function(error, filelist){...
74   });
75 });
76
77 > app.post('/create_process', function(request, response){ ...
84   });|
85
86 app.get('/update/:pageId', function(request, response){
87 >   fs.readdir('./data', function(error, filelist){...
110   });
111 });

```

var list = template.list(request.list);

Express 미들웨어 만들기

» 미들웨어 함수로 변경된 부분의 공통점 찾기

```
20 app.get('/', function(request, response) {
21 >   fs.readdir('./data', function(error, filelist){ ...
30   });
31 });
32
33 app.get('/page/:pageId', function(request, response) {
34 >   fs.readdir('./data', function(error, filelist){ ...
55   });
56 });
57
58 app.get('/create', function(request, response){
59 >   fs.readdir('./data', function(error, filelist){ ...
74   });
75 });
76
77 > app.post('/create_process', function(request, response){ ...
84   });
85
86 app.get('/update/:pageId', function(request, response){
87 >   fs.readdir('./data', function(error, filelist){ ...
110   });
111 });
```


Express 미들웨어 만들기

» get방식으로 들어오는 모든 요청에 대해서만 미들웨어 함수처리하기

```
13 app.get('*', function(request, response, next){  
14   fs.readdir('./data', function(error, filelist){  
15     request.list = filelist;  
16     next();  
17   });  
18 });
```

```
13 app.use(function(request, response, next){  
14   fs.readdir('./data', function(error, filelist){  
15     request.list = filelist;  
16     next();  
17   });  
18 });
```

