Express 미들웨어

- 익스프레스 프레임워크의 장점 중 하나
- Middleware
  - 이름처럼 요청에 대한 응답 과정 중간에 껴서 어떠한 동작을 해주는 프로그램
  - 익스프레스는 요청이 들어올 때 그에 따른 응답을 보내주는데, 응답을 보내기 전에 미들웨어가 지정한 동작을 수행함
- 예) Morgan Compression, Session, Body-parser 등

유형

- Error Handling
  - 에러 처리를 담당하는 미들웨어
  - 반드시 네개의 인자를 매개변수로 받아 이 미들웨어가 에러를 담당하는 미들웨어라는 것을 식별해야함
- Third-Party
  - 기본적으로 주어지는 Built-in middleware 외에 추가로 설치하여 사용해야하는 미들웨어

Application-Level

- request가 발생될 때마다 실행
  - `app.use()`
  - `app.METHOD() - HTTP method`
- 사용 선언시 기본적으로 해당 미들웨어가 미들웨어 스택에 쌓임
- request가 들어올 때마다 이 스택을 통과하면서 request에 관한 처리를 하고 response

```
// 어떤 request가 들어와도 작동
app.use(function(req, res, next) {
    console.log('Time: ', Date.now())
    next()
})

// /index를 통해 들어오는 요청에 대해서만 작동
app.use('/index', function(req, res, next) {
    /*
        Code
    */
})
```

- Path 지정 가능

Router-Level

- 동작방식은 Applicatoin과 같음, 하지만 라우터 단위로 묶어 넣고 express.Router() 객체 사용하는 차이점
- 사용시 Path별 요청에 따른 동작 방식을 모듈화하여 관리할 수 있음
- /user/mypage의 path로 GET 요청이 들어올 시 user라는 라우터 미들웨어가 처리
  - 안의 내용을 보면 router.get('/myage')에서 처리를 해주는 것

- path별로 처리해줄 미들웨어를 모듈화하여 관리할 수 있음

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

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

```
// 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
```

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 }))`
  - `request.on` 필요없이 `request.body` 사용해서 단번에 parse 된 데이터 가져오기 가능
- `compression`
  - 컨텐츠를 압축해서 전송하는 방법
  - 설치

    `npm install compression --save`
  - API

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

    `app.use(compression())`

- `express.static`
  - 이미지, 자바스크립트, css와 같은 파일을 서비스하는 방법
  - 이미지, CSS 파일 및 JavaScript 파일과 같은 정적 파일을 제공할 때 사용 (Express 제공)
  - 정적 자산이 포함된 디렉토리의 이름을 express.static 미들웨어 함수에 전달하면 파일 제공 시작
  - `app.use(express.static('public'))`
    - public 이라는 이름의 디렉토리에 포함된 정적 파일 제공 가능

      `localhost:3000/images/kitten.jpg`

      `localhost:3000/css/style.css`

      `localhost:3000/js/app.js`

      `localhost:3000/hello.html`
- 에러처리 (404 응답)
  - 단순히 실행해야 할 추가적인 작업이 없다는 에러
  - Express가 모든 미들웨어 함수 및 라우트를 실행했으며 이들 중 어느 것도 응답하지 않았다는 것을 나타냄
  - 404 응답 처리를 위한 미들웨어 함수를 스택의 **가장 아래**에 추가

```
app.use(function(req, res, next) {
    res.status(404).send('Sorry cant find that');
})
```

미들웨어 만들기

- 요청(req), 응답(res)
- 요청-응답 주기 중 그 다음의 미들웨어 함수에 대한 액세스 권한을 가짐
- 그 다음의 미들웨어 함수는 일반적으로 next라는 이름의 변수로 표시
- 수행할 수 있는 태스크
  - 모든 코드 실행
  - 요청 및 응답 오브젝트에 대한 변경
  - 요청-응답 주기를 종료
  - 스택 내의 다음 미들웨어 호출

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

app.listen(3000);
```

익스프레스 에러 처리

- Express에서 404응답
  - 404 응답은 단순히 실행해야 할 추가적인 작업이 없다는 에러
  - 즉 Express는 모든 미들웨어 함수 및 라우트를 실행했으며 이들 중 어느 것도 응답하지 않았다는 것을 나타내기 때문
  - 404 응답 처리를 위한 미들웨어 함수를 스택의 **가장 아래**에 추가

```
app.use(function(req, res, next) {
    res.status(404).send('Sorry cant find that!');
});
```

오류 처리기 작성

- 500 에러 처리
- 기존 상세페이지
  - 라우팅 처리 콜백함수에 `next` 인수를 추가
  - `fs.readFile` 함수의 콜백함수 `err` 인수에 대한 오류 메시지 처리
- 오류 처리 함수정의
  - 함수 인수에 세 개가 아닌 **네 개의** 인수가 있다는 점을 제외하고 다른 미들웨어 함수와 같은 방식으로 오류 처리 미들웨어 함수를 정의

    ```
    app.use(function(err, req, res, next) {
        console.error(err.stack)
        res.status(500).send('Something broke!');
    });
    ```

  - 404 처리 밑에 작성

express.Router 클래스

- express.Router 클래스 사용위해 마운트 가능한 모듈식 라우트 핸들러를 작성
- Router 인스턴스는 완전한 미들웨어 및 라우팅 시스템으로 종종 미니앱 이라고 함

Express Generator

- Express 기반의 프로젝트를 할 때 기본적으로 필요한 파일과 코드를 자동으로 만들어주는 앱
- `npm install express-generator -g`